UENF/CCT/LENEP

Setor de Modelagem Matemática Computacional

Projeto 1 - Métodos Numéricos - Integração Numérica

Disciplinas: Fundamentos de Programação C++;
Programação Orientada a Objeto com C++; Processamento Paralelo

André Duarte Bueno, Dr. Eng.

22 de fevereiro de 2021

Capítulo 1

Projeto 01 - Integração Numérica - Introdução

Apresenta-se neste capítulo um projeto de software completo com fins puramente educativos.

- O que é?
  - Um simulador para cálculo da área de funções matemáticas.
- Didática:
  - Iremos mostrar diferentes versões do programa, evoluindo de uma versão bem simples e específica, para uma versão mais completa e geral.
  - São apresentadas versões com solução direta numa única função, versões que usam paradigma de programação estruturada, depois orientada a objeto e depois acrescenta-se o uso de recursos de programação funcional.
  - Também iremos apresentar o uso de conceitos de C++11/14/17/20, mas apenas os realmente necessários.

1.1 Especificação (simplificada)

- Objetivo:
  - Desenvolver um simulador de engenharia para cálculo da área de funções matemáticas do tipo $y = f(x)$ usando os métodos do Trapézio (seção 1.2.2) e de Simpson (seção 1.2.3).
- Interface:
  - Modo texto.
- Ítens opcionais:
  - Resultados serão armazenados em disco.
  - Poderão ser gerados gráficos.

1.2 Elaboração - Métodos de Integração Numérica - Trapézio e Simpson

A estapa de elaboração consiste em estudar o problema a ser resolvido. Então, vamos apresentar brevemente o conceito matemático de integral e os métodos de integração numérica do trapézio e Simpson.

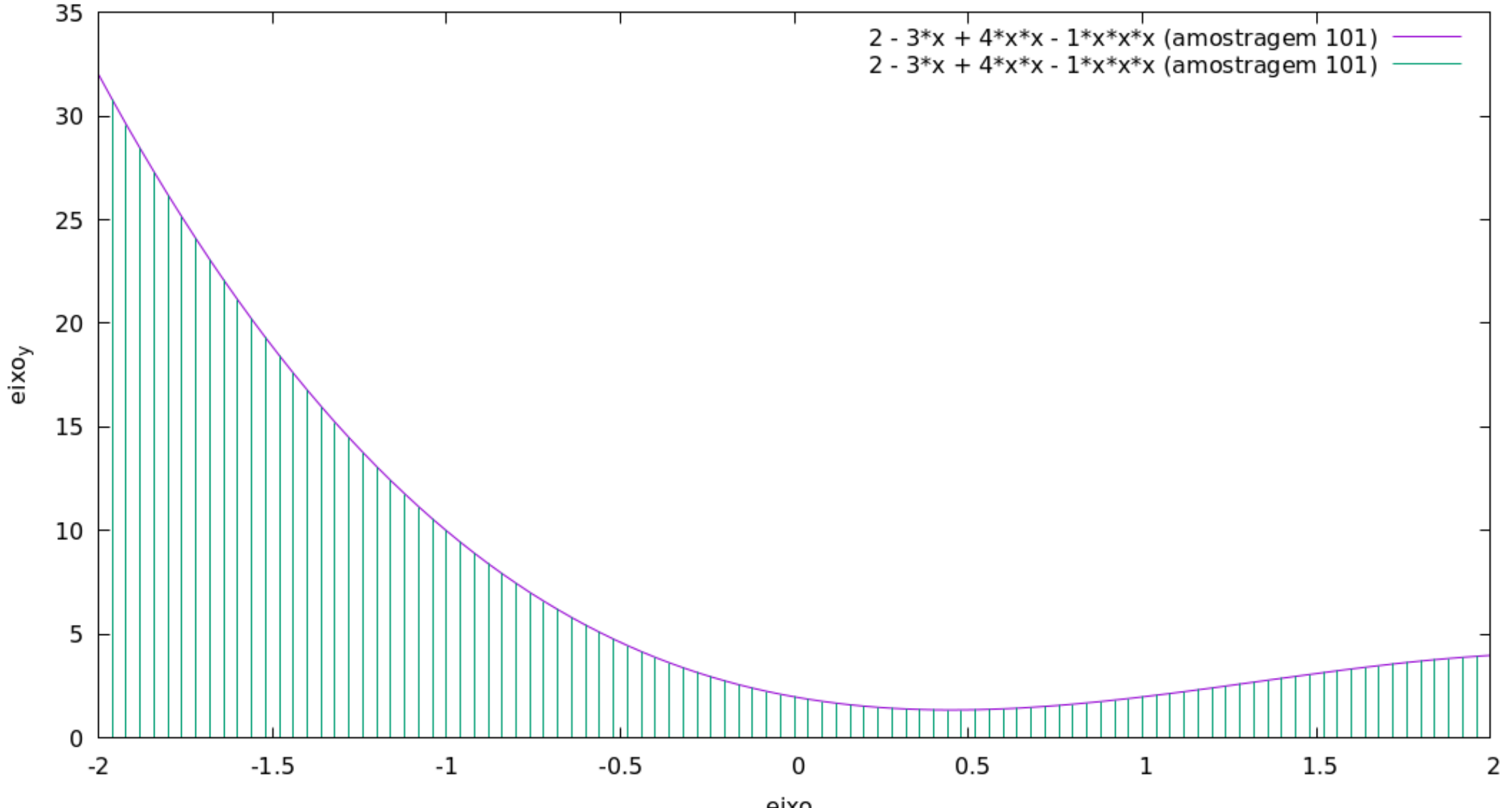
- O conceito de integral é visto em disciplinas de cálculo, normalmente "Cálculo I" nos cursos de engenharia. Será muito brevemente descrito na seção 1.2.1. Maiores detalhes podem ser obtidos em livros de cálculo e na internet.
- O conceito de integral numérica e o método da integral do trapézio é apresentado a seção 1.2.2.
- O método da integral de Simpson é apresentado a seção 1.2.3.

1.2.1 Cálculo da Área (Integral)

A Figura 1.1 mostra a plotagem da função matemática $y = 2 - 3x + 4x^2 - x^3$ no intervalo de [-2,2]. A Figura foi gerada utilizando o software `gnuplot` com uma amostragem de 101 pontos. Note na legenda que plotamos a curva duas vezes, a primeira para mostrar a linha roxa e a segunda as linhas verdes. Veja a seguir o comando para reproduzir a plotagem no `gnuplot`.

```
set xlabel "eixo\_x"
set ylabel "eixo\_y"
plot [-2:2] 2 - 3*x + 4*x*x -1*x*x*x with lines \
title "2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)", \
2 - 3*x + 4*x*x -1*x*x*x with impulses \
title "2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)"
```

Os valores no eixo horizontal, eixo x, são as variáveis independentes (os números que fornecemos). Os valores no eixo vertical, eixo y, são as variáveis dependentes, ou seja, os números que são calculados por uma equação que é função de x, dependem de x.


2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)
2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)
eixo$_y$
eixo
35
30
25
20
15
10
5
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Matematicamente podemos ter diferentes funções, as mais utilizadas em engenharia são apresentadas a seguir.
A equação do primeiro (eq: 1.1),

$$y = a + b * x, \tag{1.1}$$

do segundo (eq: 1.2)

$$y = a + b * x + c * x^2, \tag{1.2}$$

e do terceiro grau (eq: 1.3):

$$y = a + b * x + c * x^2 + d * x^3. \tag{1.3}$$

De forma genérica podemos escrever:

$$y = f(x), \tag{1.4}$$

indicando que o valor de y depende do valor de x. Por isso x é chamado independente e y dependente.

Em termos práticos, para a física e a engenharia, é muito interessante saber calcular a área abaixo da curva. Para resolver este problema, ao longo dos séculos, foram desenvolvidas várias metodologias de cálculo (veja [Bassalo 1996]).

A mais exata e geral é o cálculo da integral indefinida, cuja expressão formal é dada por:

$$área = \int f(x)dx \tag{1.5}$$

Note que na integral indefinida não existem limites e o resultado é uma equação genérica, ou seja, dado um intervalo qualquer consigo calcular a área.

Na prática da engenharia o mais usual é a integral definida, em que os limites de integração, isto é, os intervalos, de interesse são definidos. Então, a equação é reescrita com a definição dos intervalos li - limite inferior e ls - limite superior.

$$área = \int_{li}^{ls} f(x)dx \tag{1.6}$$

No exemplo da Figura 1.1 os limites são $li = -2$ e $ls = 2$.

O resultado do cálculo desta integral é o valor exato da área abaixo da curva no intervalo especificado. Para entender em detalhes o formalismo matemático associado sugere-se a consulta a livros de Calculo Diferencial e Integral.

1.2.2 Cálculo da Área Aproximada Pelo Método do Trapézio

De um modo geral um físico teórico e um pesquisador/cientista esta preocupado com o valor exato das equações. Este também é um interesse do engenheiro, mas, por questões práticas, o engenheiro aceita resultados com valores aproximados.

Uma maneira de realizar esta aproximação no cálculo da área é utilizar trapézios, conforme ilustrado na Figura 1.2. Note que é feita uma aproximação da área da curva utilizando-se diversos trapézios. A ideia é calcular a área de cada trapézio individualmente e depois somar a área de todos eles.

Neste exemplo temos uma amostragem 5 (curva em roxo) e uma amostragem 101 (curva em vermelho). Na curva em roxo conseguimos identificar que temos 4 trapézios, o primeiro é ilustrado a esquerda na cor verde. Note que para um valor de $x = -2$ o valor de $y = f(-2) = 32.0$, e que para um valor de $x = -1$ obtemos $y = f(-1) = 10.0$. A área deste trapézio é dada por $área = 1 * (f(-2) + f(-1))/2 = (32 + 10)/2 = 21$. Se calcularmos a área de cada trapézio e somarmos, teremos uma aproximação da área da curva. Esta é a ideia do método do trapézio.

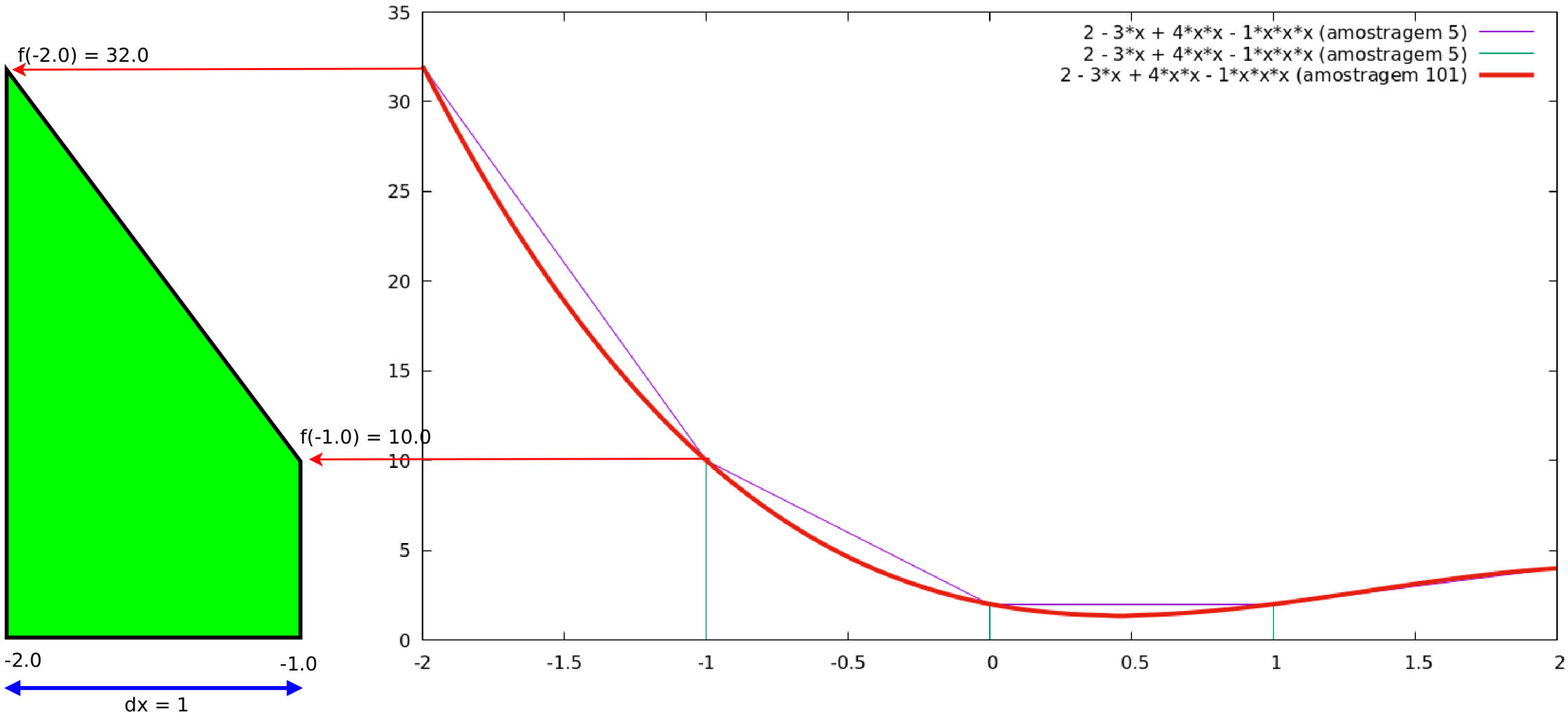


Figura 1.2: Esboço de uma função em vermelho e em azul ilustração do Método do Trapézio

Note que a área do primeiro trapézio é definida pela seguinte equação.

$$AreaTrapézio\_1 = \frac{f(x_1) + f(x_2)}{2} * dx \tag{1.7}$$

Observe que calculamos $f(x_1)$ e $f(x_2)$.

A área do segundo trapézio é definida pela seguinte equação.

$$AreaTrapézio\_2 = \frac{f(x_2) + f(x_3)}{2} * dx \tag{1.8}$$

Note que calculamos $f(x_2)$ e $f(x_3)$. Ou seja, os extremos como $f(x_1)$ e $f(x_n)$ são calculados apenas uma vez, e os valores intermediários, como $f(x_2)$ e $f(x_3)...f(x_{n-1})$, são calculados sempre duas vezes.

A área dos demais trapézios pode ser generalizada por:

$$AreaTrapézio\_i = \frac{f(x_i) + f(x_{i+1})}{2} * dx \tag{1.9}$$

Se somarmos a área de todos estes trapézios teremos a equação abaixo, que é a equação para cálculo da área aproximada da curva pelo método do trapézio:

$$\int_{li}^{ls} f(x)dx \approx \frac{dx}{2}\left[f(x_1) + 2f(x_2) + 2f(x_3) + 2f(x_4) + ... + 2f(x_{n-1}) + f(x_n)\right] \tag{1.10}$$

Podendo ser reescrita da forma

$$\int_{li}^{ls} f(x)dx \approx dx\left[f(x_1)/2 + f(x_2) + f(x_3) + f(x_4) + ... + f(x_{n-1}) + f(x_n)/2\right] \tag{1.11}$$

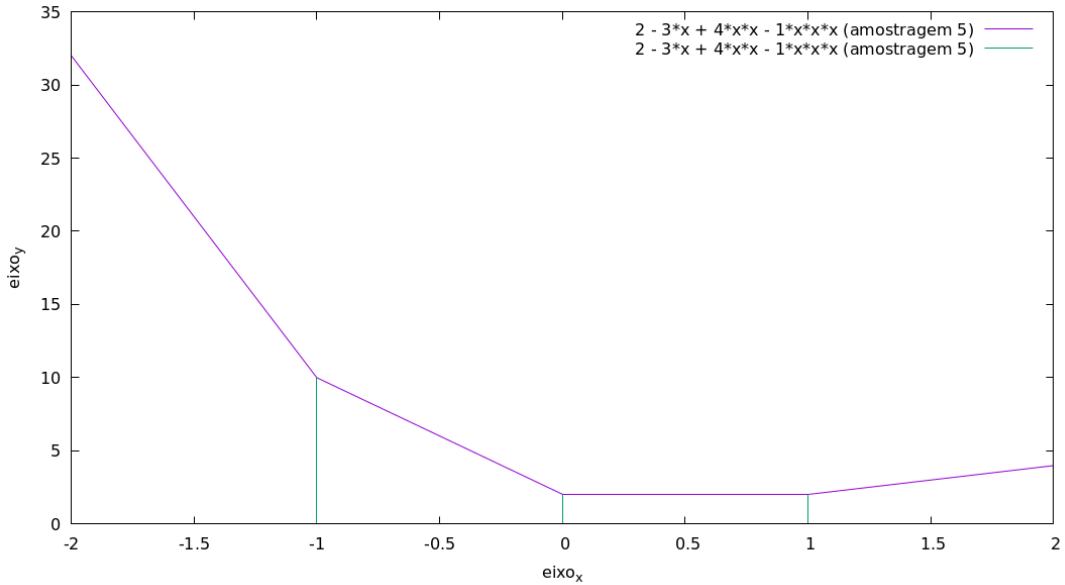
A qualidade ou precisão deste cálculo esta diretamente associada ao valor de dx, se for grande estaremos errando mais, se for pequeno estaremos nos aproximando do valor exato. Se dx for um valor infinitamente pequeno estaremos convergindo para o valor exato.

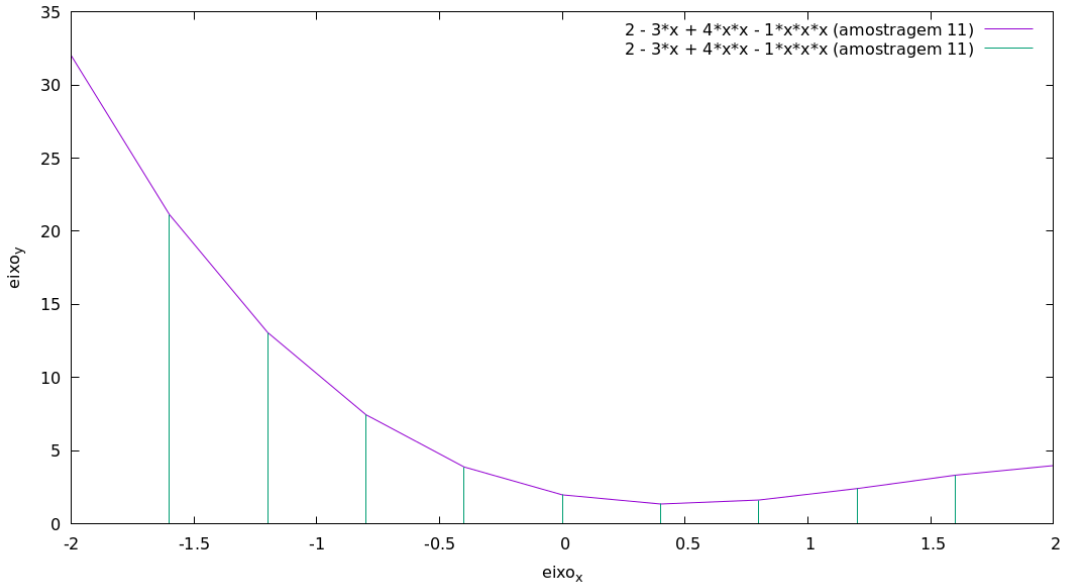
Podemos utilizar o software `gnuplot` para visualizar este comportamente. Abra o `gnuplot` e execute a sequência abaixo. Note que quanto maior a amostragem (valor de `samples`), melhor a curva obtida:

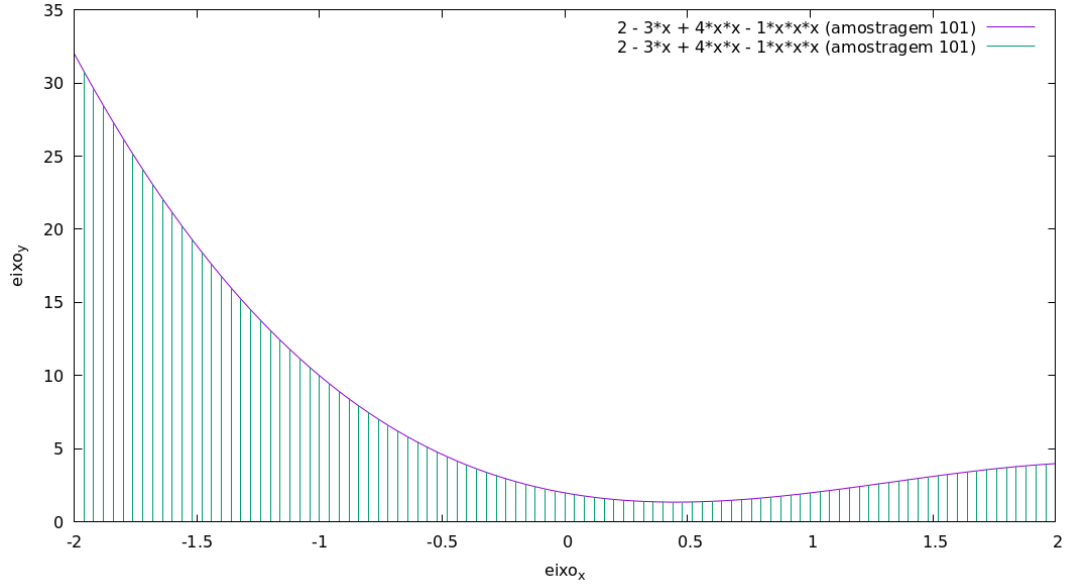
```
gnuplot
set xlabel "eixo\_x"
set ylabel "eixo\_y"
set samples 5
plot [-2:2] 2 - 3*x + 4*x*x - 1*x*x*x with impulses, \
    2 - 3*x + 4*x*x - 1*x*x*x with lines
set samples 10
replot
set samples 100
replot
```

O comando `set xlabel "eixo\_x"` é usado para definir o título do eixo x. O comando `set samples 5` define o número de intervalos, a amostragem. O comando `plot` é usado para plotar a curva. A opção [-2:2] é usada para definir o intervalo de plotagem, ou seja, li e ls. A opção `with impulses` mostra as linhas verticais e a opção `with lines` conecta os valores de $f(x_1)$, $f(x_2)$,..,$f(x_n)$ com linhas.

A medida que aumentamos a amostragem (`set samples n`), estamos aumentando o número de intervalos, reduzindo o valor de dx e melhorando a precisão na determinação da curva. Portanto, melhoramos a qualidade do cálculo da área. Veja na Figura 1.3 que quando aumentamos o número de amostras reduzimos o valor de dx e nos aproximamos da área exata.


2 - 3*x + 4*x*x - 1*x*x*x (amostragem 5)
2 - 3*x + 4*x*x - 1*x*x*x (amostragem 5)
eixo_y
eixo_x


2 - 3*x + 4*x*x - 1*x*x*x (amostragem 11)
2 - 3*x + 4*x*x - 1*x*x*x (amostragem 11)
eixo_y
eixo_x


2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)
2 - 3*x + 4*x*x - 1*x*x*x (amostragem 101)
eixo_y
eixo_x

A Figura 1.4 mostra a plotagem com valores diferentes de amostragem. Com amostragem 3 temos apenas 2 trapézios (em roxo), com amostragem 5 temos 4 trapézios (verde). Note que já nos aproximamos da curva em vermelho, que é uma curva com amostragem 101 e 100 trapézios. Enfim, quando maior a amostragem mais preciso será o cálculo da área.

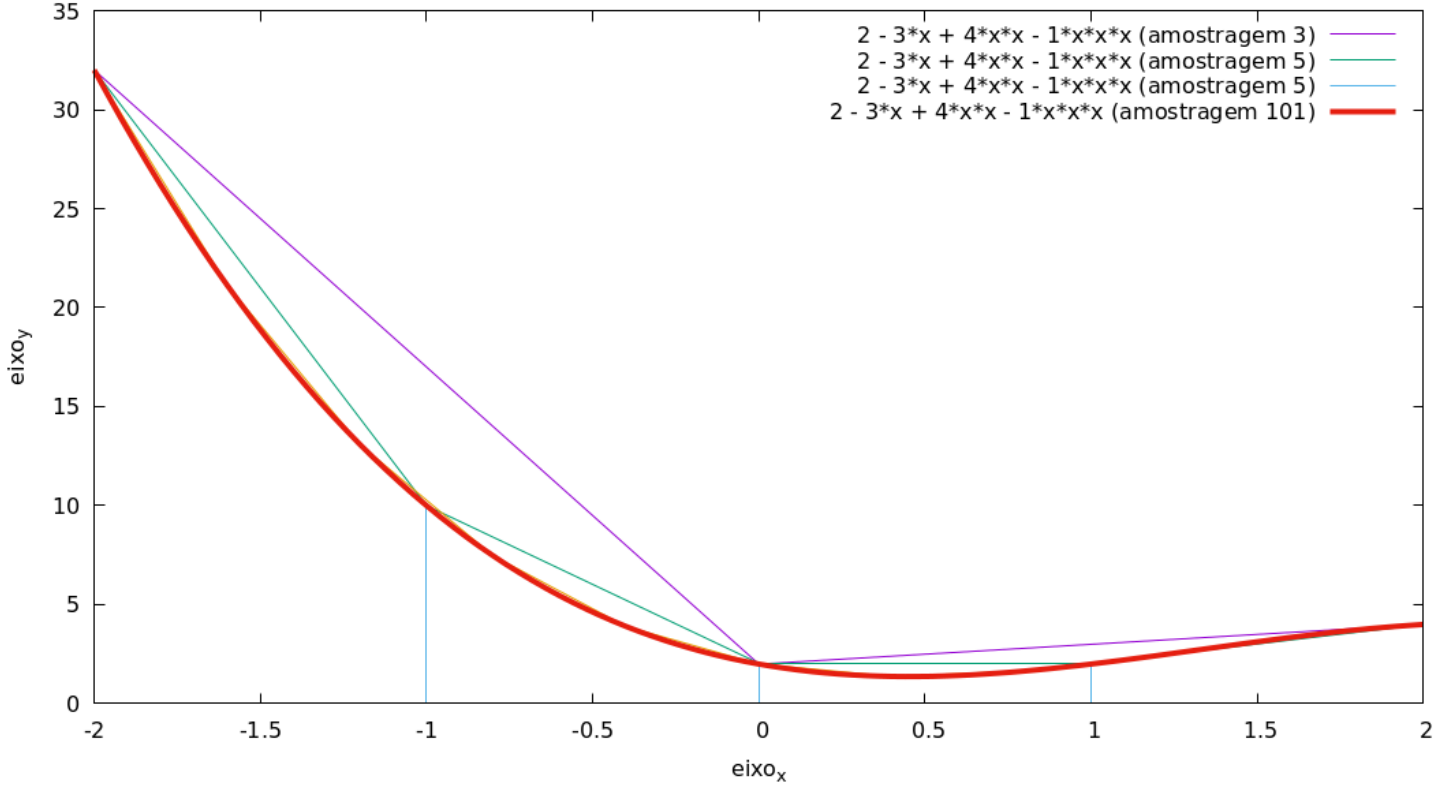


Figura 1.4: Curva de terceiro grau com diferentes amostragens - comparação direta

1.2.3 Cálculo da Área Aproximada Pelo Método de Simpson

A Figura 1.5 mostra a ideia do método de Simpson. Note que é feita uma aproximação da área da curva utilizando-se diversas parábolas (curvas do segundo grau - equação 1.2), e esta aproximação é mais precisa que a utilizada no método do trapézio (equação 1.1).

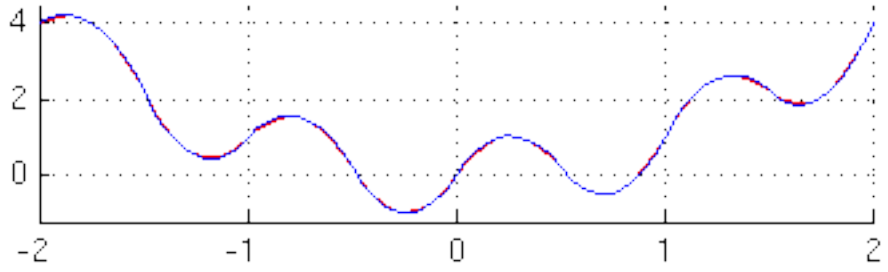


Figura 1.5: Esboço do Método de Simpson

A equação para cálculo da área da curva utilizando o método da integração por Simpson é dada por:

$$\int_a^b f(x)dx \approx \frac{dx}{3}\left[f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + 2f(x_5) + ... + 4f(x_{n-1}) + f(x_n)\right]$$

Note que a integral de Simpson requer no mínimo cinco pontos.

1.2.4 Alguns Resultados

A Tabela 1.1 mostra alguns resultados. Na primeira coluna a equação a ser calculada. Na segunda coluna valores exatos, a seguir o resultado da área calculado usando trapézio, o erro associado, o resultado calculado usando Simpson e o erro. Os campos vazioS devem ser calculados pelo leitor.

Tabela 1.1: Resultados para os Métodos do Trapézio e Simpson (101 intervalos)

Integral	Valor exato	Trapézio	Erro Trapézio	Simpson	Erro Simpson
$\int_0^1 e^x dx$	$e - 1 \approx 1,7183$	1,8591		1,7189	
$\int_0^1 \sqrt{1 - x^2} dx$	$\frac{\pi}{4} \approx 0,7854$	0,5		0,7440	
$\int_0^1 x dx$	0,5				
$\int_{-1}^1 2x^2 dx$	2/3				
$\int_{-1}^1 2x^2 - 3x^3 dx$	-1/12				
$\int_{-2}^2 2 - 3x + 4x^2 - 1x^3 dx$	88/3~29,33				
$\int_{-1}^1 sin(x) dx$	0				
$\int_{-2}^3 sin(x) + 2cos(x) dx$					

1.3 Análise

Na etapa anterior, de elaboração, aprendemos melhor os métodos de cálculo aproximado da área de uma função. Agora vamos fazer a etapa de análise, que tem por objetivo apresentar uma solução preliminar do software a ser desenvolvido. A mesma é construída com textos, esboços e diagramas. Esta solução preliminar é normalmente independente da plataforma a ser escolhida.

Uma das partes da análise consiste na escolha do paradigma de programação a ser adotado, por exemplo, se programação estruturada ou orientada a objeto.

Como este projeto tem fins educativos, vamos apresentar várias soluções para o mesmo problema. As primeiras soluções usam o paradigma de programação estruturada, depois entramos com exemplos que usam o paradigma de orientação a objeto e finalmente o paradigma de programação funcional.

- Programação estruturada
  - Versões mais simples e diretas.
  - Incluem diagramas que mostram a memória RAM, onde estão as funções e as variáveis.
- Programação orientada a objeto
  - A partir da versão v0.6, teremos início aos exemplos com orientação a objeto.
  - Nestes casos alguns diagramas estarão disponibilizados no subdiretório uml, são eles:
    - diagrama de caso de uso.
    - diagrama de classe.
    - diagrama de sequência.
  - Os arquivos de modelagem, diagramas UML, foram feitos usando o software `umbrello`, disponível no site https://umbrello.kde.org/.
- Programação funcional
  - Serão apresentados a partir da versão v1.3.

1.3.1 Como serão modeladas as versões orientadas a objeto?

- Como somos engenheiros ou estudantes de engenharia, já temos conhecimento prévio dos conceitos de funções e métodos de integração, o que facilita muito a modelagem. É fácil identificar os principais pacotes (assuntos com os quais estamos lidando):
  - Pacote de funções.
  - Pacote de métodos de integração.
  - Iremos integrar os pacotes de funções e métodos de integração em um simulador.

Objetivos além do programa em sí:

- Construir uma biblioteca de funções (pacote de funções);
- Construir uma biblioteca de métodos de integração numérica (pacote de integração);

Na prática iremos criar uma hierarquia de classes de funções (`CFuncao`) e uma hierarquia de classes de integração (`CIntegral`). Também é criada uma classe `CSimulador`, usada para gerenciar a simulação.

1.3.2 Tem documentação?

- Este arquivo contém as informações principais sobre como foram modelados e desenvolvidas as várias versões.
- Uma documentação do código é gerada pelo sistema doxygem/javadoc e está nos subdiretórios html.
  - Se a documentação não estiver disponível, basta rodar o `doxygen`.

    ```
    cd diretórioComCodigo
    doxygen
    ```

  - Se o doxygen não estiver instalado veja seção **??**.

1.3.3 Posso atualizar/ampliar os códigos disponibilizados?

- Sim. Os códigos são distribuídos com licença GPL, o que lhe garante as quatro liberdades do software livre.
- O usuário/programador pode adicionar novas classes funções e novas classes de integração, basta usar herança das classes existentes.

1.3.4 Porque tem vários diretórios com diferentes números de versões?

- Para mostrar a evolução do programa:
  - começando com versões estruturadas
  - a seguir versões orientadas a objeto,
  - e, finalmente, versões que adicionam conceitos de programação funcional.

1.3.5 Preciso saber algo mais?

- Comandos básicos do shell bash:

`cd bueno`	Muda para diretório bueno.
`cd ../x`	Muda para diretório x (.. é o diretório pai)
`pwd`	Mostra diretório corrente (path/caminho).
`ls`	Mostra lista arquivos do diretório corrente.
`ls -lah`	Mostra lista detalhada(-l), arquivos ocultos(-a).
`man ls`	Manual do ls (digite q para sair).
`cat x`	Mostra conteúdo do arquivo x.
`reset`	Reseta o terminal(zera/limpa).
`history`	Histórico de comandos.
`ctrl +`	(+) Aumenta letra, (-) Diminui letra.
`ctrl l`	Limpa tela (o mesmo que clear).
`TAB`	Digite o início do comando e a seguir complete com TAB.

1.3.6 Quais as versões e suas características?

- ProgramacaoDireta - v0.1
  - Código no estilo de C, mas usando conceitos de C++, como `cin`/`cout`;
  - Uso de `std::` ;
  - Tudo dentro da função `int main();`
  - Note que a definição da função esta misturada com definição do método de integração;
  - Para compilar e linkar:
    - g++ -std=c++11 main.cpp
  - Para rodar
    - `./a.out`
  - Sequência:
    - Entrada dos dados da função;
    - Entrada dos dados de integração;
    - Cálculo da área da função.

- ProgramacaoDireta - v0.2
  - Otimização do uso de memória
    - Nesta versão declaramos as variáveis perto de onde serão utilizadas.
    - Esta é uma das características que diferenciou C++ de C.
    - A vantagem é que além de economizar memória (o objeto é criado apenas quando vai ser usado), deixa o código mais claro.
  - Otimização do tempo de processamento
    - Otimiza-se o processamento trocando `x = x + dx` por `x += dx;`
    - Uma outra pequena mudança é colocar o `*dx` para fora do *looping*.
  - Uso de arquivo Makefile
    - Criamos e usamos um arquivo Makefile bem simples.
- ProgramacaoEstruturada -v0.3
  - Código semi estruturado - Usando uma função.
  - Parte do código que calcula a integral foi separada e colocada na função `AreaFuncao2G`.
  - A função `AreaFuncao2G` recebe os parâmetros por cópia.

  - Note que estamos começando a usar conceitos do paradigma de programação estruturada, em que os códigos são organizados em funções.
  - Note que ainda temos mistura do código do método de integração com código da função específica.
  - Também estamos usando: `using namespace std;` que permite trocar `std::cout` por `cout`.
  - Para compilar e linkar usando arquivo Makefile:
    * `make`

- ProgramacaoEstruturada -v0.4
  - Código estruturado - Usando duas funções.
  - Em relação a versão anterior parte do código que calcula a função do segundo grau foi separada da função que calcula a Area.
  - Este tipo de separação deixa o código mais organizado e legível, mas com a chamada extra de função pode ficar mais lento.
    * Declarei a função usando inline para evitar esta perda de desempenho.
    * `double inline F2G ( double& c0 , double& c1 , double& c2 , double& x );`

- Também usei referências (&) para evitar cópia dos parâmetros, o que implicaria em maior tempo de processamento.
- Note ainda que a função `main()` não foi modificada.
  - Isto ocorre porque a interface, a forma de acesso a função `AreaFuncao2G` não foi modificada.
- Note que a forma de acesso a função AreaFuncao2G não foi modificada.
- Para compilar e linkar com C++11 e definir executável como sendo `integral`:
  - `g++ -std=c++11 main.cpp -o integral`
- Para obter informações dos processos use `ps`, `ps` mostra o pid, o identificador do processo:
  - `man ps`
  - `ps`
  - `ps -aux | egrep integral`
- Para obter consumo memória (executar em outro terminal)
  - `pmap -x pid`

- ProgramacaoEstruturada -v0.4-2

  - A única mudança é o uso de `argc` e `argv` possibilitando que o usuário passe informações via linha de comando. O usuário pode passar os coeficientes da função.
    * Protótipo:
      · `./integral c0 c1 c2`
    * `Exemplo`
      · `./integral 1.1 2.1 3.1`

- ProgramacaoEstruturada -v0.5
  - Uso das variáveis globais `c0,c1,c2, limInf, limSup, numInt`, para mostrar redução passagem de parâmetros.
  - Usa ponteiro para função, cuja grande vantagem é permitir o calculo da integral de qualquer função do tipo `"double f(double x);"`
  - Adição de duas funções para entrada de dados.
  - Nesta versão as variáveis e funções que manipulam as variáveis estão visualmente próximas.

- ProgramacaoEstruturadaComUsoDeDiálogos -v0.5b

- Nesta versão o que muda é a forma de interação do usuário com o programa, pois o mesmo disponibiliza diálogos para entrada de dados.
- Iremos mostrar:
  - o uso da biblioteca gráfica Qt5, uma biblioteca gráfica bastante amigável que funciona no GNU/Linux, Windows e Mac-OS-X, sendo multiplataforma.
  - o funcionamento do `qmake-qt5 -project` e `qmake-qt5`
  - o uso das classes de diálogos:
    - `QInputDialog::getDouble`
    - `QInputDialog::getInt`
    - `QMessageBox`
  - Note que:
    - A biblioteca Qt foi instalada no sistema (https://www.qt.io/).
    - Acessamos um site (ou um livro) que explica o funcionamento dos diálogos de Qt.
    - Os arquivos da biblioteca Qt são incluídos no nosso programa.
    - Criamos objetos derivados do tipo `QDialog`.
    - Usamos os objetos criados.

- ProgramacaoOrientadaObjeto - v0.6
  - Nos exemplos que vimos até aqui usamos programação estruturada. Na versão v0.5 tivemos alguns avanços, como a possibilidade de calcular a área de qualquer função do tipo "`double funcao(double);`". Mas temos alguns problemas:
    - As variáveis da função do segundo grau c0, c1, c2, são globais e podem ser utilizadas por qualquer função que venha ser criada.
    - Se quizermos ter, simultaneamente, duas funções, teremos de refazer todo código. Isto ocorre porque na programação estruturada não existe um vínculo direto entre os coeficientes da função (c0, c1, c2) e a função em sí. Mesmo sendo a relação de fato muito próxima e direta.
  - Programa em C++ com duas classes.
    - Foram criadas as classes `CFun2G` e `CIntTrapezio`, as mesmas representam uma função de segundo grau e o método de integração do trapézio.
    - Os atributos estão públicos e o acesso é direto.
    - Os métodos foram declarados e definidos dentro das classes (nos arquivos .h).
    - Adicionado método `Entrada()`, para entrada de dados do objeto.
  - Como passaremos a ter um grande número de arquivos .h e .cpp, os códigos estão dentro do diretório src.

  - O controle ainda esta dentro de `main()`.
  - Também passamos a usar o programa `cmake` (https://cmake.org/) para gerenciar a construção/compilação do software.

- ProgramacaoOrientadaObjeto - v0.7
  - Métodos
    * Adicionados métodos `get`/`set` para leitura e modificação dos atributos.
    * Por exemplo:
      · `double limInf; // é o atributo`
      · `void LimInf( double novoLimInf ) { limInf = novoLimInf ; } // set`
      · `double LimInf() { return limInf } // get`
  - Construtures
    * Adicionados construtores.
    * Uso da palavra chafe `default` para indicar construtor *default*.
    * Adicionados exemplos com criação dos objetos de integração usando os construtores.

- ProgramacaoOrientadaObjeto - v0.8
  - Foi criada hierarquia de funções: classes `CFuncao`, `CFun1G`, `CFun2G`, `CFunExp`.
  - Foi criada hierarquia de métodos de integral: classes `CIntegral`, `CIntTrapezio`.
  - Usa métodos virtuais.
  - Usa palavras chaves do C++11 como `override`;
- ProgramacaoOrientadaObjeto - v0.9
  - Acrescenta sobrecarga operadores.
    - Acrescenta sobrecarga `operator()`.
    - Acrescenta sobrecarga operadores `operator>>` e `operator<<`.
  - Acrescenta classe `CSimulador` com conteúdo de `main`().
  - Note que o conteúdo de `main`() é quase sempre pequeno, normalmente criamos um simulador e executamos a simulação.
- ProgramacaoOrientadaObjeto - v1.0

  - Acrescenta polimorfismo, permitindo a definição de qual função e qual método de integração serão utilizados em tempo de execução.
    * Ou seja, o usuário irá selecionar a função e o método de integração.
  - Note que este exemplo conclui a parte de orientação a objeto tradicional.
    * Nos próximos adicionaremos o uso da biblioteca STL e inovações de C++11 ou posterior.

- Programação Genérica - v1.1
  - Acrescenta uso algoritmo genérico - `<vector>` da stl;
  - Uso de iteradores (`iterator`);
  - Uso de `friend std::istream& operator>>(std::istream& in, CFuncao& funcao);`
  - Dentro de `CSimulador::Executar` duas formas de uso:
    * Mostra uso de `vector`, `new`, `push_back`, `size`, `iterator` no formato de C++03;
    * Mostra uso de `vector`, `new`, `auto`, `for_each` e funções lambda de C++11 (note redução número linhas!)

- Programação Genérica - v1.2 (C++11/14)
  - Inclui mecanismos de programação genérica de C++11.

  - Novos conceitos de C++11 como `unique_ptr` e `make_ptr`.
  - Função `Area()` recebe ponteiro do tipo: `unique_ptr<CFuncao> *`
  - Funções `Entrada` e `Saida` recebem stream, permitindo uso de `Saida(cout)` ou `Saida(fout)`.

- Programação Genérica - v1.3 (C++17)

  - Inclui mecanismos de programação genérica de C++17. IMPLEMENTAR!
  - http://en.cppreference.com/w/cpp/numeric/special_math
  - http://en.cppreference.com/w/cpp/filesystem
  -

- Programação Genérica - v1.4 (C++20)

  - inclui mecanismos de C++20.
  - http://en.cppreference.com/w/cpp/io#Synchronized_output

Capítulo 2

v0.1 - Programação Direta

2.1 Características

- Código no estilo de C, mas usando conceitos de C++, como `cin`/`cout`;
- Uso de `std::` ;
- Tudo dentro da função `int main();`
- Note que a definição da função esta misturada com definição do método de integração;
- Para compilar e linkar:
  - g++ -std=c++11 main.cpp
- Para rodar
  - `./a.out`

- Sequência:
  - Entrada dos dados da função $y = a + b * x + c * x^2$;
  - Entrada dos dados do método de integração, li, ls, n;
  - Cálculo da área da função.

2.2 Códigos

Listing 2.1: Programa para cálculo da integral do trapézio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Implementação de uma pequena biblioteca de classes para funções e
4 * cálculo de integral numérica.
5 *
6 * Características da versão:
7 * - Código no estilo de C, mas usando conceitos de C++, como cin/cout.
8 * - Tudo dentro de main().
9 * - Note que a definição da função esta misturada com definição do método de integração.
10 * - Uso de std::
11 *
12 * Para compilar:
13 * g++ -std=c++11 main.cpp -o integral
14 *
15 * Nota:
16 * Códigos compilados usando o compilador da gnu (http://www.gcc.gnu.org/) na versão:
17 * g++ (GCC) 5.1.1 20150422 (Red Hat 5.1.1-1)
18*/
19#include <iostream>
20
21/// Função principal do programa
22/// Cálculo da área de uma função do segundo grau usando integração pelo método do trapézio.
23int main() {
24  // Variáveis no início da funcao main()
```

```
25  double c0{0.0};       // variáveis da função
26  double c1{0.0};
27  double c2{0.0};
28  double y{0.0};
29  double x{0.0};
30  double limInf{0.0};  // variáveis do método de integração
31  double limSup{1.0};
32  int numInt{101};
33  double dx{0.0};
34  double area{0.0};
35
36  // Inicio do código em sí
37  // Entrada dados função
38  std::cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
39  std::cout << "Entre␣com␣c0␣:␣";
40  std::cin  >> c0;
41  std::cout << "Entre␣com␣c1␣:␣";
42  std::cin  >> c1;
43  std::cout << "Entre␣com␣c2␣:␣";
44  std::cin  >> c2;
45                       // Entrada dados método integração
46  std::cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
47  std::cout << "Entre␣com␣Limite␣Inferior␣:␣";
48  std::cin  >> limInf;
49  std::cout << "Entre␣com␣limite␣Superior␣:␣";
50  std::cin  >> limSup;
51  std::cout << "Entre␣com␣numero␣intervalos␣:␣";
52  std::cin  >> numInt;
53  std::cin.get();
```

```
54
55   // Cálculo da área usando método do trapézio
56   x = limInf;
57   dx = (limSup - limInf) / double(numInt);   // uso cast de int para double
58   area = (c0 + c1 * x + c2 * x * x) * 0.5 * dx;
59 std::cout << "\nx="<< x << "␣y="<<y;
60   for( int i = 1; i < numInt ; i = i + 1 ) {
61      x = x + dx;
62      area = area + (c0 + c1 * x + c2 * x * x)*dx;// multiplica por dx dentro looping
63 std::cout << "\nx="<< x << "␣y="<<y;
64   }
65   x = limSup;
66   area = area + (c0 + c1 * x + c2 * x * x) * 0.5 * dx;
67 std::cout << "\nx="<< x << "␣y="<<y;
68   // Saída resultados
69   std::cout << "Area␣=␣" << area << std::endl;
70   std::cin.get();
71   return 0;
72 }
```

```
bueno@localhost v0.1]$ ls
dados  integral  main.cpp  Makefile

[bueno@localhost v0.1]$ g++ -std=c++11 -c main.cpp
[bueno@localhost v0.1]$ g++ main.o

[bueno@localhost v0.1]$ ./a.out
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 1
Entre com c1 : 2
Entre com c2 : -2
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 10
Entre com numero intervalos : 100
Area = -556.7
```

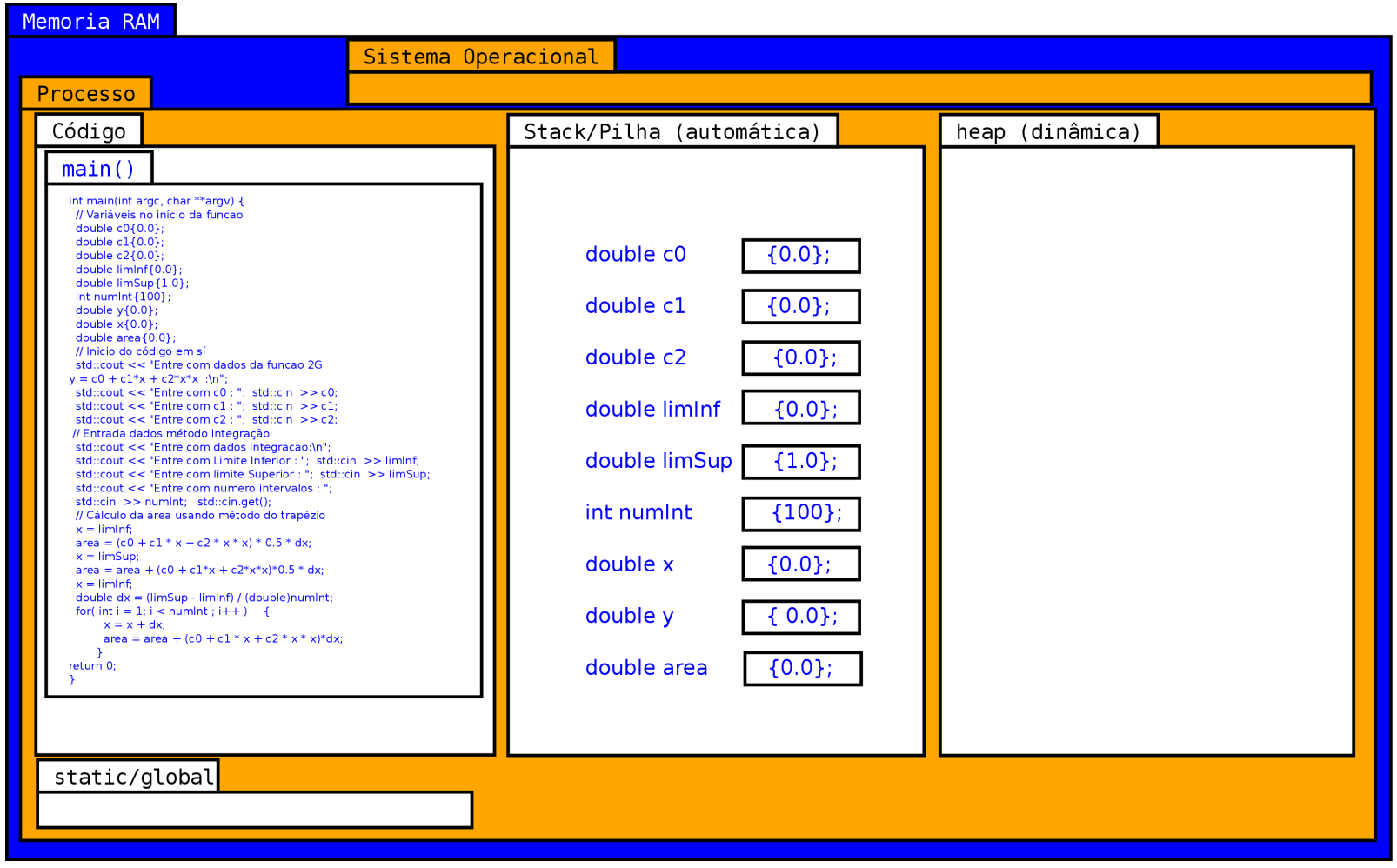


Figura 2.1: Projeto 1 - v0.1 - Integração Numérica: Organização memória RAM

Capítulo 3

v0.2 - Programação Direta - Otimizada

3.1 Características

- Otimização do uso de memória
  - Nesta versão declaramos as variáveis perto de onde serão utilizadas.
  - Esta é uma das características que diferenciou C++ de C.
  - A vantagem é que além de economizar memória (o objeto é criado apenas quando vai ser usado), deixa o código mais claro.
- Otimização do tempo de processamento
  - Otimiza-se o processamento trocando `x = x + dx` por `x += dx;`
  - Uma outra pequena mudança é colocar o `*dx` para fora do *looping*.
- Uso de arquivo Makefile
  - Criamos e usamos um arquivo Makefile bem simples.

3.2 Códigos

Listing 3.1: Programa para cálculo da integral do trapésio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Implementação de uma pequena biblioteca de classes para funções e
4 * cálculo de integral numérica.
5 *
6 * Características da versão:
7 * Nesta versão declaramos as variáveis perto de onde serão utilizadas.
8 * Esta é uma das características que diferenciou C++ de C.
9 * A vantagem é que além de economizar memória (o objeto é criado apenas quando vai ser usado),
10 * deixa o código mais claro.
11 *
12 * Para compilar:
13 * g++ -std=c++11 main.cpp -o integral
14*/
15
16#include <iostream>
17
18
19
20
21
22/// Função principal do programa
23/// Cálculo da área de uma função do segundo grau usando integração pelo método do trapézio.
24int main(int argc, char **argv) {
```

```
25  // Entrada de dados
26  std::cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
27  std::cout << "Entre␣com␣c0␣:␣";
28  double c0{0.0};
29  std::cin >> c0;
30  std::cout << "Entre␣com␣c1␣:␣";
31  double c1{0.0};
32  std:: cin >> c1;
33  std::cout << "Entre␣com␣c2␣:␣";
34  double c2{0.0};
35  std:: cin >> c2;
36
37  std::cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
38  std::cout << "Entre␣com␣Limite␣Inferior␣:␣";
39  double limInf{0.0};
40  std:: cin >> limInf;
41  std::cout << "Entre␣com␣limite␣Superior␣:␣";
42  double limSup{1.0};
43  std:: cin >> limSup;
44  std::cout << "Entre␣com␣numero␣intervalos␣:␣";
45  int numInt{101};
46  std:: cin >> numInt;
47  std::cin.get();
48
49  // Cálculo da área usando método do trapézio
50  double x = limInf;                          // Não criamos y porque não usamos y.
51  double area{0.0};
52  area = (c0 + c1*x + c2*x*x)*0.5;
53  double dx = ( limSup - limInf )/(double)numInt;
```

```
54  for( int i = 1; i < numInt ; i++ )    {        // += otimiza processamento!
55          x += dx;                               // o mesmo que x = x + dx
56          area += (c0 + c1*x + c2*x*x);
57       }
58  x = limSup;
59  area += (c0 + c1*x + c2*x*x)*0.5;
60  area *= dx ;  // colocamos multiplicação para fora reduzindo número de
61                // multiplicações e tempo processamento!
62  std::cout << "Area = " << area << std::endl;
63  std::cin.get();
64  return 0;
65 }
```

```
66 bueno@localhost v0.2]$ ls
67 main.cpp  Makefile
68
69 [bueno@localhost v0.2]$ g++ -std=c++11 main.cpp
70
71 [bueno@localhost v0.2]$ ./a.out
72 Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
73 Entre com c0 : -1
74 Entre com c1 : 2
75 Entre com c2 : 1
76 Entre com dados do metodo integracao:
77 Entre com Limite Inferior : -2
78 Entre com limite Superior : 2
79 Entre com numero intervalos : 50
80 Area = 1.3376
```

Note que para compilar e linkar o programa gerando executável com nome `integral` precisamos digitar:

- `g++ -std=c++11 main.cpp -o integral`

Podemos usar um programa chamado `make` para automatizar esta digitação. O programa `make` lê e executa comandos armazenados em um arquivo `Makefile`.

Apresenta-se a seguir um arquivo `Makefile` bem simples que pode ser usado para compilar o programa. Para usar o arquivo basta digitar o comando `make`. O programa `make` abre o arquivo `Makefile` e executa os comandos abaixo de `all`. Note que também podemos digitar `make all` ou ainda `make clean` para remover os arquivos `*.o` e `integral`.

- `make`

Listing 3.2: Arquivo Makefile.

```
1 all:
2         g++ main.cpp -o integral
3
4 clean:
5         rm *.o
```

Capítulo 4

v0.3 - Programação Estruturada - Uma Função

4.1 Características

- Código semi estruturado - Usando uma função.
- Parte do código que calcula a integral foi separada e colocada na função `AreaFuncao2G`.
- A função `AreaFuncao2G` recebe os parâmetros por cópia.
- Note que estamos começando a usar conceitos do paradigma de programação estruturada, em que os códigos são organizados em funções.
- Note que ainda temos mistura do código do método de integração com código da função específica.
- Também estamos usando: `using namespace std;` que permite trocar `std::cout` por `cout`.
- Para compilar e linkar usamos o programa `make` e o arquivo `Makefile`:
  - `make`

4.2 Códigos

Listing 4.1: Programa para cálculo da integral do trapésio da função y = c0+c1*x+c2*x*x.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Código semi estruturado - Usando funções.
 * - Parte do código que calcula a integral foi separada e colocada na função AreaFuncao2G.
 * Note que estamos começando a usar conceitos do paradigma de programação estruturada,
 * em que os códigos são organizados em funções.
 * - Note que a definição da função continua misturada com código método integração.
 * - Também estamos usando: using namespace std; que permite trocar cout por cout.
 *
 * Para compilar:
 * g++ -std=c++11 main.cpp -o integral
*/
#include <iostream>
using namespace std;



/// Cálculo da área de uma função do segundo grau usando trapézio
double AreaFuncao2G(double c0,double c1,double c2,double limInf,double limSup,int numInt) {
  double x = limInf;
```

```
25   double area = ( c0 + c1 * x + c2 * x * x ) * 0.5;
26   double dx = (limSup - limInf) / (double)numInt;
27   for( int i = 1; i < numInt ; i++ )    {
28           x += dx;
29           area += ( c0 + c1 * x + c2 * x * x );
30         }
31   x = limSup;
32   area += ( c0 + c1 * x + c2 * x * x ) * 0.5;
33   area *= dx ;
34   return area;
35 }
36
37 /// Função principal do programa
38 int main()
39 {
40   // Entrada dados
41   cout << "Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x :\n";
42   cout << "Entre com c0 : ";
43   double c0{0.0};
44   cin >> c0;
45   cout << "Entre com c1 : ";
46   double c1{0.0};
47   cin >> c1;
48   cout << "Entre com c2 : ";
49   double c2{0.0};
50   cin >> c2;
51
52   cout << "Entre com dados do metodo integracao:\n";
53   cout << "Entre com Limite Inferior : ";
```

```
54  double limInf{0.0};
55  cin >> limInf;
56  cout << "Entre␣com␣limite␣Superior␣:␣";
57  double limSup{1.0};
58  cin >> limSup;
59  cout << "Entre␣com␣numero␣intervalos␣:␣";
60  int numInt{101};
61  cin >> numInt;
62
63  // Chama função que calcula área
64  // notar que passa dados da função e do método integração
65  double area = AreaFuncao2G(c0,c1,c2,limInf,limSup,numInt);
66
67  // Saída resultados
68  cout << "Area␣=␣" << area << endl;
69
70  // Como criamos função podemos chamar a mesma várias vezes!
71  double area2 = AreaFuncao2G(c0,c1,c2,limInf,limSup,numInt*10);
72  cout << "Area2␣=␣" << area2 << endl;
73  double area3 = AreaFuncao2G(c0,c1,c2,limInf,limSup,numInt*100);
74  cout << "Area3␣=␣" << area3 << endl;
75  cin.get();
76  return 0;
77}
```

```
[bueno@localhost v0.3-ProgramacaoEstruturada-a-UsandoUmaFuncao]$ ls
integral  main.cpp  Makefile

[bueno@localhost v0.3-ProgramacaoEstruturada-a-UsandoUmaFuncao]$ cat make
all:
	g++ -std=c++11 main.cpp -o integral

clean:
	rm *.o

[bueno@localhost v0.3-ProgramacaoEstruturada-a-UsandoUmaFuncao]$ make
g++ -std=c++11 main.cpp -o integral

[bueno@localhost v0.3-ProgramacaoEstruturada-a-UsandoUmaFuncao]$ ./integral
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 3
Entre com c1 : -5
Entre com c2 : 2
Entre com dados do metodo integracao:
Entre com Limite Inferior : -1
Entre com limite Superior : 1
Entre com numero intervalos : 100
Area = 7.3336
```

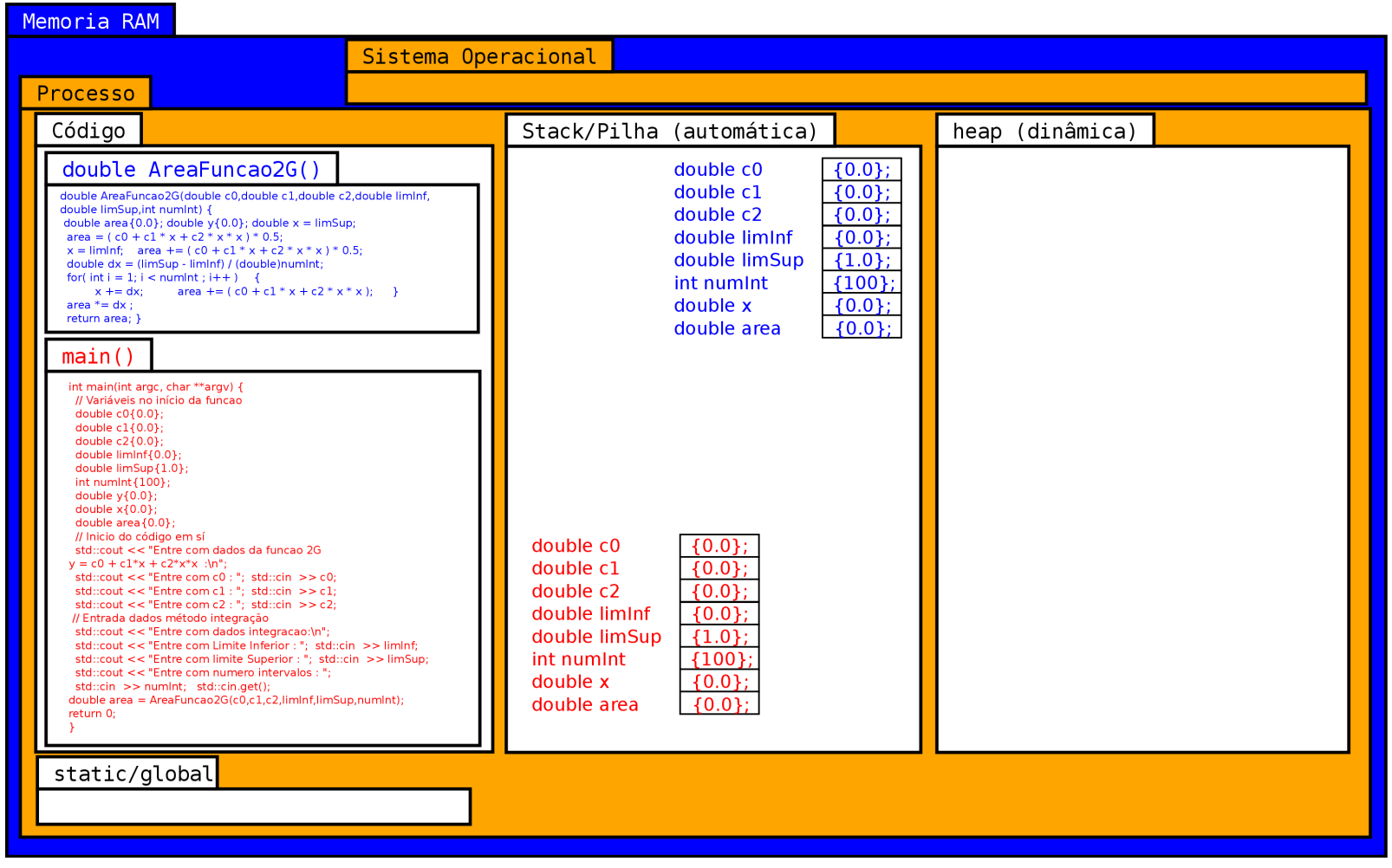


Figura 4.1: Projeto 1 - v0.1 - Integração Numérica: Organização memória RAM

Capítulo 5

v0.4 - Programação Estruturada - Duas Funções, Referências

5.1 Características

- Código estruturado - Usando duas funções.
- Em relação a versão anterior parte do código que calcula a função do segundo grau foi separada da função que calcula a Area.
- Este tipo de separação deixa o código mais organizado e legível, mas com a chamada extra de função pode ficar mais lento.
  - Declarei a função usando inline para evitar esta perda de desempenho.
  - `double inline F2G ( double& c0 , double& c1 , double& c2 , double& x );`
- Também usei referências (ex: `double& c0`) para evitar cópia dos parâmetros, o que implicaria em maior tempo de processamento.
- Note que a função `main()` não foi modificada.
  - Isto ocorre porque a interface, a forma de acesso a função `AreaFuncao2G` não foi modificada.
- Note que a forma de acesso a função AreaFuncao2G não foi modificada.

- Para compilar e linkar com C++11 e definir executável como sendo `integral`:
  - `g++ -std=C++11 main.cpp -o integral`
- Para obter informações dos processos use `ps`, `ps` mostra o pid, o identificador do processo:
  - `man ps`
  - `ps`
  - `ps -aux | egrep integral`
- Para obter consumo memória (executar em outro terminal)
  - `pmap -x pid`

5.2 Códigos

Listing 5.1: Programa para cálculo da integral do trapézio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Implementação de uma pequena biblioteca de classes para funções e
4 * cálculo de integral numérica.
5 *
6 * Características da versão:
7 * Código estruturado - Usando duas funções.
8 * Em relação a versão anterior parte do código que calcula a função do segundo grau foi separada da função que
     calcula a Area.
9 * Este tipo de separação deixa o código mais organizado e legível, mas com a chamada extra de função pode ficar
     mais lento.
10 *   Declarei a função usando inline para evitar esta perda de desempenho.
11 *   double inline F2G ( double& c0 , double& c1 , double& c2 , double& x );
12 * Também usei referências (&) para evitar cópia dos parâmetros, o que implicaria em maior tempo de processamento.
13 * Note ainda que a função main() não foi modificada.
14 *   Isto ocorre porque a interface, a forma de acesso a função AreaFuncao2G não foi modificada.
15 *
16 * Para compilar:
17 * g++ -std=c++11 main.cpp -o integral
18 * Para obter pid:
19 * ps -aux | egrep integral
20 * Para obter consumo memória (execute em outro terminal)
21 * pmap -x pid
22*/
```

```
23#include <iostream>
24#include <iomanip>
25using namespace std;
26
27/// Cálculo da função do segundo grau
28// O uso de referência elimina a cópia das variáveis!
29// O uso de inline elimina a chamada e retorno da função, a mesma é copiada onde é chamada.
30double inline F2G( double &c0, double &c1, double &c2, double &x ) {
31  return (c0 + c1*x + c2*x*x);
32}
33
34/// Cálculo da área usando método trapézio
35double AreaFuncao2G( double c0,double c1,double c2,double limInf,double limSup,int numInt ) {
36  double area{0.0};
37  double x = limInf;
38  area = F2G(c0,c1,c2,x) * 0.5; // Agora chama função F2G
39  double dx = (limSup - limInf) / (double)numInt;
40  for( int i = 1; i < numInt ; i++ ) {
41          x += dx;
42          area += F2G(c0,c1,c2,x);
43       }
44  x = limSup;
45  area += F2G(c0,c1,c2,x) * 0.5;
46  area *= dx ;
47  //int lixo[1000][1000];
48  //cout << "Pausa para mostrar em outro terminal consumo memória: ";
49  //cin.get(); // pausa para mostrar em outro terminal o consumo memória
50  return area;
51}
```

```
52
53/// Função principal do programa
54int main(int argc, char **argv) {
55  // Entrada dados
56  cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
57  cout << "Entre␣com␣c0␣:␣";
58  double c0{0.0};
59  cin >> c0;
60  cout << "Entre␣com␣c1␣:␣";
61  double c1{0.0};
62  cin >> c1;
63  cout << "Entre␣com␣c2␣:␣";
64  double c2{0.0};
65  cin >> c2;
66
67  cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
68  cout << "Entre␣com␣Limite␣Inferior␣:␣";
69  double limInf{0.0};
70  cin >> limInf;
71  cout << "Entre␣com␣Limite␣Superior␣:␣";
72  double limSup{0.0};
73  cin >> limSup;
74  cout << "Entre␣com␣Numero␣Intervalos␣:␣";
75  int numInt{101};
76  cin >> numInt;
77  cin.get();
78  //cout << "Pausa para mostrar em outro terminal consumo memória: ";
79  //cin.get(); // pausa para mostrar em outro terminal o consumo memória
80
```

```
81  // Chama função de cálculo da área várias vezes
82  double area ;
83  for ( int i = 1; i <= 1000; i *= 10  ) {
84   area = AreaFuncao2G(c0,c1,c2,limInf,limSup,numInt*i);
85   // Saída resultados; setw(n) seta largura campo
86   cout << "numInt = " << setw(8) << numInt*i
87        << ", Area = " << setprecision(10) << setw(20) << area << endl;
88  }
89    cin.get();
90  return 0;
91 }
```

```
[bueno@localhost v0.3b]$ make
g++ -std=c++11 main.cpp -o integral
[bueno@bueno v0.4]$ ./integral
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 1
Entre com c1 : 2
Entre com c2 : 3
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com Limite Superior : 1
Entre com Numero Intervalos : 100
numInt =      100, Area =              3.00005
numInt =     1000, Area =            3.0000005
numInt =    10000, Area =          3.000000005
numInt =   100000, Area =                    3
```

A título ilustrativo adicionamos alguns `cin.get()` que param o programa e permitem, em outro terminal, obter informações de consumo de memória.

```
108
109 1) Roda o programa
110 ====================
111 [bueno@bueno v0.4]$ ./integral
112
113 2) Identifica o pid
114 ====================
115 [bueno@bueno v0.4]$ man ps
116 "man ps" retorna o manual do programa ps; Digite  "q" para sair do man.
117 "ps - report a snapshot of the current processes" [Informações dos processos].
118
119 [bueno@bueno v0.4]$ ps -aux
120 [bueno@bueno v0.4]$ ps -aux | egrep integral
121 Identifica o pid.
122 O primeiro pid é de ./integral o segundo de ps -aux | egrep integral.
123
124 3) Instruções do pmap
125 =====================
126 [bueno@bueno v0.4]$ man pmap
127 "pmap - report memory map of a process" [fornece informações consumo memória].
128 [bueno@bueno v0.4]$ pmap
129 Usage:
130  pmap [options] PID [PID ...]
131 Options:
132  -x, --extended              show details
133  -X                          show even more details
```

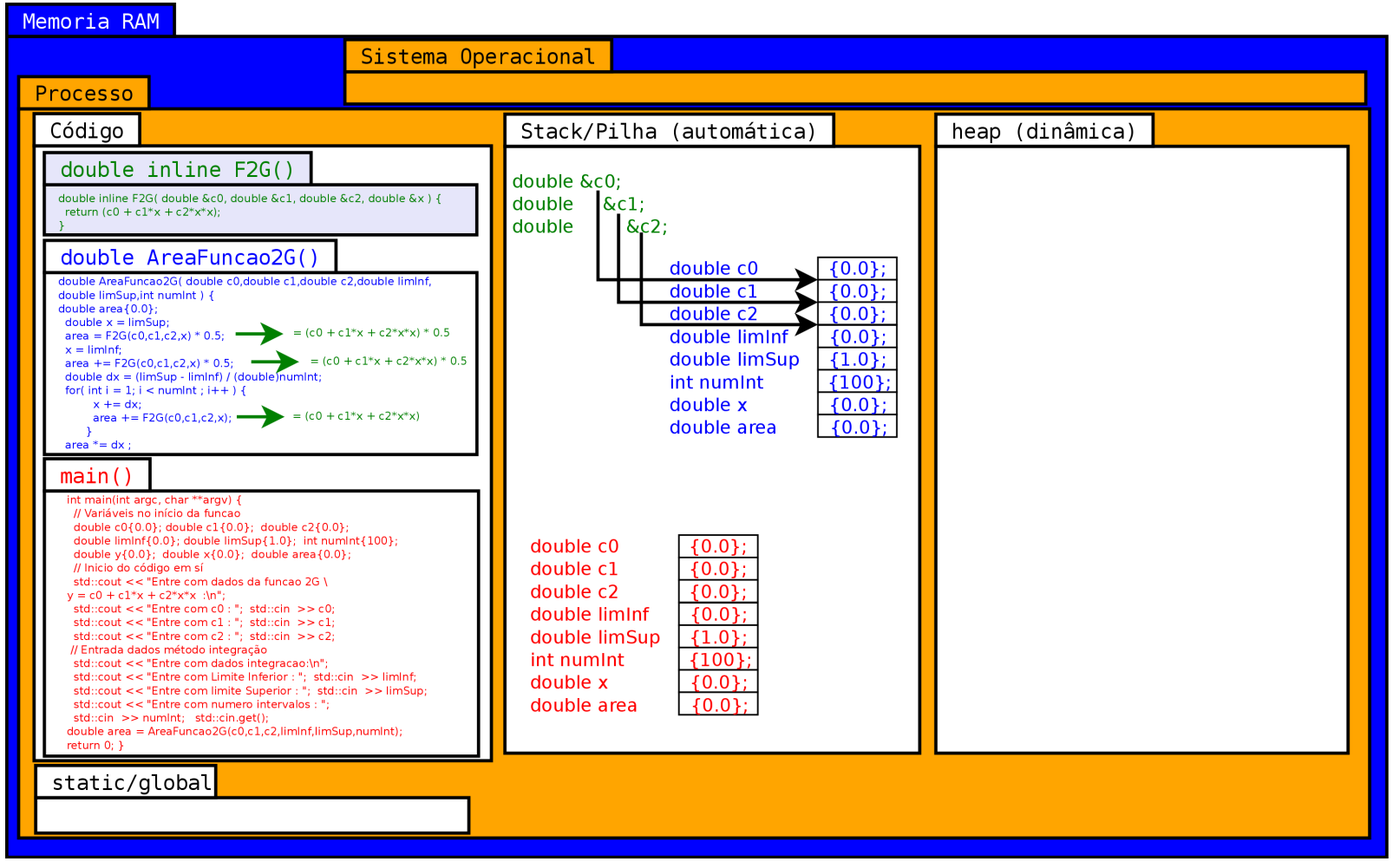


Figura 5.1: Projeto 1 - v0.1 - Integração Numérica: Organização memória RAM

```
            WARNING: format changes according to /proc/PID/smaps
 -XX                         show everything the kernel provides
 -c, --read-rc               read the default rc
 -C, --read-rc-from=<file>   read the rc from file
 -n, --create-rc             create new default rc
 -N, --create-rc-to=<file>   create new rc to file
            NOTE: pid arguments are not allowed with -n, -N
 -d, --device                show the device format
 -q, --quiet                 do not display header and footer
 -p, --show-path             show path in the mapping
 -A, --range=<low>[,<high>]  limit results to the given range

 -h, --help     display this help and exit
 -V, --version  output version information and exit
For more details see pmap(1).

4) Pede informações do programa usando "pmap  -x pid"
=====================================================
[bueno@bueno v0.4]$ pmap -x 4759
4759:   ./integral
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000       8       8       0 r-x-- integral
0000000000400000       0       0       0 r-x-- integral
0000000000601000       4       4       4 r---- integral
0000000000d3d000     200       8       8 rw---   [ anon ]
00007f4204406000    1756    1064       0 r-x-- libc-2.21.so
00007f42047c3000      16       8       8 rw---   [ anon ]
00007f42047c7000      88      88       0 r-x-- libgcc_s-5.3.1-20151207.so.1
00007f42049de000    1052     192       0 r-x-- libm-2.21.so
```

```
00007f4204ce6000    1484    1304       0 r-x-- libstdc++.so.6.0.21
00007f4205064000      16      16      16 rw---   [ anon ]
00007f4205068000     132     132       0 r-x-- ld-2.21.so
00007f4205258000      20      20      20 rw---   [ anon ]
00007f4205288000       4       4       4 r---- ld-2.21.so
00007f420528a000       4       4       4 rw---   [ anon ]
00007ffcebb90000     132      12      12 rw---   [ stack ]<<====
00007ffcebbbf000       8       0       0 r----   [ anon ]
---------------- ------- ------- -------
total kB           13224    2976     184

[bueno@bueno v0.4]$ pmap -x 4759
4759:   ./integral
Address           Kbytes     RSS   Dirty Mode  Mapping
0000000000400000       8       8       0 r-x-- integral
0000000000400000       0       0       0 r-x-- integral
0000000000601000       4       4       4 r---- integral
0000000000d3d000     200       8       8 rw---   [ anon ]
00007f4204406000    1756    1064       0 r-x-- libc-2.21.so
00007f42047c3000      16       8       8 rw---   [ anon ]
00007f42047c7000      88      88       0 r-x-- libgcc_s-5.3.1-20151207.so.1
00007f42049de000    1052     192       0 r-x-- libm-2.21.so
00007f4204ce6000    1484    1304       0 r-x-- libstdc++.so.6.0.21
00007f4205064000      16      16      16 rw---   [ anon ]
00007f4205068000     132     132       0 r-x-- ld-2.21.so
00007f4205258000      20      20      20 rw---   [ anon ]
00007f4205288000       4       4       4 r---- ld-2.21.so
00007f420528a000       4       4       4 rw---   [ anon ]
00007ffceb7dd000    3920      16      16 rw---   [ stack ] <<====
```

```
00007ffcebbbf000       8       0       0 r----   [ anon ]
---------------- ------- ------- -------
total kB           17012    2980     188

Note que o consumo de memória mudou.
Veja abaixo outro exemplo.
========================================
Antes de iniciar entrada dados:
total kB           13224    2952     180
Depois de entrar dados:
total kB           13224    3084     184
Dentro da função:
total kB           17012    3088     188

5) Veja ainda
Memória:
http://www.binarytides.com/linux-command-check-memory-usage/
Hardware:
http://www.binarytides.com/linux-commands-hardware-info/
```

Capítulo 6

v0.4-2 - Programação Estruturada - Duas Funções, Referências, argc/argv

6.1 Características

- Código estruturado - Usando duas funções.
- Uso de `argc` e `argv` possibilitando que o usuário passe informações via linha de comando.
  - Protótipo:
    - `./integral c0 c1 c2`
  - `Exemplo`
    - `./integral 1.1 2.1 3.1`
- Função `AreaFuncao` agora recebe parâmetros por referência.

6.2 Códigos

Listing 6.1: Programa para cálculo da integral do trapézio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Implementação de uma pequena biblioteca de classes para funções e
4 * cálculo de integral numérica.
5 *
6 * Características da versão:
7 * Uso de argc e argv para obter dados da função
8 *
9 * Para compilar:
10 * g++ -std=c++11 main.cpp -o integral
11 * Para obter pid:
12 * ps -aux | egrep integral
13 * Para obter consumo memória em outro terminal
14 * pmap -x pid
15*/
16#include <iostream>
17#include <iomanip>  // inclui atof
18using namespace std;
19
20/// Cálculo da função do segundo grau
21// O uso de referência elimina a cópia das variáveis!
22// O uso de inline elimina a chamada e retorno da função, a mesma é copiada onde é chamada.
23double inline F2G( double &c0, double &c1, double &c2, double &x ) {
24  return (c0 + c1*x + c2*x*x);
```

```
25}
26
27/// Cálculo da área usando método trapézio
28double AreaFuncao2G( double& c0, double& c1, double& c2, double& limInf, double& limSup, int& numInt ) {
29  double area{0.0};
30  double x = limSup;
31  area = F2G(c0,c1,c2,x) * 0.5; // Agora chama função F2G
32  x = limInf;
33  area += F2G(c0,c1,c2,x) * 0.5;
34  double dx = (limSup - limInf) / (double)numInt;
35  for( int i = 1; i < numInt ; i++ ) {
36          x += dx;
37          area += F2G(c0,c1,c2,x);
38       }
39  area *= dx ;
40  return area;
41}
42
43
44
45/// Função principal do programa
46int main(int argc, char **argv) {
47  // Entrada dados
48  if( argc < 4 ) {
49  cout << "\nUso: ./integral c0 c1 c2 (sendo c0 c1 c2 números flutuantes)\n\a";
50  return 1;
51  }
52  cout << "\nargc = "     << argc ;     // número de elementos da linha de comando
53  cout << "\nargv[0] = "  << argv[0] ;  // nome do programa
```

```
54  cout << "\nargv[1] = "  << argv[1] ;  // primeiro parâmetro
55  cout << "\nargv[2] = "  << argv[2] ;  // segundo parâmetro
56  cout << "\nargv[3] = "  << argv[3] ;  // terceiro parâmetro
57
58  double c0 = atof( argv[1] );          // atof converte string ASCII to Float
59  double c1 = atof( argv[2] );
60  double c2 = atof( argv[3] );
61
62  cout << "\nsa = " <<  s0;
63
64  cout << "\nDados da funcao 2G y = c0 + c1*x + c2*x*x :\n";
65  cout << "\nc0 = " <<  c0;
66  cout << "\nc1 = " <<  c1;
67  cout << "\nc2 = " <<  c2;
68
69  cout << "\nEntre com dados do metodo integracao:\n";
70  cout << "Entre com Limite Inferior : ";
71  double limInf{0.0};
72   cin >> limInf;
73  cout << "Entre com Limite Superior : ";
74  double limSup{0.0};
75   cin >> limSup;
76  cout << "Entre com Numero Intervalos : ";
77  int numInt{101};
78   cin >> numInt;
79  cin.get();
80
81  // Cálculo da área
82  double area ;
```

```
83   area = AreaFuncao2G(c0,c1,c2,limInf,limSup,numInt);
84   cout  << "numInt␣=␣" << setw(8) << numInt << ",␣Area␣=␣"
85         << setprecision(10) << setw(20) << area << endl;
86   cin.get();
87   return 0;
88 }
```

```
[bueno@bueno v0.4-2-Uso-argc-argv]$ make clean
rm *.o

[bueno@bueno v0.4-2-Uso-argc-argv]$ make
g++ -std=c++11 main.cpp -o integral

[bueno@bueno v0.4-2-Uso-argc-argv]$ ./integral
Uso: ./integral c0 c1 c2 (sendo c0 c1 c2 números flutuantes)

[bueno@bueno v0.4-2-Uso-argc-argv]$ ./integral 1 2 3
argc = 4
argv[0] = ./integral
argv[1] = 1
argv[2] = 2
argv[3] = 3
Dados da funcao 2G y = c0 + c1*x + c2*x*x  :

c0 = 1
c1 = 2
c2 = 3
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com Limite Superior : 1
Entre com Numero Intervalos : 100
numInt =      100, Area =              3.00005
```

Nota: note, na ilustração a seguir, que os parâmetros recebidos por referência (&) não consomem memória, pois são apenas "apelidos" para os objetos recebidos. A grande vantagem é a economia de memória e a rapides no proces-

samento.

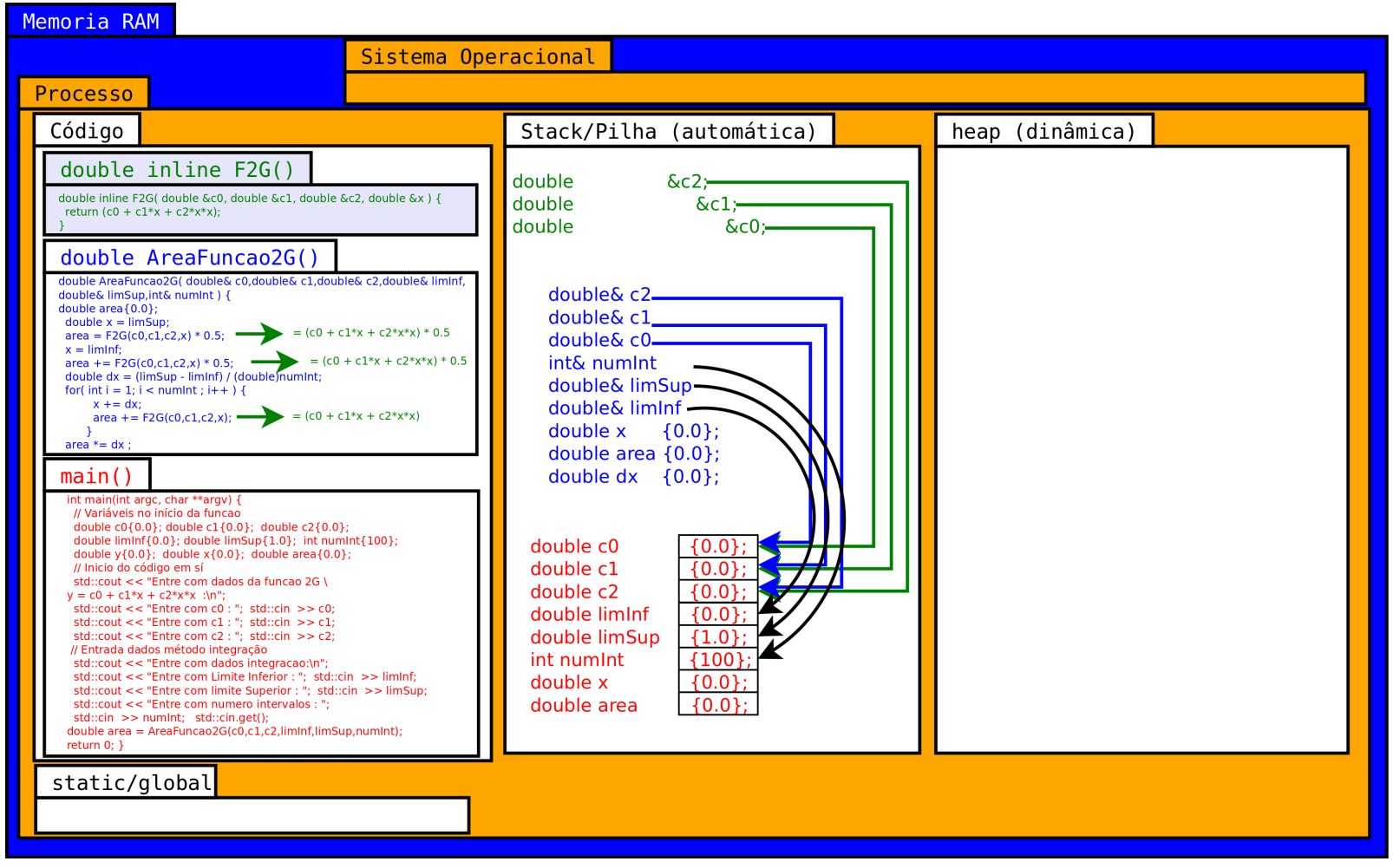


Figura 6.1: Projeto 1 - v0.1 - Integração Numérica: Organização memória RAM

Capítulo 7

v0.5 - Programação Estruturada - Ponteiro Função

7.1 Características

- Uso das variáveis globais c0,c1,c2, limInf, limSup, numInt, para mostrar redução passagem de parâmetros.
  Dica: Evite o uso de variáveis globais!

- Usa ponteiro para função, cuja grande vantagem é permitir o calculo da integral de qualquer função do tipo "`double f(double x);`"

- Usa duas funções para entrada de dados.

- Nesta versão as variáveis e funções que manipulam as variáveis estão visualmente próximas.

  - Note que poderíamos colocar as coisas da função num arquivo funcao.h e as coisas da integral num arquivo integral.h.

- Para resetar o terminal, compilar e linkar e a seguir executar o programa:

  - `reset && g++ -std=C++11 main.cpp && ./a.out`

7.2 Códigos

Listing 7.1: Programa para cálculo da integral do trapézio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Implementação de uma pequena biblioteca de classes para funções e
4 * cálculo de integral numérica.
5 *
6 * Características da versão:
7 * - Uso das variáveis globais c0,c1,c2, limInf, limSup, numInt, para mostrar redução passagem de parâmetros.
8 * - Usa ponteiro para função, cuja grande vantagem é permitir o calculo da integral de qualquer função
9 * do tipo "double f(double x);"
10 * - Adição de duas funções para entrada de dados.
11 *
12 * Nota: C/C++ permitem o uso de ponteiros, que são instrumentos poderosos de programação.
13 * Com ponteiros podemos deixar os programas mais genéricos e, ao mesmo tempo, rápidos.
14 * Se você ainda não usou ponteiros leia o capítulo que cobre seu uso.
15 *
16 * Para compilar:
17 * g++ -std=c++11 main.cpp -o integral
18*/
19
20#include <cmath>
21#include <iostream>
22using namespace std;
23
24// Conceitos relacionados a função ================================================
```

```
25 // Variáveis globais usadas pela função F2G
26 double c0{0.0};
27 double c1{0.0};
28 double c2{0.0};
29
30 // Entrada de dados da função do segundo grau
31 void F2G_Entrada() {
32   cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
33   cout << "Entre␣com␣c0␣:␣";
34   cin  >> c0; cin.get();
35   cout << "Entre␣com␣c1␣:␣";
36   cin  >> c1; cin.get();
37   cout << "Entre␣com␣c2␣:␣";
38   cin  >> c2; cin.get();
39 }
40
41 /// Cálculo da função do segundo grau
42 double inline F2G(double x) {
43   return (c0 + c1*x + c2*x*x);
44 }
45
46 // Conceitos relacionados a integral ==========================================
47 // Variáveis globais usadas pelo método de integração
48 double limInf{0.0};
49 double limSup{1.0};
50 int numInt{101};
51
52 // Entrada de dados do método de integração
53 void Trapezio_Entrada() {
```

```
54   cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
55   cout << "Entre␣com␣Limite␣Inferior␣:␣";
56   cin  >> limInf; cin.get();
57   cout << "Entre␣com␣Limite␣Superior␣:␣";
58   cin  >> limSup; cin.get();
59   cout << "Entre␣com␣Numero␣Intervalos␣:␣";
60   cin  >> numInt; cin.get();
61 }
62
63 /// Cálculo da área usando trapézio.
64 /// Agora a função recebe um ponteiro para função a ser integrada.
65 /// A mudança é significativa, pois podemos usar a função Trapezio_Area() para cálcular
66 /// a área de qualquer função do tipo "double funcao(double)".
67 double Trapezio_Area( double(*ptr_funcao)(double) ) {
68   double area{0.0};
69   double x = limSup;
70   area = ptr_funcao(x) * 0.5;
71   x = limInf;
72   area += ptr_funcao(x) * 0.5; // area = area + funcao->f(limSup);
73   double dx = (limSup - limInf) / (double)numInt;
74   for( int i = 1; i < numInt ; i++ )    {
75           x += dx;
76           area += ptr_funcao(x);
77         }
78   area *= dx ;
79   return area;
80 }
81
82 /// Função principal do programa ================================================
```

```
83 int main(int argc, char **argv) {
84   F2G_Entrada();                          // Entrada de dados função
85   Trapezio_Entrada();                     // Entrada de dados trapézio
86
87   // Compare o número de parâmetros com a versão anterior.
88   double area = Trapezio_Area( F2G );     // Cálculo da área
89
90   cout << "\nArea = " << area ;           // Saída resultados
91
92   // Abaixo calcula área da função sin(x) e cos(x) entre 0-pi
93   // mostra que cálculo da área ficou genérico.
94   limInf = 0 ;
95   limSup = M_PI; // disponível na biblioteca cmath -> antiga math.h
96   cout << "\nArea sin(x) intervalo 0->pi = " << Trapezio_Area(sin) ;
97   cout << "\nArea cos(x) intervalo 0->pi = " << Trapezio_Area(cos) << endl;
98   cin.get();
99   return 0;
100 }
```

```
[bueno@bueno v0.5]$ make
g++ -std=c++11 main.cpp -o integral

[bueno@bueno v0.5]$ ./integral
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 1
Entre com c1 :
2
Entre com c2 : 3
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com Limite Superior : 1
Entre com Numero Intervalos : 101

Area = 3.00005
Area sin(x) intervalo 0->pi = 1.99984
Area cos(x) intervalo 0->pi = -2.52093e-16
```

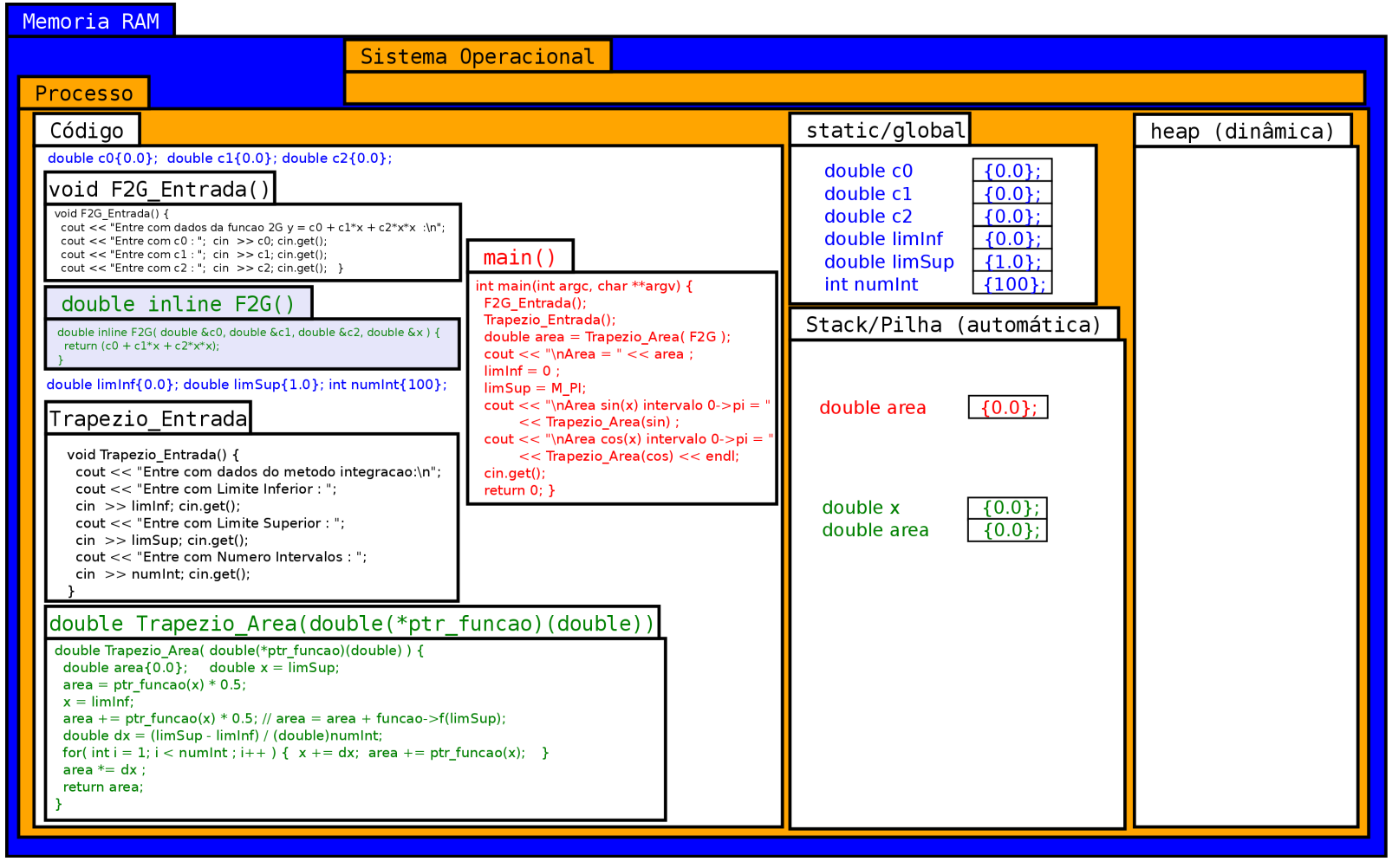


Figura 7.1: Projeto 1 - v0.1 - Integração Numérica: Organização memória RAM

Capítulo 8

v0.5-2 - Programação Estruturada - Ponteiro Função - Qt

8.1 Características

Nesta versão o que muda é a forma de interação do usuário com o programa, pois o mesmo disponibiliza diálogos para entrada de dados.

- Iremos mostrar:
  - o uso da biblioteca gráfica Qt5, uma biblioteca gráfica bastante amigável que funciona no GNU/Linux, Windows e Mac-OS-X, sendo multiplataforma.
  - o funcionamento do `qmake-qt5 -project` e `qmake-qt5`
  - o uso das classes de diálogos:
    - `QInputDialog::getDouble`
    - `QInputDialog::getInt`
    - `QMessageBox`
- Note que:
  - A biblioteca Qt foi instalada no sistema (https://www.qt.io/).
  - Acessamos um site (ou um livro) que explica o funcionamento dos diálogos de Qt.

- Os arquivos da biblioteca Qt são incluídos no nosso programa.
- Criamos objetos derivados do tipo `QDialog`.
- Usamos os objetos criados.

8.2 Códigos

Listing 8.1: Programa para cálculo da integral do trapézio da função y = c0+c1*x+c2*x*x.

```
1/** Copyright André D. Bueno
2 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
3 * Projeto: Implementação de uma pequena biblioteca de classes para funções e cálculo de integral numérica.
4 *
5 * Características da versão:
6 * - Usa diálogos da biblioteca Qt5 para obter dados de entrada e mostrar resultados.
7 * - QInputDialog::getDouble
8 * - QInputDialog::getInt
9 * - QMessageBox
10 *
11 * Para compilar no Fedora:
12 * qmake-qt5 -project && qmake-qt5 && make
13 * ou no Debian
14 * qmake -project && qmake && make
15 *
16 * Adicionar no arquivo .pro instruções :
17 * CONFIG += c++14
18 * QT += widgets
19*/
20// QT
21#include <QtWidgets/QApplication> // uma aplicação (http://doc.qt.io/qt-5/qapplication.html)
22#include <QtWidgets/QInputDialog> // um diálogo (http://doc.qt.io/qt-5/qinputdialog.html)
23#include <QtWidgets/QMessageBox>  // uma mensagem (http://doc.qt.io/qt-5/qmessagebox.html)
24
```

```
25 // Código anterior
26 #include <cmath>                    // funções sin e cos
27 #include <iostream>                 // cin e cout
28 #include <string>                   // to_string converte número para string
29 #include <sstream>                  // fornece ostringstream ostream e string
30 using namespace std;
31
32 // Função auxiliar para facilitar o uso de QInputDialog::getDouble
33 inline double getDouble (const char* titulo, const char* mensagem) {
34         return QInputDialog::getDouble( nullptr, // Janela pai (nullptr = sem pai
35                                         titulo,  // Titulo
36                                       mensagem);  // Mensagem
37 }
38
39 // Conceitos relacionados a função
40 // Variáveis globais usadas pela função F2G
41 double c0{0.0};
42 double c1{0.0};
43 double c2{0.0};
44
45 // Entrada de dados da função do segundo grau
46 void F2G_Entrada() {
47   cout << "Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :\n";
48
49   c0 = getDouble( "Entre com","double c0 =");
50   cout << "c0 = " << c0 << endl;
51   c1 = getDouble( "Entre com","double c1 =");
52   cout << "c1 = " << c1 << endl;
53   c2 = getDouble("Entre com","double c2 =");    // Usando função auxiliar
```

```
54   cout << "c2 = " << c2 << endl;
55 }
56
57 /// Cálculo da função do segundo grau
58 double inline F2G(double x) {  return (c0 + c1*x + c2*x*x); }
59
60 // Conceitos relacionados a integral.
61 // Variáveis globais usadas pelo método de integração
62 double limInf{0.0};
63 double limSup{1.0};
64 int numInt{101};
65
66 // Entrada de dados do método de integração
67 void Trapezio_Entrada() {
68   limInf = getDouble( "Entre com  ","double limInf =");
69   cout << "limInf = " << limInf << endl;
70   limSup = getDouble( "Entre com  ","double limSup =");
71   cout << "limSup = " << limSup << endl;
72   numInt = QInputDialog::getInt( nullptr, "Entre com  ","int numInt =",numInt,1,101,2);
73   cout << "numInt = " << numInt << endl;
74 }
75
76 /// Cálculo da área usando trapézio.
77 /// Agora a função recebe um ponteiro para função a ser integrada.
78 /// A mudança é significativa, pois poderemos usar a função Trapezio_Area() para cálcular
79 /// a área de qualquer função do tipo "double funcao(double)".
80 double Trapezio_Area( double(*funcao)(double) ) {
81   double area{0.0};
82   double x = limSup;
```

```
83   area = funcao(x) * 0.5;
84   x = limInf;
85   area += funcao(x) * 0.5; // area = area + funcao->f(limSup);
86   double dx = (limSup - limInf) / (double)numInt;
87   for( int i = 1; i < numInt ; i++ )    {
88           x += dx;
89           area += funcao(x);
90         }
91   area *= dx ;
92   return area;
93 }
94
95 /// Função principal do programa
96 int main(int argc, char **argv) {
97   QApplication app(argc, argv);          // Cria aplicação do Qt (gráfica/separada)
98
99   F2G_Entrada();                         // Entrada de dados função
100  Trapezio_Entrada();                    // Entrada de dados trapézio
101
102  double area = Trapezio_Area( F2G );    // Cálculo da área
103  cout << "\nTrapezio_Area(F2G)=␣" << area ;    // Saída resultados modo terminal
104
105  // Como usar um QMessageBox para mostrar resultado.
106  ostringstream os;                      // cria ostringstream, funciona como cout e string
107  os << area;
108  // Uso avançado de QMessageBox
109  QMessageBox msgBox;                              // cria objeto
110  msgBox.setWindowTitle("Resutado␣:");             // seta título
111  msgBox.setIcon(QMessageBox::Information);        // seta ícone
```

```
112   msgBox.setText("Trapezio_Area(F2G)␣=␣");       // seta texto
113   msgBox.setInformativeText(os.str().c_str());  // seta texto adicional
114   msgBox.setStandardButtons(QMessageBox::Ok);   // seta os botões
115   msgBox.exec();                                // executa o diálogo
116
117   // Abaixo calcula área da função sin(x) e cos(x) entre 0-pi
118   // mostra que cálculo da área ficou genérico.
119   limInf = 0.0 ;
120   limSup = M_PI; // valor de PI disponível na biblioteca cmath -> math.h
121   area  = Trapezio_Area(sin) ;
122   cout << "\nTrapezio_Area(sin)␣␣␣0->pi␣=␣" << area ;
123
124   // Uso simplificado de QMessageBox, informações via construtor:
125   // QMessageBox(ícone, título, mensagem)
126   QMessageBox msgBox1(QMessageBox::Information,// Icone
127                       "Resultado␣:",                        // Ttíulo
128                       string("Trapezio_Area(sin)=" + to_string(area)).c_str()); // Mensagem
129   msgBox1.exec();
130
131   area = Trapezio_Area(cos);
132   cout << "\nTrapezio_Area(cos)␣␣␣0->pi␣=␣" << area << endl;
133   // Uso simplificado de QMessageBox
134   // Usa função QMessageBox::information que é estática e pública
135   QMessageBox::information(nullptr, "Resultado␣:",
136                            string("Trapezio_Area(cos)="+ to_string(area)).c_str());
137   app.quit();
138   return 0;
139}
```

Listing 8.2: Arquivo pro - configurações do Qt.

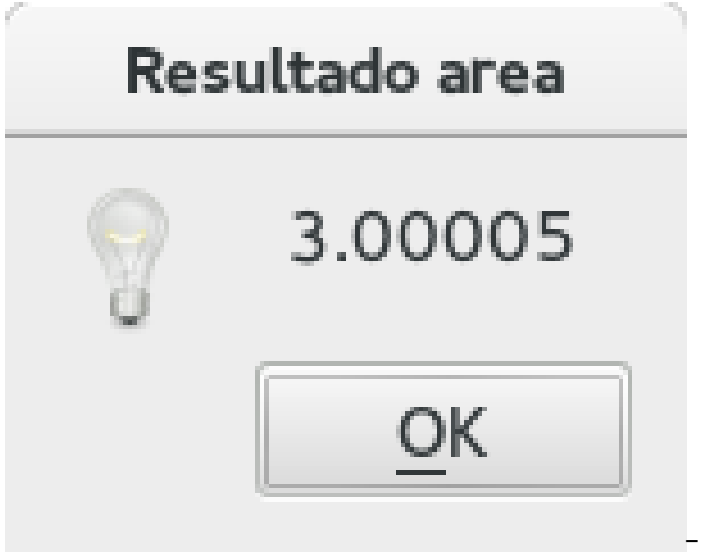
```
#############################################################################
# Automatically generated by qmake (3.0) dom mar 27 14:32:36 2016
#############################################################################

TEMPLATE = app
TARGET = qt5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText
INCLUDEPATH += .

## Acesso a QInputDialog
QT += widgets

CONFIG += c++11
#MAKE_CXXFLAGS += -std=c++11

# Input
SOURCES += main.cpp
```

```
[bueno@bueno v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText]$ make
g++ -c -pipe -O2 -g -pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -O2 -std=c++0x -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/qt5 -isystem /usr/include/qt5/QtWidgets -isystem /usr/include/qt5/QtGui -isystem /usr/include/qt5/QtCore -I. -I/usr/lib64/qt5/mkspecs/linux-g++ -o main.o main.cpp
g++ -Wl,-O1 -Wl,-z,relro -o qt5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText main.o   -lQt5Widgets -lQt5Gui -lQt5Core -lGL -lpthread


[bueno@bueno v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText]$ ls -lah ../v0.5
total 408K
-rw-r--r--.  1 bueno bueno 3.3K Mar 29 14:32 main.cpp <<---1
-rw-rw-r--.  1 bueno bueno 6.9K Mar 29 14:58 main.o   <<---2
-rw-r--r--.  1 bueno bueno   58 Jun 10  2014 Makefile <<---3
-rwxrwxr-x.  1 bueno bueno  14K Mar 29 14:58 integral <<---4

[bueno@bueno v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText]$ ls -lah
total 1.4M
-rw-r--r--.  1 bueno bueno 4.9K Mar 29 14:35 main.cpp <<---1
-rw-rw-r--.  1 bueno bueno 807K Mar 29 14:59 main.o   <<---2
-rw-rw-r--.  1 bueno bueno  15K Mar 29 14:57 Makefile <<---3
-rwxrwxr-x.  1 bueno bueno 472K Mar 29 14:59 v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText <<---4
-rw-rw-r--.  1 bueno bueno  422 Mar 29 14:56 v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText.pro

[bueno@bueno v0.5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText]$ ./qt5-2-Uso-QInputDialog-getInt-getDouble-getItem-getText
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
c0 = 1
```

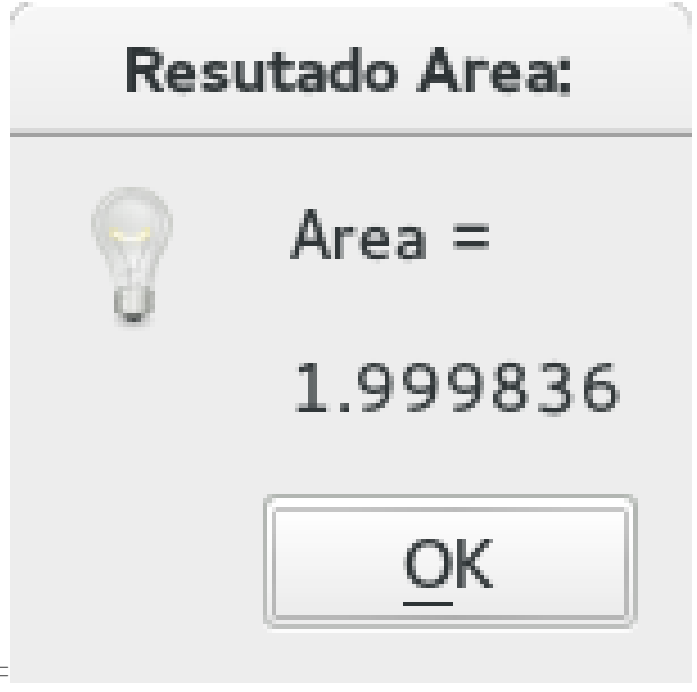
```
40 c1 = 2
41 c2 = 3
42 limInf = 0
43 limSup = 1
44 numInt = 101
45
46 Area = 3.00005
47 Area sin(x) intervalo 0->pi = 1.99984
48
49 Area cos(x) intervalo 0->pi = -2.52093e-16
```

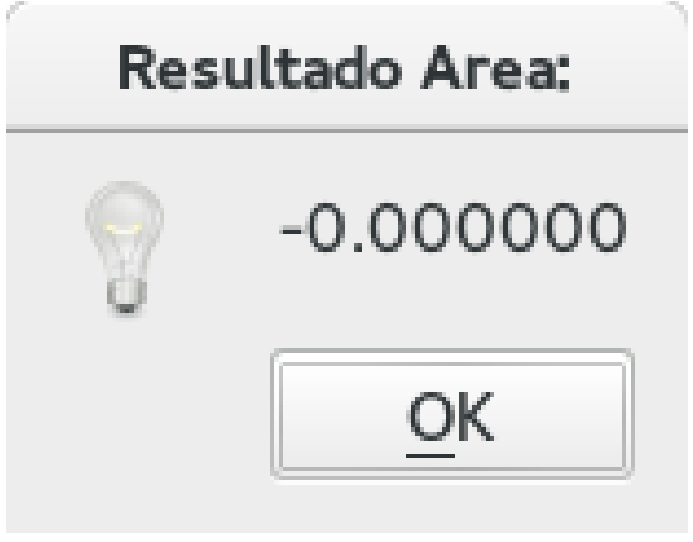

Entre com
double c0 =
1.0
Cancel
OK
->
Entre com
double c1 =
2.0
Cancel
OK
->
Entre com
double c2 =
3.0
Cancel
OK
->
Entre com
double limInf =
0.0
Cancel
OK
->
Entre com
double limSup =
1.0
Cancel
OK
->
Entre com
int numInt =
100
Cancel
OK

->

F2G = 

->

sin = 

-> cos = 

Capítulo 9

v0.6 - Programação Orientada a Objeto - UML - Classes CFun2G e CIntTrapezio

9.1 Características

- Nos exemplos que vimos até aqui usamos programação estruturada. Na versão v0.5 tivemos alguns avanços, como a possibilidade de calcular a área de qualquer função do tipo "`double funcao(double);`". Mas temos alguns problemas:
  - As variáveis da função do segundo grau c0, c1, c2, são globais e podem ser utilizadas por qualquer função que venha ser criada.
  - Se quizermos ter, simultaneamente, duas funções, teremos de refazer todo código. Isto ocorre porque na programação estruturada não existe um vínculo direto entre os coeficientes da função (c0, c1, c2) e a função em sí. Mesmo sendo a relação de fato muito próxima e direta.
- Programa em C++ com duas classes.
  - Foram criadas as classes `CFun2G` e `CIntTrapezio`, as mesmas representam uma função de segundo grau e o método de integração do trapézio.
  - Os atributos estão públicos e o acesso é direto.
  - Os métodos foram declarados e definidos dentro das classes (nos arquivos .h).

- Como passaremos a ter um grande número de arquivos .h e .cpp, os códigos estão dentro do diretório src.
- O controle ainda esta dentro de `main()`.
- Também passamos a usar o programa `cmake` para gerenciar a construção/compilação do software.

9.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 9.2 o diagrama de classes e a Figura 9.3 o diaqrama de sequência.

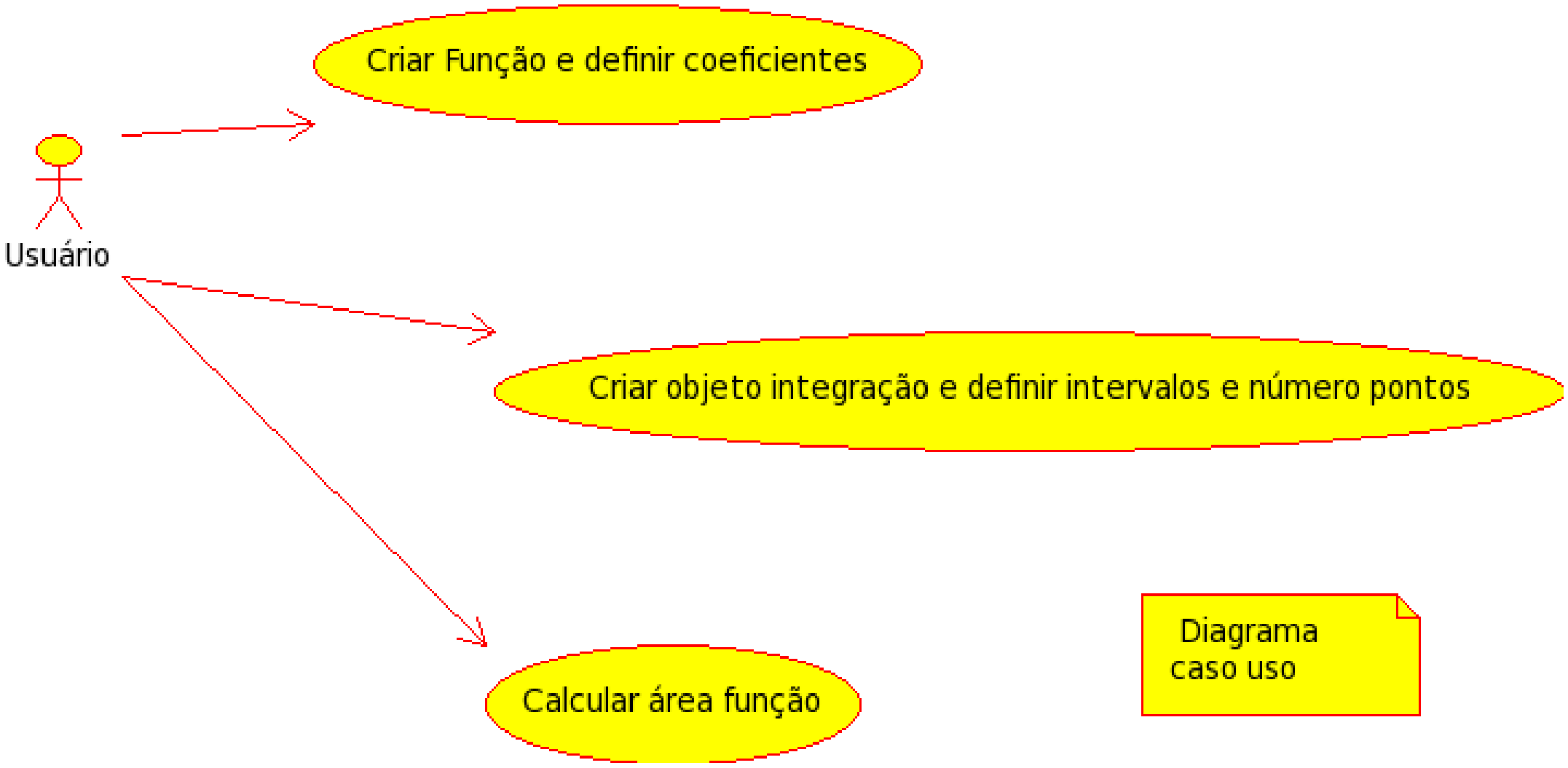


Figura 9.1: Diagrama de caso de uso - versão 0.6

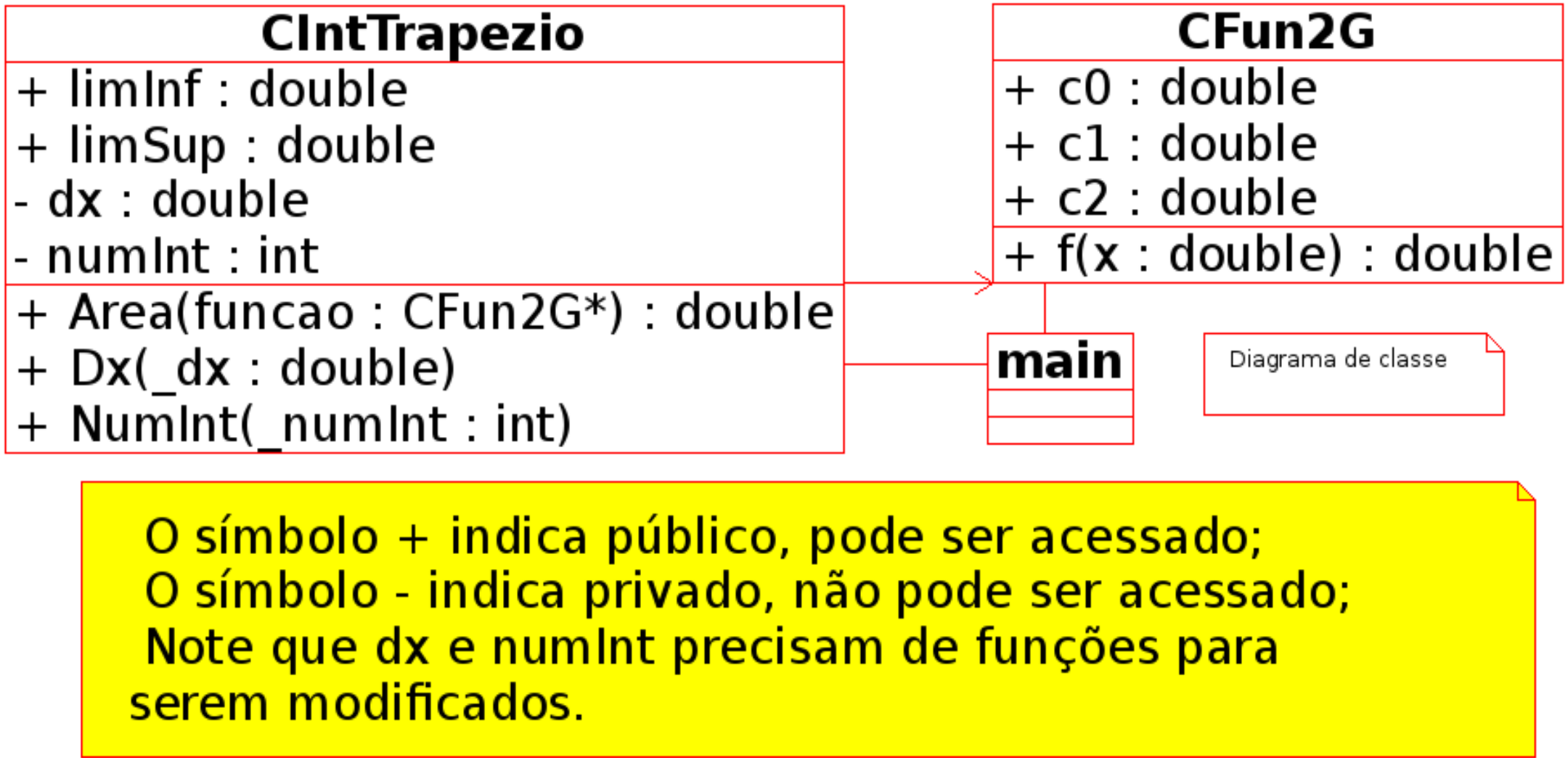


Figura 9.2: Diagrama de classe - versão 0.6

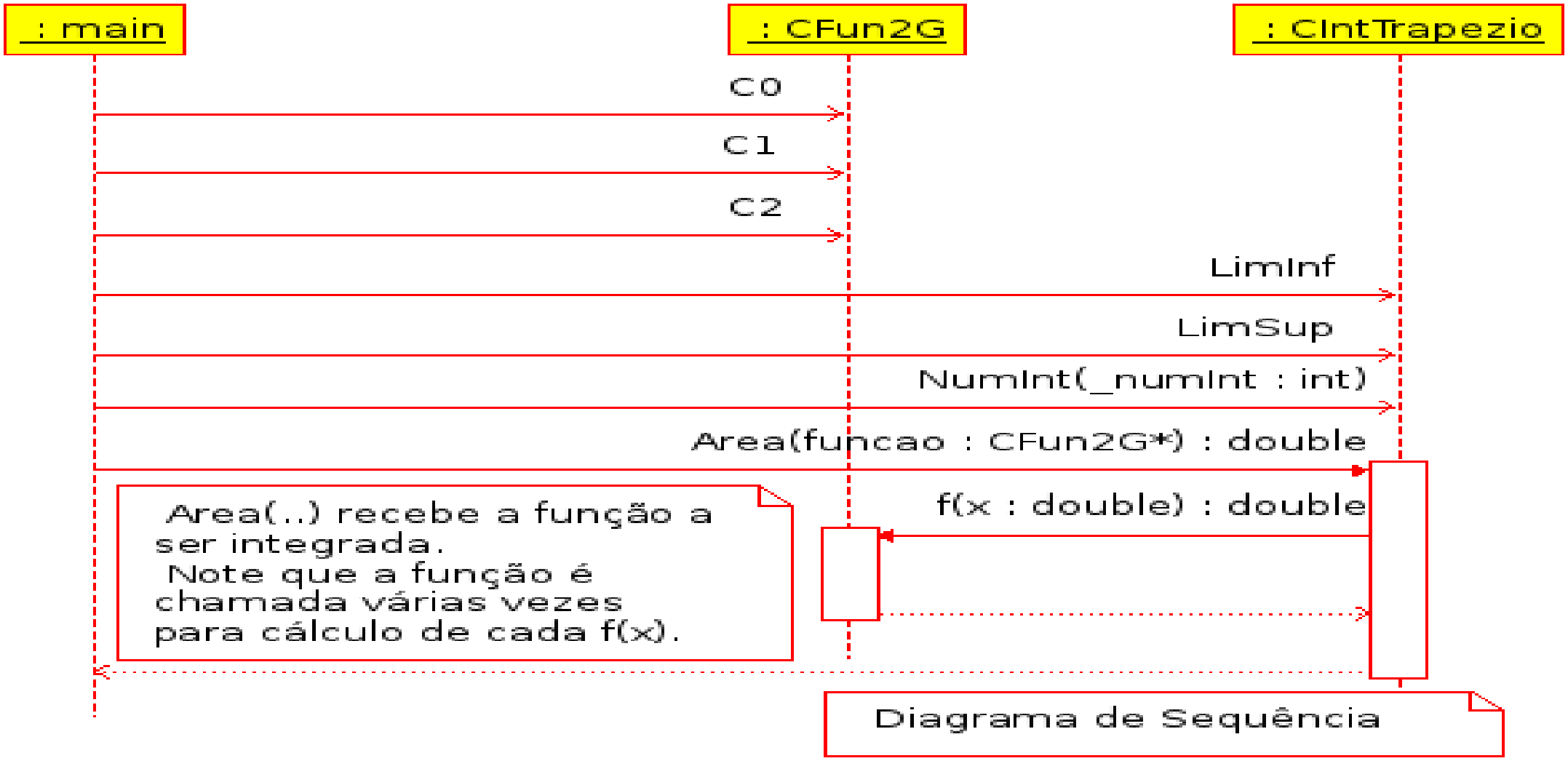


Figura 9.3: Diagrama de sequência - v0.6

9.3 Códigos

Listing 9.1: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1 /**************************************************************************
2 CFun2G.h - Copyright André D. Bueno
3 Exemplo didático ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções.
5 **************************************************************************/
6 #ifndef CFun2G_H    // Se ainda não definida a variável CFun2G_H
7 #define CFun2G_H    // definir a mesma e compilar o código
8
9 #include <iostream>// Inclui acesso a cin e cout
10
11 /// Classe que representa uma funcao do segundo grau y = c0 + c1*x + c2*x*x;.
12 class CFun2G
13 {
14  public:          //   Área de atributos e métodos públicos
15   double c0=0.0; ///< Representa coeficiente linear.
16   double c1=0.0; ///< Representa coeficiente angular.
17   double c2=0.0; ///< Representa coeficiente quadrático.
18
19  /// Solicita entrada de dados
20  void Entrada() {
21   std::cout << "Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x :\n";
22   std::cout << "Entre com c0 : ";
23   std::cin  >> c0;
24   std::cout << "Entre com c1 : ";
```

```
25   std::cin  >> c1;
26   std::cout << "Entre␣com␣c2␣:␣";
27   std::cin  >> c2; std::cin.get();
28 }
29
30 /// Calcula valor de y da função f na posição x, calcula y = c0 + c1 * x + c2 *x*x;.
31 double inline f ( double x ) {
32      return c0 + c1*x + c2*x*x;
33     }
34 };
35 #endif                // fim do bloco de preprocessamento iniciado em #ifndef CFun2G_H
```

Listing 9.2: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1 #include "CFun2G.h"      // Apenas inclui o arquivo .h
```

Listing 9.3: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /*************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca para cálculo de integral numérica.
5 *************************************************************************/
6 #ifndef CIntTrapezio_H
7 #define CIntTrapezio_H
8
9 #include <iostream>
10 #include "CFun2G.h"     // Inclui a(s) classe(s) que vamos usar.
11 /// Classe que representa o método do Trapézio, usado para cálculo da área das funções.
12 class CIntTrapezio
13 {
14  private:
15   double dx=0.00990099;///< Intervalo dx.
16   int    numInt=101;    ///< Numero intervalos.
17  public:
18   double limInf=0.0;    ///< Limite Inferior do intervalo de integração.
19   double limSup=1.0;    ///< Limite Superior do intervalo de integração.
20   double area=0.0;      ///< Valor da área calculada.
21
22   /// Solicita entrada de dados
23   void Entrada() {
24     std::cout << "Entre com dados do metodo integracao:\n";
25     std::cout << "Entre com Limite Inferior : ";
26     std::cin >> limInf;
27     std::cout << "Entre com limite Superior : ";
28     std::cin >> limSup;
```

```
29    std::cout << "Entre␣com␣numero␣intervalos␣:␣";
30    int n;
31    std::cin >> n; std::cin.get();
32    NumInt( n );
33    }
34
35  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
36  /// Note que recebe ponteiro para objeto do tipo CFun2G.
37  double Area ( CFun2G* funcao ) {
38        area =  funcao->f( limInf ) * 0.5;
39        double x = limInf;
40        for( int i = 1; i < numInt ; i++ ) {
41                x += dx;
42                area += funcao->f( x );
43        }
44        area += funcao->f( limSup ) * 0.5;
45        area *= dx ;
46        return area;
47  }
48
49  /// Seta o valor de dx. Note que após redefinir dx, recalcula numero intervalos.
50  void Dx ( double _dx ) {
51        dx = _dx;
52    numInt = (limSup - limInf) / dx;
53  }
54
55  /// Seta o valor de numInt. Observe que após redefinir numInt, recalcula dx.
56  void NumInt ( int _numInt ) {
57    numInt = _numInt;
```

```
58          dx = (limSup - limInf) / double(numInt);
59    }
60 };
61 #endif
```

Listing 9.4: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1 #include "CIntTrapezio.h"
```

Listing 9.5: Programa para cálculo da integral do trapézio: Usa classes CFun2G e CIntTrapezio.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Programa em C++ com duas classes.
 * - Como passaremos a ter um grande número de arquivos .h e .cpp,
 * os códigos agora estão dentro do diretório src.
 * - Foram criadas as classes CFun2G e CIntTrapezio, as mesmas representam
 * uma função de segundo grau e o método de integração do trapézio.
 * - Os atributos estão públicos e o acesso é direto.
 * - Os métodos foram declarados e definidos dentro das classes (arquivos .h).
 *
 * Note que a grande diferença é que agora os atributos que eram globais,
 * e que poderiam ser acessador por qualquer função, foram movidos para dentro das classes.
 * O mesmo ocorreu com as funções, que agora pertencem as classes.
 * Na prática a orientação a objetos realiza um encapsulamento de dados e funções,
 * e fornece, naturalmente, uma maior coesão e generalidade aos códigos.
 *
 * Para compilar:
 * - Passamos a usar o programa cmake para gerar automaticamente os arquivos Makefile
 * (arquivos que executam as instruções de compilação).
 * O cmake é obtido em http://www.cmake.org/.
 *
 * A compilação passa a ser da seguinte forma:
 * cmake .    [somente na primeira compilação, gera o arquivo Makefile]
 * make       [sempre que tiver que recompilar o código, executa a compilação]
```

```
29 * mas também podemos usar a forma antiga:
30 * cd v0.6/src
31 * g++ -std=c++11 CFun2G.cpp CIntTrapezio.cpp main.cpp  -o integral
32 *
33 * Nota:
34 * Códigos compilados usando o compilador da gnu (http://www.gcc.gnu.org/) na versão:
35 * g++ (GCC) 5.1.1 20150422 (Red Hat 5.1.1-1)
36 */
37 #include <iostream>
38 #include "CFun2G.h"              // Inclui acesso a classe CFun2G
39 #include "CIntTrapezio.h"        // Inclui acesso a classe CIntTrapezio
40 using namespace std;
41
42 /// Função principal do programa
43 int main() {
44   // Cria objeto funcao2G e a seguir realiza a entrada de atributos do objeto
45   CFun2G funcao2G;
46   funcao2G.Entrada();
47
48   // Cria objeto trapezio e a seguir realiza a entrada de atributos do objeto
49   CIntTrapezio trapezio;
50   trapezio.Entrada();
51
52   // Cálculo da área
53   cout << "\nArea = " << trapezio.Area(&funcao2G) <<  "\n";
54   cin.get();
55   return 0;
56 }
```

```
[bueno@localhost src]$ ./integracaonumerica
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 1
Entre com c1 : 2
Entre com c2 : -2
Entre com dados do metodo integracao:
Entre com Limite Inferior : 1
Entre com limite Superior : 10
Entre com numero intervalos : 100

Area = -558.024
```

Capítulo 10

v0.7 - Programação Orientada a Objeto - UML - Construtores

10.1 Características

- Métodos
  - Incluí métodos `get`/`set` para leitura e modificação dos atributos. Por exemplo:
    - `double limInf; // é o atributo`
    - `void LimInf( double novoLimInf ) { limInf = novoLimInf ; } // set`
    - `double LimInf() { return limInf } // get`
- Construtures
  - Adicionados construtores.
  - Uso da palavra chafe default para indicar construtor default.
  - Adicionados exemplos com criação dos objetos de integração usando os construtores.

10.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 10.1 o diagrama de classes e a Figura 10.2 o diaqrama de sequência.

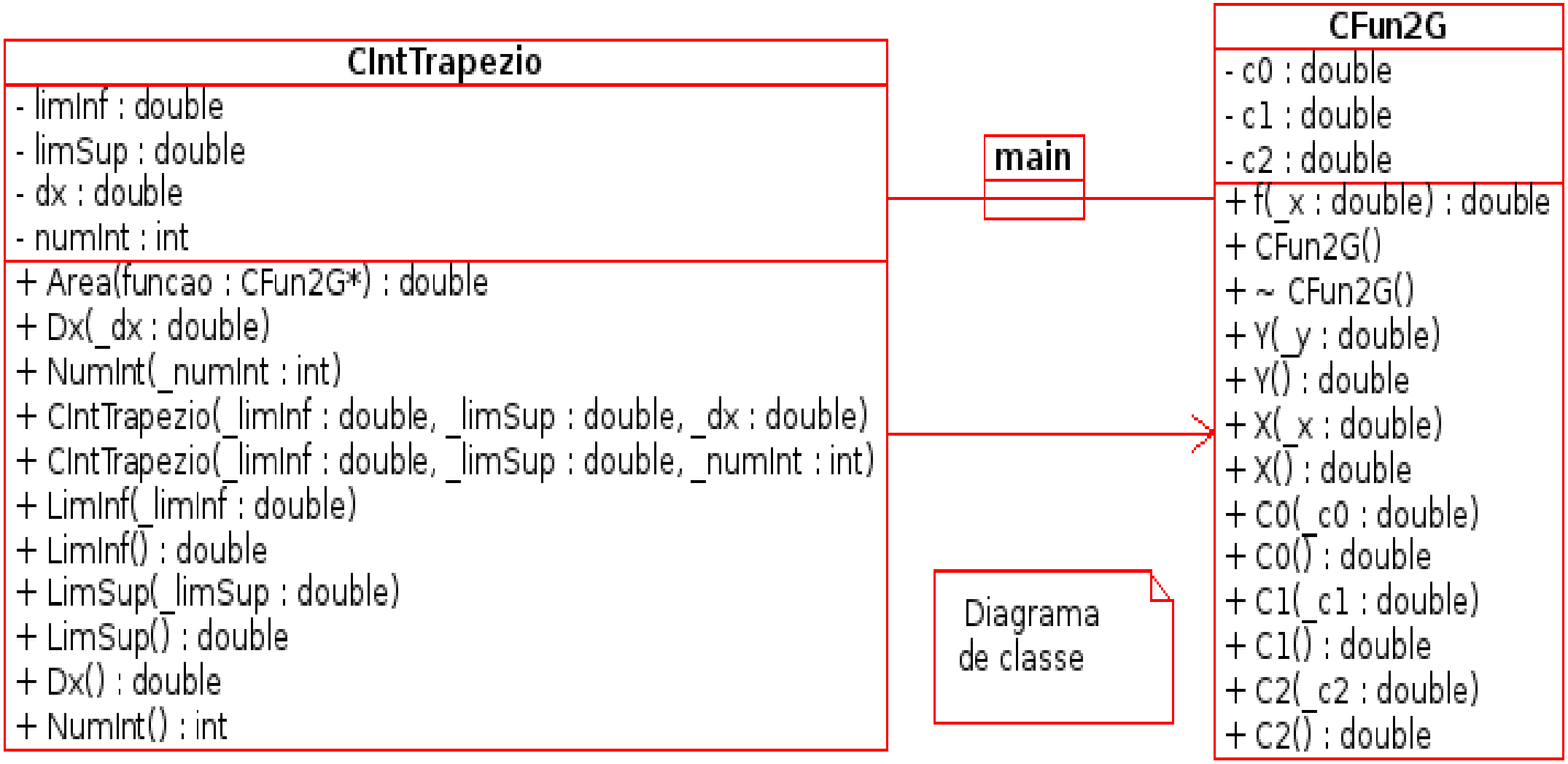


Figura 10.1: Diagrama de classe - versão 0.7

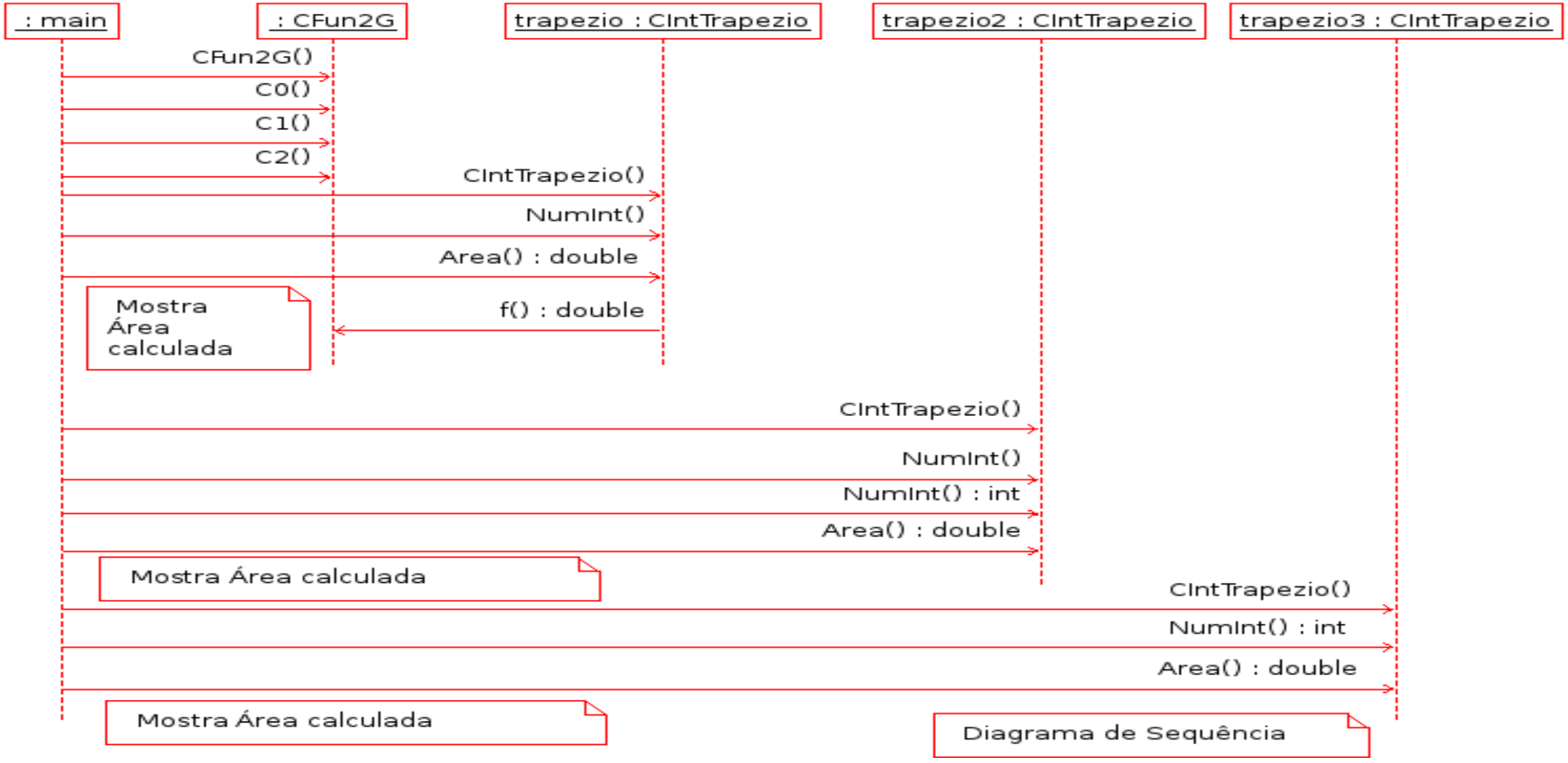


Figura 10.2: Diagrama de sequência - v0.7

10.3 Códigos

Listing 10.1: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1 /**************************************************************************
2 CFun2G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CFun2G_H          // Se ainda não foi definida variável CFun2G_H
8 #define CFun2G_H          // defina a variável CFun2G_H e continue compilando;
9                           // o código abaixo só é compilado 1x.
10 /// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
11 class CFun2G
12 {
13  protected:
14   double c0={0.0};        ///< Representa coeficiente linear.
15   double c1={0.0};        ///< Representa coeficiente angular.
16   double c2={0.0};        ///< Representa coeficiente quadrático.
17
18  public:
19  /// Construtor default
20  CFun2G()                        {} // Aloca espaço de memória, cria c0,c1,c2
21
22  CFun2G(double _c0,double _c1,double _c2){
23      c0 = _c0;
24      c1 = _c1;
```

```
25      c2 = _c2;
26 }
27 // Aloca espaço de memória, cria c0,c1,c2
28
29  /// Destrutor
30  ~CFun2G()                       {}
31
32   /// Declaração da função que calcula y = c0 + c1 * x + c2 *x*x;.
33   /// @return double
34   /// @param  x
35   double f ( double x  )          {  return c0 + c1*x + c2*x*x; }
36
37   /// Seta valor de c0.
38   /// @param _c0 novo valor de c0.
39   inline void C0 ( double _c0 ) {  c0 = _c0; };
40
41   /// Retorna valor de c0.
42   /// @return retorna valor de c0.
43   inline double C0 ( )            {  return c0; };
44
45   /// Seta valor de c1.
46   /// @param _c1 novo valor de c1
47   inline void C1 ( double _c1 ) {  c1 = _c1; };
48
49   /// Retorna valor de c1
50   /// @return retorna valor de c1
51   inline double C1 ( )            {  return c1; };
52
53   /// Seta valor de c2
```

```
54   /// @param _c2 novo valor de c2
55   inline void C2 ( double _c2 ) { c2 = _c2; };
56
57   /// Retorna valor de c2
58   /// @return retorna valor de c2
59   inline double C2 ( )          { return c2; };
60
61   /// Solicita entrada de dados
62   void Entrada();
63};
64#endif // fim do bloco CFun2G_H
```

Listing 10.2: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1// CFun2G.cpp
2#include "CFun2G.h"
3
4#include <iostream>
5using namespace std;
6
7// Note o retorno void, o nome Entrada, e que não recebe parâmetros.
8void CFun2G::Entrada() {
9   cout << "Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x :\n";
10  cout << "Entre com c0 : ";
11  cin  >> c0;
12  cout << "Entre com c1 : ";
13  cin  >> c1;
14  cout << "Entre com c2 : ";
15  cin  >> c2; cin.get();
```

```
16 }
```

Listing 10.3: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CIntTrapezio_H
8 #define CIntTrapezio_H
9
10 #include "CFun2G.h"
11 #include <iostream>
12
13 /// Representa método numérico trapésio para cálculo área funções.
14 class CIntTrapezio
15 {
16  protected:
17   double limInf={0.0}; ///< Limite Inferior do intervalo de integração
18   double limSup={1.0}; ///< Limite Superior do intervalo de integração
19   double dx={0.01};    ///< Intervalo dx
20   int    numInt={100}; ///< Numero intervalos
21   double area={0.0};   ///< Valor da área calculada
22
23  public:
24  /// Construtor default
25  CIntTrapezio() {
26      std::cout << "\nPassou␣pelo␣construtor␣CIntTrapezio()␣default.\n";
27 };
28
```

```
29 /// Construtor sobrecarregado
30 CIntTrapezio(double _limInf, double _limSup,  double _dx) {
31   limInf = _limInf;    // atributos definidos após sua construção
32   limSup = _limSup;
33   Dx( _dx );           // seta dx e calcula número intervalos.
34   std::cout << "\nPassou␣pelo␣construtor␣CIntTrapezio(d,d,d)␣sobrecarregado.\n";
35 }
36
37 /// Construtor sobrecarregado
38 // Note que também tem 3 parâmetros mas o tipo do terceiro parâmetro é diferente.
39 CIntTrapezio(double _limInf, double _limSup,  int _numInt)
40 : limInf(_limInf)      // Atributo limInf definido na sua construção
41 ,limSup(_limSup) {     // Atributo limSup definido na sua construção
42   NumInt(_numInt);     // Seta numInt e calcula dx.
43   std::cout << "\nPassou␣pelo␣construtor␣CIntTrapezio(d,d,i)␣sobrecarregado.\n";
44 }
45
46 /// Destrutor
47 ~CIntTrapezio() {
48       std::cout << "\nPassou␣pelo␣destrutor.\n";
49};
50
51  /// Solicita entrada de dados
52  void Entrada();
53
54 /// Calcula área da função no intervalo limInf->limSup considerando numInt.
55  double Area ( CFun2G* funcao );
56
57  /// Seta valor de limInf
```

```
58  inline void LimInf ( double _limInf  ) { limInf = _limInf ; };
59
60  /// Retorna valor de limInf
61  /// @return retorna valor de limInf
62  inline double LimInf ( )              { return limInf; };
63
64  /// Seta valor de limSup
65  /// @param _limSup Seta novo valor de  limSup
66  inline void LimSup ( double _limSup ) { limSup = _limSup; };
67
68  /// Retorna valor de limSup
69  /// @return retorna valor de limSup
70  inline double LimSup ( )              { return limSup; } ;
71
72  /// Seta valor de dx.
73  /// Note que após redefinir dx, recalcula numero intervalos.
74  /// O valor de dx multiplicado por numInt deve ser igual ao intervalo.
75  inline void Dx ( double _dx )         { dx = _dx; numInt = (limSup - limInf) / dx;};
76
77  /// Retorna valor de dx
78  inline double Dx ( )                  { return dx;};
79
80  /// Seta valor de numInt.
81  /// Observe que após redefinir numInt, recalcula dx.
82  /// @param _numInt the new value of numInt
83  inline void NumInt ( int _numInt )  { numInt = _numInt;
84                                        dx = (limSup - limInf) / double(numInt);   };
85
86   /// Retorna valor de numInt
```

```
87   inline int NumInt ( )                  { return numInt; } ;
88 };
89 #endif
```

Listing 10.4: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1 #include <iostream>
2 using namespace std;
3
4 #include "CFun2G.h"
5 #include "CIntTrapezio.h"
6
7 void CIntTrapezio::Entrada() {
8   cout << "Entre com dados do metodo integracao:\n";
9   cout << "Entre com Limite Inferior : ";
10  cin >> limInf;
11  cout << "Entre com limite Superior : ";
12  cin >> limSup;
13  cout << "Entre com numero intervalos : ";
14  int numInt;
15  cin >> numInt; cin.get();
16  NumInt( numInt );
17 }
18
19 double CIntTrapezio::Area (CFun2G* funcao ) {
20  area =  funcao->f( limInf ) * 0.5;
21  double x = limInf;
22  for( int i = 1; i < numInt ; i++ ) {
23           x += dx;
```

```
24            area += funcao->f( x );
25  }
26  area += funcao->f( limSup ) * 0.5;
27  area *= dx ;
28 return area;
29 }
```

Listing 10.5: Programa para cálculo da integral do trapésio: Usa classes CFun2G e CIntTrapezio.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Programa em C++ com duas classes.
 * - Atributos privados.
 * - Métodos get/set para leitura/modificação dos atributos.
 * - Métodos de Entrada e cálculo movidos para arquivos .cpp.
 * - Adicionados construtores.
 * - Adicionados exemplos com criação dos objetos de integração usando os construtores.
 *
 * Para compilar:
 * Como usa recursos de C++11, para compilar use:
 * g++ -std=c++11 CFun2G.cpp CIntTrapezio.cpp main.cpp
 *
 * Note que a vantagem do uso dos construtores esta em inicializar os atributos
 * com os valores que nos interessam, sem a necessidade de chamar cada um
 * dos métodos Set para setar os atributos.
*/
#include <iostream>
#include <iomanip>
#include "CFun2G.h"
#include "CIntTrapezio.h"
using namespace std;

/// Função principal do programa
```

```
29 int main() {
30   // Cria objeto função2G e a seguir realiza a entrada de atributos do objeto
31   CFun2G funcao2G; //Construtor default
32   funcao2G.Entrada();
33
34   cout << "Vamos testar a classe trapezio usando construtor default e sobrecarregados:\n";
35   // Usa construtor default
36   CIntTrapezio trapezio;
37   cout << "\nNumero pontos trapezio = " << trapezio.NumInt() << "\t";
38   cout << "Area (1) = " << setprecision(12) << trapezio.Area(&funcao2G) << "\n";
39
40   // Usa construtor default e depois NumInt para mudar número de pontos
41   CIntTrapezio trapezio2;
42   trapezio2.NumInt(1000);                          // Calculo da área com mais precisão
43   cout << "\nNumero pontos trapezio2 = " << trapezio2.NumInt() <<  "\t";
44   cout << "Area (2) = " << setprecision(12) << trapezio2.Area(&funcao2G) << "\n";
45
46   // Usa construtor sobrecarregado, passa limInf, limSup, dx
47   CIntTrapezio trapezio3( 0.0, 1.0 , 1.0e-4);
48   cout << "\nNumero pontos trapezio3 = " << trapezio3.NumInt() << "\t";
49   cout << "Area (3) = " << setprecision(12) << trapezio3.Area(&funcao2G) << "\n";
50
51   // Usa construtor sobrecarregado, passa limInf, limSup, numInt
52   CIntTrapezio trapezio4( 0.0, 1.0 , 100000);
53   cout << "\nNumero pontos trapezio4 = " << trapezio4.NumInt() << "\t";
54   cout << "Area (4) = " << setprecision(12) << trapezio4.Area(&funcao2G) << "\n\n\n";
55   //cin.get();
56
57   // Como agora temos set/get para cada atributo da função podemos mudar a função e recalcular
```

```
58   cout << "Entre␣com␣novo␣c0␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
59   cout << "Entre␣com␣c0␣:␣";
60   double _c0;
61   cin >> _c0; cin.get();
62   funcao2G.C0(_c0);
63   cout << "Area␣(1)␣=␣" << setprecision(12) << trapezio.Area(&funcao2G) << "\n";
64   cout << "Area␣(2)␣=␣" << setprecision(12) << trapezio2.Area(&funcao2G) << "\n";
65   cout << "Area␣(3)␣=␣" << setprecision(12) << trapezio3.Area(&funcao2G) << "\n";
66   cout << "Area␣(4)␣=␣" << setprecision(12) << trapezio4.Area(&funcao2G) << "\n\n\n";
67
68
69   return 0;
70 }
```

```
[bueno@bueno src]$ ./integracaonumerica
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 5
Entre com c1 : 2
Entre com c2 : 0
Vamos testar a classe trapezio usando construtor default e sobrecarregados:

Numero pontos trapezio = 100    Area (1) = 6

Numero pontos trapezio2 = 1000  Area (2) = 6

Numero pontos trapezio3 = 10000 Area (3) = 6

Numero pontos trapezio4 = 100000        Area (4) = 6


Entre com novo c0 da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 10
Area (1) = 11
Area (2) = 11
Area (3) = 11
Area (4) = 11
```

Capítulo 11

v0.8 - Programação Orientada a Objeto - UML - Hierarquia Classes - Métodos Virtuais

11.1 Características

- Acrescenta hierarquisa classes `CIntegral` e `CFuncao`.
- Acrescenta métodos virtuais.

11.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 11.1 o diagrama de classes e a Figura 10.2 o diaqrama de sequência.

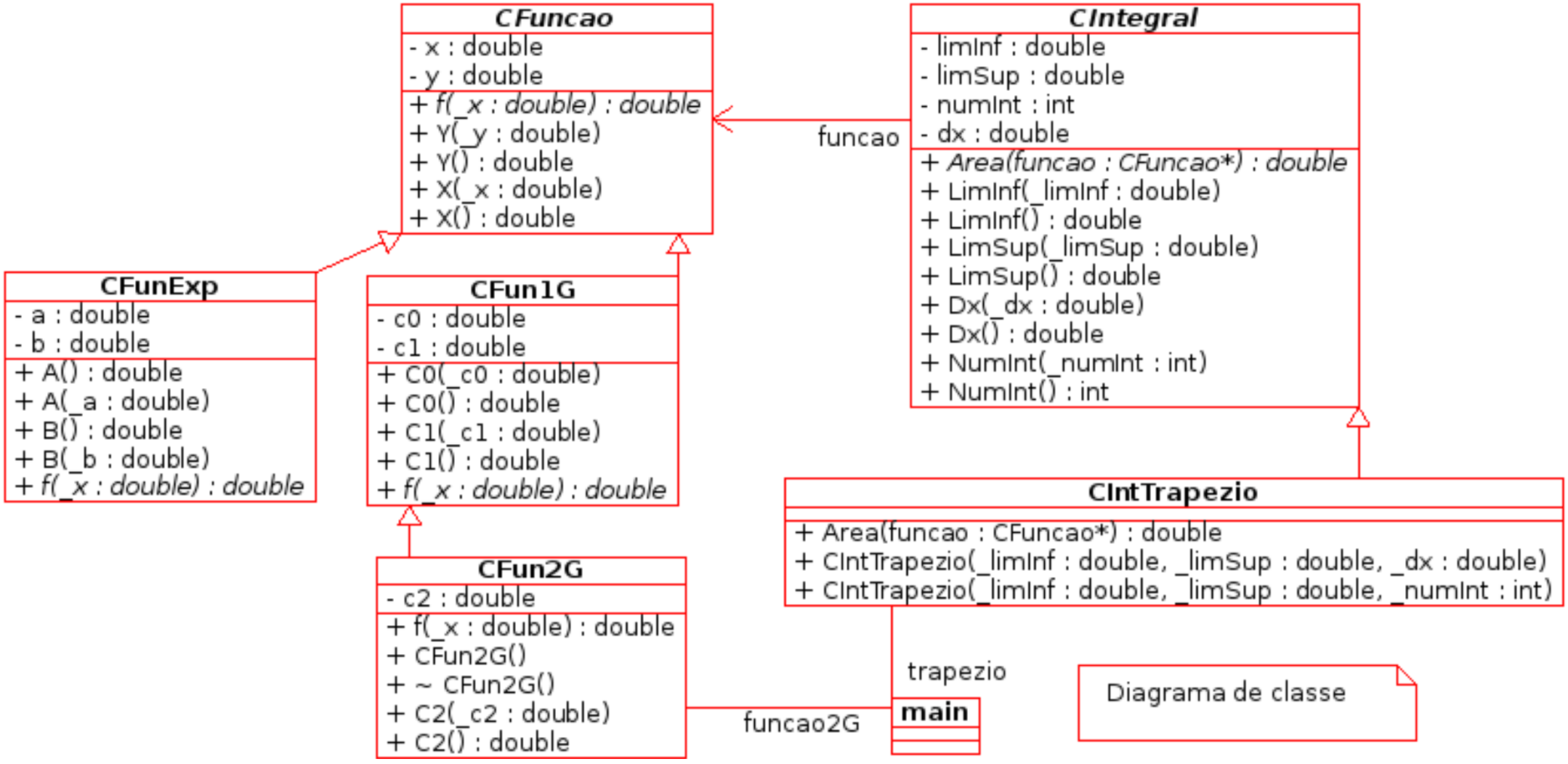


Figura 11.1: Diagrama de classe - versão 0.8

11.3 Códigos

Listing 11.1: Arquivo de declaração da classe CFuncao - CFuncao.h.

```
1 /**************************************************************************
2 CFuncao.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 // Define variável de pré-processamento que impede dupla compilação deste arquivo.
8 #ifndef CFuncao_H
9 #define CFuncao_H
10
11 /// Representa uma função genérica.
12 class CFuncao
13 {
14  protected:
15   double y{0.0}; ///< Representa variável dependente.
16   double x{0.0}; ///< Representa variável independente.
17
18  public:
19   /// Construtor
20   CFuncao()       = default;
21   /// Destrutor
22   virtual ~CFuncao()      = default;
23
24   /// Calcula valor da função na posição x, isto é, y = f(x);.
```

```
25   virtual  double f ( double _x  ) = 0 ; // virtual pura (classe abstrata).
26
27   /// Seta valor de y.
28   void Y ( double _y )          { y = _y; };
29
30   /// Retorna valor de y.
31   double Y ( )                  { return y; };
32
33   /// Seta valor de x.
34   /// @param _x novo valor de x.
35   void X ( double _x )          { x = _x; };
36
37   /// Retorna valor de x.
38   /// @return retorna valor de x.
39   double X ( )                  { return x; };
40 };
41 #endif
```

Listing 11.2: Arquivo de definição da classe CFuncao - CFuncao.cpp.

```
1 #include "CFuncao.h"
```

Listing 11.3: Arquivo de declaração da classe CFun1G - CFun1G.h.

```
1 /**********************************************************************
2 CFun1G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFun1G_H
8 #define CFun1G_H
9 #include "CFuncao.h"
10
11 /// Representa funcao do primeiro grau, y = c0 + c1 * x;.
12 class CFun1G : public CFuncao
13 {
14  protected:
15   double c0{0.0}; ///< Representa coeficiente linear
16   double c1{0.0}; ///< Representa coeficiente angular
17
18  public:
19   /// Construtor
20   CFun1G()        = default;
21
22   /// Destrutor
23   virtual ~CFun1G()       = default;
24
25   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
26   virtual double f (double _x ) override ; // declaração
27
28   /// Seta valor de c0.
```

```
29   /// @param _c0 novo valor de c0.
30   void C0 ( double _c0 )          {  c0 = _c0; };
31
32   /// Retorna valor de c0.
33   /// @return retorna valor de c0.
34   double C0 ( )                   {  return c0; };
35
36   /// Seta valor de c1.
37   /// @param _c1 novo valor de c1
38   void C1 ( double _c1 )          {  c1 = _c1; };
39
40   /// Retorna valor de c1
41   /// @return retorna valor de c1
42   double C1 ( )                   {  return c1; };
43 };
44 #endif
```

Listing 11.4: Arquivo de definição da classe CFun1G - CFun1G.cpp.

```
1 #include "CFun1G.h"
2
3 double CFun1G::f (double _x )   {
4   x = _x;
5   return y = c0 + c1 * x ;
6 }
```

Listing 11.5: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1 /**********************************************************************
2 CFun2G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFun2G_H
8 #define CFun2G_H
9 #include "CFun1G.h"
10
11 /// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
12 class CFun2G : public CFun1G
13 {
14  protected:
15   double c2{0.0}; ///< Representa coeficiente quadrático
16
17  public:
18   /// Construtor
19   CFun2G()        = default;
20
21   /// Destrutor (se é uma hierarquia o destrutor deve ser virtual)
22   virtual ~CFun2G()       = default;
23
24   /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
25   // virtual indica que muda nas classes herdeiras.
26   // override indica que sobrescreve versão da classe base (i.e.de CFun1G)
27   virtual double f (double _x ) override;
28
```

```
29   /// Seta valor de c2
30   void C2 ( double _c2 )         { c2 = _c2; };
31
32   /// Retorna valor de c2
33   double C2 ( )          { return c2; };
34 };
35 #endif
```

Listing 11.6: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1 #include "CFun2G.h"
2
3 double CFun2G::f (double _x  )  {
4   x = _x;
5   return  y = c0 + c1 * x + c2 * x * x;
6 }
```

Listing 11.7: Arquivo de declaração da classe CFunExp - CFunExp.h.

```
/**************************************************************************
CFunExp.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CFunExp_H
#define CFunExp_H
#include "CFuncao.h"

/// Representa funcao exponencial,  y = a*exp(b*x).
class CFunExp : public CFuncao
{
 protected:
  double a{0.0};         ///< Coeficiente a da funcao exponencial y = a*exp(b*x)
  double b{0.0};         ///< Coeficiente b da funcao exponencial y = a*exp(b*x)

 public:
  /// Construtor default (sem parâmetros)
  CFunExp()       = default;
  /// Construtor sobrecarregado (com parâmetros)
  CFunExp(double _a, double _b) : a(_a), b(_b) {};

  /// Destrutor
  virtual ~CFunExp()      = default;

  /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
  virtual double f (double _x ) override;
```

```
29
30   /// Seta valor de a
31   void A( double _a )    { a = _a; };
32
33   /// Retorna valor de a
34   double A( )            { return a; };
35
36   /// Seta valor de b
37   void B( double _b )    { b = _b; }
38
39   /// Retorna valor de b
40   double B( )            { return b; };
41
42 };
43 #endif // CFUNEXP_H
```

Listing 11.8: Arquivo de definição da classe CFunExp - CFunExp.cpp.

```
1 #include <cmath>          // math.h no C
2 #include "CFunExp.h"
3 using namespace std;
4
5
6 double CFunExp::f ( double _x  ) {
7  x = _x;
8  return y = a * exp( b * x );
9 }
```

Listing 11.9: Arquivo de declaração da classe CIntegral - CIntegral.h.

```
1 /**************************************************************************
2 CIntegral.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CIntegral_H
8 #define CIntegral_H
9 #include "CFuncao.h"
10
11 /// Representa método numérico para cálculo da área de funções.
12 class CIntegral
13 {
14  protected:
15   double limInf{0.0};    ///< Limite Inferior do intervalo de integração
16   double limSup{1.0};    ///< Limite Superior do intervalo de integração
17   double dx{0.01};       ///< Intervalo dx
18   int    numInt{101};    ///< Numero intervalos
19   double area{0.0};      ///< Valor da área calculada
20
21  public:
22   /// Construtor
23   CIntegral()             = default;
24
25   /// Destrutor
26   virtual ~CIntegral()    = default;
27
28   /// Calcula área da função no intervalo limInf->limSup considerando numInt.
```

```
29  virtual double Area ( CFuncao* funcao = nullptr ) = 0 ; // classe abstrata (interface)
30
31  /// Seta valor de limInf
32  /// @param _limInf novo valor de limInf
33  void LimInf ( double _limInf  )        { limInf = _limInf ; };
34
35  /// Retorna valor de limInf
36  /// @return retorna valor de limInf
37  double LimInf ( )                      { return limInf; };
38
39  /// Seta valor de limSup
40  /// @param _limSup Seta novo valor de  limSup
41  void LimSup ( double _limSup )         { limSup = _limSup; };
42
43  /// Retorna valor de limSup
44  /// @return retorna valor de limSup
45  double LimSup ( )                      {return limSup; } ;
46
47  /// Seta valor de dx.
48  /// Note que após redefinir dx, recalcula numero intervalos.
49  /// O valor de dx multiplicado por numInt deve ser igual ao intervalo.
50  /// @param _dx o novo valor de  dx
51  void Dx ( double _dx )                 { dx = _dx; numInt = (limSup - limInf) / dx;};
52
53  /// Retorna valor de dx
54  double Dx ( )                          { return dx;};
55
56  /// Seta valor de numInt.
57  /// Observe que após redefinir numInt, recalcula dx.
```

```
58   void NumInt ( int _numInt )            { numInt = _numInt;
59                                            dx = (limSup - limInf) / (double)numInt;
60   };
61
62    /// Retorna valor de numInt
63   int NumInt ( )                         { return numInt; } ;
64};
65#endif
```

Listing 11.10: Arquivo de definição da classe CIntegral - CIntegral.cpp.

```
1#include "CIntegral.h"
```

Listing 11.11: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2                         CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CIntTrapezio_H
8 #define CIntTrapezio_H
9 #include "CIntegral.h"
10
11 /**
12   * Representa método numérico trapésio para cálculo da área de funções.
13   * Note que CIntTrapezio é herdeira de CIntegral
14   * @class CIntTrapezio
15   */
16 class CIntTrapezio : public CIntegral
17 {
18 public:
19   /// Construtor
20   CIntTrapezio()                  = default;
21   /// Destrutor
22   virtual ~CIntTrapezio()         = default;
23
24   /**
25    * Calcula área da função no intervalo limInf->limSup considerando numInt.
26    * @return double
27    * @param  funcao
28    */
```

```
29  virtual double Area ( CFuncao* funcao = nullptr ) override;
30};
31#endif // CIntTrapezio_H
```

Listing 11.12: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1#include "CIntTrapezio.h"
2
3double CIntTrapezio::Area (CFuncao* funcao )  {
4  area = funcao->f( limInf ) * 0.5;
5  area += funcao->f( limSup) * 0.5; // area = area + funcao->f(limSup);
6
7  double x = limInf;
8  for( int i = 1; i < numInt ; i++ )
9          {
10 //       area += funcao->f( limInf + i * dx );
11           x += dx;
12           area += funcao->f( x );
13         }
14 area *= dx ;
15 return area;
16}
```

Listing 11.13: Programa para cálculo da integral do trapésio: Usa hierarquia classes CFuncao e CIntegral.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Programa em C++ com classes.
 * - Foi criada hierarquia de funções: classes CFuncao, CFun1G, CFun2G, CFunExp.
 * - Foi criada hierarquia de métodos de integral: classes CIntegral, CIntTrapezio.
 * - Usa métodos virtuais.
 * - Usa palavras chaves do C++11 como override (veja http://www.cppreference.com).
 *
 * Note que aqui os códigos já estão suficientemente genéricos para que possam ser reaproveitados
 * por outros programas, o que indica que o ideal seria criar uma biblioteca para as duas hierarquias.
 * Note que a vantagem do uso dos construtores esta em inicializar os atributos
 * com os valores que nos interessam, sem a necessidade de chamar cada um
 * dos métodos Set para setar os atributos.
 *
 * Para compilar:
 * make
 * ou
 * g++ -std=c++11 CFuncao.cpp CFun1G.cpp  CFun2G.cpp CFunExp.cpp  CIntegral.cpp  CIntTrapezio.cpp  main.cpp  -o
    integral
*/
#include <iostream>
#include "CFun2G.h"             // Arquivo com definição da classe CFun2G
#include "CFunExp.h"            // Arquivo com definição da classe CFunExp
#include "CIntTrapezio.h"       // Arquivo com definição da classe CIntTrapezio
```

```
28 using namespace std;
29
30 /// Função principal do programa
31 int main(int argc, char **argv) {
32   CFun2G funcao2G;
33
34   cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
35   cout << "Entre␣com␣c0␣:␣";
36   double _c;
37   cin >> _c;
38   funcao2G.C0( _c );
39
40   cout << "Entre␣com␣c1␣:␣";
41   cin >> _c;
42   funcao2G.C1( _c );
43
44   cout << "Entre␣com␣c2␣:␣";
45   cin >> _c;
46   funcao2G.C2( _c );
47
48   cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
49   CIntTrapezio trapezio;
50   cout << "Entre␣com␣Limite␣Inferior␣:␣";
51   double _lim;
52   cin >> _lim;
53   trapezio.LimInf( _lim );
54   cout << "Entre␣com␣Limite␣Superior␣:␣";
55   cin >> _lim;
56   trapezio.LimSup( _lim );
```

```
57  cout << "Entre␣com␣Numero␣Intervalos␣:␣";
58  int _numInt;
59  cin >> _numInt;
60  trapezio.NumInt(_numInt);
61  cout << "\nArea␣funcao2G␣=␣" << trapezio.Area( &funcao2G ) <<  "\n";
62
63  CFunExp fexp{1.0,2.0}; // a=1.0, b-2.0
64  cout << "\nArea␣fexp␣=␣" << trapezio.Area( &fexp ) <<  "\n";
65  cin.get();
66  return 0;
67}
```

```
[bueno@localhost src]$ ./integracaonumerica
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 5
Entre com c1 : -6
Entre com c2 : 12
Entre com dados do metodo integracao:
Entre com Limite Inferior : -1
Entre com Limite Superior : 1
Entre com Numero Intervalos : 1000

Area = 18
```

Capítulo 12

v0.9 - Programação Orientada a Objeto - UML - Sobrecarga Operador e CSimulador

12.1 Características

- Acrescenta sobrecarga operadores.
  - Acrescenta sobrecarga `operator()`.
  - Acrescenta sobrecarga operadores `operator>>` e `operator<<`.
- Acrescenta classe `CSimulador` com conteúdo de `main`().

12.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 12.1 o diagrama de classes e a Figura 10.2 o diaqrama de sequência.

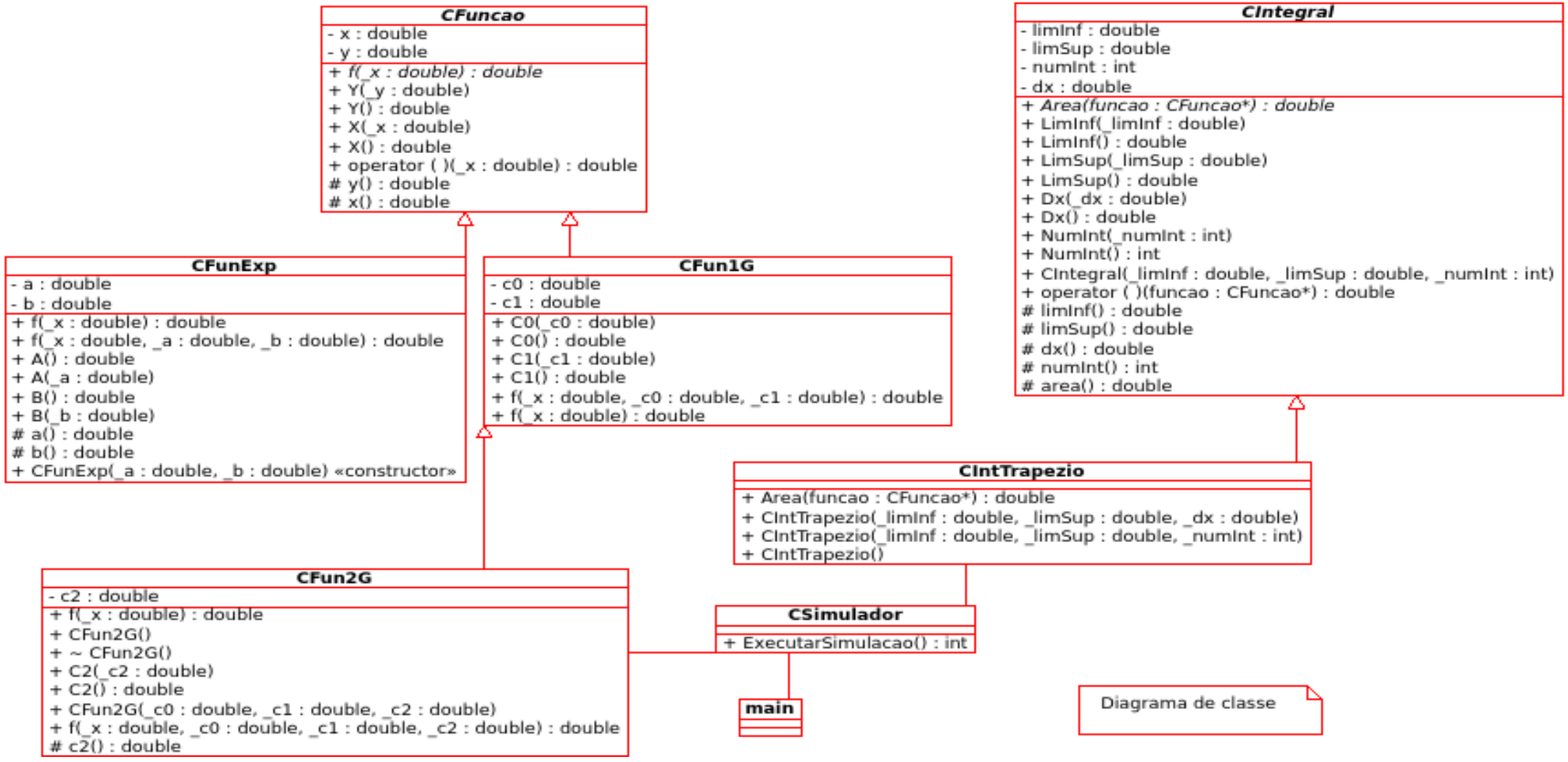


Figura 12.1: Diagrama de classe - versão 0.9

12.3 Códigos

Listing 12.1: Arquivo de declaração da classe CFuncao - CFuncao.h.

```
1 /**************************************************************************
2 CFuncao.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 // Define variável de pré-processamento que impede dupla compilação deste arquivo.
8 #ifndef CFuncao_H
9 #define CFuncao_H
10
11 /// Representa uma função genérica.
12 class CFuncao
13 {
14  protected:
15   double y{0.0}; ///< Representa variável dependente.
16   double x{0.0}; ///< Representa variável independente.
17
18  public:
19   /// Construtor default
20   CFuncao() = default;
21
22   /// Destrutor default
23   ~CFuncao() = default;
24
```

```
25   /// Calcula valor da função na posição x, isto é, y = f(x);
26   virtual  double f ( double _x  ) = 0;
27
28   /// Calcula valor da função na posição x, isto é, y = f(x);.
29   /// A sobrecarga do operador (), possibilita uso do objeto como objeto função,
30   /// objeto se comporta como uma função.
31   double operator()( double _x ) { x = _x; return f(x); }
32
33   /// Seta valor de y.
34   /// @param _y novo valor de y
35   void Y ( double _y )          { y = _y; };
36
37   /// Obtém valor de y.
38   /// Retorna valor de y.
39   double Y ( )                  { return y; };
40
41   /// Seta valor de x.
42   /// @param _x novo valor de x.
43   void X ( double _x )          { x = _x; };
44
45   /// Retorna valor de x.
46   /// @return retorna valor de x.
47   double X ( )                  { return x; };
48 };
49 #endif
50
51 /*
52 int main() {
53   CFuncao obj_funcao;
```

```
54  double r = 3.2;
55  // Objeto função executa a funcao f
56  cout << "resultado da funcao é " << obj_funcao.f(r) ;
57
58  // Objeto função executa o operador ()
59  // Sempre que sobrecarrego um objeto com operador () ele se comporta como uma funcao.
60  cout << "resultado da funcao é " << obj_funcao(r) ;
61
62
63  NomeObjeto.NomeFuncao(r); // tem de ser publicos
64  NomeObjeto.nomeAtributo;
65
66  CFun1G f1;
67  CFun2G f2;
68  CFun3G f3;
69  f1(123);
70
71  return 0;
72}*/
```

Listing 12.2: Arquivo de definição da classe CFuncao - CFuncao.cpp.

```
1#include "CFuncao.h"
```

Listing 12.3: Arquivo de declaração da classe CFun1G - CFun1G.h.

```
1 /**********************************************************************
2 CFun1G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFun1G_H
8 #define CFun1G_H
9 #include "CFuncao.h"
10
11 /// Representa funcao do primeiro grau, y = c0 + c1 * x;.
12 class CFun1G : public CFuncao
13 {
14  protected:
15   double c0{0.0}; ///< Representa coeficiente linear
16   double c1{0.0}; ///< Representa coeficiente angular
17
18  public:
19   /// Construtor default
20   CFun1G()        = default;
21
22   /// Construtor sobrecarregado
23   /// @param _c0 valor de c0 e _c1 valor de c1.
24   CFun1G(double _c0, double _c1) : c0(_c0),c1(_c1)      {};
25
26   /// Destrutor default
27   ~CFun1G()       = default;
28
```

```
29   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
30   virtual double f (double _x );
31
32   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
33   virtual double f (double _x , double _c0, double _c1)  {
34     c0 = _c0; c1 = _c1;
35     return f(_x);
36   }
37
38   /// Seta valor de c0.
39   /// @param _c0 novo valor de c0.
40   void C0 ( double _c0 )          {  c0 = _c0; };
41
42   /// Retorna valor de c0.
43   /// @return retorna valor de c0.
44   double C0 ( )                   {  return c0; };
45
46   /// Seta valor de c1.
47   /// @param _c1 novo valor de c1
48   void C1 ( double _c1 )          {  c1 = _c1; };
49
50   /// Retorna valor de c1
51   /// @return retorna valor de c1
52   double C1 ( )                   {  return c1; };
53 };
54 #endif
```

Listing 12.4: Arquivo de definição da classe CFun1G - CFun1G.cpp.

```
1#include "CFun1G.h"
2
3double CFun1G::f (double _x ) {
4 x = _x;
5 return y = c0 + c1 * x;
6}
```

Listing 12.5: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1 /**********************************************************************
2 CFun2G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFun2G_H
8 #define CFun2G_H
9 #include "CFun1G.h"
10
11 /// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
12 class CFun2G : public CFun1G
13 {
14  protected:
15   double c2{0.0}; ///< Representa coeficiente quadrático
16
17  public:
18   /// Construtor default
19   CFun2G()        = default;
20
21   /// Construtor sobrecarregado
22   CFun2G(double _c0, double _c1, double _c2):CFun1G(_c0,_c1),c2(_c2)    {};
23
24   /// Destrutor default
25   ~CFun2G()       = default;
26
27   /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
28   virtual double f (double _x ) override ;
```

```
29
30   /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
31   virtual double f (double _x , double _c0, double _c1, double _c2 )   {
32     c0 = _c0; c1 = _c1; c2 = _c2;
33     return f(_x);
34   }
35
36   /// Seta valor de c2
37   void C2 ( double _c2 )          { c2 = _c2; };
38
39   /// Retorna valor de c2
40   double C2 ( )                   { return c2; };
41 };
42 #endif
```

Listing 12.6: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1 #include "CFun2G.h"
2
3 double CFun2G::f ( double _x ) {
4   x = _x;
5   return y = c0 + c1*x + c2*x*x;
6 }
```

Listing 12.7: Arquivo de declaração da classe CFunExp - CFunExp.h.

```
/**************************************************************************
CFunExp.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CFunExp_H
#define CFunExp_H
#include "CFuncao.h"

/// Representa funcao exponencial,  y = a*exp(b*x).
class CFunExp : public CFuncao
{
 protected:
  double a{0.0};         ///< Coeficiente a da funcao exponencial y = a*exp(b*x)
  double b{0.0};         ///< Coeficiente b da funcao exponencial y = a*exp(b*x)

 public:
  /// Construtor default (sem parâmetros)
  CFunExp()       = default;
  /// Construtor sobrecarregado (com parâmetros)
  CFunExp(double _a, double _b) : a(_a), b(_b) {};

  /// Destrutor default
  ~CFunExp()      = default;

  /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
 virtual  double f ( double _x  );
```

```
29
30   /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
31  virtual  double f ( double _x , double _a, double _b )  {
32       a = _a; b = _b;
33       return f(_x);
34   }
35
36   /// Seta valor de a
37   /// @param _a novo valor de a
38   void A ( double _a )  { a = _a; };
39
40   /// Retorna valor de a
41   double A( )           { return a; };
42
43   /// Seta valor de b
44   void B ( double _b )  { b = _b; }
45
46   /// Retorna valor de b
47   double B ( )          { return b; };
48 };
49 #endif
```

Listing 12.8: Arquivo de definição da classe CFunExp - CFunExp.cpp.

```
1 #include <cmath> /// math.h no C
2 #include "CFunExp.h"
3
4 double CFunExp::f ( double _x ) {
5   x = _x;
```

```
6   return y = a * std::exp( b * x );
7 }
```

Listing 12.9: Arquivo de declaração da classe CIntegral - CIntegral.h.

```
/**************************************************************************
CIntegral.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CIntegral_H
#define CIntegral_H
#include "CFuncao.h"

/// Representa método numérico para cálculo área funções.
class CIntegral
{
 protected:
  double limInf{0.0};    ///< Limite Inferior do intervalo de integração
  double limSup{1.0};    ///< Limite Superior do intervalo de integração
  double dx{0.01};       ///< Intervalo dx
  int    numInt{100};    ///< Numero intervalos
  double area{0.0};      ///< Valor da área calculada

 public:
  /// Construtor default (sem parâmetros)
  CIntegral() = default;

  /// Construtor sobrecarregado
  CIntegral( double _limInf, double _limSup, int _numInt)
  : limInf(_limInf), limSup(_limSup), numInt(_numInt), area(0.0) {
                                      dx =  (limSup - limInf) / double(numInt);
```

```
29                                      }
30
31  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
32  virtual double Area ( CFuncao* funcao ) = 0;
33
34  /// Calcula área da função
35  double operator()( CFuncao* funcao  ) { return Area(funcao); }
36
37  /// Seta valor de limInf
38  /// @param _limInf novo valor de limInf
39  void LimInf ( double _limInf  )       { limInf = _limInf ; }
40
41  /// Retorna valor de limInf
42  /// @return retorna valor de limInf
43  double LimInf ( )                     { return limInf; }
44
45  /// Seta valor de limSup
46  /// @param _limSup Seta novo valor de  limSup
47  void LimSup ( double _limSup )        { limSup = _limSup; }
48
49  /// Retorna valor de limSup
50  /// @return retorna valor de limSup
51  double LimSup ( )                     {return limSup; }
52
53  /// Seta valor de dx
54  /// @param _dx o novo valor de  dx
55  void Dx ( double _dx )                { dx = _dx; numInt = (limSup - limInf) / dx;}
56
57  /// Retorna valor de dx
```

```
58   /// @return retorna valor de x
59   double Dx ( )                          { return dx;}
60
61   /// Seta valor de numInt
62   void NumInt ( int _numInt )            { numInt = _numInt;
63                                           dx = (limSup - limInf) / numInt;
64                                          }
65
66   /// Retorna valor de numInt
67   int NumInt ( )                         { return numInt; }
68 };
69 #endif
```

Listing 12.10: Arquivo de definição da classe CIntegral - CIntegral.cpp.

```
1 #include "CIntegral.h"
```

Listing 12.11: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7
8 #ifndef CIntTrapezio_H
9 #define CIntTrapezio_H
10 #include "CIntegral.h"
11
12 /// Representa método numérico trapésio para cálculo área funções.
13 class CIntTrapezio : public CIntegral
14 {
15  public:
16   /// Construtor default
17   CIntTrapezio() = default;
18
19   /// Construtor sobrecarregado.
20   // Note passagem de parâmetros para classe base
21   CIntTrapezio( double _limInf, double _limSup, int _numInt)
22   :CIntegral(_limInf,_limSup,_numInt) {}
23
24   /// Desstrutor default
25   virtual ~CIntTrapezio() = default;
26
27   /// Calcula área da função no intervalo limInf->limSup considerando numInt.
28   virtual  double Area ( CFuncao* funcao ) override;
```

```
29};
30#endif
```

Listing 12.12: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1#include "CIntTrapezio.h"
2
3double CIntTrapezio::Area (CFuncao *funcao ) {
4 area = funcao->f(limInf)*0.5;
5 double x = limInf;
6 for( int i = 1; i < numInt ; i++ ) {
7 //        area += funcao->f( limInf + i * dx );
8           x += dx;
9           area += funcao->f( x );
10        }
11 area += funcao->f(limSup)*0.5;
12 area *= dx ;
13 return area;
14}
```

Listing 12.13: Arquivo de declaração da classe CSimulador - CSimulador.h.

```
1 /**********************************************************************
2 CSimulador.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CSimulador_H
8 #define CSimulador_H
9
10 class CSimulador
11 {
12  public:
13   /// Construtor da classe
14   CSimulador()  = default;
15
16   /// Destrutor da classe
17   virtual ~CSimulador() = default;
18
19   /// Executa a simulação.
20   int ExecutarSimulacao();
21 };
22 #endif
```

Listing 12.14: Arquivo de definição da classe CSimulador - CSimulador.cpp.

```
1 /**********************************************************************
2 CSimulador.cpp - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
```

```
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 ***************************************************************************/
7 #include <iostream>
8 #include "CSimulador.h"
9 #include "CFun1G.h"
10 #include "CFun2G.h"
11 #include "CFunExp.h"
12 #include "CIntTrapezio.h"
13 using namespace std;
14
15 /// Executa a simulação.
16 int CSimulador::ExecutarSimulacao() {
17   // Entrada dados
18   cout << "Entre com dados da funcao 1G  y = c0 + c1*x\n";
19   cout << "Entre com dados da funcao 2G  y = c0 + c1*x + c2*x*x\n";
20   cout << "Entre com dados da funcao exp y = c0 * std::exp( c1 * x ) :\n";
21   cout << "Entre com c0 : ";
22   double c0;
23   cin >> c0; cin.get();
24   cout << "Entre com c1 : ";
25   double c1;
26   cin >> c1; cin.get();
27   cout << "Entre com c2 : ";
28   double c2;
29   cin >> c2; cin.get();
30
31   // Criação objeto função usando construtor sobrecarregado
32   CFun2G funcao2G( c0, c1, c2 );
```

```
33
34  cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
35  cout << "Entre␣com␣Limite␣Inferior␣:␣";
36  double limInf;
37  cin >> limInf; cin.get();
38  cout << "Entre␣com␣Limite␣Superior␣:␣";
39  double limSup;
40  cin >> limSup; cin.get();
41  cout << "Entre␣com␣Numero␣Intervalos␣:␣";
42  int numInt;
43  cin >> numInt; cin.get();
44
45  // Criação objeto integral usando construtor sobrecarregado
46  CIntTrapezio trapezio( limInf , limSup , numInt );
47
48  // Cálculo da área e saída de resultados
49  cout << "\nArea␣funcao2G␣=␣" << trapezio.Area(&funcao2G) <<  "\n";
50
51  CFunExp fexp{c0,c1}; // a=1.0, b-2.0
52  cout << "\nArea␣fexp␣=␣" << trapezio.Area( &fexp ) <<  "\n";
53
54  CFun1G funcao1G{c0,c1}; // a=1.0, b-2.0
55  cout << "\nArea␣funcao1G␣=␣" << trapezio.Area( &funcao1G ) <<  "\n";
56  // Uso operador sobrecarregado, note que o objeto trapezio se comporta como se fosse uma função.
57  // é por isso que classes com operator() sobrecarregados são chamados de "objeto função".
58  cout << "\nArea␣funcao1G␣(usa␣operador␣sobrecarregado)␣=␣" << trapezio( &funcao1G ) <<  "\n";
59  cin.get();
60  return 0;
61 }
```

Listing 12.15: Programa para cálculo da integral do trapésio: Usa hierarquia classes CFuncao e CIntegral.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Acrescenta sobrecarga operadores.
 * - Acrescenta classe CSimulador com conteúdo de main.
 *
 * Para compilar:
 * g++ -std=c++11 *.cpp -o integral
*/
#include <iostream>
#include "CSimulador.h"

/// Função principal do programa
int main(int argc, char **argv) {
  CSimulador simulador;
  simulador.ExecutarSimulacao();
  std::cin.get();
  return 0;
}
```

```
[bueno@localhost src]$ ./integral
Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :
Entre com c0 : 10
Entre com c1 : -20
Entre com c2 : 15
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com Limite Superior : 1
Entre com Numero Intervalos : 2000

Area = 5
```

Capítulo 13

v1.0 - Programação Orientada a Objeto - UML - Polimorfismo

13.1 Características

- Acrescenta polimorfismo, permitindo a definição de qual função e qual método de integração serão utilizados em tempo de execução. Ou seja, o usuário irá selecionar a função e o método de integração.

- Uso de `friend std::istream& operator>>(std::istream& in, CFuncao& funcao);`

13.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 13.1 o diagrama de classes e a Figura 10.2 o diaqrama de sequência.

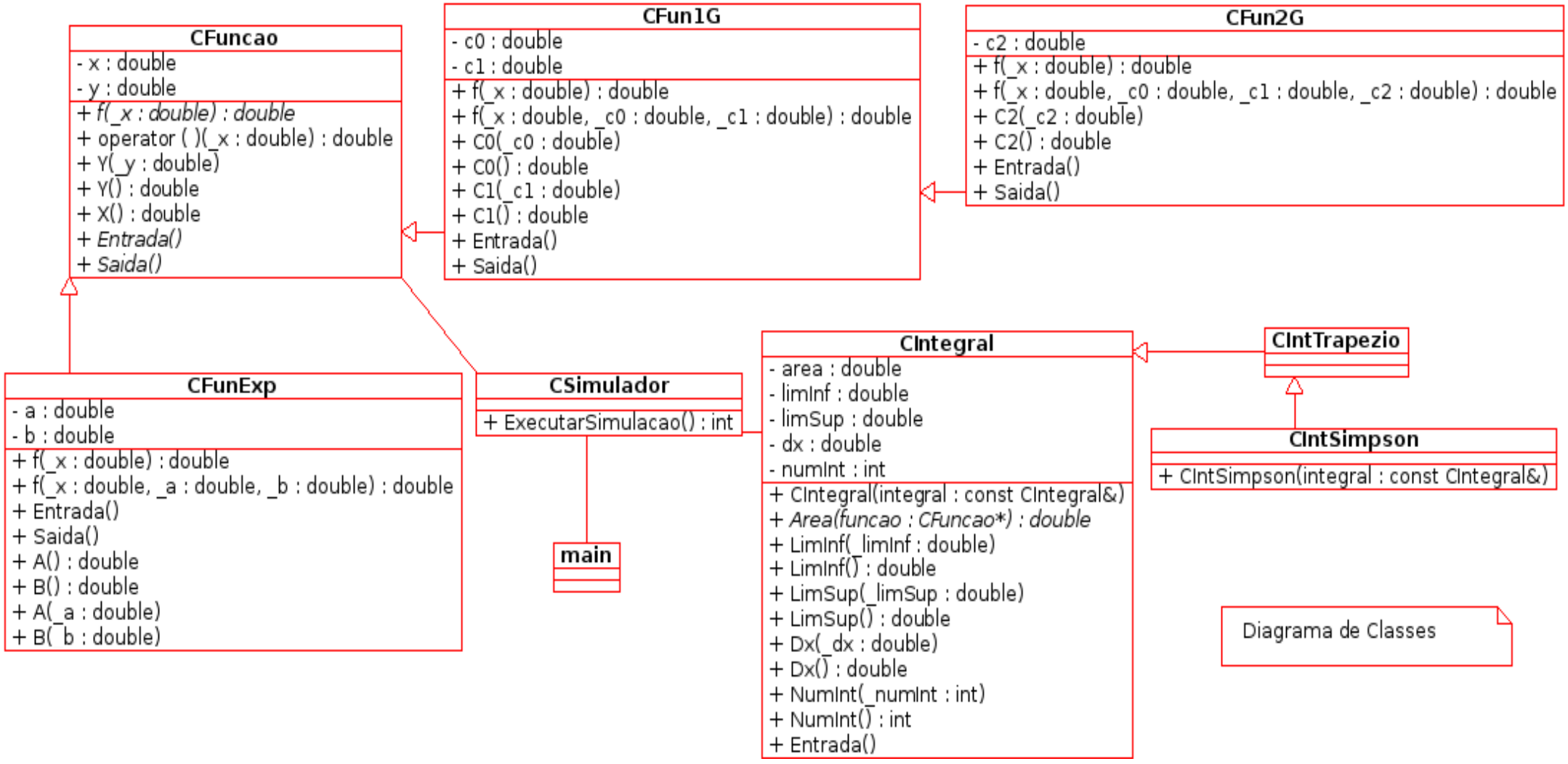


Figura 13.1: Diagrama de classe - versão 1.0

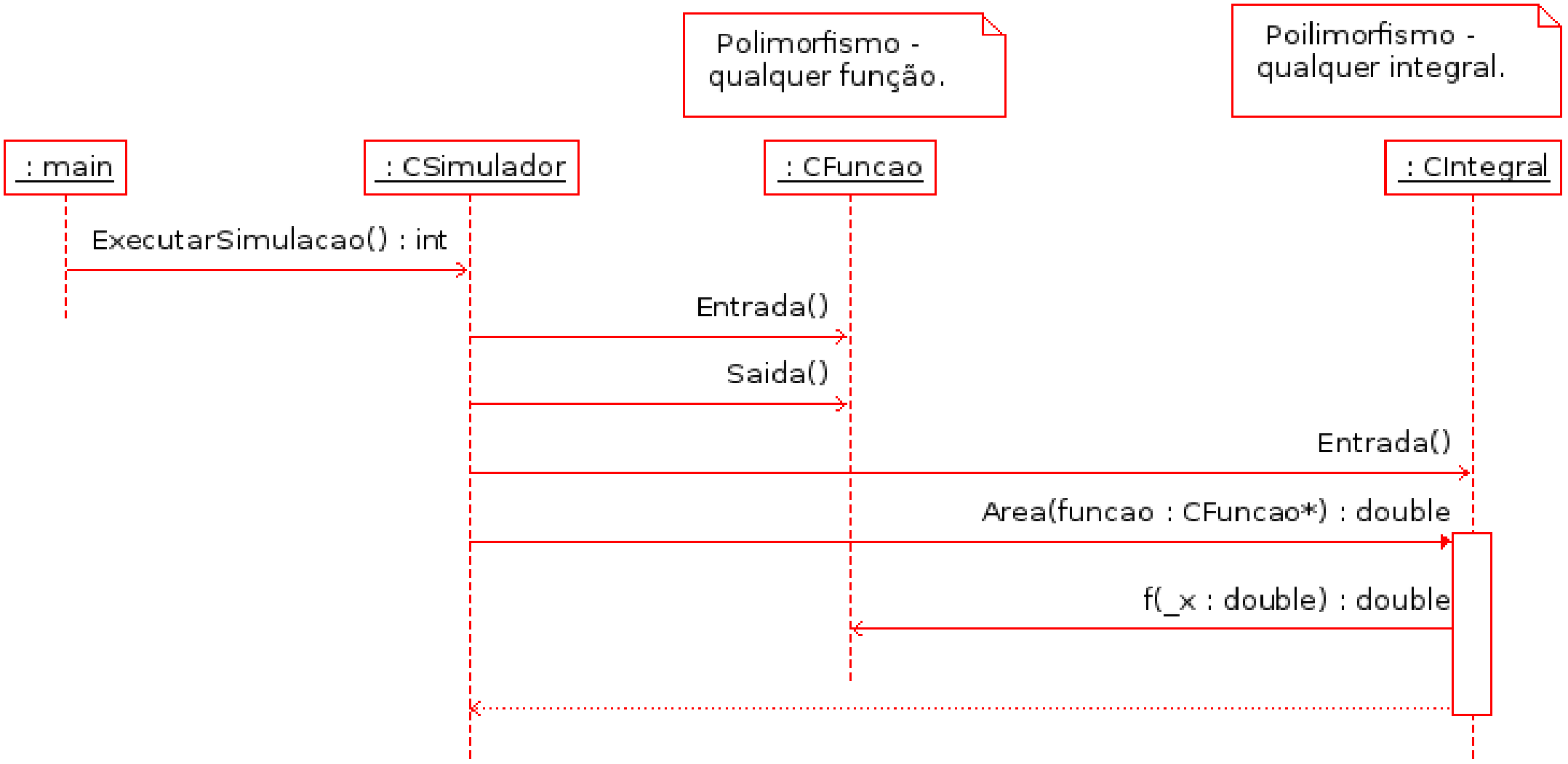


Figura 13.2: Diagrama de sequência - v1.0

13.3 Códigos

Listing 13.1: Arquivo de declaração da classe CFuncao - CFuncao.h.

```
1 /**************************************************************************
2 CFuncao.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 // Define variável de pré-processamento que impede dupla compilação deste arquivo.
8 #ifndef CFuncao_H
9 #define CFuncao_H
10 #include<iostream>
11
12 /// Representa uma função genérica.
13 class CFuncao
14 {
15  protected:
16   double y{0.0}; ///< Representa variável dependente.
17   double x{0.0}; ///< Representa variável independente.
18
19  public:
20   /// Construtor
21   CFuncao()      = default;
22   /// Destrutor
23   virtual ~CFuncao()     = default;
24
```

```
25  /// Calcula valor da função na posição x, isto é, y = f(x);.
26  virtual  double f ( double _x ) = 0;
27
28  /// Calcula valor da função na posição x, isto é, y = f(x);.
29  /// A sobrecarga do operador (), possibilita uso objeto como objeto função,
30  /// objeto se comporta como uma função.
31  double operator()( double _x )          { x = _x; return f(x); }
32
33   /// Seta valor de y.
34   /// @param _y novo valor de y
35  void Y ( double _y )                    { y = _y; };
36
37   /// Obtém valor de y.
38   /// Retorna valor de y.
39  double Y ( ) const                      { return y; };
40
41  // Seta valor de x.
42  // @param _x novo valor de x.
43  //void X ( double _x )                  { x = _x; };
44
45  /// Retorna valor de x.
46  /// @return retorna valor de x.
47  double X ( ) const                      { return x; };
48
49   /// Entrada dos valores dos atributos.
50  virtual void Entrada() = 0;
51
52  /// Saída de atributos.
53  virtual void Saida() = 0;
```

```
54
55   /// Sobrecarga operador operator>>
56   friend std::istream& operator>>(std::istream& in,  CFuncao& funcao);
57
58   /// Sobrecarga operador operator<<
59   friend std::ostream& operator<<(std::ostream& out, CFuncao& funcao);
60 };
61 #endif
```

Listing 13.2: Arquivo de definição da classe CFuncao - CFuncao.cpp.

```
1 #include "CFuncao.h"
2
3 std::istream& operator>>(std::istream& in,  CFuncao& funcao) {
4   in >> funcao.x; in.get();
5   in >> funcao.y;
6   return in;
7 }
8
9 std::ostream& operator<<(std::ostream& out, CFuncao& funcao) {
10    out <<  funcao.x << "␣" << funcao.y;
11    return out;
12 }
```

Listing 13.3: Arquivo de declaração da classe CFun1G - CFun1G.h.

```
1 /**********************************************************************
2 CFun1G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFun1G_H
8 #define CFun1G_H
9 #include "CFuncao.h"
10
11 /// Representa funcao do primeiro grau, y = c0 + c1 * x;.
12 class CFun1G : public CFuncao
13 {
14  protected:
15   double c0{0.0}; ///< Representa coeficiente linear
16   double c1{0.0}; ///< Representa coeficiente angular
17
18  public:
19   /// Construtor default (sem parâmetros)
20   CFun1G()       = default;
21
22   /// Construtor sobrecarregado (com parâmetros)
23   CFun1G(double _c0, double _c1): c0(_c0), c1(_c1) {};
24
25   /// Destrutor
26   virtual ~CFun1G()      = default;
27
28   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
```

```
29  virtual double f (double _x ) override;
30
31  /// Calcula valor da função na posição x, y = c0 + c1 * x;.
32  virtual double f (double _x , double _c0, double _c1)  {
33    c0 = _c0; c1 = _c1;
34    return f(_x);
35  }
36
37  /// Seta valor de c0.
38  /// @param _c0 novo valor de c0.
39  void C0 ( double _c0 )        {  c0 = _c0; };
40
41  /// Retorna valor de c0.
42  /// @return retorna valor de c0.
43  double C0 ( ) const   {  return c0; };
44
45  /// Seta valor de c1.
46  /// @param _c1 novo valor de c1
47  void C1 ( double _c1 )        {  c1 = _c1; };
48
49  /// Retorna valor de c1
50  /// @return retorna valor de c1
51  double C1 ( ) const   {  return c1; };
52
53  /// Entrada dos valores dos atributos.
54  virtual void Entrada();
55
56  /// Saída de atributos.
57  virtual void Saida();
```

```
58
59    /// Sobrecarga operador operator>>
60   friend std::istream& operator>>(std::istream& in,  CFun1G& funcao);
61
62   /// Sobrecarga operador operator<<
63   friend std::ostream& operator<<(std::ostream& out, CFun1G& funcao);
64 };
65 #endif
```

Listing 13.4: Arquivo de definição da classe CFun1G - CFun1G.cpp.

```
1 #include <iostream>
2 using namespace std;
3
4 #include "CFun1G.h"
5
6 void CFun1G::Entrada() {
7   cout << "Entre␣com␣dados␣da␣funcao␣1G␣y␣=␣c0␣+␣c1*x␣␣:\n";
8   cout << "Entre␣com␣c0␣:␣";
9   cin >> c0; cin.get();
10
11  cout << "Entre␣com␣c1␣:␣";
12  cin >> c1; cin.get();
13 }
14
15 void CFun1G::Saida() {
16   cout << "Funcao␣1G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣\n";
17 }
18
```

```
19 double CFun1G::f (double _x )  {
20         x = _x;
21         y = c0 + c1 * x;
22         return y;
23 }
24
25 istream& operator>>(istream& in,  CFun1G& funcao) {
26   in >> funcao.c0; in.get();
27   in >> funcao.c1;
28   return in;
29 }
30
31 ostream& operator<<(ostream& out, CFun1G& funcao) {
32     out << funcao.c0 << "␣" << funcao.c1 << "␣";
33     return out;
34 }
```

Listing 13.5: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1#ifndef CFun2G_H
2#define CFun2G_H
3#include "CFun1G.h"
4
5/// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
6class CFun2G : public CFun1G
7{
8 protected:
9  double c2{0.0}; ///< Coeficiente quadrático
10
11 public:
12  /// Construtor default (sem parâmetros)
13  CFun2G() = default;
14
15  /// Construtor sobrecarregado
16  CFun2G(double _c0, double _c1, double _c2): CFun1G(_c0,_c1), c2(_c2) {};
17
18  /// Destrutor
19  virtual ~CFun2G()      = default;
20
21  /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
22  virtual double f (double _x = 0 );
23
24  /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
25  virtual double f (double _x , double _c0, double _c1, double _c2 )  {
26    c0 = _c0; c1 = _c1; c2 = _c2;
27    return f(_x);
28  }
```

```
29
30   /// Seta valor de c2
31   void C2 ( double _c2 )         { c2 = _c2; };
32
33    /// Retorna valor de c2
34   double C2 ( ) const    { return c2; };
35
36   /// Entrada dos valores dos atributos.
37   virtual void Entrada();
38
39    /// Saída de atributos.
40   virtual void Saida();
41
42   /// Sobrecarga operador operator>> (Uso: cin >> funcao2g; )
43   friend std::istream& operator>>(std::istream& in,  CFun2G& funcao);
44
45   /// Sobrecarga operador operator<<  (Uso: cout << funcao2g; )
46   friend std::ostream& operator<<(std::ostream& out, CFun2G& funcao);
47};
48#endif
```

Listing 13.6: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1#include <iostream>
2using namespace std;
3
4#include "CFun2G.h"
5
6/// Entrada dos valores dos atributos.
```

```
7 void CFun2G::Entrada() {
8   cout << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
9   cout << "Inicia␣pedindo␣dados␣de␣1G␣y␣=␣c0␣+␣c1*x␣␣␣:\n";
10  CFun1G::Entrada(); // entrar com c0 e c1
11
12  cout << "Agora,␣entre␣com␣c2␣:␣"; // aquilo que Ã© novo, diferente
13  cin >> c2; cin.get();
14 }
15
16 /*/// Entrada dos valores dos atributos.
17 void CFun2G::Entrada() {
18   cout << "Entre com dados da funcao 2G y = c0 + c1*x + c2*x*x  :\n";
19   cout << "Entre com c0 : ";
20   cin >> c0; cin.get();
21   cout << "Entre com c1 : ";
22   cin >> c1; cin.get();
23   cout << "Entre com c2 : ";
24   cin >> c2; cin.get();
25 }*/
26
27 void CFun2G::Saida() {
28   cout << "Funcao␣2G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣+␣"<< c2 << "␣*␣x␣*␣x␣\n";
29 }
30
31 double CFun2G::f (double _x )  {
32   x = _x;
33   y = c0 + c1*x + c2*x*x;
34   return y ;
35 }
```

```
36
37istream& operator>>(istream& in,  CFun2G& funcao) {
38  in >> funcao.c0; in.get();
39  in >> funcao.c1; in.get();
40  in >> funcao.c2;
41  return in;
42}
43
44ostream& operator<<(ostream& out, CFun2G& funcao) {
45  out << funcao.c0 << "␣" << funcao.c1 << "␣"<< funcao.c2 << "␣";
46  return out;
47}
```

Listing 13.7: Arquivo de declaração da classe CFunExp - CFunExp.h.

```
1#ifndef CFunExp_H
2#define CFunExp_H
3#include "CFuncao.h"
4
5/// Representa funcao exponencial,  y = a*exp(b*x).
6// final indica que esta classe não terá herdeiras.
7class CFunExp final: public CFuncao
8{
9 protected:
10  double a{0.0};          ///< Coeficiente a da funcao exponencial y = a*exp(b*x)
11  double b{0.0};          ///< Coeficiente b da funcao exponencial y = a*exp(b*x)
12
13 public:
14  /// Construtor default
15  CFunExp()      = default;
16
17  /// Construtor sobrecarregado
18  CFunExp(double _a, double _b) : a(_a), b(_b) {} ;
19
20  /// Destrutor
21  virtual ~CFunExp()     = default;
22
23  /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
24  virtual  double f ( double _x  );
25
26  /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
27  virtual  double f ( double _x , double _a, double _b )  {
28      a = _a; b = _b;
```

```
29      return f(_x);
30  }
31
32  /// Seta valor de a
33  void A ( double _a )  { a = _a; };
34
35  /// Retorna valor de a
36  double A ( ) const            { return a; };
37
38  /// Seta valor de b
39  void B ( double _b )  { b = _b; }
40
41   /// Retorna valor de b
42  double B ( ) const            { return b; };
43
44  /// Entrada dos valores dos atributos.
45  virtual void Entrada();
46
47   /// Saída de atributos.
48  virtual void Saida();
49
50   /// Sobrecarga operador operator>>
51  friend std::istream& operator>>(std::istream& in,  CFunExp& funcao);
52
53  /// Sobrecarga operador operator<<
54  friend std::ostream& operator<<(std::ostream& out, CFunExp& funcao);
55};
56#endif
```

Listing 13.8: Arquivo de definição da classe CFunExp - CFunExp.cpp.

```
1#include <iostream>
2#include <cmath>      /// math.h no C, fornece a funÃ§Ã£o std::exp
3#include "CFunExp.h"
4using namespace std;
5
6void CFunExp::Entrada() {
7  cout << "Entre␣com␣dados␣da␣funcao␣2G␣␣y␣=␣a␣*␣exp(␣b␣*␣x␣);\n";
8  cout << "Entre␣com␣a␣:␣";
9  cin >> a; cin.get();
10
11  cout << "Entre␣com␣b␣:␣";
12  cin >> b; cin.get();
13}
14
15void CFunExp::Saida() {
16  cout << "Funcao␣Exp␣y␣=␣" << a << "␣*␣e^("<< b << "␣*␣x)␣\n";
17}
18
19double CFunExp::f ( double _x  ) {
20 x = _x;
21 return y = a * std::exp( b * x );
22}
23
24istream& operator>>(istream& in,  CFunExp& funcao) {
25  in >> funcao.a; in.get();
26  in >> funcao.b;
27  return in;
28}
```

```
29
30 ostream& operator<<(ostream& out, CFunExp& funcao) {
31     out << funcao.a << "␣" << funcao.b << "␣";
32     return out;
33 }
```

Listing 13.9: Arquivo de declaração da classe CIntegral - CIntegral.h.

```
1 /**************************************************************************
2 CIntegral.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CIntegral_H
8 #define CIntegral_H
9 #include "CFuncao.h"
10
11 /// Representa método numérico para cálculo área funções.
12 class CIntegral
13 {
14  protected:
15   double limInf{0.0};    ///< Limite Inferior do intervalo de integral
16   double limSup{1.0};    ///< Limite Superior do intervalo de integral
17   double dx{0.01};       ///< Intervalo dx
18   int    numInt{100};    ///< Numero intervalos
19   double area{0.0};      ///< Valor da área calculada
20
21  public:
22   /// Construtor default
23   CIntegral() = default;
24
25   /// Construtor sobrecarregado
26   CIntegral(double _limInf, double _limSup, double _numInt)
27         : limInf(_limInf), limSup(_limSup) {
28                         NumInt(_numInt);
```

```
29                          };
30
31                          /// Construtor sobrecarregado, cópia de CIntegral
32  CIntegral(const CIntegral& integral)  {
33                          this->limInf = integral.LimInf();
34                          this->limSup = integral.LimSup();
35                          this->numInt = integral.NumInt();
36                          this->dx     = integral.Dx();
37                          };
38
39  /// Destrutor
40  virtual ~CIntegral()    = default;
41
42  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
43  virtual  double Area ( CFuncao* funcao = nullptr ) = 0;
44
45  /// A sobrecarga do operador (), possibilita uso objeto como objeto função,
46  /// objeto se comporta como uma função.
47  double operator()( CFuncao* funcao  ) { return Area(funcao); }
48
49  /// Seta valor de limInf
50  /// @param _limInf novo valor de limInf
51  void LimInf ( double _limInf  )       { limInf = _limInf ; };
52
53  /// Retorna valor de limInf
54  /// @return retorna valor de limInf
55  double LimInf ( )     const           { return limInf; };
56
57  /// Seta valor de limSup
```

```
58   /// @param _limSup Seta novo valor de  limSup
59   void LimSup ( double _limSup )          { limSup = _limSup; };
60
61   /// Retorna valor de limSup
62   double LimSup ( )      const            {return limSup; } ;
63
64   /// Seta valor de dx
65   void Dx ( double _dx )                  { dx = _dx; numInt = (limSup - limInf) / dx;};
66
67   /// Retorna valor de dx
68   double Dx ( ) const                     { return dx;};
69
70   /// Seta valor de numInt
71   void NumInt ( int _numInt )             { numInt = _numInt;
72                                            dx = (limSup - limInf) / numInt;
73                                            };
74
75   /// Retorna valor de numInt
76   int NumInt ( )         const            { return numInt; } ;
77
78    /// Entrada dos valores dos atributos.
79   virtual void Entrada() ;
80 };
81 #endif
```

Listing 13.10: Arquivo de definição da classe CIntegral - CIntegral.cpp.

```
1 #include<iostream>
2 using namespace std;
```

```
3
4#include "CIntegral.h"
5
6/// Entrada dos valores dos atributos.
7void CIntegral::Entrada() {
8  cout << "Entre␣com␣dados␣do␣metodo␣integracao:\n";
9  cout << "Entre␣com␣Limite␣Inferior␣:␣";
10  cin  >> limInf ;
11  cout << "Entre␣com␣limite␣Superior␣:␣";
12  cin  >> limSup;
13  cout << "Entre␣com␣numero␣intervalos␣:␣";
14  cin  >> numInt; cin.get();
15
16  // Como numInt foi modificado, atualizo valor de dx
17  dx = (limSup - limInf) / numInt;
18}
```

Listing 13.11: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2                         CIntTrapezio.h - Copyright André D. Bueno
3
4 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
5 Implementação de uma pequena biblioteca de classes para funções e
6 cálculo de integral numérica.
7 **************************************************************************/
8
9 #ifndef CIntTrapezio_H
10 #define CIntTrapezio_H
11 #include "CIntegral.h"
12
13 /**
14   * Representa método numérico trapésio para cálculo área funções.
15   * @class CIntTrapezio
16   */
17 class CIntTrapezio : public CIntegral
18 {
19 public:
20   /// Construtor
21   CIntTrapezio()         = default;
22
23   /// Destrutor
24   virtual ~CIntTrapezio()        = default;
25
26   /**
27    * Calcula área da função no intervalo limInf->limSup considerando numInt.
28    * @return double
```

```
29    * @param  funcao
30    */
31   virtual  double Area ( CFuncao* funcao  ) override;
32
33 };
34 #endif
```

Listing 13.12: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1 #include "CIntTrapezio.h"
2
3 double CIntTrapezio::Area (CFuncao* funcao ) {
4   area = (*funcao)(limInf)*0.5;
5   double x = limInf;
6   for( int i = 1; i < numInt ; i++ )    {
7   //       area += funcao->f( limInf + i * dx );
8             x += dx;
9             area += (*funcao)( x );
10          }
11  area += (*funcao)(limSup)*0.5;
12  area *= dx ;
13  return area;
14 }
```

Listing 13.13: Arquivo de declaração da classe CIntSimpson - CIntSimpson.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CIntSimpson_H
8 #define CIntSimpson_H
9 #include "CIntTrapezio.h"
10
11 /// Representa método numérico simpson para cálculo área funções.
12 class CIntSimpson : public CIntTrapezio
13 {
14  public:
15   /// Construtor
16   CIntSimpson()  = default;
17   /// Destrutor
18   virtual ~CIntSimpson() = default;
19
20   /// Construtor sobrecarregado, cópia de CIntegral
21   // Possibilita construir um CIntSimpson a partir de um CIntTrapezio
22   CIntSimpson(const CIntegral& integral) {
23                         this->limInf = integral.LimInf();
24                         this->limSup = integral.LimSup();
25                         this->numInt = integral.NumInt();
26                         this->dx     = integral.Dx();
27                         };
28
```

```
29  /**
30   * Calcula área da função no intervalo limInf->limSup considerando numInt.
31   * @return double
32   * @param  funcao
33   */
34  virtual double Area ( CFuncao* funcao  ) override;
35};
36#endif
```

Listing 13.14: Arquivo de definição da classe CIntSimpson - CIntSimpson.cpp.

```
1#include "CIntSimpson.h"
2
3double CIntSimpson::Area (CFuncao* funcao ) {
4 area = 0.5* ( (*funcao)(limInf) /*ponto 0*/ + (*funcao)(limSup) ) ; /*ponto n*/      // [4]
5
6 for (int i = 1; i < (numInt - 2); i += 2)     {
7      area += 2.0 * (*funcao)(limInf + i * dx)       // ponto 1,...,n-3 // [1]
8                + (*funcao)(limInf + (i + 1) * dx);        // ponto 2,...,n-2 // [2]
9    }
10  area += 2.0 * (*funcao)(limSup - dx);               // ponto n-1       // [3]
11  area *= 2.0 * dx / 3.0;
12return area;
13}
```

Listing 13.15: Arquivo de declaração da classe CSimulador - CSimulador.h.

```
1 /**********************************************************************
2 CSimulador.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CSIMULADOR_H
8 #define CSIMULADOR_H
9
10 class CSimulador
11 {
12 public:
13   /// Construtor da classe
14   CSimulador()  = default;
15
16   /// Destrutor da classe
17   ~CSimulador() = default;
18
19   /// Executa a simulação.
20   int ExecutarSimulacao();
21 };
22 #endif
```

Listing 13.16: Arquivo de definição da classe CSimulador - CSimulador.cpp.

```
1 #include <iostream>
2 #include "CSimulador.h"
3 #include "CFun1G.h"
```

```
4#include "CFun2G.h"
5#include "CFunExp.h"
6#include "CIntTrapezio.h"
7#include "CIntSimpson.h"
8
9using namespace std;    // Permite acesso direto a cin, cout,....etc
10
11/// Executa a simulação.
12int CSimulador::ExecutarSimulacao() {
13  // looping para selecao da funcao.
14  int tipoFuncao;
15  do {
16    cout << "Qual␣funcao␣quer␣criar:␣\n␣\
17␣␣␣␣␣␣␣␣␣␣␣␣␣␣Primeiro␣grau........1␣\n␣\
18␣␣␣␣␣␣␣␣␣␣␣␣␣␣Segundo␣grau.........2␣\n␣\
19␣␣␣␣␣␣␣␣␣␣␣␣␣␣Exponencial..........3␣:␣";
20    cin >> tipoFuncao; cin.get();
21
22  }while(tipoFuncao > 3 and tipoFuncao < 1);
23
24  // switch usada para criar dinamicamente objeto de interesse.
25  CFuncao* funcao;      // Cria ponteiro para funcao
26  switch(tipoFuncao)  {
27    case 1: funcao = new CFun1G;  break;
28    case 2: funcao = new CFun2G;  break;
29    case 3: funcao = new CFunExp; break;
30  }
31
32  // Executa Entrada de dados para funcao criada
```

```
33  cout << "Entre␣com␣os␣coeficientes␣da␣função␣escolhida:\n";
34  funcao->Entrada();
35  cout << "Função␣escolhida␣ficou␣da␣seguinte␣forma:";
36  funcao->Saida();
37
38  CIntegral* integral;  // Cria ponteiro para objeto de integracão, usado para calculo da area
39
40// looping para selecao do tipo de integral.
41  int tipoIntegral;
42  do  {
43    cout << "Qual␣método␣integracao␣quer␣criar:␣\n"
44         << "Trapezio.....1␣\n"
45         << "Simpson.....2␣\n"
46         << "Gauss.....3␣:␣";
47    cin >> tipoIntegral; cin.get();
48
49  }while(tipoIntegral > 3 && tipoIntegral < 1);
50
51  // switch usada para criar dinamicamente objeto de interesse.
52  switch(tipoIntegral)  {
53    case 1:  integral = new CIntTrapezio;  break;
54    case 2:  integral = new CIntSimpson;   break;
55    case 3:                                      // implementar gauss é uma tarefa sua!
56        default: integral = new CIntSimpson;   break; // enquanto não implementamos Gauss!
57  }
58  // Executa Entrada de dados para integral escolhida
59  cout << "Entre␣com␣os␣atributos␣do␣objeto␣de␣integral:\n";
60  integral->Entrada();
61  // cin >> integral; cin.get();
```

```
62
63  // Usa objeto integral para calcular area da funcao selecionada
64  std::cout << "\nArea␣␣=␣" << integral->Area(funcao) <<  "\n";
65  std::cout << "\nArea␣(usa␣sobrecarga)␣␣=␣" << (*integral)(funcao) <<  "\n";
66
67  std::cin.get();
68  return 0;
69}
```

Listing 13.17: Programa para cálculo da integral do trapésio: Usa hierarquia classes CFuncao e CIntegral.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Acrescenta polimorfismo.
 *
 * Para compilar:
 * g++ -std=c++11 *.cpp -o integral
*/
#include "CSimulador.h"

/// Função principal do programa
int main(int argc, char **argv) {
  CSimulador simulador;
  simulador.ExecutarSimulacao();
  return 0;
}
```

```
[bueno@localhost src]$ ./integral
Qual funcao quer criar:
             Primeiro grau........1
             Segundo grau.........2
             Exponencial..........3 : 1
Entre com os coeficientes da funcao escolhida:
Entre com dados da funcao 1G y = c0 + c1*x  :
Entre com c0 : 5
Entre com c1 :
0
Funcao escolhida ficou da seguinte forma:Funcao 1G y = 5 + 0 * x
Qual metodo integracao quer criar:
Trapesio........1
Simpson.........2
Gauss...........3 : 1
Entre com os atributos do objeto de integral:
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 10
Entre com numero intervalos : 100

Area  = 50
```

Capítulo 14

v1.1 - Programação Genérica (C++03)

14.1 Características

- Acrescenta uso algoritmos genéricos da STL;
- Uso de containers;
- Uso de iteradores (`iterator`);
- Novos conceitos de C++11, como funções lambda.

14.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 14.1 o diagrama de classes e a Figura 14.2 o diaqrama de sequência.

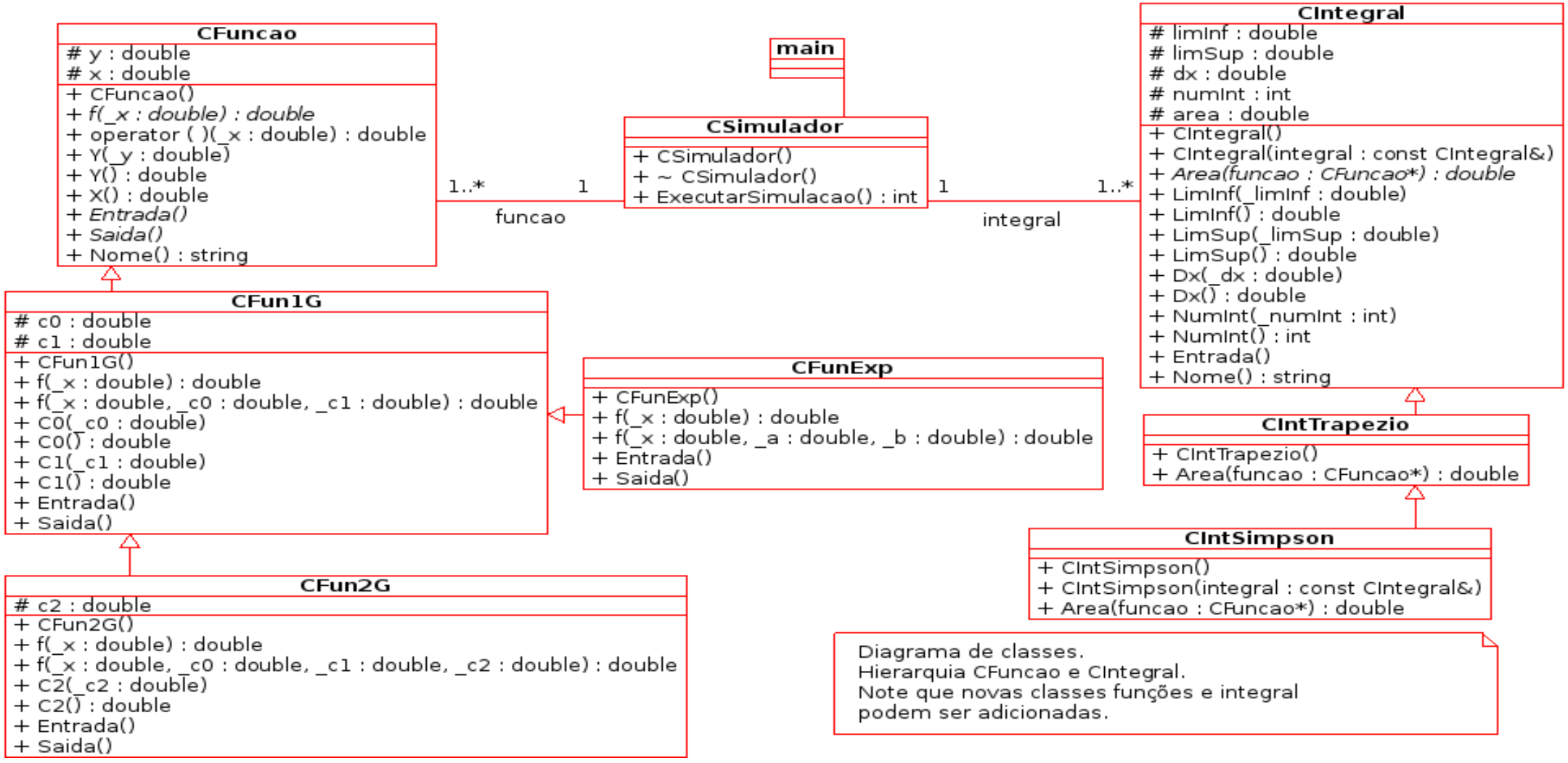


Figura 14.1: Diagrama de classe - versão 1.1

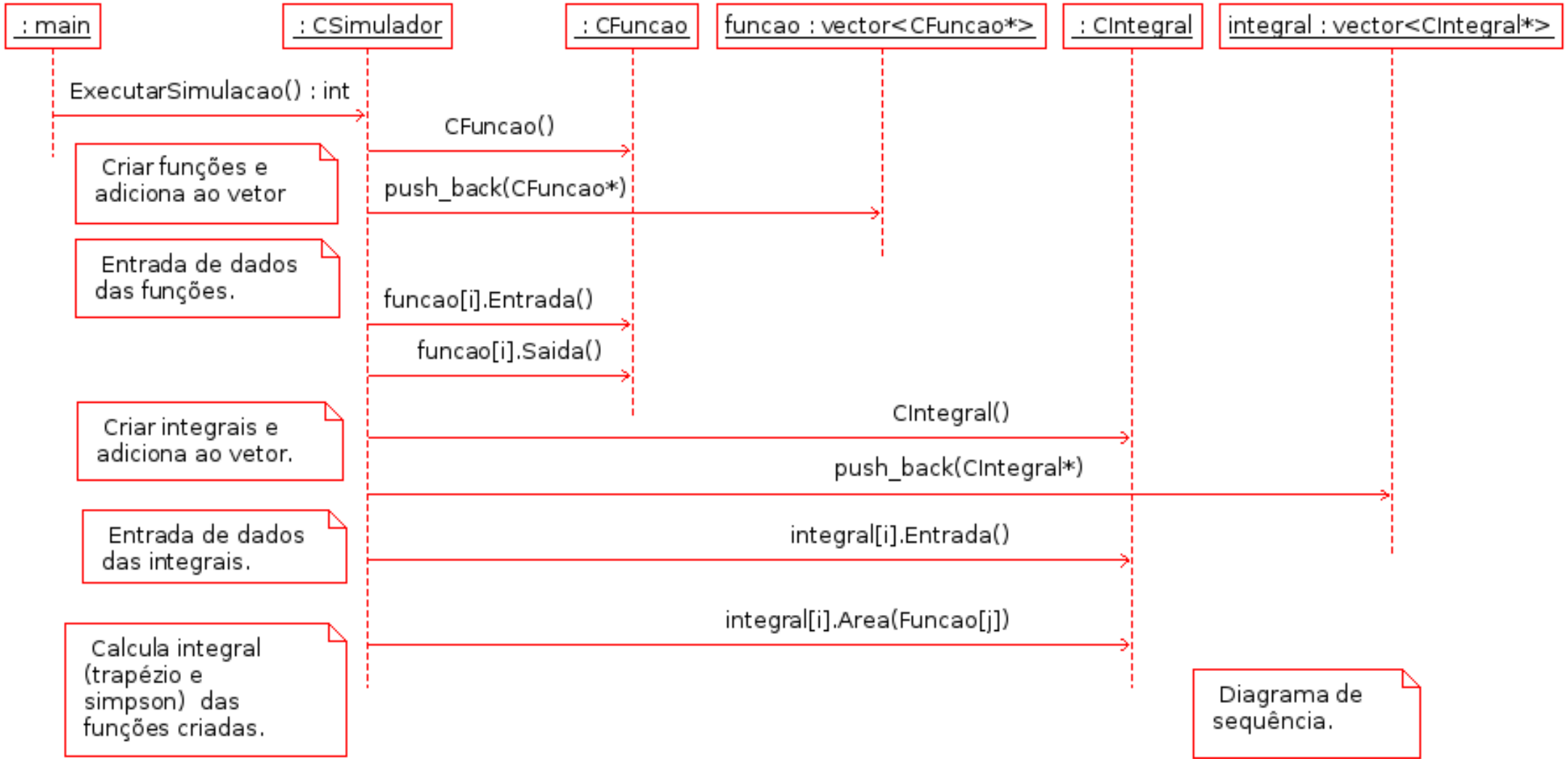


Figura 14.2: Diagrama de sequência - v1.1

14.3 Códigos

Listing 14.1: Arquivo de declaração da classe CFuncao - CFuncao.h.

```
1 /**************************************************************************
2 CFuncao.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 // Define variável de pré-processamento que impede dupla compilação deste arquivo.
8 #ifndef CFUNCAO_H
9 #define CFUNCAO_H
10
11 #include<iostream>
12
13 /// Representa uma função genérica.
14 class CFuncao
15 {
16  protected:
17   double y; ///< Representa variável dependente.
18   double x; ///< Representa variável independente.
19
20  public:
21   /// Construtor
22   CFuncao()                        {};
23
24   /// Calcula valor da função na posição x, isto é, y = f(x);.
```

```
25  virtual  double f ( double _x = 0.0 ) = 0;
26
27  /// Calcula valor da função na posição x, isto é, y = f(x);.
28  /// A sobrecarga do operador (), possibilita uso objeto como objeto função, objeto se comporta
29  /// como uma função.
30  double operator()( double _x = 0.0 )  { x=_x; return f(x); }
31
32  /// Seta valor de y.
33  /// @param _y novo valor de y
34  void Y ( double _y )          { y = _y; };
35
36  /// Obtém valor de y.
37  /// Retorna valor de y.
38  double Y ( ) const                    { return y; };
39
40  // Seta valor de x.
41  // @param _x novo valor de x.
42  //void X ( double _x )                { x = _x; };
43
44  /// Retorna valor de x.
45  /// @return retorna valor de x.
46  double X ( ) const                    { return x; };
47
48  /// Entrada dos valores dos atributos.
49  virtual void Entrada() = 0;
50
51  /// Saída de atributos.
52  virtual void Saida() = 0;
53
```

```
54   /// Nome da função
55   virtual std::string Nome() { return "CFuncao";}
56
57   /// Sobrecarga operador operator>>
58   friend std::istream& operator>>(std::istream& in,  CFuncao& funcao);
59
60   /// Sobrecarga operador operator<<
61   friend std::ostream& operator<<(std::ostream& out, CFuncao& funcao);
62 };
63 #endif
```

Listing 14.2: Arquivo de definição da classe CFuncao - CFuncao.cpp.

```
1 #include "CFuncao.h"
2 using namespace std;
3
4 istream& operator>>(istream& in,  CFuncao& funcao) {
5   in >> funcao.x; in.get();
6   in >> funcao.y;
7   return in;
8 }
9
10 ostream& operator<<(ostream& out, CFuncao& funcao)  {
11     out <<  funcao.Nome() << "␣"<<  funcao.x << "␣" << funcao.y;
12     return out;
13 }
```

Listing 14.3: Arquivo de declaração da classe CFun1G - CFun1G.h.

```
1 /**********************************************************************
2 CFun1G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFUN1G_H
8 #define CFUN1G_H
9 #include "CFuncao.h"
10
11 ///Representa funcao do primeiro grau, y = c0 + c1 * x;.
12 class CFun1G : public CFuncao
13 {
14  protected:
15   double c0; ///< Representa coeficiente linear
16   double c1; ///< Representa coeficiente angular
17
18  public:
19   /// Construtor
20   CFun1G()       = default;
21
22   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
23   virtual double f (double _x );
24
25   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
26   virtual double f (double _x , double _c0, double _c1) {
27     c0 = _c0; c1 = _c1;
28     return f(_x);
```

```
29  }
30
31  /// Seta valor de c0.
32  /// @param _c0 novo valor de c0.
33  void C0 ( double _c0 )        {  c0 = _c0; };
34
35  /// Retorna valor de c0.
36  /// @return retorna valor de c0.
37  double C0 ( ) const   {  return c0; };
38
39  /// Seta valor de c1.
40  /// @param _c1 novo valor de c1
41  void C1 ( double _c1 )        {  c1 = _c1; };
42
43  /// Retorna valor de c1
44  /// @return retorna valor de c1
45  double C1 ( ) const   {  return c1; };
46
47  /// Entrada dos valores dos atributos.
48  virtual void Entrada();
49
50  /// Saída de atributos.
51  virtual void Saida();
52
53  /// Nome da função
54  virtual std::string Nome() { return "CFun1G: y = c0 + c1 * x";}
55
56  /// Sobrecarga operador operator>>
57  friend std::istream& operator>>(std::istream& in,  CFun1G& funcao);
```

```
58
59   /// Sobrecarga operador operator<<
60   friend std::ostream& operator<<(std::ostream& out, CFun1G& funcao);
61 };
62 #endif
```

Listing 14.4: Arquivo de definição da classe CFun1G - CFun1G.cpp.

```
1 #include <iostream>
2 #include "CFun1G.h"
3 using namespace std;
4
5 void CFun1G::Entrada() {
6   cout << Nome();
7   cout << "\nEntre␣com␣c0␣:␣";
8   cin >> c0;
9   cout << "Entre␣com␣c1␣:␣";
10  cin >> c1; cin.get();
11 }
12
13 void CFun1G::Saida() {
14  cout << "Funcao␣1G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣\n";
15 }
16
17 double CFun1G::f (double _x )  {
18  x = _x;
19  y = c0 + c1 * x;
20  return y;
21 }
```

```
22
23 istream& operator>>(istream& in,  CFun1G& funcao) {
24   in >> funcao.c0; in.get();
25   in >> funcao.c1;
26   return in;
27 }
28
29 ostream& operator<<(ostream& out, CFun1G& funcao) {
30   out   << funcao.Nome() << "␣" << funcao.c0 << "␣" << funcao.c1 << "␣";
31   return out;
32 }
```

Listing 14.5: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
/*********************************************************************
CFun2G.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
*********************************************************************/
#ifndef CFUN2G_H
#define CFUN2G_H
#include "CFun1G.h"

/// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
class CFun2G : public CFun1G
{
 protected:
  double c2;                    ///< Coeficiente quadrático

 public:
  /// Construtor
  CFun2G()                      {};

  /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
  virtual double f (double _x = 0 );

  /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
  virtual double f (double _x , double _c0, double _c1, double _c2 ) {
    c0 = _c0; c1 = _c1; c2 = _c2;
    f(_x);
  }
```

```
29
30  /// Seta valor de c2
31  void C2 ( double _c2 )          { c2 = _c2; };
32
33  /// Retorna valor de c2
34  double C2 ( ) const    { return c2; };
35
36  /// Entrada dos valores dos atributos.
37  virtual void Entrada();
38
39  /// Saída de atributos.
40  virtual void Saida();
41
42  /// Nome da função
43  virtual std::string Nome() { return "CFun2G: y = c0 + c1 * x + c2 * x * x";}
44
45  /// Sobrecarga operador operator>>
46  friend std::istream& operator>>(std::istream& in,  CFun2G& funcao);
47
48  /// Sobrecarga operador operator<<
49  friend std::ostream& operator<<(std::ostream& out, CFun2G& funcao);
50};
51#endif
```

Listing 14.6: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1#include <iostream>
2#include "CFun2G.h"
3using namespace std;
```

```
4
5 void CFun2G::Entrada() {
6   CFun1G::Entrada();
7   cout << "Entre␣com␣c2␣:␣";
8   cin >> c2; cin.get();
9 }
10
11 void CFun2G::Saida() {
12   cout << "Funcao␣2G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣+␣"<< c2 << "␣*␣x␣*␣x␣\n";
13 }
14
15 double CFun2G::f (double _x  ) {
16  x = _x;
17  y = c0 + c1*x + c2*x*x;
18  return y ;
19 }
20
21 istream& operator>>(istream& in,  CFun2G& funcao) {
22   in >> funcao.c0; in.get();
23   in >> funcao.c1; in.get();
24   in >> funcao.c2;
25   return in;
26 }
27
28 ostream& operator<<(ostream& out, CFun2G& funcao) {
29   out  << funcao.Nome() << "␣"  << funcao.c0 << "␣" << funcao.c1 << "␣"<< funcao.c2 << "␣";
30   return out;
31 }
```

Listing 14.7: Arquivo de declaração da classe CFunExp - CFunExp.h.

```
/**************************************************************************
CFunExp.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CFUNEXP_H
#define CFUNEXP_H
#include "CFun1G.h"

/// Representa funcao exponencial,  y = a*exp(b*x).
class CFunExp : public CFun1G
{
 protected:
  using CFun1G::c0;      ///< Coeficiente da funcao exponencial y = c0*exp(c1*x)
  using CFun1G::c1;      ///< Coeficiente da funcao exponencial y = c0*exp(c1*x)

 public:
  /// Construtor
  CFunExp() = default;

 /// Calcula valor da função exponencial na posição x,  y = c0*exp(b*c1).
 virtual  double f ( double _x = 0 );

 /// Calcula valor da função exponencial na posição x,  y = c0*exp(b*c1).
 virtual  double f ( double _x , double _c0, double _c1 )   {
       c0 = _c0; c1 = _c1;
       return f(_x);
```

```
29  }
30
31  // Entrada dos valores dos atributos.
32  //virtual void Entrada();
33
34  /// Saída de atributos.
35  virtual void Saida();
36
37  /// Nome da função
38  virtual std::string Nome() { return "CFunExp: y = c0*exp(b*c1)";}
39
40  /// Sobrecarga operador operator>>
41  friend std::istream& operator>>(std::istream& in,  CFunExp& funcao);
42
43  /// Sobrecarga operador operator<<
44  friend std::ostream& operator<<(std::ostream& out, CFunExp& funcao);
45};
46#endif
```

Listing 14.8: Arquivo de definição da classe CFunExp - CFunExp.cpp.

```
1#include <iostream>
2#include <cmath>     /// math.h no C
3#include "CFunExp.h"
4using namespace std;
5
6// void CFunExp::Entrada() {
7//    cout << "Entre com c0 : ";
8//    cin >> c0;
```

```
9 //    cout << "Entre com c1 : ";
10 //    cin >> c1; cin.get();
11 // }
12
13 void CFunExp::Saida() {
14   cout << "Funcao␣Exp␣y␣=␣" << c0 << "␣*␣e^("<< c1 << "␣*␣x)␣\n";
15 }
16
17 double CFunExp::f ( double _x  )  {
18  x = _x;
19  return y = c0 * exp( c1 * x );
20 }
21
22 istream& operator>>(istream& in,  CFunExp& funcao) {
23   in >> funcao.c0; in.get();
24   in >> funcao.c1;
25   return in;
26 }
27
28 ostream& operator<<(ostream& out, CFunExp& funcao) {
29   out << funcao.c0 << "␣" << funcao.c1 << "␣";
30   return out;
31 }
```

Listing 14.9: Arquivo de declaração da classe CIntegral - CIntegral.h.

```
1 /**************************************************************************
2 CIntegral.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CINTEGRAL_H
8 #define CINTEGRAL_H
9 #include "CFuncao.h"
10
11 class CIntegral
12 {
13  protected:
14   double limInf{0.0};    ///< Limite Inferior do intervalo de integral
15   double limSup{1.0};    ///< Limite Inferior do intervalo de integral
16   double dx{0.0};        ///< Intervalo dx
17   int    numInt{101};    ///< Numero intervalos, numero pontos
18   double area{0.0};      ///< Valor da área calculada
19
20  public:
21   /// Construtor default
22   CIntegral()           { NumInt(101); };
23
24   /// Construtor sobrecarregado
25   CIntegral(double _limInf, double _limSup, double _numInt)
26         : limInf(_limInf), limSup(_limSup) {
27                         NumInt(_numInt);
28                         };
```

```
29
30  /// Construtor de cópia de CIntegral
31  // note que chama o outro construtor (c++11)
32  CIntegral(const CIntegral& integral):CIntegral(integral.LimInf(),
33                        integral.LimSup(),integral.NumInt()) {};
34
35  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
36  virtual  double Area ( CFuncao* funcao ) = 0;
37
38  /// A sobrecarga do operador (), possibilita uso objeto como objeto função,
39  /// objeto se comporta como uma função.
40  double operator()( CFuncao* funcao  ) { return Area(funcao); }
41
42
43  /// Nome do método integração
44  virtual std::string Nome() { return "CIntegral";}
45
46  /// Seta valor de limInf
47  void LimInf ( double _limInf  )       { limInf = _limInf ; };
48
49  /// Retorna valor de limInf
50  /// @return retorna valor de limInf
51  double LimInf ( )     const           { return limInf; };
52
53  /// Seta valor de limSup
54  /// @param _limSup Seta novo valor de  limSup
55  void LimSup ( double _limSup )        { limSup = _limSup; };
56
57  /// Retorna valor de limSup
```

```
58   /// @return retorna valor de limSup
59   double LimSup ( )      const            {return limSup; } ;
60
61   /// Seta valor de dx
62   /// @param _dx o novo valor de  dx
63   void Dx ( double _dx )                  { dx = _dx; numInt = (limSup - limInf) / dx;};
64
65   /// Retorna valor de dx
66   double Dx ( ) const                     { return dx;};
67
68   /// Seta valor de numInt
69   void NumInt ( int _numInt )
70                                           { numInt = _numInt;
71                                           dx = (limSup - limInf) / numInt;
72                                           };
73
74   /// Retorna valor de numInt
75   int NumInt ( )         const            { return numInt; } ;
76
77    /// Entrada dos valores dos atributos.
78   virtual void Entrada() ;
79};
80#endif
```

Listing 14.10: Arquivo de definição da classe CIntegral - CIntegral.cpp.

```
1#include<iostream>
2using namespace std;
3
```

```
4#include "CIntegral.h"
5
6void CIntegral::Entrada() {
7  cout << "Entre␣com␣dados␣do␣metodo␣integracao␣-␣" << Nome() << "␣:\n";
8  cout << "Entre␣com␣Limite␣Inferior␣:␣";
9   cin >> limInf ;
10  cout << "Entre␣com␣limite␣Superior␣:␣";
11   cin >> limSup;
12  cout << "Entre␣com␣numero␣intervalos␣:␣";
13   cin >> numInt; cin.get();
14
15  // Como numInt foi modificado, atualizo valor de dx
16  dx = (limSup - limInf) / numInt;
17}
```

Listing 14.11: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CINTTRAPEZIO_H
8 #define CINTTRAPEZIO_H
9 #include "CIntegral.h"
10
11 /// Representa método numérico trapésio para cálculo área funções.
12 class CIntTrapezio : public CIntegral
13 {
14  public:
15   /// Construtor
16   CIntTrapezio() = default;
17
18   /// Construtor sobrecarregado
19   CIntTrapezio(double _limInf, double _limSup, double _numInt)
20         : CIntegral(_limInf, _limSup,_numInt) {};
21
22   /// Construtor de cópia
23   // note que chama o outro construtor (c++11)
24   CIntTrapezio(const CIntegral& integral) : CIntegral(integral.LimInf(),
25                        integral.LimSup(),integral.NumInt()) {};
26
27   /// Calcula área da função no intervalo limInf->limSup considerando numInt.
28   virtual  double Area ( CFuncao* funcao ) override;
```

```
29
30   /// Nome do método integração
31   virtual std::string Nome() { return "CIntTrapezio";}
32};
33#endif
```

Listing 14.12: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1#include "CIntTrapezio.h"
2using namespace std;
3
4double CIntTrapezio::Area (CFuncao* funcao ) {
5 area = (*funcao)(limInf)*0.5;
6 double x = limInf;
7 for( int i = 1; i < numInt ; i++ )     {
8//         area += funcao->f( limInf + i * dx );
9           x += dx;
10          area += (*funcao)( x );
11        }
12 area += (*funcao)(limSup)*0.5;
13 area *= dx ;
14 return area;
15}
```

Listing 14.13: Arquivo de declaração da classe CIntSimpson - CIntSimpson.h.

```
/**************************************************************************
CIntTrapezio.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CINTSIMPSON_H
#define CINTSIMPSON_H
#include "CIntTrapezio.h"

/// Representa método numérico simpson para cálculo área funções.
class CIntSimpson : public CIntTrapezio
{
 public:
  /// Construtor
  CIntSimpson()  = default;

  /// Construtor sobrecarregado
  CIntSimpson(double _limInf, double _limSup, double _numInt)
        : CIntTrapezio(_limInf, _limSup,_numInt) {};

  /// Construtor de cópia
  CIntSimpson(const CIntegral& integral) : CIntTrapezio(integral.LimInf(),
                       integral.LimSup(),integral.NumInt() ) {};

//    /// Construtor sobrecarregado, cópia de CIntegral
//    // Possibilita construir um CIntSimpson a partir de um CIntTrapezio
//    CIntSimpson(const CIntegral& integral) {
```

```
29 //                      this->limInf = integral.LimInf();
30 //                      this->limSup = integral.LimSup();
31 //                      this->numInt = integral.NumInt();
32 //                      this->dx     = integral.Dx();
33 //                      };
34
35  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
36  virtual  double Area ( CFuncao* funcao ) override;
37
38   /// Nome do método integração
39   virtual std::string Nome() { return "CIntSimpson";}
40 };
41 #endif
```

Listing 14.14: Arquivo de definição da classe CIntSimpson - CIntSimpson.cpp.

```
1 #include "CIntSimpson.h"
2 using namespace std;
3
4 double CIntSimpson::Area ( CFuncao* funcao ) {
5  area = 0.5 * ( (*funcao)(limInf)                    // ponto 0      // [0]
6              + (*funcao)(limSup) ) ;                  // ponto n      // [4]
7
8  for ( int i = 1; i < (numInt - 2); i += 2 )
9       area += 2.0 * (*funcao)(limInf + i * dx); // ponto 1,...,n-3 // [1]
10 for ( int i = 1; i < (numInt - 2); i += 2 )
11      area += (*funcao)(limInf + (i + 1) * dx); // ponto 2,...,n-2 // [2]
12
13 area += 2.0 * (*funcao)(limSup - dx);                // ponto n-1       // [3]
```

```
14 area *= 2.0 * dx / 3.0;
15
16return area;
17}
```

Listing 14.15: Arquivo de declaração da classe CSimulador - CSimulador.h.

```
/**************************************************************************
CSimulador.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CSIMULADOR_H
#define CSIMULADOR_H

class CSimulador
{
 public:
  /// Construtor da classe
  CSimulador()                          {};

  /// Destrutor da classe
  virtual ~CSimulador()                 {};

  /// Executa a simulação.
  int ExecutarSimulacao();
};
#endif
```

Listing 14.16: Arquivo de definição da classe CSimulador - CSimulador.cpp.

```
#include <iostream>             // Biblioteca padrão
#include <vector>
#include <algorithm>
```

```
4#include <functional>
5#include <utility>
6
7#include "CSimulador.h"          // Biblioteca do programador
8#include "CFun1G.h"
9#include "CFun2G.h"
10#include "CFunExp.h"
11#include "CIntTrapezio.h"
12#include "CIntSimpson.h"
13
14using namespace std;            // Traz para nosso escopo cin, cout,....etc
15
16/// Executa a simulação.
17int CSimulador::ExecutarSimulacao() {
18/// ----------->Versão 1: Usando vector, new, push_back, size, iterator
19 {
20  vector< CFuncao* >  vfuncao;              // Cria vetor de funções
21  vfuncao.push_back ( new CFun1G );         // Adiciona objetos ao vetor
22  vfuncao.push_back ( new CFun2G );
23  vfuncao.push_back ( new CFunExp );
24
25  // Executa entrada de dados para funcao criada
26  for ( int i = 0; i < vfuncao.size(); i++ )  {
27    cout << "-->Entre com os coeficientes da função:\n" ;
28    vfuncao[i]->Entrada();
29    cout << "Função ficou da seguinte forma:\n";
30    vfuncao[i]->Saida();
31    cout << endl;
32  }
```

```
33
34  // Cria vetor vintegral com dois ponteiros
35  vector< CIntegral* > vintegral;           // Cria vetor de objetos de integracão
36  vintegral.push_back( new CIntTrapezio );  // Cria objeto trapezio e adiciona ao vetor
37  vintegral.push_back( new CIntSimpson );   // Cria objeto simpson e adiciona ao vetor
38
39  cout << "Entrada␣dados␣método␣trapézio:" << endl;
40  vintegral[0]->Entrada();                  // Executa entrada de dados trapézio
41  cout << "Entrada␣dados␣método␣simpson:" << endl;
42  vintegral[1]->Entrada();                  // Executa entrada de dados simpson
43  // Para cada função criada, calcula área usando trapezio e simpson
44  int i = 0;
45  for ( vector< CFuncao* >::iterator funcao = vfuncao.begin(); funcao != vfuncao.end(); funcao++ ) {
46    cout << "\n\n------>Função␣:␣" << i++ << "\n" ;
47    (*funcao)->Saida();
48    // Cria iterador para vetor de integrais (uma espécie de objeto ponteiro)
49    vector< CIntegral* >::iterator integral = vintegral.begin();
50    cout << "\nArea␣Trapezio␣=␣" << (*integral)->Area(*funcao) ;
51    integral++;
52    cout << "\nArea␣Simpson␣␣=␣" << (*integral)->Area(*funcao) ;
53  }
54  cout << endl;
55  }
56
57  /// ----------->Versão 2: Usando vector, new, auto, for_each e funções lambda de C++11 (note redução número
       linhas!)
58  vector< CFuncao* >  vfuncao { new CFun1G , new CFun2G , new CFunExp };
59
60  cout << "\n\nEntre␣com␣os␣coeficientes␣das␣diferentes␣funções:\n";
```

```
61  // Usando função lambda
62  for_each(vfuncao.begin(), vfuncao.end(), [](CFuncao* f) {
63          cout << "\n"; f->Entrada(); f->Saida(); });
64
65  // Cria vetor vintegral com dois ponteiros
66  vector< CIntegral* > vintegral {new CIntTrapezio , new CIntSimpson };
67
68                                          // Solicita entrada de dados dos objetos de integração
69  cout << "Entre com os coeficientes dos métodos de integração:\n";
70  for_each(vintegral.begin(), vintegral.end(), [](CIntegral* i) {i->Entrada();});
71
72  // Para cada função criada, calcula área usando trapezio e simpson
73  // note que o iterador é para CFuncao* ou seja, é um ponteiro para ponteiro
74  // por isso preciso de *funcao para desreferenciar o iterador e acessar o ponteiro da funcao
75  int i{0};
76  for ( auto funcao : vfuncao )   {
77    cout << "\n\n------>Função : " << i++ << endl;
78    funcao->Saida();
79    cout << "\nArea Trapezio = " << vintegral[0]->Area(funcao) ;
80    cout << "\nArea Simpson  = " << vintegral[1]->Area(funcao) ;
81  }
82  cout << endl;
83  return 0;
84}
```

Listing 14.17: Programa para cálculo da integral do trapézio: Usa hierarquia classes CFuncao e CIntegral.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Acrescenta uso algoritmo genérico;
 * - Uso de iteradores;
 * - Uso de friend std::istream& operator>>(std::istream& in,  CFuncao& funcao);
 * - Novos conceitos de C++11, como funções lambda.
 *
 * Para compilar:
 * g++ -std=c++11 *.cpp -o integral
*/
#include "CSimulador.h"

/// Função principal do programa
int main(int argc, char **argv) {
  CSimulador simulador;
  simulador.ExecutarSimulacao();
  return 0;
}
```

```
[bueno@localhost src]$ ./integral
-->Entre com os coeficientes da funcao:
CFun1G: y = c0 + c1 * x
Entre com c0 : 1
Entre com c1 : 2
Funcao ficou da seguinte forma:
Funcao 1G y = 1 + 2 * x

-->Entre com os coeficientes da funcao:
CFun2G: y = c0 + c1 * x + c2 * x * x
Entre com c0 : 1
Entre com c1 : 2
Entre com c2 : 3
Funcao ficou da seguinte forma:
Funcao 2G y = 1 + 2 * x + 3 * x * x

-->Entre com os coeficientes da funcao:
CFunExp: y = c0*exp(b*c1)
Entre com c0 : 1
Entre com c1 : 2
Funcao ficou da seguinte forma:
Funcao Exp y = 1 * e^(2 * x)

Entrada dados método trapézio:
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 1
Entre com numero intervalos : 101
Entrada dados método simpson:
```

```
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 1
Entre com numero intervalos : 101

------>Funcao : 0
CFun1G: y = c0 + c1 * x
Funcao 1G y = 1 + 2 * x

Area Trapezio = 2
Area Simpson  = 1.97072

------>Funcao : 1
CFun2G: y = c0 + c1 * x + c2 * x * x
Funcao 2G y = 1 + 2 * x + 3 * x * x

Area Trapezio = 3.00005
Area Simpson  = 2.94228

------>Funcao : 2
CFunExp: y = c0*exp(b*c1)
Funcao Exp y = 1 * e^(2 * x)

Area Trapezio = 3.19463
Area Simpson  = 3.12444


Entre com os coeficientes das diferentes funções:
CFun1G: y = c0 + c1 * x
```

```
Entre com c0 : 1
Entre com c1 : 2
Funcao 1G y = 1 + 2 * x
CFun2G: y = c0 + c1 * x + c2 * x * x
Entre com c0 : 1
Entre com c1 : 2
Entre com c2 : 3
Funcao 2G y = 1 + 2 * x + 3 * x * x
CFunExp: y = c0*exp(b*c1)
Entre com c0 : 1
Entre com c1 : 2
Funcao Exp y = 1 * e^(2 * x)
Entre com os coeficientes dos métodos de integração:
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 1
Entre com numero intervalos : 101
Entre com dados do metodo integracao:
Entre com Limite Inferior : 0
Entre com limite Superior : 1
Entre com numero intervalos : 101


------>Funcao : 0
Funcao 1G y = 1 + 2 * x

Area Trapezio = 2
Area Simpson  = 1.97072

```

```
------>Funcao : 1
Funcao 2G y = 1 + 2 * x + 3 * x * x

Area Trapezio = 3.00005
Area Simpson  = 2.94228

------>Funcao : 2
Funcao Exp y = 1 * e^(2 * x)

Area Trapezio = 3.19463
Area Simpson  = 3.12444
```

Capítulo 15

v1.2 - Programação Genérica (C++11)

15.1 Características

- Inclui mecanismos de programação genérica de C++11.
- Novos conceitos de C++11 como `unique_ptr` e `make_ptr`.
- Função Area recebe ponteiro do tipo: `unique_ptr<CFuncao> *`
- Funções `Entrada` e `Saida` recebem stream, permitindo uso de `Saida(cout)` ou `Saida(fout)`.

15.2 Diagramas

A Figura 9.1 mostra o diagrama de caso de uso. A Figura 15.1 o diagrama de classes e a Figura **??** o diaqrama de sequência.

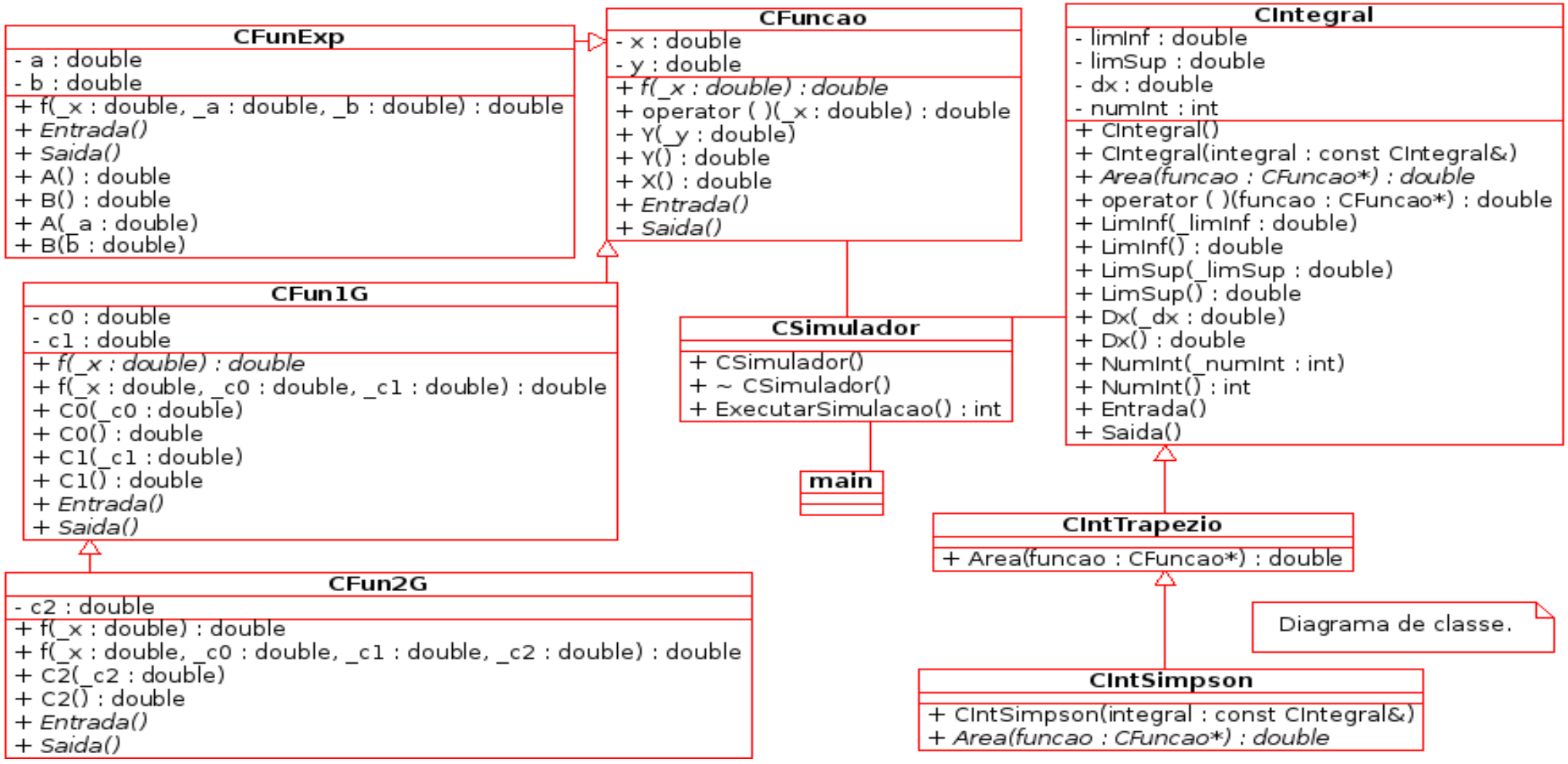


Figura 15.1: Diagrama de classe - versão 1.2

15.3 Códigos

Listing 15.1: Arquivo de declaração da classe CFuncao - CFuncao.h.

```
1 /**************************************************************************
2 CFuncao.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 // Define variável de pré-processamento que impede dupla compilação deste arquivo.
8 #ifndef CFUNCAO_H
9 #define CFUNCAO_H
10
11 #include<iostream>
12
13 /// Representa uma função genérica.
14 class CFuncao
15 {
16  protected:
17   double y;              ///< Representa variável dependente.
18   double x;              ///< Representa variável independente.
19
20  public:
21   /// Construtor
22   CFuncao() = default;
23
24   /// Desstrutor
```

```
25  virtual ~CFuncao() = default;
26
27  /// Calcula valor da função na posição x, isto é, y = f(x);.
28  virtual  double f ( double _x  ) = 0;
29
30  /// Calcula valor da função na posição x, isto é, y = f(x);.
31  /// A sobrecarga do operador (), possibilita uso objeto como objeto função, objeto se comporta
32  /// como uma função.
33  inline double operator()( double _x  ) { x=_x; return f(x); } // Uso: objetofuncao(x) ou (*ponteirofuncao)(x)
34
35  /// Seta valor de y.
36  /// @param _y novo valor de y
37  inline void Y ( double _y )           { y = _y; };
38
39  /// Obtém valor de y.
40  /// Retorna valor de y.
41  inline double Y ( ) const             { return y; };
42
43  // Seta valor de x.
44  // @param _x novo valor de x.
45  //void X ( double _x )                { x = _x; };
46
47  /// Retorna valor de x.
48  /// @return retorna valor de x.
49  inline double X ( ) const             { return x; };
50
51  /// Entrada dos valores dos atributos. // ex: funcao->Entrada(cin,cout);
52  virtual void Entrada(std::istream& in = std::cin, std::ostream& out = std::cout)  = 0;
53
```

```
54   /// Saída de atributos.
55   virtual void Saida(std::ostream& out = std::cout)  const  = 0;
56
57   /// Sobrecarga operador operator>>
58   friend std::istream& operator>>(std::istream& in,  CFuncao& funcao);
59
60   /// Sobrecarga operador operator<<
61   friend std::ostream& operator<<(std::ostream& out, CFuncao& funcao);
62 };
63
64 #endif
```

Listing 15.2: Arquivo de definição da classe CFuncao - CFuncao.cpp.

```
1 #include "CFuncao.h"
2 using namespace std;
3
4 istream& operator>>(istream& in,  CFuncao& funcao) {
5   in >> funcao.x; in.get();
6   in >> funcao.y;
7   return in;
8 }
9
10 ostream& operator<<(ostream& out, CFuncao& funcao) {
11   out <<  funcao.x << "␣" << funcao.y;
12   return out;
13 }
```

Listing 15.3: Arquivo de declaração da classe CFun1G - CFun1G.h.

```
1 /**********************************************************************
2 CFun1G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFUN1G_H
8 #define CFUN1G_H
9 #include "CFuncao.h"
10
11 /// Representa funcao do primeiro grau, y = c0 + c1 * x;.
12 class CFun1G : public CFuncao
13 {
14  protected:
15   double c0 { 0.0 }; ///< Representa coeficiente linear
16   double c1 { 0.0 }; ///< Representa coeficiente angular
17
18  public:
19   /// Construtor default
20   CFun1G() = default;
21
22   /// Destrutor
23   virtual ~CFun1G() = default;
24
25   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
26   virtual double f (double _x ) ;
27
28   /// Calcula valor da função na posição x, y = c0 + c1 * x;.
```

```
29  inline double f (double _x , double _c0, double _c1) {
30    c0 = _c0; c1 = _c1;
31    return f(_x);
32  }
33
34  /// Seta valor de c0.
35  /// @param _c0 novo valor de c0.
36  inline void C0 ( double _c0 )          {  c0 = _c0; };
37
38  /// Retorna valor de c0.
39  /// @return retorna valor de c0.
40  inline double C0 ( ) const     {  return c0; };
41
42  /// Seta valor de c1.
43  /// @param _c1 novo valor de c1
44  inline void C1 ( double _c1 ) {  c1 = _c1; };
45
46  /// Retorna valor de c1
47  /// @return retorna valor de c1
48  inline double C1 ( ) const     {  return c1; };
49
50  /// Entrada dos valores dos atributos. // ex: funcao->Entrada(cin,cout);
51  virtual void Entrada(std::istream& in = std::cin, std::ostream& out = std::cout) override;
52
53  /// Saída de atributos.
54  virtual void Saida(std::ostream& out = std::cout)  const  override;
55
56  /// Sobrecarga operador operator>>
57  friend std::istream& operator>>(std::istream& in,  CFun1G& funcao);
```

```
58
59   /// Sobrecarga operador operator<<
60   friend std::ostream& operator<<(std::ostream& out, CFun1G& funcao);
61 };
62 #endif
```

Listing 15.4: Arquivo de definição da classe CFun1G - CFun1G.cpp.

```
1 #include <iostream>
2 #include "CFun1G.h"
3 using namespace std;
4
5 void CFun1G::Entrada(istream& in , ostream& out ) {
6   out << "Entre␣com␣dados␣da␣função␣1G␣y␣=␣c0␣+␣c1*x␣␣:\n";
7   out << "Entre␣com␣c0␣:␣";
8   in >> c0;
9   out << "Entre␣com␣c1␣:␣";
10  in >> c1; cin.get();
11 }
12
13 void CFun1G::Saida(ostream& out )  const  {
14   out << "Funcao␣1G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣\n";
15 }
16
17 double CFun1G::f (double _x ) {
18   x = _x;
19   y = c0 + c1 * x;
20   return y;
21 }
```

```
22
23 istream& operator>>(istream& in,  CFun1G& funcao) {
24   in >> funcao.c0; in.get();
25   in >> funcao.c1;
26   return in;
27 }
28
29 ostream& operator<<(ostream& out, CFun1G& funcao) {
30   out << funcao.c0 << "␣" << funcao.c1 << "␣";
31   return out;
32 }
```

Listing 15.5: Arquivo de declaração da classe CFun2G - CFun2G.h.

```
1 /**********************************************************************
2 CFun2G.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFUN2G_H
8 #define CFUN2G_H
9 #include "CFun1G.h"
10
11 /// Representa funcao do segundo grau y = c0 + c1*x + c2*x*x;.
12 class CFun2G : public CFun1G
13 {
14  protected:
15   double c2 {0.0}; ///< Coeficiente quadrático
16
17  public:
18   /// Construtor
19   CFun2G() = default;
20
21   /// Destrutor
22   virtual ~CFun2G() = default;
23
24   /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
25   /// @return double
26   /// @param  _x
27   virtual double f (double _x = 0.0 ) ;
28
```

```
29   /// Calcula valor da função na posição x, y = c0 + c1 * x + c2 *x*x;.
30   /// @return double
31   /// @param  _x, _c0, _c1, _c2
32   inline double f (double _x , double _c0, double _c1, double _c2 )  {
33     c0 = _c0; c1 = _c1; c2 = _c2;
34     return f(_x);
35   }
36
37   /// Seta valor de c2
38   /// @param _c2 novo valor de c2
39   inline void C2 ( double _c2 )          { c2 = _c2; };
40
41   /// Retorna valor de c2
42   /// @return retorna valor de c2.
43   inline double C2 ( ) const     { return c2; };
44
45   /// Entrada dos valores dos atributos. // ex: funcao->Entrada(cin,cout);
46   virtual void Entrada(std::istream& in = std::cin, std::ostream& out = std::cout) override;
47
48   /// Saída de atributos.
49   virtual void Saida(std::ostream& out = std::cout)  const  override;
50
51   /// Sobrecarga operador operator>>
52   friend std::istream& operator>>(std::istream& in,  CFun2G& funcao);
53
54   /// Sobrecarga operador operator<<
55   friend std::ostream& operator<<(std::ostream& out, CFun2G& funcao);
56 };
57 #endif
```

Listing 15.6: Arquivo de definição da classe CFun2G - CFun2G.cpp.

```
1#include <iostream>
2using namespace std;
3
4#include "CFun2G.h"
5
6void CFun2G::Entrada(istream& in, ostream& out ) {
7  out << "Entre␣com␣dados␣da␣funcao␣2G␣y␣=␣c0␣+␣c1*x␣+␣c2*x*x␣␣:\n";
8  out << "Entre␣com␣c0␣:␣";
9  in >> c0;
10  out << "Entre␣com␣c1␣:␣";
11  in >> c1;
12  out << "Agora,␣entre␣com␣c2␣:␣";
13  in >> c2; cin.get();
14}
15
16void CFun2G::Saida(ostream& out )  const  {
17  out << "Funcao␣2G␣y␣=␣" << c0 << "␣+␣"<< c1 << "␣*␣x␣+␣"<< c2 << "␣*␣x␣*␣x␣\n";
18}
19
20// Definicao da funcao f que recebe x e calcula y = f(x)
21double CFun2G::f (double _x ) {
22  x = _x;
23  y = c0 + c1*x + c2*x*x;
24  return y ;
25}
26
27istream& operator>>(istream& in,  CFun2G& funcao) {
28  in >> funcao.c0; in.get();
```

```
29   in >> funcao.c1; in.get();
30   in >> funcao.c2;
31   return in;
32 }
33
34 ostream& operator<<(ostream& out, CFun2G& funcao) {
35     out << funcao.c0 << "␣" << funcao.c1 << "␣"<< funcao.c2 << "␣";
36     return out;
37 }
```

Listing 15.7: Arquivo de declaração da classe CFunExp - CFunExp.h.

```
1 /**********************************************************************
2 CFunExp.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **********************************************************************/
7 #ifndef CFUNEXP_H
8 #define CFUNEXP_H
9 #include "CFun1G.h"
10
11 /// Representa funcao exponencial,  y = c0*exp(c1*x).
12 class CFunExp : public CFun1G
13 {
14  protected:
15   using CFun1G::c0;      ///< Coeficiente c0 da funcao exponencial y = c0*exp(c1*x)
16   using CFun1G::c1;      ///< Coeficiente c1 da funcao exponencial y = c0*exp(c1*x)
17
18  public:
19   /// Construtor
20   CFunExp() = default;
21   /// Destrutor
22   ~CFunExp() = default;
23
24   /// Calcula valor da função exponencial na posição x,  y = a*exp(b*x).
25   virtual double f ( double _x  );
26
27   /// Entrada dos valores dos atributos. // ex: funcao->Entrada(cin,cout);
28   virtual void Entrada(std::istream& in = std::cin, std::ostream& out = std::cout) override;
```

```
29
30  /// Saída de atributos.
31  virtual void Saida(std::ostream& out = std::cout)  const  override;
32
33  /// Sobrecarga operador operator>>
34  friend std::istream& operator>>(std::istream& in,  CFunExp& funcao);
35
36  /// Sobrecarga operador operator<<
37  friend std::ostream& operator<<(std::ostream& out, CFunExp& funcao);
38};
39#endif
```

Listing 15.8: Arquivo de definição da classe CFunExp - CFunExp.cpp.

```
1#include <iostream>
2#include <cmath>     /// math.h no C
3#include "CFunExp.h"
4using namespace std;
5
6void CFunExp::Entrada(istream& in , ostream& out ) {
7  out << "Entre␣com␣dados␣da␣funcao␣y␣=␣c0␣*␣exp(␣c1␣*␣x␣);\n";
8  out << "Entre␣com␣c0␣:␣";
9  in >> c0;
10  out << "Entre␣com␣c1␣:␣";
11  in >> c1; cin.get();
12}
13
14void CFunExp::Saida(ostream& out )  const  {
15  out << "Funcao␣Exp␣y␣=␣" << c0 << "␣*␣e^("<< c1 << "␣*␣x)␣\n";
```

```
16}
17
18double CFunExp::f ( double _x  )  {
19 x = _x;
20 return y = c0 * exp( c1 * x );
21}
22
23istream& operator>>(istream& in,  CFunExp& funcao) {
24  in >> funcao.c0; in.get();
25  in >> funcao.c1;
26  return in;
27}
28
29ostream& operator<<(ostream& out, CFunExp& funcao) {
30  out << funcao.c0 << "␣" << funcao.c1 << "␣";
31  return out;
32}
```

Listing 15.9: Arquivo de declaração da classe CIntegral - CIntegral.h.

```
/**************************************************************************
CIntegral.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
**************************************************************************/
#ifndef CINTEGRAL_H
#define CINTEGRAL_H
#include <memory>
#include "CFuncao.h"

/// Representa método numérico para cálculo área funções.
class CIntegral
{
 protected:
  double limInf { 0.0 };  ///< Limite Inferior do intervalo de integração
  double limSup { 0.0 };  ///< Limite Inferior do intervalo de integração
  double dx     { 0.0 };  ///< Intervalo dx
  int    numInt {  0  };  ///< Numero intervalos, numero pontos
  double area   { 0.0 };  ///< Valor da área calculada

 public:
  /// Construtor
  CIntegral()   = default;

  /// Construtor sobrecarregado
  CIntegral(double _limInf, double _limSup, long int _np) :
                        limInf { _limInf },
```

```
29                      limSup { _limSup }
30                      { NumInt(_np); };
31
32  /// Construtor sobrecarregado, cópia de CIntegral
33  CIntegral(const CIntegral& integral) :
34                      limInf { integral.LimInf() },
35                      limSup { integral.LimSup() },
36                      numInt { integral.NumInt() },
37                      dx     { integral.Dx() }
38                      {};
39  // CIntegral(const CIntegral& integral)  = default;
40
41  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
42  virtual  double Area ( std::unique_ptr<CFuncao>& funcao ) = 0;
43
44  /// Retorna área da função (já foi calculada!).
45  virtual  double Area ( ) { return area; } ;
46
47  /// Calcula área da função no intervalo limInf->limSup considerando numInt.
48  inline double operator()( std::unique_ptr<CFuncao>& funcao ) { return Area( funcao ); }
49
50  /// Seta valor de limInf
51  inline void LimInf ( double _limInf  )        { limInf = _limInf ; };
52
53  /// Retorna valor de limInf
54  /// @return retorna valor de limInf
55  inline double LimInf ( )      const           { return limInf; };
56
57  /// Seta valor de limSup
```

```
58  /// @param _limSup Seta novo valor de  limSup
59  inline void LimSup ( double _limSup )          { limSup = _limSup; };
60
61  /// Retorna valor de limSup
62  /// @return retorna valor de limSup
63  inline double LimSup ( )       const           { return limSup; } ;
64
65  /// Seta valor de dx
66  /// @param _dx o novo valor de  dx
67  inline void Dx ( double _dx )                  { dx = _dx; numInt = (limSup - limInf) / dx;};
68
69  /// Retorna valor de dx
70  ///* @return the value of dx
71  inline double Dx ( ) const                      { return dx;};
72
73  /// Seta valor de numInt
74  /// @param _numInt novo valor de numInt
75  inline void NumInt ( int _numInt )              { numInt = _numInt;
76                                                  dx = (limSup - limInf) / numInt;
77                                                  };
78   /// Retorna valor de numInt
79  inline int NumInt ( )          const           { return numInt; } ;
80
81  /// Entrada dos valores dos atributos. // ex: funcao->Entrada(cin,cout);
82  virtual void Entrada(std::istream& in = std::cin, std::ostream& out = std::cout) ;
83
84  /// Saída de atributos.
85  virtual void Saida(std::ostream& out = std::cout)  const ;
86
```

```
87   /// Sobrecarga operador operator>>
88   friend std::istream& operator>>(std::istream& in,  CIntegral& integral);
89
90   /// Sobrecarga operador operator<<
91   friend std::ostream& operator<<(std::ostream& out, CIntegral& integral);
92 };
93 #endif
```

Listing 15.10: Arquivo de definição da classe CIntegral - CIntegral.cpp.

```
1 #include<iostream>
2 using namespace std;
3
4 #include "CIntegral.h"
5
6 void CIntegral::Entrada(istream& in, ostream& out ) {
7   out << "Entre com dados do metodo integracao:\n";
8   out << "Entre com Limite Inferior : ";
9   in >> limInf;
10  out << "Entre com limite Superior : ";
11  in >> limSup;
12  out << "Entre com numero intervalos : ";
13  in >> numInt; cin.get();
14
15  // Como numInt foi modificado, atualizo valor de dx
16  dx = (limSup - limInf) / numInt;
17 }
18
19 void CIntegral::Saida(ostream& out ) const {
```

```
20   out  << "\nLimite␣Inferior␣␣␣:␣" << limInf
21        << "\nLimite␣Superior␣␣␣:␣" << limSup
22        << "\nNumero␣intervalos:␣" << numInt << endl;
23 }
24
25 istream& operator>>(istream& in, CIntegral& integral) {
26   in >> integral.limInf
27      >> integral.limSup
28      >> integral.numInt;
29   // Como numInt foi modificado, atualizo valor de dx
30   integral.dx = (integral.limSup - integral.limInf) /(double) integral.numInt;
31   return in;
32 }
33
34 ostream& operator<<(ostream& out, CIntegral& integral) {
35   out   << integral.limInf << "␣"
36         << integral.limSup << "␣"
37         << integral.numInt << "␣" ;
38   return out;
39 }
```

Listing 15.11: Arquivo de declaração da classe CIntTrapezio - CIntTrapezio.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CINTTRAPEZIO_H
8 #define CINTTRAPEZIO_H
9 #include <memory>
10 #include "CIntegral.h"
11
12 /**
13  * Representa método numérico trapésio para cálculo área funções.
14  * @class CIntTrapezio
15  */
16 class CIntTrapezio : public CIntegral
17 {
18  public:
19   /// Construtor
20   CIntTrapezio() = default;
21
22   /// Construtor sobrecarregado
23   CIntTrapezio(double _limInf, double _limSup, long int _np) : CIntegral(_limInf, _limSup, _np) {}
24
25   /// Desstrutor
26   ~CIntTrapezio() = default;
27
28   /// Construtor sobrecarregado, cópia de CIntegral
```

```
29  // Possibilita construir um CIntTrapezio a partir de um CIntegral
30  CIntTrapezio(const CIntegral& integral) : CIntegral(integral) {};
31  /**
32   * Calcula área da função no intervalo limInf->limSup considerando numInt.
33   * @return double
34   * @param  funcao
35   */
36 virtual double Area ( std::unique_ptr<CFuncao>& funcao )  override;
37
38 virtual double Area ( ) { return area; } ;
39};
40#endif
```

Listing 15.12: Arquivo de definição da classe CIntTrapezio - CIntTrapezio.cpp.

```
1#include "CIntTrapezio.h"
2using namespace std;
3
4double CIntTrapezio::Area ( std::unique_ptr<CFuncao>& funcao ) {
5 area =  funcao->f( limInf ) * 0.5;
6 area += funcao->f( limSup ) * 0.5;     // area = area + funcao->f(limSup);
7
8 double x = limInf;
9 for( int i = 1; i < numInt ; i++ ) {
10  x += dx;
11  area += (*funcao)( x ); // *funcao retorna o objeto função e depois acessa operator(double x)
12  }
13 area *= dx ;
14 return area;
```

```
15 }
```

Listing 15.13: Arquivo de declaração da classe CIntSimpson - CIntSimpson.h.

```
1 /**************************************************************************
2 CIntTrapezio.h - Copyright André D. Bueno
3 Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
4 Implementação de uma pequena biblioteca de classes para funções e
5 cálculo de integral numérica.
6 **************************************************************************/
7 #ifndef CINTSIMPSON_H
8 #define CINTSIMPSON_H
9 #include "CIntegral.h"
10
11 /// Representa método numérico simpson para cálculo área funções.
12 class CIntSimpson : public CIntegral
13 {
14  public:
15   /// Construtor
16   CIntSimpson() = default;
17
18   /// Construtor sobrecarregado
19   CIntSimpson(double _limInf, double _limSup, long int _np) : CIntegral(_limInf, _limSup, _np) {}
20
21   /// Construtor de cópia de CIntegral
22   // Possibilita construir um CIntSimpson a partir de um CIntTrapezio
23   CIntSimpson(const CIntegral& integral) : CIntegral(integral) {};
24
25   /// Calcula área da função no intervalo limInf->limSup considerando numInt.
26  virtual double Area ( std::unique_ptr<CFuncao>& funcao )  override;
27
28  virtual double Area ( ) { return area; } ;
```

```
29};
30#endif
```

Listing 15.14: Arquivo de definição da classe CIntSimpson - CIntSimpson.cpp.

```
1#include "CIntSimpson.h"
2using namespace std;
3
4double CIntSimpson::Area ( std::unique_ptr<CFuncao>& funcao ) {
5 area = 0.5 * ( funcao->f(limInf)                          // ponto 0       // [0]
6               + funcao->f(limSup) ) ;                    // ponto n       // [4]
7
8 for ( int i = 1; i < (numInt - 2); i += 2 )
9      area += 2.0 * funcao->f (limInf + i * dx);          // ponto 1,...,n-3 // [1]
10 for ( int i = 1; i < (numInt - 2); i += 2 )
11      area += funcao->f (limInf + (i + 1) * dx);         // ponto 2,...,n-2 // [2]
12
13 area += 2.0 * funcao->f (limSup - dx);                  // ponto n-1       // [3]
14
15 area *= 2.0 * dx / 3.0;
16
17 return area;
18}
```

Listing 15.15: Arquivo de declaração da classe CSimulador - CSimulador.h.

```
/*************************************************************************
CSimulador.h - Copyright André D. Bueno
Exemplo didático para ensino de C++ para alunos do LENEP/CCT/UENF.
Implementação de uma pequena biblioteca de classes para funções e
cálculo de integral numérica.
*************************************************************************/
#ifndef CSIMULADOR_H
#define CSIMULADOR_H

class CSimulador
{
public:
  /// Construtor da classe
  CSimulador()  = default;

  /// Destrutor da classe
  ~CSimulador() = default;

  /// Executa a simulação.
  int ExecutarSimulacao();
};
#endif
```

Listing 15.16: Arquivo de definição da classe CSimulador - CSimulador.cpp.

```
#include <cstdio>
#include <iostream>
#include <fstream>
```

```
4#include <iomanip>          // setw setprecision
5#include <vector>
6#include <algorithm>
7#include <functional>
8//#include <tuple>
9#include <memory>           // unique_ptr e make_unique
10
11#include <thread>           // Uso de threads!
12
13
14#include "CSimulador.h"
15#include "CFun1G.h"
16#include "CFun2G.h"
17#include "CFunExp.h"
18#include "CIntTrapezio.h"
19#include "CIntSimpson.h"
20
21using namespace std;       // Traz para nosso escopo cin, cout,....etc
22
23/// Executa a simulação.
24int CSimulador::ExecutarSimulacao() {
25  double limInf;
26  double limSup;
27  /// ----------->Versão 1: Usando unique_ptr, make_unique, Entrada(cin, cout), auto
28  {
29  cout << "\nVersão 1: Usando unique_ptr, make_unique, Entrada(cin, cout), auto:\n";
30  // unique_ptr cria um ponteiro para CFuncao
31  // make_unique<CFunExp>() cria objeto do tipo CFunExp que é apontado pelo ponteiro funcao.
32  unique_ptr<CFuncao>  funcao = make_unique<CFunExp>();
```

```
33
34  cout << "\n-->Entre␣com␣os␣coeficientes␣da␣função␣:\n";
35  funcao->Entrada(cin, cout); // funcao->Entrada(); usa por padrão cin e cout
36
37  // Cria ponteiros para objetos integração, usa make_unique para retornar um unique_ptr
38  cout << "Entre␣com␣Limite␣Inferior:␣";
39  cin >> limInf;
40  cout << "Entre␣com␣Limite␣Superior:␣";
41  cin >> limSup;
42  // make_unique cria objeto e retorna ponteiro para ele (o mesmo que new)
43  auto trapezio = make_unique<CIntTrapezio>(limInf, limSup, 100);
44  auto  simpson = make_unique<CIntSimpson> (limInf, limSup, 100);
45
46  for(long int ni = 10; ni < 1000000; ni *=10) {
47    trapezio->NumInt(ni);
48    simpson->NumInt(ni);
49    cout << "\n|NumInt␣=␣" << setw(7) << ni
50         << "␣|␣Area␣Trapezio␣=␣" << setprecision(20) << setw(18) << trapezio->Area( funcao )
51         << "␣|␣Area␣Simpson␣␣=␣" << setprecision(20) << setw(18) << simpson ->Area( funcao )
52         << setw(2) <<"␣|";
53  }
54  cout << "\n\n";
55  /// -----------> Uso de saída para disco
56  cout << "\nTestando␣uso␣de␣arquivos␣de␣disco:\n";
57  ofstream fout;
58  fout.open("funcao.dat");    funcao->Saida(fout);    fout.close();
59  fout.open("trapezio.dat");  trapezio->Saida(fout);  fout.close();
60  // Uso de system
61  cout << "\nMostrando␣o␣arquivo␣de␣disco␣funcao.dat:\n";
```

```
62  system("cat␣funcao.dat");
63  cout << "\nMostrando␣o␣arquivo␣de␣disco␣trapezio.dat:\n";
64  system("cat␣trapezio.dat");
65  }
66
67  /// ----------->Versão 2: Usando threads!
68  cout << "\nVersão␣1:␣Usando␣threads!:\n";
69  vector< CFuncao* >  vfuncao { new CFun1G , new CFun2G , new CFunExp };
70  cout << "\n\nEntre␣com␣os␣coeficientes␣das␣diferentes␣funções:\n";
71  for_each(vfuncao.begin(), vfuncao.end(), [](CFuncao* f) { cout << "\n"; f->Entrada(); f->Saida(); });
72
73  // Para cada função
74  int i = 0;
75  cout << "Número␣máximo␣de␣threads␣suportadas,␣thread::hardware_concurrency()␣=␣" << thread::hardware_concurrency
       () << endl;
76  for ( auto funcao : vfuncao ) {
77    CIntTrapezio trapezio(limInf, limSup, 100);
78    CIntSimpson  simpson (limInf, limSup, 100);
79
80    // Cria thread que recebe objeto CIntTrapezio e o parametro funcao.
81    thread  th_trapezio( trapezio, funcao);
82    // Cria thread que recebe objeto CIntSimpson e o parametro funcao.
83    thread  th_simpson ( simpson, funcao);
84
85    // Informações das threads
86    cout    << "\nInformações␣das␣threads"
87            << "\nthis_thread::get_id()␣=␣" << this_thread::get_id()
88            << "\ntrapezio.get_id()␣=␣"     << th_trapezio.get_id()
89            << "\nsimpson.get_id()␣=␣"      << th_simpson.get_id() << endl;
```

```
90
91      cout << "\n\n------>Função␣:␣" << i++ << endl;
92      funcao->Saida();                                        // no processador 1
93      // Quando processador 2 encerrar cálculo área, mostra resultado
94      th_trapezio.join(); cout << "\nArea␣Trapezio␣=␣" << trapezio.Area() ;
95      // Quando processador 3 encerrar cálculo área, mostra resultado
96      th_simpson.join();  cout << "\nArea␣Simpson␣␣=␣" << simpson.Area() ;
97    }
98    cout << endl;
99
100   return 0;
101 }
102 /*
103   cout << "\nVersão 1: Usando threads!:\n";
104   vector< CFuncao* >  vfuncao { new CFun1G , new CFun2G , new CFunExp };
105   cout << "\n\nEntre com os coeficientes das diferentes funções:\n";
106   for_each(vfuncao.begin(), vfuncao.end(), [](CFuncao* f) { cout << "\n"; f->Entrada(); f->Saida(); });
107
108   vector< CIntegral* > vintegral {new CIntTrapezio , new CIntSimpson };
109   cout << "Entre com os coeficientes dos métodos de integração:\n";
110   for_each(vintegral.begin(), vintegral.end(), [](CIntegral* i) { i->Entrada();} );
111
112   // Para cada função
113   int i = 0;
114   for ( auto funcao : vfuncao ) {
115     // Cria thread que recebe objeto função vintegral[0 ou 1] e o parametro funcao.
116     thread  trapezio( *vintegral[0], funcao);
117     thread  simpson ( *vintegral[1], funcao);
118
```

```
119     // Informações das threads
120     cout    << "\nInformações das threads"
121             << "\nthis_thread::get_id() = " << this_thread::get_id()
122             << "\ntrapezio.get_id() = "     << trapezio.get_id()
123             << "\nsimpson.get_id() = "      << trapezio.get_id() << endl;
124
125     cout << "\n\n------>Função : " << i++ << endl;
126     funcao->Saida();                                    // no processador 1
127     // Quando processador 2 encerrar cálculo área, mostra resultado
128     trapezio.join(); cout << "\nArea Trapezio = " << vintegral[0]->Area() ;
129     // Quando processador 3 encerrar cálculo área, mostra resultado
130     simpson.join();  cout << "\nArea Simpson  = " << vintegral[1]->Area() ;
131   }
132 */
133 /*
134    vector< thread > t;   // vetor que armazena as threads
135   for ( auto funcao: vfuncao )   {
136     t.push_back( []() { vintegral[0]->Area(funcao) } );  // trapezio da função (usa função lambda)
137     t.push_back( []() { vintegral[1]->Area(funcao) } );  // simpson  da função
138   }
139
140   for_each(t.begin(), t.end(), [](thread& th) { th.join() } ); // espera thread encerrar
141   //for_each( t.begin(), t.end(), std::men_fun_ref(&std::thread::join) );
142
143   for ( int i = 0; i < vfuncao.size() ; i++ ) {
144
145     cout << "\n\n------>Função : " << i << endl;
146     funcao[i]->Saida();
147     cout << "\nArea Trapezio = " << vintegral[0]->Area() ;
```

```
148     cout << "\nArea Simpson  = " << vintegral[1]->Area() ;
149   }
150
151 */
152
153 /*int ReturnFrmThread() {
154     return 100;
155 }
156 int main() {
157     std::future<int> GetRetVal= std::async(ReturnFrmThread);  // Execution of ReturnFrmThread starts
158     int answer = GetAnAnswer.get(); // gets value as 100;
159     // Waits until ReturnFrmThread has finished
160 */
```

Listing 15.17: Programa para cálculo da integral do trapézio: Usa hierarquia classes CFuncao e CIntegral.

```
/** Copyright André D. Bueno
 * Exemplo didático: Ensino de C++ para alunos do LENEP/CCT/UENF.
 * Implementação de uma pequena biblioteca de classes para funções e
 * cálculo de integral numérica.
 *
 * Características da versão:
 * - Acrescenta uso algoritmo genérico;
 * - Novos conceitos de C++11 como unique_ptr e make_ptr;
 * - Função Area recebe ponteiro do tipo: unique_ptr<CFuncao>;
 * - Função Entrada e Saida recebem stream;
 * - Em ExecutarSimulacao foi incluida saída para disco usando Saida(fout);
 *
 * Para compilar:
 * g++ -std=c++11 *.cpp -o integral
*/
#include "CSimulador.h"

/// Função principal do programa
int main(int argc, char **argv) {
  CSimulador simulador;
  simulador.ExecutarSimulacao();
  return 0;
}
```

Capítulo 16

Usando a Classe `CGnuplot` para Fazer Gráficos

16.1 O que é o `gnuplot`?

- Programa multiplataforma (Windows, GNU/Linux, Mac OS X) utilizado por cientistas e pesquisadores para montar gráficos avancados em 2D e 3D.

16.2 Onde obter o `gnuplot`?

- http://sourceforge.net/projects/gnuplot/files/

16.3 Onde obter maiores informações sobre o `gnuplot`?

- Dê uma olhada no diretório `CGnuplot`, você vai encontrar uma aula em pdf sobre o gnuplot.
- Outras informações do `gnuplot` podem ser obtidas em http://www.gnuplot.info/

16.4 O que é a classe `CGnuplot`?

- A classe `CGnuplot` fornece acesso direto ao programa gnuplot.
- Com a classe `CGnuplot` você pode montar um programa em C++ que faz gráficos avançados, fazendo chamadas diretas ao gnuplot.
- Vantagens:
  - Você não precisa aprender os comandos internos do gnuplot. Somente os métodos da classe `CGnuplot`.
  - Outra vantagem em relação ao uso de <pstream.h>, é que o terminal fica "liberado", assim, podemos abrir vários gráficos do gnuplot ao mesmo tempo.

A Figura 16.1 mostra um diagrama UML simplificado da classe `CGnuplot`.

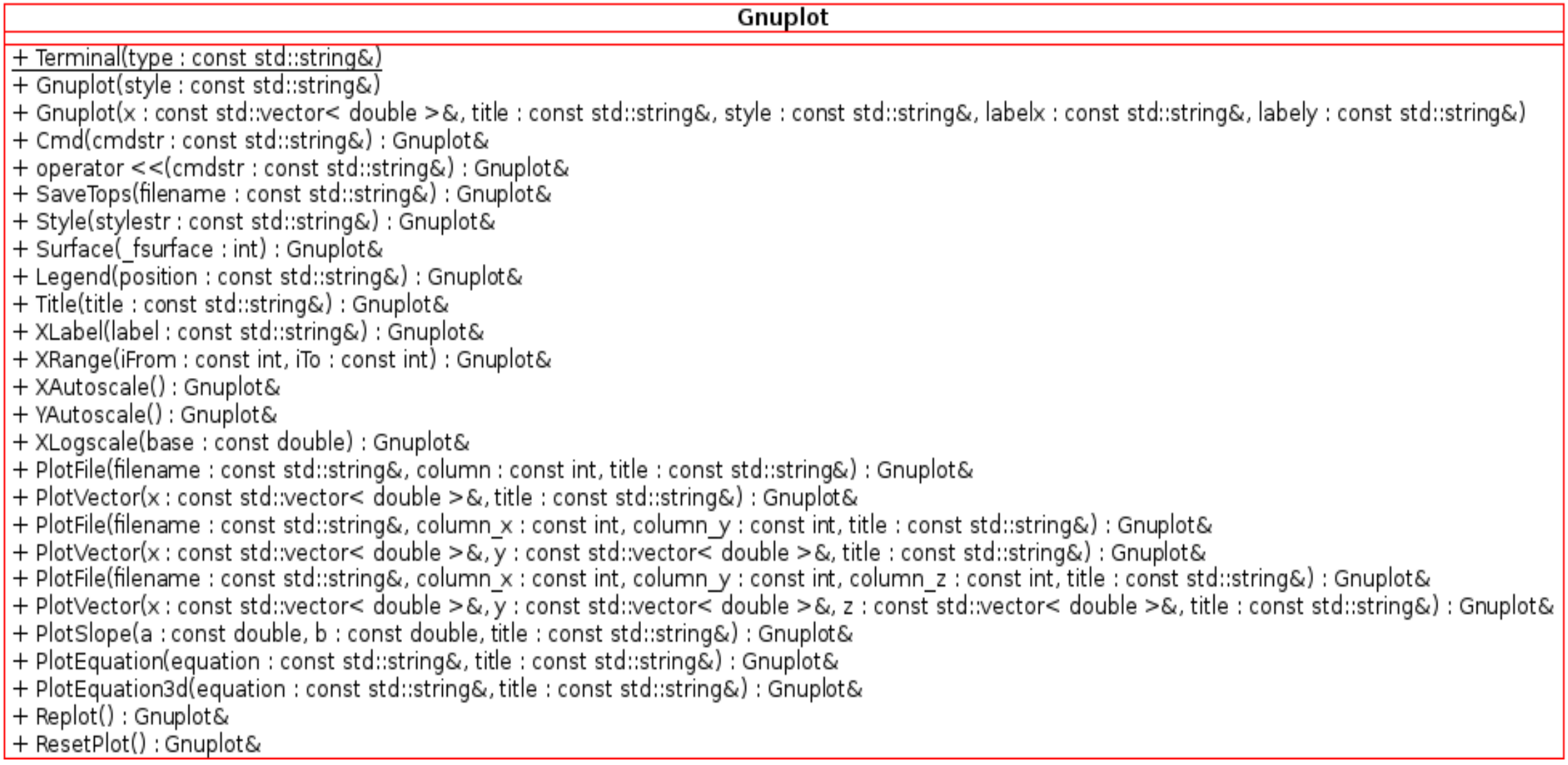


Figura 16.1: Diagrama da Classe - `CGnuplot`

A listagem 16.1 mostra uma referência para uso da classe `CGnuplot`.

Listing 16.1: Uma referência para uso da classe CGnuplot.

```
class  Gnuplot
{
public:
  static void   Terminal (const std::string & type);
  //-------------------------------------------------------------------------------Construtores
  /// @brief Construtor, seta o estilo do grafico na construcao.
  Gnuplot (const std::string & style = "points");
  /// @brief Construtor, plota um grafico a partir dde um vector, diretamente na construcao.
  Gnuplot (const std::vector < double >&x,const std::string & title = "",const std::string & style = "points",
           const std::string & labelx = "x",const std::string & labely = "y");
  /// @brief Construtor, plota um grafico do tipo x_y a partir de vetores, diretamente na construcao.
  Gnuplot (const std::vector < double >&x,const std::vector < double >&y,const std::string & title = "",
           const std::string & style = "points",const std::string & labelx = "x",const std::string & labely = "y");
  /// @brief Construtor, plota um grafico de x_y_z a partir de vetores, diretamente na construcao.
  Gnuplot (const std::vector < double >&x,const std::vector < double >&y,const std::vector < double >&z,
           const std::string & title = "",const std::string & style = "points",const std::string & labelx = "x",
           const std::string & labely = "y",const std::string & labelz = "z");
  /// @brief Destrutor, necessario para deletar arquivos temporarios.
  ~Gnuplot ();
  //-------------------------------------------------------------------------------------------
  /// @brief Envia comando para o gnuplot.
  Gnuplot & cmd (const std::string & cmdstr);
  /// @brief Sobrecarga operador <<, funciona como Comando.
  Gnuplot & operator<< (const std::string & cmdstr);
  //------------------------------------------------------------------------------- set e unset
  /// @brief  Seta estilos de linhas (em alguns casos sao necessarias informacoes adicionais).
```

```
27  /// lines, points, linespoints, impulses, dots, steps, fsteps, histeps,
28  /// boxes, histograms, filledcurves
29  Gnuplot & Style (const std::string & stylestr = "points");
30  /// @brief  Ativa suavizacao.
31  /// Argumentos para interpolacoes e aproximacoes.
32  /// csplines, bezier, acsplines (para dados com valor > 0), sbezier, unique,
33  /// frequency (funciona somente com plot_x, plot_xy, plotfile_x,
34  /// plotfile_xy (se a suavizacao esta ativa, set_style nao tem efeito na plotagem dos graficos)
35  Gnuplot & Smooth (const std::string & stylestr = "csplines");
36  /// @brief Escala o tamanho do ponto usado na plotagem.
37  Gnuplot & PointSize (const double pointsize = 1.0);
38  /// @brief Ativa/Desativa o grid (padrao = desativado).
39  Gnuplot & Grid (bool _fgrid = 1);
40  /// @brief Seta taxa de amostragem das funcoes, ou dos dados de interpolacao.
41  Gnuplot & Samples (const int samples = 100);
42  /// @brief Seta densidade de isolinhas para plotagem de funcoes como superficies (para plotagen 3d).
43  Gnuplot & IsoSamples (const int isolines = 10);
44  /// @brief Ativa/Desativa remocao de linhas ocultas na plotagem de superficies (para plotagen 3d).
45  Gnuplot & Hidden3d (bool _fhidden3d = 1);
46  /// @brief Ativa/Desativa desenho do contorno em superficies (para plotagen 3d).
47  /// @param base, surface, both.
48  Gnuplot & Contour (const std::string & position = "base");
49  /// @brief  Ativa/Desativa a visualizacao da superficie (para plotagen 3d).
50  Gnuplot & Surface (int _fsurface = 1);  /// @brief Ativa/Desativa a legenda (a legenda é setada por padrao).
51  Gnuplot & Legend (const std::string & position = "default");
52  /// @brief Ativa/Desativa a legenda (a legenda é setada por padrao).
53  Gnuplot & Legend (int _flegend);
54  /// @brief Ativa/Desativa o titulo da secao do gnuplot.
55  Gnuplot & Title (const std::string & title = "");  /// @brief Seta o rotulo (nome) do eixo y.
```

```
56  Gnuplot & YLabel (const std::string & label = "y");  /// @brief Seta o rotulo (nome) do eixo x.
57  Gnuplot & XLabel (const std::string & label = "x");
58  /// @brief Seta o rotulo (nome) do eixo z.
59  Gnuplot & ZLabel (const std::string & label = "z");
60  /// @brief Seta intervalo do eixo x.
61  Gnuplot & XRange (const int iFrom, const int iTo);
62  /// @brief Seta intervalo do eixo y.
63  Gnuplot & YRange (const int iFrom, const int iTo);  /// @brief Seta intervalo do eixo z.
64  Gnuplot & ZRange (const int iFrom, const int iTo);
65  /// @brief Seta escalonamento automatico do eixo x (default).
66  Gnuplot & XAutoscale ();
67  /// @brief Seta escalonamento automatico do eixo y (default).
68  Gnuplot & YAutoscale ();
69  /// @brief Seta escalonamento automatico do eixo z (default).
70  Gnuplot & ZAutoscale ();
71  /// @brief Ativa escala logaritma do eixo x (logscale nao e setado por default).
72  Gnuplot & XLogscale (const double base = 10);
73  /// @brief Ativa/Desativa escala logaritma do eixo x (logscale nao e setado por default).
74  Gnuplot & XLogscale (bool _fxlogscale);  /// @brief Ativa escala logaritma do eixo y (logscale nao e setado por
      default).
75  Gnuplot & YLogscale (const double base = 10);
76  /// @brief Ativa/Desativa escala logaritma do eixo y (logscale nao e setado por default).
77  Gnuplot & YLogscale (bool _fylogscale);  /// @brief Ativa escala logaritma do eixo y (logscale nao e setado por
      default).
78  Gnuplot & ZLogscale (const double base = 10);  /// @brief Ativa/Desativa escala logaritma do eixo y (logscale nao
       e setado por default).
79  Gnuplot & ZLogscale (bool _fzlogscale);
80  /// @brief Seta intervalo da palette (autoscale por padrao).
81  Gnuplot & CBRange (const int iFrom, const int iTo);
```

```
82  //-------------------------------------------------------------------------
83  /// @brief Plota dados de um arquivo de disco.
84  /// @brief Plota dados de um arquivo de disco.
85  Gnuplot & PlotFile (const std::string & filename,
86                      const int column = 1, const std::string & title = "");
87    /// @brief Plota dados de um vector.
88  Gnuplot & PlotVector (const std::vector < double >&x,
89                        const std::string & title = "");
90  /// @brief Plota pares x,y a partir de um arquivo de disco.
91  Gnuplot & PlotFile (const std::string & filename,
92                      const int column_x = 1,
93                      const int column_y = 2, const std::string & title = "");
94  /// @brief Plota pares x,y a partir de vetores.
95  Gnuplot & PlotVector (const std::vector < double >&x,
96                        const std::vector < double >&y,
97                        const std::string & title = "");
98  /// @brief Plota pares x,y com barra de erro dy a partir de um arquivo.
99  Gnuplot & plotfile_xy_err (const std::string & filename,
100                             const int column_x = 1,
101                             const int column_y = 2,
102                             const int column_dy =
103                             3, const std::string & title = "");
104 /// @brief Plota pares x,y com barra de erro dy a partir de um arquivo.
105 Gnuplot & PlotFileXYErrorBar (const std::string & filename,
106                               const int column_x = 1,
107                               const int column_y = 2,
108                               const int column_dy =
109                               3, const std::string & title = "");
110 /// @brief Plota pares x,y com barra de erro dy a partir de vetores.
```

```
111  Gnuplot & PlotVectorXYErrorBar (const std::vector < double >&x,
112                                  const std::vector < double >&y,
113                                  const std::vector < double >&dy,
114                                  const std::string & title = "");
115  /// @brief  Plota valores de x,y,z a partir de um arquivo de disco.
116  Gnuplot & PlotFile (const std::string & filename,
117                      const int column_x = 1,
118                      const int column_y = 2,
119                      const int column_z = 3, const std::string & title = "");
120  /// @brief  Plota valores de x,y,z a partir de vetores.
121  Gnuplot & PlotVector (const std::vector < double >&x,
122                        const std::vector < double >&y,
123                        const std::vector < double >&z,
124                        const std::string & title = "");
125  /// @brief Plota uma equacao da forma y = ax + b, voce fornece os coeficientes a e b.
126  Gnuplot & PlotSlope (const double a, const double b,
127                       const std::string & title = "");
128  /// @brief Plota uma  equacao fornecida como uma std::string y=f(x).
129  /// Escrever somente a funcao f(x) e nao y=
130  /// A variavel independente deve ser x.
131  /// Exemplo: gnuplot->PlotEquation(CFuncao& obj);
132  Gnuplot & PlotEquation (const std::string & equation,
133                          const std::string & title = "");
134  /// @brief Plota uma equacao fornecida na forma de uma std::string z=f(x,y).
135  /// Escrever somente a funcao f(x,y) e nao z=, as vaiaveis independentes sao x e y.
136  // gnuplot->PlotEquation3d(CPolinomio());
137  Gnuplot & PlotEquation3d (const std::string & equation,
138                            const std::string & title = "");
139  /// @brief Plota uma imagem.
```

```
140   Gnuplot & PlotImage (const unsigned char *ucPicBuf,
141                        const int iWidth, const int iHeight,
142                        const std::string & title = "");
143   //-------------------------------------------------------------------------
144   // Repete o ultimo comando de plotagem, seja plot (2D) ou splot (3D)
145   // Usado para visualizar plotagens, após mudar algumas opcoes de plotagem
146   // ou quando gerando o mesmo grafico para diferentes dispositivos (showonscreen, savetops)
147   Gnuplot & Replot ();
148   // Reseta uma sessao do gnuplot (próxima plotagem apaga definicoes previas)
149   Gnuplot & ResetPlot ();
150   // Reseta uma sessao do gnuplot (próxima plotagem apaga definicoes previas)
151   Gnuplot & Reset ();  // Reseta uma sessao do gnuplot e seta todas as variaveis para o default
152   Gnuplot & ResetAll ();
153   // Verifica se a sessao esta valida
154   bool  is_valid ();
155   // Verifica se a sessao esta valida
156   bool  IsValid ();
157 };
158 typedef Gnuplot   CGnuplot;
159 #endif
```

- Veja a seguir um exemplo simples de uso da classe `CGnuplot`.

```
int main(int argc, char* argv[]) {
CGnuplot g2d = CGnuplot("lines");// Construtor, cria o gráfico
g2d.Title("Titulo do grafico");  // Titulo do grafico
g2d.XLabel("rotulo eixo x");     // Rotulo eixo x
g2d.YLabel("rotulo eixo y");     // Rotulo eixo y
g2d.XRange(-10,10);              // Seta intervalo do eixo x.
g2d.PlotEquation( "x*x*sin(x)"); // Plota uma determinada equação
std::cout << "Pressione enter para encerrar." << std::endl;
std::cin.get(); return 0; }
```

- Para compilar:

```
g++ CGnuplot.cpp exemploBemSimples.cpp -o exemploBemSimples
```

- Para rodar: `./exemploBemSimples`

16.5 Quais os arquivos do diretório `CGnuplot`?

- O diretório `CGnuplot` contem os seguintes arquivos:
  - `CGnuplot.h`
    - Declara e define uma classe de acesso ao programa `gnuplot`.
  - `CGnuplot.cpp`
    - Define métodos da classe `CGnuplot`. leiame.txt Este arquivo.
  - `CGnuplot.teste.min.cpp`
    - Um programa pequeno que usa a classe `CGnuplot`.
  - `CGnuplot.teste.cpp`
    - Um outro programa que usa a classe `CGnuplot`.

16.6 Instruções para uso do pacote CGnuplot.tar.gz

- O pacote CGnuplot.tar.gz é um pacote utilizado em parte da disciplina de C++ para fazer gráficos.
- Para iniciar, descompacte o arquivo CGnuplot.tar.gz. Abra um terminal, vá para o diretório onde armazenou o arquivo CGnuplot.tar.gz e digite:

```
tar -xvzf CGnuplot.tar.gz
```

- Vá para o diretório criado:

```
cd CGnuplot
```

- Compile o programa de teste usando o makefile fornecido

```
make
```

ou compile o programa de teste minimo

```
make CGnuplot.testemin
```

ou compile diretamente os arquivos cpp (gera a.out)

```
g++ CGnuplot.cpp CGnuplot.teste.cpp
```

- Rode o programa de teste:

```
./CGnuplot.teste
```

ou

```
./CGnuplot.testemin
```

ou

```
./a.out
```

  - O programa vai gerar um gráfico, para gerar outros modelos de gráficos basta ir pressionando enter.
  - Observe que o programa de teste esta gerando diferentes tipos de gráficos.

- Agora que já compilou, rodou, e viu os diferentes tipos de gráficos (2D e 3D no exemplo completo), esta na hora de entender o código.
- Use seu navegador para abrir o arquivo CGnuplot/doc/html/index.html leia as descrições dos métodos.
- Abra o arquivo CGnuplot.testemin.cpp e veja como o programa de teste esta criando um objeto da classe `CGnuplot`, e acessando seus métodos para plotar e setar parâmetros dos gráficos do gnuplot.
- Modifique o arquivo CGnuplot.testemin.cpp, compile novamente:

  ```
  make CGnuplot.testemin
  ```

  - rode

    ```
    ./CGnuplot.testemin
    ```

  - e veja como ficou a saída após suas modificações.
- Abra o arquivo com o teste completo, arquivo CGnuplot.teste.cpp e a seguir rode o programa ./CGnuplot.teste.

- Leia o código em CGnuplot.teste.cpp e veja como fica a saída correspondente. Pressione enter para gerar o novo gráfico e verifique o código que gerou o gráfico. Desta forma você vê, passo a passo, o código e a saída gerada.

- Pronto, você já pode utilizar a classe `CGnuplot` em seus programas.

16.7 Exemplo de uso da classe `CGnuplot`

Vimos na Figura 16.1 um diagrama UML simplificado da classe `CGnuplot` e na listagem 16.1 uma referência para uso da classe `CGnuplot`.

A listagem 16.2 mostra um exemplo simplificado de uso da classe `CGnuplot`.

Listing 16.2: Um exemplo simplificado de uso da classe CGnuplot.

```
1// Programa de teste da classe CGnuplot.
2#include <iostream>
3
4#include "CGnuplot.h"
5
6using namespace std;       // Usando espaco de nomes da std
7
8void wait_for_key ()
9{
10  cout << endl << "Pressione␣ENTER␣para␣continuar..." << endl;
11  std::cin.clear();                                   // Zera estado de cin
12  std::cin.ignore(std::cin.rdbuf()->in_avail());// Ignora
13  std::cin.get();                                     // Espera o pressionamento do enter
14  return;
15}
16
17int main(int argc, char* argv[]) {
18 cout    << "\n=============================================================================="
19         << "\n==================␣Programa␣de␣teste␣da␣classe␣CGnuplot======================="
20         << "\n==============================================================================";
```

```
21          << "\n"
22          << "\n================================================================================"
23          << "\nUSO:"
24          << "\n./cgnuplot.teste.min"
25          << "\n================================================================================" << endl;
26
27  Gnuplot::Terminal("wxt");       // Tipo de terminal gráfico
28
29  // ------------- Graficos 2D ----------------
30  Gnuplot g2d ("lines"); // Construtor
31  g2d.Legend("inside").Legend("left").Legend("bottom").Legend("box");
32  g2d.Title("Titulo do grafico"); // Titulo do grafico
33  g2d.XLabel("rotulo eixo x");    // Rotulo eixo x
34  g2d.YLabel("rotulo eixo y");    // Rotulo eixo y
35  g2d.XRange(-10,10);             // Seta intervalo do eixo x.
36  g2d.PlotEquation( "x*x*sin(x)");// Plota uma determinada equacao
37  wait_for_key();
38                                  // Usando os diferentes estilos de graficos
39
40  // Muda o estilo da funcao para linhas e replota
41  cout << "g2d.Style(\"points\")" << endl;
42  g2d.Style("points");    // Muda estilo linha
43  g2d.PlotEquation( "x*x*x*sin(x)");// Plota uma determinada equacao
44  //g2d.Replot();                   // Replota o gráfico
45  wait_for_key();
46
47  // Muda o estilo da funcao para impulsos, muda titulo e plota nova equacao
48  cout << "Style(\"impulses\")" << endl;
49  g2d.Style("impulses").Title("Style( impulses )").PlotEquation( "x*x+ 5");
```

```
50 wait_for_key();
51
52 g2d.Reset();                    // Reseta estado do grafico
53 cout << endl << "***␣Fim␣do␣exemplo␣" << endl;
54 return 0;
55}
```

Listing 16.3: Arquivo Makefile.

```
DEFINES  =
INCLUDES =
CC       = g++
CPPFLAGS = -std=c++11
LIBS     = -lstdc++
OBJ      = CGnuplot.o
EXEMPLO  = CGnuplot.teste.o CGnuplot.o
EXEMPLOMIN  =  CGnuplot.teste.min.o CGnuplot.o
EXEMPLO_OSX = CGnuplot.teste_osx.o CGnuplot.o
.cpp.o:
        $(CC) -c $(CPPFLAGS) $(DEFINES) $(INCLUDES) $<

all::   CGnuplot.teste CGnuplot.teste.min
min::   CGnuplot.teste.min
osx::   CGnuplot.teste_osx

CGnuplot.o:             CGnuplot.cpp CGnuplot.h
CGnuplot.teste.o:       CGnuplot.teste.cpp CGnuplot.cpp
CGnuplot.testemin.o:    CGnuplot.teste.min.cpp CGnuplot.cpp
CGnuplot.teste_osx.o:   CGnuplot.teste_osx.cpp CGnuplot.cpp

CGnuplot.teste: $(EXEMPLO)
        $(CC) -o $@ $(CFLAGS) $(EXEMPLO) $(LIBS)

CGnuplot.teste.min: $(EXEMPLOMIN)
        $(CC) -o $@ $(CFLAGS) $(EXEMPLOMIN) $(LIBS)

CGnuplot.teste_osx: $(EXEMPLO_OSX)
```

```
29          $(CC) -o $@ $(CFLAGS) $(EXEMPLO_OSX) $(LIBS)
30
31 doc:
32          doxygen
33 dox:
34          doxygen
35 clean:
36          rm *.o *~ CGnuplot.teste a.out
```

A listagem 16.4 mostra um outro exemplo de uso da classe `CGnuplot`, neste exemplo criamos vetores do tipo `vector<double>` e os plotamos usando a classe `CGnuplot`.

Listing 16.4: Um exemplo simplificado de uso da classe CGnuplot.

```
1 ////////////////////////////////////////////////////////////////////////////
2 //                          Programa de teste da classe CGnuplot.         //
3 ////////////////////////////////////////////////////////////////////////////
4 // Esta interface usa pipes e nao ira funcionar em sistemas que nao suportam
5 // o padrao POSIX pipe.
6 //
7 // O mesmo foi testado em sistemas Windows (MinGW e Visual C++) e GNU/Linux(GCC/G++)
8 //
9 // Este programa foi originalmente escrito por:
10 // Historico de versoes:
11 // 0. Interface para linguagem C
12 //    por N. Devillard (27/01/03)
13 // 1. Interface para C++: tradução direta da versao em C
14 //    por Rajarshi Guha (07/03/03)
15 // 2. Correcoes para compatibilidadde com Win32
16 //    por V. Chyzhdzenka (20/05/03)
17 // 3. Novos métodos membros, correcoes para compatibilidade com Win32 e Linux
18 //    por M. Burgis (10/03/08)
19 // 4. Traducao para Portugues, documentacao e modificacoes na interface
20 //    por Bueno.A.D. (30/07/08)
21 //
22 ////////////////////////////////////////////////////////////////////////////
23 //
24 // Requisitos:
```

```
25// - O programa gnuplot deve estar instalado (veja http://www.gnuplot.info/download.html)
26// - No Windows: setar a Path do Gnuplot (i.e. C:/program files/gnuplot/bin)
27//       ou setar a path usando: Gnuplot::set_GNUPlotPath(const std::string &path);
28//       antes de criar qualquer objeto da classe.
29//
30////////////////////////////////////////////////////////////////////////////////
31
32// ------------------------- Inclusão de arquivos -------------------------------
33#include <iostream>
34
35/// A classe CGnuplot  usa pipes no estilo POSIX para se comunicar com o gnuplot.
36// POSIX-Pipe-communikation.
37#include "CGnuplot.h"
38
39// Se estamos no windows
40#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)
41#include <conio.h>          // Para acesso a Ch(), necessario em wait_for_key()
42#include <windows.h>        // Para acesso a Sleep()
43void sleep(int i) { Sleep(i*1000); }
44#endif
45
46// ------------------------- Variaveis globais ----------------------------------
47const int SLEEP_LGTH = 2;  // Tempo de espera em segundos
48const int NPOINTS   = 50;  // Dimensao do array (vetor)
49//#define  SLEEP_LGTH  2
50//#define  NPOINTS   50
51
52using namespace std;       // Usando espaco de nomes da std
53
```

```
54 // ------------------------- Funcoes globais ----------------------------------------
55 /// @brief O programa para ate o pressionamento de uma tecla.
56 void wait_for_key();
57
58 // ------------------------- Funcao Principal ---------------------------------------
59 /// @brief Funcao principal
60 int main(int argc, char* argv[]) {
61  cout    << "\n=========================================================================="
62          << "\n====================␣Programa␣de␣teste␣da␣classe␣CGnuplot========================="
63          << "\n==========================================================================";
64          << "\n"
65          << "\n=========================================================================="
66          << "\nUSO:"
67          << "\n./cgnuplot.teste.min"
68          << "\n==========================================================================" << endl;
69
70   // Se a variavel do gnuplot nao esta setada, faca isto antes de
71   // criar objetos da classe CGnuplot, usando o método publico e estatico:
72   // Gnuplot::set_GNUPlotPath("C:/program files/gnuplot/bin/");
73
74   // Seta o terminal padrao para visualizacao dos graficos  (normalmente nao necessario),
75   // Usuarios de Mac devem usar a opcao "aqua", e nao x11.
76   // Gnuplot::set_terminal_std("x11");
77   cout << "-------------------------------------------------------------\n"
78        << "-->␣Plotando␣graficos␣do␣gnuplot␣usando␣a␣classe␣CGnuplot␣<--\n"
79        << "-->␣Exemplo␣de␣controle␣do␣gnuplot␣usando␣C++␣␣␣␣␣␣␣␣␣␣␣␣␣<--\n"
80        << "-->␣Os␣titulos␣do␣grafico␣ilustram␣a␣operacao␣realizada␣␣␣␣<--\n"
81        << "-------------------------------------------------------------\n" << endl;
82
```

```
83  // Controla a ocorrencia de excessoes, usa a classe GnuplotException
84  try {
85      // Teste geral
86      // Terminal padrao do gnuplot no fedora 9
87      // Se nao funcionar em seu sistema, comente a linha
88      Gnuplot::Terminal("wxt");
89
90      // ------------- Graficos 2D ----------------
91      Gnuplot g2d = Gnuplot("lines"); // Construtor
92      g2d.PointSize(0.8);             // Escala o tamanho do ponto usado na plotagem
93                                      // Legenda
94      g2d.Legend("inside").Legend("left").Legend("bottom").Legend("box");
95      g2d.Title("Titulo do grafico"); // Titulo do grafico
96      g2d.XLabel("rotulo eixo x");    // Rotulo eixo x
97      g2d.YLabel("rotulo eixo y");    // Rotulo eixo y
98      g2d.XRange(-10,10);             // Seta intervalo do eixo x.
99      g2d.PlotEquation( "x*x*sin(x)");// Plota uma determinada equacao
100
101                                     // Usando os diferentes estilos de graficos
102     cout << "Style(\"lines\")" << endl;
103     g2d.Style("Style( lines )").Replot();
104     wait_for_key();
105     g2d.Reset();                    // Reseta estado do grafico
106
107     cout << "Style(\"points\")" << endl;
108     g2d.Style("points").Title("Style( points )").PlotEquation( "x");
109     wait_for_key();
110     g2d.Reset();
111
```

```
112     cout << "Style(\"linespoints\")" << endl;
113     g2d.Style("linespoints").Title("Style(␣linespoints␣)").PlotEquation( "x*x");
114     wait_for_key();
115     g2d.Reset();
116
117     cout << "Style(\"impulses\")" << endl;
118     g2d.Style("impulses").Title("Style(␣impulses␣)").PlotEquation( "x*x+␣5");
119     wait_for_key();
120     g2d.Reset();
121
122     cout << "Style(\"dots\")" << endl;
123     g2d.Style("dots").Title("Style(␣dots␣)").PlotEquation( "x*x*x");
124     wait_for_key();
125     g2d.Reset();
126
127     cout << "Style(steps)" << endl;
128     g2d.Style("steps").Title("Style(␣steps␣)").PlotEquation( "x*x*x*x");
129     wait_for_key();
130     g2d.Reset();
131
132     cout << "Style(\"fsteps\")" << endl;
133     g2d.Style("fsteps").Title("Style(␣fsteps␣)").PlotEquation( "x*x*sin(x)");
134     wait_for_key();
135     g2d.Reset();
136
137     cout << "Style(\"histeps\")" << endl;
138     g2d.Style("histeps").Title("Style(␣histeps␣)").PlotEquation( "x*x*sin(x)");
139     wait_for_key();
140
```

```
141                                  // Legendas, posicoes possiveis
142     cout << "Legend(\"inside␣left␣top␣nobox\")" << endl;
143     g2d.Legend("inside␣left␣top␣nobox").Title("Legend(␣inside␣left␣top␣nobox␣)").Replot();
144     wait_for_key();
145
146     cout << "Legend(\"inside␣center␣center␣nobox\")" << endl;
147     g2d.Legend("inside␣center␣center␣nobox").Title("Legend(␣inside␣center␣center␣nobox␣)").Replot();
148     wait_for_key();
149
150     cout << "Legend(\"inside␣right␣bottom␣box\")" << endl;
151     g2d.Legend("inside␣right␣bottom␣box").Title("Legend(␣inside␣right␣bottom␣box␣)").Replot();
152     wait_for_key();
153
154     cout << "Legend(\"outside␣right␣top␣box\")" << endl;
155     g2d.Legend("outside␣right␣top␣box").Title("Legend(␣outside␣right␣top␣box␣)").Replot();
156     wait_for_key();
157
158     // ------------- Graficos 3D ----------------
159     Gnuplot g3d = Gnuplot("lines"); // Construtor
160     g3d.Grid(1);                    // Ativa/Desativa o grid
161     g3d.Samples(50);                // Seta taxa de amostragem
162     g3d.IsoSamples(50);             // Seta densidade de isolinhas
163     g3d.Hidden3d();                 // Ativa/Desativa remocao de linhas ocultas
164     g3d.Surface();                  // Ativa/Desativa a visualizacao da superficie
165     g3d.Title("Titulo␣do␣grafico"); // Titulo do grafico
166     g3d.XLabel("rotulo␣eixo␣x");    // Rotulo eixo x
167     g3d.YLabel("rotulo␣eixo␣y");    // Rotulo eixo y
168     g3d.ZLabel("rotulo␣eixo␣z");    // Rotulo eixo z
169     g3d.XRange(-10,10);             // Seta intervalo do eixo x.
```

```
170     g3d.ZAutoscale();                   // Seta autoescala de z
171     g3d.PlotEquation3d( "x*sin(x)*sin(y)+4" );
172     wait_for_key();
173
174                                         // Suavizacao
175     cout << "Smooth(0)" << endl;
176     g3d.Smooth(0).Title("Smooth(0)").Replot();          // Desativa suavizacao
177     wait_for_key();
178     cout << "Smooth(1)" << endl;
179     g3d.Smooth(1).Title("Smooth(1)").Replot();          // Ativa suavizacao
180     wait_for_key();
181                                         // Ativar/desativar grid
182     cout << "Grid()" << endl;
183     g3d.Grid().Title("Grid()").Replot();
184     wait_for_key();
185     cout << "Grid(0)" << endl;
186     g3d.Grid(0).Title("Grid(0)").Replot();
187     wait_for_key();
188
189                                         // Ocultar linhas escondidas
190     cout << "Hidden3d(0)" << endl;
191     g3d.Hidden3d(0).Title("Hidden3d(0)").Replot();
192     wait_for_key();
193     cout << "Hidden3d()" << endl;
194     g3d.Hidden3d().Title("Hidden3d()").Replot();
195     wait_for_key();
196
197                                         // Taxa amostragem
198     cout << "Samples(10)" << endl;
```

```
199     g3d.Samples(10).Title("Samples(10)").Replot();
200     wait_for_key();
201     cout << "Samples(50)" << endl;
202     g3d.Samples(50).Title("Samples(50)").Replot();
203     wait_for_key();
204
205                                     // Densidade de isolinhas
206     cout << "IsoSamples(10)" << endl;
207     g3d.IsoSamples(10).Title("IsoSamples(10)").Replot();
208     wait_for_key();
209     cout << "IsoSamples(50)" << endl;
210     g3d.IsoSamples(50).Title("IsoSamples(50)").Replot();
211     wait_for_key();
212
213                                     // Contorno em superficies, base, surface, both.
214     cout << "Contour(\"base\")" << endl;
215     g3d.Contour("base").Title("Contour(base)").Replot();
216     wait_for_key();
217     cout << "Contour(\"surface\")" << endl;
218     g3d.Contour("surface").Title("Contour(surface)").Replot();
219     wait_for_key();
220     cout << "Contour(\"both\")" << endl;
221     g3d.Contour("both").Title("Contour(both)").Replot();
222     wait_for_key();
223
224
225     // ------------- A seguir exemplos do codigo original ----------------
226     Gnuplot g1 = Gnuplot("lines");
227     cout << "***␣Plota␣uma␣equacao␣da␣forma␣y␣=␣ax␣+␣b;␣com␣a=1,␣b=0" << endl;
```

```
228     g1.Title("PlotSlope y = x");                    // Seta o titulo.
229     g1.PlotSlope(1.0,0.0,"PlotSlope y = x");        // Plota Reta
230     wait_for_key();
231
232     cout << "*** Plota uma equacao da forma y = ax + b; com a=2, b=0" << endl;
233     cout << "PlotSlope y = 2*x" << endl;
234     g1.PlotSlope(2.0,0.0,"y = 2x");
235     wait_for_key();
236
237     cout << "*** Plota uma equacao da forma y = ax + b; com a=-1, b=0" << endl;
238     cout << "PlotSlope y = -x" << endl;
239     g1.PlotSlope(-1.0,0.0,"y = -x");
240     wait_for_key();
241     g1.Title();
242
243                                             // Equacoes
244     g1.ResetPlot();                         // Reseta o grafico
245     cout << endl << endl << "*** Plotando Equacoes" << endl;
246
247     cout << "*** PlotEquation y = sin(x)" << endl;
248     g1.PlotEquation("sin(x)","PlotEquation sine, sin(x)");      // Plota uma equacao
249     wait_for_key();
250
251     cout << "*** y = log(x)" << endl;
252     g1.Legend("box").Legend("left").PlotEquation("log(x)","PlotEquation logarithm, log(x)"); // Plota uma equacao
253     wait_for_key();
254
255     cout << "*** y = sin(x) * cos(2*x)" << endl;
256     g1.PlotEquation("sin(x)*cos(2*x)","PlotEquation, sin(x)*cos(2*x)"); // Plota uma equacao
```

```
257     wait_for_key();
258
259                                             // Controlando estilos de graficos - styles
260     g1.ResetPlot();
261     cout << endl << endl << "*** Mostrando estilos - styles" << endl;
262
263     cout << "*** sin(x) usando PointSize(0.8).Style(\"points\")" << endl;
264     g1.PointSize(0.8).Style("points");
265     g1.PlotEquation("sin(x)","PlotEquation sin(x), usando points");
266     wait_for_key();
267
268     cout << "*** sine usando estilo de impulses" << endl;
269     g1.Style("impulses");
270     g1.PlotEquation("sin(x)","PlotEquation sin(x), usando impulses");
271     wait_for_key();
272
273     cout << "*** sine usando estilo de steps" << endl;
274     g1.Style("steps");
275     g1.PlotEquation("sin(x)","PlotEquation sin(x), usando steps");
276     wait_for_key();
277
278                                             // Salvando para arquivo postscript - ps
279     g1.ResetAll();                          // Reseta todos os dados
280     cout << endl << endl << "*** Salvando para arquivo postscript - ps" << endl;
281
282     cout << "y = sin(x) salvo no arquivo test_output.ps no diretorio corrente" << endl;
283     g1.SaveTops("test_output");
284     g1.Style("lines").Samples(300).XRange(0,5);
285     g1.PlotEquation("sin(12*x)*exp(-x)").PlotEquation("exp(-x)");
```

```
286
287     cout << "*** Plotando novamente em uma janela" << endl;
288     g1.ShowOnScreen();                          // Ativa janela de saida grafica
289
290                                                 // Usando vetores do usuario (conjunto de dados)
291     cout << "*** Criando vetores x, y, y2, dy, z a serem plotados" << endl;
292     std::vector<double> x, y, y2, dy, z;
293
294     for (int i = 0; i < NPOINTS; i++) {         // Preenche os vetores x, y, z
295         x.push_back((double)i);                 // x[i] = i
296         y.push_back((double)i * (double)i);     // y[i] = i^2
297         z.push_back( x[i]*y[i] );               // z[i] = x[i]*y[i] = i^3
298         dy.push_back((double)i * (double)i / (double) 10); // dy[i] = i^2 / 10
299       }
300     y2.push_back(0.00); y2.push_back(0.78); y2.push_back(0.97); y2.push_back(0.43);
301     y2.push_back(-0.44); y2.push_back(-0.98); y2.push_back(-0.77); y2.push_back(0.02);
302
303     g1.ResetAll();
304     cout << endl << endl << "*** Plota vetor y de doubles" << endl;
305     g1.Style("impulses").PlotVector(y,"PlotVector y, usando impulses");
306     wait_for_key();
307
308     g1.ResetPlot();
309     cout << endl << endl << "*** Plota vetores x e y, pares ordenados (x,y)" << endl;
310     g1.Grid();
311     g1.Style("points").PlotVector(x,y,"PlotVector x e y, pares ordenados (x,y), usando points");
312     wait_for_key();
313
314     g1.ResetPlot();
```

```
315     cout << endl << endl << "*** Plota vetores x e y e z, valores ordenados (x,y,z)" << endl;
316     g1.Grid(0);
317     g1.PlotVector(x,y,z,"PlotVector (x,y,z), usanddo points");
318     wait_for_key();
319
320     g1.ResetPlot();
321     cout << endl << endl << "*** Plota vetores x, y e dy, valores ordenados e barra de erro (x,y,dy)" << endl;
322     g1.PlotVectorXYErrorBar(x,y,dy,"PlotVectorXYErrorBar  valores ordenados x,y e barra de erro (x,y,dy)");
323     wait_for_key();
324
325                                                 // Usando multiplas janelas de saida
326     cout << endl << endl;
327     cout << "*** multiple output windows" << endl;
328
329     g1.ResetPlot();
330     g1.Style("lines");
331     cout << "window 1: sin(x)" << endl;
332     g1.Grid(1).Samples(600).XRange(0,300);
333     g1.PlotEquation("sin(x)+sin(x*1.1)", "PlotEquation Grid(1).Samples(600).XRange(0,300)");
334     wait_for_key();
335
336     g1.XAutoscale().Title("XAutoscale()").replot();
337     wait_for_key();
338
339     Gnuplot g2;
340     cout << "Janela 2: plotando vetores" << endl;
341     g2.PlotVector(y2,"Pontos de y2");
342     g2.Smooth().PlotVector(y2,"Smooth(cspline)");
343     g2.Smooth("bezier").PlotVector(y2,"Smooth(bezier)");
```

```
344     g2.Smooth();
345     wait_for_key();
346
347     cout << "Janela␣3:␣plotando␣equacoes,␣log(x)/x" << endl;
348     Gnuplot g3("lines");
349     g3.Grid(1);
350     g3.PlotEquation("log(x)/x","log(x)/x");
351     wait_for_key();
352
353     cout << "Janela␣4:␣splot␣x*x+y*y" << endl;
354     Gnuplot g4("lines");
355     g4.ZRange(0,100);
356     g4.XLabel("x-axis").YLabel("y-axis").ZLabel("z-axis");
357     g4.PlotEquation3d("x*x+y*y");
358     wait_for_key();
359
360     cout << "Janela␣5:␣splot␣usando␣Hidden3d" << endl;
361     Gnuplot g5("lines");
362     g5.IsoSamples(25).Hidden3d();
363     g5.PlotEquation3d("x*y*y");
364     wait_for_key();
365
366     Gnuplot g6("lines");
367     cout << "Janela␣6:␣splot␣usando␣Contour" << endl;
368     g6.IsoSamples(60).Contour();
369     g6.Surface().PlotEquation3d("sin(x)*sin(y)+4");
370     wait_for_key();
371
372     g6.Surface().Replot();
```

```
373         wait_for_key();
374
375         Gnuplot g7("lines");
376         cout << "Janela 7: usando Samples" << endl;
377         g7.XRange(-30,20).Samples(40);
378         g7.PlotEquation("besj0(x)*0.12e1").PlotEquation("(x**besj0(x))-2.5");
379         wait_for_key();
380
381         g7.Samples(400).Replot();
382         wait_for_key();
383
384         Gnuplot g8("filledcurves");
385         cout << "Janela 8: filledcurves" << endl;
386         g8.Legend("outside right top").XRange(-5,5);
387         g8.PlotEquation("x*x").PlotEquation("-x*x+4");
388
389                                                 // Plota uma imagem
390         Gnuplot g9;
391         cout << "Janela 9: plot_image" << endl;
392         const int iWidth  = 255;
393         const int iHeight = 255;
394         g9.XRange(0,iWidth).set_yrange(0,iHeight).CBRange(0,255);
395         g9.Command("set palette gray");
396         unsigned char ucPicBuf[iWidth*iHeight];
397                                                 // Gera imagem em tons dde cinza
398         for(int iIndex = 0; iIndex < iHeight*iWidth; iIndex++) {
399             ucPicBuf[iIndex] = iIndex % 255;
400           }
401         g9.plot_image(ucPicBuf,iWidth,iHeight,"greyscale");
```

```
402       wait_for_key();
403
404       g9.PointSize(0.6).Legend(0).PlotSlope(0.8,20);
405       wait_for_key();
406
407                                          // Controle manual
408       Gnuplot g10;
409       cout << "Janela 10: controle manual" << endl;
410       g10.Cmd("set samples 400").Cmd("plot abs(x)/2"); // Usando Cmd()
411       g10 << "replot sqrt(x)" << "replot sqrt(-x)";    // Usando operador  <<
412       wait_for_key();
413
414     }
415   catch (GnuplotException ge)    {
416       cout << ge.what() << endl;
417     }
418
419   cout << endl << "*** Fim do exemplo " << endl;
420
421   return 0;
422 }
423
424 void wait_for_key () {
425 // Todos as teclas serao considedadas, inclusive as setas
426 #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)
427   cout << endl << "Pressione qualquer tecla para continuar..." << endl;
428   FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
429   _Ch();
430 #elif defined(unix) || defined(__unix) || defined(__unix__) || defined(__APPLE__)
```

```
431   cout << endl << "Pressione␣ENTER␣para␣continuar..." << endl;
432   std::cin.clear();                                   // Zera estado de cin
433   std::cin.ignore(std::cin.rdbuf()->in_avail());// Ignora
434   std::cin.get();                                     // Espera o pressionamento do enter
435 #endif
436   return;
437 }
```

Capítulo 17

Resumo de C++11/C++14

A listagem **??** apresenta um resumo dos principais comandos de C++11/C++14.

Listing 17.1: Um resumo de C++11 - C++14.

```
1 /**=========================================================================
2 Lista com as principais novidades de C++11
3 Autor: André Duarte Bueno.
4 =========================================================================
5 Lista com as principais novidades de C++11 -> Núcleo da linguagem
6 ========================================================================= */
7 // [ ] ----------------------------------------------------------------> auto
8 auto x = 3;
9
```

```
10 // [ ] ------------------------------------------------------------------> decltype
11 decltype(x) y;
12
13 // [ ] ------------------------------------------------------------------> nullptr
14 char *pc = nullptr;      // OK
15 int  *pi = nullptr;      // OK
16
17 // [ ] ------------------------------------------------------------------> rvalue reference
18 template<typename T> void f(T&& param);                  // parâmetro do tipo rvalue reference
19 template<typename T> void f(std::vector<T>&& param);    // parâmetro do tipo rvalue reference
20 int main() { int&& var1 = 10;                           // rvalue reference (nome para uma constante)
21 cout << "\nvar1 = " << var1 ; }
22
23 // [ ] ------------------------------------------------------------------> novas strings
24 char     s8[]  = u8"UTF-8 cstring. ";                   // const char[].
25 char16_t s16[] = u"UTF-16 cstring. ";                   // const char16_t[].
26 char32_t s32[] = U"UTF-32 cstring. ";                   // const char32_t[].
27 cout << u8"This is a Unicode Character: \u2018"       << endl;
28
29 // [ ] ------------------------------------------------------------------> raw strings
30 // C++11 fornece a opção de raw strings literal: R("string \t 1"), não é interpretada
31 cout <<  "(\a xx \b yy \t zz \n)" << endl;
32 cout << R"(\a xx \b yy \t zz \n)" << endl;
33
34 // [ ] ------------------------------------------------------------------> sizeof
35 // C++11 permite obter o sizeof de membro da classe
36 class CPonto { public: double x; double y;};
37 cout << "sizeof(CPonto) = "    << sizeof(CPonto)    << endl;
38 cout << "sizeof(CPonto::x) = " << sizeof(CPonto::x) << endl;
```

```
39
40 // [ ] ------------------------------------------------------------------------> enum class
41 // A vantagem de enum class é que a mesma não pode ser convertida para int;
42 enum class EDiaSemanaCpp11 { segunda = 2, terca = 3, quarta = 4, quinta = 5, sexta = 6 };
43
44 // [ ] ------------------------------------------------------------------------> enum class: tipo
45 // Também podemos definir o tipo usado pela enumeração
46 enum class EMesesAnoCpp11: unsigned int { janeiro = 1, fevereiro, marco, abril,
47 maio, junho, julho, agosto, setembro, outubro, novembro, dezembro };
48
49 // [ ] ------------------------------------------------------------------------> union
50 union UMiscelanea {   bool b;  int i;  double d;
51   CPonto p;   // C++11 retirou restrição dos tipos aceitos em uniões
52   UMisccelanea() { new( &p ) CPonto(); }
53 };
54 // [ ] ------------------------------------------------------------------------> static_assert
55 // Mensagem de erro em tempo de compitalção
56 static_assert(constant-expression, error-message);
57 static_assert(sizeof(int) <= sizeof(T), "A␣dimensão␣de␣T␣não␣é␣suficiente!");
58 static_assert(std::is_integral<T>::value, "O␣parâmetro␣de␣f␣deve␣ser␣do␣tipo␣integral.");
59 static_assert((pi < 3.14) && ( pi > 3.15), "Aumentar␣precisão␣de␣pi!");
60
61 // [ ] ------------------------------------------------------------------------> constexpr
62 constexpr int FuncaoConstante() { return 5; }
63 constexpr int XY (int x, int y) { return x * y; }
64 constexpr double aceleracaoGravidade = 9.8;
65
66 // [ ] ------------------------------------------------------------------------> range based for
67 int vetor_estilo_c[5] = { 1, 2, 3, 4, 5 };
```

```
68 for (int &elemento_vetor : vetor_estilo_c) cout << elemento_vetor << endl;
69 vector<int> v(5);
70 for( auto elemento_vetor : v )              cout << elemento_vetor << endl;
71
72 // [ ] ---------------------------------------------------------------> Inicialização uniforme
73 int x3{5.0}, x4 ={5.3}; // Com inicialização uniforme de C++11 aponta erro pois 5.0 é double
74 int x5, x6{};           // x5 valor indefinido, x6 valor padrão = 0
75 int *ptr1,*ptr2{};      // ptr1 valor indefinido, ptr2 em C++11 assume valor nullptr
76 char cx{14},cy{35000}; // cx OK, 14 é do tipo inteiro; cy Erro, estouro do intervalo
77 std::vector<int> vx { 0, 1, 2, 4, 5 };    // OK
78 std::vector<int> vy { 1, 2.3, 4, 5.6 };   // Erro
79 int vc1[3];   // Cria vetor estilo de C, com 3 elementos [0->2], valores indefinidos
80 int vc2 []= {1,2,3};  // Cria vetor estilo de C, com 3 elementos [0->2], valores definidos
81
82 // [ ] ---------------------------------------------------------------> Lista inicialização
83 class CLista { private:  vector< float > v;
84                public:   CLista( std::initializer_list<float> lista ): v(lista){};
85                          vector< float > V()    { return v; }; };
86
87 class CPonto {  double x;   double y;        // Construtor
88 CPonto(double _x, double _y): x(_x), y(_y) { std::cout << "Passou pelo construtor de SPonto;";} };
89 CPonto GetCPonto() { return { 0.0, 0.9 }; } // O objeto é criado sem usarmos o tipo CPonto
90
91 void Saida ( std::initializer_list<int> l)  // Note que recebe como parâmetro a lista l.
92 {    for (auto it = l.begin(); it != l.end(); ++it) std::cout << *it << "\n"; }
93
94 int main() {                                // Chama função saida, imprime valores na tela
95   Saida ({0,1,2,3,4,5,6,7,8,9});            // Usando initializer_list
96   CLista lista{ 5.1, 5.2, 5.3, 5.4 };       // Cria objeto lista
```

```
97   vector<int> v{ 1,2,3,4,5 };                    // Usando initializer_list com biblioteca padrão
98
99   CPonto p2  { 5.1 , 6.1 };                      // Usando inicialização uniforme padrão C++11
100  // Criando uma lista de pontos usando inicialização uniforme padrão C++11
101  CPonto lista_pontos[4] =  { { 5.2,6.2 }, {5.3,6.3}, {5.4,6.4}, {5.5,6.5} };
102}
103
104// [ ] ----------------------------------------------------------------------> Funções lambda
105/*auto nomeFuncao = [captura](parametros)->tipoRetorno {corpo da funcao}
106[]: Não capturar nada.
107[=]: Todas as variáveis externas são capturadas por valor.
108[&]: Todas as variáveis externas são capturadas por referência.
109[x, &y]: capturar x por valor(cópia) e y por referência.
110[&, x]: Todas as variáveis externas são capturadas por referência, exceto x que é por valor.
111[=, &z]: Todas as variáveis externas são capturadas por valor, exceto z que é por referência. */
112
113// Função lambda anônima é criada e já executada. O () executa a função.
114[] { std::cout << "Função lambda criada e já executada" << std::endl;   } ();
115
116// Função lambda criada e chamada a seguir
117auto l = [] { std::cout << "Função lambda criada e chamada a seguir" << std::endl; };
118l();                                                    // Chama função lambda
119
120// Definição de função lambda que não captura nada e que não recebe parâmetros.
121auto ptr_funcao = [] () { cout << "Olá mundo!\n"; };
122ptr_funcao();
123
124// Definição de função lambda que não captura nada e que recebe os parâmetros x e y.
125auto ptr_funcao2 = [](int x, int y) { return x + y; }
```

```
126 cout << "␣x␣+␣y␣=␣" << ptr_funcao2(3,4) << endl;
127
128 // Usando função lambda com captura por referencia
129 int soma = 0;
130 auto Soma = [&soma]( int x ) { soma += x; cout << "Soma␣=␣" << soma << endl; };
131 Soma(10);
132
133 // [ ] ----------------------------------------------------------------> novo formato funções
134 auto X(int _x) -> void          { x = _x; }
135 auto X() -> int                 { return x; }
136 auto Set(int _x, int _y) -> void;
137 auto CPonto::Set(int _x, int _y) -> void{ x = _x;  y = _y; }
138
139 // [ ] ----------------------------------------------------------------------> Ponteiro Função
140 void FC(int x)       { cout << "FC␣x=" << x << endl; } // // Declara e define função
141 void (*pFC)(int) = &FC;                          // ponteiro para função C++98
142 typedef void (*PonteiroFuncao)(double);          // ponteiro para função C++98
143
144 using PonteiroFuncao = void (*)(double);         // ponteiro para função C++11
145 std::function<void(int)> pF11 = &F11;            // ponteiro para função C++11
146 auto autopF11 = &F11;                            // ponteiro para função C++11
147
148 class C {    void F11(int x)   { cout << "F11␣x=" << x << endl; } };
149 void (C::*pF03)(int) = &C::F03;                  //  ponteiro para função C++98
150 std::function<void (C&, int)> pF11_ref = &C::F11; // funciona como referencia
151 std::function<void (C*, int)> pF11_ptr = &C::F11; // funciona como ponteiro
152
153
154 // [ ] ----------------------------------------------------------------> delegação construtor
```

```
155 explicit CPonto (int _x, int _y):x(_x),y(_y)    {}
156 CPonto(int xy) : CPonto(xy,xy)  {}  // um construtor chama o outro
157
158 // [ ] ----------------------------------------------------------------> default e delete
159 class NonCopyable { public:  // Diz para o compilador desabilitar o operator= (não criar)
160   NonCopyable& operator=(const NonCopyable&) = delete;
161
162   // Diz para o compilador desabilitar o construtor de cópia (não criar)
163   NonCopyable(const NonCopyable&) = delete;
164
165   // Diz para o compilador criar o construtor default
166   NonCopyable() = default; };
167
168 // [ ] ----------------------------------------------------------------> override e final
169 class CPonto { virtual auto Entrada() -> void;
170                virtual auto Saida() -> void; };
171
172 class CCirculo: public CPonto {
173     virtual void Entrada() override ; // sobrecreve método virtual da classe base
174     virtual void Saida() final; };    // última atualização de Saida
175
176 // [ ] ----------------------------------------------------------------> for_each
177 char s[] = "Olá␣Mundo!";
178 for_each( s, s + sizeof(s), [] (char c){ cout << c << endl; });
179 int vc[] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 };
180 for_each( begin(vc), end(vc), [](int x) { cout << x << '␣';});
181 int soma = 0;
182 for_each( begin(vc), end(vc), [&soma] (int x) { soma += x;});
183
```

```
184
185
186
187 ================================================================================
188 Lista com as principais novidades de C++11 -> Biblioteca std
189 ================================================================================
190 // [ ] ---------------------------------------------------------------------> array
191 #include <array>
192 std::array<int, 4> array_4_int { {1,2,3,4} };           // Precisa de duplo {}
193 array<int, 3> array_3_int = {1, 2, 3};                  // Apos = precisa {} simples
194 array<string, 2> array_2_string = {"a", "b"} ;
195 sort(array_4_int.begin(), array_4_int.end());
196 for(auto& ae: array_4_int)          cout << ae << ' ';
197
198 // [ ] ---------------------------------------------------------> all_of any_of none_of
199 void Teste( vector<int> &v , string msg )
200 { cout << "Vetor " << msg << endl;
201   if ( all_of(v.begin(), v.end(), [](int ev) { return ev > 0;} ) )   // Todos positivos?
202     cout << "Todos positivos\n";
203   if ( any_of(v.begin(), v.end(), [](int ev) { return ev > 0;} ) )   // Pelo menos um positivo?
204     cout << "Pelo menos um positivo\n"; }
205 int main() {  vector<int> v1{   1, 2, 3, 4, 5};    Teste( v1 , "v1"  );
206              vector<int> v2{ 0,-1, 2, 3, 4, 5};   Teste( v2  , "v2" ); }
207
208 // [ ] ---------------------------------------------------------------> unique_ptr
209 std::unique_ptr<int> ptr_int3( new int(3) );      // Cria ponteiro e objeto
210 cout << *ptr_int3 << endl;                        // Usa
211 unique_ptr<int> ptr_int5  = std::move(ptr_int3); // Transfere propriedade
212 cout << *ptr_int5 << endl;                        // Usa
```

```
213 ptr_int5.reset();                                   // Destrõe
214 ptr_int3.reset();                                   // Não faz nada.
215
216 // [ ] ------------------------------------------------------------------> shared_ptr
217 // Use shared_ptr? quando quizer vários ponteiros apontando para mesmo objeto,
218 // somente quando o último for deletado o objeto será efetivamente deletado.
219 shared_ptr<int> ptr_int6(new int(6));               // Cria ponteiro e objeto
220 cout << *ptr_int6 << endl;                          // Usa
221 shared_ptr<int> ptr_int7 = ptr_int6;               // ptr7 aponta p/ mesmo objeto que ptr6
222 cout << *ptr_int7 << endl;
223 ptr_int6.reset();                                   // Não destrõe e objeto
224 cout << *ptr_int7 << endl;                          // Usa objeto
225 ptr_int7.reset();                                   // Agora deleta objeto
226
227 // [ ] ------------------------------------------------------------------> weak_ptr
228 shared_ptr<int> ptr_int8(new int(8));
229 cout << *ptr_int8 << endl;
230 weak_ptr<int> wptr_int8 = ptr_int8;             // ptr_int8 owns the memory.
231   {
232   shared_ptr<int> ptr_int9 = wptr_int8.lock(); // Agora p8 e p9 acessam a mesma memória.
233   cout << *ptr_int9 << endl;
234   if( ptr_int9 )                                // Sempre verifique o ponteiro
235     cout << *ptr_int9 << endl;                  // Faça algo com ptr_int9
236   }     // ptr_int9 é destruído; ptr_int8 volta a ter a propriedade.
237   cout << *ptr_int8 << endl;
238   ptr_int8.reset();                             // A memoria é deletada.
239 }
240
241 // [ ] ------------------------------------------------------------------> function
```

```
242#include <functional>
243function<double(double)> fx2 = [](double x) { return x*x;};  // funcao f
244function<double(double)> f2x = [](double x) { return 2.0*x;};// funcao g
245// Cada vez mais perto da notação matemática! Agrupando funções, como g(f(x));
246std::function<double(double)> gf(function<double(double)> f, function<double(double)> g )
247     { return [=](double x) { return g(f(x)); };
248// Uso de gf, cria funcao fx4, retorna double, recebe funcao
249function<double(double)> fx4 = gf(fx2, fx2);
250int main() {    double x = 3;
251   cout << "x = " << x << end;
252   cout << "fx2 = " << fx2(x) << endl;
253   cout << "fx4 = " << fx4(x) << endl; }
254
255 // [ ] ---------------------------------------------------------------------> bind
256// Declara função f que recebe dois parâmetros, um int e uma string
257void fis( int x, string s) { cout << "int x = " << x << " string s = " << s << endl; }
258int main() {
259  fis( 2, " oi tudo bem ");
260  // Cria ponteiro para função fis que recebe apenas a string
261  std::function<void( string )> fs = std::bind(&fis, 3 , std::placeholders::_1);
262  fs("Usando fs, passando apenas a string");
263  // Cria ponteiro para função alternativa que recebe apenas o inteiro
264  function<void( int )> fi = std::bind(&fis, "Usando fi" ,std::placeholders::_2);
265  fi(7); }
266
267// [ ] ---------------------------------------------------------------------> pair
268// Mostra uso de tie com pair<> e equivalencia de pair com tuple
269  pair<double,double> p = make_pair(1.1,2.2);
270  cout << "get<0>(p) = " << get<0>(p)   << " , get<1>(p) = "    << get<1>(p) << endl;
```

```
271   cout << "p.first␣␣␣=␣" << p.first     << "␣,␣p.second␣␣=␣"    << p.second << endl;
272
273 // [ ] ---------------------------------------------------------------------> tuple
274 // Mostra uso de tuple, get<>, tie, pair
275 #include <tuple>
276 // Cria tuple com 3 doubles
277 tuple<double, double, double> notasJoao(8.7,4.2,5.7);
278 cout<< "\nJoao\n"
279       << "P1:␣" << get<0>(notasJoao) << ",␣"    // Acesso aos elementos da tuple
280       << "P2:␣" << get<1>(notasJoao) << ",␣"    // usando funcao get<indice>(objeto_tuple)
281       << "P3:␣" << get<2>(notasJoao) << '\n';
282 std::get<2>(notasJoao) = 6.3;                   // Nota p3 corrigida, usa referencia.
283
284 // Mostra uso da funcao tie() para obter, separadamente, os valores da tuple
285 double n1,n2,n3;
286 tie(n1, n2, n3) = notasJoao;
287 cout<< "\nJoao\n" << "n1:␣" << n1 << ",␣" << "n2:␣" << n2 << ",␣" << "n3:␣" << n3 << '\n';
288
289 // [ ] ---------------------------------------------------------------------> forward_as_tuple
290 // forward_as_tuple cria objeto temporario que funciona como uma tupla
291 // para objetos rvalue (right value). Note que como sao rvalue, nao alocam espaco em disco;
292 #include <tuple>        // std::tuple e std::make_tuple
293 // Note que os parametros da tuple sao right value
294 void print_pack (std::tuple<std::string&&,double&&> pack)
295 {   std::cout << std::get<0>(pack) << ",␣" << std::get<1>(pack) << std::endl; }
296 int main() { print_pack (std::forward_as_tuple(string("Joao"), 8.7)); }
297
298 // [ ] ---------------------------------------------------------------------> remove_if
299 bool is_even(int N) { return N % 2 == 0; } // Retorna verdadeiro se for par
```

```
300 int main() {   vector<int> v{1,2,3,4,5,6};
301   for_each (v.begin(),v.end(),[](int ev){ cout << ev << '\t'; });  // Vetor v antes de remove_if
302   remove_if(v.begin(),v.end(),is_even);
303   for_each (v.begin(),v.end(),[](int ev){ cout << ev << '\t'; });  // Vetor v depois de remove_if
304
305   // Efetivamente remove elementos no intervalo final do vetor
306   v2.erase(remove_if(v2.begin(), v2.end(), is_even), v2.end());  }
307
308 // [ ] ----------------------------------------------------------------------> random
309 // O gerador números randomicos tem duas partes; um motor que gera números randomicos
310 // e uma distribuição matemática.
311 // Motores: linear_congruential_engine,subtract_with_carry_engine e mersenne_twister_enginee.
312 // Distribuições: uniform_int_distribution, uniform_real_distribution,
313 // bernoulli_distribution, binomial_distribution, geometric_distribution, poisson_distribution,
314 // normal_distribution, student_t_distribution, chi_squared_distribution,
315 // exponential_distribution, gamma_distribution, lognormal_distribution,
316 // cauchy_distribution, lognormal_distribution, weibull_distribution,
317 // extreme_value_distribution, fisher_f_distribution, negative_binomial_distribution,
318 // discrete_distribution, piecewise_constant_distribution, piecewise_linear_distribution.
319
320 #include <random>
321 int main()
322 { uniform_int_distribution<int> distribuicao(-20, 20); // Cria distribuição uniforme
323   mt19937 motor;                                       // Cria motor "Mersenne twister MT19937"
324   int numeroRandomico = distribuicao(motor);           // Gera número aleatório
325
326   std::normal_distribution<double>  normal(0.0,1.0);   // Normal, media 0 e desvio padrao 1
327   cout << "␣media␣=␣" << normal.mean() << "␣desvio␣padrao␣=␣" << normal.stddev()
328        << "␣max␣=␣" << normal.max() << "␣min␣=␣" << normal.min() << endl;
```

```
329   normal = normal_distribution<double>(12,3);           // Seta media = 12 e desvio padrao = 3
330
331 std::default_random_engine motor2;                     // Cria motor, usa default
332 auto Normal = std::bind(normal, motor2);               // Cria gerador de número aleatorio
333 vector<double> vna(500);                               // Cria vetor de numeros aleatorios
334 for( double &ev : vna ) ev = Normal();                 // Gera números aleatóros
335 }
336
337
338
339
340
341
342
343
344 // [ ] ---------------------------------------------------------------------> chrono
345 #include <chrono> // Biblioteca date time no C++11
346 #include <ctime>  // Biblioteca date time no C++03 (<time> no C)
347 int main() { // Cria objeto time_point
348   chrono::time_point<chrono::system_clock> start;
349 // Define valor de start como sendo agora (antes do processamento)
350   start = chrono::system_clock::now();
351 // Chama função com determinado tempo de processamento
352   int result = sin(45);
353 // Define valor de end como sendo agora (depois do processamento)
354   auto end = chrono::system_clock::now();
355 // count() retorna numero ticks, a diferença é convertida em segundos.
356   int elapsed_seconds = chrono::duration_cast<chrono::seconds>(end-start).count();
357   time_t end_time = chrono::system_clock::to_time_t(end);
```

```
358   cout << "Computação␣terminada␣em␣␣" << ctime(&end_time)
359        << "tempo(s)␣decorrido␣:␣" << elapsed_seconds << "s\n";
360 }
361 // [ ] ----------------------------------------------------------------------> regex (-lregex)
362 #include <regex>         // regex, replace, match_results
363 // regex - Classe que representa uma Expressão Regular - ER.
364 // match_results - representa as ocorrências, casos em que a ER foi encontrada.
365 // regex_search - função usada para localizar uma ocorrência da ER.
366 // regex_replace - função que substitue a ocorrência encontrada por outro texto.
367 // As funções regex_search e regex_replace recebem uma expressão regular e uma string e
368 // escreve as ocorrências encontradas na estrutura match_results.
369 int main(){
370 if (regex_match ("Palmeiras,␣Campeão␣Mundial␣1951", regex("r") ) )
371  cout << "\nA␣expressão␣regular␣\"(ras)\"␣foi␣encontrada␣em␣\"Palmeiras,␣Campeão␣Mundial␣1951\"";
372
373 // A procura pela expressao regular er, sera feita em s pela funcao regex_match.
374 string s ("Palmeiras␣campeão␣mundial␣1951");     // string a ser pesquisada
375 regex er ("r)");                                  // expressao regular usada na pesquisa
376 if (regex_match (s,er))                           // faz a procura
377  cout << "\nA␣expressão␣regular␣\"(ras)\"␣foi␣encontrada␣em␣\"Palmeiras,␣Campeão␣Mundial␣1951\"";
378
379 // Faz a procura usando iteradores
380 if ( regex_match ( s.begin(), s.end(), er ) )     cout << "range␣matched\n";
381
382 // o mesmo que match_results<const char*> cm;
383 cmatch cm;
384 regex_match ("Palmeiras,␣Campeão␣Mundial␣1951",cm,er);
385 cout << "string␣literal␣with␣" << cm.size() << "␣matches\n";
386
```

```
387// o mesmo que match_results<string::const_iterator> sm;
388smatch sm;
389regex_match (s,sm,er);
390cout << "string␣object␣with␣" << sm.size() << "␣matches\n";
391
392regex_match ( s.cbegin(), s.cend(), sm, er);
393cout << "Usando␣intervalo,␣foram␣encontradas␣" << sm.size() << "␣ocorrências\n";
394
395// usando os flags de forma explicita:
396regex_match ( "subject", sm, er, regex_constants::match_default );
397cout << "As␣ocorrências␣são:␣";
398for (unsigned i=0; i<sm.size(); ++i) {    cout << "[" << sm[i] << "]␣";  }
399}
400---
401#include <regex>
402int main(){ std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};
403std::regex txt_regex("[a-z]+\\.txt");
404for (const auto &fname : fnames)
405        std::cout << fname << ":␣" << std::regex_match(fname, txt_regex) << '\n';}
406// [ ] ------------------------------------------------------------------->
```

Referências Bibliográficas

[Bassalo 1996]BASSALO, J. M. F. *Nascimentos da Física*. [S.l.: s.n.], 1996.

.