Programação de Computadores - CI-208

Prof. Murilo da Silva - DINF/UFPR

Aula 14

Problema: Seu programa deve ler uma sequência de números n (até que n=0) e fazer o seguinte:

Para cada n, deve executar uma, dentre 3 operações possíveis:

1: raíz quadrada de n
2: raíz cúbica de n
3: calcular exp(n)

Atenção: a operação escolhida deve ser indicada pelo código 1, 2 ou 3. Se o código digitado for incorreto, o programa deve ler código novamente.

Problema: Seu programa deve ler uma sequência de números n (até que n=0) e fazer o seguinte:

Para cada n, deve executar uma, dentre 3 operações possíveis:

1: raíz quadrada de n
2: raíz cúbica de n
3: calcular exp(n)

Atenção: a operação escolhida deve ser indicada pelo código 1, 2 ou 3. Se o código digitado for incorreto, o programa deve ler código novamente.

```
/* Programa 'menuCodigo' */
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int cod;  float n, x;

    cin >> n;
    while ( n != 0 ) {
        cin >> cod;
        while ( cod <= 0 || cod >= 4 )
            cin >> cod;

        if ( cod == 1 )          x = sqrt(n);
        else if ( cod == 2 )     x = cbrt(n);
        else if ( cod == 3 )     x = exp(n);
        cout << x  << endl;

        cin >> n;
    }

    return 0;
}
```

Duas repetições aninhadas baseadas em entrada do usuário.

Motivação

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

int lerCodigo ( ) {
      int code;
    cin >> cod;
    while (cod<=0||cod>=4)
        cin >> cod;
    return code;
}

int main( ) {
   int op;  float n, x;

   cin >> n;
   while ( n != 0 ) {

     op = lerCodigo ( );

     if ( op == 1 )        x = sqrt (n);
     else if ( op == 2 )   x = cbrt (n);
     else if ( op == 3 )   x = exp (n);
     cout << x << endl;

      cin >> n;
   }

   return 0;
}
```

Definir uma **FUNÇÃO** responsável por **APENAS** obter um código válido do usuário

O resultado gerado pela execução da função é DEVOLVIDO para quem USA a função

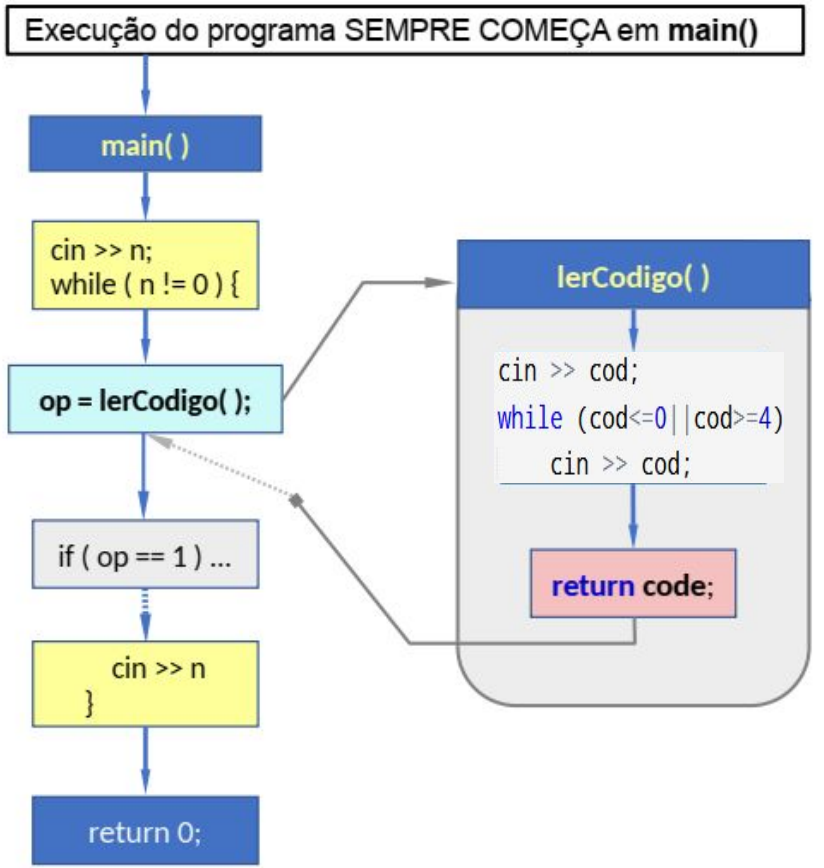
A EXECUÇÃO do programa SEMPRE começa em **main().**

→ A execução é desviada para a função.
→ Terminada a execução da função
→ resultado é devolvido à atribuição
→ Só então a atribuição é executada

Se escreve o programa principal sem se preocupar com detalhes do código da função. Basta saber o que ela faz e o que devolve como resultado.

Como acontece a execução?

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

int lerCodigo ( ) {
      int code;
    cin >> cod;
    while (cod<=0||cod>=4)
        cin >> cod;
     return code;
}

int main( ) {
   int op;  float n, x;

   cin >> n;
   while ( n != 0 ) {
      op = lerCodigo ( );
      if ( op == 1 )        x = sqrt (n);
      else if ( op == 2 )   x = cbrt (n);
      else if ( op == 3 )   x = exp (n);
      cout << x << endl;
      cin >> n;
   }
   return 0;
}
```



Definição de função simples

- Forma geral de **definição**:

```
cabeçalho
{
    declarações de variáveis;
    comandos ;
    return valor_retorno;
}
```

  - Bloco de comandos (entre **{ }**) → código que é executado pela função

- Para funções simples, que apenas devolvem um resultado:
  - **cabeçalho: tipo_retorno** `nomeFuncao ( )`
    - **tipo_retorno:** indica o tipo de **valor_retorno**
      - tipo do resultado que a função deve retornar ao final de sua execução
        - comando `return` indica final da execução da função e o valor a ser devolvido por ela → **valor de retorno**

    - Uma função definida nesta forma simples não recebe dados:
      - É o que indicam os **( )** sem nada entre eles.

Onde definir uma função no código?

- Função pode ser **definida** antes ou depois de **main()**:
  - Se definida **antes**, não há passo adicional;
  - Se definida **depois**, é preciso **declarar** o **protótipo da função** antes de **main()**:
    - O **protótipo** consiste em escrever o mesmo cabeçalho de definição da função seguido apenas de **ponto-e-vírgula**.
    - O importante é que ou a **definição** ou **protótipo** de uma função estejam escritas no código-fonte ANTES da PRIMEIRA VEZ que ela é **referenciada** no código-fonte de todo o programa.

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

// Definição da função lerCodigo()
int lerCodigo ( ) {
      int code;
     cin >> cod;
     while (cod<=0||cod>=4)
         cin >> cod;
      return code;
}

int main( ) {
   int op;  float n, x;

   cin >> n;
   while ( n != 0 ) {

     op = lerCodigo ( );

     if ( op == 1 )        x = sqrt (n);
     else if ( op == 2 )   x = cbrt (n);
     else if ( op == 3 )   x = exp (n);
     cout << x << endl;
     cin >> n;
   }
   return 0;
}
```

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

// Protótipo da função lerCodigo()
int lerCodigo ( );

int main( ) {
   int op;  float n, x;

   cin >> n;
   while ( n != 0 ) {
     op = lerCodigo ( );
     if ( op == 1 )        x = sqrt (n);
     else if ( op == 2 )   x = cbrt (n);
     else if ( op == 3 )   x = exp (n);
     cout << x << endl;
     cin >> n;
   }
   return 0;
}

// Definição da função lerCodigo()
int lerCodigo ( ) {
      int code;
     cin >> cod;
     while (cod<=0||cod>=4)
         cin >> cod;
      return code;
}
```

O comando **return**

- O comando **return** tem um duplo objetivo:
  - Sinalizar que a execução da função terminará:
    - Nenhum comando escrito após um **return** será executado
    - A execução do **programa** continua no ponto onde a função foi **usada**.
  - Devolver um valor a quem chamou a função
    - valor deve ter o mesmo **tipo_retorno** definido no **cabeçalho** da função
    - Somente é possível devolver **UM ÚNICO VALOR**
- **Não se retorna valores com cout**
  - **cout** serve para saída de dados para usuário
  - **return** é usado para devolver um valor de função para o programa

Uso (Chamada) de uma função

- Como o programa usa ou invoca a execução de uma função?
  - Através da **chamada**
    - Para uma funçao simples:

```
nomeFuncao ( )
```

  - A chamada da função pode ser feita em qualquer ponto de um **programa** onde seu valor de retorno deva ser usado:
    - op = **codigoOp( )**;
      → O **valor retornado** pela função será atribuído à variável **op**
    - if (**codigoOp( )** == 1) x = x + 3;
      → Se o **valor retornado** pela função é igual a 1, somar 3 a **x**
    - y = **codigoOp( )** + x;
      → Soma com **x** o **valor retornado** pela função e atribui resultado a **y**

Funções e variáveis

- Variáveis de funções diferentes estão em locais diferentes na memória
  - mesmo que tenham o mesmo nome
- Variáveis de uma função são **locais** a esta função
  - São válida APENAS dento do bloco de comandos da função
  - Não são visíveis em outra função

```
/* Programa 'menuOpcoes' */
#include <iostream>
using namespace std;

int func ( ) {

        int code;
        .......
        code = op;  // ERRO: op não foi declarada aqui
                    // e NÃO SE REFERE à variável
                    // declarada em main()
        .......
        return code;
}

int main( ) {

    int  op;
    .......
    op = func ();
    return 0;
}
```

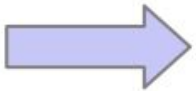
```
/* Programa 'menuOpcoes' */
#include <iostream>
using namespace std;

int func ( ) {

        int code, op;
        .....
        op = 23;  // op NÃO SE REFERE à variável
                  // declarada em main()
        cout  << op << endl;   // imprime 23
        return code;
}

int main( ) {

    int x, op; // op NÃO SE REFERE à variável declarada em func()

    op = 37;
    x = func ();
    cout << op << endl;   // imprime 37
    return 0;
}
```

A função **main()**

- O programa principal **main()** é uma função especial
  - possui um tipo de retorno fixo → **int**
  - é invocada automaticamente quando se inicia a execução do programa
- O comando **return** que se coloca ao final do bloco do **main()** informa ao Sistema Operacional (SO) se o programa terminou corretamente ou não.
  - Por convenção:
    - **return 0** (valor zero) indica que tudo correu bem,
    - qualquer valor diferente de zero indica ao SO que houve erro no programa.

Parâmetros de funções

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

int lerCodigo ( ) {
      int code;

     cin >> cod;
     while (cod<=0||cod>=4)
         cin >> cod;
     return code;
}

int main( ) {
    int op;  float n, x;

    cin >> n;
    while ( n != 0 ) {
      op = lerCodigo ( );

      if ( op == 1 )          x = sqrt (n);
      else if ( op == 2 )   x = cbrt (n);
      else if ( op == 3 )   x = exp (n);

      cout << x << endl;
      cin >> n;
    }
    return 0;
}
```



```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

int lerCodigo ( ) {
      int code;
      cin >> cod;
      while (cod<=0||cod>=4)
          cin >> cod;
      return code;
}

float calcOp ( float v, int op ) {
      float res;

      if ( op == 1 )          res = sqrt ( v );
      else if ( op == 2 )   res = cbrt ( v );
      else if ( op == 3 )   res = exp ( v );
      return res;
}

int main( ) {
    int op;  float n, x;

    cin >> n;
    while ( n != 0 ) {
      op = lerCodigo ( );

      x = calcOp ( n, op );

      cout << x << endl;
      cin >> n;
    }
    return 0;
}
```

Definição de função

- Formato geral:

```
tipo_retorno nomeFuncao ( lista_parâmetros )
{
   declarações de variáveis;
   comandos;
   return valor_retorno;
}
```

  - **tipo_retorno:** indica o tipo de **valor_retorno**
    - resultado que a função deve retornar ao final de sua execução

  - **lista_parâmetros:** lista de **declarações de variáveis** usadas para passagem de valores para a função.
    - variáveis separadas por **vírgula**, cada variável com seu **tipo**.

      tipo1 param_1 , tipo2 param_2 , ...

    - Em funções simples que não recebem parâmetros, esta lista é **vazia**

Definição de função

- Exemplo:

```
float calcOp ( float n, int op )
```

  - ao ser chamada, esta função espera receber 2 valores nas duas variáveis:

    n → um valor real
    op → um valor inteiro

  - estas variáveis serão usadas dentro do bloco de comandos da função

- Os parâmetros são **variáveis locais** da função

  - Recebem um valor inicial quando a função é **chamada** durante a execução do programa

Protótipos de funções com parâmetros

- O **protótipo** consiste em escrever o mesmo cabeçalho de definição da função seguido apenas de **ponto-e-vírgula**.

- O importante é que ou a **definição** ou **protótipo** de uma função estejam escritas no código-fonte ANTES da PRIMEIRA VEZ que ela é **referenciada** no código-fonte de todo o programa.

- No **protótipo**, pode-se apenas colocar os tipos dos parâmetros, sem colocar os nomes dos parâmetros:

```
float calcOp ( float n, int op );

OU

float calcOp ( float, int );
```

Definição de função

- Os parâmetros são **variáveis locais** da função
  - **Não devem** ser re-declaradas dentro do bloco da função

```
int somaValor ( int qt )
{
      int  res, i, qt;   // ERRO !!! Não se re-declara parâmetro de função

      i=0;
      res = 0;
      while (i < qt) {
         res = res + i;
         i = i+1;
      }

      return res;
}
```

Chamada de uma função

- Como o programa usa ou executa uma função?
  - Através da **chamada:**

```
nomeFuncao ( lista_argumentos )
```

    - **lista_argumentos:** lista de variáveis / valores / expressões cujo **valor** será atribuído aos parâmetros da função
      - **argumentos** são separados por **vírgula**

arg_1, arg_2, …

Chamada de uma função

- **Não se colocam os tipos em argumentos na chamada**
  - **Apenas** o nome da função chamada, seguida da lista de **argumentos** entre parênteses, **sem especificar tipos**

```
int somaValor ( int qt ) {
        int  res, i;
        I=0;
        res = 0;
        while (i < qt) {
            res = res + i;   i=i+1;
        }
        return res;
}

int main() {
      int v, x, n;
      .....
      n = somaValor ( v );          // Forma de chamada da função → OK.
      .......
      x = int somaValor ( int v);   // ERRO !!! Não se colocam tipos na chamada
      ......
      return 0;
}
```

Chamada de funções

- Ao **chamar** uma função, cada **argumento** deve ser uma variável/valor/expressão do **mesmo tipo** do respectivo **parâmetro** na **definição** da função.

- Associação argumento-parâmetro é feita pela **posição**:
  - **Valor** do 1º argumento é copiado para o 1º parâmetro,
  - **Valor** do 2º argumento é copiado para o 2º parâmetro,
  - e assim por diante

```
float calcOp ( float n, int op ) { … }

int main () {
   float x, n;
   int opt;
   x = calcOp ( n, opt );
   x = calcOp ( n + x, 2 );
   x = calcOp ( 1.73, 3 );
   …
}
```

Erros comuns em chamada de funções

```
int somaValor ( int qt, int p ) {
        int  res, i;
        i=0;
        res = p;
        while (i < qt) {
            res = res + i;   i=i+1;
        }
        return res;
}

int main() {
      int quant, x, n;
      float y;
      …..
      x = somaValor ( quant, n );       // Forma de chamada da função → OK.
      …….
      x = somaValor ( y , n );          // ERRO !!! Tipo do 1º argumento (float)
      ……                                // não é o mesmo do 1º parâmetro da função (int)
      x = somaValor ( n  ,  quant );      // ATENÇÃO !!! Os dois argumentos são do tipo certo
      ……                                // mas a ordem pode estar errada.
      return 0;                         // O programa terá erro ao executar !!
}
```

Mecanismo da passagem de parâmetros **por valor**

```
/* Programa 'menuOpcoes' */
#include <iostream>
#include <cmath>
using namespace std;

int lerCodigo ( ) { ... }

float calcOp ( float v, int op ) {
      float res;

      if ( op == 1 )        res = sqrt ( v );
      else if ( op == 2 )   res = cbrt ( v );
      else if ( op == 3 )   res = exp ( v );

      return res;
}

int main( ) {
    int op;  float n, x;

    cin >> n;
    while ( n != 0 ) {
      op = lerCodigo ( );

      x = calcOp ( n, op );

      cout << x << endl;
      cin >> n;
    }
    return 0;
}
```