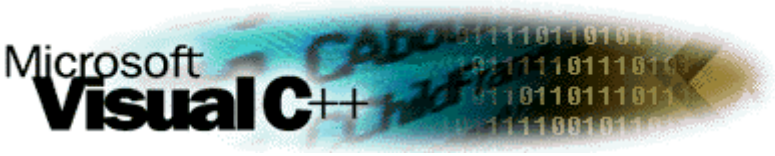
Apostila
Curso de Extensão

MFC – Básico

PROF: ANDRÉ BERNARDI
andrebernardi@hotmail.com.br



Capítulo 0

1 Introdução

1.1 Um pouco de História

Nos anos 60 é lançado a Simula-67 que apresentou pela primeira vez os conceitos de classes, rotinas correlatas e subclasses.

Na década de 70 a Seros-PARC cria a Smalltak até hoje considerada a mais pura das LPOO. No final da década de 80 aparece a C++, uma linguagem híbrida. Já a o Visual C++ surgiu no início da década de 90.

Orientação a Objetos é o maior avanço em software destes últimos anos. É uma forma mais natural de se analisar o mundo. Ela nos permite construir sistemas melhores e, além disso, de maneira mais fácil. Será a mais importante das tecnologias emergentes na área de software nos anos 90.

As técnicas estruturadas que, sem dúvida, atualmente são as mais populares na comunidade de informática, obtiveram grande aceitação desde que foram lançadas no final dos anos 70. Contudo a medida que foram sendo utilizadas, a decomposição funcional mostrou-se inadequada em situações de sistemas complexos e principalmente para profissionais iniciantes. Os aperfeiçoamentos introduzidos em 1984, por Sthephen M e John F Palmer e, mais tarde, complementados por Stephen Mellor e Paul Ward, para sistemas em tempo real, ajudaram a análise estruturada a se tornar mais eficiente. Contudo os sistemas criados com as técnicas estruturadas ainda são difíceis de serem incrementados com novas funções e as alterações em funções já existentes, muitas vezes, provocam sérios problemas em outras partes do software.

Na prática de programação orientada a objetos estaremos atentos em nossos programas para pontos como:

- Compatibilidade, portabilidade.
- Segurança.
- Reusabilidade.
- Facilidade de integração.
- Facilidade de extensão.
- Eficiência.

1.2 Reutilização

A reutilização está baseada na padronização a qual é adotada há longa data em toda a indústria moderna, seja no projeto de carros, televisores, computadores, etc. A padronização traz inúmeras vantagens, entre elas podemos citar a seguintes:

- as peças padrões são mais baratas;
- são mais confiáveis;
- geralmente são mais fáceis de serem consertadas ou substituídas.

Na informática a reutilização de códigos ainda é praticada em escala muito reduzida e decorrente da iniciativa isolada de alguns programadores e projetistas. São vários os motivos para isso, entre eles:

- existência de uma Biblioteca e de um Sistema de Catálogo;

- facilidade para documentação;
- sistemática rigorosa para testes;
- novas técnicas para especificação de sistemas tendo em vista a reutilização de módulos;
- linguagem, ferramentas e ambientes de desenvolvimento que estimulem a adoção, em larga escala, de tais métodos dentro das empresas;
- criação de novos geradores de sistemas que operem solucionando e combinando módulos padrões e que satisfaçam as necessidades específicas das aplicações;
- mudança administrativas e gerenciais nas empresas de modo a apoiarem e estimularem os criadores de módulos padrões.

1.3 Vantagens da Orientação a Objetos

A Orientação a Objetos traz vários benefícios no desenvolvimento e manutenção de software. Para melhor compreensão dessas vantagens vamos dividi-las em dois grupos. No primeiro, que chamaremos de "Vantagens Diretas" colocamos aquelas que representam conseqüências diretas da adoção da Orientação a Objetos e , no segundo grupo, as "Vantagens Reais", estarão aquelas que são de fato o que procuramos com essa tecnologia.

Vantagens Diretas:

- maior facilidade para reutilização de código e por conseqüência do projeto;
- possibilidade do desenvolvedor trabalhar em um nível mais elevado de abstração;
- utilização de um único padrão conceitual durante todo o processo de criação de software;
- maior adequação à arquitetura cliente/servidor;
- maior facilidade de comunicação com os usuários e com outros profissionais de informática.

Vantagens Reais:

- ciclo de vida mais longo para os sistemas;
- desenvolvimento acelerado de sistemas;
- possibilidade de se construir sistema muito mais complexos, pela incorporação de funções prontas;
- menor custo para desenvolvimento e manutenção de sistemas;

1.4 Princípios da Orientação a Objetos

Abstração - Consiste na concentração nos aspectos essenciais. Preserva a liberdade de se tomar decisões evitando, tanto quanto possível comprometimentos prematuros com detalhes.

- É o processo pelo qual a mente ou a inteligência compreende o objeto, suas características e suas funcionalidade.

Agregação - é o relacionamento "parte-todo" ou "uma-parte-de" no qual os objetos que representam os componentes de alguma coisa são associados a um objeto que representa a estrutura inteira.

Comportamento - é como um objeto age e reage, em termos de mudança de seu estado e envio de mensagem. externamente visível e comprovada atividade.

Encapsulamento - Também chamado de ocultamento de informações, é o resultado de ocultar os detalhes de implementação de um objeto.
É o termo formal que descreve a junção de métodos e dados dentro de um objeto de maneira que o acesso aos dados seja permitido somente por meio dos próprios métodos do objeto. Nenhuma outra parte do programa pode operar diretamente em um dado do objeto.

A comunicação de objetos ocorre exclusivamente por meio de mensagens explícitas. Também definido como um processo de "esconder" a complexidade interna de uma classe para suportar ou reforçar a abstração.

Estado - são resultados cumulativos do comportamento do objeto - uma das possíveis condições na qual o objeto existe, caracterizada pelas quantidades que são distintas de outras quantidades.
Em qualquer ponto do tempo, o estado do objeto abraça todas as (usualmente estática) propriedades do objeto mais os atuais (normalmente dinâmico) valores dessa propriedade.

Herança - um método de derivar novas classes a partir de outras previamente existentes. A classe derivada herda a descrição de cada uma das classes bases, permitindo sua extensão através da adição de novas variáveis e funções membro e usar funções virtuais.
É o compartilhamento de similitudes entre componentes preservando as diferenças.
Representa generalização e especialização, tornando explícitos os atributos e comportamentos comuns em uma hierarquia de classes.
É o mecanismo através do qual os atributos e o comportamento dos objetos de uma classe são assumidos pelos objetos de outra classe.

É uma relação entre uma super-classe e suas sub-classes, isto é, o mecanismo pelo qual elementos mais específicos incorporam estrutura e comportamento de elementos mais gerais relacionados.

Há duas formas de se descobrir heranças:
- Generalização
- Especialização

Mensagem - é uma operação que um objeto executa sobre outro. Os termos mensagem, método e operação são usualmente intercambiáveis. Representa uma ação a ser praticada pelo objeto.
Exemplo: MenuPrincipal.Seleciona (Segunda Opção);
Onde:
MenuPrincipal = Objeto Seleciona = Método Segunda Opção = Parâmetro

Módulo - É uma unidade de programa que contém declarações, expressas num vocabulário de uma linguagem de programação particular, que forma a realização física de parte ou de todas as classes e objetos do projeto lógico do sistema.
É uma unidade de códigos que serve como um bloco construído para uma estrutura física de um sistema.
Composto das seguintes partes:

Interface - É a visão externa de uma classe, objeto ou módulo, os quais enfatizam suas abstrações enquanto esconde suas estruturas e os segredos de seus comportamentos.

Implementação - É a visão interna de uma classe, objeto ou módulo, incluindo os segredos de seu comportamento.

Polimorfismo - Processos que executem funções semelhantes em componentes diferentes devem ser chamados pelo mesmo nome.
É a habilidade de duas ou mais classes responderem à mesma solicitação, cada uma a seu modo. Conceito de permitir uma única interface para múltiplas funções.

Objeto é uma abstração encapsulada (qualquer coisa, real ou abstrata) que inclui: informações de estado (descrita por dados), um conjunto claramente definido de protocolos de acesso (mensagens que cada objeto responde) e possui um comportamento que se interesse acompanhar ou que seja útil.
Um objeto pode ser: uma pessoa, um material, uma instituição, um fato, um lugar, um conceito.

- tudo que é aprendido pelo conhecimento que não é o sujeito do conhecimento.
- tudo que é manipulável e/ou manufaturáveis.

- tudo que é perceptível por qualquer sentido.
- tudo que é externo ao sujeito.
- um objeto tem estado, comportamento e identidade.
- a estrutura e comportamento de objetos similares são definidos em suas classes.
- os termos instâncias e objetos são intercambiáveis.

Dois objetos exatamente iguais em atributos, relacionamentos e funções podem ser diferenciados através de seu identificador ÚNICO,
Exemplo: o número da matrícula do EMPREGADO, o CPF etc.

Tipos de Objetos:

- Concretos (pessoa, lápis, carro, relógio etc.)
- Intangíveis (hora, idéia, organização, projeto etc.)
- Papel (médico, paciente, professor, aluno etc.)
- Relacional (casamento, parceira, propriedade etc.)
- Evento (venda, admissão, pane_no_sistema etc.)
- De Interface (janela, ícone, string de caracteres etc.)

O comportamento de um objeto é implementado por meio de métodos que podem ser dos seguintes tipos:

- Relativos ao ciclo de vida;
- Recuperadores;
- Relativos à administração de armazenamento;

Instância - Os objetos são *instâncias* de uma classe. As propriedades de qualquer instância (objeto) são determinadas pela *descrição da classe* à qual pertence.

Subclasse - É um subconjunto de objetos pertencentes a uma classe que têm em comum em relação aos demais objetos desta classe: - dados adicionais e/ou, - operações adicionais.

Classe e Objetos - É um conjunto de objetos que são descritos pelos mesmos dados e possuem o mesmo comportamento.

- conjunto de objetos que compartilham uma estrutura e um comportamento comum.
- o termo classe e type são usualmente (mas não sempre) intercambiáveis uma classe tem conceito ligeiramente diferente de um tipo, no qual ela enfatiza a classificação da estrutura e comportamento.
- uma classe é um tipo definido pelo usuário que contém o molde, a especificação para os objetos, assim como o tipo inteiro contém o molde para as variáveis declaradas como inteiros.
- a classe envolve, associa funções e dados controlando o acesso a estes.
- definir classe implica em especificar os seus atributos (dados) e suas funções (métodos).

As interfaces de uma classe ou objetos são representadas através de seus métodos e de suas funcionalidades.

Em resumo, **objetos** são **instâncias** de **classes** que respondem **mensagens** de acordo com **métodos** determinados pelo *protocolo de descrição de classes* .Os objetos também têm **variáveis de estado** definidas no protocolo de descrição de classe que podem ter valores **iguais** ou **diferentes** nas várias instâncias da classe.

1.5 Classes e Objetos

Classes e Objetos estão intimamente relacionados, porém não são iguais. Uma Classe contém informação de como o objeto deverá se parecer e se comportar. É um tipo definido

pelo usuário que contém o molde, a especificação para os objetos, assim como o tipo inteiro contém o molde para as variáveis declaradas como inteiros.

Um primeiro exemplo desse relacionamento pode ser:

- entenda o esquema elétrico e o layout de um telefone como aproximadamente uma classe;
- o objeto, ou instância de uma classe, seria o telefone.

Uma classe determina as características de um objeto: *propriedades, eventos e métodos relacionados*.

Propriedades:

Um objeto tem certas propriedades, ou atributos. Para o exemplo acima, um telefone possui cor e tamanho. Quando um telefone é colocado em sua casa ou escritório, ele terá uma certa posição sobre uma mesa. O receptor pode estar no ou fora do gancho.

Objetos criados com o Visual C++, contém certas características que são determinadas pela classe em que o objeto está baseado. Estas propriedades podem ser fixadas a em seu desenvolvimento ou em tempo de execução. Por exemplo um *check box*, poderá conter as seguintes propriedades, descritas na tabela abaixo:

Propriedade	**Descrição**
Titulo	Um texto descritivo antes do check box
Habilitado	Quando o check box puder ser selecionado pelo usuário.
Esquerda	Alinhamento a esquerda do check box
Visível	Quando o check box está visível na janela

Eventos:

Cada objeto reconhece e pode responder a certas ações chamadas eventos. Um evento é uma atividade específica e pré determinada, iniciada tanto pelo usuário quanto pelo sistema. Na maioria dos casos eventos são gerados pela interação do usuário. Por exemplo, para o telefone, um evento é disparado toda vez que o usuário retira o fone do gancho. Eventos são gerados quando o usuário pressiona um botão para discar, etc.

Na programação para Windows, uma ação do usuário que pode provocar o disparo de um evento pode ser um click ou movimento do mouse, ou pressionar uma tecla do teclado. Quando um erro ocorre ao inicializar um objeto um evento do sistema é disparado.

A tabela a seguir mostra alguns eventos relacionados com o check box.

Evento	**Descrição**
Click do mouse	Usuário pressiona o check box
GotFocus	Usuário seleciona o check box por um click ou tab.
LostFocus	Usuário seleciona outro controle

Métodos:

Métodos são funções que estão associadas aos objetos. Suas chamadas são feitas através de mensagens. Um método pode estar relacionado a um Evento. Exemplo se for escrito um método para responder ao evento de click do mouse, este será chamado toda vez que o botão do mouse for pressionado. Métodos podem existir independente de eventos, estes métodos são chamados explicitamente dentro do código.

A tabela a seguir mostra alguns métodos relacionados com o check box.

Método	**Descrição**
Atualizar	O valor do check box é atualizado para refletir as alterações que possam Ter ocorrido na base de dados subjacente.
SetFocus	O foco atual é passado ao check box como resultado de um usuário pressionar a tecla TAB até o check box ser selecionado.

A classe envolve, associa, funções e dados, controlando o acesso a estes, definí-la implica em especificar os seus atributos (dados) e suas funções membro (métodos).

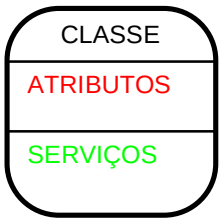
Exemplos:

1. Um programa que utiliza uma interface controladora de um motor elétrico provavelmente definiria a classe motor. Os atributos desta classe seriam: temperatura, velocidade, tensão aplicada. Estes provavelmente seriam representados na classe por tipos como float ou long . As funções membro desta classe seriam funções para alterar a velocidade, ler a temperatura, etc.
2. Um programa editor de textos definiria a classe parágrafo que teria como um de seus atributos uma string ou um vetor de strings, e como funções membro, funções que operam sobre estas strings. Quando um novo parágrafo é digitado no texto, o editor cria a partir da classe parágrafo um objeto contendo as informações particulares do novo texto. Isto se chama instanciação ou criação do objeto.
3. Um programa de controle de vendas definirias as classes Cliente, Venda, Produto, Fatura, Vendedor. As classes de Cliente e Venda teriam os seguintes dados e métodos:

Classe	Atributos	Métodos
Cliente	id_cliente	Incluir
	nome_cliente	Alterar
	endereco	Excluir
	pessoa_autorizada	Consultar
	Status	
Venda	id_cliente	registrar
	id_vendedor	calcular_comissão
	data_venda	calcular_fatura
	produto	

2 Notação utilizada para Análise de Projeto Baseado em Objetos.

2.1 Classe

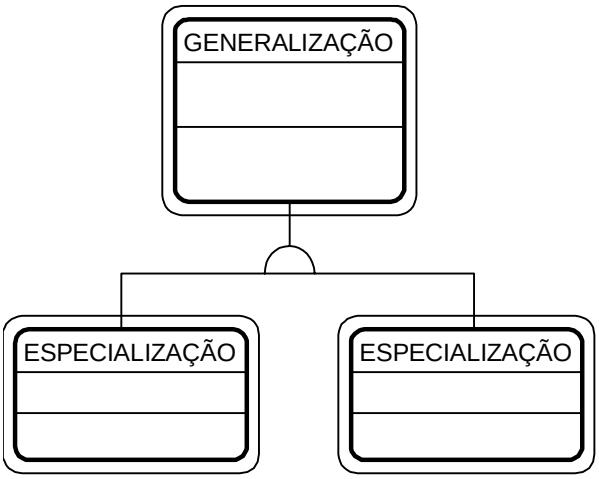


Representa uma Classe abstrata

2.2 Classe & Objeto

Retângulo mais interno representa a Classe (molde para os objetos). O retângulo mais externo representa os objetos da classe. Uma classe e seus objetos.

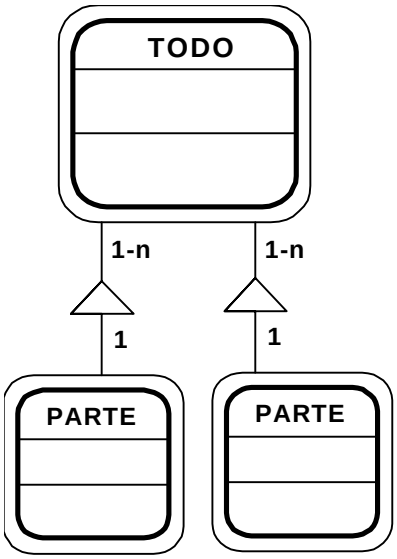
2.3 Estrutura Generalização - Especialização (Gen-Spec)



Objeto generalizador: apresenta características comuns a todos os objetos especializados

Objetos especializados: apresentam, além das características do objeto generalizador (atributos e serviços), novas características que podem incluir novos atributos e redefinição de serviço

2.4 Estrutura Todo - Parte



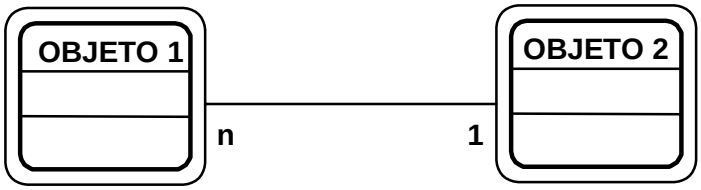
Objeto **Todo**

(1-n) Cada Todo é composto por 1 ou mais partes

Objetos **Parte**

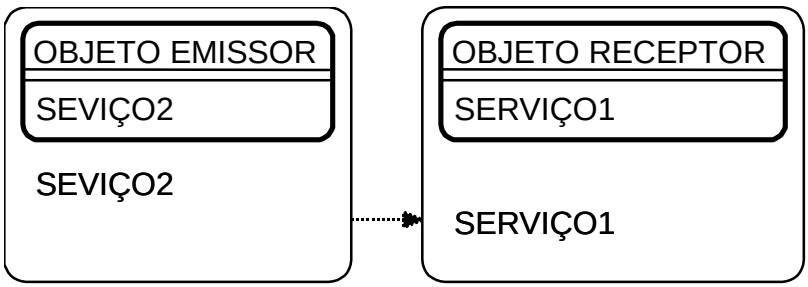
(1) Cada parte pertence a um único Todo

2.5 Conexões de Ocorrência



Objetos associados entre si. Uma conexão de ocorrência é um mapeamento do domínio do problema que um objeto precisa ter com outros objetos, para cumprir suas responsabilidades.
A numeração ao lado do objeto representa o número de ocorrências do objeto que participam da associação.

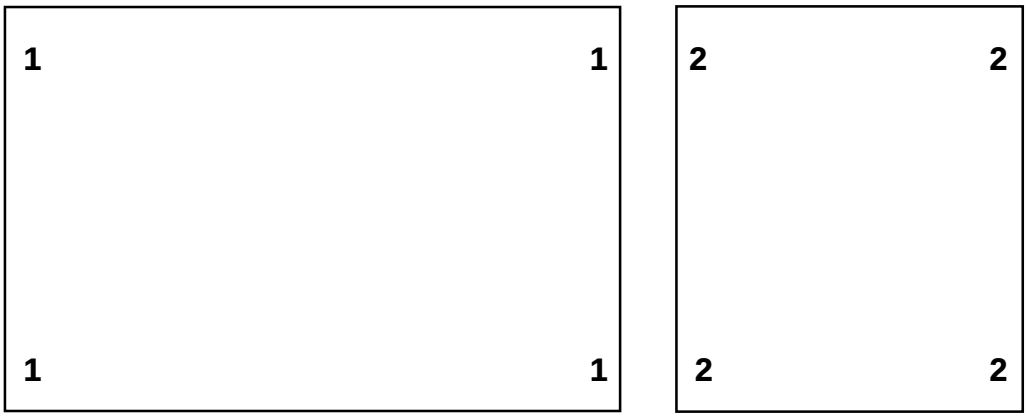
2.6 Conexão de Mensagens



Um emissor envia uma mensagem para um receptor para a realização de um processamento.

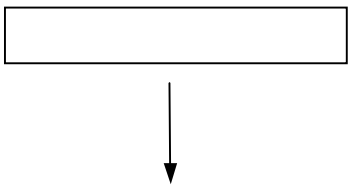
2.7 Assunto

Notação para Assunto, quebrada



Notação para Assunto, expandida (quando é mostrada com outras camadas).

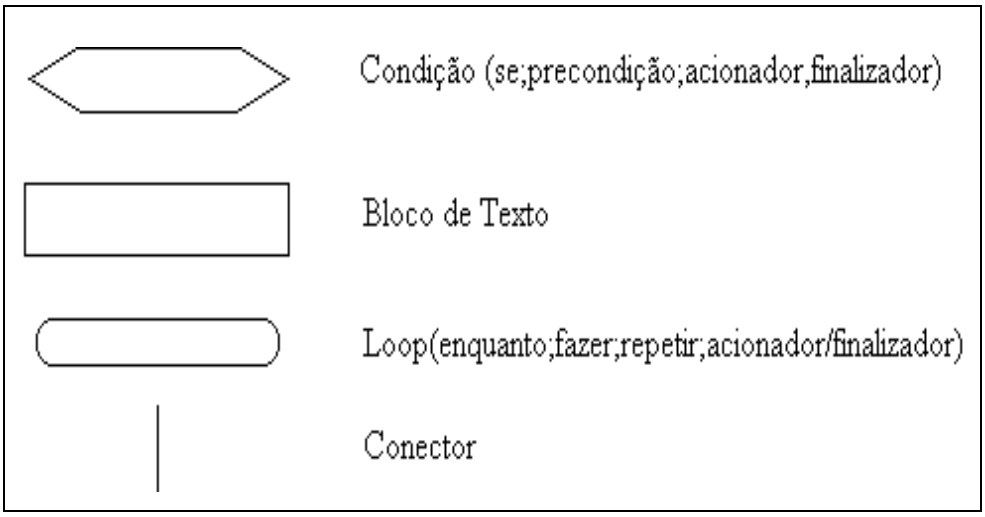
2.8 Notação para Diagrama de Estado do Objeto



Estado

Transição

2.9 Notação para Diagrama de Serviço



3 Descrição de protocolos de classe

3.1 Especificando uma classe

Suponha um programa que controla um motor elétrico através de uma saída serial (como exemplo da seção 1.5). A velocidade do motor é proporcional a tensão aplicada, e esta proporcional aos bits que vão para saída serial e passando por um conversor digital analógico.

Vamos abstrair todos estes detalhes por enquanto e modelar somente a interface do motor como uma classe, a pergunta é que funções e que dados membro deve ter nossa classe, e que argumentos e valores de retorno devem ter essas funções membro:

Representação da velocidade:

A velocidade do motor será representada por um atributo, ou dado membro, inteiro (int). Usaremos a faixa de bits que precisarmos, caso o valor de bits necessário não possa ser fornecido pelo tipo, usaremos então o tipo long, isto depende do conversor digital analógico utilizado e do compilador.

Representação da saída serial:

O motor precisa conhecer a sua saída serial, a sua ligação com o "motor do mundo real". Suponha uma representação em hexadecimal do atributo endereço de porta serial, um possível nome para o atributo: enderecomotor. Não se preocupe em saber como usar a representação hexadecimal.

Alteração do valor da velocidade:

Internamente o usuário da classe motor pode desejar alterar a velocidade, cria-se então o método (em C++ função membro): void altera_velocidade(int novav); . O código anterior corresponde ao cabeçalho da função membro, ela é definida junto com a classe motor, associada a ela. O valor de retorno da função é void (valor vazio), poderia ser criado um valor de retorno (int) que indicasse se o valor de velocidade era permitido e foi alterado ou não era permitido e portanto não foi alterado.

Não faz sentido usar, chamar, esta função membro separada de uma variável do tipo motor, mas então porque na lista de argumentos não se encontra um motor? Este pensamento reflete a maneira de associar dados e código (funções) das linguagens procedurais. Em linguagens orientadas a objetos o código e os dados são ligados de forma diferente, a própria declaração de um tipo definido pelo usuário já engloba as declarações das funções inerentes a este tipo.

Note que não fornecemos o código da função, isto não é importante, por hora a preocupação é com a interface definida pela classe: suas funções membro e dados membro. Apenas pense que sua interface deve ser flexível de modo a não apresentar entraves para a criação do código que seria feita numa outra etapa. Nesta etapa teríamos que imaginar que o valor numérico da velocidade deve ir para o conversor onde irá se transformar numa diferença de potencial a ser aplicada nos terminais do motor, etc.

3.2 Componentes de uma descrição de classe

Uma descrição de classe é composta por três elementos:

1. Declarações de campos de dados
2. Protótipos para funções membros
3. Definições das funções membros

Existem muitas opções com relação ao lugar em que podemos colocar as várias partes de uma descrição de classe. Uma maneira simples é incluir todas as três partes acima em um único arquivo de cabeçalho. Uma segunda alternativa é colocar os campos de dados e os protótipos em um arquivo de cabeçalho e colocar as definições das funções em um ou mais arquivos de compilação. Uma terceira opção é incluir a descrição completa da classe em um único arquivo que também contenha o programa executável. Esta última escolha é indesejável quando outros usuários forem utilizar esta classe, o que geralmente acontece.

Uma descrição de classe formal começa com a palavra reservada **class** (seguida de um abre chaves **{**) e termina com um fecha chaves e um ponto e virgula. (**};**) conforme o exemplo abaixo.

A descrição de classe tem três seções cada uma delas contendo campo de dados e funções membros. As três seções são: privada (**private**:), protegida (**protected**:) e pública (**public**:). Os dados e as funções membros, na seção privada podem ser acessados apenas pelo protocolo da descrição de classe. Os dados e as funções membros na seção pública podem ser acessados pelo código de qualquer arquivo usuário que "inclua" a descrição de classe. As variáveis (objetos) declaradas como do tipo classe têm acesso direto apenas à seção pública de suas descrições de classe. A categoria protegida tem aplicação no uso de classes derivadas (Herança).

Exemplo de Sintaxe para Descrição de Classe

```
class CNomeClasse
{
private:
        tipo  m_dados1;
        tipoRetorno funcao1( lista parametros );

protected:
        tipo m_dados2;
        tipoRetorno funcao2( lista parametros );

public:
        CNomeClasse ( lista parametros a);      // Construtor 1
        CNomeClasse ( lista parametros b);      // Construtor 2
        CNomeClasse ( void );                   // Construtor sem parametros
```

```
        ~CNomeClasse ( void );                  // Destrutor
        tipo m_dados3;
        tipoRetorno funcao3( lista parametros ) {  instrucoes  };
};
```

Os construtores são identificados como funções membros cujos nomes são iguais ao nome da classe. A listagem acima mostra três construtores para a classe **CNomeClasse** . A sobrecarga do nome do construtor precisa satisfazer as mesmas exigências das outras funções, isto é, as listas de parâmetros precisam ser diferentes. Um construtor pode ter uma lista de parâmetros void. Os construtores podem fornecer valores default para um ou mais de seus parâmetros.

A função membro ~**CNomeClasse** é um destrutor. Ela precisa ter uma lista de parâmetros void. Pode ter um corpo de código executável se necessário. Na listagem, apenas protótipos são fornecidos para o Construtor1, Construtor2, Construtor3, destrutor, funcao1 e funcao2. Tanto protótipos como definições são fornecidos para funcao3. Quando a definição de uma função membro está incluida na descrição de classe, ela é tratada como código inline. As restrições de implementação geral pedem que o código inline não contenha nenhuma construção de controle, exceto **if** (como por exemplo **do**, **for**, **while**, **break**, **continue**, **goto**, **switch**). Recomenda-se que o código inline seja usado com ponderação e apenas no caso de implementações relativamente simples de funções membros.

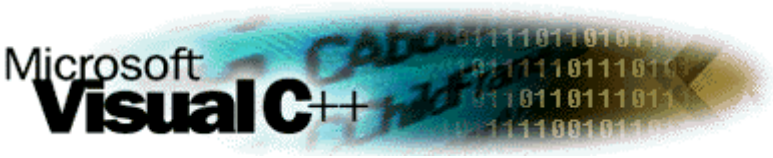
Os detalhes relativos às definições de Construtor1, Construtor2, Construtor3, destrutor, funcao1 e funcao2, são fornecidos fora da descrição formal da classe. Eles pecisam então ser identificados com suas classes, conforme indica a listagem abaixo. Estes detalhes podem estar contidos no mesmo arquivo da descrição formal da classe ou em um outro arquivo. Se forem implementados em um outro arquivo, o arquivo cabeçalho precisa estar “incluido”, como mostra a listagem abaixo. As definições isoladas de funções são identificadas com a classe apropriada colocando-se o qualificador **CNomeClasse::** antes do nome de cada função.

Definições Externas para Funções Membros de uma Descrição de Classe

```
// Conteúdo do arquivo para definições externas de funções membros

#include “NomeClasse.h”

tipoRetorno CNomeClasse::funcao1( lista parametros )
{
        instrucoes;
};

tipoRetorno CNomeClasse::funcao2( lista parametros )
{
        instrucoes;
};

CNomeClasse::CNomeClasse ( lista parametros a)  // Construtor 1
{
        instrucoes;
};

CNomeClasse::CNomeClasse ( lista parametros b)  // Construtor 2
{
        instrucoes;
};

CNomeClasse::CNomeClasse ( void )               // Construtor sem parametros
{
        instrucoes;
};

CNomeClasse::~CNomeClasse ( void )              // Destrutor
```

```
{
        instrucoes;
};
```

Se alguém tentar incluir campos de dados no arquivo externo que é parte de uma descrição de classe, ocorrerá um erro. Todas as declarações precisam estar na descrição formal da classe. Apenas as definições das funções membros e membros de dados estáticos podem ser externas à descrição formal da classe.



Capítulo 1
Microsoft Windows e Visual C++

Muito já foi escrito sobre a aceitação de Microsoft Windows e os benefícios de sua interface gráfica com o usuário (GUI). Esta introdução resume o modelo de programação Windows (Win32 em particular) e mostra como os componentes do Visual C++ trabalham em conjunto para lhe ajudar a escrever aplicações para Windows. No decorrer do capítulo, você aprenderá algumas coisas novas sobre o Windows.

1 O Modelo de programação do Windows

Não importa que ferramentas de desenvolvimento você usa, a programação para Windows é diferente de velho-estilo de programação orientado a grupo ou orientado a transação. Para iniciar é preciso saber alguns fundamentos do Windows. Como uma armação de referência, nós usaremos o famoso modelo de programação do MS-DOS. Até mesmo se você não programa atualmente para MS-DOS claro, você está provavelmente familiarizado com isto.

1.1 Processamento de Mensagens

Quando você escreve uma aplicação MS-DOS, baseada em C, a única exigência absoluta é a existência de uma função chamada *main*. O sistema operacional chama *main* quando o usuário roda o programa, e daquele ponto em diante pode-se usar qualquer estrutura de programação que desejar. Se seu programa precisa adquirir entradas do teclado via usuário ou ainda usar serviços do sistema operacional, é possível chamar as funções apropriadas, como *getchar*, ou talvez usar uma biblioteca de funções de manipulação de caracteres.

Quando o sistema operacional Windows inicia um programa, chama a função *WinMain* do programa. Em algum lugar de sua aplicação tem que ter *WinMain* que executa algumas tarefas específicas. Sua tarefa mais importante é criar a janela principal da aplicação que tem que ter seu próprio código para processar mensagens que o Windows envia. Uma diferença essencial entre um programa escrito para o MS-DOS e um programa escritos para Windows é que um programa baseado em MS-DOS chama o sistema operacional para adquirir as entradas do usuário, mas um programa baseado em Windows o processa as entradas do usuário por mensagens do sistema operacional.

> **NOTE**: Muitos ambientes de desenvolvimento para Windows, inclusive Microsoft Visual C++ versão 5.0 com a Microsoft Foundation Class (MFC) versão 4.21, simplifica a programação ocultando a função *WinMain* e estruturando o processo de manipulação de mensagens. Quando você usa a biblioteca MFC, você precisa não escreva uma função *WinMain* mas é essencial que você entenda o vínculo entre o sistema operacional e seus programas.

A maioria das mensagens do Windows são estritamente definidas e aplicam-se a todos os programas. Por exemplo, a mensagem WM_CREATE é enviada quando uma janela está sendo criada, a mensagem WM_LBUTTONDOWN é enviada quando o usuário aperta o botão esquerdo do mouse, a mensagem WM_CHAR é enviada quando o usuário digita um caractere, e a mensagem de WM_CLOSE é enviada quando o usuário fecha uma janela.

Todas as mensagens têm dois parâmetros de 32-bit que carregam informações como as coordenadas do cursor, código da tecla, e assim por adiante. Windows envia mensagens de WM_COMMAND à janela apropriada em resposta para seleções do usuário no menu, pressiona um botão em um diálogo, etc. Os parâmetros da mensagem Command variam e dependendo layout do menu da janela. Você pode definir suas próprias mensagens que seu programa pode enviar a qualquer janela no desktop. Estas mensagens definidas pelo usuário na verdade fazem com que o C++ se pareça um pouco como Smalltalk.

Não se preocupe ainda como estas mensagens são conectadas a seu código. Isso é o trabalho da Estrutura de Aplicação (Application Framework). Entretanto, esteja atento que o processamento de mensagens do Windows impõe muitas exigências na estrutura de seu programa. Não tente forçar seu programa Windows para ele se parecer com seus programas velhos de MS-DOS. Estude os exemplos neste curso, bem como os exemplos do Visual C++ e então estrá preparado para iniciar em sua nova filosofia.

1.2 Interface de Dispositivos de Gráficos do Windows (GDI)

Muitos programas de MS-DOS escreveram diretamente à memória de vídeo e a porta de impressora. A desvantagem desta técnica era a necessidade de prover driver de software para toda placa de vídeo e todo modelo de impressora. Windows introduziu uma camada de abstração chamada a Interface de Dispositivo de Gráficos (GDI). Windows provê o driver de vídeo, assim seu programa não precisa saber o tipo de placa de vídeo e impressora de seu sistema. Ao invés de acessar diretamente o hardware, seu programa chama funções da GDI que referencia uma estrutura de dados chamada um contexto de dispositivo. Windows mapeia a estrutura de contexto de dispositivo para um dispositivo físico que reconhece as instruções de entrada/saída apropriadas. A GDI é quase tão rápida quanto o acesso direto de vídeo, e permite a diferentes aplicações escritas para Windows compartilhar a exibição na tela.

1.3 Programação Baseada em Recursos

Quando você programa para Windows, você armazena dados em um arquivo de recurso que usa vários formatos estabelecidos. O linker combina este arquivo de recurso binário com a saída do compilador de C++ para gerar um programa executável. Arquivos de recurso podem incluir bitmaps, ícones, definições de menu, layouts de caixas de diálogo, e strings. Podem ainda incluir formatos de recurso personalizado definidos pelo próprio usuário.

Você usa um editor de texto para editar um programa, mas você geralmente usa ferramentas WYSIWYG (What You See Is What You Get - o que você vê é o que você adquire) para editar recursos. Se você está dispondo uma caixa de diálogo, por exemplo, que você seleciona elementos (botões, caixas de lista, e 50 adiante) de um array de ícones chamado Control Palette, e você posiciona e classifica segundo o tamanho os elementos com o mouse. O ambiente de desenvolvimento integrado para Visual C++, Microsoft Developer Studio 97, tem os editores de recursos gráficos para todos os formatos de recurso padrões.

1.4 Gerenciamento de Memória

A cada versão nova do Windows, o gerenciamento de memória se torna mais fácil. Se você ouviu falar histórias de horror de travar handles de memória, thunks, e burgermaster, não se preocupe. Isso é tudo passado. Hoje simplesmente você aloca a memória que você precisa e o Windows cuida dos detalhes.

1.5 Bibliotecas de Vínculo Dinâmico (DLL's)

No ambiente MS-DOS, todos os módulos do programa são vinculados(linked) estaticamente durante o processo de construção (build). Windows permite o vínculo dinâmico, por meio de bibliotecas especificamente construídas que podem ser lidas e vinculadas em tempo de execução do programa. Essas bibliotecas de vínculo dinâmico(DLL's) podem ser compartilhadas por múltiplos programas, o que permite economizar memória e espaço em disco. A modularidade de um programa é reforçada através do uso do Vinculo Dinâmico, pois as DLL's podem ser testadas e compiladas separadamente.

Os desenvolvedores de MFC tiveram sucesso combinando todas as classes que compõe a estrutura das aplicações em algumas DLLs. Isto significa que você pode vincular estática ou dinamicamente as classes da estrutura das aplicação em seu aplicativo.

1.6 A Interface de Programação de Aplicações Win32 (API)

Antigos programadores em Windows escreviam aplicativos em C para a interface de programação Win16 (API). Hoje, para escrever aplicativos 32-bit, é necessário utilizar a nova Win32 API, ainda que direta ou indiretamente. Muitas funções Win16 teem suas equivalentes em Win32, mas muitos dos parâmetros são diferentes, por exemplo parâmetros de 16 bits foram substituídos por parâmetros de 32 bits. A Win32 API oferece muitas funções novas, incluindo funções de manipulação de disk I/O, que eram formalmente manipuladas por funções chamadas do MS-DOS. Com a versão de 16-bit do Visual C++, programadores com MFC estão bem separados destas diferenças de API porque eles escreveram no padrão da MFC que foi projetado para trabalhar com Win16 ou Win32.

2 Os Componentes do Visual C++

Microsoft Visual C++ se comporta como dois aplicativos Windows de desenvolvimento de sistemas completos em um só produto. É possível criar programas para Windows utilizando a linguagem C através da Win32 API. A programação em linguagem C Win32 é descrita no livro de Charles Petzold *Programming Windows 95* (Microsoft Press, 1996). Você pode usar muitas ferramentas do Developer Studio e de Visual C++, inclusive os editores de recurso, para tornar a programação de baixo nível Win32 mais fácil.

Visual C++ ainda inclue a ActiveX Template Library (ATL), que pode ser utilizada para desenvolver controles ActiveX controls para a Internet. A programação utilizando a ATL não é igual a programação Win32 em C nem a programação usando a MFC, e não será abordada neste curso.

Este curso não é sobre linguagem C, programação Win32, ou programação ATL. Trata sobre programação C++ dentro da estrutura das aplicações da biblioteca MFC que é parte do Visual C++. Serão utilizadas as classes C++ que estão documentadas dentro da Microsoft Foundation Class Reference and será possível utilizar de ferramentas específicas do Visual C++ para a estrutura das aplicações, como o AppWizard e o ClassWizard.

> NOTA: O uso da interface da MFC não impede que sejam utilizadas função da API diretamente. De fato muitas vezes é necessário utilizar as funções da API diretamente dentro de programas baseados na MFC.

Um avaliação dos componentes do Visual C++, lhe ajudarão a adquirir seus portes antes de você comece com a estrutura das aplicações. A Figura 1 a seguir mostra um overview do processo de construção(build) de um aplicativo no Visual C++.

2.1 O processo de construção e o Microsoft Developer Studio 97

Developer Studio é um ambiente de desenvolvimento integrado (IDE) que é compartilhado pelo Microsoft Visual C++, Microsoft Visual J++, Microsoft Visual Basic, e uma série de outros produtos. Este IDE é uma ferramenta de desenvolvimento que está longe do que foi o original Visual Workbench, que estava baseado no QuickC for Windows. Fazem parte do atual Developer Studio, Docking Windows, barras de ferramentas configuráveis, além de um editor que pode ser personalizado e utilizar macros. A ajuda online para esse sistema (InfoViewer) trabalha como um Web browser. A Figura 2 a seguir mostra o Developer Studio em ação.

Quem já foi usuário de versões anteriores do Visual C++ ou Borland IDE, já compreende como o Developer Studio opera. Mas quem nunca utilizou uma dessas ferramentas (IDEs), precisará saber o que é um projeto. Uma coleção de códigos fontes interrelacionado que são compilados e linkados para criar um programa executável baseado no Windows ou uma DLL é um projeto. Arquivos fontes de cada projeto são geralmente armazenados em subdiretórios separados. Além disso , um projeto depende de muitos outros arquivos que não estão em seu subdiretório, tais como arquivos de cabeçalhos (H), arquivos de bibliotecas (LIB). Programadores experientes estão familiarizados com makefiles. Um makefile armazena as opções de compilação e linker, bem como expressa o inter-relacionamento dos arquivos fontes. O programa make lê esses makefile e chama o compilador, o montador, o compilador de recursos, e linker para produzir a saída final, que geralmente é um arquivo executável. O programa make usa regras de inferência interna para saber como chamar o compilador para criar um arquivo OBJ a partir de seu código fonte CPP.

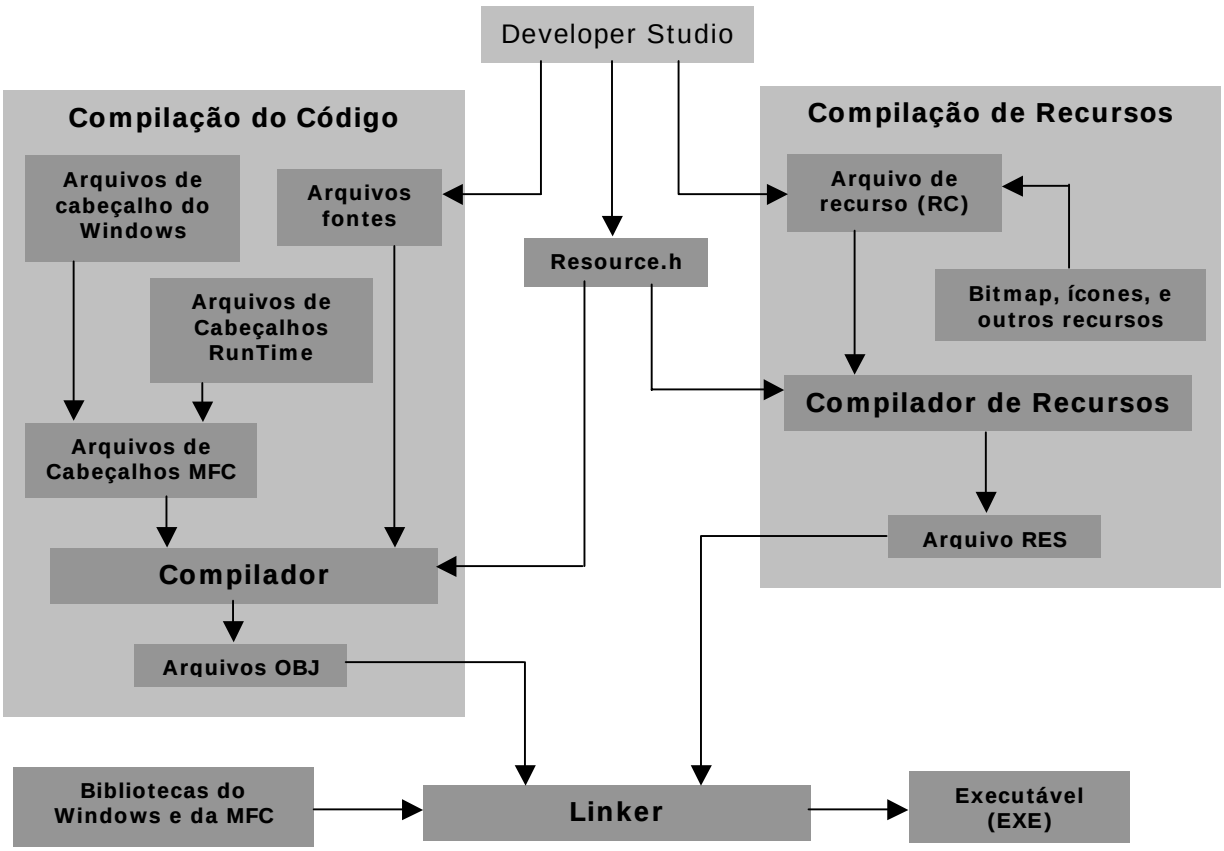


Figura 1 – Processo de construção do Visual C++

Em um projeto do Visual C++ 5.0, não há makefile (com a extensão MAK) até que você especifique ao sistema para exportar um. Um arquivo de projeto no formato texto (com a extensão DSP) serve para o mesmo propósito. Um outro arquivo texto para o

workspace (com a extensão DSW) possui uma entrada para cada projeto que compõe este workspace. É possível ter vários projetos dentro de um mesmo workspace, porém todos exemplos deste curso terão apenas um projeto por workspace. Para trabalhar com um projeto já existente, abra o arquivo com a extensão DSW dentro do Developer Studio e será possível manipular este projeto.(editar compilar,etc.).

Arquivos intermediários também são criados pelo Developer Studio. A tabela a seguir contém uma lista dos arquivos que são gerados pelo Developer Studio para cada workspace:

Extensão	Descrição
APS	Utilizado pelo Resouce View
BSC	Arquivo de informações do Browser
CLW	Utilizado pelo ClassWizard
DSP	Arquivo do Project*
DSW	Arquivo do Workspace*
MAK	makefile externo
NCB	Utilizado pelo ClassView
OPT	Armazena as configurações do workspace
PLG	Arquivo de log do Build

* do not delete or edit in text editor.

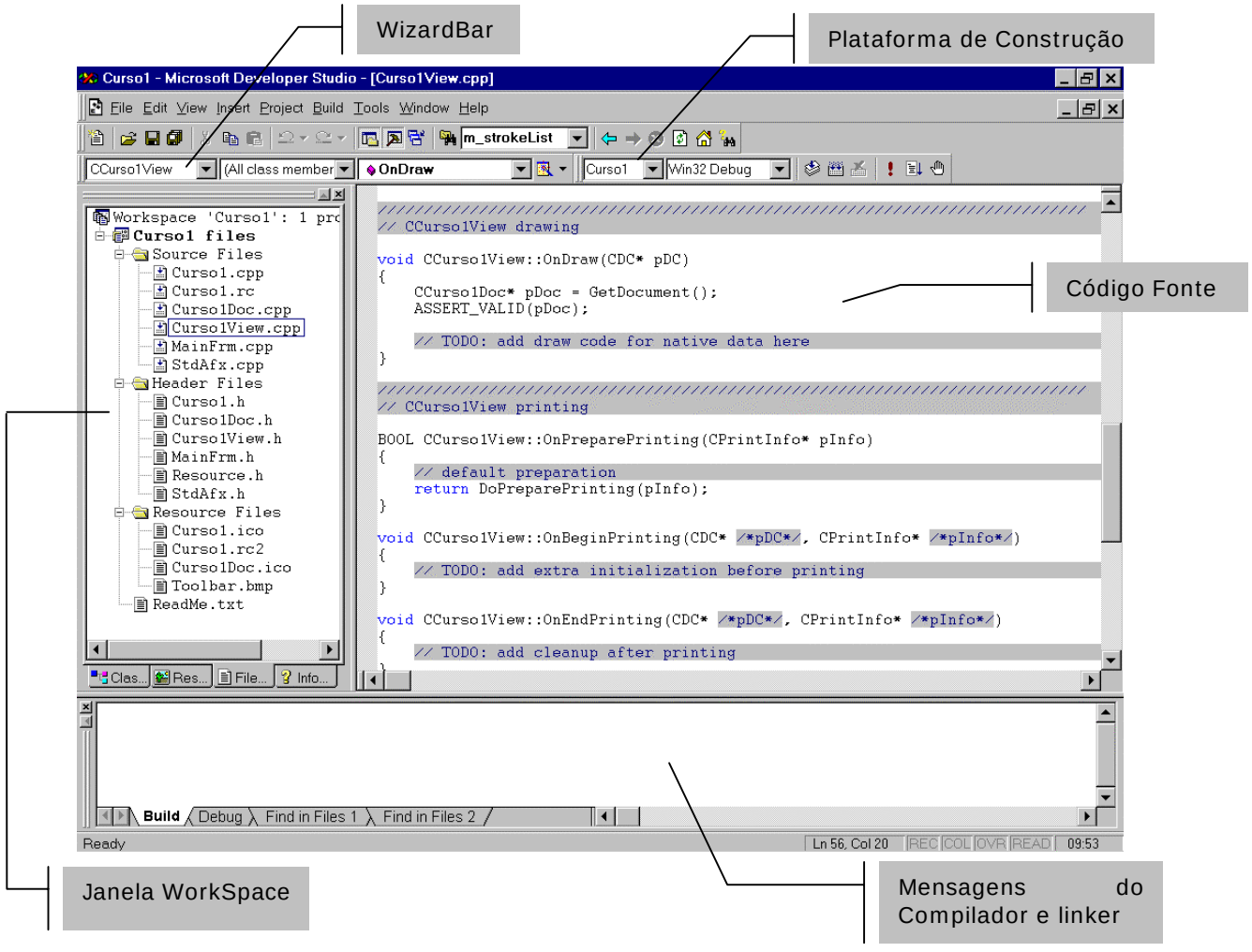


Figura 2 – Visual C++ e Developer Studio

2.2 O Editor de Recursos - Workspace ResourceView

Clicando-se na pasta ResourceView na janela Workspace do Visual C++, é possível selecionar um recurso para a edição. A janela principal é anfitriã de um editor de recurso destinado a cada tipo de recurso selecionado. A janela pode hospedar um editor wysiwyg para menus e um potente editor gráfico para caixas de dialogo, e inclui ferramentas para editar ícones, bitmaps, e strings. O editor de diálogos permite ainda que sejam adicionados controles ActiveX, controles padrões do Windows os novos Windows common controls.

Usualmente, cada projeto contém um arquivo de recurso script (RC) no formato texto, que descreve os menu, diálogos, string, e aceleradores do projeto. O arquivo RC possui declarações *#include* para trazer recursos de outros subdiretórios. Estes recursos incluem itens específicos do projeto, como arquivos de bitmap (BMP) e de ícone (ICO), e recursos comum a todo programa em Visual C++ tal como strings de mensagem de erro. A edição do arquivo RC fora do editor de recurso não é recomendada. Os editores de recurso também podem processar arquivos EXE e DLL, assim pode-se utilizar o clipboard para "roubar" recursos, como bitmaps e ícones de outras aplicações do Windows.

2.3 O Compilador C/C++

O compilador do Visual C++ pode processar tanto arquivos fontes em ***C*** e em ***C++***. A linguagem é determinada pela extensão do arquivo do código fonte. Um extensão ***C*** indica um código fonte em C, e ***CPP*** ou ***CXX*** indica um código fonte em ***C++***. O compilador é compatível com todos padrões ANSI, incluindo as últimas recomendações de um grupo de trabalhos em bibliotecas C++, e também possui Microsoft extensions. Templates, exceptions e runtime type identification (RTTI) são plenamente suportados no Visual C++ versão 5.0. A nova Standard Template Library (STL) é também incluída, embora não foi integrado na MFC.

2.4 O Compilador de Recursos

O compilador de recursos do Visual C++ lê o arquivo script ASCII de recurso (RC) do editor de recursos e escreve um arquivo de recurso binário (RES) para o linker.

2.5 O Linker

O linker processa os arquivos **OBJ** e **RES**, produzidos pelo compilador C/C++ e o compilador de recursos, e acessa os arquivos LIB para os códigos da MFC, runtime library, e do Windows. É responsável pela criação do arquivo executável EXE do projeto. A opção de link incremental minimiza o tempo de execução quando pequenas mudanças são feitas nos arquivos de códigos fontes. Os cabeçalhos dos arquivos da MFC contém blocos especiais *#pragma* (diretivas de compilação) que especificam os arquivos LIB que são necessários, assim você não tem que contar explicitamente para o linker quais bibliotecas ele deve ler.

2.6 O Debugger

Caso seu programa funcione da primeira vez, não é necessário usar o debugger. Todos nós poderiamos precisar dele de vez em quando. O debugger do Visual C++ tem melhorado continuamente, contudo não fixa de fato os bugs. O debugger trabalha junto com o Developer Studio para assegurar que os breakpoints são armazenados em disco. Existem botões na barra de ferramentas para inserir e remover breakpoints e controlar a execução passo a passo. A Figura 3 ilustra o debugger do Visual C++ em ação. Notar que as janelas ***Variables*** e ***Watch*** podem expandir um ponteiro de um objeto para mostrar todos seus membros das classes derivadas e base. Se o cursor do mouse for posicionado

sobre uma variável, o debugger mostra seu valor em uma pequena janela (Tooltip). Para debugar um programa, é necessário construí-lo (build) com as opção e de compilador e linker setadas para gerar informações de debug.

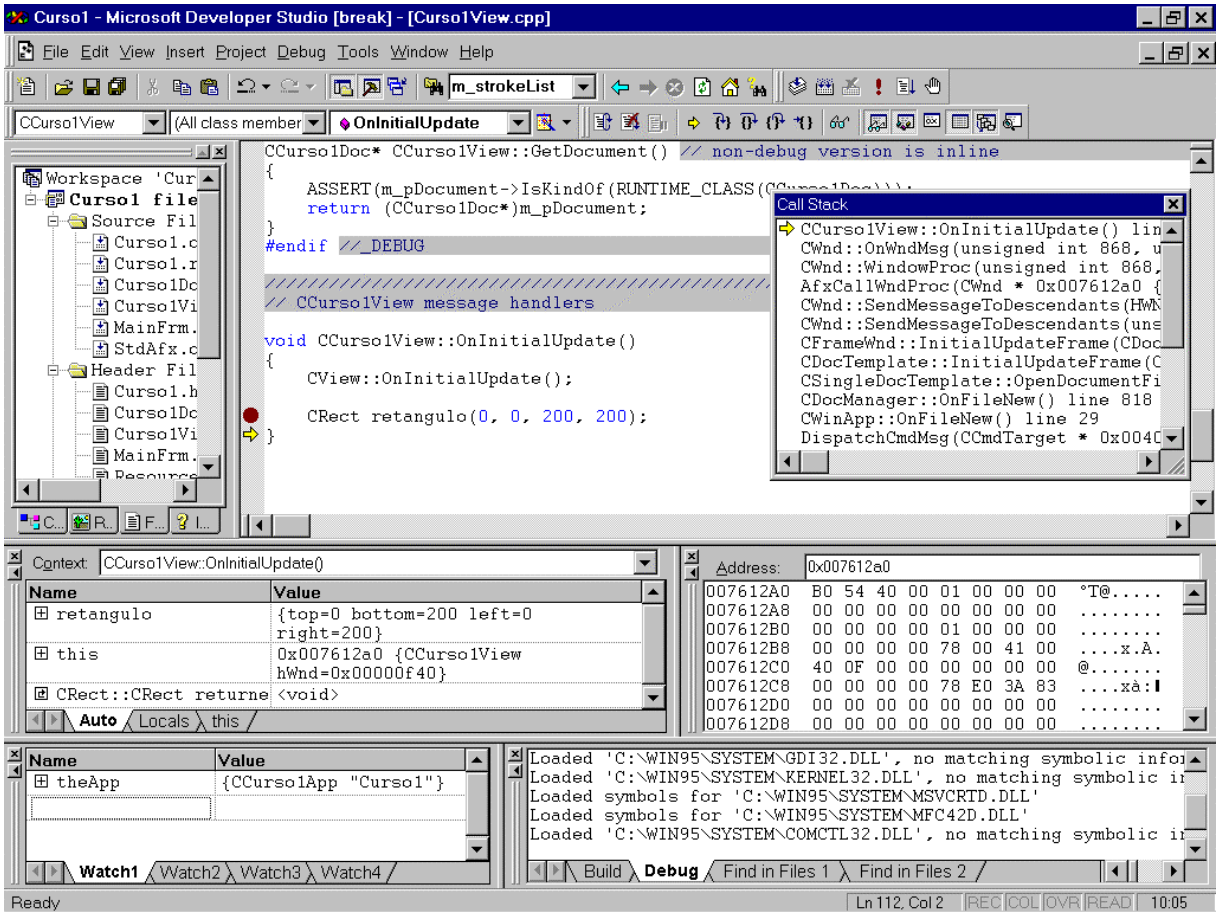


Figura 3 – O Debugger em Ação

2.7 AppWizard

O gerador de código que cria todo o esqueleto de um aplicativo baseado em Windows é chamado de AppWizard. Os nomes das classes geradas, e de seus arquivos de código fonte, é baseado no que foi especificado em suas caixas de diálogo. O AppWizard será utilizado exaustivamente por todo este curso, para gerar os programas de exemplo. Não confunda o AppWizard com os antigos geradores de código, que geravam todo o código para um aplicativo. O código gerado pelo AppWizard minimalista; a funcionalidade está dentro das classes bases da application Framework. O AppWizard permite que você inicialize uma aplicação nova rapidamente.

2.8 ClassWizard

ClassWizard é um programa (implementado como uma DLL) que está acessível através do menu View do Developer Studio. O ClassWizard o alivia da maçante função de manter e gerar um código de classe no Visual C++. Se precisar de uma classe nova, de uma função virtual, ou uma nova função de manipulador de mensagem nova? O ClassWizard escreve os protótipos, os corpos de função, e (se necessário) o código para unir a mensagem do Windows para a função. ClassWizard pode atualizar códigos de classe já escritos, assim pode-se evitar os problemas de manutenção comum a geradores de código ordinários. Algumas características do ClassWizard estão disponíveis na WizardBar do Developer Studio, como mostrado na Figura 2.

2.9 The Source Browser

Se você escreve uma aplicação do nada, você provavelmente tem um bom quadro mental de seus arquivos de código fonte, classes, e funções membros. Se você assume um programa de outra pessoa, você precisará de alguma ajuda. O Visual C++ Source Browser (o browser, resumidamente) permite que você examine (e edite) uma classe ou função de um aplicativo do ponto de vista da função em vez de do ponto de vista do arquivo. É um a ferramentas "o inspetor" disponível par a as bibliotecas orientadas a objeto no Smalltalk. O browser permite os seguintes modos de pontos de vista:

- **Definições de Referencias –** você pode selecionar qualquer função, variável, tipo, classe ou macro e poderá ver onde ela está definida em seu projeto.

- **Call Graph/Caller Graph –** Para uma função selecionada é possível ver uma representação gráfica de quais funções a chamam e quais funções ela chama.

- **Derived Class Graph/Base Class Graph –** Mostra o diagrama gráfico de hierarquia de classes. Para uma classe selecionada é possível ver seus membros e os de sua classe base. A expansão da hierarquia pode ser controlada pelo mouse.

- **File Outline –** Para um arquivo selecionado, as classes, funções e membros de dados são mostradas onde são definidas e usadas em seu projeto.

NOTA: Se simplesmente uma linha de código mudar de posição, em qualquer um dos códigos fontes que pertençam ao projeto, o banco de dados para o browser tem que ser regerado pelo Developer Studio, o que faz com que haja um aumento no tempo de compilação.

Além do Browser, o Developer Studio tem a opção de utilizar a janela ClassView do Workspace que não depende da geração de uma base de dados para o browser. Ela é uma *tree view* com todas as classes do projeto, mostrando as funções membros e variáveis membros. Basta um duplo click em um de seus elementos, e o código fonte onde está a declaração ficará visível imediatamente. A janela ClassView não contém informações sobre a hierarquia das classes que está disponível no browser.

2.10 Ajuda Online

Um modo completamente novo de ajuda online baseado em HTML foi incluído no Developer Studio 97. Cada tópico foi convertido em um documento HTML individual; e todos foram combinados em um arquivo *InfoViewer title* (IVT) com seu respectivo arquivo de *InfoViewer index* (IVI). A janela InfoView utiliza o código do Microsoft Internet Explorer 3.0, que trabalha como um Web browser já conhecido. Developer Studio pode acessar arquivos IVI do CD-ROM Visual C++ ou do disco rígido, e ainda pode acessar arquivos HTML na Internet.

Quatro formas de acesso a ajuda online são permitidas no Developer Studio:

- By book – Quando é selecionado a opção *Contents* no menu Help do Developer Studio, a janela Workspace é trocada para o modo InfoView. Nela a documentação sobre Developer Studio, Visual C++, e Win32 SDK é organizada hierarquicamente por livros e capítulos.

- By topic - Quando é selecionado a opção *Search* no menu *Help* do Developer Studio e a pasta *Index*, uma palavra chave pode ser digitada e é possível visualizar os tópicos e artigos que incluem essa palavra.

- By word - Quando é selecionado a opção *Search* no menu *Help* do Developer Studio e a pasta *Query*, é possível digitar uma combinação de palavras e ver os artigos que contém essas palavras.

- F1 help – Esse é o melhor amigo do programador. Movimente o cursor para dentro de uma a função, macro, ou nome de classe, e pressione a Tecla F1 e o help do sistema faz todo o trabalho. Se esse nome constar em mais de um arquivo de help, uma lista é mostrada para se selecionar o tópico desejado.

Seja qual for o modo que o help online foi acessado, é possível copiar qualquer texto de ajuda para o clipboard para inclusão em seu programa.

2.11 Ferramentas de Diagnóstico

O visual C++ contém várias ferramentas úteis para diagnóstico. SPYXX lhe dá uma visão da árvore dos processos de seu sistema, threads, e janelas. Também lhe deixa ver mensagens e examina as janelas de aplicações correntes. Você achará útil por finalizar terefas errôneas que não estão visíveis na lista de tarefas Windows 95. O visual C++ também inclui várias ferramentas para ActiveX, um programa de teste de controles ActiveX, o HCRTF compilador de help, gerenciador de biblioteca, viewers e editores de arquivos binários, e outras ferramentas.

Porém, a ferramenta DBWIN das versões de 16-bit do Visual C++, não existem mais. Essa tarefa foi incorporada pelo Developer Studio Debugger, e para ver as mensagens de depuração de seu programa é necessário executa-lo no modo Debug.

2.12 A Galeria

A galeria do Developer Studio permite que se compartilhem componentes de software entre diferentes projetos. A Galeria gerencia três tipos de módulos:

- **ActiveX controls –** quando é instalado um controle ActiveX (OCX - OLE control), uma entrada é feita dentro do Registro do Windows. Todos os controles ActiveX registrados aparecem na janela da Galeria, e podem ser selecionados em qualquer projeto.
- **C++ source modules** – quando uma nova classe é escrita, esse código pode ser adicionado na Galeria. Esse código pode ser selecionado e copiado nos outros projetos. Recursos também pode ser adicionados a Galeria.
- **Componentes do Developer Studio -** A Galeria contém ferramentas que podem ser usadas para adicionar características nos projetos. Tal ferramenta pode inserir novas classes, funções, membros de dados, e recursos em um projeto existente. Alguns componentes são fornecidos pela Microsoft (processamento de tempo Inativo, apoio de Paleta, e Splash Screen, por exemplo) como parte do Visual C++. Outros são fornecidos por outras firmas de Software.

DICA: Se você decidir utilizar alguns dos componentes da Galeria Developer Studio, experimente primeiro em um projeto bobo para ver se é o que você realmente quer. Caso contrário, pode ser difícil de remover o código gerado de seu projeto regular.

Todos os códigos gerados para a Galeria podem ser importados de um ou exportados para um arquivo OCX. Estes arquivos fazem parte de um novo meio de distribuição e compartilhamento para componentes do Visual C++.

2.13 A Microsoft Foundation Class Library Versão 4.21

A biblioteca Microsoft Foundation Class versão 4.21 (a biblioteca MFC) é realmente o assunto desse curso. Ela define a estrutura dos aplicativos, item a ser amplamente explorado. A seção 3 contém um exemplo de código fonte e introduz alguns conceitos importantes.

2.14 A Microsoft ActiveX Template Library

ATL é uma ferramenta, separada da MFC, designada para construir controles ActiveX. Os controles ActiveX podem ser criados tanto usando a MFC quanto a ATL, mas com a ATL os controles ficam menores e consequentemente mais rápidos de serem carregados via Internet.

3 Microsoft Foundation Class Library Application Framework (Overview)

As próximas seções:

- Porque utilizar uma Estrutura de Aplicações?
- O que é uma Estrutura de Aplicações ?
- Mapeamento de Mensagens na MFC Library
- Documents e Views

o introduzem na estrutura das aplicações da biblioteca Microsoft Foundation Class versão 4.2.1 (a MFC) e explica seus benefícios. Um primeiro exemplo é mostrado na página 27 onde é possível introduzir parte da operacionalidade da MFC library na programação para Microsoft Windows e começar a entender o conceito de de estrutura de aplicação. Teoria é mantida a um mínimo aqui, mas as seções sobre mapeamento de mensagens e em documents e views contem informações importantes que o ajudará com os exemplos que virão a seguir.

4 Porque utilizar uma Estrutura de Aplicações?

Se você for desenvolver um aplicativo para o Windows, terá que escolher um ambiente de desenvolvimento. Assumindo que opções não-C como Microsoft Visual Basic e Borland Delphi, foram rejeitadas, aqui estão algumas opções restantes:

- Programar em C com a Win32 API,
- Escrever sua prórpria biblioteca de classes em C++ para o Windows que utilize a Win32 API,
- Utilizar a estrutura de aplicações da MFC,
- Utilizar outra estrutura de aplicações baseada em Windows como a Object Windows Library (OWL) da Borland.

Se você é a partir do zero, qualquer opção envolve uma curva de aprendizagem grande. Se você já é um programador Win16 ou Win32, você ainda terá uma curva de aprendizagem com a MFC libray. Assim que benefícios podem justificar este esforço?

Não quero parecer um pastor da Microsoft. Mas a MFC foi totalmente aceita pelo mercado e até mesmo foi usado por outro publicador de compilador como Symantec. Bem

como pela maioria de fabricantes de software profissionais do mercado como a Oracle, AutoDesk, entre outras. A própria Microsoft utiliza o Visual C++ para desenvolver seus aplicativos como por exemplo o pacote do Office e até mesmo o Internet Explorer. Ainda assim é uma boa idéia, navegar pelas características desta escolha de programação.

A MFC Library é a Microsoft Windows API em C++. Assumindo a premissa que a linguagem C++ é agora um padrão para plataforma de desenvolvimento de aplicativos profissionais, você teria que dizer que é natural para Windows ter uma interface de programação em C++. Que interface melhor há, do que aquela produzida pelo pessoal da Microsoft, criador de Windows? Essa interface é a MFC library.

Aplicativos que usam uma Application framework possuem uma estrutura padronizada. Qualquer programador que começa em um projeto grande desenvolve algum tipo de estrutura para o código. O problema é que a estrutura de cada programador é diferente, e é difícil para um membro novo do time aprender a estrutura se conformar a isto. A estrutura de aplicação da MFC library inclui sua forma própria para uma aplicação - uma que é aprovado em muitos ambientes de software e em muitos projetos. Se você escreve um programa para Windows que usa a MFC, você pode se aposentar seguramente para uma ilha caribenha e saber que seus subordinados podem manter e aumentar seu código facilmente em casa.

Não pense que a estrutura da MFC torna seus programas inflexíveis. Com a MFC, seu programa pode chamar funções Win32 a qualquer hora a, assim você poder tirar proveito do Windows.

As ferramentas do Visual C++ reduzem a tarefa de codificar. O editor de recursos do Developer Studio, AppWizard, e ClassWizard reduzem significativamente o tempo necessário para escrever código que é específico da sua aplicação. Por exemplo, o editor de recursos cria um arquivo de cabeçalho que contém valores constantes de #define para os identificadores. AppWizard gera o esqueleto do código para sua aplicação inteira, e ClassWizard gera protótipos e corpos de função para manipuladores de mensagem.

A estrutura de aplicações da MFC Library é rica em características. A versão 1.0 da MFC library, continha todas as classes suportadas pelo Microsoft C/C++ versão 7.0, incluindo as seguintes características:

- Uma interface em C++ para a API do Windows API
- Classes de propósito geral (não específicas do Windows), incluindo
  - Classes de Collection para listas, arrays, e maps
  - Uma eficiente e útil classe para strings
  - Classes para manipular horas e datas
  - Classes de acesso a arquivos que permitem independência do sistema operacional
  - Suporte a objetos de armazenamento e recuperação de dados em disco.
- Uma classe raiz comum a toda a hierarquia
- Suporte a aplicativos do tipo MDI (Multiple Document Interface)
- Algum suporte para OLE versão 1.0

A MFC library versão 2.0 classes (no Visual C++ 1.0) atualizou a versão 1.0 para permitir o uso de muitas características comuns em programas baseados em Windows, além de introduzir o conceito de Estrutura de Aplicações. A seguir a lista das mais importantes características adicionadas:

- Suporte total aos itens de menu File Open, Save, and Save As, além de permitir o uso da lista de mais recentes usados
- Suporte a Impressão e visualização de impressão
- Suporte para scrolling windows e splitter windows
- Suporte a barra de ferramentas e barra de status
- Acesso aos controles do Microsoft Visual Basic
- Suporte para help sensitivo ao contexto
- Suporte automático para processameto de dados em uma caixa de diálogo.
- Uma atualização para melhor suportar OLE 1.0
- Suporte a DLL's

A MFC library 2.5 (no Visual C++ 1.5) contribuiu com o seguinte:

- Suporte a ODBC (Open Database Connectivity) que permite que seus aplicativos acessem dados armazenados com os formatos dos mais populares bancos de dados como o Microsoft Access, FoxPro, e Microsoft SQL Server
- Uma interface para OLE 2.01, que permite a edição de um objeto no local, linking, arrastar e soltar, e OLE Automation

O Visual C++ 2.0 foi a primeira versão em 32-bit do produto, Que incluía suporte para Windows NT 3.5. Esta é a versão 3.0 da MFC library, que incluiu as seguintes características:

- Suporte a Tab dialog (property sheet)
- Docking control bars implementadas dentro da MFC
- Suporte a thin-frame windows
- Em anexo a ferramenta Control Development Kit (CDK) para construir controles OLE (OCX) para 16 e 32 bits,

A versão 2.1 do Visual C++, que inclui a MFC 3.1, adicionou o seguinte:

- Suporte para os novos common controls do Microsoft Windows 95 (beta)
- Novos drivers de ODBC para integrar com a Access jet database engine
- Classe Winsock que permite a comunicação via TCP/IP

A Microsoft decidiu pular a versão 3.0 do Visual C++ e ir diretamente a versão 4.0 com o intuito de sincronizar a versão do Visual C++ com a versão da MFC. MFC 4.0 contém as seguintes características adicionais:

- New OLE-based Data Access Objects (DAO) classes for use with the Jet engine
- Use of the Windows 95 docking **control** bars instead of the MFC control bars
- Full support for the common Controls in the released version of Windows 95, with new tree view and richedit view classes
- New classes for thread synchronization
- OLE Control container support

Visual C++ 4.2 (subscription) inclui a MFC 4.2. As seguintes características foram adicionadas:

- Winlnet classes
- ActiveX Documents server classes
- ActiveX synchronous and asynchronous moniker classes
- Enhanced MFC ActiveX Control classes, with features such as window less activation, optimized drawing code, and so forth

- Improved MFC ODBC support, including recordset bulk fetches and data transfer without binding

A versão 5.0 do Visual C++ com a MFC 4.21, (com alguns bugs da 4.2 corrigidos). Dentre as principais caracteríticas adicionadas ao Visual C++ 5.0 estão:

- A redesigned IDE, Developer Studio 97, which includes an HTML-based online help system and integration with other languages, including Java
- The ActiveX Template Library (ATL) for efficient ActiveX Control construction for the Internet

4.1 A curva de aprendizado

Todos os benefícios listados acima soam muito bem, não? Você provavelmente está pensando "Você não adquire algo por nada". Sim, isso é verdade. Para usar a Estrutura das Aplicações efetivamente ela deve ser aprendida por completo, e isso leva tempo. Se você tem que aprender C++, Windows, e a MFC library (sem OLE) tudo ao mesmo tempo, levará seis meses pelo menos antes de você se torne realmente produtivo. De maneira interessante isso está perto do tempo de aprendizagem só do Win32 API.

Como isso pode acontecer se a MFC library oferece muito? Em primeiro lugar, você pode evitar muitos detalhes de programação em linguagem C que programadores Win32 são forçados a aprender. Uma estrutura de aplicações orientada a objeto, torna a programação para Windows mais fácil para se aprender, uma vez você entenda a programação orientada a objetos.

A biblioteca de MFC não trará a programação Windows até as massas. Programadores de aplicações para Windows normalmente demandam salários mais altos que outros programadores e esta situação continuará. A curva de aprendizado da MFC library, junto com o poder da estrutura das aplicações, deveria assegurar que programadores em MFC library continuarão estando em forte demanda.

5 O que é uma Estrutura de Aplicações ?

Uma definição para estrutura das aplicações (application framework) é "uma coleção integrada de componentes de software orientado a objeto que oferece tudo que é necessário para a construção de um Aplicativo genérico". Está é uma definição muito útil, não? Se você realmente quer saber o que uma estrutura das aplicação é, você terá que ler o resto desta apostila. O exemplo de Estrutura de Aplicação nesta seção é um bom ponto de partida para este conhecimento.

5.1 Uma Estrutura de Aplicações X uma biblioteca de Clases

Uma das razões para a linguagem C++ ser popular que é ela pode ser “estendida” com bibliotecas de classe. Algumas bibliotecas de classe são entregues com compiladores de C++ outras são vendidas através de firmas de software, e ainda outras são desenvolvidos em casa. Uma biblioteca de classe é um conjunto de classes de C++ relacionadas que podem ser usadas em uma aplicação. Uma biblioteca de classes de matemática, por exemplo, poderia executar operações matemáticas comuns e uma biblioteca de classe de comunicações poderiam apoiar a transferência de dados em um link serial. Às vezes você constrói objetos das classes providas; às vezes você deriva suas próprias classes - tudo depende do design particular da biblioteca de classes.

Uma estrutura de aplicações é um super conjunto de bibliotecas de classes. Uma biblioteca ordinária é um conjunto isolado de classes projetada para ser incorporada em qualquer programa, mas uma Estrutura de Aplicações define a estrutura do programa. A Microsoft não inventou o conceito de Estrutura de Aplicações. Surgiu no Mundo acadêmico, e a primeira versão comercial era foi a MacApp para o Apple da Macintosh. Desde que a MFC 2.0 foi introduzida, outras companhias, inclusive Borland, lançaram produtos semelhantes.

5.2 Um exemplo de uma Estrutura de Aplicações

O exemplo abaixo é um código fonte - não pseudocódigo mas código real que de fato compila e roda com a MFC library. É o velho e bom "Hello world !", com algumas modificações. Está é a quantia mínima de código para um aplicativo Windows baseado na MFC library funcionar. Não é necessário entender todas linhas agora. Não perca tempo digitando este exemplo, espere pelos próximos capítulos onde você começará usando um aplicativo "real" usando a estrutura de aplicação.

NOTA : Por convenção, as classes da MFC library iniciam com a letra C.

A seguir está listado o código fonte dos arquivos para o cabeçalho e a implementação do aplicativo MYAPP. As classes *CMyApp* e *CMyFrame* são derivadas das classes bases da MFC library.

Arquivo de cabeçalho MyApp.h do aplicativo MYAPP:

```
// application class
class CMyApp : public CWinApp
{
public:
        virtual  BOOL InitInstance(); 1;
};

// frame window class
class CMyFrame: public CFramewnd
{
public:
        CMyFrame( );
protected:
        // “afx_msg” indica que as próximas duas funções fazem parte
        // do sistema de controle de mensagens da MFC.
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnPaint();
};
```

Este é o arquivo de implementação do primeiro exemplo.

```
#include <afxwin.h> // biblioteca da MFC onde estão declaradas as classes
bases
#include “myapp.h”

CMyApp theApp;

BOOL CMyApp::InitInstance()
{
        m_pManWnd = new CMyFrame();
        m_pManWnd->ShowWindow(m_nCmdShow);
        m_pManWnd->UpdateWindow();
        return TRUE;
}
```

```
CMyFrame::CMyFrame()
{
        Create(NULL “Aplicativo MYAPP”) ;
}

CMyFrame::OnLButtonDown(UINT nFlags, CPoint point)
{
        TRACE(“Entrando em CMyFrame::OnLButtonDown - %lx, %d, %d\n”,
                (long) nFlags, point.x, point.y);
}

CMyFrame::OnPaint()
{
        CPaintDC dc(this);
        dc.TextOut(0,0,”Hello world!”);
}
```

Aqui estão alguns elementos desse programa:

A função *WinMain* – Lembrar que o Windows requer que os aplicativos tenham a função *WinMain*. Aqui *WinMain* não é visível porque está encapsulada dentro da estrutura das aplicações.

A classe *CMyApp* – Um objeto da classe *CMyApp* representa um aplicativo. O programa define um objeto global do tipo *CMyApp*, th*eApp.* A classe base CWinApp determina a maioria do comportamento de theApp.

Inicialização do aplicativo – Quando o usuário inicializa o aplicativo, o Windows chama a função *WinMain* interna da estrutura de aplicação, e *WinMain* procura pelo construtor do objeto global do aplicativo da classe derivada de *CWinApp*. Não esquecer que num programa em C++, objetos globais são construídos antes do programa principal ser executado.

A função membro *CMyApp::InitInstance* – Quando a função *WinMain* procura o objeto aplicativo, a função membro virtual *InitInstance* é chamada, que faz as chamadas necessárias para construir e exibir a janela da mainframe do aplicativo. Você tem que sobrecarregar a *InitInstance* em sua classe derivada da aplicação porque o CWinApp que é a classe base não conhece que tipo de janela principal você quer chamar.

A função membro *CWinApp::Run* – A função *Run* está encapsulada na classe base, mas despacha as mensagens da aplicação para suas janelas e assim mantém o funcionamento da aplicação. *WinMain* chama *Run* depois que chama *InitInstance.*

A classe *CMyFrame* – Um objeto da classe *CMyFrame* representa a janela da mainframe do aplicativo. Quando o construtor chama a função membro C*reate* da classe base *CFrameWnd, o* Windows cria a estrutura da janela e a application framework a vincula a um objeto em C++. As funções *ShowWindow* e *UpdateWindow*, também membros da classe base, devem ser chamadas para exibir a janela.

A função *CMyFrame::OnLButtonDown* - Isto é uma pré-estréia da capacidade de manipular mensagens da MFC. O evento pressionar o botão esquerdo do mouse é mapeado “map” para uma função membro de *CMyFrame*. Você aprenderá os detalhes das mensagens coma a biblioteca MFC mais adiante. Por enquanto aceite que esta função é chamada quando o usuário aperta o botão de esquerdo do mouse. A função invoca a macro *TRACE da MFC* para exibir uma mensagem na janela de depuração.

A função *CMyFrame::OnPaint*- A estrutura das aplicações chama esta importante função membro mapeada na classe *CMyFrame* toda vez que for necessário redesenhar a janela: ao inicializar o programa, ao redimensionar um janela e quando qualquer parte da janela for revelada após estar sobre outra. A declaração de *CPaintDC* relaciona-se com a Interface de Dispositivo de Gráficos (GDI) e é explicado em mais adiante. A função *TextOut* exibe "Hello, world!".

Finalização do Aplicativo- O usuário termina a aplicação fechando a janela da mainframe. Esta ação inicia uma sucessão de eventos que termina com a destruição do objeto CMyFrame, a saída de *Run* e a saída de *WinMain*, e a destruição do objeto *CMyApp*.

Examine o código fonte do exemplo novamente. A maior parte da funcionalidade da aplicação está nas classes bases da MFC *CWinApp* e *CFrameWnd*. Ao escrever MYAPP, nós seguimos algumas regras simples de estrutura e escrevemos funções chaves em nossas classes derivadas. C++ nos "deixa obter emprestado" muito código sem precisar copiar isto. Pense nisto como uma sociedade entre nós e a application framework. A application framework prove a estrutura, nós provemos o código que faz a aplicação ser unica.

Agora você está começando a ver por que a application framework é mais do que uma biblioteca de classes. A application framework não faz só a estrutura de um aplicativo, mas também fornece mais que classes bases C++. Você já viu como a função encapsulada WinMain trabalha. Outros elementos suportam o processamento de mensagem, diagnósticos, DLLs, e assim sucessivamente.

6 Mapeamento de Mensagens na MFC Library

Referindo-se à função membro OnLButtonDown do exemplo anterior. Você poderia pensar que OnLButtonDown seria um candidato ideal uma função virtual. Uma classe base janela definiria funções virtuais para mensagens de evento de mouse e outras mensagens padrões, e as classes derivadas da janela poderiam sobrecarregar as funções quando necessário. Algumas bibliotecas de classe para Windows trabalham deste modo.

A application framework da MFC library não utiliza funções virtuais para as mensagens do Windows. Ao invés disso, utilizam macros para "mapear" as mensagens especificadas para funções membros das classe derivadas. Porque esta rejeição com as funções virtuais? Suponha que a MFC usasse funções virtuais para as mensagens. A classe *CWnd* teria que declarar as funções virtuais para 110 mensagens. C++ requer uma tabela de dispacho para funções virtuais, chamada vtable, para classe derivada que for utilizada no programa. Cada vtable precisa de uma entrada de 4-byte para cada função virtual, mesmo que as funções não são sobrecarregadas de fato na classe derivada. Assim, para cada tipo de janela ou controle, a aplicação precisaria de uma tabela de 440-byte para apoiar os manipuladores de mensagem virtuais.

E sobre os manipuladores de mensagem para mensagens de comando de menu e mensagens de click de botão? Estes não podem ser definidos estes como funções virtuais em uma classe base de janela porque cada aplicação pode ter um conjunto diferente de comandos de menu e botões. O sistema de mapeamento de mensagens da MFC evita vtables grandes, e acomoda mensagens de comando específicas da aplicação em paralelo com as mensagens ordinárias do Windows. Também permite non-window classes, como classes de documento e a classe de aplicação, manipulem mensagens de comandos. MFC usa macros para conectar ("map") mensagens do Windows para funções membros em C++. Nenhuma extensão para a linguagem C++ é necessária.

Um manipulador de mensagem na MFC requer um protótipo de função, um corpo de função, e uma entrada (chamada da macro) no mapa de mensagem. ClassWizard lhe ajuda a acrescentar os manipuladores de mensagem em suas classes. Você seleciona um identificador de mensagem do Windows ID de um list box, e o assistente gera o código com os parâmetros e valores de retorno correto para a função.

7 Documents e Views

O exemplo acima usou um objeto application e um objeto frame window. Muitos aplicativos baseados na MFC library serão mais complexos. Tipicamente, conterão além das classes application e frame, mais duas outras classes que representam o "document" e a "view". Esta *arquitetura document-view* é o coração da estrutura das aplicações e está baseado nas classes de Model/View/Controller do mundo do Smalltalk.

Um "*document*" é um objeto de dados que é manipulado pelo usuário em uma seção de edição. É criado pelos comando do menu File New ou Open e são tipicamente armazenados em arquivos. Uma "*view*" é um objeto janela que permite que o usuário interaja com o documento ao qual ela está associada.

Em termos gerais, a arquitetura de document-view separa os dados, da visão pelo usuário dos dados. Um benefício óbvio é visões múltiplas dos mesmos dados. Considere um documento que consiste no preço de um mês de citações acionárias armazenado em disco. Suponha que existam uma visão de tabela e uma visão gráfica dos dados. O usuário atualiza valores na janela da tabela, e a janela do gráfico muda porque ambas as janelas exibem a mesma informação (mas em visões diferentes).

Em aplicativos da MFC library, documents e views são representados por instâncias de classes C++.

A classe base do document interage com menu File Open e File Save; a classe de document derivada faz a leitura e escrita dos dados do objeto de documento. (A estrutura das aplicações faz a maioria do trabalho de exibir o diálogo de abertura de Arquivos e gravação de arquivos e abrindo, fechando, lendo, e escrevendo arquivos.) A classe base view representa uma janela que é contida dentro de uma janela de frame; a classe de visão derivada interage com sua classe de documento associada e faz a exibição da aplicação e impressão. As classes view derivada e base manipulam as mensagens de Windows. A MFC libray organiza todas as interações entre documentos, views, frame-window, e o objeto application, principalmente por funções virtuais.

Não pense que um objeto de documento deve ser associado com um arquivo de disco que é lido completamente na memória. Se um "documento" real fosse um banco de dados, por exemplo, pode-se sobrecarregar as funções membro que servem para selecionar o documento a função que responde ao menu File Open para mostrar uma lista de bancos de dados em vez de uma lista de arquivos.

Capítulo 2
Iniciando com o AppWizard

Neste Capítulo é abordado um dos componentes da arquitetura Document-View, a classe view, que está intimamente relacionada com a janela. Por hora os outros três componentes básicos da estrutura de um programa com a MFC não serão abordados.

Isto será feito de forma prática demonstrando a utilização do AppWizard para criar um esqueleto de um programa Windows.

É claro que estes exemplos desse capítulo não serão capazes de armazenar dados no disco e não suportará múltiplas janelas, suportes estes que serão abordados nos próximos capítulos.

Como os recursos são uma parte importante de um programa para Windows, será abordada uma introdução do uso da ResourceView, para visualizar e explorar o programa. Serão informadas ainda dicas de como melhorar o tempo de compilação e melhorar a velocidade de execução de ser programa.

1 O que é uma View?

Do ponto de vista de um usuário, uma view é uma janela que pode ser redimensionada, arrastada, fechada do mesmo modo que qualquer outro programa do Windows.

Na visão do programador, uma view é um objeto C++ de uma classe derivada da classe *CView* que pertence a biblioteca MFC. Como qualquer outro objeto em C++, ela é determinada por seus membros e herda as características de suas classes bases.

Com o Visual C++, é possível criar aplicativos interessantes, simplesmente pela adição de códigos a classe derivada da *CView*, que é criada pelo gerador de código AppWizard.

Como é de costume em um programa em C++ o código da classe view está dividido em dois arquivos, um arquivo de cabeçalho (H), e um arquivo de implementação (CPP).

2 SDI x MDI

Dois tipos de aplicativos distintos são suportados pela MFC: Single Document Interface (SDI) e Multiple Document Interface (MDI). Um aplicativo SDI, do ponto de vista do usuário é apenas uma janela. Se este aplicativo depende de um arquivo em disco “document”, apenas um será suportado por este tipo de aplicativo. Exemplo NotePad. Um aplicativo MDI, suporta múltiplas janelas filhas, cada uma associada a um documento correspondente. Exemplo Word.

Quando o AppWizard é executado, o tipo MDI é o padrão sugerido. Certifique-se de mudar essa opção para criar os primeiros exemplos. O tipo MDI será abordado mais tarde.

3 Exemplo Curso1a

Nesta seção será abordado um exemplo completo de esqueleto de programa, bem como os passos necessários para sua criação utilizando o AppWizard. Este exemplo é apenas uma janela vazia com um menu e uma barra de ferramentas acoplada.

Quando você criar o primeiro programa executável MFC , você usará o MFC AppWizard (versão de EXE) que o conduzirá por uma série de caixas de diálogo nas quais você escolhe opções para a arquitetura, características, e funções de seu projeto.

A série é um caminho ramificado: você pode avançar pelos passos e pode fazer mudanças às opções que você selecionou. Ajuda está disponível para toda opção no MFC AppWizard: click o botão direito do mouse no controle para mais informação sobre cada opção.

Passos para criar um programa MFC EXE usando a ferramenta AppWizard:

1. No menu File, click New e então clique na pasta de Projects.
2. Especifique o Nome de Projeto, Localização, Workspace, Dependência, e opções de Plataformas e então click duas vezes no ícone MFC AppWizard (exe).
3. Complete os passos do MFC AppWizard escolhendo as opções apropriadas para seu programa.
4. Prepare seu programa MFC para uso. (Criar o executável)

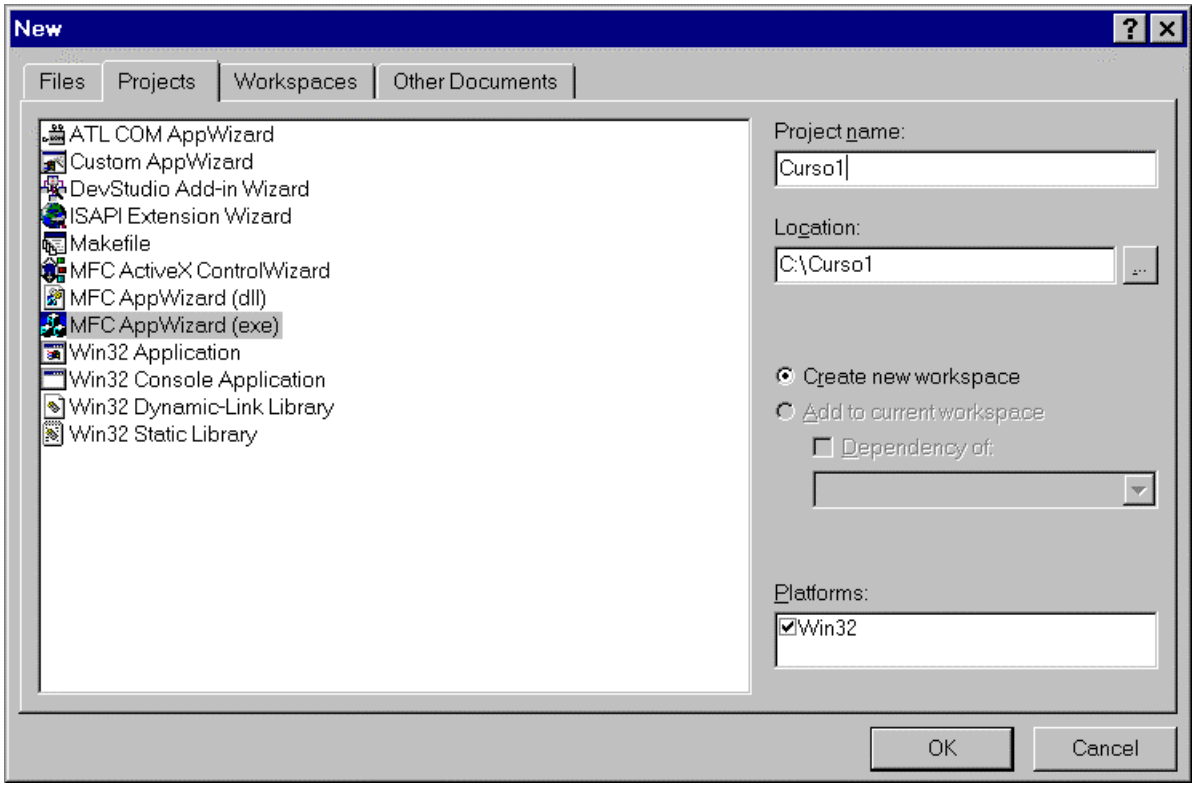


Figura 4 – Diálogo de Tipos de novos projetos.

3.1 O MFC AppWizard (versão de EXE): Passos

O MFC AppWizard é uma série de caminho ramificado (Assistente) de 4 ou 6 passos que dependem da arquitetura que você selecione para seu programa. Você pode avançar pelos passos e pode fazer mudanças às opções que você selecionou.

Ajuda está disponível para toda opção em cada passo: A seguir encontra-se uma explicação passo a passo do MFC AppWizard.

Passo 1

1- Escolha uma das três arquiteturas para seu programa: Single Document (SDI), Multiple Document (MDI), ou Dialog-Based.
2- Selecione um idioma para o texto de recurso e então click Next.
3- Se você selecionou uma arquitetura Dialog-Based, o Passo 2 se encontra na próxima seção.

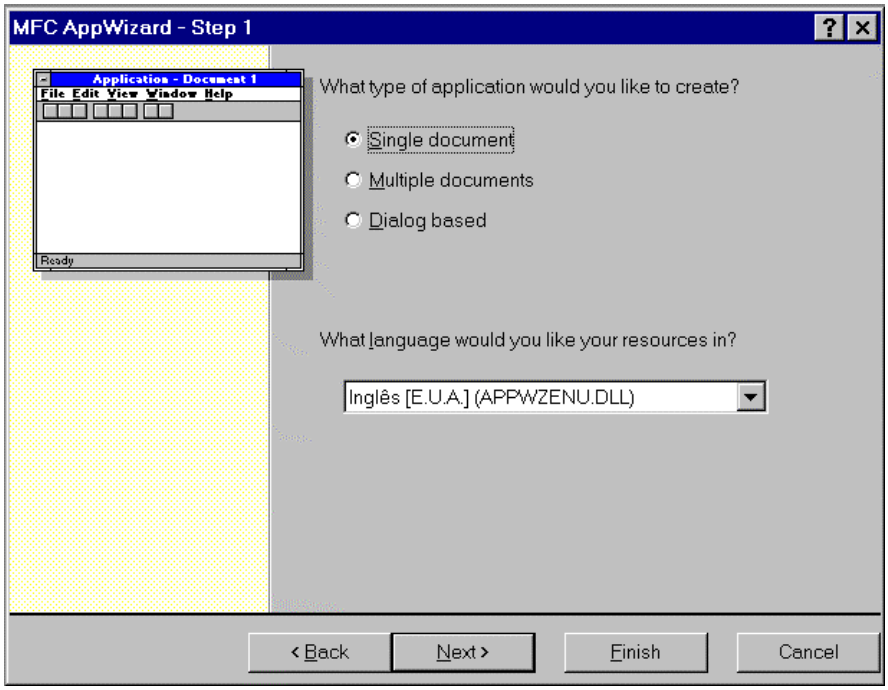


Figura 5 – Passo 1 do MFC AppWizard.

NOTA: Para o exemplo Curso1 selecione a opção SDI.

Passo 2 (para programas SDI ou MDI)

1- Escolha uma das quatro opções de apoio de banco de dados para seu programa: None, Header file support, Database View (with file support) ou Database View (without file support).
2- Se você selecionou Database View support, click Data Source e escolha qualquer uma base de dados externa via ODBC ou um banco de dados DAO com o arquivo apropriado e uma tabela a ser utilizada.
3- Click Next.

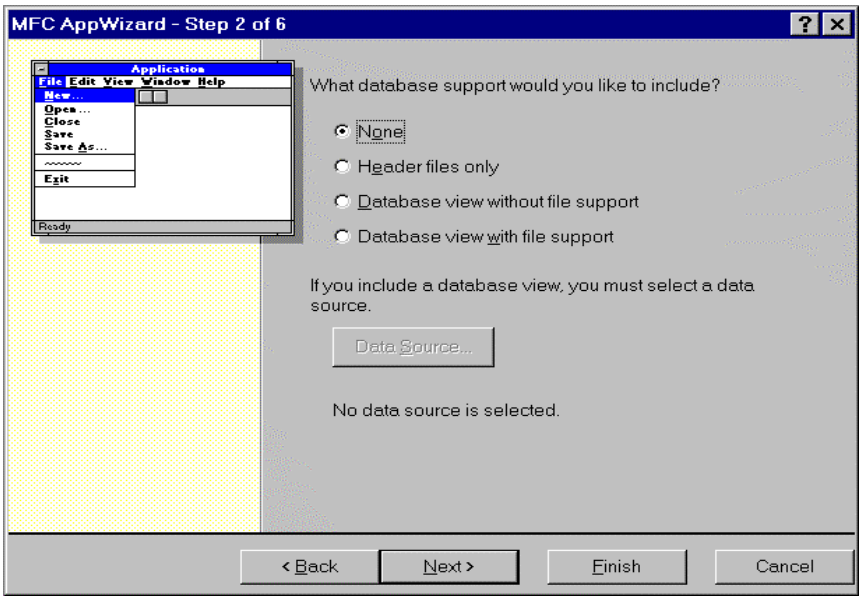


Figura 6 – Passo 2 do MFC AppWizard para SDI ou MDI.

NOTA: Para o exemplo Curso1, não selecione nenhum suporte a banco de dados, simplesmente aceite o padrão.

Passo 3 (para programas SDI ou MDI)

1- Escolha a combinação de apoio de documento que você gostaria de incluir em seu programa e então click Next. Selecionando estas opções habilita-se os recursos de ActiveX e adiciona Automatização extra comanda ao menu da aplicação.
2- Click Next.

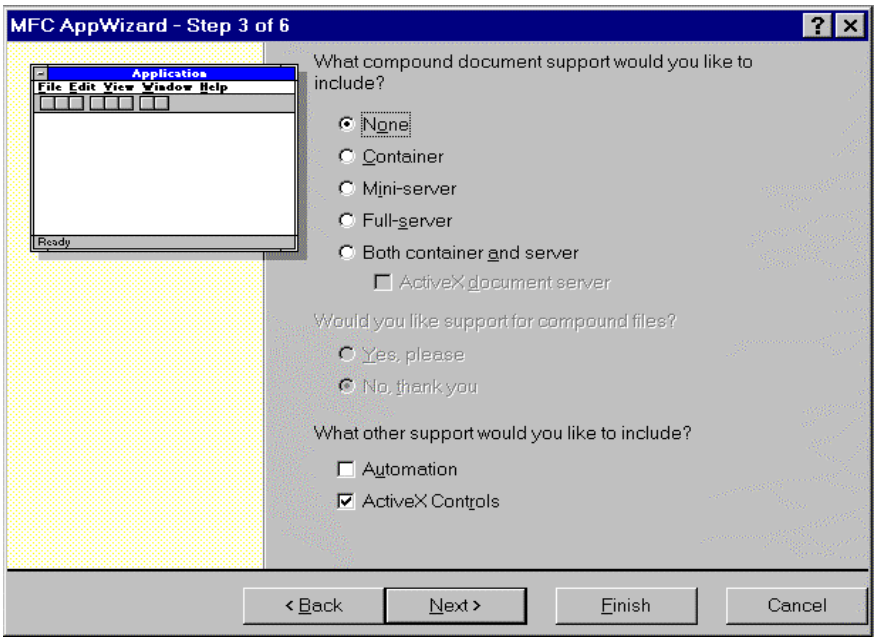


Figura 7 – Passo 3 do MFC AppWizard para SDI ou MDI.

NOTA: Para o exemplo Curso1, não selecione nenhum apoio a documento, simplesmente aceite o padrão.

Passo 4 (para programas SDI ou MDI)

1- Escolha qual interface básica dos usuários caracteriza o que você quer para seu programa e o que WOSA apoiam para incluir. Veja Windows Sockets em MFC e MAPI Support em MFC para mais informações.
2- Se você quer modificar os nomes de arquivos e extensões que seu programa usará ou quer ajustar a estrutura da janela de interface com usuário para seu programa, click Advanced.
3- Click Next.

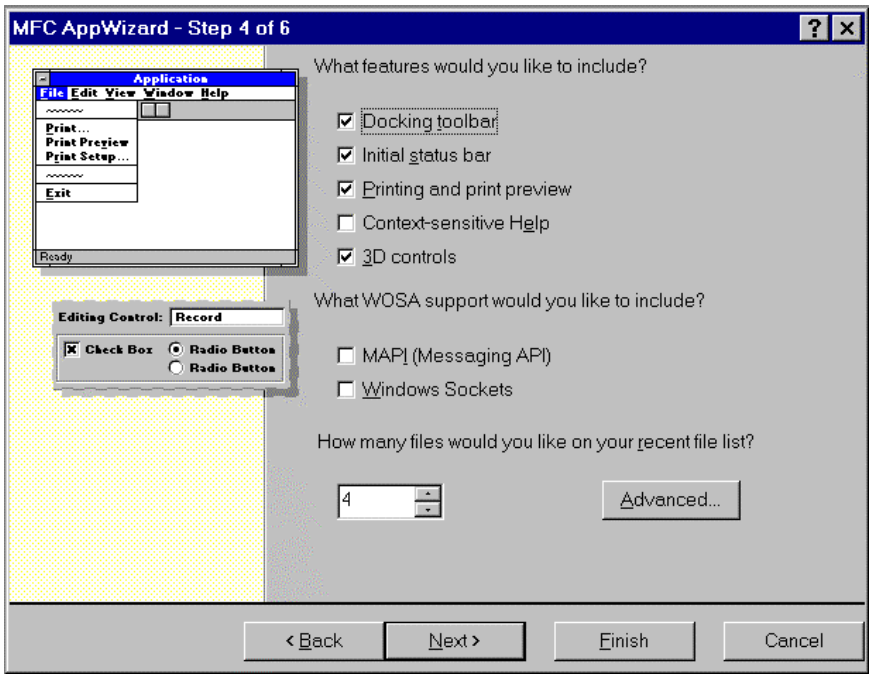


Figura 8 – Passo 4 do MFC AppWizard para SDI ou MDI.

NOTA: Para o exemplo Curso1, simplesmente aceite o padrão.

Passo 5 (para programas SDI ou MDI)

1- Escolha se você gostaria ou não que o AppWizard gere comentários no arquivo de fonte que o guiará para escrever seu programa.
2- Escolha se você gostaria de usar a biblioteca da MFC como uma DLL compartilhada ou linkada estaticamente.

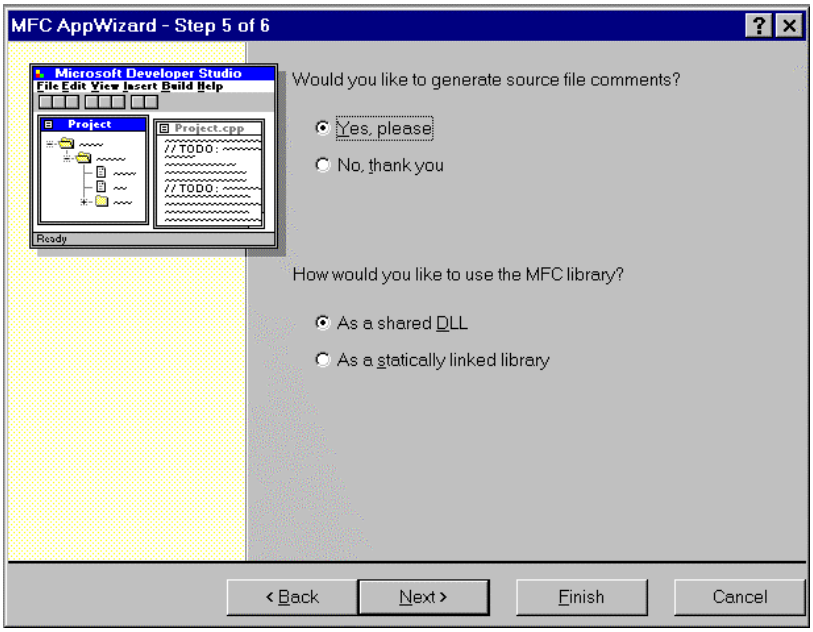


Figura 9 – Passo 5 do MFC AppWizard para SDI ou MDI.

Esta característica só é possível nas versões *Professional* e *Enterprise Edition*.

Passo 6 (para programas SDI ou MDI)

1- Se você quer mudar o nome de classe padrão, classe base, header ou nomes de arquivo de implementação providos pelo AppWizard, entre nos nomes novos. Para mudar a classe Base, selecione o classe View de seu programa.
2- Click Finish.

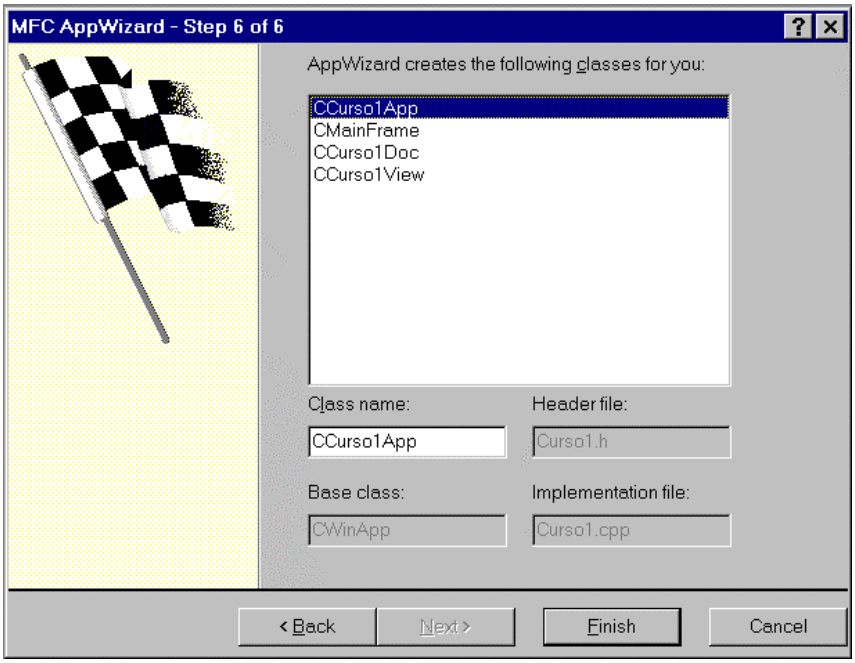


Figura 10 – Passo 6 do MFC AppWizard para SDI ou MDI.

Passo 7

O AppWizard exibe uma tela com as informações sobre o programa que está acabando de ser criado.

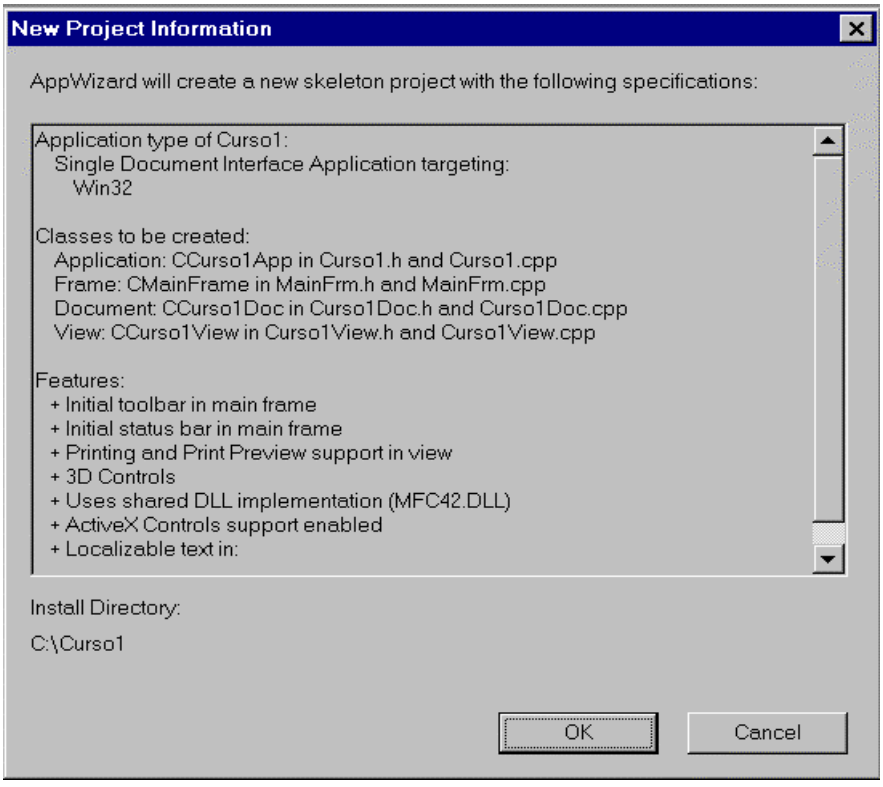


Figura 11 – Tela de informações sobre novo projeto do MFC AppWizard.

Quando o botão Ok do diálogo da figura acima for pressionado, o AppWizard começa a criar o código fonte de seu novo projeto. Quando o AppWizard finalizar, visualize o sub-diretório criado para o projeto. Os seguintes arquivos são de interesse por enquanto:

Arquivo	Descrição
Curso1.dsp	Arquivo de projeto que permite ao Developer Studio construir seu aplicativo
Curso1.dsw	Arquivo de WorkSpace que contém um entrada para o projeto.
Curso1.rc	Arquivo script ASCII de recursos
Curso1View.cpp	Arquivo de implementação da classe View, que contém as funções membro da classe *CCurso1View*.
Curso1View.h	Arquivo de definição da classe View, que contém a descrição da classe *CCurso1View*
Curso1.opt	Arquivo binário que informa ao Developer Studio que arquivos foram abertos para o projeto e como suas janelas estão arranjadas.
ReadMe.txt	Arquivo texto que explica o propósito dos arquivos gerados.
Resource.h	Arquivo de cabeçalho que contém as constantes #define dos identificadores usados pelos recursos.

A Figura 12 mostra a janela FileView do Workspace. Utilize esta janela para localizar os arquivos que compõe o projeto atual.

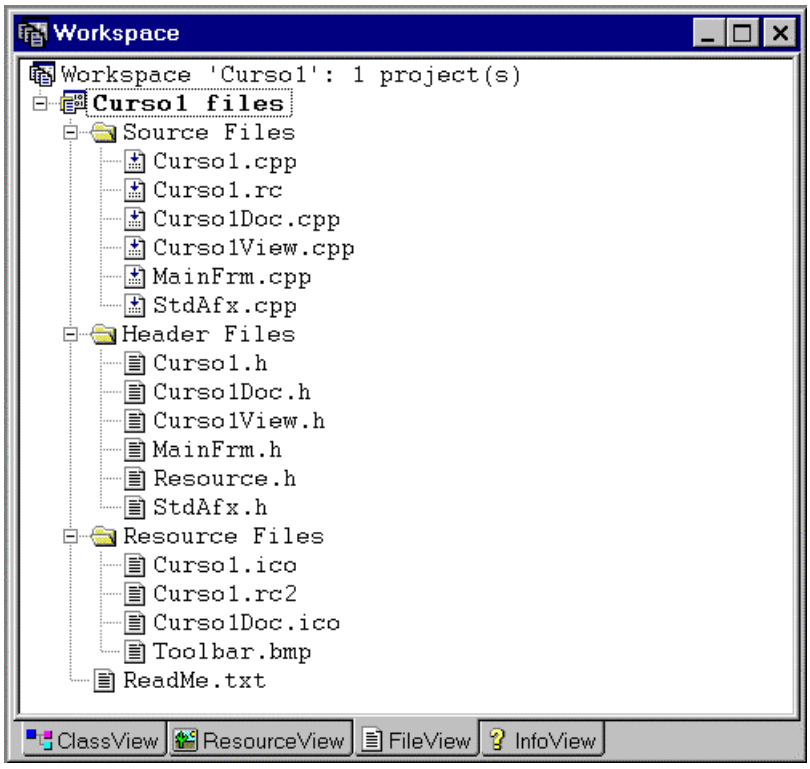


Figura 12 –Janela FileView do Workspace

3.2 Gerando o aplicativo do código gerado

O AppWizard gera não apenas os códigos fontes para um esqueleto de programa Windows, como também cria os arquivos de projeto e workspace para o aplicativo. Dentro do arquivo de projeto encontra-se todas as interdependência dos arquivos bem como as opções de compilação e link. Devido ao novo projeto tornar-se o atual para o Developer

Studio, é possível gerar o executável para o aplicativo selecionando a opção ***Build Curso1*** no menu Build ou pressionando o botão build da barra de ferramentas.

Se o processo ocorreu com sucesso, o programa executável chamado Curso1.exe é criado no novo subdiretório Debug dentro do diretório de seu projeto. Nesse subdiretório ficam também armazenados os arquivos temporários utilizados pelo compilador e linker.

NOTA: O processo de construção de um aplicativo (build),é de certa forma demorado. Este tempo de execução pode ser reduzido com a utilização de um computador com mais memória(mínimo de 32MB).

3.3 Testando o aplicativo gerado

No menu Build, selecione a opção Execute Curso1.exe. Experimente o programa criado, ele não é capaz de realizar muitas tarefas, (ainda não acrescentado código nenhum). Porém o aplicativo criado já possui muitas das características esperadas de um programa windows.

Termine a experiência fechando a janela do programa.

3.4 Copiando um código fonte para o disquete

Para copiar um código fonte gerado pelo AppWizard para um disquete:

1- Exclua o subdiretório Debug;
2- Copie todo o conteúdo do subdiretório criado para seu projeto para o disquete;
3- Não se esqueça de copiar também o diretório ***\res*** onde estão localizados os arquivos utilizados pelo editor de recursos.

Para o exemplo que acabou de ser criado, o conteúdo dos diretórios ***C:\Curso1*** e ***C:\Curso1\res*** devem ser copiados para que o projeto possa ser recompilado quando necessário.

4 Desenhando em uma janela, classe View

Após a criação de um aplicativo que é apenas o esqueleto de um programa Windows, será inserido um código dentro da classe View. Essas alterações podem ser feitas no código fonte do programa Curso1a.

4.1 A função membro OnDraw

OnDraw é uma função virtual membro da classe *CView* que pela application framework chama toda vez que uma janela precise ser redesenhada. Uma janela precisa ser redesenhada toda vez que um usuário a redimensiona, ou revela uma de suas partes que estava escondida atrás de outra, ou se o aplicativo alterar o conteúdo de exibição dessa janela. Se o usuário redimensiona a janela, a application framework faz a chamada automática. Se o aplicativo altera o conteúdo que deve ser exibido em uma janela, deve informar ao Windows que é necessário redesenhá-la. Para realizar esta tarefa é preciso proceder a chamada da função *Invalidate* (ou *InvalidateRect*). A chamada dessa função causará uma posterior chamada do membro *OnDraw*.

Recomenda-se que todo os desenhos que sejam necessários à janela sejam chamados do método OnDraw. Isto permite que tanto o sistema operacional quanto o seu programa possam redesenhar sua janela adequadamente através de mensagens.

4.2 O contexto de dispositivo para Janela

Como dito anteriormente, o Windows não permite que sejam feitos acessos diretos a placa de vídeo, mas comunicações podem são realizadas por uma camada de abstração chamada contexto de dispositivo associado a uma janela. Dentro da MFC, um contexto de dispositivo é um objeto de uma classe CDC que é passada como parâmetro para a função *OnDraw*. De posse de um ponteiro para um contexto de dispositivo, é possível utilizar as funções membros desse contexto para trabalhar com desenhos.

4.3 Adicionando código de desenho ao programa Curso1a

Agora vamos escrever o código para que o programa seja capaz de desenhar um texto e um circulo dentro de uma janela.

Certifique-se que o exemplo Curso1a está aberto no Developer Studio. É possível utilizar a janela de Workspace ClassView para localizar o código de uma função (duplo click em cima da OnDraw), ou também pode-se abrir o código fonte(source) do arquivo Curso1aView.cpp, na janela FileView.

A seguir estão os passos que devem ser seguidos para efetuar a operação de alteração do conteúdo de exibição de uma janela.

Passo 1: Editar a função OnDraw em Curso1aView.cpp.

Procure a função OnDraw no arquivo de implementação da classe view Curso1aView.cpp:

```
void CCurso1aView::OnDraw(CDC* pDC)
{
        CCurso1aDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: add draw code for native data here
}
```

Substitua o código encontrado pelo em código destaque a seguir:

```
void CCurso1aView::OnDraw(CDC* pDC)
{
        pDC->TextOut(0,0,"Hello, world!!");// imprime no canto superior esquerdo
                                          // com a fonte e tamanho padrão
        pDC->SelectStockObject(GRAY_BRUSH);// seleciona um preenchimento para o
                                                  // interior do circulo
        pDC->Ellipse(CRect(0,20, 100, 120));       // desenha um circulo cinza
                                                  // com diâmetro de 100 unidades
}
```

A chamada da função *GetDocument* pode ser seguramente retirada pois ainda não estamos trabalhando com documentos nesse programa.

As funções *TextOut*, *SelectStockObject*, *Ellipse* são membros da classe de contexto de dispositivo da Application Framework, CDC. A função *Ellipse* desenha um circulo se o retângulo que o limitar possuir dimensões iguais.

A MFC provê uma classe de útil para trabalhar com retângulos no Windows, *CRect*. Um objeto temporário de *CRect* é utilizado como parâmetro da função *Ellipse* para definir a região onde ela será desenhada. Mais adiante voltaremos a entrar em mais detalhes sobre a classe *CRect*.

Passo 2: Recompilar e Testar Curso1a

No menu *Build* selecione a opção *Build Curso1a.exe* , se nenhum erro ocorrer teste o aplicativo novamente. Agora você possui um programa que visivelmente faz alguma coisa.

5 Um preview do Editor de Recursos.

Agora que temos um aplicativo, é uma boa hora para darmos uma breve olhada no editor de Recursos. O arquivo de recursos para o exemplo é o Curso1a.rc. Este é um arquivo texto, porém modificá-lo com um edito de textos não é uma boa idéia, pois este é o trabalho do Editor de Recursos.

5.1 O conteúdo do arquivo Curso1a.rc

Muito que o aplicativo Curso1a "vê e sente" é determinado pelo arquivo de recursos. O arquivo Curso1a.rc contém os seguintes recursos do Windows listado na tabela abaixo:

Recurso	Descrição
Accelerator	Definição de teclas que simulam uma seleção do menu ou barra de ferramentas.
Dialog	Layout e conteúdo das caixas de diálogo. Curso1a contem apenas a caixa de diálogo About.
Icon	Figuras de 16x16 pixels ou 32x32 pixels, como os ícones que são vistos no Windows Explorer e no diálogo About do aplicativo. Curso1a usa o logotipo da MFC como seu icone.
Menu	O menu do aplicativo bem como seus pop-up associados.
String Table	Strings que não fazem parte do código fonte em C++ do programa.
Toolbar	Fila de botões imediatamente abaixo do menu
Version	Descrição do programa, numero de versão, língua, etc.

Além dos recursos citados na tabela acima, Curso1a.rc com os includes:

```
#include "afxres.h"
#include "afxres.rc"
```

que trazem alguns recursos da MFC que são comuns a todos os aplicativos. E

```
#include "resource.h"
```

que define três constantes para o aplicativo através das diretivas #define, que são: IDR_MAINFRAME, IDR_CURSO1TYPE e IDD_ABOUTBOX.

O editor de recursos deve ser utilizado para se inserir novas constantes nesse arquivo, uma vez que se editá-lo manualmente, na próxima utilização do Editor de Recursos essas alterações podem ser descartadas.

5.2 Executando o Editor de Diálogos

Alguns passos devem ser seguidos para a utilização desse editor.

Passo 1: Abrir o arquivo RC do aplicativo.

Pressione o botão ResourceView na janela Workspace. Se for expandido cada item, a janela de recursos será como mostrada abaixo na Figura 13:

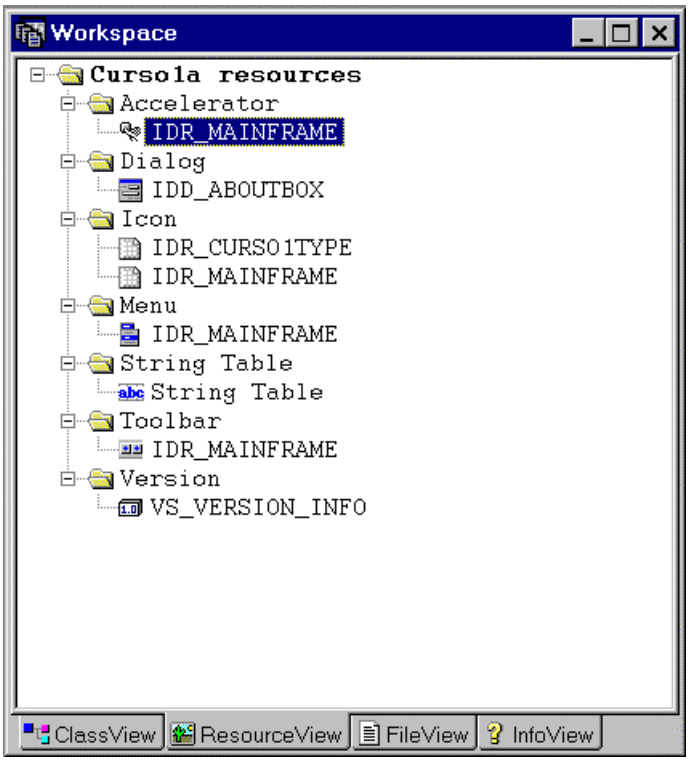


Figura 13 – Janela ResourceView do Workspace

Passo 2: Examine os recursos do aplicativo.

Explore cada tipo individual de recursos. Para selecionar um recursos para edição basta utilizar um double-click sobre ele. Uma janela abrirá com a ferramenta necessária para editar aquele tipo de recurso. Se uma caixa de diálogo for aberta para edição a barra de controles aparecerá na janela.

Passo 3: Modifique o diálogo IDD_ABOUTBOX

Faça algumas alterações na janela do diálogo About, como por exemplo:
Altere o tamanho de sua janela arrastando a borda direita ou inferior, movimente o botão Ok de posição, mude o texto, etc. Um click sobre o objeto o seleciona, pressionando o botão direito é possível alterar suas propriedades.

Passo 4: Reconstrua o aplicativo, com as modificações no arquivo de recurso.

No menu *Build* selecione a opção *Build Curso1a.exe. O Developer Studio* grava as alterações efetuadas no editor de recursos para o arquivo Curso1a.rc e chama o compilador de recursos para gerar a versão compilada do arquivo de recursos que chamará *Curso1a.res* que será linkada ao aplicativo executável.

Passo 5: Teste essa nova versão de aplicativo.

Execute o programa *Curso1a.exe* e escolha a opção About no menu Help, para confirmar que suas alterações foram feitas no diálogo About.

6 Versões Win32 Debug X Win32 Release

Abrindo-se o combo na Build toolbar, encontran-se duas opções. Win32 Debug e Win32 Release. Estes itens são alvos que representam opção de construção/compilação diferente. Quando um projeto é criado, o AppWizard gera dois alvos(targets) com diferentes opções de compilação/construção que estão resumidos na tabela abaixo:

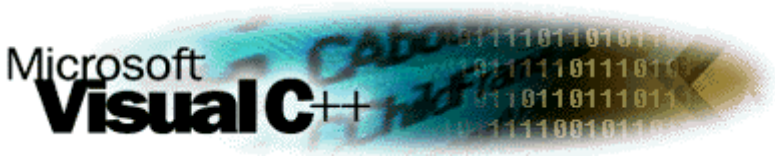
	Release Build	**Debug Build**
Source Code debugging	Desabilitado	Habilitado
MFC diagnostic macros	Desabilitado	Habilitado
Library linkage	MFC Release Library	MFC Debug Library
Compiler optimization	Máxima velocidade	Nenhuma otimização

Em resumo: você deve desenvolver seus programas na plataforma Debug e recompila-los na plataforma Release antes de distribuí-lo. O Arquivo executável gerado na plataforma Release é menor e mais rápido.

Os arquivos de saída e intermediários são armazenados respectivamente no diretório Debug e Release, que são criados como sub-diretórios do seu projeto.

7 Dois modos para executar um programa.

É possível executa um programa de forma direta (Ctrl+F5) ou através do debugger (F5). Executar o programa de forma direta é muito mais veloz porque o Developer Studio não precisa carregar o debugger primeiro. Se não for preciso visualizar as mensagens de diagnóstico ou usar breakpoints, execute o programa de forma direta que pode ser feita através do acelerador Ctrl+F5 ou ícone de exclamação na barra de ferramentas.



Capítulo 3
Manipulação básica de eventos, modos de Mapeamento e a ScrollView

Neste capítulo é abordado, como a MFC realiza a chamada da função membro virtual OnDraw. É mostrado mais uma forma de padronização de nomes de funções membros das classes bases da CView, onde as funções que respondem a eventos iniciam com as letras *On*, como por exemplo: OnKeyDown, OnLButtonDown, a própria OnDraw, entre outras.

Muitas dessas funções não são virtuais, o que faz com que mais passos de programação sejam realizados. O ClassWizard, será utilizado para facilitar a tarefa de associar um evento a funções membros das classes derivadas. Essa conexão é feita através do message map, que a estrutura realizada para ligar os eventos as funções membros na MFC.

Os dois primeiros exemplos usarão classes derivadas de CView para mostrar como eventos do usuário podem interagir com a função membro OnDraw, bem como o que afeta em uma janela o modo de mapeamento de desenho utilizado. Um outro exemplo será utilizado para demonstrar as características da classe CScrollView, que permite que barras de rolamento sejam conectadas a janela de vista do aplicativo.

1 Manipulando entradas do usuário através de funções mapeadas.

O exemplo Curso1a do capítulo anterior não aceitava nenhuma entrada feita pelo usuário, exceto o redimensionamento ou fechamento da janela que é padrão no Windows. O aplicativo contém menu e barra de ferramentas, porém estes não estão ligados ao código fonte da classe View. A ligação do menu e barra de ferramentas ao código fonte será discutido mais tarde devido a sua dependência com relação a janela frame que ainda não foi estudado. Porém inúmeros outras formas de entrada de dados os deixarão ocupados até lá.

1.1 O Mapa de mensagens.

Quando o usuário pressiona o botão esquerdo do mouse sobre uma janela View do aplicativo, o Windows envia uma mensagem (WM_LBUTTONDOWN) para esta janela. Se seu programa precisa executar uma ação em resposta a esta mensagem ele deve possuir uma função membro como esta:

```
CCurso3AView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // event processing code here
}
```

Dentro do arquivo de cabeçalho desta classe deve conter o correspondente protótipo para essa função:

```
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

A notação afx_msg, não gera código, porém o alerta que este protótipo é de uma função mapeada. Portanto, é necessário que haja uma entrada no mapa de mensagens (macro) para fazer a conexão da função OnLButtonDown com a estrutura das aplicações.

```
BEGIN_MESSAGE_MAP(CCurso3AView, CView)
        ON_WM_LBUTTONDOWN()     // entrada específica para OnLButtonDown
        // outras entradas ao mapa de mensagens
END_MESSAGE_MAP()
```

Finalmente o arquivo de cabeçalho da classe também deve conter a declaração:

```
        DECLARE_MESSAGE_MAP()
```

Todos estes passos podem ser feitos manualmente, mas o Visual C++ disponibiliza uma ferramenta, o ClassWizard, que automatiza a geração desses códigos para a maiioria das mensagens que podem ser mapeadas.

1.2 Guardando o estado da janela (Atributos)

Como seu programa aceitará entradas do usuário, o mesmo espera alguma resposta do sistema. A função membro OnDraw desenha a janela baseada em seu estado atual e o usuário pode alterar este estado. Em um aplicativo baseado na estrutura das aplicações da MFC, o documento deve guardar as informações de estado do aplicativo, mas este ainda não é o caso. Por enquanto isto será feito dentro da própria classe View, através da criação de duas variáveis membros m_rectElipse e m_nCor. Respectivamente uma é membro da classe CRect, para definir o retangulo que conterá a elipse, e a outra um inteiro para armazenar a cor de preenchimento da elipse.

NOTA: Por convenção, todas as variáveis membros não estáticas de classes derivadas da MFC carregam o prefixo ***m***_.

O exemplo dessa seção irá mapear as mensagens para alterar a cor da elipse (estado) toda vez que o mouse for pressionado sobre ela. A inicialização dessas variáveis membros deve ser feita no construtor da classe.

Dica: Por que não usar uma variável global para guardar o estado da janela ? Para evitar problemas quando da utilização de múltiplas janelas, além de variáveis globais furarem o conceito de encapsulamento da programação orientada a objeto.

Inicializando os membros de dados da classe View –

O lugar mais eficiente para se fazer uma inicialização de membros de dados é no construtor. Como mostrado no código abaixo.

```
CCurso3AView::CCurso3AView() : m_rectElipse(0,0,200,200)
{
        m_nCor = GRAY_BRUSH;
}
```

Note os meios como podem ser realizada a inicialização em um construtor.

1.3 Teoria do retângulo inválido.

A função membro *OnLButtonDown* pode alterar a variável *m_nCor*, a qualquer tempo, porém só terá efeito na tela após o método *OnDraw* ser chamado (por exemplo quando a janela é redimensionada). Dentro da função *OnLButtonDown* pode ser chamado um membro da classe CView herdado de CWnd, a função *InvalidateRect*, que faz com o o Windows gere uma mensagem WM_PAINT que é mapeada dentro de CView e faz uma chamada ao método OnDraw. O método *OnDraw* pode acessar o “retângulo inválido” que é passado como parâmetro por *InvalidateRect*, para decidir qual região da janela precisa ser redesenhada.

Lembrar que a função OnDraw não é chamada apenas pelo método InvalidateRect, e é utilizada para redesenhar qualquer parte da janela.

1.4 A área cliente de uma janela do Windows

Define-se área cliente de uma janela, como a região retangular que exclui, as bordas, barra de título, menu, e qualquer barra de ferramentas que a ela esteja ligada. A classe *CWnd* possui uma função membro *GetClientRect*, que retorna as dimensões da área cliente. Normalmente não é permitido desenhar fora da área cliente de uma janela, e muitas mensagens do mouse só são recebidas pela janela quando o ponteiro do mouse está sobre sua área cliente.

1.5 Aritmética das classes de auxílio ao posicionamento

As classes *CRect*, *CPoint*, *CSize* são derivadas diretamente das estruturas RECT, POINT e SIZE da API do Windows, e possuem os membros de dados inteiros públicos como a seguir:

CRect	*left, top, right, bottom*
CPoint	*x,y*
CSize	*cx, cy*

Visualizando o guia de referência da MFC (help), fica visível que estas classes possuem vários operadores sobrecarregados, que permitem entre outras coisas:

- Somar ou subtrair objetos CSize a objetos CPoint;
- Subtrair um objeto CPoint de outro e armazenar em um objeto CSize;
- Somar ou subtrair objetos CPoint e CSize de objetos CRect.

A classe CRect possui ainda outros membros que retornam objetos CPoint e CSize. Neste ponto já podemos perceber que objetos do tipo CSize resultam de uma diferença entre dois objetos CPoint, e que é possível deslocar o canto superior esquerdo de um objeto CRect para coincidir com um objeto CPoint.

É possível ainda determinar se um ponto está dentro de um retângulo utilizando a função membro PtInRect. Por exemplo para trocar o estado do membro de dados m_nCor pode ser feito um teste para determinar se o mouse está sendo pressionado dentro do retângulo que define a elipse. O código que ficaria dentro da função OnLButtonDown seria assim:

```
if (m_rectElipse.PtInRect(point))
{// se for verdadeiro o ponto pertence ao retângulo.
        if (m_nCor == GRAY_BRUSH)
                m_nCor = WHITE_BRUSH;
        else
                m_nCor = GRAY_BRUSH;
        InvalidateRect(m_rectElipse)
}
```

Para determinar se o ponto pertence exatamente a uma região elíptica, é necessário construir um objeto do tipo CRgn que corresponda a uma elipse utilizar um de seus membros PtInRegion. Este seria o código:

```
CRgn rgn;
rgn.CreateEllipticRgnIndirect(m_rectElipse);
if (rgn.PtInRegion(point))
{// se for verdadeiro o ponto pertence ao retângulo.
        if (m_nCor == GRAY_BRUSH)
                m_nCor = WHITE_BRUSH;
        else
                m_nCor = GRAY_BRUSH;
```

```
            InvalidateRect(m_rectElipse)
    }
```

A classe CRgn pode tamber ser utilizada para definir uma região poligonal.

1.6 Usando o ClassWizard Exemplo Curso3A.

Este exemplo destina-se a desenhar uma elipse (com cara de círculo) na janela cliente e aceitar o evento de pressionar o botão esquerdo do mouse dentro do retângulo que define a elipse e exibir como resultado a troca de estado (cor) do preenchimento da mesma.

No exemplo Curso2A, desenhar em uma janela dependia apenas da função membro OnDraw. O exemplo Curso3A requer que três funções sejam alteradas e dois membros de dados criados, para que o programa realize mais algumas tarefas.

Para criar este exemplo utilizaremos o AppWizard, como anteriormente, só que em conjunto com o ClassWizard. Esta combinação AppWizard-ClassWizard é diferente de um gerador de código convencional, pois este criaria todo o código de uma só vez. O código gerado pelo AppWizard, que é executado apenas uma vez, pode ser acessado quantas vezes for necessário utilizando o ClassWizard, mesmo depois que ele já tenha sido editado manualmente. Isto é possível pois o AppWizard gera alguns comentários dentro do código fonte que serão utilizado pelo ClassWizard para identificar os locais apropriados da inserção automática de código. Um exemplo desses comentários é mostrado nas listagens abaixo:

No arquivo de inclusão é gerado

```
//{{AFX_MSG(CCurso3AView)
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
```

No arquivo de implementação é gerado

```
//{{AFX_MSG_MAP(CCurso3AView)
ON_WM_LBUTTONDOWN()     // entrada específica para OnLButtonDown
//}}AFX_MSG_MAP
```

Utilizando o AppWizard e ClassWizard em cojunto

Os seguintes passos mostram como utilizar o AppWizard em conjunto com o ClassWizard para gerar o exemplo Curso3A:

1.6.1 Passo1: Execute AppWizard para criar Curso3A.

Utilize o AppWizard para gerar um projeto SDI chamado Curso3A no subdiretório c:\Curso1A. Proceder da mesma forma que para o exemplo Curso1a.

1.6.2 Passo2: Adicionar os membros de dados m_nCor e m_rectElipse ao Curso3A

Com a janela Workspace na pasta ClassView, pressione o botão direito do mouse sobre a classe Curso3AView, e selecione Add Member Variable, e insira os dois membros de dados.

```
protected:
        int     m_nCor;         // guarda cor do preenchimento
        CRect   m_rectElipse;   // guarda posição e tamanho da elipse
```

Se preferir pode digitar o código acima dentro da declaração da classe que está no arquivo Curso3AView.h.

1.6.3 Passo3: Use o ClassWizard para adicionar um mapeamento de mensagens a classe CCurso3AView.

Selecione a opção ClassWizard do menu View do Developer Studio, ou pressione o botão direito dentro da janela com o código fonte. Quando a caixa de diálogo do ClassWizard aparecer, certifique-se que a classe *CCurso3AView* está selecionada como

mostra a figura a seguir. Agora pressione o botão do mouse sobre *CCurso3AView* no topo do list box *Object ID's* e role o list box *Messages* para selecionar a mensagem WM_LBUTTONBOWN, e pressione o botão *Add Function*. A função *OnLButtonDown* aparecerá no list box *Member Functions*.

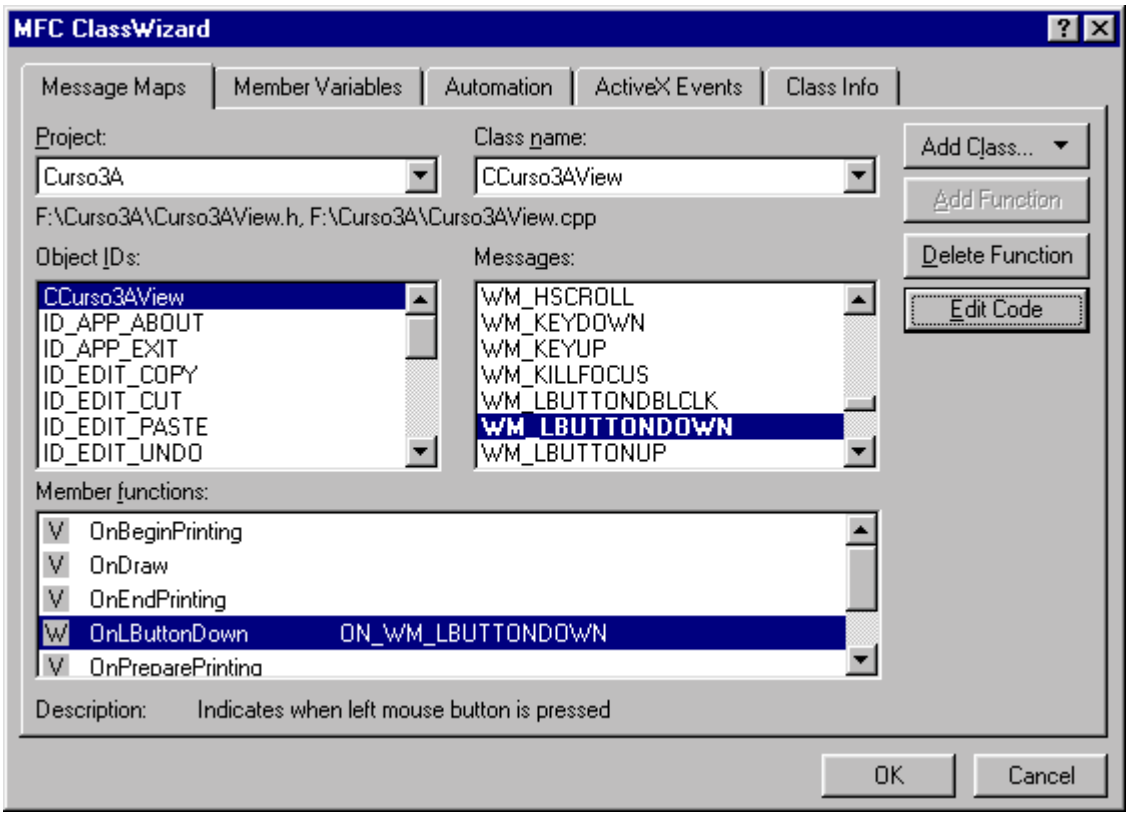


Figura 14 – Caixa de Diálogo do ClassWizard.

1.6.4 Passo4: Edite o código de OnLButtonDown em Curso3AView.cpp

Pressione o botão *Edit Code*. O ClassWizard abre uma janela de edição posicionada sobre a nova função OnLButtonDown que acaba de ser gerada. O código em destaque abaixo deve ser digitado dentro da função substituindo o antigo.

```
void CCurso3AView::OnLButtonDown(UINT nFlags, CPoint point)
{
	if (m_rectElipse.PtInRect(point))
	{// se for verdadeiro o ponto pertence ao retângulo.
		if (m_nCor == GRAY_BRUSH)
			m_nCor = WHITE_BRUSH;
		else
			m_nCor = GRAY_BRUSH;
		InvalidateRect(m_rectElipse);
	}
}
```

1.6.5 Passo5: Edite o código do construtor e da função OnDraw em Curso3AView.cpp.

O código em destauqe a seguir deve substituir o conteúdo previamente existente:

```
CCurso3AView::CCurso3AView() : m_rectElipse(0,0,200,200)
{
	m_nCor = GRAY_BRUSH;

}
```

```
void CCurso3AView::OnDraw(CDC* pDC)
{
        pDC->SelectStockObject(m_nCor); // seleciona objeto do estoque
        pDC->Ellipse(m_rectElipse);     // desenha uma elipse preenchida
}
```

1.6.6 Passo6: Compile e execute o programa Curso3A

Selecione Build do menu Build do Developer Studio.
Execute o programa Curso3A através da opção Execute Curso3A.exe no menu Build. Pressione o botão esquerdo do mouse dentro do circulo que aparecerá na janela para verificar o resultado de seu programa. (Não pressione repetidamente o botão do mouse pois o Windows pode interpretar como uma mensagem de duplo click)

2 Modos de Mapeamento

Os desenhos realizados até agora foram baseados em unidades de pixels, também conhecida como *Coordenadas do Dispositivo*. No exemplo Curso3A, as unidades utilizadas eram pixels devido ao Contexto de Dispositivo estar ajustado para o Modo de Mapeamento Padrão, MM_TEXT. A declaração

```
pDC->Rectangle(CRect(0,0,200,200));
```

desenha um quadrado de 200 por 200 pixels, coincidente com o canto superior esquerdo da área cliente. Notar que se a resolução do monitor for alterada, para por exemplo 1024x768, este mesmo quadrado reduzirá de tamanho. Se for utilizado um contexto de Impressão (600dpi) este quadrado aparecerá com um tamanho bem pequeno.

E se meu programa tivesse que desenhar um quadrado de 4x4 cm, independente do dispositivo que esle estivesse sendo mostrado? O Windows possui outros modos de mapeamento, ou sistema de coordenadas, que podem ser associados a um contexto de dispositivo. As coordenadas no modo atual de mapeamento são chamadas *Coordenadas Lógicas*.

Pode parecer fácil manipular estes sistemas de coordenadas, porém não é possível trabalhar apenas em coordenadas lógicas. O programa terá que chavear entre coordenadas lógicas e coordenadas do dispositivo, e você precisará conhecer quando convertê-las.

2.1 O modo de mapeamento MM_TEXT

À primeira vista, MM_TEXT pode não parecer um modo de mapeamento mas um outro nome para coordenadas do dispositivo. Neste modo de mapeamento, valores de *x* crescem para a direita e valores de *y* crescem para baixo, porém a origem desse sistema de coordenadas pode ser mudada utilizando as funções membros da classe CDC, *SetViewportOrg* e *SetWindowOrg*. Uma view deslocável (Scroll) utiliza este tipo de transformação.

O seguinte código pode ser utilizado para deslocar a origem do sistema de coordenadas.

```
void CMyView::OnDraw(CDC* pDC)
{
        pDC->SetMapMode(MM_TEXT);
        pDC->SetWindowOrg( CPoint(100,100) );
        pDC->Rectangle(CRect(100,100,300,300));
}
```

2.2 Modos de mapeamento de escala fixa.

Um grupo importante de modos de mapeamento providos no Windows é o mapeamento de escala fixa. Nestes modos de mapeamento a abcissa *x* cresce da esquerda para direita e a ordenada *y* decresce de cima para baixo, e não é possível mudar isto.

A tabela a seguir mostra os modos de mapeamento de escala fixa, sendo que a única diferença entre eles é o fator de escala.

Modo de mapeamento	Unidade lógica
MM_LOENGLISH	0,01 in
MM_HIENGLISH	0,001 in
MM_LOMETRIC	0,1 mm
MM_HIMETRIC	0,01 mm
MM_TWIPS	1/1440 in

Obs:

O modo MM_TWIPS será utilizado preferencialmente em contexto de impressão. A unidade twip se refere a 1/20 pontos. Um ponto é uma unidade de medida que no windows corresponde exatamente a 1/72 in. Para desenhar um caractere de tamanho 12 pontos (fonte tamanho 12), ajuste a altura do caractere para 12*20, ou 240 twips.

2.3 Modos de mapeamento de escala variável.

Dois modos de mapeamento com escala variável são suportados pelo Windows, MM_ISOTROPIC e MM_ANISOTROPIC, onde tanto o fator de escala quando a origem do sistema de coordenadas podem ser alteradas. Com estes modos de mapeamento o desenho em uma janela pode variar de tamanho quando a janela é redimensionada.

Com o modo MM_ISOTROPIC, a razão 1:1 é sempre mantida, em outras palavras um circulo será sempre um círculo mesmo que o fator de escala mude. Com o modo MM_ANISOTROPIC, as escalas em x e y podem variar independentemente, o que significa que círculos podem ser espremidos a elipses.

O código abaixo desenha um elipse que ocupa exatamente o tamanho da área cliente:

```
void CMyView::OnDraw(CDC* pDC)
{
        CRect rectCliente;
        GetClientRect(rectCliente);
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt( 1000,1000 );
        pDC->SetViewPortExt(rectCliente.right, rectCliente.bottom);
        pDC->SetViewPortOrg(rectCliente.right/2, rectCliente.bottom/2);
        pDC->Ellipse(CRect(-500,-500,500,500));
}
```

No código acima, as funções *SetWindowExt* e *SetViewportExt*, trabalham juntas para definir a escala baseada no tamanho da área cliente retornada pela função *GetClientRect*. O resultado é que sempre o tamanho da janela conterá 1000x1000 unidades lógicas. A função *SetViewportOrg* posiciona a origem do sistema de coordenadas para o dentro da janela. Finalmente uma elipse de raio 500 unidades lógicas preenche exatamente a janela.

Aqui estão as fórmulas para converter as unidades lógicas para unidades de dispositivo:

Fator de escala x = extensão x do viewport / extensão x da janela
Fator de escala y = extensão y do viewport / extensão y da janela
x do dispositivo = x lógico * Fator de escala x + deslocamento da origem em x
y do dispositivo = y lógico * Fator de escala y + deslocamento da origem em y

Substituindo-se MM_ANISOTROPIC e MM_ISOTROPIC, a elipse sempre será vista como um círculo, que se expande e ajusta as menores dimensões do retângulo da área cliente.

2.4 Conversão de Coordenadas

Uma vez escolhido o modo de mapeamento do contexto de dispositivo, as coordenadas lógicas podem ser utilizadas como parâmetros para a maioria dos membros da classe CDC. Porém as coordenadas do ponteiro do mouse que são passadas como parâmetro pelo Windows para as mensagens do mouse (parâmetro *point* de *OnLButtonDown*) estão representadas em coordenadas do dispositivo. Muitas outras funções da MFC, particularmente da classe CRect, trabalham corretamente apenas com coordenadas do dispositivo.

> NOTA: As funções aritmética da classe CRect, assumem que o membro *right* é maior que o *left* e que o membro *bottom* é maior que o *top*. Quando isto não ocorrer, funções como por exemplo a *CRect::PtInRect* não funcionarão corretamente a menos que o membro *CRect::NormalizeRect* seja previamente chamado para regularizar a situação.

A transformação de coordenadas lógicas para coordenadas de dispositivo são feitas diretamente pela GDI. As funções membros da classe CDC, *LPtoDP* e *DPtoLP* realizam a conversão entre os dois sistemas, assumindo o modo de mapeamento atual e os parâmetros associados que foram previamente selecionados. Seu trabalho fica limitado a decidir quando utilizar cada um desses sistemas.

Algumas regras básicas estão listadas abaixo.

- Assumir que as Funções membros da classe CDC levam coordenada lógicas como parâmetros.
- Assumir que as Funções membros da classe *CWnd* levam coordenada de dispositivo como parâmetros.
- Faça qualquer hit-test em coordenadas de dispositivo. (funções como *CRect::PtInRect* trabalham melhor em coordenadas de dispositivo)
- Armazene valores em coordenadas lógicas ou físicas. Se você armazenar um ponto em coordenadas do dispositivo e o usuário rolar a janela, este ponto não será mais válido.

O código a seguir serve para testar se botão esquerdo do mouse foi pressionado dentro de um retângulo:

```
//assumir que m_rect é um membro do tipo CRect de uma classe derivada de
//CView utilizando-se do modo de mapeamento de coordenadas lógicas
//MM_LOENGLISH .

void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CRect rect = m_rect;    // faz uma cópia temporária do objeto m_rect
        CClientDC dc(this);     // isto é como se deve pegar um contexto de
                                // dispositivo para o uso das funções
                                // SetMapMode e LPtoDP.
        dc.SetMapMode(MM_LOENGLISH);
        dc.LPtoDP(rect);        // rect ficou em coordenadas do dispositivo
        if (rect. PtInRect(point))
                TRACE("cursor do mouse está dentro do retângulo");
}
```

O melhor lugar para selecionar o modo de mapeamento é dentro da função virtual *OnPrepareDC* membro da classe *CView*, ao invés de fazer isso na OnDraw.

2.5 Exemplo Curso3B

O exemplo Curso3B é o exemplo Curso3A convertido para o modo de mapeamento MM_HIMETRIC.

2.5.1 Passo 1: utilizar o ClassWizard para sobrecarregar a função virtual *OnPrepareDC*

As funções virtuais das classes bases da MFC, podem ser sobrecarregadas utilizando-se o ClassWizard. Ele gera o protótipo correto para a função dentro da descrição da classe e o esqueleto da função no arquivo de implementação(CPP).

Selecione o nome da classe CCurso3AView na lista Object ID, em seguida encontre a função OnPrepareDC na lista Messages. Edite o código da função para que fique como o abaixo:

```
void CCurso3AView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
        pDC->SetMapMode(MM_HIMETRIC);
        CView::OnPrepareDC(pDC,pInfo);
}
```

A estrutura das aplicações chama a função virtual OnPrepareDC antes de chamar a OnDraw.

2.5.2 Passo 2: Edite o construtor da classe view

Altere o valor das coordenadas do retângulo que define a elipse. O retângulo agora será mostrado com 4x4 cm ao invés de 200x200 pixels. Notar que o valor da coordenada y é negativo,

```
CCurso3AView::CCurso3AView() : m_rectElipse(0,0,4000,-4000)
{
        m_nCor = GRAY_BRUSH;
}
```

2.5.3 Passo 3: Edite a função OnLButtonDown

Esta função deve converter o retângulo que define a elipse para coordenadas do dispositivo para testar a posição do ponto em relação a elipse (hit-test).

Altere o código da função para que ela fique como abaixo:

```
void CCurso3AView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CClientDC dc(this);
        OnPrepareDC(&dc);
        CRect rectDevice = m_rectElipse;
        dc.LPtoDP(rectDevice);
        if (rectDevice.PtInRect(point))
        {       // se for verdadeiro o ponto pertence ao retângulo.
                if (m_nCor == GRAY_BRUSH)
                        m_nCor = WHITE_BRUSH;
                else
                        m_nCor = GRAY_BRUSH;

                InvalidateRect(rectDevice)
        }
}
```

2.5.4 Passo 4: Compile e execute o programa Curso3B

A apresentação deste exemplo é bem similar ao anterior, exceto pelo fato da elipse possuir sempre o mesmo tamanho, independente da resolução do vídeo. Experimente também a opção Print Preview, desta vez a elipse se apresentará bem maior que no outro exemplo.

3 Uma view deslocável (CScrollView)

A falta de barras de rolagem nos exemplos Curso3A e Curso3B, indica que a classe base CView da MFC não suporta diretamente as barras de rolagem. Outra classe da MFC CScrollView suporta essas barras. A classe CScrollView é derivada da classe CView.

O exemplo Curso3C mostra o uso dessa classe como base para a view.

A classe CScrollView suporta as barras de rolagem, porém não suporta o rolamento pelo teclado. Este suporte será incluído também nesse exemplo.

3.1 Uma janela maior do pode ser exibida.

Se o mouse for utilizado para encolher o tamanho de uma janela, o conteúdo dessa janela permanece fixo no canto superior esquerdo da mesma, porém o conteúdo do canto inferior direito da janela pode desaparecer. Quando a janela é reespandida, esses itens reaparecem. Com isso podemos concluir que a janela é maior do que o viewport que pode ser visto na tela. Este viewport não precisa ser necessariamente fixo ao canto superior esquerdo da janela. O uso das funções membros da classe CWnd ScrollWindow e SetWindowOrg pela CScrollView permite que o viewport seja movido para qualquer ponto da janela.

3.2 Barras de rolamento

O Windows permite que seja exibida a barra de rolagem nas bordas das janelas, porém sozinho ele não conecta estas barras a suas janelas. Isto é feito pela classe CScrollView, que processa as mensagens WM_HSCROLL e WM_VSCROLL enviadas pelas barras as views. Estas funções movimentam o viewport dentro da janela.

A classe CScrollView se aplica quando for utilizada um janela grande sendo vista por um viewport pequeno. Cada item presente dentro desta janela possui um posição única.

> **OBS**: o Windows NT utiliza variáveis de 32 bits para manipular coordenadas lógicas, porém o Windows 95 utiliza variáveis de 16 bits para coordenadas lógicas, o que limita esses valores a uma faixa de –32768 a 32767. Pense nisso quando for desenvolver seus programas.

3.3 A função membro OnInitialUpdate.

A função virtual OnInitialUpdate é a primeira função a ser chamada pela estrutura das aplicação após a janela ser criada. A estrutura das aplicações chama a função OnInitialUpdate antes de chamar a função OnDraw pela primeira vez, portanto é o local apropriado para inicializar o modo de mapeamento e o tamanho lógico de uma janela do tipo scrollview. Este parâmetros são ajustados chamando-se a função membro *CScrollView::SetScrollSizes*.

3.4 Trabalhando com entradas do teclado.

Como visto durante a utilização do Spy++, as entradas do teclado são processadas em duas etapas. O Windows envia as mensagens WM_KEYDOWN e WM_KEYUP com os códigos virtuais das teclas. Estas mensagens são processadas pelo próprio Windows para identificar se alguns caracter padrão ANSI foi pressionado e também o status da tecla shift para gerar a mensagem WM_CHAR com o código próprio para letras maiúsculas e minúsculas.

O ClassWizard pode ser usado para mapear essas mensagens dentro de uma view. Se um caracter ANSI for pressionado utilize a mensagem WM_CHAR para ligá-lo a sua view, porém se outras teclas forem esperadas, a mensagem WM_KEYDOWN deve ser mapeada dentro de uma view.

3.5 Exemplo Curso3C

O exemplo Curso3C cria uma janela lógica de 20 cm de largura por 30 cm de comprimento. O programa desenha ainda a mesma elipse do exemplo Curso3B que poderia ser utilizado como base para este novo programa, porém é mais fácil criar um novo programa utilizando o AppWizard que gerará automaticamente a função OnInitialUpdate para você.

Para criar esse programa proceda os seguintes passos.

3.5.1 Passo1 : Execute o AppWizard para criar o exemplo Curso3C.

Utilize o AppWizard para gerar um projeto SDI chamado Curso3C no subdiretório c:\Curso1C.

No sexto passo do assistente, selecione a classe CCurso3CView e altere sua classe base na caixa de seleção como mostrado na figura abaixo.

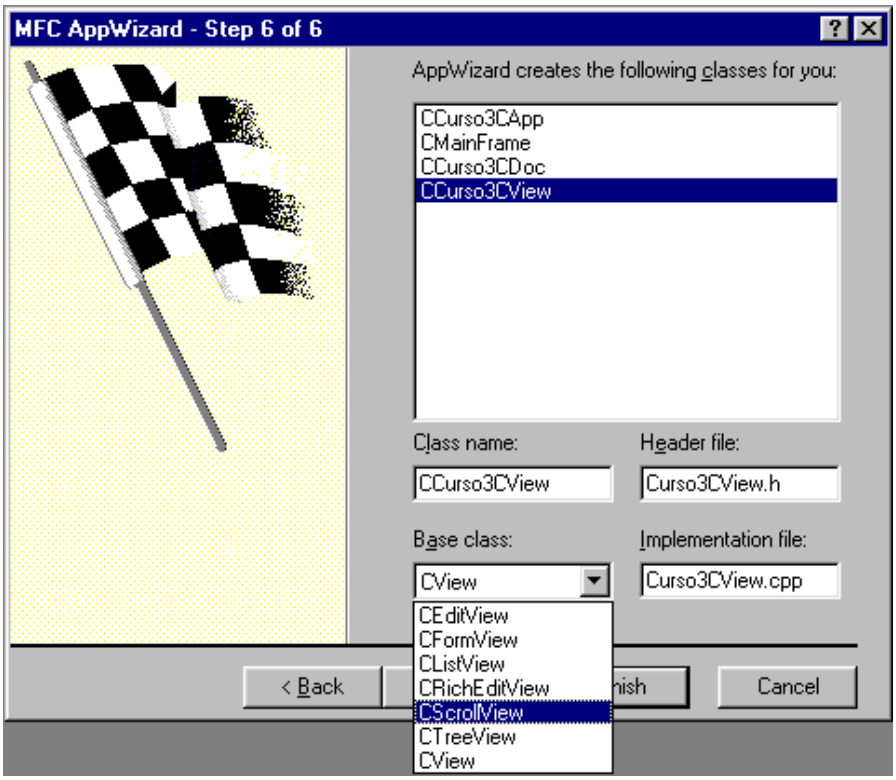


Figura 15 – Passo 6 da criação do exemplo Curso3C

3.5.2 Passo2: Adicionar os membros de dados m_nCor e m_rectElipse ao Curso3C.

Com a janela Workspace na pasta ClassView, pressione o botão direito do mouse sobre a classe Curso3CView, e selecione *Add Member Variable*, e insira os dois membros de dados.

```
protected:
        int     m_nCor;         // guarda cor do preenchimento
        CRect   m_rectElipse;   // guarda posição e tamanho da elipse
```

Se preferir pode digitar o código acima dentro da declaração da classe que está no arquivo Curso3CView.h.

3.5.3 Passo3 : Modifique a função OnInitialUpdate criada pelo AppWizard.

Edite a função OnInitialUpdate do arquivo Curso3CView.cpp como mostrado abaixo:

```
void CCurso3CView::OnInitialUpdate()
{
        CScrollView::OnInitialUpdate();
        CSize sizeTotal(20000, 30000);  // 20 por 30 cm
        CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);
        CSize sizeLine(sizeTotal.cx / 50, sizeTotal.cy / 50);
        SetScrollSizes(MM_HIMETRIC, sizeTotal, sizePage, sizeLine);
}
```

3.5.4 Passo 4 : Utilize o ClassWizard para mapear a mensagem WM_KEYDOWN.

O ClassWizard gera a função membro OnKeyDown, com seu protótipo, entrada no mapa de mensagens e seu esqueleto. Edite seu código como a seguir:

```
void CCurso3CView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        switch (nChar) {
        case VK_HOME:
                OnVScroll(SB_TOP, 0, NULL);
                OnHScroll(SB_LEFT, 0, NULL);
                break;
        case VK_END:
                OnVScroll(SB_BOTTOM, 0, NULL);
                OnHScroll(SB_RIGHT, 0, NULL);
                break;
        case VK_UP:
                OnVScroll(SB_LINEUP, 0, NULL);
                break;
        case VK_DOWN:
                OnVScroll(SB_LINEDOWN, 0, NULL);
                break;
        case VK_PRIOR:
                OnVScroll(SB_PAGEUP, 0, NULL);
                break;
        case VK_NEXT:
                OnVScroll(SB_PAGEDOWN, 0, NULL);
                break;
        case VK_LEFT:
                OnHScroll(SB_LINELEFT, 0, NULL);
                break;
        case VK_RIGHT:
```

```
                OnHScroll(SB_LINERIGHT, 0, NULL);
                break;
        default:
                break;
        }
}
```

3.5.5 Passo5: Edite o código do construtor e da função OnDraw em Curso3CView.cpp.

O código em destaque a seguir deve substituir o conteúdo previamente existente:

```
CCurso3CView::CCurso3CView() : m_rectElipse(0,0,4000,-4000)
{
        m_nCor = GRAY_BRUSH;

}
void CCurso3CView::OnDraw(CDC* pDC)
{
        pDC->SelectStockObject(m_nCor); // seleciona objeto do estoque
        pDC->Ellipse(m_rectElipse);     // desenha uma elipse preenchida
}
```

3.5.6 Passo 6 : Utilize o ClassWizard para mapear a mensagem WM_LBUTTONDOWN.

As seguintes alterações devem ser feitas na função gerada pelo ClassWizard:

```
void CCurso3CView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CClientDC dc(this);
        OnPrepareDC(&dc);
        CRect rectDevice = m_rectElipse;
        dc.LPtoDP(rectDevice);
        if (rectDevice.PtInRect(point))
        {       // se for verdadeiro o ponto pertence ao retângulo.
                if (m_nCor == GRAY_BRUSH)
                        m_nCor = WHITE_BRUSH;
                else
                        m_nCor = GRAY_BRUSH;
                InvalidateRect(rectDevice);
        }
}
```

3.5.7 Passo 7: Compile e execute o programa Curso3C

Selecione Build do menu Build do Developer Studio.

Execute o programa Curso3C através da opção Execute Curso3C.exe no menu Build. Pressione o botão esquerdo do mouse dentro do circulo que aparecerá na janela para verificar o resultado de seu programa. Reposicione a janela utilizando as barra de rolamento, e verifique se o hit-test ainda continua funcionando. Verifique se o teclado também pode ser usado para rolar a janela.

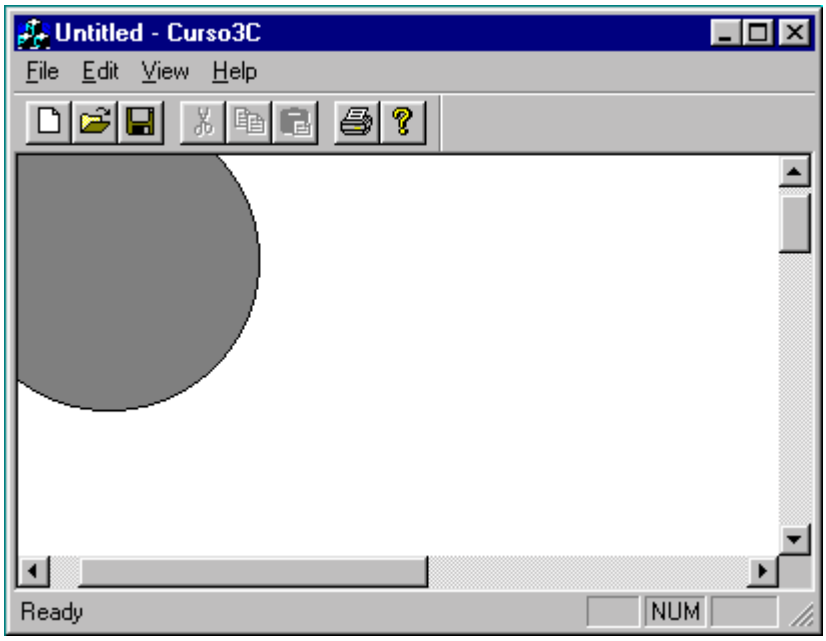


Figura 16 – Exemplo Curso3C

4 Outras mensagens do Windows

A MFC pode manipular até 140 tipos de mensagens geradas pelo Windows. Essas mensagens serão exploradas nos próximos capítulos. Porém existem cinco mensagens que são de especial interesse no momento.

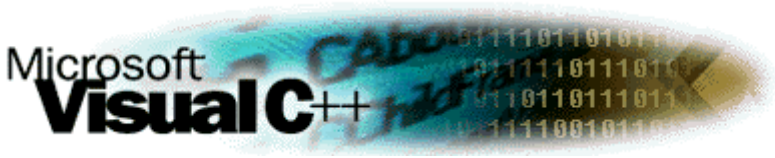
A mensagem WM_CREATE
Está é a primeira mensagem que o Windows envia para uma view.

A mensagem WM_CLOSE
O Windows envia esta mensagem quando o usuário pressiona o botão fechar a partir do menu do sistema.

A mensagem WM_QUERYENDSESSION
O Windows envia esta mensagem para todos os aplicativos que estão ativos quando o usuário solicita a saída do Windows.

A mensagem WM_DESTROY
O Windows envia esta mensagem após a mensagem WM_CLOSE. Quando o programa recebe esta mensagem ele assume que a janela não está mais visível.

A mensagem WM_NCDESTROY
Esta é a ultima mensagem que o Windows envia quando uma janela está sendo destruída.



Capítulo 4
Interface de dispositivos gráficos, cores e fontes.

Alguns elementos da GDI já foram vistos em capítulos anteriores. Toda vez que um programa desenha na tela ou na impressora está utilizando funções da GDI. A GDI possui funções para desenhar pontos, linhas, retângulos, polígonos, elipses, bitmaps e textos. O desenho de círculos, quadrados pode ser feito intuitivamente apenas com um breve estudo das funções disponíveis, porém a programação para o desenho de textos é um pouco mais complicada. Este capítulo se destina a fornecer informações suficientes para iniciar com o uso da GDI efetivamente dentro do ambiente do Visual C++. Será apresentado como utilizar fontes efetivamente tanto na impressora como na tela.

1 As classes de Contexto de Dispositivo.

Nos exemplos anteriores, capitulo 2 e 3, a função membro da classe view OnDraw, recebia como parâmetro um ponteiro para um objeto contexto de dispositivo que foi utilizado para selecionar um preenchimento e desenhar uma elipse. No Windows, um contexto de dispositivo é um elemento da GDI que representa um dispositivo físico.

A MFC possui varias classes que representam um contexto de dispositivo (ver Hierarchy Chart) . A classe base CDC, possui todos os membros necessários a realização de um desenho. Quando um objeto contexto de dispositivo é criado, seu ponteiro pode ser passado a funções como, por exemplo, a OnDraw.

Para contexto de tela, normalmente são utilizadas as classes derivadas CClientDC e CWindowDC. Para outros dispositivos, como impressoras e buffers de memória, a classe base CDC é normalmente utilizada.

1.1 Classes de contexto de tela CClientDC e CWindowDC

Lembrando que a área cliente de uma janela exclui as bordas, a barra de título, o menu e as barras de ferramentas e status, se um objeto do tipo CClientDC for criado, ele corresponderá a um contexto de dispositivo que estará mapeando apenas a área cliente portanto, não será possível desenhar fora dela. O ponto (0,0) desse contexto estará referenciando o canto superior esquerdo da área cliente da janela.

Se um objeto do tipo CWindowDC for criado, o ponto (0,0) estará se referenciando ao canto superior esquerdo da área não-cliente da janela. Lembrar que a janela view não possui uma área não cliente, portanto o contexto CWindowDC é mais aplicável a janelas frames.

1.2 Construindo e destruindo objetos CDC

Após construir um objeto do tipo CDC, é muito importante destruí-lo, logo após ele não mais estar sendo utilizado. O Windows limita o número de contextos de dispositivo disponíveis, e se um contexto não for destruído (release), uma pequena quantidade de memória ficará perdida até o fechamento de seu programa.

Freqüentemente é necessário criar um objeto contexto de dispositivo dentro de funções que mapeiam as mensagens, como por exemplo a OnLButtonDown. O meio mais

fácil de garantir que o objeto contexto de dispositivo seja destruído, é criá-lo na pilha (stack). O código a seguir ilustra esta operação:

```
void CCurso3BView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CClientDC dc(this);                     // constroi um dc na pilha
        CRect rectDevice = m_rectElipse;
        dc.LPtoDP(rectDevice);                  // converte coordenadas
}
```

Repare que, desta forma o construtor da classe CClientDC recebe um ponteiro para uma janela view. O destrutor da classe CClientDC é chamado quando a função termina (retorna).

Outra forma de conseguir um contexto em uma função mapeada é utilizando o membro da classe mãe CWnd::GetDC, como mostrado no código abaixo. Sempre que este método for utilizado deve-se chamar a função ReleaseDC para destruir o contexto de dispositivo criado.

```
void CCurso3BView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CDC* pDC = GetDC();             // ponteiro para contexto interno
        CRect rectDevice = m_rectElipse;
        pDC->LPtoDP(rectDevice);        // converte coordenadas
        ReleaseDC(pDC);                 // não deve ser esquecido.
}
```

NOTA: um contexto criado para ser enviado como parâmetro para a função OnDraw não deve ser destruido, a estrutura das aplicações se encarregará disso.

1.3 Estado de um contexto de dispositivo

Como dito anteriormente um contexto de dispositivo é necessário para fazer desenhos. Quando um objeto da classe CDC é utilizado para desenhar um elipse, por exemplo, o que é visto na tela (ou impressora) depende do estado atual desse contexto. Esses estados incluem:

- Objetos de desenho da GDI fixados como por exemplo pens, brushes, e fontes.
- O modo de mapeamento que determinará as escalas que os objetos serão desenhados;
- Outros detalhes como alinhamento de texto, modos de preenchimento de um polígono etc.

No visto nos exemplos anteriores, a seleção de preenchimento cinza antes de desenhar a elipse faz com que seu interior apareça preenchido com a cor cinza. Quando um objeto do tipo CDC é criado, ele possui certas características que são padrões, como por exemplo um pena preta para desenhar linhas e bordas, nenhum preenchimento, etc. Objetos da GDI podem ser fixados a um contexto utilizando-se a função membro SelectObject. Um contexto de dispositivo, pode conter uma pena, um preenchimento, uma fonte selecionados (fixados) ao mesmo tempo. Outras características desse contexto podem alteradas através do uso de sua interface.

2 Objetos da GDI.

Objetos da GDI, são representados por uma classe da MFC. CGdiObject é uma classe base abstrata para encapsular os objetos da GDI, que são representados dentro do Visual C++ por classes derivadas desta. As seguintes classes são suportadas pelo Visual C++:

- CBitmap
- CBrush
- CFont
- CPalette
- CPen
- CRgn

2.1 Construindo e destruindo objetos da GDI

Nunca deve ser construído um objeto da classe base *CGdiObject*, apenas objetos de suas classes derivadas (característica das classes virtuais). Os construtores das classes CPen, CBrush permitem que sejam especificados todos os parâmetros para a sua criação de uma só vez. Outros como os das classes CFont e CRgn, necessitam de um segundo passo para processar a criação. Para estas classes, o construtor padrão deve ser utilizado e em seguida um outra função membro de inicialização como por exemplo CreateFont ou *CreatePolygonRgn* devem ser chamadas para completar a criação do objeto.

A classe *CGdiObject* possui um destrutor virtual. Cada classe derivada deve se encarregar de destruir os objetos de seu tipo. A destruição de um objeto da GDI deve ser feita após o mesmo não esta mais fixado a nenhum contexto de dispositivo.

> NOTA: não deletar um objeto da GDI era um falha grave dentro do win16, pois o mesmo permanecia alocado até o encerramento do Windows. Com o Win32 os programas possuem sua própria memória da GDI que é desalocada toda vez que o programa for terminado. Porém não cuidar de deletar os objetos da GDI podem causar uma grande perda de memória durante o uso de seu programa principalmente se o objeto for do tipo CBitmap.

Agora que já foi discutido a importância de deletar um objeto da GDI, pode-se falar em como desconectar desses objetos de um contexto de dispositivo. Os membro da classe CDC::*SelectObject*, constituem uma família de funções que se encarregam de selecionar o tipo apropriado ao contexto e retornam um ponteiro para o objeto do mesmo tipo anteriormente selecionado. O problema consiste de que não é possível desselecionar um objeto sem selecionar outro. Isto pode ser resolvido da seguinte forma. O objeto anteriormente selecionado que retorna com a função *SelectObject* deve ser guardado quando um objeto criado é selecionado. Este objeto "guardado" deve ser restaurado a GDI quando não mais for utilizado o objeto criado. O trecho abaixo exemplifica como deve ser procedido para utilizar objetos GDI com eficiência:

```
CMyView::OnDraw(CDC* pDC)
{
        CPen pena(PS_DASHDOTDOT, 2, (COLORREF) 0); //pena preta, tamanho2,
                                                   // estilo ponto traço

        CPen* pOldPen = pDC->SelectObject(&pena);

        pDC->MoveTo( 10, 10 );
        pDC->LineTo(110, 10 );

        pDC->SelectObject(pOldPen);             // pena é desselecionada

}//pena é automaticamente destruída na saída, pois não está mais sendo usada
```

Quando um contexto é destruído, todos objetos da GDI que estavam fixados a ele são desselecionados.

2.2 Objetos GDI do Estoque

O Windows mantém alguns objetos da GDI em estoque. Porque estes objetos fazem parte do Windows não é necessário se preocupar em deletá-los. A função membro CDC::*SelectStockObject* pode ser utilizada para associar um objeto do estoque a um contexto. Esses objetos podem ser utilizados como uma alternativa para substituir a forma descrita no item anterior de manter o objeto anteriormente selecionado. Ao invés disso quando um objeto da GDI não for mais ser utilizado, seleciona-se um objeto do estoque para desassocia-lo do contexto. Com isso o exemplo anterior ficaria com a seguinte forma:

```
CMyView::OnDraw(CDC* pDC)
{
        CPen pena(PS_DASHDOTDOT, 2, (COLORREF) 0); //pena preta, tamanho2,
                                                   // estilo ponto traço

        pDC->SelectObject(&pena);

        pDC->MoveTo( 10, 10 );
        pDC->LineTo(110, 10 );

        pDC->SelectStockObject(BLACK_PEN);      // pena é desselecionada

}//pena é automaticamente destruída na saída, pois não está mais sendo usada
```

2.3 Tempo de vida de uma seleção da GDI

Para contexto de tela, um novo contexto é sempre que necessário no início de uma função mapeada. Nenhuma seleção feita para este contexto permanece após a saída da função. Será necessário inicializar o contexto toda vez que for necessário. A função *OnPrepareDC* é útil para selecionar o modo de mapeamento, porém os objetos da GDI tem que ser gerenciados toda vez que for necessário utilizá-los.

Para outros contextos, como por exemplo o de impressão ou um contexto de memória, esta associação pode tornar-se longa. Para contextos com uma longa vida útil, as coisa se tornam um pouco mais complicadas. Essa complexidade resulta da natureza temporária dos ponteiros para objetos retornados pela função *SelectObject*. (Objetos temporários podem ser destruídos pela estrutura das aplicações quando não mais são utilizados).

Esses ponteiros podem ser convertidos em um Windows handle (Identificador permanente da GDI) utilizando-se a função *GetSafeHdc*, e armazenados como membros de dados de uma classe. A seguir temos um exemplo dessa técnica:

```
void CMyView::TrocaParaCourier(CDC* pDC)
{
        m_pFonteImpressao->CreateFont(30,10,0,0, 400, FALSE, FALSE, 0,
                                      ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                      CLIP_DEFALUT_PRECIS, DEFAULT_QUALIT,
                                      DEFAULT_PITCH | FF_MODERN,
                                       "Courier New"); // fonte True Type
        CFont* pOldFont = pDC->SelectObject(m_pFonteImpressao);
        m_hOldFont = (HFONT) pOldFont->GetSafeHandle();
}

void CMyView::TrocaParaOriginal(CDC* pDC)
{
        if (m_hOldFont)
                pDC->SelectObject(CFont::FromHandle(m_hOldFont);
}
```

3 Cores no Windows.

A GDI proporciona uma independência do hardware utilizado para exibir as cores. Os programas trabalham com cores absolutas e a GDI se encarrega de fazer as combinações necessárias para que essas cores fiquem visíveis no Windows.

Atualmente o Windows suporta 4 tipos de modalidades de cores que são processadas pelas respectivas placas de vídeo.

3.1 Placas de vídeo VGA padrão (16 cores)

Com estas placas de vídeo só é possível visualizar 16 cores diferentes ao mesmo tempo na janela. Essas cores são formadas por uma combinação de três cores básicas o vermelho (R), o verde (G) e o Azul (B).

Funções da GDI que trabalham com cores aceitam como parâmetro o tipo COLORREF de 32 bits que é composto por 8bits de cada cor vermelho, verde e azul. A macro do Windows, RGB converte os três parâmetros de 8 bits de cada em uma variável do tipo COLORREF.

O exemplo a seguir mostra como criar um brush utilizando esta macro.

```
CBrush brush(RGB(0,0,255));
```

As funções membros da classes CDC, SetBkColor e SetTextColor podem ser utilizadas para trocar a cor de fundo de um texto e trocar a cor das letras respectivamente. Neste modo de vídeo não é possível exibir cores compostas para fontes, a cor sólida mais próxima será mostrada.

A tabela a seguir mostra alguns exemplos de cores sólidas que podem ser utilizadas neste modo.

Vermelho (R)	Verde (G)	Azul (B)	Cor
0	0	0	Preto
0	0	255	Azul
0	255	0	Verde
255	0	0	Vermelho
0	255	255	Turquesa
255	0	255	Rosa
255	255	0	Amarelo
255	255	255	Branco
0	0	128	Azul escuro
0	128	0	Verde escuro
0	128	128	Turquesa escuro
128	0	0	Vermelho escuro
128	0	128	Rosa escuro
128	128	0	Amarelo escuro
128	128	128	Cinza escuro
192	192	192	Cinza claro

3.2 Placas de vídeo de 256 cores

Considerado como o denominador comum de todos os modos de vídeo, permite a utilização simultânea de 256, bem como o uso de paletes para melhorar uma apresentação de cores.

3.3 Placas de vídeo de 16 bit's de cores

Permite o uso simultâneo de 32.768 cores, porém não suporta o uso de paletes. Utiliza no mínimo 2M de memória de vídeo para conseguir este modo com uma resolução de 1024x768, cada cor primária é representada por 5 bits, portanto só possível uma variação de 32 níveis para cada cor primária.

3.4 Placas de vídeo de 24 bit's de cores

Permite o uso simultâneo de mais de 16.7 milhões de cores. Também não suporta uso de paletes. Utiliza no mínimo 2.5M de memória de vídeo para conseguir este modo com uma resolução de 1024x768, cada cor primária é representada por 8 bits, permitindo uma variação de 256 níveis para cada cor primária. (Obs. em uma floresta são encontrados em média 236 níveis de verde, devido a isso esse modo de cores é também chamado de True Colors)

4 Fontes.

Nos programas anteriores, quando um texto era exibido na tela ele utilizava a fonte System, do Windows. Como é comum a todos, o Windows suporta múltiplas fontes, independentes do dispositivo e com tamanhos variáveis. Fontes True-Type, são mais fáceis de ser manipuladas dentro dos programas do que as fontes que eram dependentes de dispositivos (ex. uma fonte para impressora e outra para a tela). Após esta introdução teórica sobre as fontes serão apresentados exemplos ao final deste capítulo.

4.1 Fontes são objetos da GDI

As fontes são parte integral da GDI do Windows. Por esta razão as fontes seguem as mesmas regras descritas anteriormente para objetos da GDI, por exemplo podem ser selecionadas dentro de um contexto de dispositivo, devem ser desselecionadas após o uso e também devem seguir a regra de deleção.

4.2 Escolhendo uma fonte

Dois tipos de fontes podem ser escolhidas, fontes independente do dispositivo (True Type) e fontes dependentes do dispositivo como por exemplo a fonte System do Windows e a fonte LinePrinter de um LaserJet ou alguma fonte de uma DeskJet. Ou simplesmente escolher a categoria e características e deixar que o Windows escolha a fonte por você.

O diálogo de seleção de fontes do Windows é suportado pela MFC, e é exibido de acordo com a impressora ativa. Com isso é possível ao usuário selecionar uma fonte que seja perfeita na impressão, porém na tela ela deverá ser exibida de modo aproximado da melhor forma possível.

4.3 Imprimindo com fontes

Em programas que utilizem a impressão de textos, é comum especificar o tamanho das fontes em pontos (1 ponto = 1/72 in). Devido ao fato das fontes das impressoras serem especificadas também em pontos.

Fontes True Type podem ser espeificadas para qualquer número de pontos. Para trabalhar com pontos o modo de mapeamento mais apropriado é o MM_TWIPS. Lembrar que para uma fonte de tamanho 8,5 pontos a altura do caractere deve ser especificada por 8,5 x 20, ou 170 twips.

4.4 Exibindo Fontes

A Flexibilidade para exibir fontes apenas na tela, esquecendo-se da impressora, é bem maior. Para fontes True Type não interessa o modo de mapeamento, apenas escolha o tamanho e use-a. Não é necessário se preocupar com os pontos.

4.5 Polegadas físicas e lógicas do Vídeo

A função GetDeviceCaps, membro da classe CDC, retorna várias medidas a respeito da tela que são importantes para programas gráficos. As seis medidas mostradas na tabela abaixo referem-se a uma placa de vídeo padrão configurada para a resolução de 640x480.

Indice	Descrição	Valor
HORZSIZE	Largura física em milímetros	169
VERTSIZE	Altura física em milímetros	127
HORZRES	Largura em pixels	640
VERTRES	Altura em pixels	480
LOGPIXELSX	Pontos por polegadas lógicas na horizontal	96
LOGPIXELSY	Pontos por polegadas lógicas na vertical	96

Os índices HORZSIZE e VERTSIZE representam as dimensões físicas do monitor. (esses indices podem não corresponder com a realidade pois o Windows não sabe exatamente o tamanho do monitor que está acoplado a placa de vídeo). O tamanho da tela pode ser calculada multiplicando-se HORZRES e VERTRES por LOGPIXELSX e LOGPIXELSY respectivamente.

Um modo de mapeamento utilizado pelo Word, é chamado de logical twips, onde cada unidade lógica corresponde exatamente a 1/1440 polegadas lógicas. Este modo de mapeamento é independente do sistema operacional e da resolução do display.

Para setar este modo pode utilizar o seguinte código

```
pDC->SetMapMode(MM_ANISOTROPIC);
pDC->SetWindowEst(1440,1440);
pDC->SetViewportExt(pDC->GetDeviceCaps(LOGPIXELSX),
                    pDC->GetDeviceCaps(LOGPIXELSY) );
```

4.6 Altura dos caracteres.

A função GetTextMetrics, membro da classe CDC, permite que seja acessados cinco medidas da altura dos caracteres, porém apenas três são realmente importantes. O parâmetro `tmHeight`, representa o tamanho total de um caractere incluído a parte abaixo da linha (para os caracteres g,p,q,j, y) e os espaços para os acentos. O parâmetro `tmExternalLeading` representa o espaçamento entre a parte abaixo da linha e os acentos. Pode-se pensar que o parâmetro `tmHeight` seja o tamanho da fonte em pontos, porém para calcular o tamanho da fonte em pontos é necessário outro parâmetro, `tmInternalLeading` onde o tamanho em pontos da fonte é dado pela diferença entre esses dois parâmetros.

5 Exemplo Curso4A

5.1 Execute o AppWizard para gerar o exemplo Curso4A

Escolha a opção *File New* e selecione *MFC AppWizard (EXE)* na pasta *Project*. Selecione a opção *Single Document* (SDI) e desmarque a opção *Print Preview*, Aceite todas outras opções padrões.

As opções de criação e o nomes das classes a serem criadas está mostrada na figura abaixo.

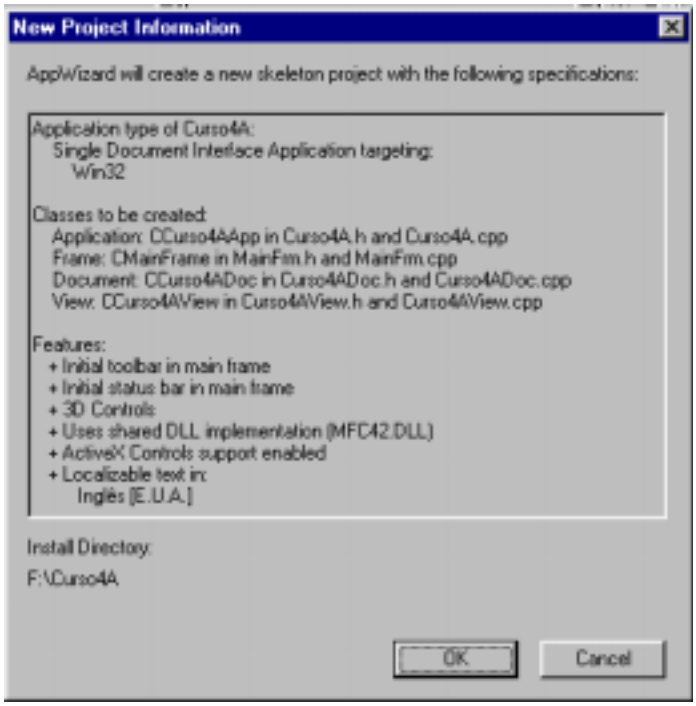


Figura 17 – Opções para o exemplo a ser gerado

5.2 Utilize o ClassWizard para sobrecarregar a função virtual OnPrepareDC da classe view

Edite o código no arquivo Curso4AView.cpp deixando-o como o a seguir:

```
void CCurso4AView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
        pDC->SetMapMode(MM_ANISOTROPIC);
        pDC->SetWindowExt(1440, 1440);
        pDC->SetViewportExt(pDC->GetDeviceCaps(LOGPIXELSX),
                            -pDC->GetDeviceCaps(LOGPIXELSY));
}
```

5.3 Crie uma função protegida ExibirFonte

Adicione o protótipo para ela dentro do arquivo Curso4AView.h

```
protected:
        void ExibirFonte(CDC* pDC, int& nPos, int nPoints);
```

E o corpo da função deve ser como o mostrado abaixo:

```
void CCurso4AView::ExibirFonte(CDC* pDC, int& nPos, int nPoints)
{
        TEXTMETRIC tm;
        CFont      fontText;
        CString    strText;
        CSize      sizeText;

        fontText.CreateFont(-nPoints * 20, 0, 0, 0, 400, FALSE, FALSE, 0,
                            ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                            CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                            DEFAULT_PITCH | FF_SWISS, "Arial");
        CFont* pOldFont = (CFont*) pDC->SelectObject(&fontText);
        pDC->GetTextMetrics(&tm);
        TRACE("pontos = %d, tmHeight = %d, tmInternalLeading = %d,"
              " tmExternalLeading = %d\n", nPoints, tm.tmHeight,
              tm.tmInternalLeading, tm.tmExternalLeading);
        strText.Format("Esta é uma fonte Arial de %d-pontos", nPoints);
        sizeText = pDC->GetTextExtent(strText);
        TRACE("largura da string = %d, altura da string = %d\n", sizeText.cx,
              sizeText.cy);
        pDC->TextOut(0, nPos, strText);
        pDC->SelectObject(pOldFont);
        nPos -= tm.tmHeight + tm.tmExternalLeading;
}
```

5.4 Edite o membro OnDraw no arquivo Curso4Aview.cpp

AppWizard sempre cria a função membro OnDraw, procure-a e edite seu código como o abaixo:

```
void CCurso4AView::OnDraw(CDC* pDC)
{
        int nPosition = 0;

        for (int i = 6; i <= 24; i += 2) {
                ExibirFonte(pDC, nPosition, i);
        }
        TRACE("LOGPIXELSX = %d, LOGPIXELSY = %d\n",
              pDC->GetDeviceCaps(LOGPIXELSX),
              pDC->GetDeviceCaps(LOGPIXELSY));
        TRACE("HORZSIZE = %d, VERTSIZE = %d\n",
              pDC->GetDeviceCaps(HORZSIZE),
              pDC->GetDeviceCaps(VERTSIZE));
        TRACE("HORZRES = %d, VERTRES = %d\n",
              pDC->GetDeviceCaps(HORZRES),
              pDC->GetDeviceCaps(VERTRES));
}
```

5.5 Compile e execute o programa.

O programa criado deve ser executado no modo debug (Go) para que a macro trace exiba as informações de execução do programa na janela Output.

A tela de saída do programa deve ser como a mostrada abaixo:

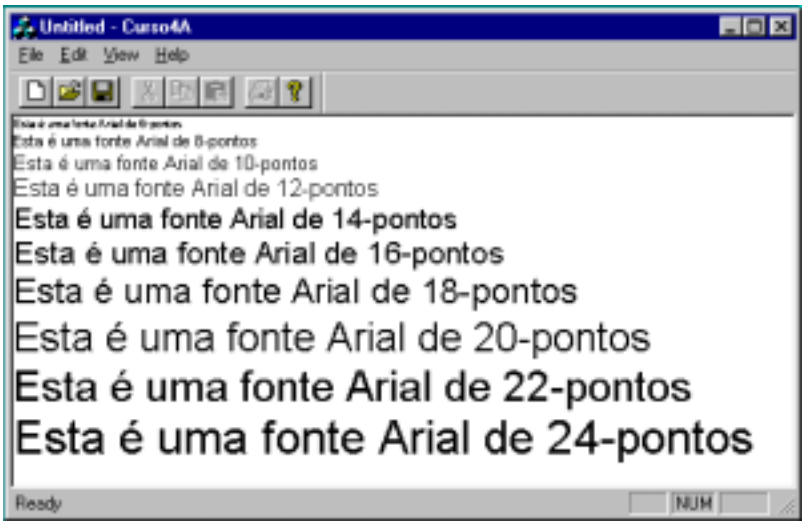


Figura 18 – Tela de saída do programa Curso4A.exe

As informações contidas na janela Output sobre a execução do programa são as a seguir:

```
pontos = 6, tmHeight = 150, tmInternalLeading = 30, tmExternalLeading = 0
largura da string = 1725, altura da string = 150
pontos = 8, tmHeight = 210, tmInternalLeading = 45, tmExternalLeading = 0
largura da string = 2505, altura da string = 210
pontos = 10, tmHeight = 240, tmInternalLeading = 45, tmExternalLeading = 0
largura da string = 3120, altura da string = 240
pontos = 12, tmHeight = 270, tmInternalLeading = 30, tmExternalLeading = 15
largura da string = 3735, altura da string = 270
pontos = 14, tmHeight = 330, tmInternalLeading = 45, tmExternalLeading = 15
largura da string = 4545, altura da string = 330
pontos = 16, tmHeight = 360, tmInternalLeading = 45, tmExternalLeading = 15
largura da string = 5025, altura da string = 360
pontos = 18, tmHeight = 405, tmInternalLeading = 45, tmExternalLeading = 15
largura da string = 5775, altura da string = 405
pontos = 20, tmHeight = 480, tmInternalLeading = 75, tmExternalLeading = 15
largura da string = 6525, altura da string = 480
pontos = 22, tmHeight = 495, tmInternalLeading = 60, tmExternalLeading = 15
largura da string = 6945, altura da string = 495
pontos = 24, tmHeight = 540, tmInternalLeading = 60, tmExternalLeading = 15
largura da string = 7575, altura da string = 540
LOGPIXELSX = 96, LOGPIXELSY = 96
HORZSIZE = 211, VERTSIZE = 158
HORZRES = 800, VERTRES = 600
```

5.6 Os elementos do programa Curso4A

A seguir encontra-se uma discussão sobre os elementos mais importantes desse exemplo:

5.6.1 Selecionando o modo de mapeamento na função OnPrepareDC

A estrutura das aplicações chama a função OnPrepareDC antes de chamar a função OnDraw, portanto este é o melhor lugar para preparar o contexto de dispositivo a ser utilizado. Para utilizar o modo de mapeamento e as características corretas para um dispositivo, basta chamar esta função após o mesmo ser criado.

5.6.2 A função membro protegida ExibirFonte

Essa função contém o código que é executado 10 vezes dentro de um loop. Em C, esta seria uma função global, mas em C++ o melhor meio de declarar esta função é como membro protegido da classe View. Esses membros são muitas vezes chamados de funções de apoio, pois não podem ser executadas de fora do objeto. Portanto não fazem parte de sua interface, apenas de sua implementação.

Ela cria uma fonte, a seleciona em um contexto de dispositivo e imprime na tela. Se o programa for executado no modo Debug, esta função mostra informações importantes sobre as dimensões das fontes exibidas na janela de Output.

5.6.3 Chamando CFont::CreateFont

A chamada dessa função inclui muitos parâmetros, porém os mais importantes são os dois primeiros, altura e largura da fonte.

O ultimo parâmetro da função especifica o nome da fonte.

6 Exemplo Curso4B

Este exemplo é similar ao anterior porém possui duas diferenças básicas:

1. Utiliza do modo de mapeamento MM_ANISOTROPIC com uma escala dependente do tamanho da janela.
2. Exibe múltiplas fontes na mesma tela

Os seguintes passos deve ser seguidos para a criação desse exemplo:

6.1 Passo1: Execute o AppWizard para criar o projeto Curso4B.

As opções selecionadas e os padrões estão listados na figura abaixo.

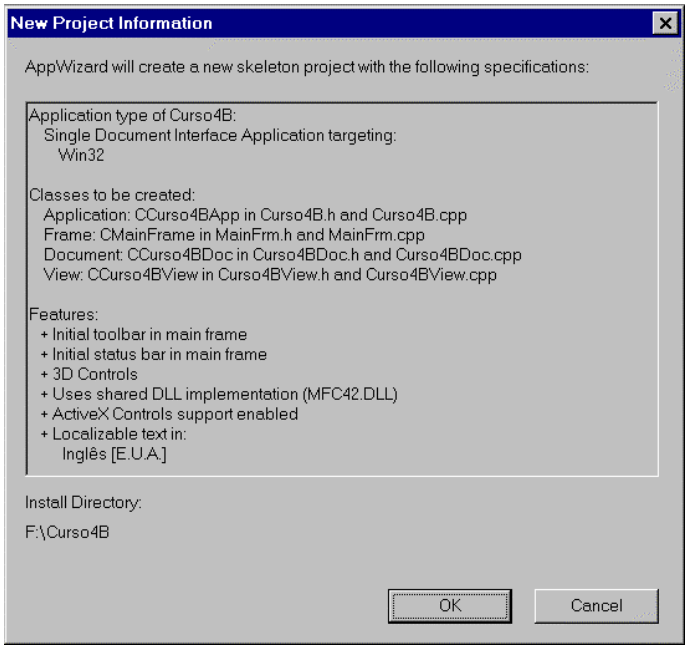


Figura 19 – Opções para o projeto Curso4B

6.2 Passo2: Utilize o ClassWizard para sobrecarregar a função OnPrepareDC da classe CCurso5BView.

Edite o código no arquivo Curso4BView.cpp e torne-o como o a seguir:

```
void CCurso4BView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
        CRect clientRect;

        GetClientRect(clientRect);
        pDC->SetMapMode(MM_ANISOTROPIC); // +y = down
        pDC->SetWindowExt(400,450);
        pDC->SetViewportExt(clientRect.right, clientRect.bottom);
        pDC->SetViewportOrg(0, 0);
}
```

6.3 Passo3: Crie a função protegida TraceMetrics

Adicione o seguinte protótipo no arquivo Curso4BView.h

```
protected:
        void TraceMetrics(CDC* pDC);
```

Adicione o corpo da função no arquivo de implementação Curso4BView.cpp.

```
void CCurso4BView::TraceMetrics(CDC* pDC)
{
        TEXTMETRIC tm;
        char       szFaceName[100];

        pDC->GetTextMetrics(&tm);
        pDC->GetTextFace(99, szFaceName);
        TRACE("font = %s, tmHeight = %d, tmInternalLeading = %d,"
              " tmExternalLeading = %d\n", szFaceName, tm.tmHeight,
              tm.tmInternalLeading, tm.tmExternalLeading);
}
```

6.4 Passo4: Editar a função membro OnDraw para deixá-la como no código abaixo:

Torne o código da função OnDraw como o a seguir:

```
void CCurso4BView::OnDraw(CDC* pDC)
{
        CFont fontTest1, fontTest2, fontTest3, fontTest4;

        fontTest1.CreateFont(50, 0, 0, 0, 400, FALSE, FALSE, 0,
                             ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                             CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                             DEFAULT_PITCH | FF_SWISS, "Arial");
        CFont* pOldFont = pDC->SelectObject(&fontTest1);
        TraceMetrics(pDC);
        pDC->TextOut(0, 0, "This is Arial, default width");

        fontTest2.CreateFont(50, 0, 0, 0, 400, FALSE, FALSE, 0,
```

```
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                          DEFAULT_PITCH | FF_MODERN, "Courier"); // not TrueType
        pDC->SelectObject(&fontTest2);
        TraceMetrics(pDC);
        pDC->TextOut(0, 100, "This is Courier, default width");

        fontTest3.CreateFont(50, 10, 0, 0, 400, FALSE, FALSE, 0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_ROMAN, NULL);
        pDC->SelectObject(&fontTest3);
        TraceMetrics(pDC);
        pDC->TextOut(0, 200, "This is generic Roman, variable width");

        fontTest4.CreateFont(50, 0, 0, 0, 400, FALSE, FALSE, 0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                DEFAULT_PITCH | FF_MODERN, "LinePrinter");
        pDC->SelectObject(&fontTest4);
        TraceMetrics(pDC);
        pDC->TextOut(0, 300, "This is LinePrinter, default width");
        pDC->SelectObject(pOldFont);
}
```

6.5 Passo5: Compile e execute o programa Curso4B.exe

Realize teste de redimensionamento da janela e note que as fontes alteram seu tamanho e algumas a forma. Os resultados desse exemplo estão mostrados nas figuras a seguir:

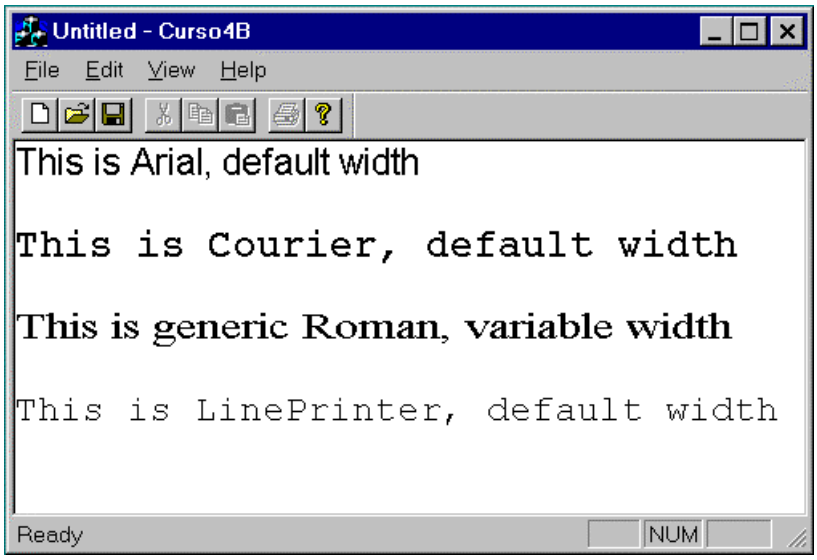


Figura 20 – Tela original do programa Curso4b.exe

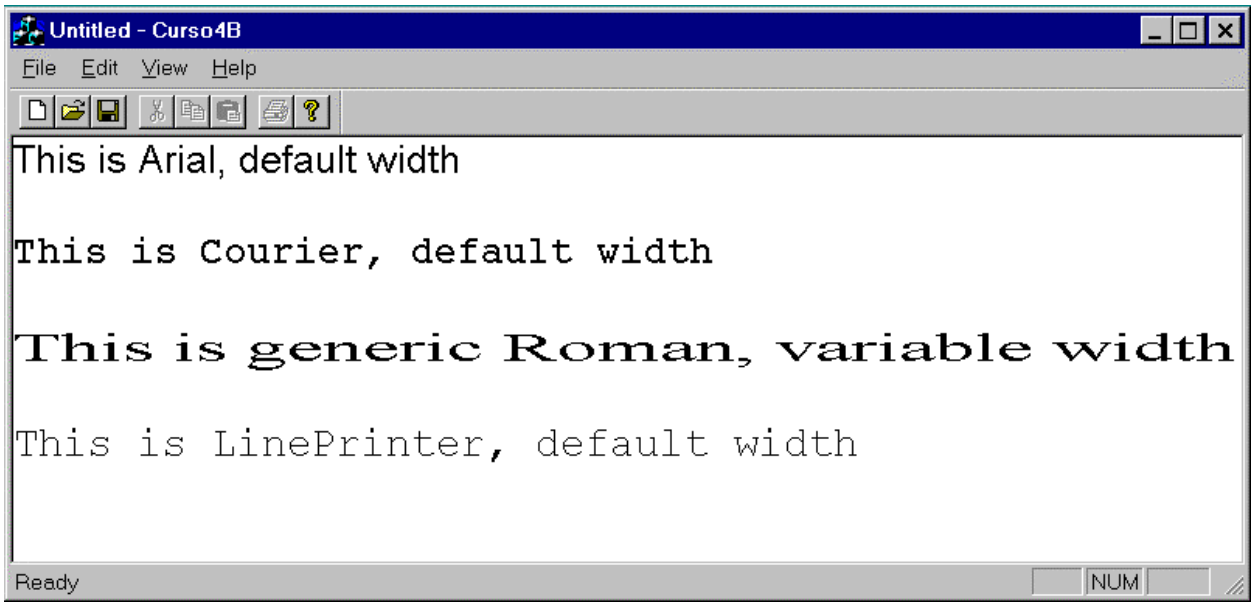


Figura 21 – Tela redimensionada do programa Curso4B.exe

6.6 Elementos do programa Curso4B.exe

6.6.1 A função membro OnDraw:

Nesta função é exibida uma string em quatro fontes diferentes:

- FontTest1
- FontTest2
- FontTest3
- FontTest4

6.6.2 A função TraceMetrics

Dentro dessa função são chamados os membros CDC::*GetTextMetrics* e CDC::*GetTextFace*, com os parâmetros da fonte atual selecionados, e exibe as informações sobre os caracteres na janela Output.

7 Exemplo Curso4C

No exemplo Curso3C, foi inciado a utilização da classe *CScrollView*. Neste exemplo será mostrado como proceder para mover um elipse com o mouse, utilizando uma janela rolável com o modo de mapeamento MM_LOENGLISH. A rolagem pelo teclado foi deixada de fora, porém se desejável, basta utilizar os passos descritos no exemplo Curso3C para criar a função *OnKeyDown* semelhante.

Em vez de utilizar um brush do estoque, será utilizado um brush hachurado para o preenchimento da elipse. Só há uma restrição para o uso de brushes hachurados, a origem da janela rolante deve ser resetada, caso contrário o efeito de preenchimento pode parecer esquisito.

Como no programa Curso3C, este exemplo utiliza uma classe View derivada de *CScrollView*. (Não esqueça de mudar isso no passo 6 do AppWizard) aqui estão os passos para criar este aplicativo:

7.1 Passo1: Execute o AppWizard para gerar o projeto Curso4C.

Certifique-se que a classe base da view seja uma CScrollView, e aceite todas outras opções como padrão.

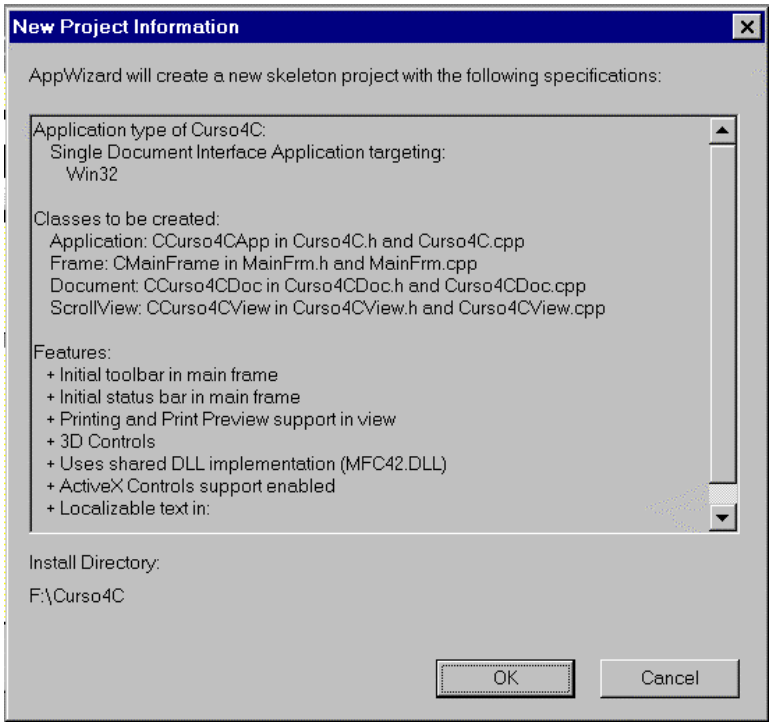


Figura 22- Opções para o programa Curso4C

7.2 Passo2: Edite o arquivo de descrição da classe CCurso4CView

Adicione as seguintes variáveis membros a classe:

```
protected:
        const CSize m_sizeElipse; // logical
        CPoint m_pointTopLeft;    // logical, canto superior esquerdo do
                                  // retangulo que define a elipse
        CSize  m_sizeOffset;      // dispositivo, do canto superior esquerdo
                                  // até o ponto de captura
        BOOL   m_bCaptured;       // flag que indica estado da captura do mouse
```

7.3 Passo3:Utilize o ClassWizard para mapear as três mensagens do mouse

Adicione os manipuladores de mensagem mostrados na tabela abaixo:

Mensagens	**Função membro**
WM_LBUTTONDOWN	OnLButtonDown
WM_LBUTTONUP	OnLButtonUp
WM_MOUSEMOVE	OnMouseMove

7.4 Passo4: Editar as funções que acabaram de ser mapeadas tornando-as como as da listagem abaixo.

O ClassWizard gera o esqueleto das funções que foram mapeadas no passo anterior, encontre-as no arquivo de implementação curso4CView.cpp e codifique-as como a seguir:

```
void CCurso4CView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CRect rectEllipse(m_pointTopLeft, m_sizeElipse); // ainda logicas
        CRgn  circle;

        CClientDC dc(this);
        OnPrepareDC(&dc);
        dc.LPtoDP(rectEllipse);   // Agora em coordenadas do dispositivo
        circle.CreateEllipticRgnIndirect(rectEllipse);
        if (circle.PtInRegion(point)) {
                // Capturando o mouse até a próxima mensagem LButtonUp
                SetCapture();
                m_bCaptured = TRUE;
                CPoint pointTopLeft(m_pointTopLeft);
                dc.LPtoDP(&pointTopLeft);
                m_sizeOffset = point - pointTopLeft; //coordenadas do dispositivo
                // Novo cursor enquanto o mouse estivar capturado
                ::SetCursor(::LoadCursor(NULL, IDC_CROSS));
        }
}

void CCurso4CView::OnLButtonUp(UINT nFlags, CPoint point)
{
        if (m_bCaptured) {
                ::ReleaseCapture();
                m_bCaptured = FALSE;
        }
}

void CCurso4CView::OnMouseMove(UINT nFlags, CPoint point)
{
        if (m_bCaptured) {
                CClientDC dc(this);
                OnPrepareDC(&dc);
                CRect rectOld(m_pointTopLeft, m_sizeElipse);
                dc.LPtoDP(rectOld);
                InvalidateRect(rectOld, TRUE);
                m_pointTopLeft = point - m_sizeOffset;
                dc.DPtoLP(&m_pointTopLeft);
                CRect rectNew(m_pointTopLeft, m_sizeElipse);
                dc.LPtoDP(rectNew);
                InvalidateRect(rectNew, TRUE);
        }
}
```

7.5 Passo5: Editar o construtor da classe CCurso4CView, a função membro OnDraw e a função membro OnInitialUpdate

O AppWizard gera o esqueleto dessas funções, encontre-as no arquivo de implementação curso4CView.cpp e codifique-as como a seguir:

```
CCurso4CView::CCurso4CView() : m_sizeElipse(100, -100),
                               m_pointTopLeft(0, 0),
                               m_sizeOffset(0, 0)
{
```

```
        m_bCaptured = FALSE;
}


void CCurso4CView::OnDraw(CDC* pDC)
{
        CBrush brushHatch(HS_DIAGCROSS, RGB(255, 0, 0));
        CPoint point(0, 0);             // logical (0, 0)

        pDC->LPtoDP(&point);            // Em coordenadas do dispositivo,
        pDC->SetBrushOrg(point);        //  Alinhar o brush com a origem
                                        //  da janela
        pDC->SelectObject(&brushHatch);          // Seleciona Brush
        pDC->Ellipse(CRect(m_pointTopLeft, m_sizeElipse));
        pDC->SelectStockObject(BLACK_BRUSH); // Desseleciona brushHatch
        // Testa retangulo invalido
        pDC->Rectangle(CRect(100, -100, 200, -200));
}

void CCurso4CView::OnInitialUpdate()
{
        CScrollView::OnInitialUpdate();

        CSize sizeTotal(800, 1050);             // 8-por-10.5 polegadas
        CSize sizePage(sizeTotal.cx / 2, sizeTotal.cy / 2);
        CSize sizeLine(sizeTotal.cx / 50, sizeTotal.cy / 50);
        SetScrollSizes(MM_LOENGLISH, sizeTotal, sizePage, sizeLine);
}
```

7.6 Passo6: Compile e execute o programa Curso4C.exe

Este programa permite que a elipse possa ser arrastada com o mouse, além de permitir que a janela seja rolada.

Ao mover a elipse sobre o retângulo preto é possível ver o efeito do retângulo inválido. A tela do exemplo é mostrada na figura abaixo:

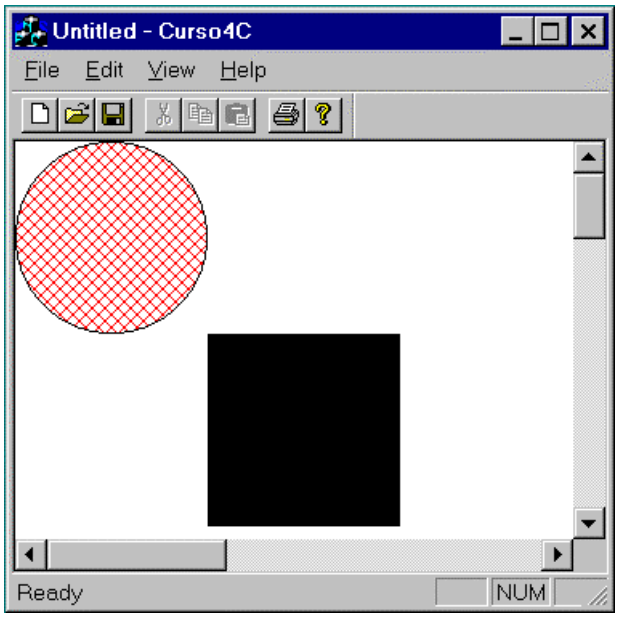


Figura 23 – Execução do programa Curso4C

7.7 Elementos importantes desse programa.

7.7.1 As variáveis membros m_sizeElipse e m_pointTopLeft

Ao invés de armzenar o retângulo que define a elipse em uma variável do tipo CRect, este exemplo o faz de forma separada, armazena o tamanho (m_sizeElipse) e a posição do canto superior esquerdo (m_pointTopLeft). Para mover esta elipse o programa simplesmente calcula a nova posição do canto superior esquerdo, eliminando com isso quanquer erro de arrefondamento que possa ocorrer no tamanho da elipse.

7.7.2 A variável membro m_sizeOffset

Quando a função OnMouseMove movimenta a elipse, isto é feito relativo a uma posição do mouse dentro da elipse, que é determinada quando o usuário pressiona o botão esquerdo do mouse. O objeto m_sizeOffset armazena a diferença entre o ponto dentro da elipse onde o mouse foi pressionado e o canto superior esquerdo do retângulo que define a elipse.

7.7.3 A variável membro m_bCaptured

A variável lógica m_bCaptured é setada para TRUE, toda vez que o processo de arrasto elipse com o mouse estiver acontecendo.

7.7.4 As funções SetCapture e ReleaseCapture

A função membro de CWnd SetCapture “captura” o mouse, fazendo com que as mensagens de movimento do mouse continuem sendo enviadas a janela mesmo que o cursor do mouse esteja posicionado fora dela. Como efeito do uso dessa função, ocorre que a elipse pode ser arrastada para fora da janela e “perdida”. O feito desejável quando utiliza-se esta função é que todas as mensagens do mouse continuam sendo enviadas para a janela, inclusive a WM_LBUTTONUP, que de outra forma poderia ficar perdida. A função da Win32 ReleaseCapture, desliga a captura do mouse para a janela.

7.7.5 As funções SetCursor e LoadCursor da Win32

Para definir que uma função da Win32 está sendo chamada, ao invés de uma função membro do classe, o operador de qualificação de escopo **::** deve ser utilizado precedendo o nome da função da Win32 que está sendo chamada.

A função LoadCursor possui dois parâmetros, quando o primeiro é NULL, indica que um recurso cursor padrão do Windows está sendo requisitado. Este recurso é ativado pela função SetCursor e permanece ativo enquanto o mouse estiver capturado.

7.7.6 A função membro CScrollView::OnPrepareDC

A função membro virtual OnPrepareDC é utilizada para definir os modos de mapeamento do contexto, e sua origem. A estrutura das aplicações chama a função OnPrepareDC antes de chamar o método OnDraw, portanto não é necessário se preocupar com isso. A chamada da função OnPrepareDC deve ser feita dentro de funções que corresponde a manipulação de mensagens e que necessitam utilizar um contexto, como é o caso das funções OnLButtonDown e OnMouseMove por exemplo.

7.7.7 A transformação de coordenadas dentro da função OnMouseMove.

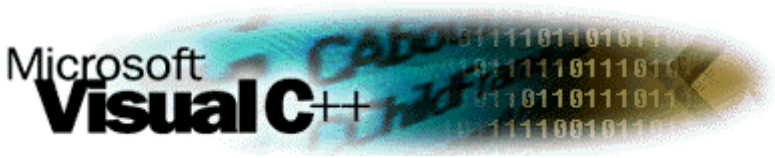
Como pode-se notar existem uma sequencia de comandos que são realizados dentro da função OnMouseMove para proceder transformações de coordenadas e apagar o desenho anterior. Estes comandos podem ser resumidos em:

1. Construir o retângulo atual da elipse e convertê-lo de coordenadas lógicas para coordenadas do dispositivo.
2. Invalidar o retângulo atual.
3. Atualizar o canto superior esquerdo do retângulo que define a elipse
4. Construir o novo retângulo e convertê-lo para coordenadas dodispositivo.
5. Invalidar o novo retângulo.

Notar que essa função chama *InvalidateRect* duas vezes. Porém a execução é realizada de uma só vez, pois o Windows guarda os retângulos e os mescla em um único antes de chamar a função OnDraw para redesenhar a tela.

7.7.8 A função OnDraw.

Dentro da função OnDraw é necessário chamar a função SetBrushOrg, para assegurar que as linhas que preenchem a elipse sejam atualizadas quando a janela é rolada. O alinhamento do brush é feito referenciando-se a um ponto, que neste caso é o topo da janela lógica, convertido para coordenadas do dispositivo. Essa é uma das únicas exceções em que um membro de CDC precisa utilizar coordenadas do dispositivo e não coordenadas lógicas.



Capítulo 5
Caixas de Diálogo e Controles do Windows

Quase todos os programas para o Windows possuem janelas diálogo para interagir com o usuário. Um diálogo pode ser desde um simples Message Box com um botão OK até um formulário complexo para entrada de dados. Esses diálogos são elementos que se comportam como uma verdadeira janela, recebem mensagens, podem ser movidos e fechados, além de aceitar elementos de desenho vistos anteriormente em seu interior.

Há dois tipos de diálogos, Modal e Modeless. Estaremos abordando o tipo mais comum que é o diálogo modal. Ainda neste capítulo serão implementados exemplos de interação com diálogos através de controles do Windows.

1 Diálogo Modal x Diálogo Modeless

A classe base CDialog da MFC suporta ambos os tipos de diálogos. A diferença básica entre eles é que quando um diálogo modal for aberto, o usuário não poderá trabalhar com o aplicativo que chamou este diálogo até que o mesmo seja fechado. Já se um diálogo for do tipo Modeless, o usuário pode trabalhar tanto dentro do diálogo como no aplicativo que o chamou ao mesmo tempo. Um exemplo de diálogo modal seria o Diálogo Abrir, que não permite que nada seja feito no aplicativo até que ele seja fechado. Um exemplo de diálogo Modeless, é a opção Localizar e Substituir do Microsoft Word, quando este diálogo está aberto ainda é possível editar o texto, bem como alterar a palavra que se deseja procurar ou substituir dentro do diálogo.

A escolha entre os tipos Modal e Modeless dependerá da aplicação, porém como os diálogos modais são mais fáceis de serem utilizados muitas vezes eles serão escolhidos.

2 Recursos e controles

Agora que você já sabe que um diálogo é uma janela, o que o torna diferente de uma janela CView será visto nessa seção. Diálogos sempre fazem parte dos recursos de um programa Windows que conterá seu identificador e o layout de como esse diálogo se apresentará na tela. O editor de recursos poderá ser utilizado para construir o layout desses diálogos de forma amigável e completamente visual.

Um diálogo pode conter muitos elementos em seu interior. Esses elementos recebem o nome de Controles. Fazem parte desses controles, as caixas de texto (edit box), botões, list box, caixas de seleção (combo box), textos estáticos(labels), entre outros. Esses controles podem ser referenciados dentro do programa por ponteiros para CWnd, bem como pelo uso de identificadores associados aos recursos, que é o meio mais comum. O controle envia uma mensagem ao diálogo em resposta a uma interação com o usuário como por exemplo o pressionamento de um botão.

A MFC e o ClassWizard trabalham juntos para facilitar o uso de diálogos dentro de um programa Windows. O ClassWizard cria uma classe derivada de CDialog para cada dialogo e permite ainda que sejam associadas variáveis membros a controles do diálogo. Podem ser especificados os limites de números e tamanho de strings para as variáveis membros criadas, que o ClassWizard se encarrega criar as chamas a data validation e data exchange que farão o controle de troca entre os dados que constam na caixa de diálogo e as variáveis membros.

3 Programando um Diálogo Modal

Diálogos modais são mais freqüentemente usados. Uma ação do usuário, como por exemplo uma escolha de um item do menu, pode ser usada para exibir um diálogo na tela. O usuário entra com os dados na caixa de diálogo e em seguida o fecha para continuar utilizando o programa.

A seguir encontram-se os passos necessário para a adição de um diálogo a um projeto já existente:

1. Utilize o editor de diálogos (parte do editor de recursos) para criar um diálogo contendo vários controles. O editor de diálogo atualiza o arquivo script de recursos (RC) para incluir o novo recurso diálogo, e também coloca as constantes criadas (identificadores) no arquivo resource.h do projeto.
2. Utilize o ClassWizard para criar uma classe derivada de CDialog e associada ao recurso criado no passso1. O ClassWizard criará o arquivo de cabeçalho e de implementação da nova classe e o adicionará ao projeto.

   NOTA: Quando o ClassWizard gera uma classe derivada de CDialog, cria um construtor que chama a classe CDialog que possui um identificador de recurso (ID) como parâmetro. Na descrição da classe criada encontra-se a definição da constante IDD que está relacionada com o identificador único do diálogo. No arquivo de implementação o construtor possui o seguinte formato:

   ```
   CMyDialog:: CMyDialog(CWnd* pParent /*NULL*/ )
           : CDialog (CMyDialog::IDD, pParent)
   {
           // initialization code here
   }
   ```

   Caso o identificador que represente seu diálogo tenha que ser alterado, não se esqueça de alterar o valor do novo identificador dentro da descrição da classe.

3. Utilize o ClassWizard para adicionar os membros de dados do diálogo bem como suas funções de exchange e validação.
4. Utilize o ClassWizard para mapear as mensagens cos controles para funções membros da nova classe.
5. Escreva o código necessário para inicializações especiais dos controles (em OnInitDialog) e os códigos necessários ao processamento das mensagens.
6. Escreva o código necessário a chamada do diálogo dentro de sua classe vies. Este código corresponde a uma chamada a função membros da classe CDialog, DoModal. O retorno dessa função só é feito após o usuário fechar a caixa de diálogo.

A seguir encontra-se um exemplo real passo a passo para a criação de um diálogo em um programa baseado em Windows.

4 Exemplo Curso5A

O diálogo criado para o exemplo Curso5A.exe, não é apenas uma caixa de diálogo e sim um enorme diálogo que inclui um controle de cada tipo. Essa tarefa se torna mais fácil utilizando o editor de diálogos do Developer Studio. O resultado final está mostrado na Figura 24.

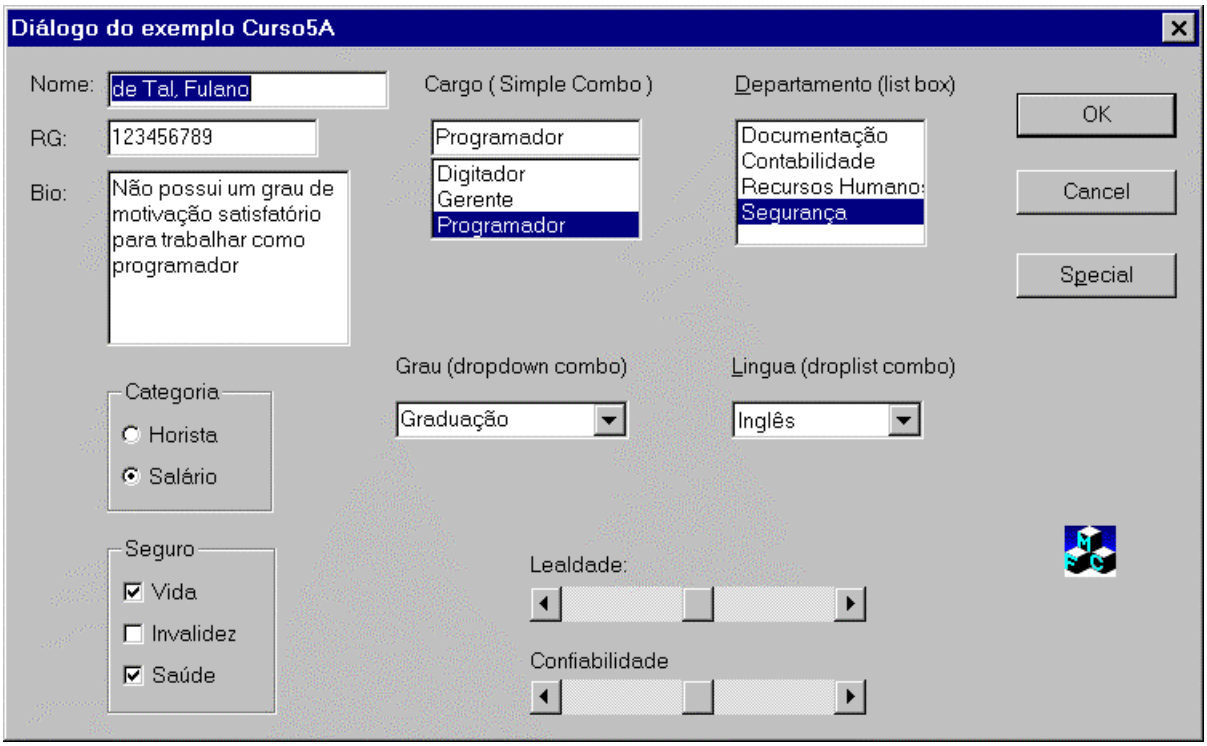


Figura 24 – Diálogo integrante do exemplo Curso5A.exe

4.1 Construindo um recurso Diálogo

Aqui estão os passos para a criação de um recurso Diálogo

4.1.1 Passo1: Gerar o projeto Curso5A com o AppWizard

Escolha a opção *File New* e selecione *MFC AppWizard (EXE)* na pasta *Project*. Selecione a opção *Single Document* (SDI) e aceite todas outras opções padrões.

As opções de criação e o nomes das classes a serem criadas está mostrada na figura abaixo.

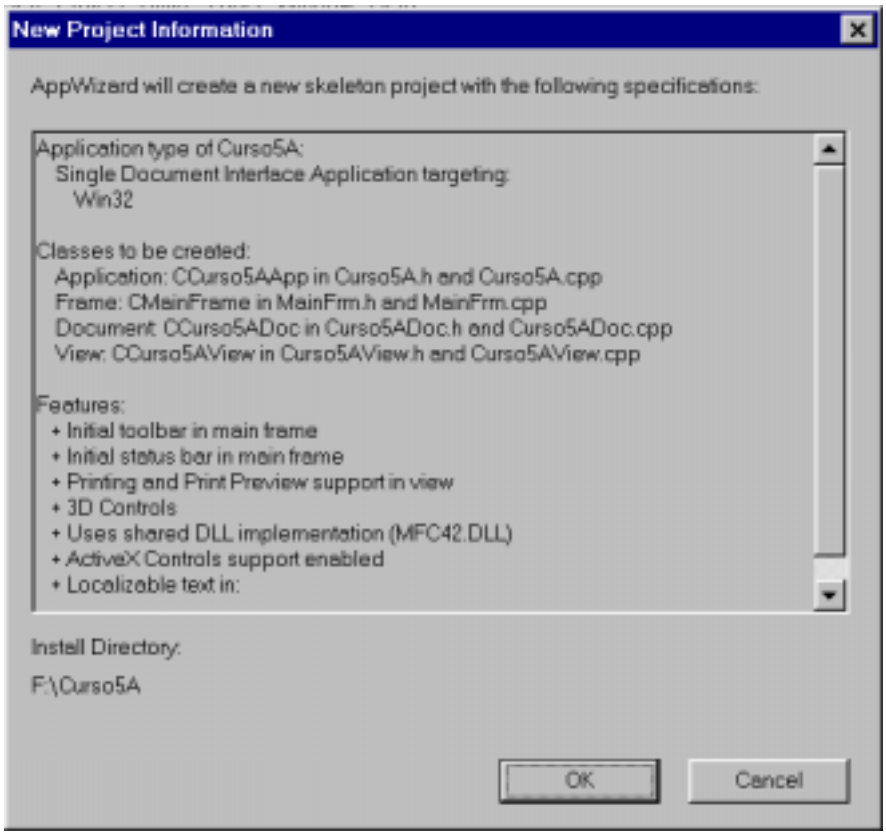


Figura 25 – Opções para o projeto Curso5A

4.1.2 Passo2: Crie um novo recurso diálogo com o identificador IDD_DIALOG1

Selecione a opção Resource Ctrl+R do menu Insert do Developer Studio. O diálogo Insert Resource (Figura 26) será exibido na tela. Selecione a opção Dialog e pressione o botão New.

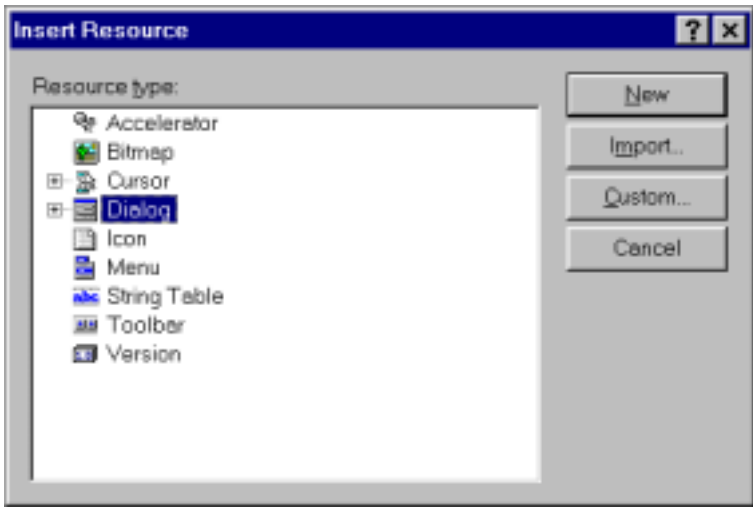


Figura 26 – Dialogo de seleção de inserção de recurso

Na tela do Developer Studio será exibido o novo diálogo que acabou de ser criado como mostrado na Figura 27.

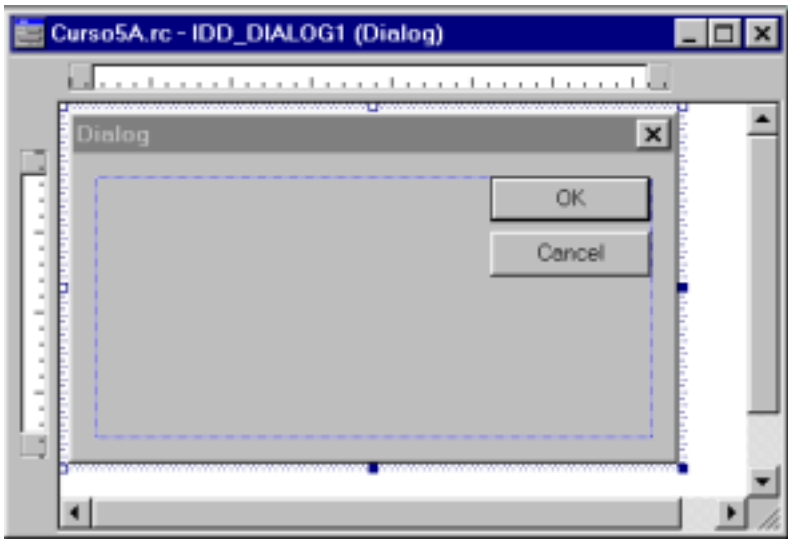


Figura 27 – Tela do Editor de Recurso com o novo diálogo criado

O identificador IDD_DIALOG1 foi associado a esse recurso. Note também que o diálogo já possui um botão Ok e Cancel.

4.1.3 Passo3: Redimensione o diálogo e troque seu título

Altere o tamanho do diálogo até que ele fique com 370x215 DLU, este valor pode ser observado na barra de status.

Para alterar seu título será necessário mostrar sua janela de propriedades. Isto pode ser feito pressionando-se o botão direito do mouse sobre a caixa de diálogo (certifique-se que não está sob nenhum controle) e selecionando a opção Properties no menu que aparecerá. A caixa de edição Capition será utilizada para atribuir um novo título a caixa de diálogo.

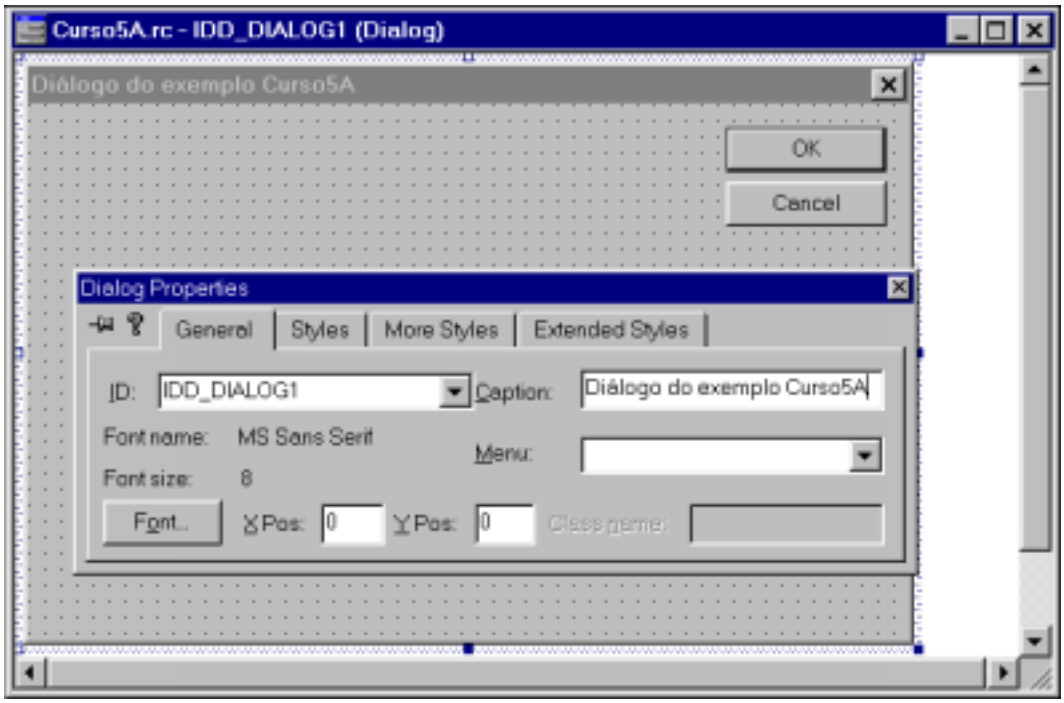


Figura 28 – Diálogo redimensionado com sua caixa de propriedades

Obs: para mater a janela de propriedades sempre visível, fixe-a utilizando o botão existente em seu canto superior esquerdo.

4.1.4 Passo4: Altere o estilo do diálogo

Selecione a pasta *Styles* no diálogo de propriedades, e altere o estilo do diálogo como mostrado na Figura 29.

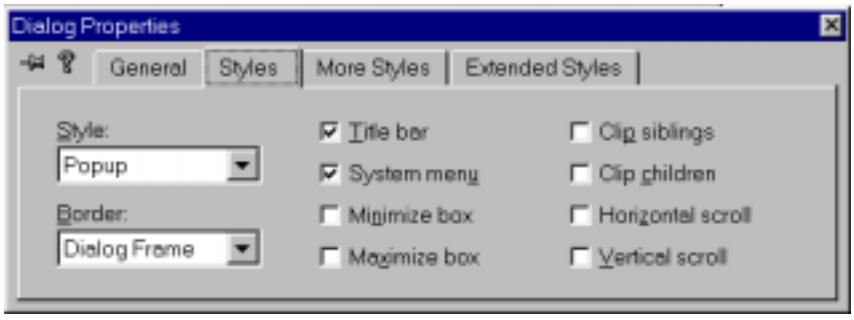


Figura 29 – Pasta de estilos do diálogo

4.1.5 Passo5: Altere os estilos adicionais do diálogo

Selecione a pasta *More Styles* no diálogo de propriedades, e altere o estilo do diálogo como mostrado na Figura 30.

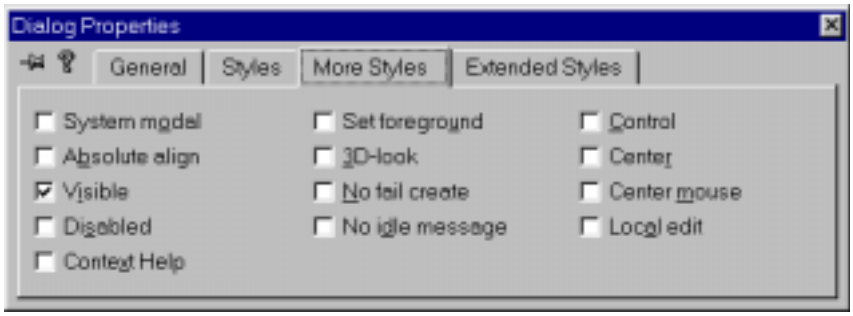


Figura 30 - Pasta de estilos adicionais do diálogo

4.1.6 Passo6: Adicione os controles ao diálogo

Utilize a barra de controle (Control palette) para adicionar cada um dos controles. (Se esta barra não estiver visível, clique o botão direito do mouse sobre uma barra de ferramentas e ative a opção *Controls*).

Arraste os controles para as posições e tamanhos como mostrado na Figura 24.

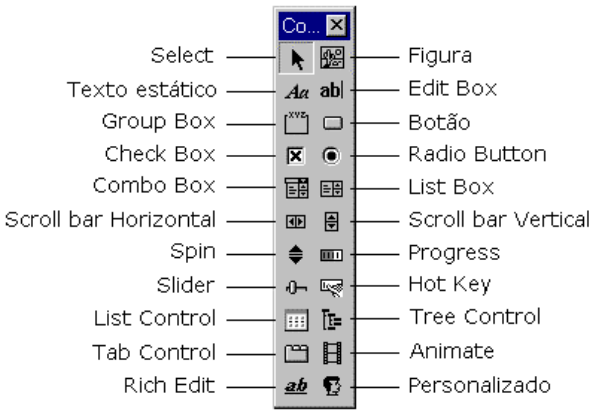


Figura 31 – Barra de Controles

A seguir encontra-se uma breve descrição de cada um dos controles que serão adicionados ao novo diálogo.

- **Controle *Texto estático* para o campo Nome**. Um controle texto estático, simplesmente coloca caracteres na tela. Nenhuma interação com o usuário ocorre neste tipo de controle. O texto que será exibido nesse controle pode ser alterado na caixa de edição Capition das propriedades desse controle. Para inserir um texto estático no diálogo basta selecionar o botão apropriado na barra de controle e definir sua posição e tamanho dentro do diálogo utilizando o mouse. Nenhum outro controle estático será descrito adiante pois todos são iguais para o diálogo, possuem o mesmo ID e são adicionados da mesma forma, o que não causa problema ao programa pois ele não terá que acessar nenhum deles.
- **Controle Edit Box para o Nome**. Um edit control é a meio mais fácil de se entrar com um texto dentro de um diálogo. Pressione o botão direito do mouse sobre esse controle e selecione a opção Properties. Altere o identificador do controle de IDC_EDIT1 para IDC_NOME. Aceite todas as outras opções padrões.
- **Controle Edit Box para o RG**. Este controle é adicionado como o anterior exceto que seu identificador tem que ser alterado para IDC_RG. Mais tarde esse identificador será utilizado pelo ClassWizard para adicionar um campo numérico.
- **Controle Edit Box para a Bio (biografia)**. Este é um controle que deve possuir mais de uma linha (multiline). Altere seu identificador para IDC_BIO, e selecione suas propriedades como mostrado na Figura 32.

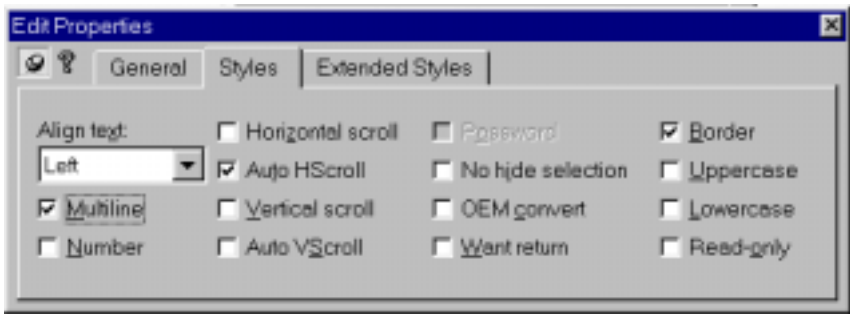


Figura 32 – Propriedades de um edit box multiline

- **Controle Group Box para a categoria**. Este é um controle serve apenas para separar visualmente os radio buttons das categorias. Altere seu Capition para *Categoria* e aceite o identificador padrão.
- **Os Radio Buttons para o Horista e Salário**. Posicione estes radio buttons dentro do group box categoria. Altere o identificador do botão Horista para IDC_CAT e selecione as outras propriedades como mostrado na figura abaixo:

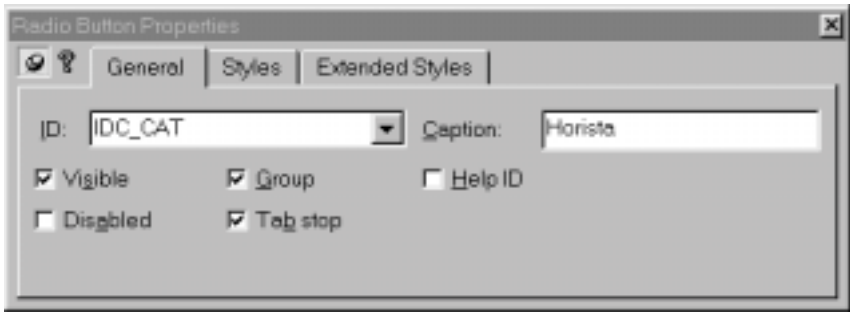


Figura 33 – Propriedades para o radio button Horista

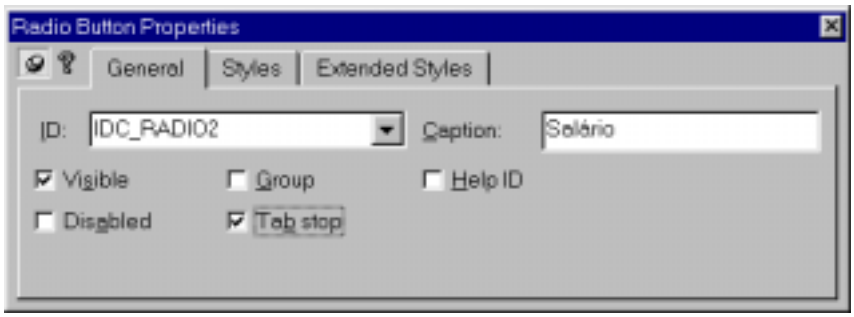


Figura 34 – Propriedades para o radio button Salário

Certifique-se que a propriedade Auto está setada (padrão) na pasta Styles, e apenas o botão horista está com a propriedade Group setada. Quando estas propriedades estão corretamente configuradas o Windows garante que apenas um estará selecionado ao mesmo tempo. O Group box Categoria não tem nenhum efeito sob a operação desses botões.

- **Controle Group Box para o Seguro**. Este é um controle serve apenas para separar visualmente check box dos tipos de seguro contratados. Altere seu Capition para *Seguro* e aceite o identificador padrão.

  Nota: Mais tarde quando for alterar a ordem de tabulação do diálogo, certifique-se que este controle será o posterior ao radio button do grupo das categorias. Altere a propriedade desse controle para Group de modo que identifique para o Windows o fim do grupo das categorias. Caso isto não seja feito, varias mensagens de Warning serão exibidas na janela Output quando o programa estiver rodando em modo Debug.

- **Os check Box para Vida, Invalidez e Saúde**. Posicione estes controles dentro do Group box Seguro. Aceite todos os estilos padrões, porém altere os identificadores para respectivamente, IDC_VIDA, IDC_INVALID, IDC_SAUDE.
- **Controle Combo Box para o Cargo**. Este é primeiro dos três tipos de combos existentes. Altere seu identificador para IDC_CARGO, selecione a pasta estilo e escolha o tipo Simple. Click na pasta Data e adicione os tres cargos como mostrado na figura abaixo (Obs: cada final de linha no list box deve ser informado a caixa de propriedades utilizando-se a combinação de teclas Ctrl+Enter). Com o tipo Simple, um texto pode ser informado digitando-se um novo valor bem como selecionado um pré-existente na lista.

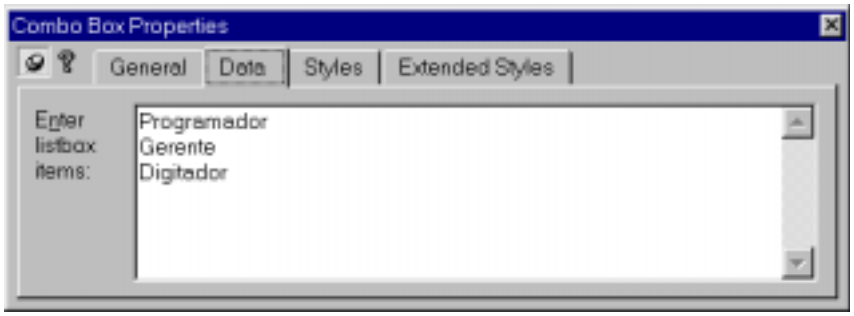


Figura 35 – Dados que constarão na lista de seleção do combo

- **Controle Combo Box para o Grau**. Altere seu identificador para IDC_GRAU, e aceite todos outros estilos como o padrão. Click na pasta Data e adicione os tres níveis de instrução: Primário, Colegial, Graduação. No estio DropDown, o usuário

pode digitar um texto no edit box ou selecioná-lo da lista, pressionando-se a seta que aparece no canto direito do controle.

- **Controle List Box para o Departamento**. Altere seu identificador para IDC_DEPT, aceite todos outras propriedades como padrões. Em um list box o usuário pode selecionar apenas um dos elementos utilizando o mouse, as teclas UP e DOWN ou pressionando o caractere correspondente a primeira letra da opção desejada. Note que utilizando o editor de diálogo não é possível adicionar nenhuma opção inicial a um list box, isto será visto mais tarde no exemplo.
- **Controle Combo Box para a Língua**. Altere seu identificador para IDC_LANG, selecione a pasta Styles e escolha o tipo DropList. Adicione tres entradas de dados (Ingles, Frances, Espanhol) na pasta Data. Este tipo de combo permite que o usuário apenas selecione um item da lista, não é possível editar um novo valor que não esteja na lista. A seleção pode ser feita utilizando-se o mouse ou o teclado.
- **Controle Barra de Scroll para a Leadade e Confiabilidade**. Não confunda um controle barra de scroll com uma barra de rolamento de uma janela. São criados e dimensionados em tempo de projeto, não de execução. Altere seus identificadores para respectivamente IDC_LEALDADE, IDC_CONFIABILIDADE.
- **O Botões Ok, Cancel e Special**. Verifique se os Capition dos botões são respectivamente Ok, Cancel e Special. Altere o identificador do botão Special para IDC_SPECIAL.
- **Icone**. Qualquer ícone ou bitmap pode ser exibido dentro de um diálogo utilizando-se este controle (Picture).

4.1.7 Passo7: Verifique a ordem de tabulação da caixa de diálogo

Escolha a opção Tab Order do menu Layout do editor de diálogo. Utilize o mouse para altera a ordem de tabulação como a mostrada na Figura 36. Clique em cada controle na ordem abaixo e ao final pressione Enter.

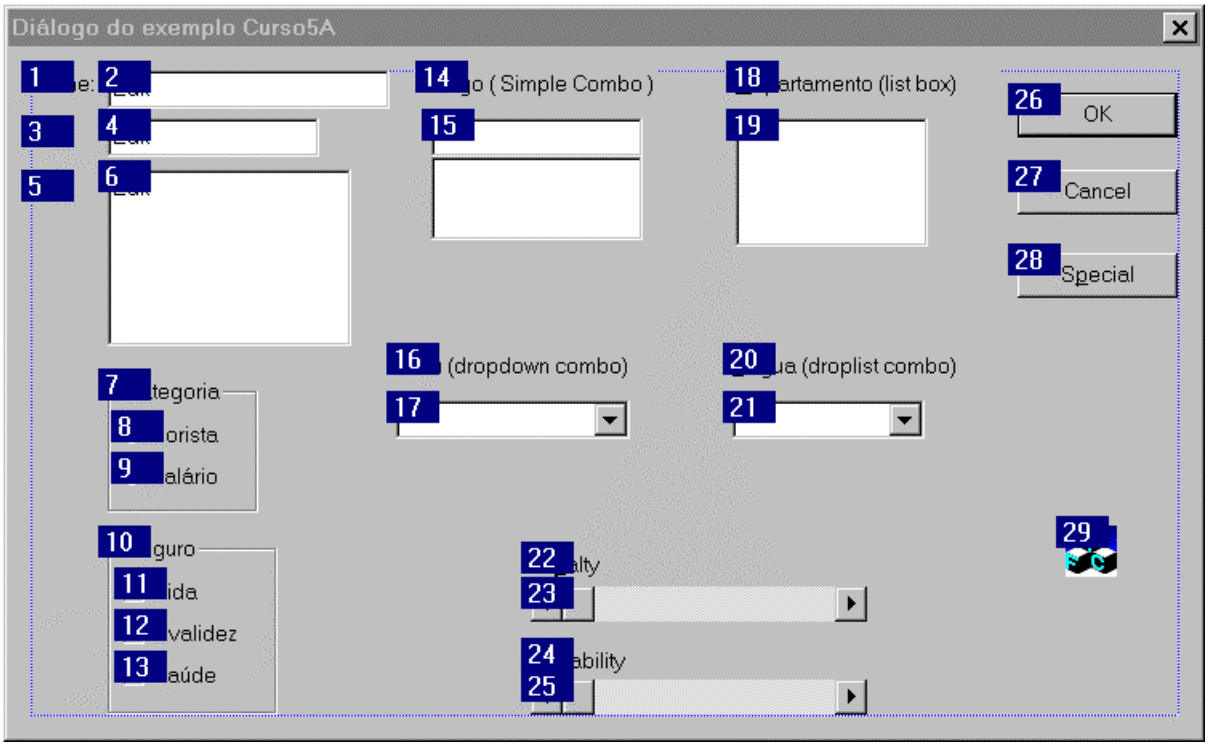


Figura 36 – Ordem de tabulação para o diálogo

4.1.8 Passo8: Grave o arquivo de recursos no disco

Para maior segurança selecione a opção Save do menu File para gravar as alterações para o arquivo curso5A.rc. Mantenha o editor de diálogo funcionando e visível na tela.

4.2 ClassWizard e a classe Diálogo

Um recurso de diálogo não pode ser utilizado pelo programa até que seja criada uma classe correspondente a ele.

O ClassWizard será utilizado em conjunto com o editor de diálogo para criar uma classe relacionada com esse recurso, como será visto nos passos a seguir:

4.2.1 Passo1: Selecione ClassWizard no menu View do Developer Studio

Mantenha o editor de diálogo funcionando e visível na tela, com o novo diálogo criado selecionado (IDD_DIALOG1). Pressione o botão direito do mouse sobre o diálogo e selecione a opção ClassWizard no menu popup.

4.2.2 Passo2: Adicione a classe CCurso5ADlg

O ClassWizard detecta o fato de o diálogo criado ainda não estar associado a uma classe do C++, e exibe o diálogo a seguir perguntando se você deseja criar uma nova classe para esse recurso.

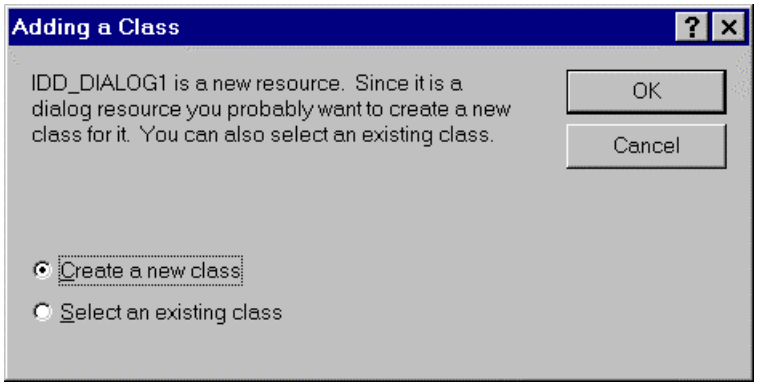


Figura 37 – Diálogo de adição de nova Classe

Aceite a opção padrão Create a new class, e pressione Ok. Preencha os dados para a nova classe como mostrado na figura a seguir.

Coloque o nome da classe que seja relacionado com um diálogo. Por exemplo as ultimas três letras do nome da classe como ***Dlg***. E pressione Ok.

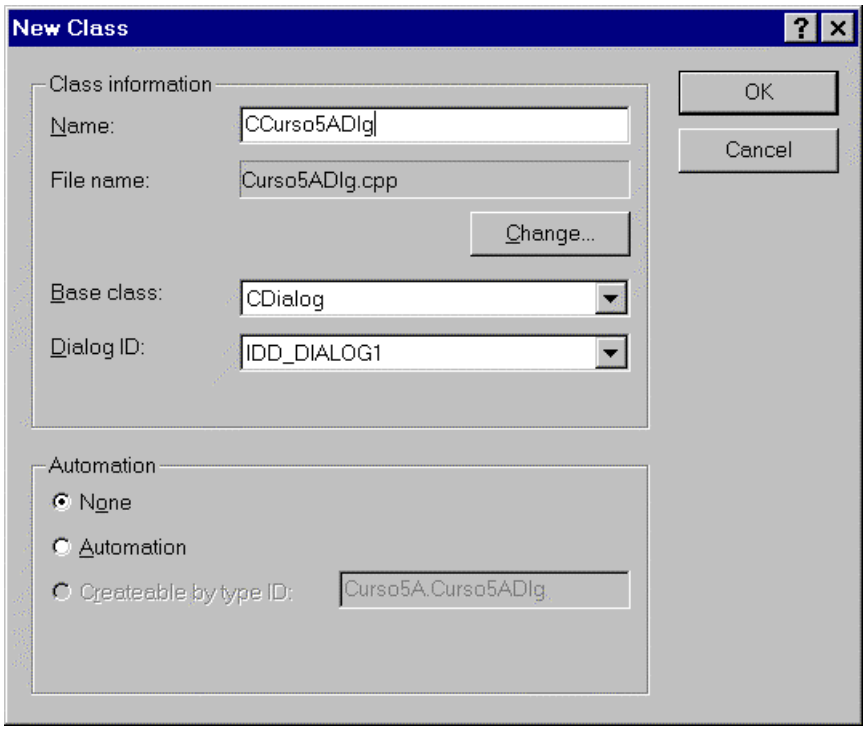


Figura 38 – Tela de criação da nova classe

4.2.3 Passo3: Adicione as variáveis membros da classe CCurso5ADlg

Após a criação da classe CCurso5ADlg, o diálogo do ClassWizard aparecerá na tela. Selecione a pasta Member Variables. O diálogo passa a Ter a forma da Figura 39.

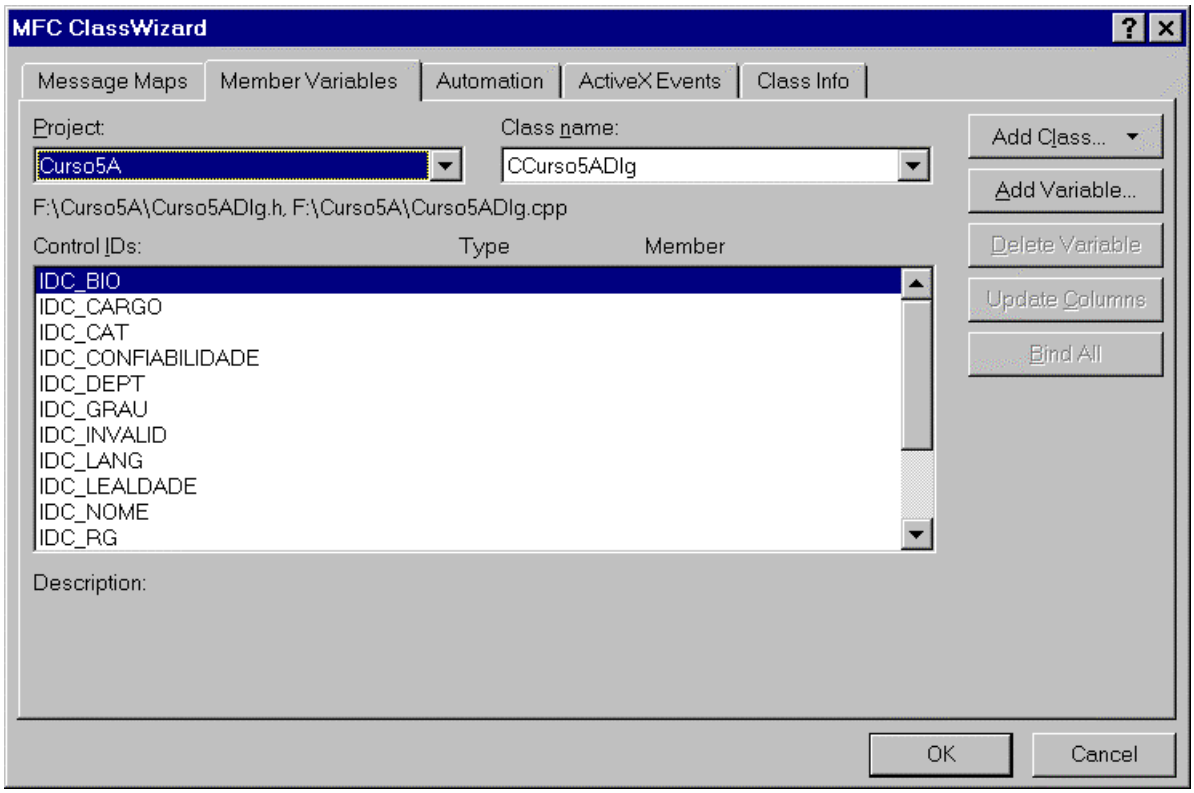


Figura 39 – Pasta Member Variables do ClassWizard

É necessário associar membros de dados a cada um dos controles do diálogo. Para isto, selecione um identificador correspondente a um controle e pressione o botão Add Variable. O diálogo Add Member Variable será exibido como mostrado na Figura 40:

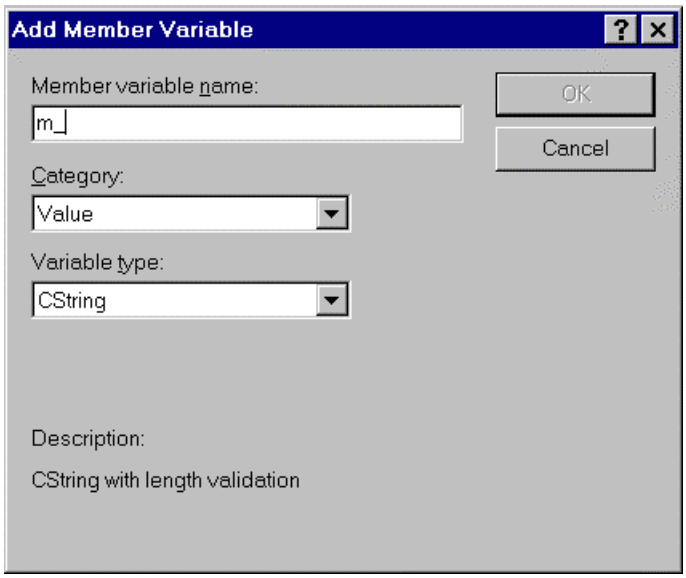


Figura 40 – Diálogo Add Member Variable

Digite o nome da variável e escolha seu tipo de acordo com a tabela abaixo. Certifique-se que os nomes das variáves são exatamente como os mostrados na tabela abaixo, pois maiusculas e minusculas afetam os nomes de variáveis em C++. Pressione Ok e repita a operação para todos os outros controles.

ID do controle	Membro de dados	Tipo
IDC_BIO	m_strBio	CString
IDC_NOME	m_strNome	CString
IDC_RG	m_nRg	int
IDC_CARGO	m_strCargo	CString
IDC_CAT	m_nCat	int
IDC_CONFIABILIDADE	m_nConfiabilidade	int
IDC_LEALDADE	m_nLealdade	int
IDC_LANG	m_strLingua	CString
IDC_SAUDE	m_bSegSaude	BOOL
IDC_VIDA	m_bSegVida	BOOL
IDC_INVALID	m_bSegInvalid	BOOL
IDC_GRAU	m_strGrau	CString
IDC_DEPT	m_strDepart	CString

Selecionando as variáveis criadas no diálogo do ClassWizard, alguns edit box podem aparecer na parte inferior da janela onde podem ser especificados as restrições para aquele tipo de variável. Por exemplo se uma variável do tipo CString for selecionada aparecerá um edit box onde pode ser especificado o máximo numero de caracteres que pode conter esta string.

4.2.4 Passo4: Adicione uma função membro para o botão Special

Selecione a pasta Message Maps do ClassWizard. Na lista Object ID's contem uma entrada para o IDC_SPECIAL. Selecione esta entrada e em seguida na lista de mensagens

BN_CLICKED, e pressione o botão Add Function, O diálogo Add Member Function aparecerá sobre o ClassWizard como mostrado na figura abaixo:

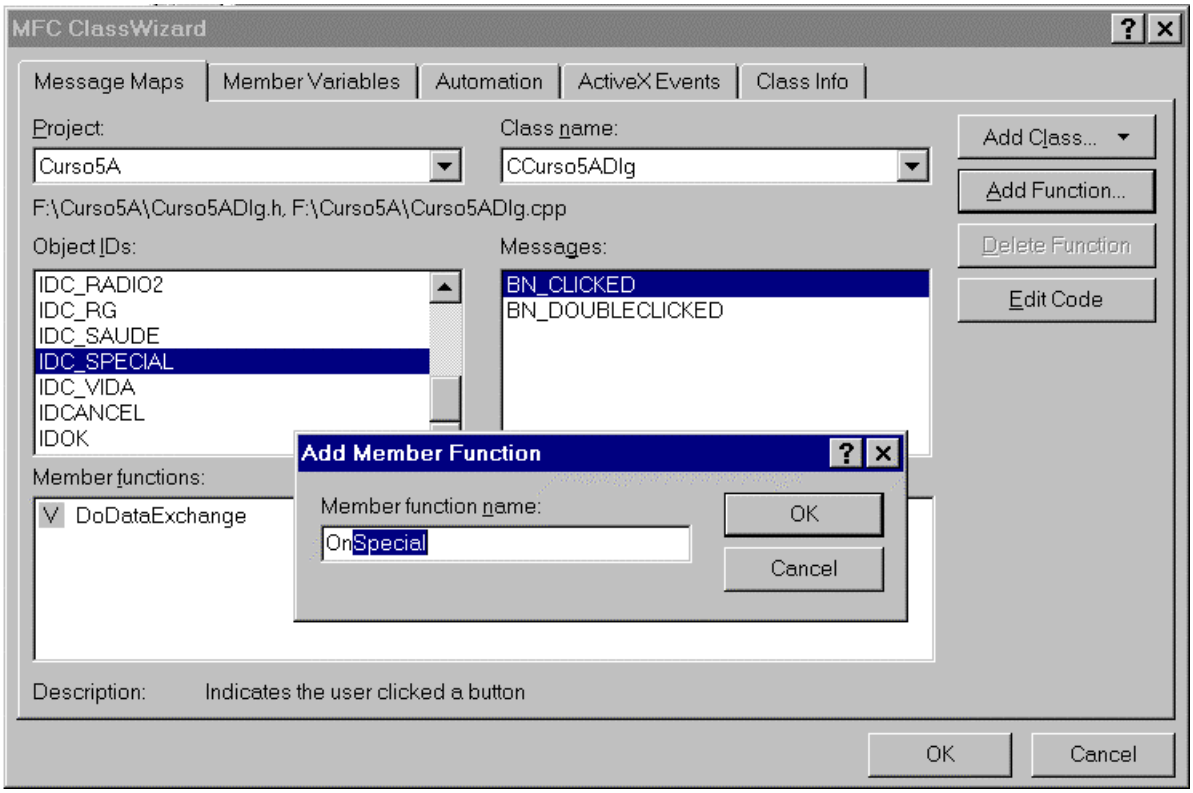


Figura 41 – Diálogo Add Member Function sobre o ClassWizard

Um nome pode ser digitado para a função, porém o ClassWizard sempre sugere um nome que é significativo ao evento e quase sempre será aceito sem alterações.

Após pressionar OK, a função passou a ser mapeada para a classe e o botão Edit Code pode ser usado para a edição do corpo da função, que deverá conter as seguintes inscrições:

```
void CCurso5ADlg::OnSpecial()
{
        TRACE("CCurso5ADlg::OnSpecial\n");
}
```

Obs: outros membros, como por exemplo a função que responderá ao botão ok já estão mapeadas pela classe mãe CDialog, portanto não é necessário que sejam adicionadas funções para esses membros.

4.2.5 Passo5: Utilize o ClassWizard para adicionar a função OnInitDialog

Até o presente momento o ClassWizard se encarregou de gerar o código para inicializar os controles do diálogo. Porém as funções DDX (Dialog Data Exchange) não inicializam as escolhas para o list box, entretanto é possível sobrecarregar a função virtual CDialog::OnInitDialog.

O ClassWizard pode ser utilizado para inserir o manipulador da mensagem WM_INITDIALOG ao código fonte da classe. Para isto selecione CCurso5ADlg no list box Object ID's e realize um duplo clique sobre a mensagem WM_INITDIALOG que aparecerá no list box Messages. Utilize o botão Edit Code para deixar a implementação da função OnInitDialog como mostrada abaixo:

```
BOOL CCurso5ADlg::OnInitDialog()
{
        CListBox* pLB = (CListBox*) GetDlgItem(IDC_DEPT);
        pLB->InsertString(-1, "Documentação");
        pLB->InsertString(-1, "Contabilidade");
        pLB->InsertString(-1, "Recursos Humanos");
        pLB->InsertString(-1, "Segurança");

        // Chame a função após a inicialização
        return CDialog::OnInitDialog();
}
```

Uma inicialização para os combo boxes pode ser feita de forma semelhante ao invés de preenche-los no momento de sua criação.

4.3 Conectando o diálogo a uma view

O recurso e o código fonte para manipular o diálogo já estão prontos, falta porém agora conectá-lo a uma view. Em um aplicativo real, provavelmente esta conexão se dará através de uma escolha de um item de menu. Entretanto neste exemplo a conexão será feita utilizando-se a mensagem WM_LBUTTONDOWN para inicializar o diálogo. Os passos para isto estão listados abaixo:

4.3.1 Passo1: Selecione a classe CCurso5AView no ClassWizard

Neste ponto é importante notar que o exemplo Curso5A seja o projeto atual do Developer Studio.

4.3.2 Passo2: Utilize o ClassWizard para adicionar a função membro OnLButtonDown.

Isto já foi feito em exemplos anteriores. Basta selecionar a classe CCurso5AView, selecione o nome da classe CCurso5AView no list box Object IDs, e realize um duplo clique na mensagem WM_LBUTTONDOWN que aparecerá no list box Messages.

4.3.3 Passo3: Escreva o código fonte para a função OnLButtonDown no arquivo curso5Adlg.cpp

Adicione o código em destaque abaixo no corpo da função OnLButtonDown. Uma boa parte do código consiste de funções TRACE que são utilizadas para exibir o conteúdo do diálogo após o mesmo ser fechado.

A chamada do construtor da classe CCurso5ADlg e a posterior chamada do método DoModal são de especial interesse nesse momento.

```
void CCurso5AView::OnLButtonDown(UINT nFlags, CPoint point)
{
        CCurso5ADlg dlg;

        dlg.m_strNome   = "de Tal, Fulano";
        dlg.m_nRg               = 123456789;
        dlg.m_nCat              = 1;  // 0 = horista, 1 = salario
        dlg.m_strBio    = "Não possui um grau de motivação satisfatório
                           para trabalhar como programador";
        dlg.m_bSegVida  = TRUE;
        dlg.m_bSegInvalid  = FALSE;
```

```
        dlg.m_bSegSaude = TRUE;
        dlg.m_strDepart = "Segurança";
        dlg.m_strCargo  = "Programador";
        dlg.m_strLingua = "Inglês";
        dlg.m_strGrau   = "Graduação";
        dlg.m_nConfiabilidade = dlg.m_nLealdade = 50;

        int ret = dlg.DoModal();

        TRACE("DoModal return = %d\n", ret);
        TRACE("Nome = %s, RG = %d, Cat = %d\n",
              dlg.m_strNome, dlg.m_nRg, dlg.m_nCat);

        TRACE("Dept = %s, Cargo = %s, Lingua = %s, Grau = %s\n",
           dlg.m_strDepart, dlg.m_strCargo, dlg.m_strLingua, dlg.m_strGrau);

        TRACE("Vida = %d, Invalid = %d, Saude = %d, bio = %s\n",
           dlg.m_bSegVida, dlg.m_bSegInvalid, dlg.m_bSegSaude, dlg.m_strBio);

        TRACE("Confiabilidade = %d, Lealdade = %d\n",
              dlg.m_nConfiabilidade, dlg.m_nLealdade);
}
```

4.3.4 Passo4: Adicione o código a função membro virtual OnDraw no arquivo Curso5AView.cpp

Para informar ao usuário que ele deve pressionar o botão esquerdo do mouse sobre a janela o seguinte código deve ser adicionado ao método OnDraw:

```
void CCurso5AView::OnDraw(CDC* pDC)
{
        pDC->TextOut(0,0, "Pressione o botão esquerdo do mouse aqui");
}
```

4.3.5 Passo5: Adicione o include da classe do diálogo ao arquivo Curso5AView.cpp

A função membro OnLButtonDown depende da declaração da classe CCurso5ADlg em seu conteúdo. Para isso é necessário que o include seja adicionado ao arquivo Curso5AView.cpp em seu início, logo após o include da própria descrição da classe view.

O código a ser inserido é como o mostrado abaixo:

```
// deve ser incluido para a classe do dialogo poder ser reconhecida dentro da view
#include "Curso5ADlg.h"
//
```

4.3.6 Passo6: Compile e execute o aplicativo

Se todos os passos até agora foram realizados com sucesso, compile e execute o programa Curso5A.exe utilizando a opção Go do menu Debug, para que as funções TRACE informem na janela Output o estado do aplicativo.

4.4 Entendendo o Aplicativo Curso5A

Quando o programa chama o método DoModal, o controle é retornado ao programa somente após o usuário fechar o diálogo. Se compreender isto compreenderá os diálogos modais.

Uma seqüência de funções são chamadas, em conseqüência da chamada de DoModal. Isto é resumido abaixo:

```
CDialog::DoModal
     CCurso5ADlg::OnInitDialog
          ... inicializações adicionis...
          CDialog::OnInitDialog
               CWnd::UpdateDate(FALSE)
                    CCurso5ADlg::DoDataExchange
     Usuario entra com dados...
     Usuario pressiona botão Ok
     CCurso5ADlg::OnOk
          ... validações adicionais...
          CDialog::OnOk
               CWnd::UpdateDate(TRUE)
                    CCurso5ADlg::DoDataExchange
          CDialog::EndDialog(IDOK)
```

As funções virtuais OnInitDialog e DoDataExchange são sobrecarregadas dentro da classe do diálogo para realizar as tarefas específicas do diálogo. A listagem da função DoDataExchange está exibida a seguir:

```
void CCurso5ADlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CCurso5ADlg)
        DDX_Text(pDX, IDC_BIO, m_strBio);
        DDX_Radio(pDX, IDC_CAT, m_nCat);
        DDX_LBString(pDX, IDC_DEPT, m_strDepart);
        DDX_Check(pDX, IDC_SAUDE, m_bSegSaude);
        DDX_Check(pDX, IDC_VIDA, m_bSegVida);
        DDX_Check(pDX, IDC_INVALID, m_bSegInvalid);
        DDX_Text(pDX, IDC_NOME, m_strNome);
        DDX_Text(pDX, IDC_RG, m_nRg);
        DDX_CBString(pDX, IDC_CARGO, m_strCargo);
        DDX_Scroll(pDX, IDC_CONFIABILIDADE, m_nConfiabilidade);
        DDX_Scroll(pDX, IDC_LEALDADE, m_nLealdade);
        DDX_CBString(pDX, IDC_LANG, m_strLingua);
        DDX_CBString(pDX, IDC_GRAU, m_strGrau);
        //}}AFX_DATA_MAP
}
```

As funções DoDataExchange, DDX_ e DDV_ são bidirecionais. Se o método UpdateData é chamado com o parâmetro FALSE, essas funções transferem o conteúdo das variáveis membros para os controles do diálogo. Se o parâmetro for TRUE, ocorre o processo inverso, ou seja essas funções transferem o conteúdo dos controles para as variáveis membros.

A função EndDialog é critica para o procedimento de saída do diálogo. DoModal retorna o parâmetro passado por EndDialog IDOK, para aceitar os dados do diálogo e IDCANCEL para desconsiderar as alterações feitas no diálogo.

4.5 Manipulando os controles barras de Scroll

Um controle Scroll Bar possui um posição atual e uma faixa de valores. Por exemplo se a faixa de valores for de (0,100), o valor 50 (posição atual), corresponderia ao centro da faixa. A função que define a faixa de valores de um scrollbar é a CScrollBar::SetScrollRange e a função utilizada para definir a posição atual é CScrollBar::SetScrollPos. As barras de Scroll, enviam as mensagens WM_HSCROLL e WM_VSCROLL para o diálogo quando o usuário realiza alguma operação com o scroll bar. O diálogo tem que manipular estas mensagens para manter a posição do scroll atualizada com sua variável membro.

A seguir encontram-se os passos para manipular barras de scroll em um diálogo:

4.5.1 Passo1: Adicione as declarações de enum para as faixas de scroll de mínimo e máximo.

No arquivo Curso5ADlg.h, adicione as seguintes linhas no início da descrição da classe.

```
enum { nMin = 0 };
enum { nMax = 100 };
```

4.5.2 Passo2: Edite a função OnInitDialog para inicializar a faixa do scroll.

Adicione o seguinte código fonte a função OnInitDialog

```
CScrollBar* pSB = (CScrollBar*) GetDlgItem(IDC_LEALDADE);
pSB->SetScrollRange(nMin, nMax);

pSB = (CScrollBar*) GetDlgItem(IDC_CONFIABILIDADE);
pSB->SetScrollRange(nMin, nMax);
```

4.5.3 Passo3: Utilize o ClassWizard para mapear a função que manipula as mensagens do Scroll

Escolha a mensagem WM_HSCROLL, e adicione uma função membro OnHScroll e digite o seguinte código em destaque:

```
void CCurso5ADlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        int nTemp1, nTemp2;

        nTemp1 = pScrollBar->GetScrollPos();
        switch(nSBCode) {
        case SB_THUMBPOSITION:
                pScrollBar->SetScrollPos(nPos);
                break;
        case SB_LINELEFT: // left arrow button
                nTemp2 = (nMax - nMin) / 10;
                if ((nTemp1 - nTemp2) > nMin) {
                        nTemp1 -= nTemp2;
                }
                else {
                        nTemp1 = nMin;
                }
                pScrollBar->SetScrollPos(nTemp1);
                break;
        case SB_LINERIGHT: // right arrow button
                nTemp2 = (nMax - nMin) / 10;
```

```
                if ((nTemp1 + nTemp2) < nMax) {
                        nTemp1 += nTemp2;
                }
                else {
                        nTemp1 = nMax;
                }
                pScrollBar->SetScrollPos(nTemp1);
                break;
        }
}
```

4.5.4 Passo4: Compile e execute o aplicativo

Agora parte das entradas do usuário são manipuladas ao trabalhar com as barras de scroll.

5 Desenhando em uma caixa de diálogo

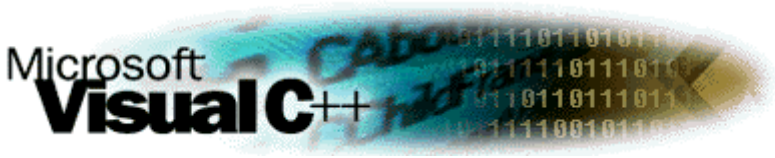
É possível desenhar diretamente dentro da área cliente de uma janela de dialogo. Porém isso deve ser feito com cuidado para não sobrescrever os controles que ali estão presentes.

Se for desejável apenas exibir um texto, o meio mais fácil de se fazer isso seria incluindo um controle do tipo Texto Estático, sem Capition e com um ID único (não serve IDC_STATIC) e enviar a mensagem CWnd::SetDlgItemText na função membro OnInitDialog para adicionar o texto dinamicamente ao diálogo.

Exibir gráficos é uma tarefa um pouco mais complicada. É necessário utilizar o ClassWizard para sobrecarregar a função OnPaint. Dentro desta função é transformado um controle estático em um ponteiro para CWnd, o qual é possível extrair seu contexto para poder realizar o desenho. Esta técnica é utilizada para desenhar dentro de um controle prevenindo que o Windows apague seu trabalho mais tarde. As funções Invalidade/UpdateWindow são utilizada para informar que seu controle foi alterado e precisa ser redesenhado.

A seguir encontra-se um código para a função OnPaint que desenhar um retângulo preto no interior de um controle estatico.

```
void CMyDialog::OnPaint()
{
        CWnd* pWnd = GetDlgItem(IDC_STATIC1);   // IDC_STATIC1 deve ser criado
                                                // utilizando-se o ClassWizard
        CDC* pControlDC = pWnd->GetDC();

        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SelectStockObject(BLACK_BRUSH);
        pControlDC->Rectangle(0,0,10,10);
        pWnd->ReleaseDC(pControlDC);
}
```



Capítulo 6
Separando o Documento de sua Vista

Neste capítulo será discutido como os Documentos (doc) mantêm os dados do aplicativo e como as vistas (view) apresentam esses dados ao usuário. Além de aprender como essas classes interagem entre si enquanto o aplicativo está em execução.

Ainda neste capítulo serão mostrados dois exemplos que utilizam a classe CFormView como base para suas classes de vista. No primeiro exemplo a classe doc mantém os dados de apenas um objeto do tipo CEstudante que representa um único registro do tipo estudante e a classe view oferece meios para a alteração desses dados. A classe CEstudante servirá como de exemplo para a implementação de classes que representem elementos do mundo real dentro de um programa. O segundo exemplo introduz o conceito de ponteiro para coleções, através das classes CObList e CTypePtrList. Agora o documento será capaz de armazenar uma coleção de registros do tipo estudante, e a classe view permitirá o acesso individual a cada um desses registros.

1 Funções de interação entre Document-View

Já se sabe que as classes doc's armazenam os objetos de dados que serão mostrados e editados pelos usuários através de uma classe view. Um complexo processo de handshaking ocorre entre as classes doc e view e o resto da estrutura das aplicações. Para entender este processo é necessário que cinco funções membros sejam explicadas.

1.1 A função CView::GetDocument

Todo objeto de uma classe view possui um e somente um objeto documento associado a ele. Suponha que um usuário digite um novo dado em uma caixa de texto contida dentro de uma view. Essa classe view deve se encarregar de informar a classe doc que seus dados precisam ser atualizados e isto é feito alterando-se o conteúdo dos dados na classe doc através do uso de seu ponteiro. A função GetDocument retorna um ponteiro para o documento associado a essa classe, que pode ser utilizado para acessar as funções membros e variáveis membros públicas da classe doc.

Quando o AppWizard cria a classe derivadas de CView, cria uma versão especial da função GetDocument que retorna não um ponteiro para a classe CDocument mas sim um ponteiro para a classe derivada de CDocument que pertence a seu aplicativo, criando assim um tipo seguro que conhecerá todos os dados exclusivos de seu aplicativo.

1.2 A função CDocument::UpdateAllViews

Quando os dados de um documento são alterados por qualquer motivo, todas as classes view que estiverem associadas a esse documento terão que ser notificadas para que atualizem a representação dos dados. A função UpdateAllViews serve exatamente a esse propósito.

Se esta função for chamada a partir de uma classe derivada de CDocument o primeiro parâmetro, *pSender* necessariamente tem que ser NULL. Se ela for chamada de por uma função membro de uma classe view, o parâmetro *pSender* deve ser atribuído a janela que enviou o pedido dessa mensagem. Como mostrado no código a seguir:

```
GetDocument()->UpdateAllViews(this);
```

O parâmetro pSender não nulo, identifica para a estrutura das aplicações que a view atual não precisa ser notificada e assume que ela já contém os dados atualizados. Isto previne que uma atualização da view onde os dados foram alterados não seja realizada.

Essa função possui ainda parâmetros opcionais hint, que informam a janela como ela deverá ser atualizada. O uso desses parâmetros é dependente do aplicativo e constituem um uso avançado desse tipo de função.

Como a classe view é notificada quando a função UpdateAllViews é chamada é exatamente o assunto da próxima função membro a ser discutida, a OnUpdate.

1.3 A função CView::OnUpdate

Esta função virtual é chamada pela estrutura das aplicações em resposta a uma chamada da função membro CDocument::UpdateAllViews. Obviamente esta função também pode ser chamada diretamente por um membro da classe view. Tipicamente no conteúdo dessa função há acessos a classe doc e posterior atualização dos membros de dados ou controles para refletir as alterações no documento. Alternativamente essa função pode ser utilizada para invalidar regiões da view, fazendo com que a OnDraw se encarregue de fazer as atualizações necessárias no contexto.

Uma função OnUpdate, tipicamente possui o seguinte aspecto:

```
void CMyView::OnUpdate(CView* pSender; LPARAM lHint, CObject* pHint)
{
        CMyDocument* pDoc = GetDocument();
        CString nome = pDoc->GetNome();
        m_pstaticNome->SetWindowText(nome); // m_pstaticNome é membro da view
}
```

A implementação padrão da classe OnUpdate causa um invalidate em toda a área cliente da janela.

Se a função CDocument::UpdateAllViews é chamada com o parâmetro pSender contendo um ponteiro para um objeto view, a função OnUpdate será chamada para todas as views que estiverem associadas ao documento exceto a view que é indicada pelo pnteiro.

1.4 A função CView::OnInitialUpdate

Essa função virtual é chamada quando o aplicativo inicia, quando o usuário selecina a opção New do menu arquivo, ou quando o usuário seleciona a opção Open do menu Arquivo. A implementação da função OnInitialUpdate na classe base CView, contém apenas uma chamada a função OnUpdate. Se esta função for sobrecarregada dentro de uma classe derivada de CView, certifique-se que a função OnInitialUpdate da classe base é chamada ou simplesmente chame a função OnUpdate diretamente.

Esta função pode ser utilizada para inicializar o objeto view.

1.5 A função CDocument::OnNewDocument

A estrutura das aplicações chama esta função virtual quando um documento for criado ou quando o usuário escolher a opção New do menu Arquivo. Este é o lugar apropriado para inicializar os membros de dados das classes doc. Se esta função for sobrecarregada, lembre-se de chamar a função OnNewDocument da classe mãe.

2 Uma aplicação simples para Document-View

Suponha que o recurso de múltiplas views para o mesmo documento não seja necessário ao seu programa, mas é desejável utilizar a estrutura das aplicações para gerenciar o suporte a arquivos. Neste caso as funções UpdateAllViews e OnUpdate não serão utilizadas. Simplesmente siga os passos a seguir para desenvolver seu aplicativo.

1. Na descrição da classe doc, declare seus membros de dados. Esses membros de dados serão utilizados para o armazenamento primário de seu aplicativo.
2. Sobrecarregue a função virtual OnInitialUpdate na classe derivada de CView. A estrutura das aplicações chama esta função após os membros de dados do documento serem inicializados ou lidos do disco.
3. Na classe derivada de CView, permita que as mensagens de comando, mensagens do Windows e método OnDraw alterem diretamente os membros de dados da classe doc utilizando a função GetDocument para acessar o objeto documento.

3 A classe CFormView

A classe CFormView é uma classe view útil pois contém muitas das características das caixas de diálogos. Como uma classe derivada de CDialog, uma classe derivada de CFormView é associada a um recurso de diálogo que define as características de sua janela além de enumerar seus controles. A classe CFormView suporta as mesmas funções DDX e DDV discutidas no capítulo 5 para as caixas de diálogo.

> **OBS**: Quando o AppWizard cria um diálogo que será associado a uma FormView, ele altera corretamente as propriedades que esse diálogo deve obedecer. Se um recurso de diálogo for criado via o Editor de diálogos as seguintes propriedades devem ser respeitadas para o perfeito funcionamento desse recurso quando associado a uma FormView:
>
> Style = Child
> Border = None
> Visible = desselecionado

Um objeto do tipo CFormView recebe as notificações tanto dos controles como da estrutura das aplicações.

A classe CFormView é derivada da classe CView e não da classe CDialog, portanto as funções membros de CDialog OnInitDialog, OnOK e OnCancel não estão presentes nos objetos do tipo formview. Além da função UpdateData e DDX não serem chamadas automaticamente. Portanto essas funções devem ser manipuladas pelo programador no tempo certo em resposta as notificações dos controles e comandos.

O AppWizard pode ser utilizado para gerar um aplicativo cuja classe base para a view seja uma CFormView. Quando A CFormView for selecionada o AppWizard gera um recurso de diálogo com as propriedades já corretamente selecionadas. O próximo passo seria utilizar o ClassWizard para mapear as mensagem e adicionar os membros de dados com seus respectivos critérios de validação.

4 A classe CObject

Ao observar a hierarquia de classes da MFC, note que a classe CObject aparece bem no topo e muitas classes são derivadas dela, o que a considerar como a classe raiz. Quando uma classe é derivada de CObject ela recebe características importantes, Muitos desses benefícios serão mostrados em capítulos posteriores.

Neste capítulo serão abordadas as características de diagnóstico por esvaziamento e a opção de utilizar objetos desse tipo como elementos de coleções.

5 Diagnóstico por Esvaziamento (Dumping)

A MFC possui ferramentas úteis que permitem um diagnóstico por esvaziamento. Essas ferramentas ficam habilitadas quando o aplicativo for criado para o modo Debug. Quando o modo Release é selecionado, essas ferramentas ficam desabilitadas e seu código fonte não é linkado ao executável. Todas as saídas desse tipo de diagnóstico são relizadas dentro da pasta Debug da janela de Output.

5.1 A macro TRACE

Essa macro foi utilizada em alguns exemplos anteriores. Declarações TRACE são ativadas somente quando a constante _DEBUG está definida. Essa macro trabalha de forma semelhante a função printf do C, porém é completamente desativada na versão Release do programa.

Um típico exemplo do uso dessa função é mostrado abaixo:

```
int nContador = 10;
CString str(“total”);
TRACE("Contador = %d, String = %s\n", nContador, str);
```

5.2 O objeto afxDump

Uma alternativa ao uso da macro TRACE mais compatível com a linguagem C++ é o objeto da MFC afxDump, que aceita variáveis de forma semelhante ao cout, objeto de saída padrão do C++. Não é necessário complexas strings de formatação ao invés disso utiliza-se de operadores sobrecarregados para realizar a saída. O objeto afxDump tem o mesmo destino de saída que a macro TRACE, porém ele só está definido na versão Debug da MFC.

Um trecho que executa tarefa semelhante ao exemplo anterior para a macro TRACE está listado abaixo:

```
int nContador = 10;
CString str(“total”);
#ifdef _DEBUG
        afxDump << “Contador = “ << nContador << “String = “
                << str << “\n”;
#endif // _DEBUG
```

Ambos, afxDump e cout, utilizam o mesmo operador de inserção (<<), porém não compartilham seu código. O objeto cout é parte da biblioteca iostream do Visual C++, enquanto que o objeto afxDump é parte da biblioteca MFC.

Classes que não são derivadas de CObject, possuem operadores de inserção para objetos do tipo CDumpContext.

5.3 O contexto dump e a classe CObject

Como o operador de inserção de CDumpContext aceita ponteiros e referências para CObject, deve também aceitar ponteiros e referências para suas classes derivadas. Considere o a seguinte classe trivial:

```
class CEvent : public CObject
{
public:
        int m_nTime;
};
```

O que acontecerá quando a seguinte instrução for executada?

```
#ifdef _DEBUG
        afxDump << event; // event é um objeto do tipo CEvent
#endif // _DEBUG
```

A função virtual Dump de CObject será chamada. Se ela não foi sobrecarregada em CEvent, não serão obtidas muitas informações, exceto o endereço do objeto. Entretanto se ela foi sobrecarregada, podemos obter o estado interno do Objeto. O exemplo a seguir exibe uma função Dump para a classe CEvent:

```
#ifdef _DEBUG
void CEvent::Dump(CDumpContext& dc) const
{
        CObject::Dump(dc);
        dc << "tempo = " << m_nTime << "\n";
}
#endif // _DEBUG
```

A função Dump da classe base (CObject) imprime uma linha como esta:

```
a CObject at $762A60
```

Se as macros *DECLARE_DINAMIC* tivesse sido clamada na definição da classe CEvent e a macro *IMPLEMENT_DINAMIC* na implementação de CEvent, a linha anterior seria mostrada da seguinte forma:

```
a CEvent at $762A60
```

5.4 Dump automático de objetos não excluídos

Quando a versão Debug é selecionada, a estrutura das aplicações se encarrega de executar o Dump dos objetos que não foram deletados ao sair do programa. Esse Dump pode ser utilizado como uma ferramenta útil no diagnóstico dos programas, localizando exatamente onde poderá ocorrer perda de memória no programa.

6 Exemplo Curso6A

Este exemplo trata de uma interação simples entre Documentos e Vistas. A classe CCurso6ADoc, derivada de CDocument, contém um objeto do Tipo CEstudante. A classe CEstudante representa um registro de um estudante composto por dois membros, um CString para representar o nome e um inteiro para representar a nota. A classe CCurso6AView, é derivada de uma CFormView contém uma representação visual do registro, com caixas de edição para representar o nome e nota. O Botão Enter, atualiza os dados do documento com o conteúdo das caixas de edição. A figura a seguir mostra o programa Curso6A em execução:

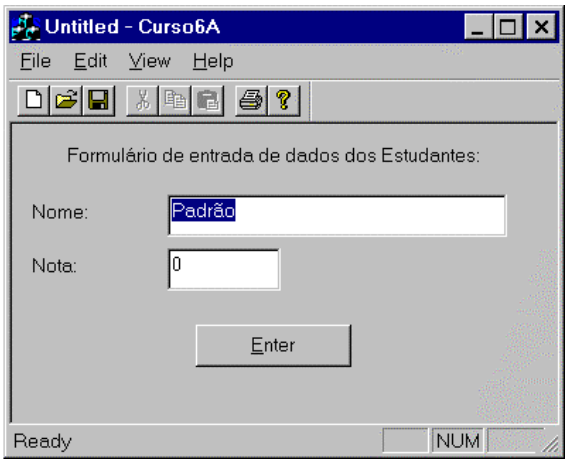


Figura 42 – Tela de execução do Programa Curso6A

Os passos a seguir devem ser seguidos para criar este exemplo.

6.1 Passo 1 – Utilize o AppWizard para gerar o exemplo Curso6A

Gerar como um projeto SDI, e no Step 6 of 6 , alterar a classe base da view do projeto para CFormView como mostrado na figura abaixo:

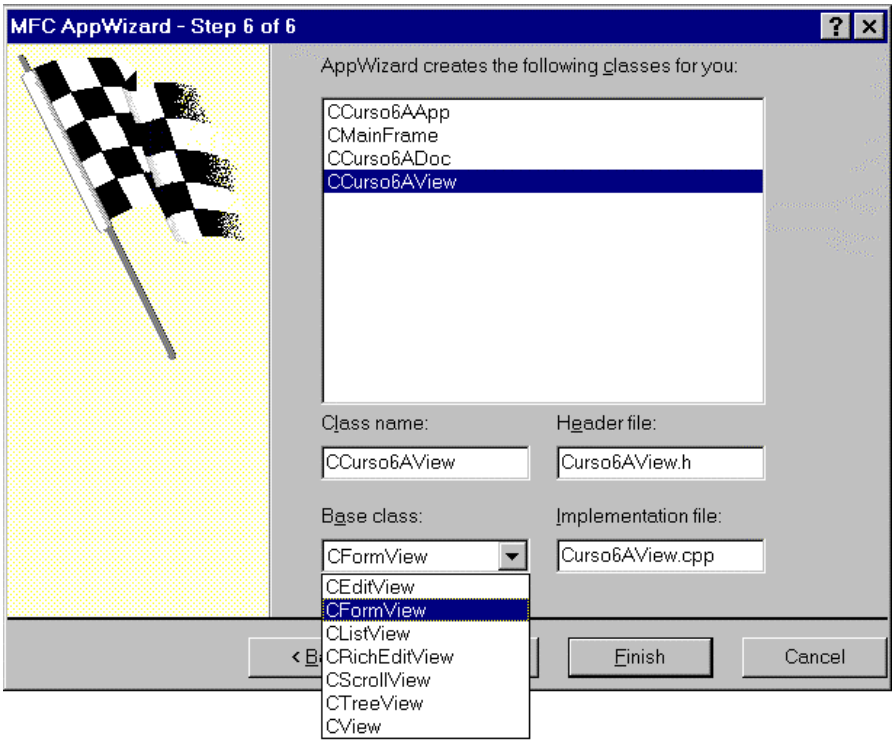


Figura 43 – Step 6 of 6 do projeto Curso6A

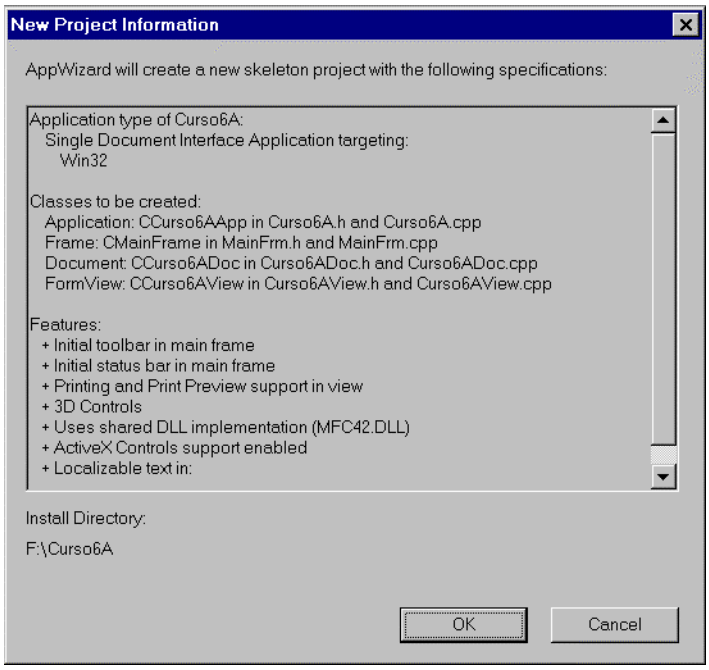


Figura 44 – Opções de criação do exemplo Curso6A

6.2 Passo 2 – Utilize o editor de menus para acrescentar uma opção ao menu Edit

Excluir as opções constantes no menu Edit e inserir a opção *Limpar Tudo* como mostrado abaixo:

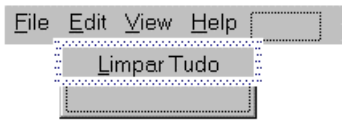


Figura 45 – Menu Edit do aplicativo Curso6A.

Utilize a constante padrão sugerida pela estrutura das aplicações ID_EDIT_LIMPARTUDO. Edite a caixa de texto Prompt para torná-lo igual a caixa de propriedades abaixo:

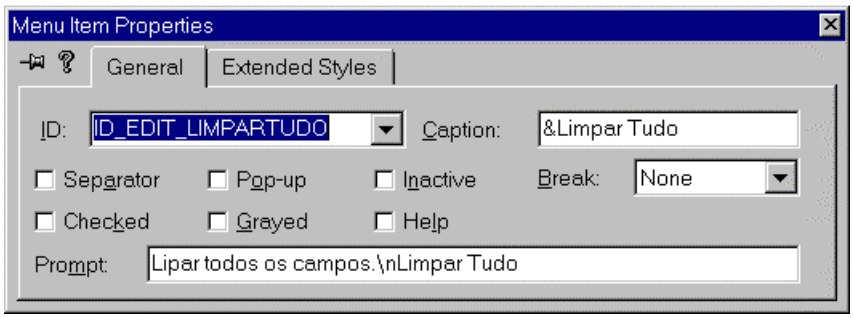


Figura 46 – Tela de propriedades do item de menu.

6.3 Passo 3 – Utilize o editor de diálogo para alterar o diálogo IDD_CURSO6A_FORM.

Abra o diálogo gerado pelo AppWizard, IDD_CURSO6A_FORM, e adicione os controles como mostrados na Figura 47.

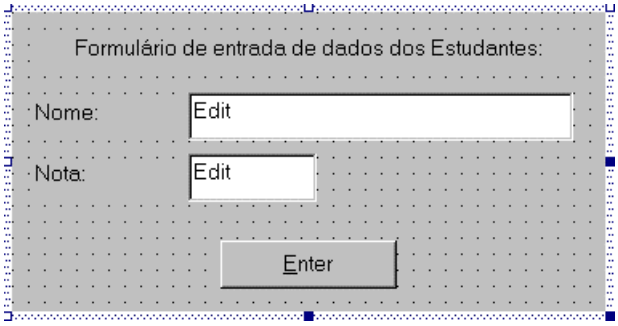


Figura 47 – Form para entrada dos dados

Controle	Identificador
Caixa de Texto *Nome*	IDC_NOME
Caixa de Texto *Nota*	IDC_NOTA
Botão *Enter*	IDC_ENTER

Certifique-se que as propriedades do diálogo que será a base da FormView sejam exatamente como as mostradas na figura a seguir.

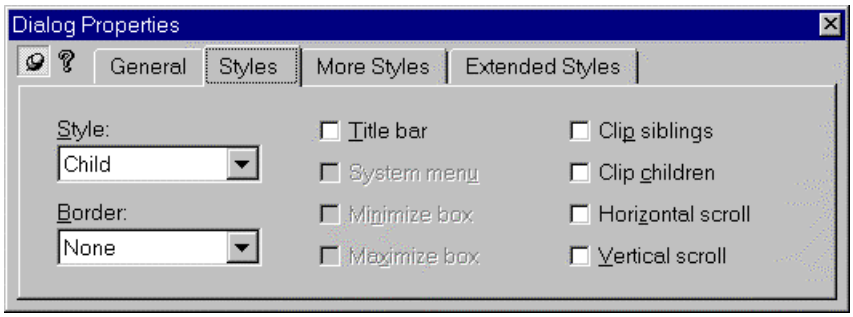


Figura 48 – Propriedades da Form

Obs: elas devem coincidir também com o comentário do item 3.

6.4 Passo 4 – Utilize o ClassWizard para mapear as mensagens para a classe CCurso6AView

Selecione a classe CCurso6AView e adicione os manipuladores para as mensagens listadas na tabela abaixo. Aceite os nomes padrões sugeridos pelo ClassWizard.

Identificador	Mensagem	Função Membro
IDC_ENTER	BN_CLICKED	OnEnter
ID_EDIT_LIMPARTUDO	COMMAND	OnEditLimpartudo
ID_EDIT_LIMPARTUDO	UPDATE_COMMAND_UI	OnUpdateEditLimpartudo

6.5 Passo 5 – Utilize o ClassWizard para adicionar variáveis membros da classe CCurso6AView

Selecione a pasta Member Variables, do diálogo do ClassWizard e adicione as variáveis mostradas na tabela abaixo:

Identificador	Variável Membro	Categoria	Tipo
IDC_NOME	m_strNome	Value	CString
IDC_NOTA	m_nNota	Value	Int

Para a variável membro m_nNota, entre os valores 0 e 100, nas caixas de edição Minimum Value e Maximum Value respectivamente.

6.6 Passo 6 – Utilize o ClassWizard para sobrecarregar a função membro virtual OnInitialUpdate da classe CCurso6AView

Selecione a classe CCurso6AView na lista *Object ID's* e localize a entrada para a função virtual *OnInitialUpdate* na lista *Messages* e sobrecarregue essa função para a classe.

6.7 Passo 7 – Adicione o protótipo para a função membro UpdateControlsFromDoc

Na janela ClassView do Workspace, pressione o botão direito do mouse sobre a classe CCurso6AView, e selecione a opção Add Member Function. Preencha os dados do diálogo com os valores mostrados abaixo.

```
private:
        void UpdateControlsFromDoc();
```

6.8 Passo 8 – Edite o arquivo Curso6AView.cpp

O ClassWizard cria o esqueleto da função OnInitialUpdate, e o CLassView gera o esqueleto da função UpdateControlsFromDoc. Utilize o editor para tornar as funções citadas como mostrado abaixo:

```
// Função chamada durante inicialização
void CCurso6AView::OnInitialUpdate()
{
        UpdateControlsFromDoc() ;
}

// Função chamada por OnInitialUpdate e OnEditLimpartudo
void CCurso6AView::UpdateControlsFromDoc()
{
        CCurso6ADoc* pDoc = GetDocument();
        m_strNome = pDoc->m_estudante.m_strNome;
        m_nNota = pDoc->m_estudante.m_nNota;
        UpdateData(FALSE);      // Atualizar a tela via DDX (inicialização)
}
```

A função OnEnter substitui a OnOK, como descrito no capítulo anterior. Essa função é responsável por transferir os dados das caixas de edição para as variáveis membros da classe Doc. Edite o código dessa função para torná-la como abaixo:

```
void CCurso6AView::OnEnter()
{
        CCurso6ADoc* pDoc = GetDocument();
        UpdateData(TRUE);               // Retornar dados da tela
        pDoc->m_estudante.m_strNome = m_strNome;
        pDoc->m_estudante.m_nNota = m_nNota;
}
```

Em um aplicativo com múltiplas views, a função OnEditLimpartudo deve ser mapeada dentro da classe document, como este exemplo contém apenas uma view associada ao documento ela pode ser mapeada dentro da view. A função OnUpdateEditLimpartudo é chamada pela estrutura das aplicação para desabilitar a opção do menu se o objeto estudante estiver vazio. Edite seus códigos para torná-los como mostrado abaixo:

```
void CCurso6AView::OnEditLimpartudo()
{
        GetDocument()->m_estudante = CEstudante(); // cria novo objeto vazio
        UpdateControlsFromDoc() ;
}

void CCurso6AView::OnUpdateEditLimpartudo(CCmdUI* pCmdUI)
{
        pCmdUI->Enable( GetDocument()->m_estudante != CEstudante() );
}
```

6.9 Passo 9 – Edite o projeto Curso6A para adicionar os arquivos para a classe CEstudante

A implementação da classe CEstudante serve como um bom guia para a representação de objetos do mundo real dentro de um programa de computador.

Arquivo de Descrição da Classe CEstudante, Estudante.h

```
//
// Arquivo de descrição da classe CEstudante
//
// Estudante.h

#ifndef _CURSOVC_CESTUDANTE
#define _CURSOVC_CESTUDANTE

class CEstudante : public CObject
{
        DECLARE_DYNAMIC(CEstudante)
public:
        CString m_strNome;
        int m_nNota;

        // Construtor Padrão
        CEstudante()
        {
                m_strNome = "";
                m_nNota = 0;
        }

        // Construtor com parâmetros
        CEstudante(const char* szName, int nGrade) : m_strNome(szName)
        {
                m_nNota = nGrade;
```

```
	}

	// Construtor de Inicialização
	CEstudante(const CEstudante& s) : m_strNome(s.m_strNome)
	{
		m_nNota = s.m_nNota;
	}

	// Operador =   (Atribuição)
	const CEstudante& operator =(const CEstudante& s)
	{
		m_strNome = s.m_strNome;
		m_nNota = s.m_nNota;
		return *this;
	}

	// Operador ==  (Comparação)
	BOOL operator ==(const CEstudante& s) const
	{
		if ((m_strNome == s.m_strNome) && (m_nNota == s.m_nNota)) {
			return TRUE;
		}
		else {
			return FALSE;
		}
	}

	// Operador !=  (Comparação)
	BOOL operator !=(const CEstudante& s) const
	{
	// Basta utilizar uma negação (!) com o operador == já implementado
		return !(*this == s);
	}

#ifdef _DEBUG
	void Dump(CDumpContext& dc) const;
#endif // _DEBUG
};

#endif // _CURSOVC_CESTUDANTE
```

Arquivo de implementação da Classe CEstudante, Estudante.cpp

```
// Arquivo de implementação da Classe CEstudante
// Estudante.cpp

#include "stdafx.h"
#include "Estudante.h"

IMPLEMENT_DYNAMIC(CEstudante, CObject)

#ifdef _DEBUG
void CEstudante::Dump(CDumpContext& dc) const
{
	CObject::Dump(dc);
	dc << "m_strNome = " << m_strNome << "\nm_nNota = " << m_nNota;
}
#endif // _DEBUG
```

Após a edição desses arquivos selecione a opção Add To Project, no menu Project, em seguida escolha a opção Files e selecione os arquivos Estudante.h e Estudante.cpp. O

Developer Studio acrescenta os nomes dos arquivos ao projeto e quando ele for compilado os arquivos também o serão.

6.10 Passo 10 – Adicione um membro de dados do tipo CEstudante a classe CCurso6ADoc

Utilize o ClassView para a adicionar o membros de dados abaixo e o #include será adiciionado automaticamente.

```
public:
        CEstudante m_estudante;
```

O construtor da classe estudante será chamado no momento da criação do objeto documento assim como o destrutor também será chamado ao objeto documento ser destruído.

6.11 Passo 11 – Edite o arquivo Curso6ADoc.cpp

Utilize o construtor da classe Doc para inicializar os membros de dados do objeto estudante, como mostrado abaixo:

```
CCurso6ADoc::CCurso6ADoc() : m_estudante("Padrão", 0)
{
        TRACE("Objeto Documento criado\n");
}
```

Para efeito de visualização do funcionamento do programa, utiliza-se o destrutor da classe doc para chamar a Função Dump, que também deve se encarregar de mostrar o conteúdo do objeto estudante instanciado durante a seção do programa. Altere o destrutor ~CCurso6ADoc() e a função Dump para torná-las como descrito abaixo:

```
CCurso6ADoc::~CCurso6ADoc()
{
#ifdef _DEBUG
        Dump(afxDump);
#endif //_DEBUG
}

void CCurso6ADoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
        dc << "\n" << m_estudante << "\n";
}
```

6.12 Passo 12 – Construa e teste o aplicativo Curso6A

Digite um nome uma graduação, pressione o botão Enter em seguida saia do aplicativo. A janela Debug exibirá o seguinte conteúdo.

```
a CCurso6ADoc at $7601C0
m_strTitle = Untitled
m_strPathName =
m_bModified = 0
m_pDocTemplate = $760E00

a CEstudante at $760214
```

```
m_strNome = Antenor José
m_nNota = 75
```

7 Interações mais avançadas entre Document-View

Para o caso em que múltiplas views serão utilizadas no aplicativo, as interações entre Document-View são mais complexas do que as mostradas no exemplo Curso6A. O problema fundamental consiste de que o usuário pode editar os dados em uma janela e as outras tem que ser notificadas dessa alteração para exibí-las também. As funções UpdateAllViews e OnUpdate serão utilizadas para manter todas as janela atualizadas com os dados da Document. Os passos para o desenvolvimento de um aplicativo deste tipo estão mostrados a seguir.

1. Na descrição da classe derivada de CDocument, declare seus membros de dados.
2. Na classe derivada de CView, utilize o ClassWizard para sobrecarregar a função virtual OnUpdate. A estrutura das aplicações chama esta função quando os dados do documento forem alterados por qualquer razão. Utilize esta função para atualizar cada uma das views com os dados atuais do documento.
3. Avalie todos os comandos que gerarão mensagens e os posicione na classe apropriada. Um bom exemplo é a função OnEditLimpartudo, que deve ser mapeada dentro da classe Doc para que possa ser utilizada por todas as views que utilizem esse documento.
4. Na classe derivada de CView, permita que as mensagens de comando, mensagens do Windows atualizem os membros de dados da classe doc. Certifique-se que todas as funções que alterem os dados da classe doc, chamem o manipulador da mensagem CDocument::UpdateAllViews antes de sair da função que mapeie um comando. Utilize a função membro CView::GetDocument para acessar o objeto documento de dentro de uma view.
5. Na descrição da classe derivada de CDocument, permita que as mensagens de comando atualizem os membros de dados da classe doc. Porém certifique-se de chamar o manipulador da mensagem CDocument::UpdateAllViews antes de sair da função que mapeia um comando.

8 A função CDocument::DeleteContents

Em algum ponto do programa será necessário um função para excluir todos os dados de seu documento. Isto poderia se conseguido através da escrita de uma função membro privada, entretanto a estrutura das aplicações declara a função virtual DeleteContents para a classe CDocument. A estrutura das aplicações faz a chamada de sua função sobrecarregada quando o documento é fechado e em outras oportunidades que serão comentadas no próximo capítulo.

9 A classe de coleção CObList

Uma vez descoberta a existência das classes de coleção, ficaremos nos perguntando como foi possível ficar sem elas. A classe CObList é uma boa representante da família de classes de coleção. Uma vez familiarizados com esta classe, será mais fácil aprender as outras classes de coleção fornecidas com a MFC.

A classe CObList suporta listas ordenadas de ponteiros para objetos de classes derivadas de CObject. Devem ser derivadas de CObject para possuírem as características de facilidade ao diagnóstico e serialização discutidos nos próximos capítulos.

Uma importante característica de CObList é poder misturar ou combinar ponteiros para objetos diferentes.

9.1 Usando CObList com listas primeiro-a-entrar, primeiro-a-sair FIFO

Uma forma mais simples para a utilização de um objeto CObList seria a adição de novos elementos na parte inferior de uma lista, fazendo a remoção de elementos pelo topo dessa mesma lista. O primeiro elemento incluído na lista seria sempre o primeiro elemento a ser removido dessa lista.

Ao ser removido um ponteiro de um objeto de uma coleção, o objeto não será automaticamente eliminado. Uma instrução *delete* será necessária para a eliminação dos objetos inseridos.

As funções membros da classe CObList utilizadas para esta aplicação limitam-se a *AddTail*() ou *AddHead*() e *RemoveRead*() ou *RemoveTail*(), bem como o membro *IsEmpty*() para controlar um loop para a remoção dos elementos.

9.2 Interação com CObList – o tipo POSITION

Supondo que seja necessário alguma interação entre os elementos contidos em uma lista, a classe CObList possui uma função GetNext, a qual retorna um ponteiro para o próximo elemento da lista, contudo, a utilização dessa função requer alguns cuidados. A função GetNext requer um parâmetro do tipo POSITION, que é um long. A variável POSITION é uma representação interna da posição na lista ocupada pelos elementos recuperados. O parâmetro POSITION é declarado como referência, portanto seu valor pode ser alterado dentro da função.

Exemplo de utilização de um loop com GetNext:

```
CEvent * pEvent;
POSITION pos = eventList.GetHeadPosition();
while (pos != NULL) {
        pEvent = (CEvent *) eventList.GetNext(pos);
        pEvent->PrintTime();
}
```

A função GetNext() retorna o ponteiro para o elemento da lista e incrementa a variável pos para o próximo elemento. Quando não é necessário incrementar o ponteiro deve-se utilizar a função membro GetAt().

9.3 O template para a classe de coleção CTypedPtrList

A classe CObList trabalha muito bem se o desejado for uma coleção de ponteiros de tipos diferentes. Em outras palavras se é desejado uma coleção de tipo-seguro, ou seja que contém ponteiros para o mesmo tipo de objeto uma classe de template para coleções deve ser utilizado. CTypedPtrList é um bom exemplo. Este template é utilizado para criar uma coleção personalizada tendo como base as classes CObList e CPtrList.

Para declarar um objeto que é uma lista de objetos do tipo CEvent precisamos da segu8inte linha de código:

```
CTypedPtrList<CObList, CCEvent*> m_pEventList;
```

O primeiro parâmetro é a classe base utilizada para a coleção, e o segundo parâmetro se refere ao tipo da coleção, ou seja o valor utilizado como retorno ou como parâmetro.

Ao utilizar o template declarado acima, o compilador garante que todas as funções membro da lista retornem um ponteiro para CEvent, não necessitando portanto de um operador de casting para converter o retorno da função. Por exemplo o seguinte código é verdadeiro.

```
        pEvent = m_pEventList.GetNext(pos);
```

Para manter uma notação mais simples é possível utilizar a declaração typedef para gerar o que é mostrado a seguir:

```
typedef CTypedPtrList<CObList, CCEvent*> CEventList;
```

Agora m_pEventList pode ser declarado como a seguir:

```
CEventList m_pEventList;
```

10 Exemplo Curso6B

Este exemplo possui algumas diferenças básicas em relação ao anterior, sendo elas:

- Ao invés de conter um único objeto CEstudante incorporado, a classe doc possui agora uma lista de objetos do tipo CEstudante. Essa é a principal vantagem de se utilizar classes em vez de simplesmente adicionar o nome e a nota como uma variável membro da classe doc;
- Possui uma barra de ferramentas que permite o usuário navegar através da lista;
- O aplicativo foi reformulado para permitir o uso de múltiplas views. Notar que o comando de menu Edit Clear All, está mapeado agora para a classe doc, além das funções UpdateAllViews e OnUpdate serem utilizadas em conjunto.

A janela do aplicativo mostrada na Figura 49, possui a barra de ferramentas como diferença básica ao aplicativo Curso6A, além de que os botões da barra ficam habilitados apropriadamente apenas quando devem realizar algum evento.

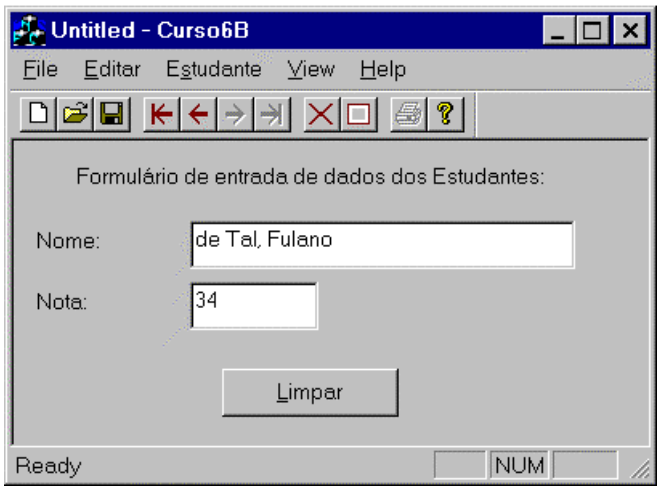


Figura 49 – Tela do programa Curso6B

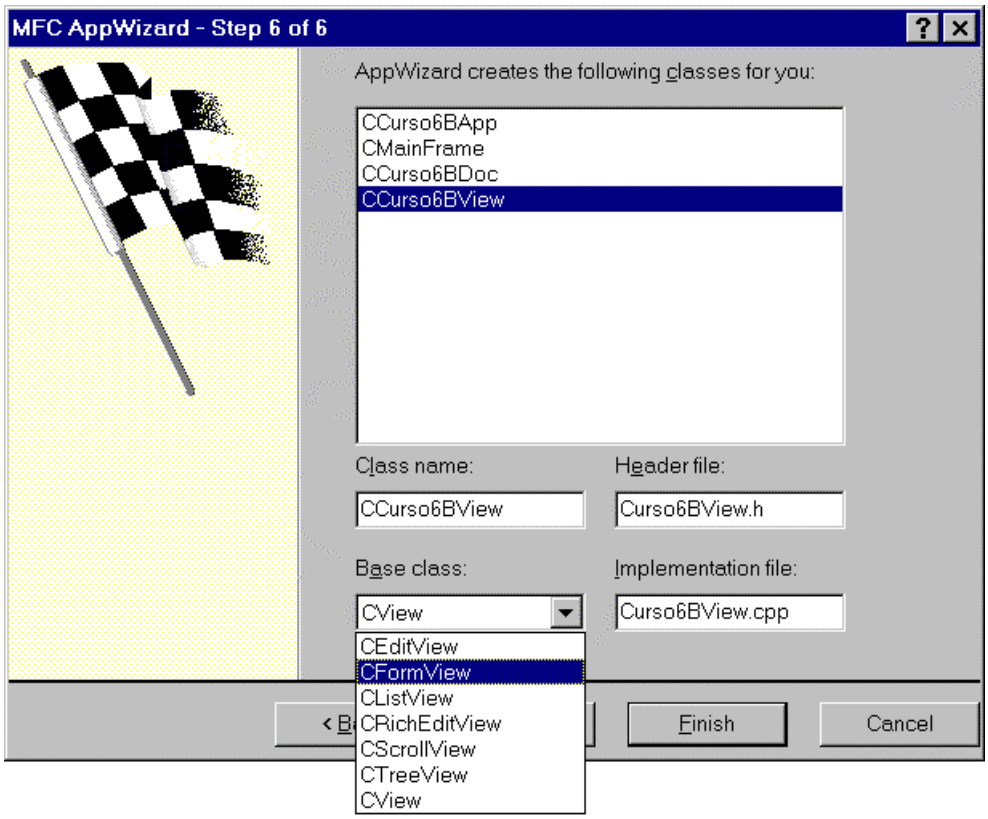


Figura 50 – Step 6 of 6 na criação do exemplo Curso6B

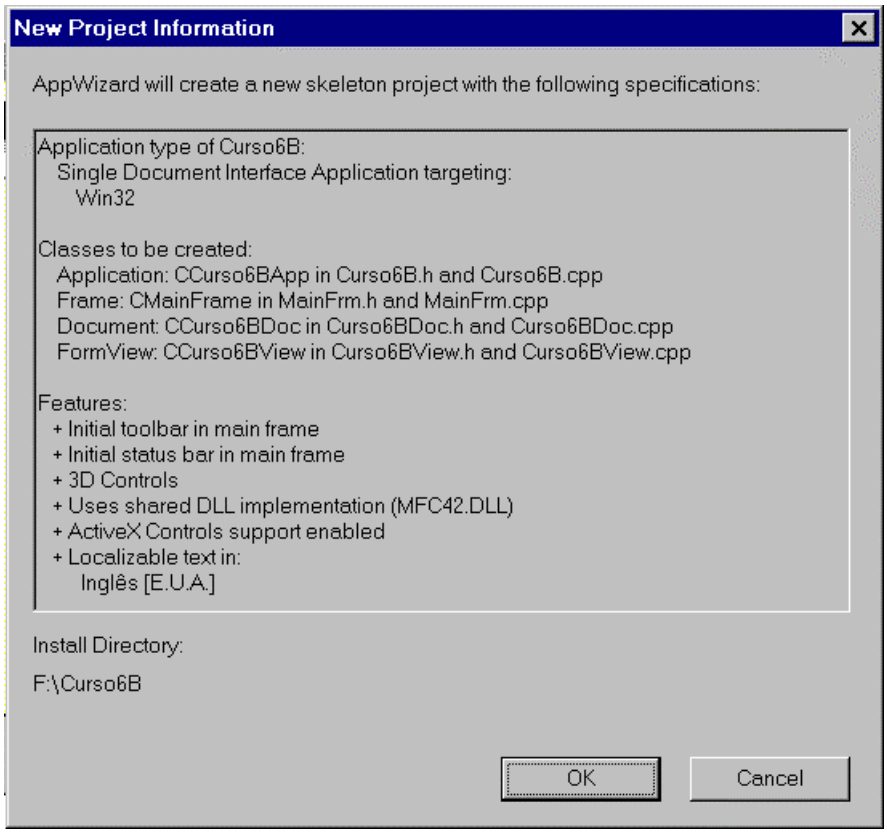


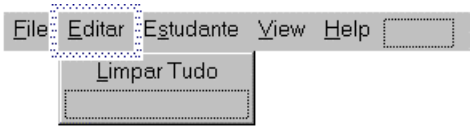
Figura 51 – Opções de criação para o exemplo Curso6B

10.1 Recursos Exigidos

O Arquivo Curso3B.rc deve conter os seguintes recursos que serão necessários para este aplicativo:

10.1.1 Menu Editar

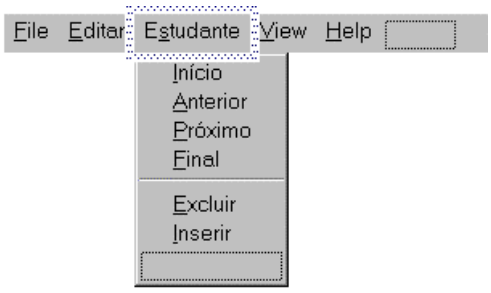
- Selecionar o item *Menu* do arquivo de recursos e escolher o identificador IDR_MAINFRAME (duplo-clique);
- Alterar o submenu Edit, para torná-lo igual ao da figura abaixo;



- Este passo é semelhante ao utilizado no exemplo Curso6A.

10.1.2 Menu Estudante

- Selecionar o item *Menu* do arquivo de recursos e escolher o identificador IDR_MAINFRAME (duplo-clique);
- Criar o submenu Estudante, alterando as propriedades (Capition) do retângulo em branco que aparece na barra do menu que está sendo editado;
- Acrescentar os itens de menu como mostrado na figura abaixo;



- As propriedades de cada um dos itens que serão adicionados devem estar de acordo com a tabela descrita a seguir. Os identificadores são criados automaticamente quando se acrescenta um Capition ao item de menu.

Identificador	Capition	Prompt
ID_ESTUDANTE_INICIO	&Início	Volta ao início da lista de estudantes.\nInício
ID_ESTUDANTE_FINAL	&Final	Avança para o ultimo registro da lista.\nFinal
ID_ESTUDANTE_ANTERIOR	&Anterior	Retorna ao registro anterior na lista.\nAnterior
ID_ESTUDANTE_PROXIMO	&Próximo	Mover para o próximo registro na lista.\nPróximo
ID_ESTUDANTE_INSERIR	&Inserir	Insere um novo registro a lista.\nInserir
ID_ESTUDANTE_EXCLUIR	Excluir	Excluir o registro atual.\nExcluir

10.1.3 Barra de ferramentas

- Selecionar o item *Toolbar* do arquivo de recursos e escolher o identificador IDR_MAINFRAME (duplo-clique);

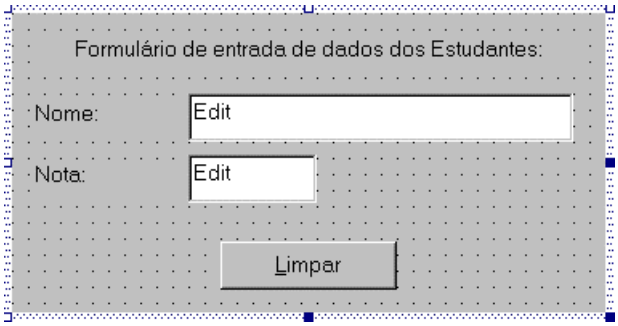
- Alterar a barra de ferramentas para torná-la como na figura abaixo;

- Para acrescentar um botão basta selecionar o botão vazio e desenhar algum ícone sobre ele;
- Para excluir um botão basta arrastá-lo para fora da barra de ferramentas;
- Para criar um separador arraste lateralmente um dos botões que é desejado um espaço entre eles.
- Altere as propriedades dos botões (ID) para que eles possuam as mesmas funções do menu. Obs.: Ao alterar o ID o prompt deve aparecer automaticamente, pois já está associado ao mesmo.

Identificador	Botão	Prompt
ID_ESTUDANTE_INICIO		Volta ao início da lista de estudantes.\nInício
ID_ESTUDANTE_FINAL		Avança para o ultimo registro da lista.\nFinal
ID_ESTUDANTE_ANTERIOR		Retorna ao registro anterior na lista.\nAnterior
ID_ESTUDANTE_PROXIMO		Mover para o próximo registro na lista.\nPróximo
ID_ESTUDANTE_INSERIR		Insere um novo registro a lista.\nInserir
ID_ESTUDANTE_EXCLUIR		Excluir o registro atual.\nExcluir

10.1.4 Layout do diálogo IDD_CURSO6B_FORM

- Selecionar o item *Dialog* do arquivo de recursos e escolher o identificador IDD_CURSO6B_FORM (duplo-clique);
- Editar o formulário para tornálo semelhante ao mostrado na figura a seguir;



- Utilize os seguintes identificadores para cada um dos controles:

Controle	Identificador
Caixa de Texto *Nome*	IDC_NOME
Caixa de Texto *Nota*	IDC_NOTA
Botão *Limpar*	IDC_LIMPAR

- Obs.: Este passo é semelhante utilizado no exemplo Curso6A .

10.2 Código

As seguintes classes serão alteradas/criadas para atender os requisitos de código deste exemplo:

Arquivo de cabeçalho	Arquivo fonte	Classes	Descrição
Curso6BDoc.h	Curso6BDoc.cpp	CCurso6bDoc	Interface com o documento. (Dados)
Curso6BView.h	Curso6BView.cpp	CCurso6bView	Interface com o usuário. (Janela)
Estudante.h	Estudante.cpp	CEstudante	Registro de um estudante.
		CEstudanteList	Coleção de Registros.
StdAfx.h	StdAfx.cpp		Arquivos de inclusão padrão.

10.2.1 Classe CEstudante e CEstudanteList

A classe CEstudante é semelhante a criada para o exemplo Curso6A.

A classe CEstudanteList será definida como uma coleção de registros do tipo CEstudante.

A seguinte linha de código deve ser acrescentada ao final do arquivo estudante.h, para declarar a classe CEstudanteList.

```
};      //fim da declaração da classe CEstudante

// Declaração da classe CEstudanteList
typedef CTypedPtrList<CObList, CEstudante*> CEstudanteList;

#endif // _CURSOVC_CESTUDANTE
```

10.2.2 Arquivo StdAfx.h

O uso do template CTypedPtrList da MFC requer que o arquivo de cabeçalho *afxtempl.h*, esteja incluído dentro do arquivo StdAfx.h

A seguinte linha de código deve ser acrescentada ao arquivo StdAfx.h.

```
#include <afxtempl.h>
```

10.2.3 Classe CCurso6BDoc

As seguintes alterações devem ser feitas na classe CCurso6BDoc para tornar o exemplo funcional:

- **Alterar o Construtor** da classe para permitir o uso do contexto Dump, o código é mostrado a seguir:

```
CCurso6BDoc::CCurso6BDoc()
{
        TRACE("Dentro do construtor da classe CCurso6BDoc\n");
#ifdef _DEBUG
        afxDump.SetDepth(1); // Garante o dump dos elementos da lista
#endif // _DEBUG
}
```

- **Utilize o ClassWizard** para mapear o comando Limpar do menu Editar, conforme tabela mostrda a seguir;

Identificador	Mensagem	Função Membro
ID_EDIT_LIMPARTUDO	COMMAND	OnEditLimpartudo

ID_EDIT_LIMPARTUDO	UPDATE_COMMAND_UI	OnUpdateEditLimpartudo

- **Utilize o ClassWizard** para sobrecarregar a função virtual *DeleteContents*;
- **Edite os códigos** das funções geradas para torná-las como a seguir:

```
void CCurso6BDoc::DeleteContents()
{
#ifdef _DEBUG
	Dump(afxDump);
#endif
	// Excluindo os elementos da lista.
	while (m_estudanteList.GetHeadPosition()) {
		delete m_estudanteList.RemoveHead();
	}
}

void CCurso6BDoc::OnEditLimpartudo()
{
	DeleteContents();		// Esvaziar a lista
	UpdateAllViews(NULL);		// Atualizar todas as janelas
}

void CCurso6BDoc::OnUpdateEditLimpartudo(CCmdUI* pCmdUI)
{
	pCmdUI->Enable(!m_estudanteList.IsEmpty());
}
```

- **Edite o arquivo** Curso6BDoc.h, para acrescentar as seguintes características:

1. Acescentar o include para a classe CEstudante ao início do arquivo:

```
#include "Estudante.h"
```

2. Incluir o método GetList(), que por ser um forma de acesso a um atributo da classe fica implementado de forma inline. O seguinte Código deve ser acrescentado:

```
// Attributes
public:
	CEstudanteList* GetList(){
		return &m_estudanteList;
	}
```

3. Declarar o objeto incorporado m_estudanteList, para manipular os acessos a lista de estudantes:

```
private:
	CEstudanteList m_estudanteList;
```

- **Edite o Código** da função membro Dump para torná-lo como a seguir:

```
void CCurso6BDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
	dc << "\n" << m_estudanteList << "\n";
}
```

10.2.4 Classe CCurso6BView

- **Utilizar o ClassWizard** para acrescentar as seguintes variáveis membros. Através da pasta *Member Variables* selecione os identificadores (duplo-clique) e crie as variáveis mostradas na tabela abaixo:

Identificador	Variável Membro	Categoria	Tipo
IDC_NOME	m_strNome	Value	CString
IDC_NOTA	m_nNota	Value	Int

- **Utilizar o ClassWizard** para mapear a mensagem BN_CLICKED do identificador IDC_LIMPAR (duplo-clique) e adicione a função descrita na tabela abaixo:

Identificador	Mensagem	Função Membro
IDC_LIMPAR	BN_CLICKED	OnLimpar

- **Utilizar o ClassWizard** para mapear os comandos (COMMAND) do menu (duplo-clique) para as funções descritas na tabela a seguir:

Identificador	Mensagem	Função Membro
ID_ESTUDANTE_INICIO	COMMAND	OnEstudanteInicio
ID_ESTUDANTE_FINAL	COMMAND	OnEstudanteFinal
ID_ESTUDANTE_ANTERIOR	COMMAND	OnEstudanteAnterior
ID_ESTUDANTE_PROXIMO	COMMAND	OnEstudanteProximo
ID_ESTUDANTE_INSERIR	COMMAND	OnEstudanteInserir
ID_ESTUDANTE_EXCLUIR	COMMAND	OnEstudanteExcluir

- **Utilizar o ClassWizard** para mapear as atualizações dos comandos (UPDATE_COMMAND_UI) do menu (duplo-clique) para as funções descritas na tabela a seguir:

Identificador	Mensagem	Função Membro
ID_ESTUDANTE_INICIO	UPDATE_COMMAND_UI	OnUpdateEstudanteInicio
ID_ESTUDANTE_FINAL	UPDATE_COMMAND_UI	OnUpdateEstudanteFinal
ID_ESTUDANTE_ANTERIOR	UPDATE_COMMAND_UI	OnUpdateEstudanteInicio
ID_ESTUDANTE_PROXIMO	UPDATE_COMMAND_UI	OnUpdateEstudanteFinal
ID_ESTUDANTE_EXCLUIR	UPDATE_COMMAND_UI	OnEstudanteExcluir

- **Edite os Códigos** das funções mapeadas para torná-los como a seguir:

```
void CCurso6BView::OnLimpar()
{
        TRACE("Dentro de CCurso6BView::OnLimpar\n");
        LimparItem();
}

//
void CCurso6BView::OnEstudanteInicio()
{
        TRACE("Dentro de CCurso6BView::OnEstudanteInicio\n");
```

```
        // este teste é necessário para eliminar problemas com a lista vazia
        if (!m_pList->IsEmpty())
        {
                m_position = m_pList->GetHeadPosition();
                GetItem(m_position);
        }
}

void CCurso6BView::OnEstudanteFinal()
{
        TRACE("Dentro de CCurso6BView::OnEstudanteFinal\n");
        if (!m_pList->IsEmpty())
        {
                m_position = m_pList->GetTailPosition();
                GetItem(m_position);
        }
}

//
void CCurso6BView::OnEstudanteAnterior()
{
        TRACE("Dentro de CCurso6BView::OnEstudanteAnterior\n");
        POSITION pos;
        if ((pos = m_position) != NULL)
        {
                m_pList->GetPrev(pos);
                if (pos)
                {
                        GetItem(pos);
                        m_position = pos;
                }
        }
}

//
void CCurso6BView::OnEstudanteProximo()
{
        POSITION pos;
        TRACE("Dentro de CCurso6BView::OnEstudanteProximo\n");
        if ((pos = m_position) != NULL)
        {
                m_pList->GetNext(pos);
                if (pos)
                {
                        GetItem(pos);
                        m_position = pos;
                }
        }
}

//
void CCurso6BView::OnEstudanteInserir()
{
        TRACE("Dentro de CCurso6BView::OnEstudanteInserir\n");
        InserirItem(m_position);
        GetDocument()->SetModifiedFlag();
        GetDocument()->UpdateAllViews(this);
}

//
void CCurso6BView::OnEstudanteExcluir()
{
        // Excluir o item atual e reposiciona no próximo ou no início
```

```
        POSITION pos;
        TRACE("Dentro de CCurso6BView::OnEstudanteExcluir\n");
        if ((pos = m_position) != NULL)
        {
                m_pList->GetNext(pos); //guarda a posição do próximo elemento
                if (pos == NULL)        // verifica se existe o próximo
                {
                        pos = m_pList->GetHeadPosition();
                        TRACE("GetHeadPos = %ld\n", pos);
                        if (pos == m_position)
                        {
                                pos = NULL;
                        }
                }
                GetItem(pos);                   // posiciona no novo item
                CEstudante* ps = m_pList->GetAt(m_position);
                m_pList->RemoveAt(m_position);  // faz a exclusão do item
                delete ps;                      // exclui o ponteiro
                m_position = pos;
                GetDocument()->SetModifiedFlag();
                GetDocument()->UpdateAllViews(this);
        }
}


//
void CCurso6BView::OnUpdateEstudanteInicio(CCmdUI* pCmdUI)
{
// chamado durante um processo idle e quando o menu Estudante é aberto
        POSITION pos;
        // habilita o botão se a lista não estiver vazia e o não estivar no
        // início da lista.
        pos = m_pList->GetHeadPosition();
        pCmdUI->Enable((m_position != NULL) && (pos != m_position));
}


//
void CCurso6BView::OnUpdateEstudanteFinal(CCmdUI* pCmdUI)
{
// chamado durante um processo idle e quando o menu Estudante é aberto
        POSITION pos;
        // habilita o botão se a lista não estiver vazia e o final da
        // lista não foi atingido.
        pos = m_pList->GetTailPosition();
        pCmdUI->Enable((m_position != NULL) && (pos != m_position));
}


//
void CCurso6BView::OnUpdateEstudanteExcluir(CCmdUI* pCmdUI)
{
// chamado durante um processo idle e quando o menu Estudante é aberto
        // habilita o botão se a posição não for nula
        pCmdUI->Enable(m_position != NULL);
}
```

- **Edite o arquivo** Curso6BView.h, para acrescentar as seguintes características:

  1. Adicionar os membros de Dados, m_position e m_pList:

```
protected:
        POSITION      m_position; // Posição atual no documento
        CEstudanteList*  m_pList; // copiado do documento
```

2. Declarar os protótipos das funções auxiliares

```
protected:
        virtual void LimparItem();
        virtual void InserirItem(POSITION position);
        virtual void GetItem(POSITION position);
```

- **Edite o arquivo** Curso6BView.cpp, para um corpo para as funções qie foram declaradas no .h. O Código para essas funções é mostrado a seguir:

```
//
void CCurso6BView::LimparItem()
{
        m_strNome       = "";
        m_nNota         = 0;
        UpdateData(FALSE);
}

//
void CCurso6BView::InserirItem(POSITION position)
{
        if (UpdateData(TRUE))
        {
                // UpdateData retorna FALSE se algum erro for detectado
                CEstudante* pStudent = new CEstudante;
                pStudent->m_strNome      = m_strNome;
                pStudent->m_nNota        = m_nNota;
                m_position = m_pList->InsertAfter(m_position, pStudent);
        }
}

//
void CCurso6BView::GetItem(POSITION position)
{
        if (position)   // testa se a posição é diferente de NULL
        {
                CEstudante* pStudent = m_pList->GetAt(position);
                m_strNome = pStudent->m_strNome;
                m_nNota = pStudent->m_nNota;
        }
        else
        {                                   // se for nulo apenas limpa a tela
                LimparItem();
        }
        UpdateData(FALSE);
}
```

10.3 Crie e Teste o aplicativo

Execute o aplicativo e digite *Teste de nome*, no campo nome e *44* no campo nota, em seguida pressione o botão *Inserir*. Digite Teste de Nota 100, no campo nome e *100* no campo nota, em seguida pressione o botão *Inserir*. Feche o aplicativo e as seguintes inscrições aparecerão na janela Output.

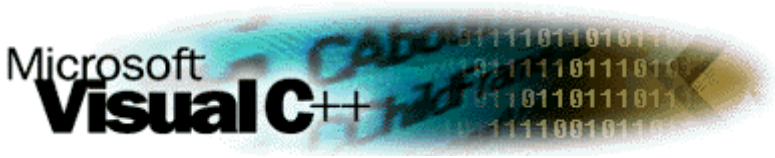
```
Dentro do construtor da classe CCurso6BDoc
```

```
Dentro do construtor de CCurso6BView
Dentro de CCurso6BDoc::OnNewDocument
a CCurso6BDoc at $7601C0
m_strTitle = Untitled
m_strPathName =
m_bModified = 0
m_pDocTemplate = $760E00
with view $7612E0

a CObList at $760214
with 0 elements

Dentro de CCurso6BView::OnInitialUpdate
Dentro de CCurso6BView::OnUpdate
Dentro de CCurso6BView::OnEstudanteInserir
Dentro de CCurso6BView::OnEstudanteInserir
a CCurso6BDoc at $7601C0
m_strTitle = Untitled
m_strPathName =
m_bModified = 1
m_pDocTemplate = $760E00

a CObList at $760214
with 2 elements
        a CEstudante at $762E80
m_strNome = Teste de nome
m_nNota = 44
        a CEstudante at $762640
m_strNome = Teste de Nota 100
m_nNota = 100

Warning: destroying an unsaved document.
```

Utilize o programa para inserir vários dados e navegar por eles utilizando os botões na barra ou os itens do menu. Notar que quando o final da lista é atingido as opções próximo e final ficam desativadas, assim como quando a lista está no início, as opções Anterior e Início também estão desativadas. Isto se deve ao fato do Windows chamar durante um processo de idle, as funções designadas em ON_UPDATE_COMMAND_UI. No interior dessas funções fica o código para habilitar ou desabilitar os respectivos Comandos.

11 Exercício proposto:

Modificar o programa Curso6B para permitir que ele tenha a característica de alterar um valor de um elemento da lista. Você deve criar o item de menu bem como o botão na barra de ferramentas para gerar este comando.



Capítulo 7
Escrevendo e lendo Documentos - SDI

Como deve Ter sido notado, todos os exemplos gerados até agora possuem um menu File com os comandos familiares New, Open, Save e Save As. Neste capítulo é mostrado como os aplicativos podem responder à leitura e gravação de documentos.

O exemplo desse capítulo Curso7A é um aplicativo do tipo SDI (Single Document Interface) baseado no programa Curso6B do capítulo anterior. O programa utiliza o documento de lista de alunos com uma classe view derivada a partir de *CFormView*. Porem a lista de alunos poderá agora ser lida e escrita no disco através de um processo chamado serialização.

1 Serialization – O que é isto?

O termo "serialização" (serialize) pode parecer novo, mas apresenta uma certa utilização no mundo da programação baseada em objetos. A idéia é que os objetos podem ser persistentes, significando que podem ser armazenados em disco quando um programa for encerrado e restaurado quando o for reiniciada a operação desse programa. O Processo de armazenamento e restauração de objetos é denominado serialização. Dentro da MFC as classes possuem uma função membro chamada Serialize. Quando a estrutura das aplicações chama Serialize para um objeto em particular, os dados referentes ao objeto serão restaurados ou armazenados no disco.

Na biblioteca de classes este não será um substituto para o sistema de gerenciamento de banco de dados. Todos os objetos associados a um documento serão lidos ou gravados seqüencialmente em um único arquivo em disco.

1.1 Os arquivos e os Objetos de Arquivamento

Utilizando-se a MFC os arquivos são representados por objetos da classe CFile. A aplicação não executa diretamente os processos de I/O em disco, procurando, como alternativa utilizar o processo de serialização, podemos evitar o uso direto de objetos CFile. Entre a função Serialize e o objeto CFile há um objeto de arquivamento (CArchive), como mostrado na Figura 52 abaixo.

O objeto CArchive coloca dados em buffers para o objeto CFile, mantendo um flag interno que indica se o arquivamento envolve um armazenamento (gravação em disco) ou um carregamento (leitura no disco). A estrutura das aplicações se encarrega da construção dos Objetos CFile e CArchive, abrindo o arquivo em disco para o objeto CFile e associando o objeto de arquivamento ao arquivo. Ao ser utilizada a função Serialize, será necessário apenas carregar ou armazenar dados no objeto de arquivamento.

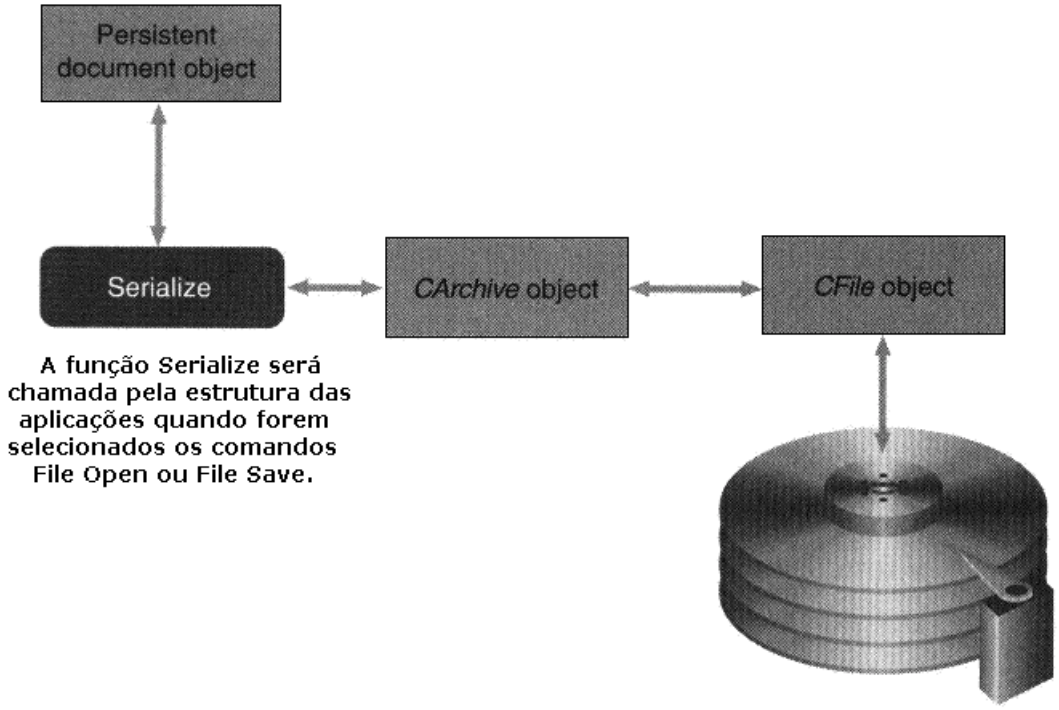


Figura 52 - O processo de serialização

1.2 Criando um classe Serializável

Uma classe serializável deve ser derivada direta ou indireta de CObject. A chamada da macro DECLARE_SERIAL deve ser incluída na declaração da classe, e a macro IMPLEMENT_SERIAL deve ser chamada no arquivo de implementação da classe. A classe CEstudante que será utilizada no exemplo já utiliza estas macros.

1.3 Escrevendo a função Serialize

No capítulo anterior foi discutida a classe CEstudante, derivada de CObject, com os seguintes membros de dados:

```
public:
        CString m_strNome;
        int m_nNota;
```

A classe CEstudante deve possuir a função Serialize para permitir a capacidade de armazenamento de seus objetos, nossa tarefa é implementá-la. Devido ao fato da função membro Serialize ser um membro virtual da classe base CObject devemos garantir que o valor e os tipos de parâmetros retornados coincidam com a declaração de CObject. A seguir encontra-se o código da função Serialize a ser implementada para a classe CEstudante.

```
void CEstudante::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
                ar << m_strNome << m_nNota;
        else
                ar >> m_strNome >> m_nNota;
}
```

Normalmente as funções de serialização chamam a função Serialize de suas classes básicas. Por exemplo se a classe CEstudante fosse derivada de CPessoa, a primeira linha da função Serialize seria a seguinte:

```
CPessoa::Serialize(ar);
```

As funções Serialize das classes CObject e CDocument não executam ações úteis, portanto não são chamadas.

1.4 Serializando coleções

Como todas as classes de coleção são derivadas da classe *CObject* e também devido as declarações das classes de coleção conterem chamadas para a macro *DECLARE_SERIAL,* podemos serializar de forma conveniente as coleções com uma chamada à função-membro *Serialize* da classe de coleção. Por exemplo, se chamarmos *Serialize* para uma coleção de *CObList* referente a objetos *CEstudante,* a função *Serialize* de cada objeto *CEstudante* será chamada em cada oportunidade. Entretanto, devemos conhecer alguns detalhes específicos a respeito do carregamento de coleções a partir de um objeto de arquivamento:

- Se uma coleção possuir ponteiros de objetos pertencentes a varias classes (todas derivadas de *CObject),* em essência os nomes individuais das classes serão armazenados no objeto de arquivamento, de modo que os objetos possam ser construídos adequadamente com o uso do construtor apropriado para a classe.
- Se um objeto de armazenamento, como um documento, possuir uma coleção *interna,* os dados carregados serão anexados à coleção existente. Pode ser necessário esvaziar a coleção antes de fazer o carregamento a partir do objeto de arquivamento. Isso normalmente será feito em uma função *DeleteContents,* a qual será chamada pela estrutura das aplicações.
- Se um objeto de armazenamento possuir um ponteiro para uma coleção, será construído um novo objeto de coleção quando o operador de extração carregar dados a partir do objeto de arquivamento. Um ponteiro para a nova coleção será armazenado no membro de dados de ponteiro do objeto de armazenamento. Pode ser necessária a destruição do objeto antigo de coleção (após o mesmo ter sido esvaziado) antes de ser feito o carregamento a partir do objeto de arquivamento.
- Quando uma coleção de ponteiros *CObject* for carregada a partir de um objeto de arquivamento, os seguintes passos de processamento irão ocorrer para cada objeto contido na coleção:
  - A classe do objeto será identificada.
  - Será feita a alocação de um espaço de armazenamento no heap para o objeto.
  - Os dados do objeto serão carregados na área de armazenagem recém-alocada.
  - Um ponteiro para o novo objeto será armazenado na coleção.

O exemplo Curso7A mostra a serialização de uma coleção interna de *CObList.*

1.5 A função Serialize e a estrutura das aplicações.

Bem, agora sabemos como podem ser criadas as funções de serialização e também sabemos que essas chamadas de função podem estar na forma multinivelada. Contudo, será que sabemos quando a primeira função *Serialize* será chamada para iniciar o processo de serialização? Ao ser utilizada a estrutura de aplicações, tudo dependerá do documento (o objeto de uma classe derivada de *CDocument).* Ao ser feita uma seleção Save ou Open no menu File, a estrutura das aplicações cria um objeto *CArchive* (e um objeto *CFile* subjacente) e, em seguida, chama a função *Serialize* da classe de documento, passando

uma referencia para o objeto *CArchive.* A função *Serialize* da classe derivada de documento, então, serializa cada um dos membros de dados não-temporários.

2 Aplicativo SDI

Já vimos muitas aplicações da SDI que possuem uma classe de documento e uma classe de vista. Vamos manter o uso de apenas uma classe de vista neste capítulo, mas iremos explorar as inter-relações existentes entre os elementos da estrutura das aplicações criados para os programas. Como por exemplo o objeto aplicativo, a janela principal (mainframe), o documento, a vista e menu.

2.1 O Objeto Aplicativo do Windows.

Para cada programa criado via AppWizard, uma classe derivada de CWinApp é gerada, além de inserir a instrução a seguir dentro de seu arquivo de implementação:

```
CMyApp theApp;
```

Esse é o mecanismo pelo qual a MFC inicializa seus aplicativos. A classe CMyApp será derivada da classe CWinApp, com theApp sendo uma instância (objeto) da classe declarada de forma global. O Objeto global é denominado *objeto do aplicativo para Windows.*

Ao iniciar o programa o objeto global será criado e a estrutura das aplicações se encarrega de chamar a função membro InitInstance, onde é feita a conexão entre os elementos frame, doc e view, através do chamado Template. Pode-se entender como Template a aparência de um programa. Após feitas as conexões, a estrutura das aplicações chama o método Run que mantém o aplicativo ativo e inicia o processo de envio de mensagens de janelas e de comandos.

2.2 Relacionando a abertura de arquivo ao código serializável

Quando o AppWizard gera um aplicativo, fará também o mapeamento do item de menu File Open para a função membro OnFileOpen de CDocument, a qual, através de uma chamada à função OpenDocumentFile de CWinApp, executa os passos a seguir.

1. Chama a função membro virtual OnOpenDocument para o Objeto de Documento já existente. Essa função solicita ao usuário a seleção de um arquivo e, em seguida, abre o arquivo, constrói um objeto CArchive definido para o carregamento de dados e, então chama DeleteContents.
2. Chama a função Serialize do documento, a qual carrega os dados existentes no objeto de arquivamento.
3. Chama a função OnInitialUpdate da vista.

2.3 Relacionando a gravação do arquivo ao código serializável

Quando o AppWizard gera um aplicativo, fará também o mapeamento do item de menu File Save para a função membro OnFileSave de CDocument. A função OnFileSave chama a função função OnSaveDocument de CDocument, a qual, por sua vez chama a função Serialize do documento com o objeto de arquivamento definido para o armazenamento (gravação). O item de menu File Save As será processado de maneira semelhante; este item de menu será mapeado para a função OnFileSaveAs de CDocument, a qual chama OnSaveDocument. Para esta situação, a estrutura das aplicações executa todo o gerenciamento de arquivos necessário ao salvamento do documento no disco.

2.4 A função DeleteContents

Ao ser carregado um objeto existente de documento da SDI a partir do disco, teremos que de alguma forma, apagar o conteúdo atual do objeto de documento. A melhor maneira de se fazer isso é sobrecarregar a função virtual DeleteContents de CDocument existente na classe derivada de documento. Em resposta aos item de menu File New e File Open, as funções OnFileNew e OnFileOpen de CDocument chamam a função DeleteContents, significando que DeleteContents será chamada imediatamente após o objeto de documento ser inicialmente construído. A função será chamada novamente ao ser fechado um documento.

2.5 O flag de documento alterado

Muitos aplicativos do Windows baseadas em documentos rastreiam possíveis modificações feitas no documento pelo usuário. Se o usuário tentar fechar o documento ou encerrar o programa, uma caixa de mensagem irá perguntar se o documento deve ser salvo. A estrutura das aplicações da MFC suporta diretamente este comportamento, com o membro de dados m_bModified de CDocument. Esta variável lógica será TRUE se o documento possuir modificações; caso contrário, terá um valor FALSE. O flag protegido m_bModified será acessado através das funções membro SetModifiedFlag e IsModified de CDocument.

3 Exemplo Curso7A

Este exemplo é similar ao programa Curso6B.

3.1 As alterações necessárias durante a criação do projeto:

- Tornar o exemplo baseado no modelo de documento SDI (Single Document) no passo 1 do AppWizard.
- Alterar a string que vai ser associada aos documento do exemplo para *07A* durante o passo 4 do AppWizard, através do botão Advanced... que ao ser pressionado exibe a caixa de diálogo mostrada na Figura 53.
- No passo 6 do AppWizard, alterar a classe base de CCurso7AView para uma CFormView, como mostrado no exemplo Curso6A e Curso6B.
- A tela com o resumo dos arquivos e características acrescentadas para o programa Curso7A está mostrada na

3.2 Recursos

Os recursos utilizados para programa são os mesmos utilizados no exemplo Curso6B.

Utilize o editor de recursos para copiá-los para o exemplo Curso7A, através da abertura do arquivo curso6B.rc e exploração de seu conteúdo.

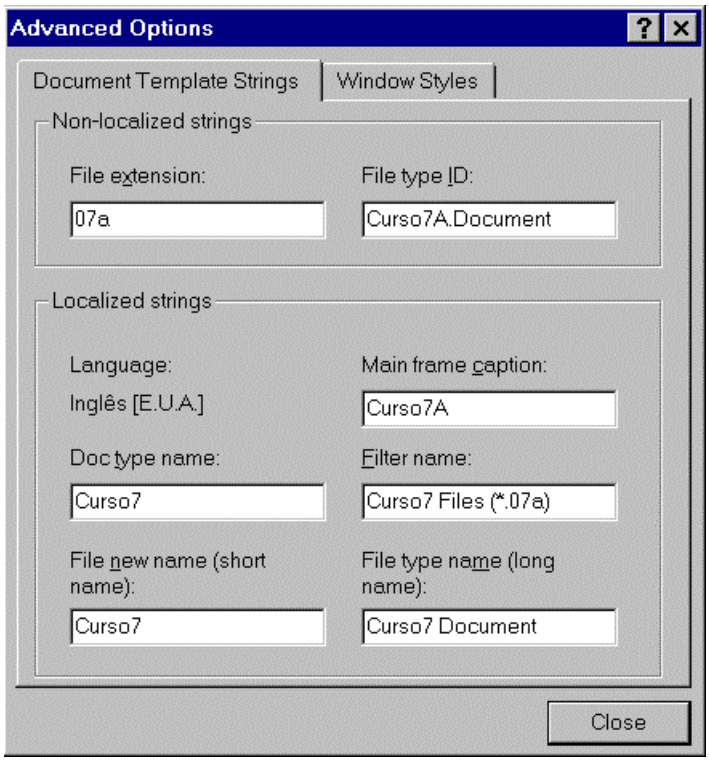


Figura 53 - Diálogo para alteração das strings relacionadas com o modelo de Documento.

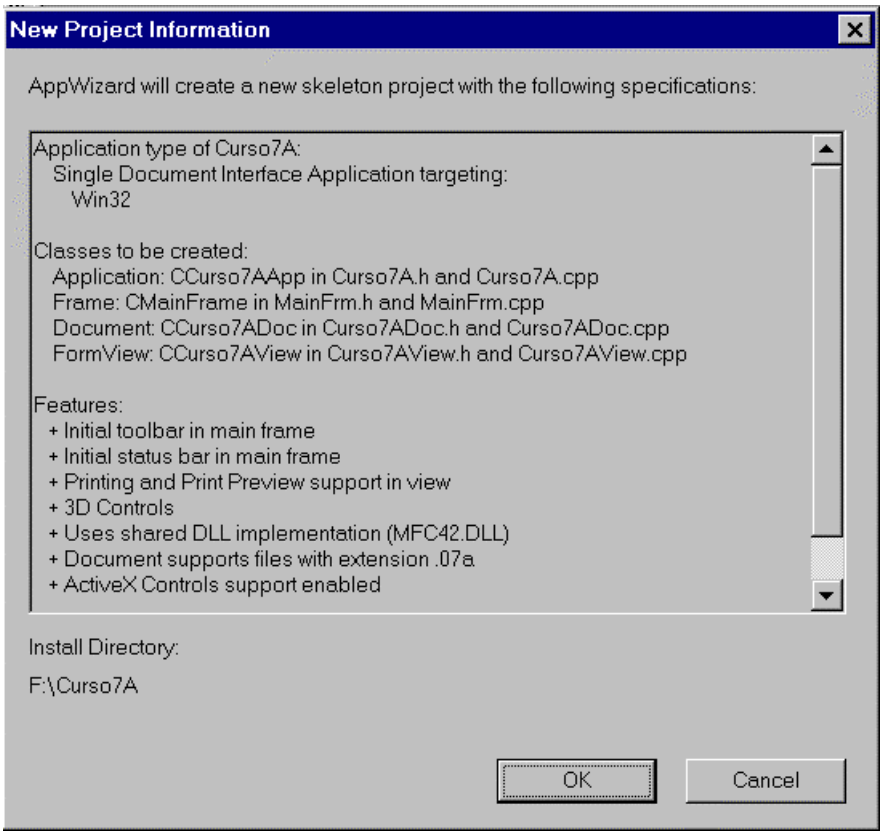


Figura 54 - Características do esqueleto do programa Curso7A

3.3 Código

Os requisitos de código que diferem do exemplo anterior serão listados a seguir, respeitando a notação de realçado em cinza o código que deve ser substituído ou alterado em relação ao exemplo Curso6B.

3.3.1 Classe CEstudante

A nova listagem para o Objeto estudante encontra-se a seguir:

Arquivo de descrição da classe CEstudante, **Estudante.h**:

```
//
// Arquivo de descrição da classe CEstudante
//
// Estudante.h

#ifndef _CURSOVC_CESTUDANTE
#define _CURSOVC_CESTUDANTE

class CEstudante : public CObject
{
        DECLARE_SERIAL(CEstudante)
public:
        CString m_strNome;
        int m_nNota;

        // Construtor Padrão
        CEstudante()
        {
                m_strNome = "";
                m_nNota = 0;
        }

        // Construtor com parâmetros
        CEstudante(const char* szName, int nGrade) : m_strNome(szName)
        {
                m_nNota = nGrade;
        }

        // Construtor de Inicialização
        CEstudante(const CEstudante& s) : m_strNome(s.m_strNome)
        {
                m_nNota = s.m_nNota;
        }

        // Operador =           (Atribuição)
        const CEstudante& operator =(const CEstudante& s)
        {
                m_strNome = s.m_strNome;
                m_nNota = s.m_nNota;
                return *this;
        }

        // Operador ==          (Comparação)
        BOOL operator ==(const CEstudante& s) const
        {
                if ((m_strNome == s.m_strNome) && (m_nNota == s.m_nNota)) {
                        return TRUE;
                }
```

```
                else {
                        return FALSE;
                }
        }

        // Operador !=          (Comparação)
        BOOL operator !=(const CEstudante& s) const
        {
        // Basta utilizar uma negação (!) com o operador == já implementado
                return !(*this == s);
        }

        virtual void Serialize(CArchive& ar);

#ifdef _DEBUG
        void Dump(CDumpContext& dc) const;
#endif // _DEBUG
};              //fim da declaração da classe CEstudante

//Declaração da classe CEstudanteList
typedef CTypedPtrList<CObList, CEstudante*> CEstudanteList;

#endif // _CURSOVC_CESTUDANTE
```

Arquivo de Implementação da classe CEstudante, **Estudante.cpp**

```
// Arquivo de implementação da Classe CEstudante
// Estudante.cpp

#include "stdafx.h"
#include "Estudante.h"

IMPLEMENT_SERIAL(CEstudante, CObject, 0)

void CEstudante::Serialize(CArchive& ar)
{
        TRACE("Dentro de CEstudante::Serialize\n");
        if (ar.IsStoring()) {
                ar << m_strNome << m_nNota;
        }
        else {
                ar >> m_strNome >> m_nNota;
        }
}

#ifdef _DEBUG
void CEstudante::Dump(CDumpContext& dc) const
{
        CObject::Dump(dc);
        dc << "m_strNome = " << m_strNome << "\nm_nNota = " << m_nNota;
}
#endif // _DEBUG
```

3.3.2 Classe CCurso7ADoc

- Utilize o ClassWizard para sobrecarregar as funções virtuais OnOpenDocument e DeleteContents;

- Utilize o ClassWizard para adicionar a função OnUpdateFileSave, que é a resposta a atualização do comando de menu File Save (mensagem UPDATE_COMMAND_UI, para o Objeto ID_FILE_SAVE);
- O conteúdo dessas funções bem como as alterações necessárias para o funcionamento deste exemplo estão mostradas abaixo seguindo a notação utilizada.

Arquivo de descrição da classe CCurso7ADoc, **Curso7ADoc.h.**

```
// Curso7ADoc.h : interface of the CCurso7ADoc class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CURSO7ADOC_H__5092750A_099C_11D3_B817_D53F1F592A0E__INCLUDED_)
#define AFX_CURSO7ADOC_H__5092750A_099C_11D3_B817_D53F1F592A0E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "Estudante.h"

class CCurso7ADoc : public CDocument
{
protected: // create from serialization only
        CCurso7ADoc();
        DECLARE_DYNCREATE(CCurso7ADoc)

// Attributes
public:
        CEstudanteList* GetList(){
                return &m_estudanteList;
        }
// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CCurso7ADoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual void DeleteContents();
        virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CCurso7ADoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
```

```
        //{{AFX_MSG(CCurso7ADoc)
        afx_msg void OnEditLimpartudo();
        afx_msg void OnUpdateEditLimpartudo(CCmdUI* pCmdUI);
        afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

private:
        CEstudanteList m_estudanteList;

};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
#endif // AFX_CURSO7ADOC_H__5092750A_099C_11D3_B817_D53F1F592A0E__INCLUDED_
```

Arquivo de implementação da classe CCurso7ADoc, **Curso7ADoc.cpp**

```
// Curso7ADoc.cpp : implementation of the CCurso7ADoc class
//

#include "stdafx.h"
#include "Curso7A.h"

#include "Curso7ADoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCurso7ADoc

IMPLEMENT_DYNCREATE(CCurso7ADoc, CDocument)

BEGIN_MESSAGE_MAP(CCurso7ADoc, CDocument)
        //{{AFX_MSG_MAP(CCurso7ADoc)
        ON_COMMAND(ID_EDIT_LIMPARTUDO, OnEditLimpartudo)
        ON_UPDATE_COMMAND_UI(ID_EDIT_LIMPARTUDO, OnUpdateEditLimpartudo)
        ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CCurso7ADoc construction/destruction

CCurso7ADoc::CCurso7ADoc()
{
        TRACE("Dentro do construtor CCurso7ADoc \n");
#ifdef _DEBUG
        afxDump.SetDepth(1); // garante o esvaziamento da lista
#endif // _DEBUG
}

CCurso7ADoc::~CCurso7ADoc()
{
```

```
}

BOOL CCurso7ADoc::OnNewDocument()
{
	TRACE("Dentro do CCurso7ADoc::OnNewDocument\n");
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CCurso7ADoc serialization

void CCurso7ADoc::Serialize(CArchive& ar)
{
	TRACE("Dentro do CCurso7ADoc::Serialize\n");

	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}
	m_estudanteList.Serialize(ar);
}

/////////////////////////////////////////////////////////////////////////////
// CCurso7ADoc diagnostics

#ifdef _DEBUG
void CCurso7ADoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CCurso7ADoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
	dc << "\n" << m_estudanteList << "\n";

}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CCurso7ADoc commands

void CCurso7ADoc::DeleteContents()
{
	TRACE("Dentro de CCurso7ADoc::DeleteContents\n");

	// Excluindo os elementos da lista.
	while (m_estudanteList.GetHeadPosition()) {
		delete m_estudanteList.RemoveHead();
	}
}
```

```
BOOL CCurso7ADoc::OnOpenDocument(LPCTSTR lpszPathName)
{
        TRACE("Dentro de CCurso7ADoc:: OnOpenDocument \n");
        if (!CDocument::OnOpenDocument(lpszPathName))
                return FALSE;

        // TODO: Add your specialized creation code here

        return TRUE;
}

void CCurso7ADoc::OnEditLimpartudo()
{
        DeleteContents();               // Esvaziar a lista
        UpdateAllViews(NULL);   // Atualizar todas as janelas
}

void CCurso7ADoc::OnUpdateEditLimpartudo(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(!m_estudanteList.IsEmpty());
}

void CCurso7ADoc::OnUpdateFileSave(CCmdUI* pCmdUI)
{
// Desabilita o botão da barra de ferramentas se o documento não foi alterado
        pCmdUI->Enable(IsModified());
}
```

3.3.3 Classe CCurso7AView

Não há alterações necessárias em relação ao código da classe CCurso6BView, portanto não é necessário listar seu código aqui.

3.4 Testando o Programa Curso7A

Construa o programa e em seguida faça alguns testes digitando dados e salvando-os em disco. Utilize para isso o nome de arquivo Teste.07A .

Neste ponto a janela do aplicativo deve estar parecida com a Figura 55.

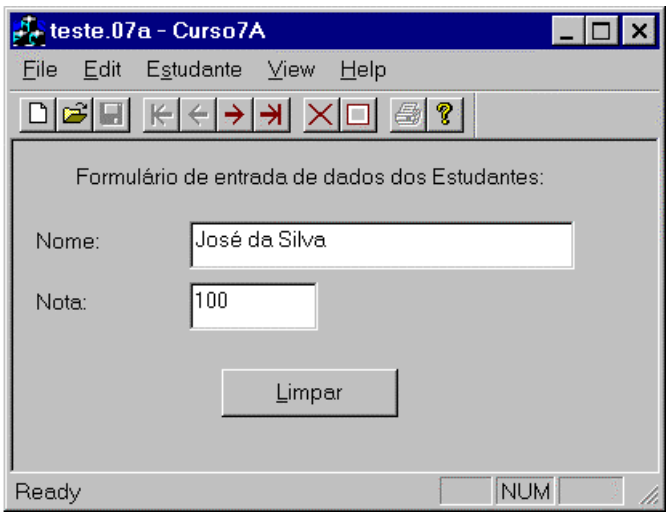


Figura 55 - Tela e execução do programa Curso7A

Encerre o programa e em seguida inicie-o novamente, abrindo o arquivo salvo anteriormente. Os nomes retornaram? E observe as entradas na janela Debug, que nos dão uma pista de como a estrutura das aplicações manuseia o Salvamento e recuperação em disco.

Capítulo 8
Escrevendo e lendo Documentos - MDI

Neste capítulo é introduzido um aplicativo usando a interface de múltiplos documentos (MDI Multiple Document Interface) com o uso da MFC, explicando como serão feitas leituras e gravações nos arquivos de documento. Um aplicativo MDI será efetivamente o estilo de preferência para um programa da biblioteca de classes. O AppWizard permite algumas características específicas aos aplicativos MDI, como o uso do Arrastar e Soltar, e duplo clique no Explorer. Além disso a maior parte dos exemplos de programa que acompanham o Visual C++ são aplicativos MDI.

Vamos abordar ainda neste capítulo as similaridades e também as diferenças entre aplicativos SDI(Single Document Interface) e MDI. Sugere-se que, antes de ser iniciada o estudo dos aplicativos destinados a MDI, haja uma compreensão completa a respeito dos Aplicativos baseados na SDI, conforme descrito no capítulo anterior.

1 Aplicativo MDI

Antes de analisar o código gerado pela biblioteca de classes para os aplicativos MDI, deve haver uma certa familiaridade com a operação de programas do Windows que utilizem a MDI. Observe com cuidado o Developer Studio. Esse é um aplicativo MDI cujos "múltiplos documentos" são arquivos contendo códigos fontes para os programas. Contudo o Developer Studio não representa uma das mais típicas aplicativos MDI, porque seus documentos são reunidos em projetos. Seria mais interessante examinar o Microsoft Word, ou, ainda melhor, uma aplicação real da MDI da biblioteca de classes - do tipo gerado pelo AppWizard.

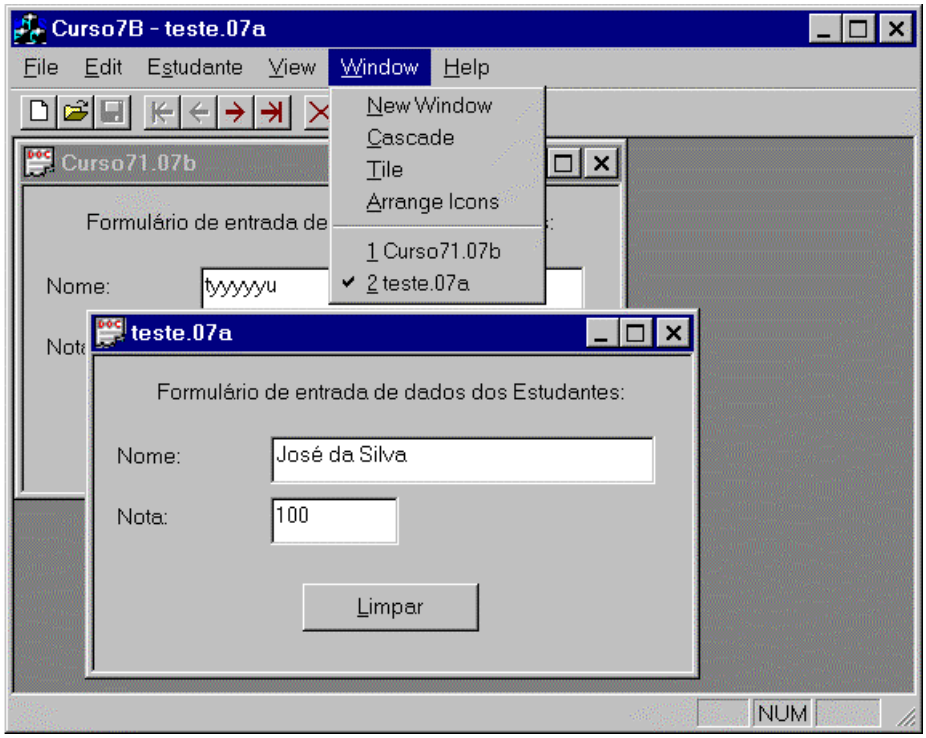


Figura 56 - Típico aplicativo MDI gerado utilizando a MFC

1.1 Típico aplicativo MDI – estilo MFC

O primeiro exemplo deste capítulo Curso8A, é uma versão MDI do programa Curso7A. Consulte a Figura 4 do Capítulo 7, onde é mostrado a ilustração da versão SDI após o usuário ter selecionado um arquivo. Compare-o com o equivalente para a MDI que está mostrado na Figura 56.

O usuário possui abertos dois arquivos diferentes de documento, cada um deles em uma janela filha da MDI separada, mas apenas uma das janelas filhas estará ativa - A janela inferior que está sobre as outras janelas filhas. O aplicativo possui apenas um menu e uma barra de ferramentas com todos os comandos sendo direcionados para a janela filha ativa. A barra de título da janela principal reflete o nome de arquivo de documento presente na janela filha ativa.

A caixa de minimização da janela filha permite que o usuário reduza a mesma para o estado de ícone dentro da janela principal. O menu Window do aplicativo (mostrado na Figura 56) permite ao usuário o controle da apresentação através dos seguintes itens.

Item de Menu	Ação
New Window	Abre uma janela filha para o documento selecionado.
Cascade	Arranja as janelas existentes usando um padrão sobreposto (cascata)
Tile	Arranja as janelas existentes usando um padrão lado a lado (não sobreposto)
Arrange Icons	Arranja as janelas iconizadas na mainframe.
(nomes de documentos)	Seleciona a janela filha correspondente e traz a mesma para a posição de topo

Se o usuário salvar e fechar as duas janela filhas (e abrir o menu file), o aplicativo terá o aspecto da Figura 57

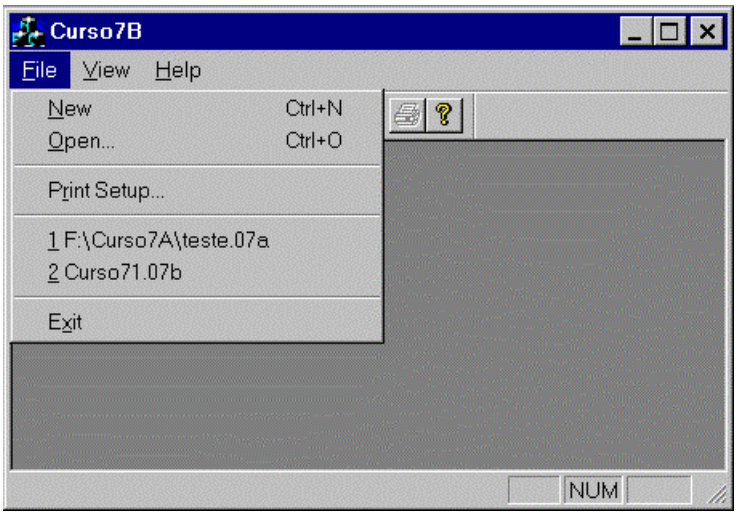


Figura 57 - Programa Curso7B sem janelas filhas

O menu é diferente: a maior parte dos botões existentes na barra de ferramentas se apresenta desativada, e a Barra de título da janela não mostra o nome do arquivo. O usuário praticamente só poderá executar duas ações: o início de um novo documento ou a abertura de um documento já existente no disco.

A Figura 58 mostra o aplicativo logo após a execução ser iniciada sendo criado um novo documento. A única janela filha foi maximizada.

A única janela filha vazia possui o nome básico de documento Curso7. Este nome se baseia no item Doc Type Name selecionado no diálogo classes do AppWizard Curso7. O

primeiro arquivo novo Curso71, o segundo será Curso72 e assim por diante. O usuário normalmente seleciona um nome diferente ao salvar o documento.

Os aplicativos MDI geradas com o uso da MFC, assim como acontece no caso de muitos aplicativos comerciais da MDI, iniciam suas operações com um novo documento vazio. Se for desejado que o aplicativo inicie com a janela mainframe em branco, pode ser alterado o argumento da função *ProcessShellCommand* chamado no arquivo de implementação de classe do aplicativo como mostrado no exemplo Curso8A.

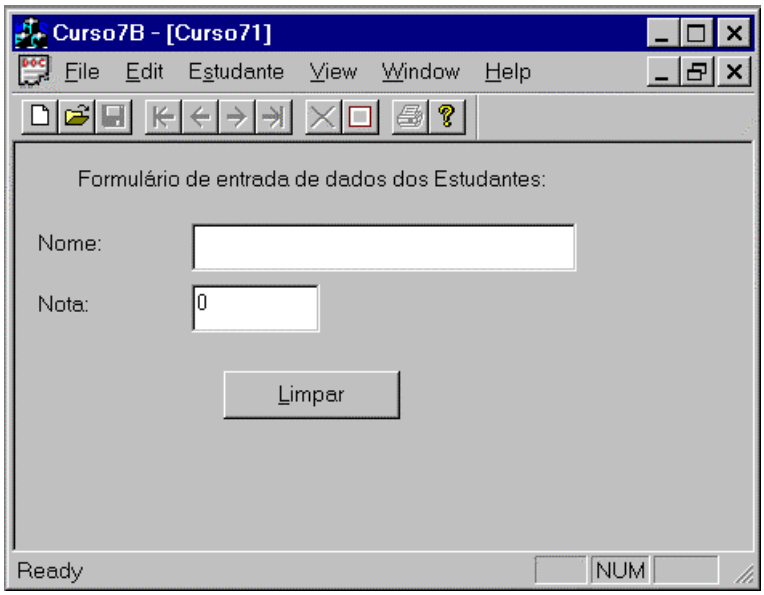


Figura 58 - Programa Curso8A com janela inicial

1.2 O Objeto Aplicativo MDI do Windows.

Você deve estar se perguntando como um aplicativo MDI funciona e qual o código que o diferencia de um aplicativo SDI. Na realidade o processo de partida são muito parecidos. Um objeto Aplicativo de uma classe derivada a partir da classe CWinApp possui uma função membro sobrecarregada InitInstance. Esta função InitInstance é um pouco diferente da função InitInstance correspondente para um aplicativo SDI, iniciando com uma chamada a AddDocTemplate.

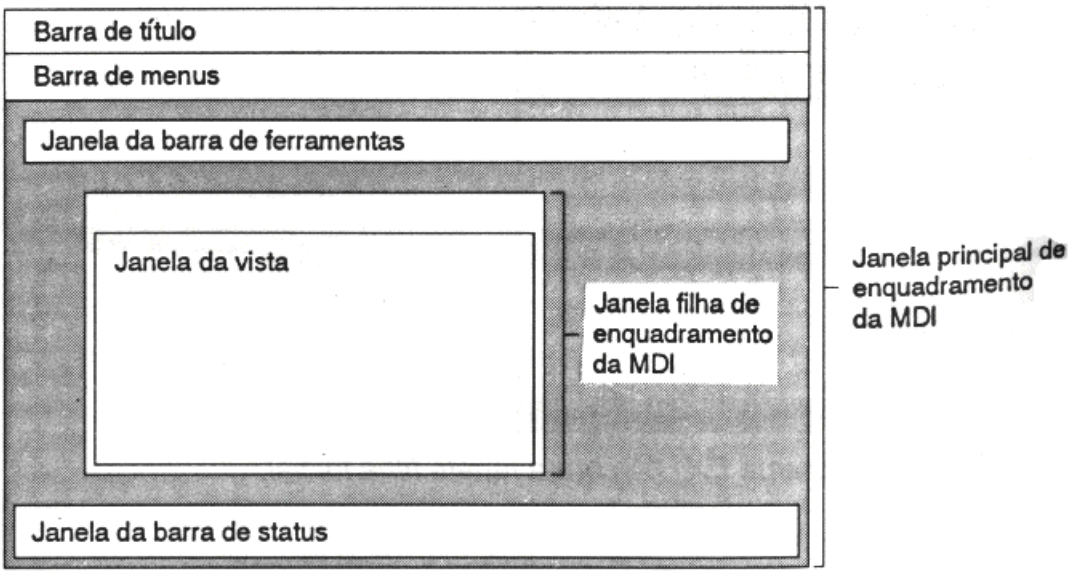


Figura 59 - Relacionamento entre view / frame

1.3 A MDI Frame Window e MDI Child Window

Os exemplos apresentados para SDI possuíam apenas uma classe Frame e apenas um objeto Frame. No caso dos Aplicativos SDI, o AppWizard gerava uma classe CMainFrame a qual era derivada a partir da classe CFrameWnd. Um Aplicativo MDI possui duas classes básicas de frame e muitos objetos de frame (enquadramento) como mostrado na tabela abaixo. O relacionamento entre as janelas view/frame na MDI é mostrado na

Figura 59.

Em um aplicativo SDI, o objeto CMainFrame enquadrava a aplicação e continha o objeto view. Em uma aplicação MDI estes dois papeis estão separados. Agora o Objeto CMainFrame será construído em InitInstance e o objeto CMDIChildWnd irá conter a view.

A estrutura das aplicações pode criar objetos CMDIChildWnd de forma dinâmica, porque a classe CMDIChildWnd será passada para o construtor de CMultDocTemplate.

Classe Base	Classe gerada pelo AppWizard	Número de Objetos	Barra de controle e menus	Possui uma view	Construção do Objeto
CMDIFrameWnd	CMainFrame	Apenas 1	sim	não	Na função InitInstance da classe do aplicativo
CMDIChildWnd	Sem derivação	1 por janela filha	não	sim	Pela Estrutura das Aplicações quando uma nova janela filha for aberta

1.4 Lendo e gravando documentos

Nos aplicativos MDI, os documentos serão lidos e gravados da mesma forma utilizada em aplicativos SDI, mas com duas diferenças importantes: Um novo objeto de documento será construído todas as vezes que um arquivo de documento for carregado do disco com o objeto de documento sendo destruído quando a janela filha for fechada. Não se preocupe em limpar o conteúdo de um documento antes de carrega-lo mas, de qualquer forma, devemos sobrecarregar a função DeleteContents de CDocument para que a classe seja portável para um ambiente SDI.

2 Exemplo Curso8A

O exemplo Curso8A é um aplicativo MDI baseado no exemplo anterior Curso7A. Suas principais diferenças são as capacidades de arrastar e soltar, bem como a abertura de um arquivo através do Explorer. Notar a diferença dentro da função InitInstance da classe CCurso8AApp.

2.1 Criando o Exemplo

Utilizar o AppWizard para criar um novo projeto denominado Curso8A, e seguir os passos mostrados nas figuras a seguir:

- Para o passo 1 utilize a interface para documento MDI (Multiple Document) (Figura 60).
- No passo 4 do AppWizard (Figura 61), selecione o botão Advanced, e quando a tela mostrada na Figura 62, aparecer digite o texto ***08A***, na caixa de edição ***File Extension***.

- No passo 6 do AppWizard (Figura 63), modifique a classe mãe (*Base Class*), para a classe CCurso8AView, fazendo com que a mesma fique sendo derivada de CFormView.
- Na Figura 64, observe as opções de criação para o exemplo Curso8A.

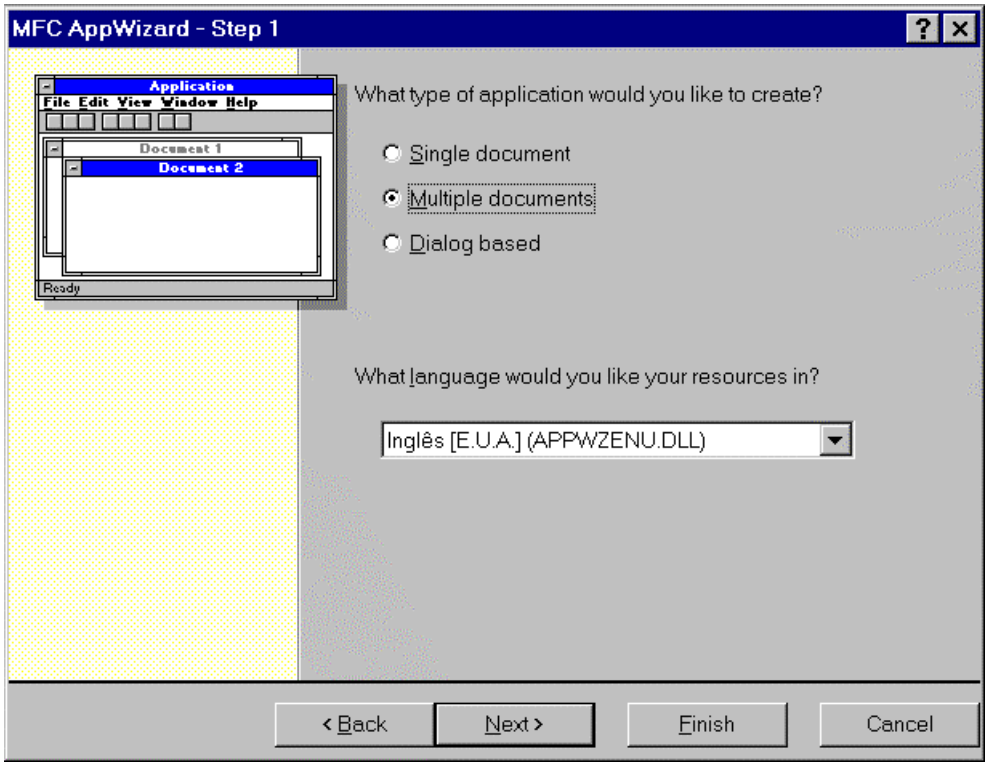


Figura 60 - Passo 1 na criação do exemplo Curso8A

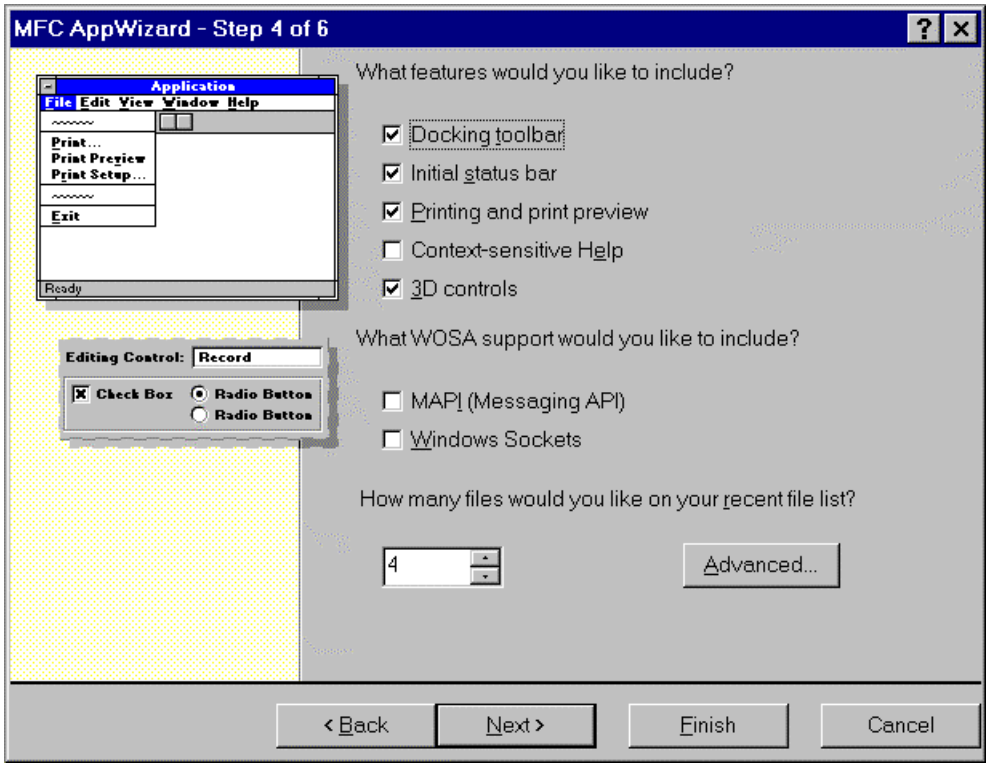


Figura 61 - Passo 4 na criação do exemplo Curso8A

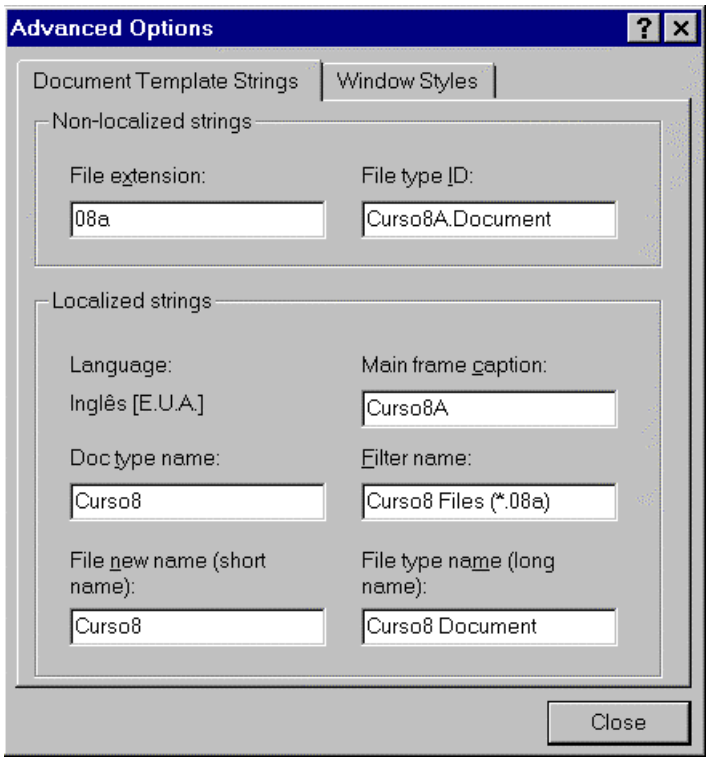


Figura 62 - Tela para seleção das strings para o documento

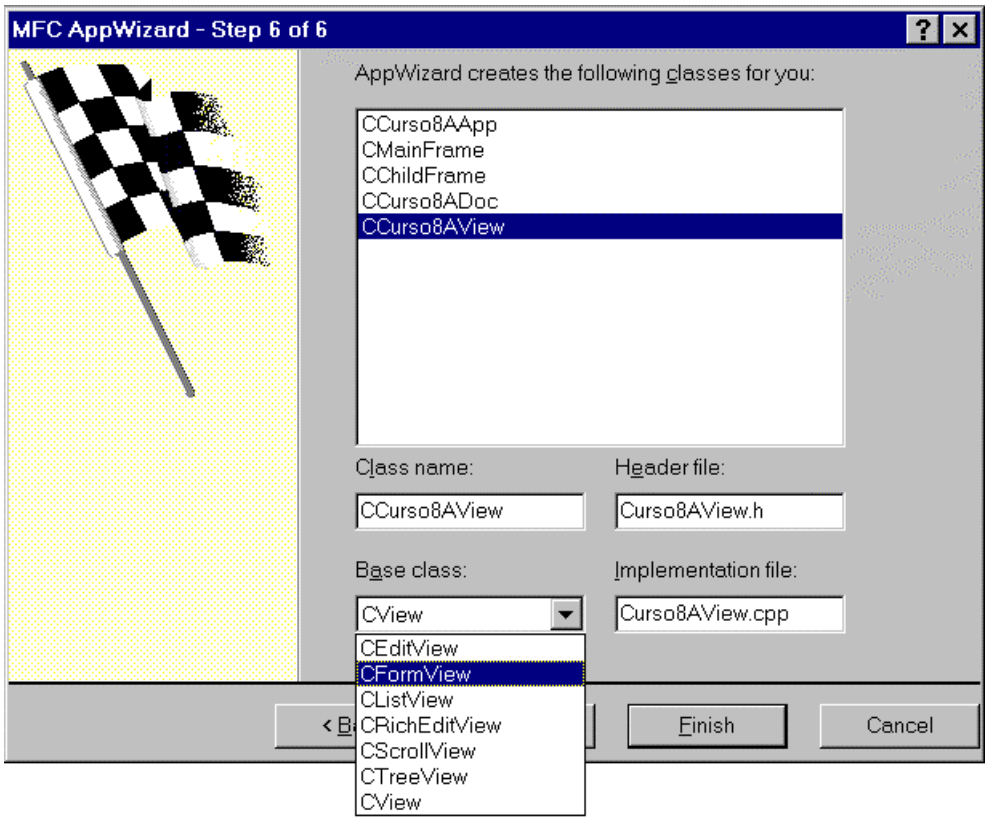


Figura 63 - Passo 6 na criação do exemplo Curso8A

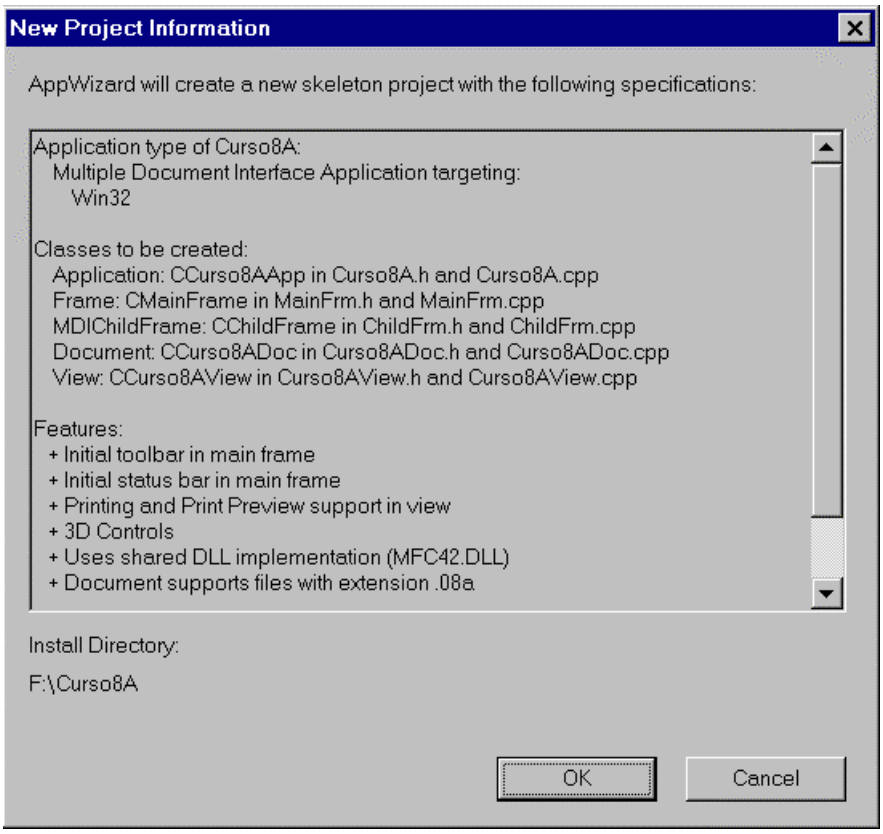


Figura 64 - Opções de criação para o exemplo Curso8A

2.2 Completando o exemplo Curso8A

Após a etapa de criação do exemplo Curso8A, os recursos e classes criados para o exemplo Curso7A podem ser reutilizados para gerar a versão MDI desse exemplo.

Comece importando o menu, em seguida a barra de ferramentas, bem como o layout da tela, ou seja o recurso Dialog. Em seguida, exclua os arquivos dos fontes das classes Curso8AView e Curso8ADoc e copie os arquivos fontes das classes Curso7AView, Curso7ADoc, renomeando-as respectivamente para Curso8AView e Curso8ADoc. A classe CEstudante também deve ser inserida ao exemplo Curso8A, através da cópia dos arquivos fontes para o diretório do exemplo e posterior inserção dos mesmos ao projeto.

Observe que para este exemplo uma classe a mais é gerada pelo AppWizard (CChildFrame).

2.3 Testando o exemplo Curso8A

Quando o programa Curso8A.exe é executado, observe que uma janela filha é criada dentro de sua mainframe.

Quando o menu File New é invocado, uma nova janela aparece sobre a atual, para exibir o novo documento que está sendo criado.

A Figura 65 mostra o Aplicativo Curso8A em atividade.

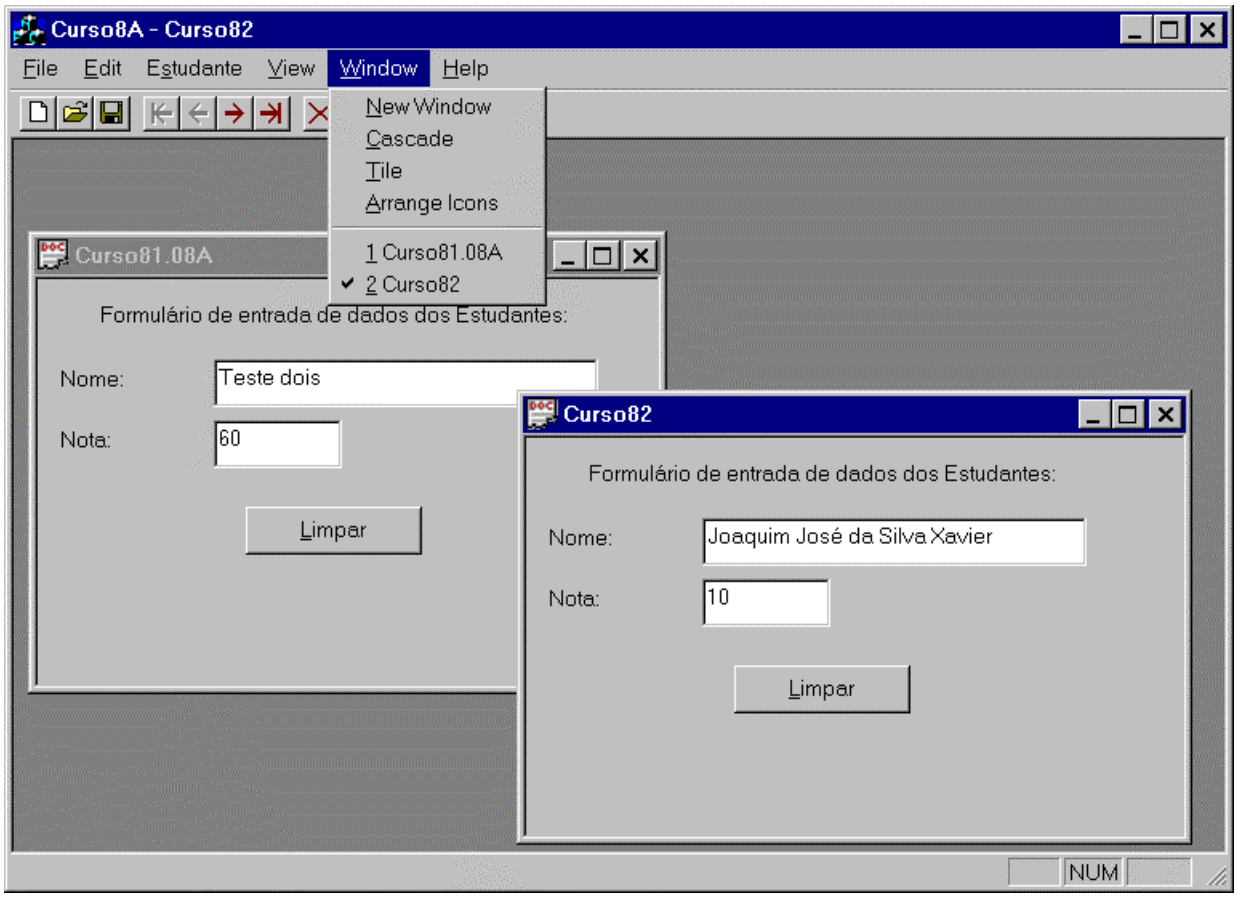


Figura 65 - Tela do Aplicativo em execução

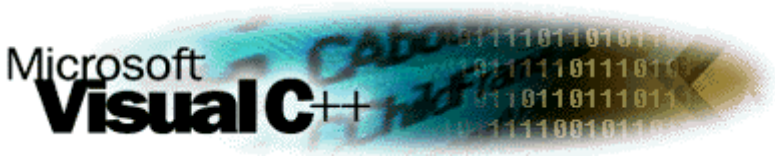
2.4 Janela filha ao iniciar o programa

Como deve ter sido notado, ao inicializar o programa Curso8A, automaticamente uma janela filha é gerada, indicando que um novo documento pode ser alterado. Em alguns programas esta capacidade não é desejável, portanto ao inicializar o programa o mesmo deve ficar esperando para executar alguma ação. Para retirar esta opção do programa Curso8A, A função membro InitInstance() da classe CCurso8AApp deve ser alterada como mostrado a seguir:

```
BOOL CCurso8AApp::InitInstance()
{
        AfxEnableControlContainer();

        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        // Call this when using MFC in a shared DLL
        Enable3dControls();
#else
        // Call this when linking to MFC statically
        Enable3dControlsStatic();
#endif

        // Change the registry key under which our settings are stored.
        // You should modify this string to be something appropriate
        // such as the name of your company or organization.
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));
```

```
        // Load standard INI file options (including MRU)
        LoadStdProfileSettings();

        // Register the application's document templates.  Document templates
        // serve as the connection between documents, frame windows and views.

        CMultiDocTemplate* pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(
                IDR_CURSO7TYPE,
                RUNTIME_CLASS(CCurso8ADoc),
                RUNTIME_CLASS(CChildFrame), // custom MDI child frame
                RUNTIME_CLASS(CCurso8AView));
        AddDocTemplate(pDocTemplate);

        // create main MDI Frame window
        CMainFrame* pMainFrame = new CMainFrame;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
                return FALSE;
        m_pMainWnd = pMainFrame;

        // Enable drag/drop open
        m_pMainWnd->DragAcceptFiles();

        // Enable DDE Execute open
        EnableShellOpen();
        RegisterShellFileTypes(TRUE);

        // Parse command line for standard shell commands, DDE, file open
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);

        // nenhum documento vazio ao iniciar o programa
        if (cmdInfo.m_nShellCommand == CCommandLineInfo::FileNew)
                cmdInfo.m_nShellCommand = CCommandLineInfo::FileNothing;

        // Dispatch commands specified on the command line
        if (!ProcessShellCommand(cmdInfo))
                return FALSE;

        // The main window has been initialized, so show and update it.
        pMainFrame->ShowWindow(m_nCmdShow);
        pMainFrame->UpdateWindow();

        return TRUE;
}
```

Compile e execute o programa novamente para verificar o resultado.



Capítulo 9
Introdução ao uso de banco de dados com a MFC

Neste capítulo é abordado uma introdução a utilização de bancos de dados comerciais, via ODBC (Open Database Connectivity) ou DAO (Data Access Object) utilizando a MFC. As classes que serão utilizadas neste capítulo serão derivadas de uma classe CRecordset (CDaoRecordset) e uma classe CRecordView (CDaoRecordView).

O exemplo Curso9A mostra uma interação entre essas duas classes a ainda serve como um visualizador de banco de dados. Durante sua criação serão mostrados os passos a serem seguidos para criar um aplicativo simples de banco de dados utilizando a MFC.

1 Associando um banco de dados a um projeto.

A associação de uma Tabela/Consulta de um banco de dados pode ser feita utilizando-se o AppWizard, durante o step 2 of 6. Um programa gerado pelo AppWizard com suporte a banco de dados, possuirá itens de menu e botões na barra de ferramentas apropriados para a navegação entre os registro de uma tabela/consulta. A Figura 67 mostra a janela de seleção da base de dados.

2 A Classe CRecordset/CDaoRecordset

Um objeto de CDaoRecordset representa um conjunto de registros selecionados de uma fonte de dados. Conhecidos como "recordsets", objetos de CDaoRecordset estão disponíveis nos três seguintes formatos:

- *Recordsets tipo Table* - representa uma tabela básica que pode ser utilizada para examinar, acrescentar, mudar, ou apagar registros de uma única tabela de banco de dados.
- *Recordsets tipo Dynaset* - é o resultado de uma consulta que pode conter registros atualizáveis. Este recordsets é um conjunto de registros que pode ser utilizado para examinar, acrescentar, mudar, ou apagar registros de tabelas de um banco de dados. Podem conter campos de uma ou mais tabelas de um banco de dados.
- *Recordsets tipo Snapshot* - é uma cópia estática de um conjunto de registros que você pode usar, achar dados ou gerar relatórios. Estes recordsets podem conter campos de uma ou mais tabelas em um banco de dados mas não pode ser atualizado.

Cada formato de recordset representa um conjunto de registros definidos no momento em que o mesmo é aberto. Quando o registro é alterado em recorsets do tipo *Table* ou *Dynaset*, essas mudanças serão refletidas no registro do arquivo depois que o recordset for aberto, ou por outros usuários ou por outro recordsets em sua aplicação. (Um recordset de tipo *Snapshot* não pode ser atualizado.) Você pode usar CDaoRecordset diretamente ou pode derivar um recordset específico para seu aplicativo.

Possuindo um objeto recordset no aplicativo é possível:

- Navegar pelos registros.
- Fixar um índice e procurar registros rapidamente usando Busca (apenas para recordsets de tipo Table).

- Encontrar Registros baseado em uma comparação de strings: "<", "<=", "=", ">=", ou " >" (Apenas para os tipos *Dynaset*, *Snapshot*).
- Atualizar os registros e especificar um modo de travamento de registros (menos recordsets tipo *Snapshot*).
- Filtrar o recordset para restringir quais registros estarão disponíveis na base de dados.
- Ordenar o recordset.
- Parametrizar o recordset para personalizar sua seleção com informações somente disponíveis em tempo de execução do programa.

Classe CDaoRecordset provê uma interface semelhante a da classe CRecordset. A diferença principal é uma classe CDaoRecordset tem acesso aos dados por um Objeto de Acesso de Dados (DAO) baseado em OLE. Classe CRecordset acessa o DBMS por Open Database Connectivity (ODBC) e driver de ODBC para aquele DBMS.

Notar que existem classes distintas na MFC para o acesso a banco de dados via DAO ou ODBC e que as classes de acesso através do DAO levam o prefixo "CDao". As classes DAO geralmente oferecem capacidades superiores porque elas são específicas à Microsoft Jet DB engine.

CDaoRecordset utiliza a DAO record field exchange (DFX) para suportar a leitura e atualização dos campos de um registro para os membros de uma classe CDaoRecordset ou sua derivada.

3 A Classe CRecordView / CDaoRecordView

Um objeto de CDaoRecordView é uma view que exibe registros de banco de dados em controles. Essa view é uma form conectada diretamente um objeto de CDaoRecordset. O layout da janela é criado a partir de um recurso de modelo de diálogo e exibe os campos do objeto CDaoRecordset em seus controles. O objeto CDaoRecordView usa *dialog data exchange* (DDX) e DAO *record field exchange* (DFX) para automatizar o movimento de dados entre os controles no formulário e os campos do recordset. CDaoRecordView também provê um implementação padrão destinada a mover ao primeiro, próximo, prévio, ou último registro e uma interface por atualizar à vista o registro corrente.

Como descrito no item anterior na MFC existem classes distintas para utilizar a interface DAO e ODBC, sendo elas respectivamente uma CDaoRecordView e uma CRecordView. Ambas possuem características semelhantes porém a interface DAO é mais eficiente.

A maneira mais fácil de se criar uma RecordView é através do AppWizard. AppWizard cria a classe RecordView e sua recordset associada como parte do esqueleto de um novo aplicativo, ver mais a frente Exemplo Curso9A.

Para os casos em que apenas um formulário é requerido, o AppWizard oferece o meio mais simples de implementar, entretanto é necessário mais de um formulário para a mesma tabela ou nos casos de várias tabelas, o ClassWizard oferece uma opção mais flexível, pois é possível criar uma nova RecordView e decidir qual Recordset estará ligado a ela no momento de sua criação. Se a classe de vista não for criada pelo o AppWizard, é possível criá-la mais tarde utilizando o ClassWizard. Esta característica também permite ter múltiplas janelas associadas ao mesmo recordset.

A fim de tornar a interface com o usuário mais amigável, e fácil de movimentar por entre os registros o AppWizard cria recursos itens de menu (e toolbar) para mover ao primeiro, próximo, prévio, ou último registro. Se uma classe derivada de CDaoRecordView for criada utilizando o ClassWizard, é necessário também criar estes recursos porém manualmente.

CDaoRecordView mantém registrado a posição do usuário no recordset de forma que a RecorView pode atualizar a interface de usuário. Por exemplo quando o usuário move para o final do recordset, a RecordView desabilita a interface do usuário (as opções de menu e botões na Toolbar) para não permitir que o usuário avance além do final dos registros.

4 Exibindo registro de um banco de dados existente: Exemplo Curso9A

O exemplo Curso9A é um aplicativo SDI com suporte a banco de dados e conectado ao arquivo Estudante.mdb. Suas principais características estão no suporte a banco de dados gerado pelo AppWizard que inclui ícones na barra de ferramentas e item de menu apropriados à navegação entre os registros de uma tabela.

Este exemplo mostra ainda como exibir os registro de um banco de dados em uma CDaoRecordView.

4.1 Criando o Exemplo

Utilizar o AppWizard para criar um novo projeto denominado Curso9A, e seguir os passos mostrados nas figuras a seguir:

- Para o passo 1 utilize a interface para documento SDI (Single Document),
- No passo 2 do AppWizard (Figura 66), seleciona a opção ***Database View Without file support***, e pressione o botão Data Source....
- Selecione a utilização de banco de dados a partir da interface DAO (Figura 67) , e selecione o botão ... para localizar o arquivo que contém os dados a serem exibidos.
- Ao confirmar a localização dos dados fechando o diálogo mostrado na Figura 67, será pedido o nome do conjunto de registros (Tabela/Consulta) que será utilizado como fonte de dados para o programa. (Figura 68),
- Com a base de dados selecionada, o passo 2 do AppWizard aparece como mostrado na Figura 69,
- Notar que no passo 6 do AppWizard (Figura 70), a classe mãe (*Base Class*), para a classe CCurso9AView, já está selecionada como uma CDaoRecordView e que uma classe CCurso9ASet foi criada pelo AppWizard, com a referência de onde os dados serão obtidos.
- Na Figura 71, observe as opções de criação para o exemplo Curso9A.

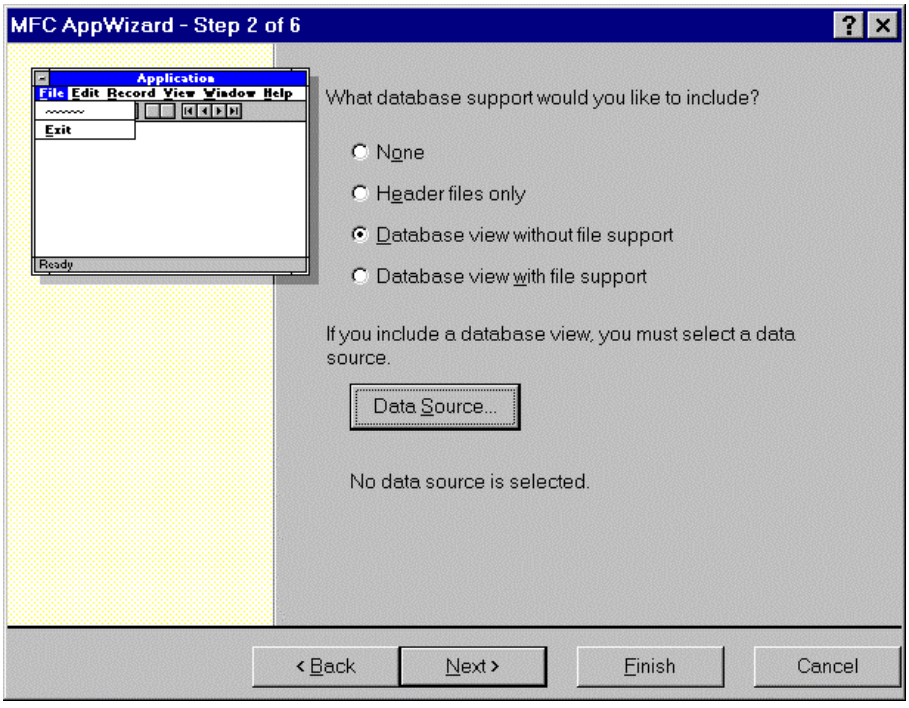


Figura 66 - Passo 2 na criação do exemplo Curso9A

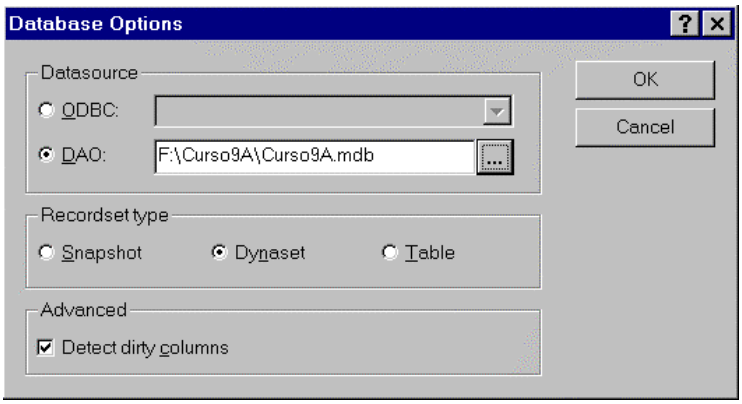


Figura 67 - Diálogo para seleção do tipo de suporte a banco de dados a ser utilizado.

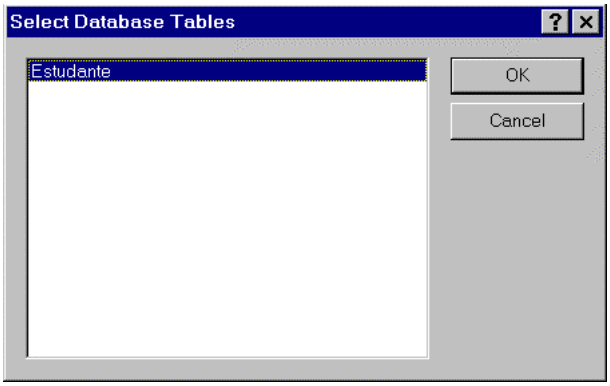


Figura 68 - Diálogo para seleção da tabela/consulta onde os dados estão armazenados

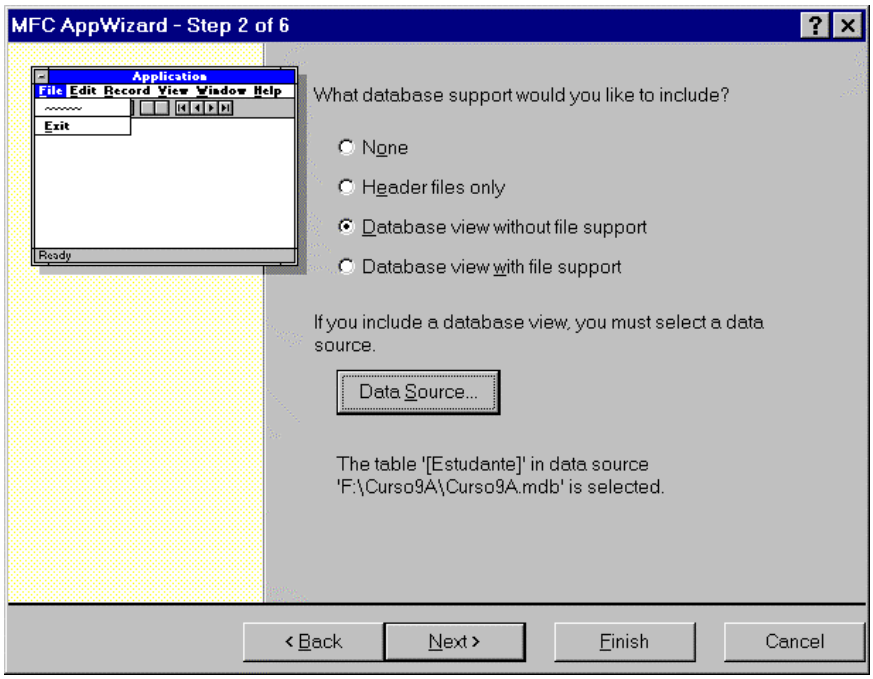


Figura 69 - Passo 2 do AppWizard mostrando a base de dados selecionada

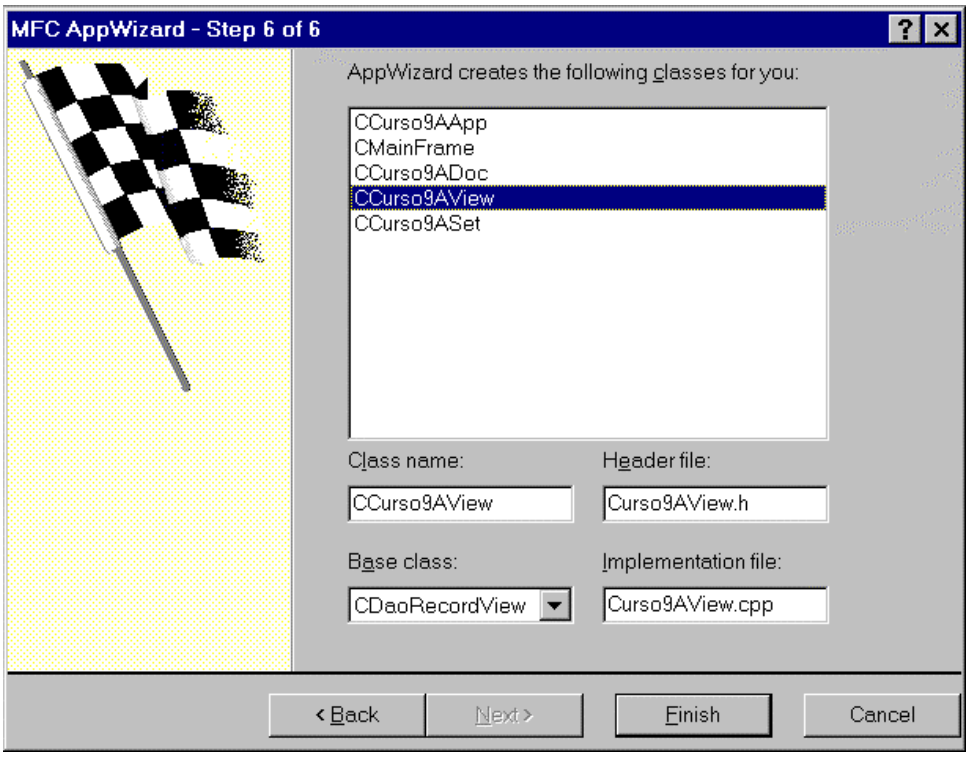


Figura 70 - Passo 6 do AppWizard para a criação do exemplo Curso9A

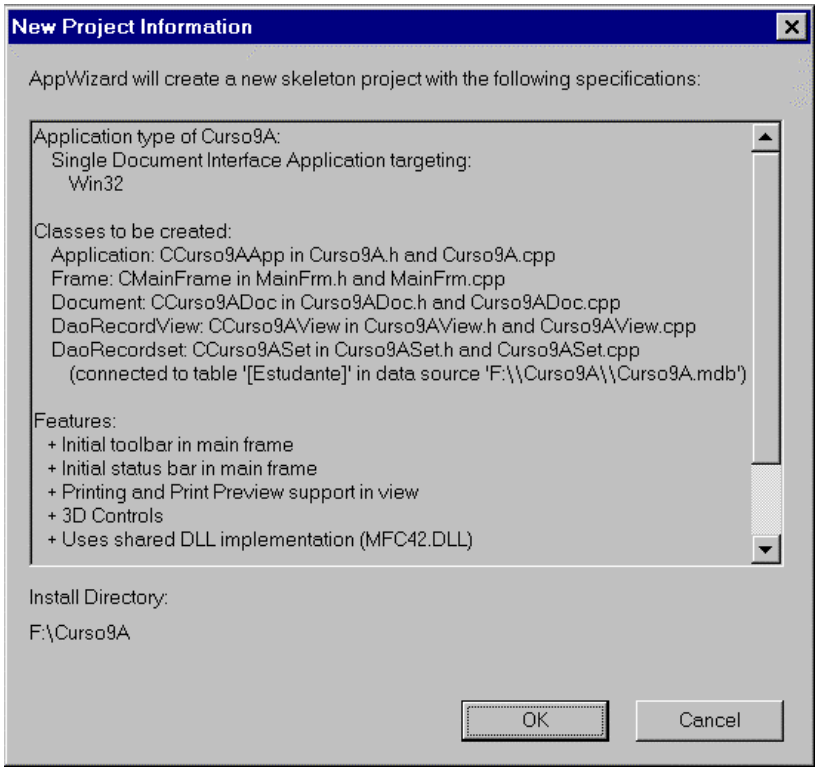


Figura 71 - Opções de criação do exemplo Curso9A

4.2 Completando o exemplo Curso9A

4.2.1 Recursos:

Modifique o formulário IDD_CURSO9A_FORM para torná-lo igual ao da figura abaixo, lembrando que cada caixa de edição (Edit Box) deve possuir um identificador significativo os seu conteúdo, Ex IDC_NOME e IDC_NOTA.

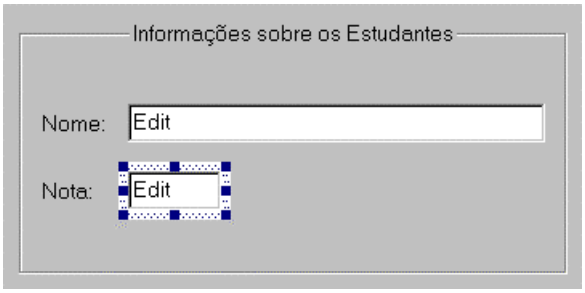


Figura 72 - Formulário para exibir dados do estudante.

4.2.2 Código:

Utilize o ClassWizard para conectar as caixas de edição aos campos correspondentes do banco de dados a ser exibido.

Selecione as variáveis associadas (Figura 73) a tabela de dados para cada um dos identificadores correspondentes as caixas de edição criadas no formulário. Observe que a variável criada é um ponteiro para um membro da classe Set, e na lista aparecem com um indicador -> antes de seu nome.(Figura 74)

Obs: Variáveis que existem apenas temporariamente, ou seja não fazem parte da tabela ou é um resultado entre elementos da tabela podem ser ainda podem ser criadas diretamente dentro da classe View, o que permite que parte dos dados exibidos na tela sejam resultados de cálculos e não necessariamente membros de uma tabela.

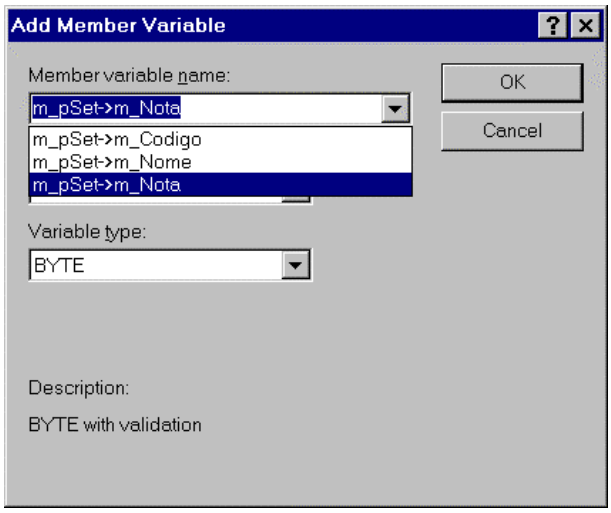


Figura 73 - Seleção de variável para a caixa de edição

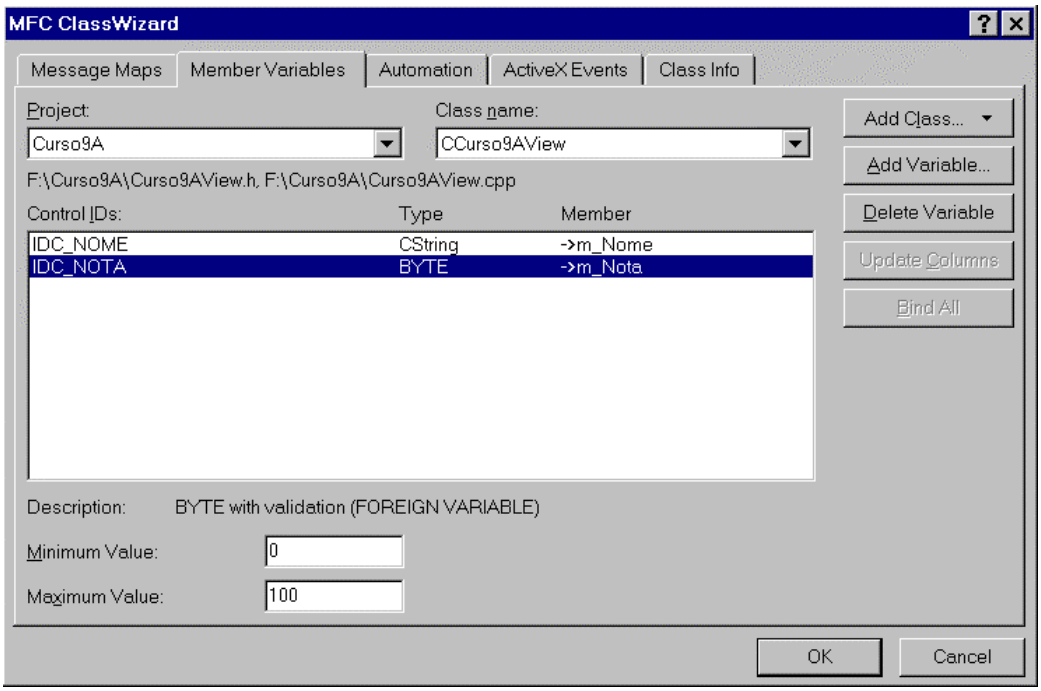


Figura 74 - Pasta Member Variables do ClassWizard

4.3 Testando o exemplo Curso9A

Compile e execute o programa Curso9A.exe. Navegue entre os registro e verifique agora que o controle dos botões da barra de ferramentas não precisa mais ser feito dentro da classe CCurso9AView, pois dentro de sua classe Base ele já é implementado.

Observe também que não foi necessário escrever nenhuma linha de código para que o programa gerado pelo AppWizard fosse capaz de exibir os dados contidos em uma tabela de um banco de dados comercial.

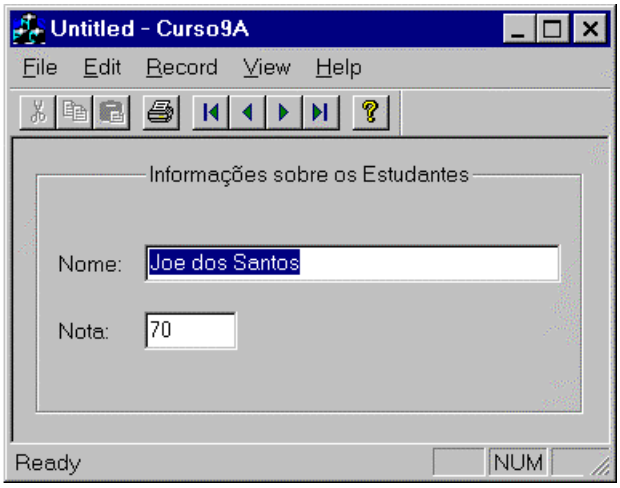


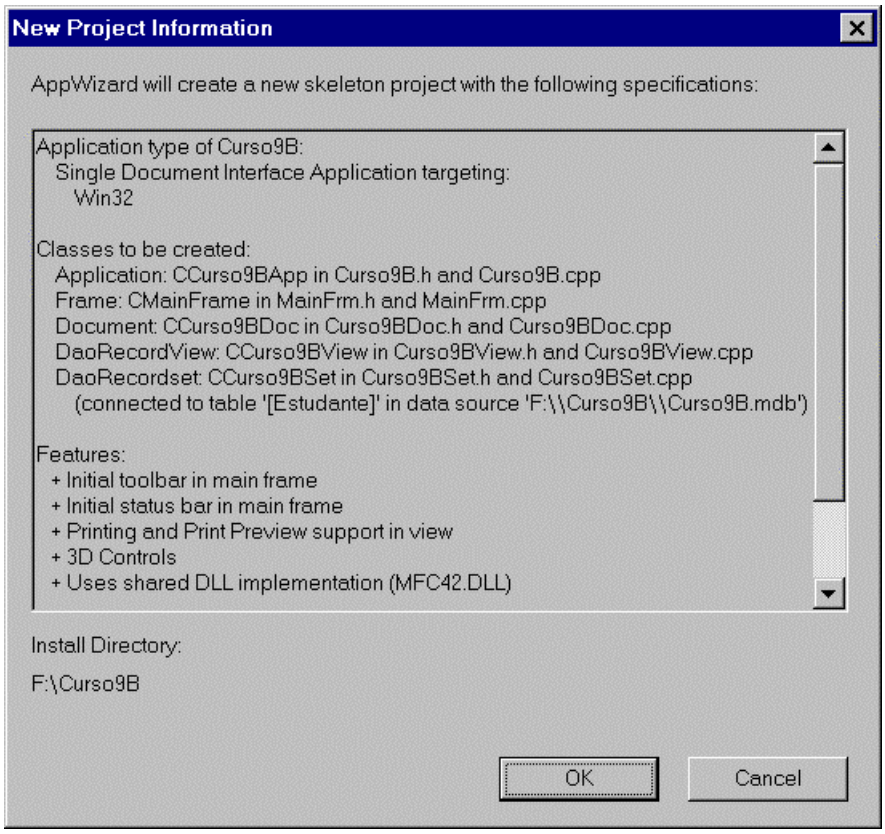
Figura 75 - Execução do programa Curso9A.exe

5 Alterando os registros a um banco de dados existente: Exemplo Curso9B

O exemplo Curso9B aproveita todas as características do exemplo Curso9A e acrescenta a ele a capacidade de incluir novos registros e excluir registros existentes.

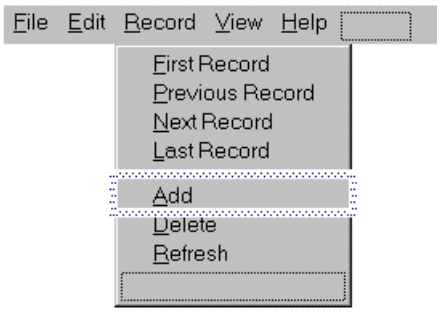
5.1 Criando o Exemplo

Siga os passos descritos para a criação do exemplo curso9A e acrescente os seguinte itens:



5.1.1 Passo 1 – Utilize o editor de menus para acrescentar opções ao menu Record:

- Selecionar o item *Menu* do arquivo de recursos e escolher o identificador IDR_MAINFRAME (duplo-clique);
- Selecionar o submenu Record;
- Acrescentar os itens de menu como mostrado na figura abaixo;



- As propriedades de cada um dos itens que serão adicionados devem estar de acordo com a tabela descrita a seguir. Obs.: os identificadores são criados automaticamente quando se acrescenta um Capition ao item de menu.

Identificador	Capition	Prompt
ID_RECORD_ADD	&Add	Adicionar novos registros ao Banco de dados.\nAdicionar
ID_RECORD_DELETE	&Delete	Excluir o registro atual.\nExcluir
ID_RECORD_REFRESH	&Refresh	Desfazer ultimas alterações.\nDesfazer

5.1.2 Passo 2 – Acrescentar a classe CAddForm ao projeto:

Incluir os arquivos AddForm.h e AddForm.cpp ao projeto.

Listagem do arquivo AddForm.h

```
// addform.h : interface of the CAddForm class
//
/////////////////////////////////////////////////////////////////////////////
#if !defined (AFX_ADDFORM_INCLUDED)
#define AFX_ADDFORM_INCLUDED

class CAddForm : public CDaoRecordView
{
protected:
        CAddForm(UINT nIDTemplate);
        DECLARE_DYNAMIC(CAddForm)

public:
        BOOL m_bAddMode;
        //{{AFX_DATA(CAddForm)
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

// Attributes
protected:

// Operations
public:
        virtual BOOL OnMove(UINT nIDMoveCommand);
        virtual BOOL RecordAdd();
        virtual BOOL RecordRefresh();
        virtual BOOL RecordDelete();

// Implementation
public:
        BOOL Gravar();
        virtual ~CAddForm();

// Generated message map functions
protected:
        //{{AFX_MSG(CAddForm)
        afx_msg void OnRecordAdd();
        afx_msg void OnRecordRefresh();
        afx_msg void OnRecordDelete();
        afx_msg void OnUpdateRecordFirst(CCmdUI* pCmdUI);
        //}}AFX_MSG

        //{{AFX_VIRTUAL(CAddForm)
        virtual void OnInitialUpdate(); // called first time after construct
        //}}AFX_VIRTUAL

        DECLARE_MESSAGE_MAP()
};

#endif // !defined(AFX_ADDFORM_INCLUDED)
```

Listagem do arquivo AddForm.cpp

```
// addform.cpp : implementation of the CAddForm class
//
```

```
#include "stdafx.h"
#include "Curso9B.h"    //  OBs: deve ser o include do projeto
#include "addform.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNAMIC(CAddForm, CDaoRecordView)

BEGIN_MESSAGE_MAP(CAddForm, CDaoRecordView)
        //{{AFX_MSG_MAP(CAddForm)
                ON_COMMAND(ID_RECORD_ADD, OnRecordAdd)
                ON_COMMAND(ID_RECORD_REFRESH, OnRecordRefresh)
                ON_COMMAND(ID_RECORD_DELETE, OnRecordDelete)
        ON_UPDATE_COMMAND_UI(ID_RECORD_FIRST, OnUpdateRecordFirst)
        //}}AFX_MSG_MAP

END_MESSAGE_MAP()

CAddForm::CAddForm(UINT nIDTemplate)
        : CDaoRecordView(nIDTemplate)
{
        //{{AFX_DATA_INIT(CAddForm)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
        m_bAddMode = FALSE;
}

CAddForm::~CAddForm()
{

}

BOOL CAddForm::OnMove(UINT nIDMoveCommand)
{
        CDaoRecordset* pRecordset = OnGetRecordset();
        if (m_bAddMode)
        {
                if (!UpdateData())
                        return FALSE;
                try
                {
                        pRecordset->Update();
                }
                catch (CDaoException* e)
                {
                        e->ReportError();
                        e->Delete();
                        return FALSE;
                }

                pRecordset->Requery();
                UpdateData(FALSE);
                m_bAddMode = FALSE;
                return TRUE;
        }
        else
        {
                return CDaoRecordView::OnMove(nIDMoveCommand);
        }
```

```
}

BOOL CAddForm::RecordAdd()
{
	// If already in add mode, then complete previous new record
	if (m_bAddMode)
		OnMove(ID_RECORD_FIRST);
	OnGetRecordset()->AddNew();
	m_bAddMode = TRUE;
	UpdateData(FALSE);
	return TRUE;
}

BOOL CAddForm::RecordDelete()
{
	if (IDYES==AfxMessageBox("Você tem certeza que deseja prosseguir a
exclusão?", MB_ICONQUESTION|MB_YESNO))
	{
		CDaoRecordset* pRecordset = OnGetRecordset();
		try
		{
			pRecordset->Delete();
		}
		catch (CDaoException* e)
		{
			e->ReportError();
			e->Delete();
			return FALSE;
		}

	// Move to the next record after the one just deleted
			pRecordset->MoveNext();

	// If we moved off the end of file, then move back to last record
		if (pRecordset->IsEOF())
			try
			{
				pRecordset->MoveLast();
			}
			catch (CDaoException* e)
			{
				pRecordset->SetFieldNull(NULL);
				e->ReportError();
				e->Delete();
				return FALSE;
			}

		// If the recordset is now empty, then clear the fields
		// left over from the deleted record
		if (pRecordset->IsBOF())
		{
			pRecordset->SetFieldNull(NULL);
		}

		UpdateData(FALSE);
	}
	return TRUE;
}


BOOL CAddForm::RecordRefresh()
{
	if (m_bAddMode == TRUE)
```

```
        {
                OnGetRecordset()->CancelUpdate();
                OnGetRecordset()->Move(0);
                m_bAddMode = FALSE;
        }
        // Copy fields from recordset to form, thus
        // overwriting any changes user may have made
        // on the form
        UpdateData(FALSE);

        return TRUE;
}

void CAddForm::OnRecordAdd()
{
        RecordAdd();
}

void CAddForm::OnUpdateRecordFirst(CCmdUI* pCmdUI)
{
        if (m_bAddMode)
                pCmdUI->Enable(TRUE);
        else
                CDaoRecordView::OnUpdateRecordFirst(pCmdUI);
}

void CAddForm::OnRecordRefresh()
{
        RecordRefresh();
}

void CAddForm::OnRecordDelete()
{
        RecordDelete();
}

BOOL CAddForm::Gravar()
{
        CDaoRecordset* pRecordset = OnGetRecordset();
        if (m_bAddMode)
        {
                OnMove(ID_RECORD_LAST);
                return  OnMove(ID_RECORD_LAST);
        }

        if (pRecordset->IsBOF() || pRecordset->IsEOF())
                return FALSE;

        pRecordset->Edit();
        if (!UpdateData())
                return FALSE;
        try
        {
                pRecordset->Update();
        }
        catch (CDaoException* e)
        {
                e->ReportError();
                e->Delete();
                return FALSE;
        }
        return TRUE;
}
```

```
void CAddForm::OnInitialUpdate()
{
        CDaoRecordView::OnInitialUpdate();

}
```

5.1.3 Passo 3 – Tornar a classe CCurso9BView derivada de CAddForm:

No arquivo de descrição da classe Curso9BView.h alterar as linhas como mostrado abaixo.

```
#include "addform.h"

class CCurso9BView : public CAddForm
{
```

No arquivo de Implementação da classe Curso9BView.h , trocar todas as ocorrencias de ***CDaoRecordView*** por ***CAddForm***.

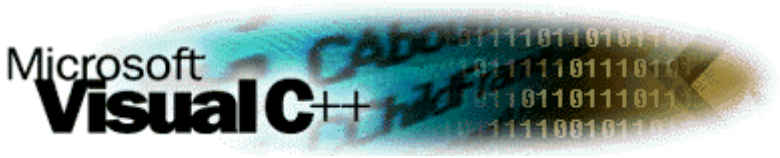
5.2 Testando o exemplo Curso9B

Compile e execute o programa Curso9B.exe.

Utilize os item da barra de ferramentas para inserir um novo registro bem como para tentar excluir um registro previamente existente.

5.3 Exercício Proposto

Adicionar botões a barra de ferramentas para os Itens de menu *Record Add*, *Record Delete* e *Record Refresh*.



Trabalho de fim de Curso

Projetar um programa com interface MDI para cadastrar uma lista de alunos, com as suas respectivas notas obtidas nos bimestres, calcular a média, verificar se o estudante ficou de prova final, calcular a nova média e verificar se o estudante foi aprovado. As listas de alunos devem ser gravadas em disco com um arquivo que seja "NomeDisciplina.10A".

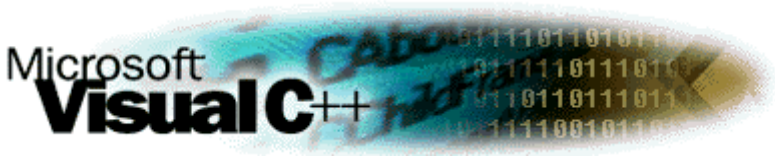
1 Características adicionais ao programa proposto:

O programa a ser criado deve possuir pelo menos as características listadas abaixo:

1. Criar uma opção de menu para calcular a nota necessária para o aluno não ficar de exame ou não ser reprovado. E exibir resultado em uma caixa de diálogo ou controle específico para este fim.
2. Alterar o diálogo Sobre, para acrescentar referências ao trabalho que está sendo criado como por exemplo o Nome do Programador, entre outras.
3. Acrescentar opção para alteração dos dados.
4. Exibir diálogo para confirmação de exclusão de um registro da lista.

2 Observações

1. As médias devem ser calculadas utilizando as fórmulas que são obtidas no manual do aluno.
2. Utilizar os exemplos do curso como base para a criação desse programa.
3. Gerar uma versão Release do programa para entrega.



Bibliografia

1 Bibliografia consultada

Os seguintes referência foram utilizadas como base para a construção dessa apostila:

BARKAKATI, : Visual C++, Makron Books

HOLZNEI, S.: Programando com Visual C++, LTC Editora, 1993.

KRUGLINSKI, D. J.: Explorando o Visual C++, Editora Campus.

KRUGLINSKI, D. J.: Inside Visual C++, 4ª Ed, Microsoft Press , 1997.

WIENER, R. S.: C++ Programação Orientada para Objeto. Manual Prático e Profissional, Makron Books 1991

BERNARDI, A. : Notas de aula da Disciplina de Programação Avançada. FEPI, Itajubá 1998.