Linguagens de Programação

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

Parte II

Introdução à Programação em C++
(Continuação)

Relembrando da Última Aula...

- Classes e objetos
- Mais exemplos de programação orientada a objetos

Variáveis Não-inicializadas ou Truncadas

- Variáveis não inicializadas
  - Contém valores indefinidos
    - Não inicializar contadores e acumuladores pode provocar erros de lógica
- Divisão de inteiros e truncamento
  - Divisão de inteiros
    - Divisão de dois inteiros leva a resultado truncado
      - Uma fração do quociente resultante é perdida
      - Assumir que a divisão de inteiros arredonda (em lugar de truncar) pode gerar resultados incorretos

Números de Ponto Flutuante

- Um número real com um ponto decimal
- C++ fornece os tipos de dados `float` e `double`
  - Os números `double` podem ter maior magnitude e maior precisão
  - Valores de ponto flutuante são tratados como valores `double` por padrão
    - Ex.: Resultado da divisão de dois inteiros
- Valores de ponto flutuante em geral são apenas aproximações

Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 2
 * Arquivo: GradeBookCap5Ex2.h
 * Autor: Miguel Campista
*/

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            // Construtor inicializa courseName com a string-argumento
            GradeBook(string);
            // Função que configura o nome do curso
            void setCourseName(string);
            // Função que obtém o nome do curso
            string getCourseName();
            // Calcula a media das notas inseridas pelos usuários
            void determineMedia();
            void displayMessage();

    private:
            string courseName;
};
```

Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 2
 * Arquivo: GradeBookCap5Ex2.cpp
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include <iomanip>
#include "GradeBookCap5Ex2.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
```


Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 2
 * Arquivo: GradeBookCap5Ex2.cpp
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include <iomanip>
#include "GradeBookCap5Ex2.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
```

Biblioteca para definir a precisão da saída numérica


```
// Calcula a media das notas inseridas pelos usuários
void GradeBook::determineMedia() {
      int total, gradeCounter, grade;
      double average;

      total = 0;
      gradeCounter = 0;

      cout << "Entre com a nota ou -1 para sair: ";
      cin >> grade;

      while (grade != -1) {
            total = total + grade;
            gradeCounter = gradeCounter + 1;
            cout << "Entre com a nota ou -1 para sair: ";
            cin >> grade;
      }

      if (gradeCounter > 0) {
            average = static_cast<double> (total) / gradeCounter;
            cout << "Total das " << gradeCounter << " notas eh: " << total << endl;
            cout << "Media eh: " << setprecision(2) << fixed << average << endl;
      } else {
            cout << "Nenhuma nota foi atribuida..." << endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
            << getCourseName() << "!" << endl;
}
```

```
// Calcula a media das notas inseridas pelos usuários
void GradeBook::determineMedia() {
      int total, gradeCounter, grade;
      double average;

      total = 0;
      gradeCounter = 0;

      cout << "Entre com a nota ou -1 para sair: ";
      cin >> grade;

      while (grade != -1) {
            total = total + grade;
            gradeCounter = gradeCounter + 1;
            cout << "Entre com a nota ou -1 para sair: ";
            cin >> grade;
      }

      if (gradeCounter > 0) {
            average = static_cast<double> (total) / gradeCounter;
            cout << "Total das " << gradeCounter << " notas eh: " << total << endl;
            cout << "Media eh: " << setprecision(2) << fixed << average << endl;
      } else {
            cout << "Nenhuma nota foi atribuida..." << endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
            << getCourseName() << "!" << endl;
}
```

A variável `average` agora é double

O `while` executa desde que `grade` não seja igual ao valor de sentinela -1

```
// Calcula a media das notas inseridas pelos usuários
void GradeBook::determineMedia() {
      int total, gradeCounter, grade;
      double average;

      total = 0;
      gradeCounter = 0;

      cout << "Entre com a nota ou -1 para sair: ";
      cin >> grade;

      while (grade != -1) {
            total = total + grade;
            gradeCounter = gradeCounter + 1;
            cout << "Entre com a nota ou -1 para sair: ";
            cin >> grade;
      }

      if (gradeCounter > 0) {
            average = static_cast<double> (total) / gradeCounter;
            cout << "Total das " << gradeCounter << " notas eh: " << total << endl;
            cout << "Media eh: " << setprecision(2) << fixed << average << endl;
      } else {
            cout << "Nenhuma nota foi atribuida..." << endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
            << getCourseName() << "!" << endl;
}
```

Calcula a nota média usando `static_cast<double>` para executar uma conversão explícita da variável total. No resultado da divisão prevalece o tipo `double`

```
// Calcula a media das notas inseridas pelos usuários
void GradeBook::determineMedia() {
      int total, gradeCounter, grade;
      double average;

      total = 0;
      gradeCounter = 0;

      cout << "Entre com a nota ou -1 para sair: ";
      cin >> grade;

      while (grade != -1) {
            total = total + grade;
            gradeCounter = gradeCounter + 1;
            cout << "Entre com a nota ou -1 para sair: ";
            cin >> grade;
      }

      if (gradeCounter > 0) {
            average = static_cast<double> (total) / gradeCounter;
            cout << "Total das " << gradeCounter << " notas eh: " << total << endl;
            cout << "Media eh: " << setprecision(2) << fixed << average << endl;
      } else {
            cout << "Nenhuma nota foi atribuida..." << endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
            << getCourseName() << "!" << endl;
}
```

Define a precisão do número de ponto flutuante

Fixa o número de casas decimais que são impressas

Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 2
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include "GradeBookCap5Ex2.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao de Computadores e Sistemas Distribuidos");
    GradeBook gradeBook2("Compl");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    gradeBook1.determineMedia();
    gradeBook1.determineMedia();

    return 0;
}
```


Exemplo utilizando Classes em

```
shell>$ g++ -c gradebook.cpp –o gradebook.o
shell>$ g++ -c principal.cpp –o principal.o
shell>$ g++ -o ex10 gradebook.o principal.o
shell>$ ./ex10
Warning: Nome "Programacao de Computadores e Sistemas Distribuidos" excede o limite
maximo de 25 caracteres...
Nome limitado aos primeiros 25 caracteres: Programacao de Computador
Nome do curso 1 eh: Programacao de Computador
Nome do curso 2 eh: Compl
Entre com a nota ou -1 para sair: 2
Entre com a nota ou -1 para sair: 1
Entre com a nota ou -1 para sair: 1
Entre com a nota ou -1 para sair: -1
Total das 5 notas eh: 4
Media eh: 1.33
Entre com a nota ou -1 para sair: 2
Entre com a nota ou -1 para sair: 2
Entre com a nota ou -1 para sair: 2
Entre com a nota ou -1 para sair: -1
Total das 5 notas eh: 6
Media eh: 2.00
shell>$
```

Conversão de Tipos

- Operador de Coerção Unário
  - Cria uma cópia temporária de seu operando com um tipo de dado diferente
    - Conversão explícita
      - Converte tipos numéricos e tipos de classes relacionados (polimorfismo)
  - Ex.: `static_cast< double > ( total )`
    - Cria uma cópia do ponto flutuante temporária de `total`
- Promoção → Conversão implícita
  - Conversão de um valor (p. ex., `int`) em outro tipo de dado (p. ex., `double`) para realizar um cálculo

Formatação de Números de Ponto Flutuante

- Manipulador de fluxo parametrizado **`setprecision`**
  - Especifica o número de dígitos de precisão
    - Junto com o **`fixed`** está relacionado com o número de casas decimais
  - A precisão-padrão é de seis dígitos
- Manipulador de fluxo não parametrizado **`fixed`**
  - Indica que os valores de ponto flutuante devem ser enviados para a saída no formato de ponto fixo
    - Em oposição à notação científica (3.1×10^3)
- Manipulador de fluxo **`showpoint`**
  - Força a exibição do ponto decimal

Leitura de Caracteres do Teclado

- Uso da função **`cin.get()`**
  - Função `istream::get` → `int get()`
    - Função lê **um caractere** do teclado e retorna o valor lido
    - O valor de retorno pode ser armazenado também em uma variável **`int`**

```
cout << "O caractere (" << 'a' << ") tem valor "
     << static_cast< int > ( 'a' ) << endl;
```

```
O caractere (a) tem valor 97
```


Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 6
 * Arquivo: GradeBookCap5Ex5.h
 * Autor: Miguel Campista
 */

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string);
        // Função que configura o nome do curso
        void setCourseName(string);
        // Função que obtém o nome do curso
        string getCourseName();
        // Função para dar entrada nos conceitos dos alunos
        void entradaNotas();
        // Função para exibir os conceitos
        void displaySumarioNotas();
        void displayMessage();

    private:
        string courseName;
        int aCount, bCount, cCount;
};
```

Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 5
 * Arquivo: GradeBookCap5Ex5.h
 * Autor: Miguel Campista
 */

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string);
        // Função que configura o nome do curso
        void setCourseName(string);
        // Função que obtém o nome do curso
        string getCourseName();
        // Função para entrada de notas nos alunos
        void entradaNotas();
        // Função para exibir os conceitos
        void displaySumarioNotas();
        void displayMessage();

    private:
        string courseName;
        int aCount, bCount, cCount;
};
```

Função para entrada de notas pelo teclado

Função para exibir um sumário das notas

Contadores para contabilizar os conceitos

```
/*
 * Aula 5 -- Exemplo 5
 * Arquivo: GradeBookCap5Ex5.cpp
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include "GradeBookCap5Ex5.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
      aCount = bCount = cCount = 0;
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

```
/*
 * Aula 5 -- Exemplo 5
 * Arquivo: GradeBookCap5Ex5.cpp
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include "GradeBookCap5Ex5.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
      aCount = bCount = cCount = 0;
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

Inicialização das variáveis no construtor

```
// Função para dar entrada nos conceitos dos alunos
void GradeBook::entradaNotas() {
      int grade;

      cout << "Entre com o conceito." << endl
           << "Entre com o caractere EOF para finalizar." << endl;

      // Repetição até o usuário entrar com o EOF
      while ( (grade = cin.get()) != EOF ) {

            switch (grade) {
                  case 'A':
                  case 'a':
                        aCount++;
                        break;
                  case 'B':
                  case 'b':
                        bCount++;
                        break;
                  case 'C':
                  case 'c':
                        cCount++;
                        break;
                  case '\n': // ignora newlines,
                  case '\t': // tabs
                  case ' ': // e espaços na entrada
                        break; // exit switch
                  default:
                        cout << "Conceito desconhecido." << endl
                             << "Insira um novo conceito." << endl;
            }
      }
}
```

```
// Função para dar entrada nos conceitos dos alunos
void GradeBook::entradaNotas() {
      int grade;

      cout << "Entre com o conceito." << endl
           << "Entre com o caractere EOF para finalizar." << endl;

      // Repetição até o usuário entrar com o EOF
      while ( (grade = cin.get()) != EOF ) {

            switch (grade) {
                  case 'A':
                  case 'a':
                        aCount++;
                        break;
                  case 'B':
                  case 'b':
                        bCount++;
                        break;
                  case 'C':
                  case 'c':
                        cCount++;
                        break;
                  case '\n': // ignora newlines,
                  case '\t': // tabs
                  case ' ': // e espaços na entrada
                        break; // exit switch
                  default:
                        cout << "Conceito desconhecido." << endl
                             << "Insira um novo conceito." << endl;
            }
      }
}
```

Uso da função **`cin.get()`** para leitura de caractere do teclado. O caractere lido é atribuído à variável **`grade`**

```
// Função para dar entrada nos conceitos dos alunos
void GradeBook::entradaNotas() {
      int grade;

      cout << "Entre com o conceito." << endl
           << "Entre com o caractere EOF para finalizar." << endl;

      // Repetição até o usuário entrar com o EOF
      while ( (grade = cin.get()) != EOF ) {

            switch (grade) {
                  case 'A':
                  case 'a':
                        aCount++;
                        break;
                  case 'B':
                  case 'b':
                        bCount++;
                        break;
                  case 'C':
                  case 'c':
                        cCount++;
                        break;
                  case '\n': // ignora newlines,
                  case '\t': // tabs
                  case ' ': // e espaços na entrada
                        break; // exit switch
                  default:
                        cout << "Conceito desconhecido." << endl
                             << "Insira um novo conceito." << endl;
            }
      }
}
```

Após a inicialização da variável **`grade`**, ela é comparada ao **`EOF`**. Em sistemas UNIX, o **`EOF`** pode ser um **`Ctrl+d`** e em WINDOWS, o **`Ctrl+z`**

```
// Função para dar entrada nos conceitos dos alunos
void GradeBook::entradaNotas() {
     int grade;

     cout << "Entre com o conceito." << endl
          << "Entre com o caractere EOF para finalizar." << endl;

     // Repetição até o usuário entrar com o EOF
     while ( (grade = cin.get()) != EOF ) {

          switch (grade) {
               case 'A':
               case 'a':
                    aCount++;
                    break;
               case 'B':
               case 'b':
                    bCount++;
                    break;
               case 'C':
               case 'c':
                    cCount++;
                    break;
               case '\n': // ignora newlines,
               case '\t': // tabs
               case ' ': // e espaços na entrada
                    break; // exit switch
               default:
                    cout << "Conceito desconhecido." << endl
                         << "Insira um novo conceito." << endl;
          }
     }
}
```

Entradas podem ser em letras maiúsculas ou minúsculas

```
// Função para dar entrada nos conceitos dos alunos
void GradeBook::entradaNotas() {
     int grade;

     cout << "Entre com o conceito." << endl
          << "Entre com o caractere EOF para finalizar." << endl;

     // Repetição até o usuário entrar com o EOF
     while ( (grade = cin.get()) != EOF ) {

          switch (grade) {
               case 'A':
               case 'a':
                    aCount++;
                    break;
               case 'B':
               case 'b':
                    bCount++;
                    break;
               case 'C':
               case 'c':
                    cCount++;
                    break;
               case '\n': // ignora newlines,
               case '\t': // tabs
               case ' ': // e espaços na entrada
                    break; // exit switch
               default:
                    cout << "Conceito desconhecido." << endl
                         << "Insira um novo conceito." << endl;
          }
     }
}
```

Evita que caracteres diferentes dos permitidos sejam considerados pelo programa. Ex.: ao teclar **ENTER** um caractere especial é lido do teclado

Exemplo utilizando Classes em C++

```
// Função para exibir os conceitos
void GradeBook::displaySumarioNotas() {
     cout << "Numero de alunos que receberam cada um dos conceitos:"
          << "\nA: " << aCount
          << "\nB: " << bCount
          << "\nC: " << cCount
          << endl;
}
```


Exemplo utilizando Classes em C++

```
/*
 * Aula 5 -- Exemplo 5
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include <string>
#include "GradeBookCap5Ex5.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao de Computadores e Sistemas Distribuidos");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    gradeBook1.entradaNotas();
    gradeBook1.displaySumarioNotas();

    return 0;
}
```


Exemplo utilizando Classes em

```
shell>$ g++ -c gradebook.cpp –o gradebook.o
shell>$ g++ -c principal.cpp –o principal.o
shell>$ g++ -o ex13 gradebook.o principal.o
shell>$ ./ex13
Warning: Nome “Programacao de Computadores e Sistemas Distribuidos” excede o limite
maximo de 25 caracteres...
Nome limitado aos primeiros 25 caracteres: Programacao de Computador
Nome do curso 1 eh: Programacao de Computador
Nome do curso 2 eh: Comp1
Entre com o conceito.
Entre com o caractere EOF para finalizar.
a
a
a
b
b
c
^Z
Numero de alunos que receberam cada um dos conceitos:
A: 3
B: 2
C: 1
shell>$
```


Operadores Lógicos

- And (&&), Or (||), Not (!)
- Manipulador de fluxo **boolalpha**
  - Exibe o valor de cada expressão booleana
    - **true** ao invés de 1
    - **false** ao invés de 0

```
/*
 * Aula 5 -- Exemplo 6
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {

    // cria uma tabela-verdade para o operador && (AND lógico)
    cout << boolalpha << "Logico AND (&&)"
         << "\nfalso && falso: " << ( false && false )
         << "\nfalso && verdadeiro: " << ( false && true )
         << "\nverdadeiro && falso: " << ( true && false )
         << "\nverdadeiro && verdadeiro: " << ( true && true ) << "\n\n";

    // cria uma tabela-verdade para o operador || (OR lógico)
    cout << "Logico OR (||)"
         << "\nfalso || falso: " << ( false || false )
         << "\nfalso || verdadeiro: " << ( false || true )
         << "\nverdadeiro || falso: " << ( true || false )
         << "\nverdadeiro || verdadeiro: " << ( true || true ) << "\n\n";

    // cria uma tabela-verdade para o operador ! (NOT lógico)
    cout << "Logico NOT (!)"
         << "\n!falso: " << ( !false )
         << "\n!verdadeiro: " << ( !true ) << endl;

    return 0;
}
```

```
/*
 * Aula 5 -- Exemplo 6
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {

    // cria uma tabela-verdade para o operador && (AND lógico)
    cout << boolalpha << "Logico AND (&&)"
         << "\nfalso && falso: " << ( false && false )
         << "\nfalso && verdadeiro: " << ( false && true )
         << "\nverdadeiro && falso: " << ( true && false )
         << "\nverdadeiro && verdadeiro: " << ( true && true ) << "\n\n";

    // cria uma tabela-verdade para o operador || (OR lógico)
    cout << "Logico OR (||)"
         << "\nfalso || falso: " << ( false || false )
         << "\nfalso || verdadeiro: " << ( false || true )
         << "\nverdadeiro || falso: " << ( true || false )
         << "\nverdadeiro || verdadeiro: " << ( true || true ) << "\n\n";

    // cria uma tabela-verdade para o operador ! (NOT lógico)
    cout << "Logico NOT (!)"
         << "\n!falso: " << ( !false )
         << "\n!verdadeiro: " << ( !true ) << endl;
```

Uso do manipulador de fluxo **boolalpha**

```
/*
 * Aula 5 -- Exemplo 6
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {

    // cria uma tabela-verdade para o operador && (AND lógico)
    cout << boolalpha << "Logico AND (&&)"
         << "\nfalso && falso: " << ( false && false )
         << "\nfalso && verdadeiro: " << ( false && true )
         << "\nverdadeiro && falso: " << ( true && false )
         << "\nverdadeiro && verdadeiro: " << ( true && true ) << "\n\n";
```

```
shell>$ g++ gradebook.cpp –o ex14
shell>$ ./ex14
Logico AND (&&)
falso && falso: false
falso && verdadeiro: false
verdadeiro && falso: false
verdadeiro && verdadeiro: true

Logico OR (||)
falso || falso: false
falso || verdadeiro: true
verdadeiro || falso: true
verdadeiro || verdadeiro: true

Logico NOT (!)
!falso: true
!verdadeiro: false
shell>$
```

```
    // cria uma tabela-verdade para o operador || (OR lógico)
    cout << "Logico OR (||)"
         << "\nfalso || falso: " << ( false || false )
         << "\nfalso || verdadeiro: " << ( false || true )
         << "\nverdadeiro || falso: " << ( true || false )
         << "\nverdadeiro || verdadeiro: " << ( true || true ) << "\n\n";

    // cria uma tabela-verdade para o operador ! (NOT lógico)
    cout << "Logico NOT (!)"
         << "\n!falso: " << ( !false )
         << "\n!verdadeiro: " << ( !true ) << endl;

    return 0;
}
```

Exemplo 1: Cadastro

- Escreva uma agenda em C++ para armazenar em memória três cadastros contendo nome, telefone e endereço. Cada um dos cadastros deve ser um objeto da classe Cadastro. A classe Cadastro ainda deve oferecer uma função para exibição dos dados de cada cadastro.

?

Exemplo 1: Cadastro

```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo: cadastro.h
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

class Cadastro {
      public:
            // Contrutor da classe Cadastro recebe 3 parâmetros
            Cadastro(string, string, string);
            // Função de exibição de todos os cadastros
            void displayData();

      private:
            string nomeCad, telCad, endCad;
            // Função para exibir o nome do cadastro
            string getNome();
            // Função para exibir o telefone do cadastro
            string getTel();
            // Função para exibir o endereço do cadastro
            string getEnd();
};
```



```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo: cadastro.cpp
 * Autor: Miguel Campista
 */
#include "cadastro.h"

// Contrutor da classe Cadastro recebe 3 parâmetros
Cadastro::Cadastro(string nome, string tel, string endereco) {
       nomeCad = nome;
       telCad = tel;
       endCad = endereco;
}
// Função de exibição de todos os cadastros
void Cadastro::displayData() {
        cout << left << setw(25) << getNome()
             << setw(10) << getTel()
             << setw(25) << getEnd() << endl;
}
// Função para exibir o nome do cadastro
string Cadastro::getNome() {
       return nomeCad;
}
// Função para exibir o telefone do cadastro
string Cadastro::getTel() {
       return telCad;
}
// Função para exibir o endereço do cadastro
string Cadastro::getEnd() {
       return endCad;
}
```

Exemplo 1: Cadastro

```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
*/
#include <iostream>
#include <string>
#include "cadastro.h"

using namespace std;

int main() {

    int ncad = 3;
    string nome[ncad], tel[ncad], endereco[ncad];

    for (int i = 0; i < ncad; i++) {
          cout << "Entre com o novo cadastro:" << endl;
          cout << "Nome: ";
          getline(cin, nome[i]);
          cout << "Telefone: ";
          getline(cin, tel[i]);
          cout << "Endereco: ";
          getline(cin, endereco[i]);
    }

    Cadastro cad0(nome[0], tel[0], endereco[0]);
    Cadastro cad1(nome[1], tel[1], endereco[1]);
    Cadastro cad2(nome[2], tel[2], endereco[2]);
```


Exemplo 1: Cadastro

```
    cout << left << setw(25) << "Nome:"
         << setw(10) << "Tel:"
         << setw(25) << "End:" << endl;
    cad0.displayData();
    cad1.displayData();
    cad2.displayData();

    return 0;
}
```


Exemplo 2: Agenda

- Escreva uma agenda em C++ para armazenar em memória três cadastros contendo nome, telefone e endereço. Uma classe Agenda deve ser criada e nela três cadastros devem ser inseridos. Cada cadastro é um objeto da classe Cadastro.

?

Exemplo 2: Agenda

```
/*
 * Aula 5 -- Exemplo 8
 * Arquivo Principal
 * Autor: Miguel Campista
*/
#include <iostream>
#include <string>
#include "agenda-ex8.h"

using namespace std;

int main() {
    Agenda agenda;
    int ncad = 3;
    string nome, tel, endereco;

    for (int i = 0; i < ncad; i++) {
        cout << "Entre com o novo cadastro:" << endl;
        cout << "Nome: ";
        getline(cin, nome);
        cout << "Telefone: ";
        getline(cin, tel);
        cout << "Endereco: ";
        getline(cin, endereco);
        // Inserção do cadastro na agenda
        agenda.insereCad(i, nome, tel, endereco);
    }
```


Exemplo 2: Agenda

```
    cout << "\n" << left << setw(25) << "Nome:"
         << setw(10) << "Tel:"
         << setw(25) << "End:" << endl;

    for (int i = 0; i < ncad; i++) {
        agenda.getCad(i);
    }

    return 0;
}
```


Exemplo 2: Agenda

```
/*
 * Aula 5 -- Exemplo 8
 * Arquivo: agenda-ex8.h
 * Autor: Miguel Campista
*/
#include <iomanip>
#include "cadastro-ex8.h"

using namespace std;

class Agenda {
    public:
        // Função para inserir os dados de cada cadastro
        void insereCad(int id, string, string, string);
        // Função para imprimir todos os dados relacionados com o cadastro
        void getCad(int);
    private:
        Cadastro cad0, cad1, cad2;
};
```

Exemplo 2: Agenda

```
/*
 * Aula 5 -- Exemplo 8
 * Arquivo: agenda-ex8.cpp
 * Autor: Miguel Campista
*/
#include "agenda-ex8.h"

// Função para inserir os dados de cada cadastro
void Agenda::insereCad(int id, string nome, string tel, string endereco) {
    switch (id) {
        case 0:
            cad0.insereDados(nome, tel, endereco);
            break;
        case 1:
            cad1.insereDados(nome, tel, endereco);
            break;
        case 2:
            cad2.insereDados(nome, tel, endereco);
    }
}
```


Exemplo 2: Agenda

```
// Função para imprimir todos os dados relacionados com o cadastro
void Agenda::getCad(int id) {
    switch (id) {
        case 0:
            cout << left << setw(25) << cad0.getNome()
                 << setw(10)  << cad0.getTel()
                 << setw(25)  << cad0.getEnd() << endl;
            break;
        case 1:
            cout << left << setw(25) << cad1.getNome()
                 << setw(10)  << cad1.getTel()
                 << setw(25)  << cad1.getEnd() << endl;
            break;
        case 2:
            cout << left << setw(25) << cad2.getNome()
                 << setw(10)  << cad2.getTel()
                 << setw(25)  << cad2.getEnd() << endl;
    }
}
```


Exemplo 2: Agenda

```
/*
 * Aula 5 -- Exemplo 8
 * Arquivo: cadastro-ex8.h
 * Autor: Miguel Campista
*/
#include <iostream>
#include <string>

using namespace std;

class Cadastro {
    public:
        // Função para inserção de dados em cada cadastro
        void insereDados(string, string, string);
        // Funções para obtenção dos dados de cada cadastro
        string getNome();
        string getTel();
        string getEnd();

    private:
        string nomeCad, telCad, endCad;
};
```


Exemplo 2: Agenda

```
/*
 * Aula 5 -- Exemplo 8
 * Arquivo: cadastro-ex8.cpp
 * Autor: Miguel Campista
*/
#include "cadastro-ex8.h"

// Função para inserção de dados em cada cadastro
void Cadastro::insereDados(string nome, string tel, string endereco) {
    nomeCad = nome;
    telCad = tel;
    endCad = endereco;
}
// Funções para obtenção dos dados de cada cadastro
string Cadastro::getNome() {
    return nomeCad;
}
string Cadastro::getTel() {
    return telCad;
}
string Cadastro::getEnd() {
    return endCad;
}
```


Exemplo 2: Agenda

Arquivo Makefile

```
# Comentario

CC = g++
LD = g++

CFLAGS = -Wall
LFLAGS = -Wall

EXEO1OBJS = cadastro-ex8.o agenda-ex8.o aula5-ex8.o

EXEMPLOS = ex8

.cpp.o:
        $(CC) $(CFLAGS) -c $<

all: $(EXEMPLOS)

ex8: $(EXEO1OBJS)
        $(LD) $(LFLAGS) -o $@ $(EXEO1OBJS) -lm

clean:
        rm -f *.o $(EXEMPLOS)
```


Funções

- Facilitam o projeto, a implementação, a operação e a manutenção de programas grandes
  - Podem empregar técnicas para redução do tempo de convergência
    - Técnica dividir para conquistar
      - Constrói um grande programa por meio de peças simples e pequenas
  - Funções da *C++ Standard Library*

Componentes de um Programa em C++

- *C++ Standard Library*
  - É uma coleção de funções para a execução de operações comuns como:
    - Cálculos matemáticos
    - Manipulação de strings
    - Manipulação de caracteres
    - Entrada/Saída
    - Verificação de erros
  - É fornecida como parte do ambiente de programação do *C++*

Componentes de um Programa em C++

- As funções devem ser limitadas à realização de uma única tarefa bem definida
  - Programas simples são mais fáceis de escrever, testar, depurar e manter
- Dica: O nome da função deve expressar essa tarefa efetivamente
  - Caso isso não seja possível, é provável que a função esteja tentando realizar um número muito grande de tarefas
    - Nesse caso, é melhor dividir essa função em funções menores

Funções da Biblioteca de Matemática

- Funções globais
  - Não pertencem a uma classe particular
  - Os protótipos de função são colocados nos arquivos de cabeçalho
    - Podem ser reutilizadas em qualquer programa que inclua o arquivo de cabeçalho e que possa se vincular ao código-objeto da função
    - Ex.: arquivo de cabeçalho `sqrt` in `<cmath>`
      - `sqrt( 900.0 )`
      - Todas as funções em `<cmath>` são funções globais

Protótipo de uma Função e Coerção de Argumentos

- Protótipo de função
  - Também chamado de declaração de função
  - Indica ao compilador:
    - O nome da função
    - O tipo de dados retornado à função
    - Os parâmetros que a função espera receber
      - O número de parâmetros
      - Os tipos de parâmetros
      - A ordem desses parâmetros

```
int soma(int a , int b);
```


Protótipo de uma Função e Coerção de Argumentos

- Assinatura de função (ou simplesmente assinatura)
  - Parte de um protótipo de função que inclui o nome da função e os respectivos tipos de argumento
    - Não especifica o tipo de retorno da função
  - As funções no mesmo escopo devem ter assinaturas exclusivas
    - O escopo de uma função é a região de um programa em que a função é conhecida e acessível

```
int soma(int a , int b);
          ≠
int soma(double a, double b);
```


Protótipo de uma Função e Coerção de Argumentos

- Assinatura de função (ou simplesmente assinatura)
  - Parte de um protótipo de função que inclui o nome da função e os respectivos tipos de argumento
    - Não especifica o tipo de retorno da função
  - As funções no mesmo escopo devem ter assinaturas exclusivas
    - O escopo de uma função é a região de um programa em que a função é conhecida e acessível

```
int soma(int a , int b);
          ≠
int soma(double a, double b);
```

Protótipo de uma Função e Coerção de Argumentos

- Assinatura de função (ou simplesmente assinatura)
  - É um erro de compilação se duas funções do mesmo escopo tiverem a mesma assinatura, mas diferentes tipos de retorno

```
int soma(int a , int b);
          =
void soma(int a, int b);
```

Assinaturas iguais

Protótipo de uma Função e Coerção de Argumentos

- Coerção de argumentos
  - Forçar argumentos aos tipos apropriados especificados pelos parâmetros correspondentes
    - Por exemplo, chamar uma função com um argumento inteiro, mesmo que o protótipo da função especifique um argumento `double`
      - A função ainda assim continuará a funcionar corretamente

```
int a, b;
double soma(double a , double b);
```


Protótipo de uma Função e Coerção de Argumentos

- Regras de promoção C++
  - A promoção ocorre também quando o tipo de um argumento de função não corresponde ao tipo de parâmetro especificado
    - A promoção é como se o valor do argumento tivesse sido atribuído diretamente ao tipo do parâmetro
  - A conversão de um valor em um tipo mais baixo
    - Pode provocar a perda de dados ou valores incorretos
    - Só pode ser executada explicitamente
      - Atribuindo o valor a uma variável ou tipo mais baixo (alguns compiladores emitirão um aviso nesse caso)

Protótipo de uma Função e Coerção de Argumentos

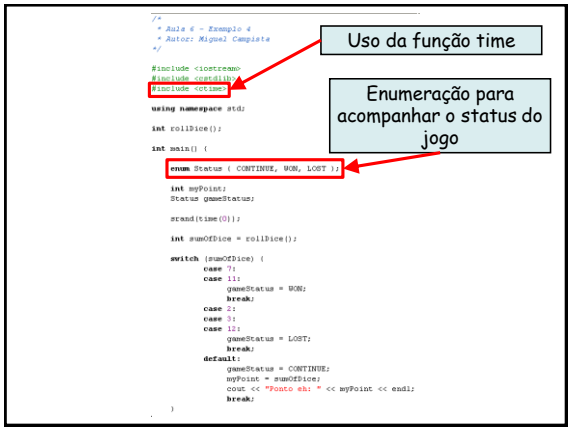
Tipos de dados
long double
double
float
unsigned long int (sinônimo de unsigned long)
long int (sinônimo de long)
unsigned int (sinônimo de unsigned)
int
unsigned short int (sinônimo de unsigned short)
short int (sinônimo de short)
unsigned char
char
bool

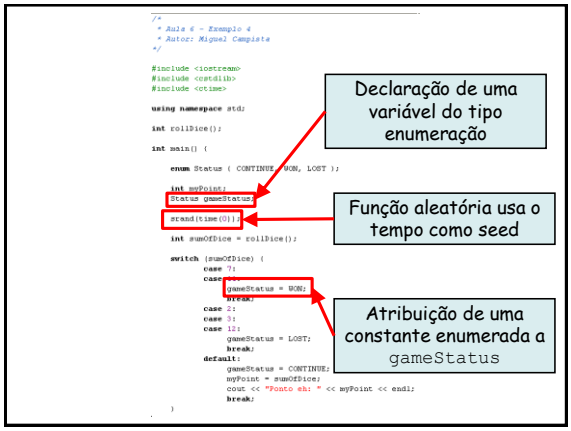
Tipo fundamental mais alto

Enumeração

- Conjunto de constantes inteiras representadas por identificadores
  - Os valores das constantes de enumeração iniciam em `0`, por padrão, e incrementam por `1`
  - Os identificadores em uma **enum** devem ser exclusivos, mas constantes enumeradas separadas podem ter o mesmo valor inteiro
- Definindo uma enumeração
  - Palavra-chave: **enum**
  - Um nome de tipo
  - Lista de nomes de identificadores separada por vírgulas entre chaves
    - Ex.: `enum Months { JAN = 1, FEB, MAR, APR };`


```
/*
 * Aula 6 - Exemplo 4
 * Autor: Miguel Campista
 */

#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int rollDice();

int main() {

    enum Status { CONTINUE, WON, LOST };

    int myPoint;
    Status gameStatus;

    srand(time(0));

    int sumOfDice = rollDice();

    switch (sumOfDice) {
            case 7:
            case 11:
                gameStatus = WON;
                break;
            case 2:
            case 3:
            case 12:
                gameStatus = LOST;
                break;
            default:
                gameStatus = CONTINUE;
                myPoint = sumOfDice;
                cout << "Ponto eh: " << myPoint << endl;
                break;
    }
```





Exemplo utilizando Enumeração em C++

```
        while (gameStatus == CONTINUE) {
            sumOfDice = rollDice();
            if (sumOfDice == myPoint)
                gameStatus = WON;
            else if (sumOfDice == 7)
                gameStatus = LOST;
        }

        if (gameStatus == WON)
            cout << "Vencedor!" << endl;
        else
            cout << "Perdedor!" << endl;

        return 0;
    }

    int rollDice() {
        int die1 = 1 + rand() % 6;
        int die2 = 1 + rand() % 6;

        int sum = die1 + die2;

        cout << "Jogador rolou: " << die1 << " + "
             << die2 << " = " << sum << endl;
        return sum;
    }
```


Exemplo utilizando Enumeração em C++

```
        while (gameStatus == CONTINUE) {
            sumOfDice = rollDice();
            if (sumOfDice == myPoint)
                gameStatus = WON;
            else if (sumOfDice == 7)
                gameStatus = LOST;
        }

        if (gameStatus == WON)
            cout << "Vencedor!" << endl;
        else
            cout << "Perdedor!" << endl;

        return 0;
    }

    int rollDice() {
        int die1 = 1 + rand() % 6;
        int die2 = 1 + rand() % 6;

        int sum = die1 + die2;

        cout << "Jogador rolou: " << die1 << " + "
             << die2 << " = " << sum << endl;
        return sum;
    }
```

Comparação utilizando uma variável constante enumerada

Exemplo utilizando Enumeração em C++

```
shell>$ g++ exemplo.cpp –o ex4
shell>$ ./ex4
Jogador rolou: 2 + 6 = 8
Ponto eh: 8
Jogador rolou: 6 + 4 = 10
Jogador rolou: 1 + 5 = 6
Jogador rolou: 5 + 4 = 9
Jogador rolou: 3 + 6 = 9
Jogador rolou: 1 + 1 = 2
Jogador rolou: 4 + 2 = 6
Jogador rolou: 5 + 4 = 9
Jogador rolou: 5 + 6 = 11
Jogador rolou: 4 + 4 = 8
Vencedor!
shell>$
```


Recomendações para o Uso de Enumeração

- Torne maiúscula a primeira letra de um identificador utilizado como um nome de tipo definido pelo usuário
- Utilize somente letras maiúsculas nos nomes das constantes enumeradas
  - Destaca essas constantes em um programa e lembra o programador que essas constantes não são variáveis
- Utilize enumerações ao invés de constantes do tipo inteiro para tornar os programas mais claros
  - O valor de uma constante enumerada pode ser configurada uma vez na declaração da enumeração

Recomendações para o Uso de Enumeração

- Constituem erros de compilação
  - Atribuir o equivalente inteiro de uma constante enumerada a uma variável do tipo enumerado
    - Ex.:
```
enum Months {JAN, FEV, MAR};
Months month;
month = 0;
```
  - Atribuir outro valor à constante enumerada depois que uma constante enumerada já tiver sido definida
    - Ex.:
```
enum Months {JAN, FEV, MAR};
Months month;
month = ABR;
```


Classes de Armazenamento

- Cada identificador tem diversos atributos
  - Nome, tipo, tamanho e valor
  - Além desses, classe de armazenamento, escopo e ligação (link)
- C++ oferece cinco especificadores de classe de armazenamento:
  - `auto, register, extern, mutable e static`
- Classe de armazenamento do identificador
  - Determina o **período durante o qual esse identificador permanece na memória**
- Escopo do identificador
  - Determina em que **lugar o identificador pode ser referenciado** em um programa

Classes de Armazenamento

- Ligação do identificador
  - **Determina se um identificador é conhecido apenas no arquivo de fonte em que é declarado ou nos múltiplos arquivos** que são compilados e depois ligados
- O especificador de classe de armazenamento do identificador ajuda a determinar a respectiva classe de armazenamento e ligação

Classes de Armazenamento

- Classe de armazenamento **automática**
  - Declarada com as palavras-chave `auto` e `register`
  - Variáveis automáticas
    - Criadas quando a execução do programa entra no bloco em que são definidas
    - Existem enquanto o bloco estiver ativo
    - São destruídas quando o programa sai do bloco
  - Apenas variáveis locais e parâmetros podem ser da classe de armazenamento automática
    - Essas variáveis normalmente são da classe de armazenamento automática

Classes de Armazenamento

- Especificador de classe de armazenamento `auto`
  - Declara explicitamente variáveis da classe de armazenamento automática
  - Variáveis locais são da classe de armazenamento automática por padrão
    - Portanto, a palavra-chave `auto` raramente é utilizada

Classes de Armazenamento

- Especificador de classe de armazenamento `register`
  - Dados na versão de linguagem de máquina de um programa normalmente são carregados em registradores para a execução de cálculos e outros tipos de processamento
    - O compilador tenta armazenar variáveis da classe de armazenamento automática em um registrador
  - Existe a possibilidade do compilador ignorar declarações `register`
    - Talvez não haja registradores suficientes para o compilador usar

Classes de Armazenamento

- Classe de armazenamento **estática**
  - Declarada com as palavras-chave **`extern`** e **`static`**
  - Variáveis da classe de armazenamento estática
    - Existem desde o momento em que o programa inicia a execução
    - São inicializadas assim que as declarações são encontradas
    - Duram enquanto o programa estiver executando
  - Funções da classe de armazenamento estática
    - O nome da função existe quando o programa começa a execução
    - Isso é válido para todas as outras funções

Classes de Armazenamento

- Classe de armazenamento **estática**
  - Mesmo que as variáveis e os nomes de função existam desde o início da execução do programa
    - Não significa que esses identificadores podem ser utilizados durante todo o programa

Classes de Armazenamento

- Dois tipos de identificadores com classe de armazenamento estática
  - **Variáveis e funções globais**
    - Declaradas com o especificador de classe de armazenamento **`extern`**
  - **Variáveis locais**
    - Declaradas com o especificador de classe de armazenamento **`static`**

Classes de Armazenamento

- Variáveis globais
  - São criadas inserindo-se declarações fora da definição de qualquer classe ou função
  - Retêm seus valores enquanto o programa estiver executando
  - Podem ser referenciadas por qualquer função que siga suas declarações ou definições no arquivo de fonte

Classes de Armazenamento

- Variáveis globais
  - Podem provocar efeitos colaterais indesejáveis quando uma função que não precisa de acesso à variável a modifica acidental ou maliciosamente
    - Em geral, exceto por recursos verdadeiramente globais, como `cin` e `cout`, o uso de variáveis globais deve ser evitado
      - A não ser em certas situações em que haja requisitos de desempenho exclusivos

Classes de Armazenamento

externVariable.cpp

```
// declaração de g_nValue
int g_nValue = 5;
```

mainVariable.cpp

```
#include <iostream>
using namespace std;
int g_nValue;
int main () {
        cout << "g_nValue = " << g_nValue << endl;
        g_nValue = 7;
        cout << "g_nValue = " << g_nValue << endl;
        return 0;
}
```

O que acontece com esse programa se for compilado como: `g++ -Wall externVariable.cpp mainVariable.cpp -o e` ?

Classes de Armazenamento

externVariable.cpp

```
// declaração de g_nValue
int g_nValue = 5;
```

mainVariable.cpp

```
#include <iostream>

using namespace std;

int g_nValue;

int main () {
        cout << "g_nValue = " << g_nValue << endl;

        g_nValue = 7;

        cout << "g_nValue = " << g_nValue << endl;

        return 0;
}
```

```
miguel@pegasus-linux:~$ g++ -Wall externVariable.cpp externMain.cpp -o e
/tmp/ccqoC3t3.o:(.bss+0x0): multiple definition of `g_nValue'
/tmp/ccopYI6V.o:(.data+0x0): first defined here
collect2: ld returned 1 exit status
```


Classes de Armazenamento

externVariable.cpp

```
// declaração de g_nValue
int g_nValue = 5;
```

mainVariable.cpp

```
// extern diz ao compilador que essa variável está
//declarada em outro lugar
#include <iostream>

using namespace std;

extern int g_nValue;

int main () {
        cout << "g_nValue = " << g_nValue << endl;

        g_nValue = 7;

        cout << "g_nValue = " << g_nValue << endl;

        return 0;
}
```

E agora se for compilado da mesma forma como:
`g++ -Wall externVariable.cpp mainVariable.cpp -o e`
?

Classes de Armazenamento

externVariable.cpp

```
// declaração de g_nValue
int g_nValue = 5;
```

mainVariable.cpp

```
// extern diz ao compilador que essa variável está
//declarada em outro lugar
#include <iostream>

using namespace std;

extern int g_nValue;

int main () {
        cout << "g_nValue = " << g_nValue << endl;

        g_nValue = 7;

        cout << "g_nValue = " << g_nValue << endl;

        return 0;
}
```

```
miguel@pegasus-linux:~$ g++ -Wall externVariable.cpp externMain.cpp -o e
miguel@pegasus-linux:~$ ./e
g_nValue = 5
g_nValue = 7
```

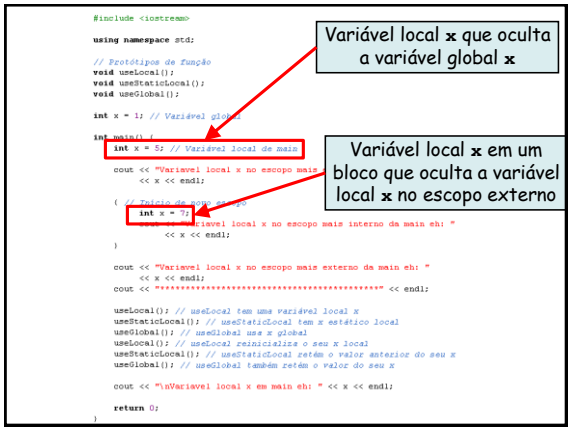

Classes de Armazenamento

- Variáveis locais declaradas com a palavra-chave **`static`**
  - Conhecidas apenas na função em que são declaradas
  - Mantêm seus valores quando a função retornar ao seu chamador
    - Na próxima vez em que a função for chamada, as variáveis locais `static` conterão os valores de quando a função completou pela última vez
  - Se as variáveis numéricas da classe de armazenamento estática não forem explicitamente inicializadas pelo programador
    - São inicializadas em zero


```
#include <iostream>

using namespace std;

// Protótipos de função
void useLocal();
void useStaticLocal();
void useGlobal();

int x = 1; // Variável global

int main() {
    int x = 5; // Variável local de main

    cout << "Variavel local x no escopo mais externo da main eh: "
         << x << endl;

    { // Inicio de novo escopo
        int x = 7;
        cout << "Variavel local x no escopo mais interno da main eh: "
             << x << endl;
    }

    cout << "Variavel local x no escopo mais externo da main eh: "
         << x << endl;
    cout << "*****************************************" << endl;

    useLocal(); // useLocal tem uma variável local x
    useStaticLocal(); // useStaticLocal tem x estático local
    useGlobal(); // useGlobal usa x global
    useLocal(); // useLocal reinicializa o seu x local
    useStaticLocal(); // useStaticLocal retém o valor anterior do seu x
    useGlobal(); // useGlobal também retém o valor do seu x

    cout << "\nVariavel local x em main eh: " << x << endl;

    return 0;
}
```

```
#include <iostream>

using namespace std;

// Protótipos de função
void useLocal();
void useStaticLocal();
void useGlobal();

int x = 1; // Variável global

int main() {
    int x = 5; // Variável local de main

    cout << "Variavel local x no escopo mais externo da main eh: "
         << x << endl;

    { // Inicio de novo escopo
        int x = 7;
        cout << "Variavel local x no escopo mais interno da main eh: "
             << x << endl;
    }

    cout << "Variavel local x no escopo mais externo da main eh: "
         << x << endl;
    cout << "*****************************************" << endl;

    useLocal(); // useLocal tem uma variável local x
    useStaticLocal(); // useStaticLocal tem x estático local
    useGlobal(); // useGlobal usa x global
    useLocal(); // useLocal reinicializa o seu x local
    useStaticLocal(); // useStaticLocal retém o valor anterior do seu x
    useGlobal(); // useGlobal também retém o valor do seu x

    cout << "\nVariavel local x em main eh: " << x << endl;

    return 0;
}
```

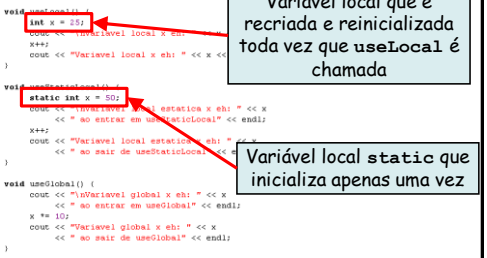
Declaração de uma variável global fora de qualquer classe ou definição de função



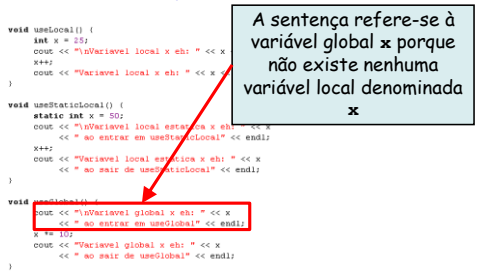
Exemplo utilizando Funções em C++

```
void useLocal() {
    int x = 25;
    cout << "\nVariavel local x eh: " << x << " ao entrar em useLocal" << endl;
    x++;
    cout << "Variavel local x eh: " << x << " ao sair de useLocal" << endl;
}

void useStaticLocal() {
    static int x = 50;
    cout << "\nVariavel local estatica x eh: " << x
         << " ao entrar em useStaticLocal" << endl;
    x++;
    cout << "Variavel local estatica x eh: " << x
         << " ao sair de useStaticLocal" << endl;
}

void useGlobal() {
    cout << "\nVariavel global x eh: " << x
         << " ao entrar em useGlobal" << endl;
    x *= 10;
    cout << "Variavel global x eh: " << x
         << " ao sair de useGlobal" << endl;
}
```

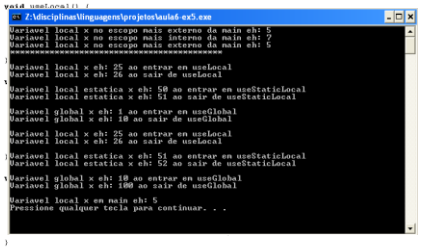

Exemplo utilizando Funções em C++



Exemplo utilizando Funções em C++



Exemplo utilizando Funções em C++



Funções Inline

- Reduzem o overhead de chamadas de função
  - Especialmente para funções pequenas
- Colocam o qualificador **`inline`** antes do tipo de retorno de uma função na definição de função
  - "Adverte" o compilador para que gere uma cópia do código da função em seu lugar (quando apropriado) para evitar uma chamada de função

Funções Inline

- Troca de funções **inline**
  - Múltiplas cópias do código da função são inseridas no programa (em geral tornando o programa maior)
- O compilador pode ignorar o qualificador **inline** e normalmente o faz para todas as funções
  - Exceto para as menores

Funções Inline

- Qualquer alteração em uma função **inline** pode exigir que todos os clientes da função sejam recompilados
  - Isso pode ser significativo em algumas situações de desenvolvimento e manutenção de programas
- O qualificador **inline** deve ser utilizado somente com funções pequenas
  - Funções **inline** podem reduzir o tempo de execução
    - Mas podem aumentar o tamanho do programa
- O qualificador **const** deve ser utilizado para sinalizar ao compilador que uma variável não pode ser alterada

Exemplo utilizando Funções Inline em C++

```
/*
 * Aula 6 - Exemplo 7
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

inline double cube (const double side) {
       return side * side * side;
}

int main() {
     double sideValue;
     cout << "Entre com o tamanho do lado: ";
     cin >> sideValue;

     cout << "O volume do cube de lado " << sideValue
          << "eh: " << cube(sideValue) << endl;

     return 0;
}
```


Exemplo utilizando Funções Inline em C++

O uso do qualificador **inline**

```
/*
 * Aula 6 - Exemplo 7
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

inline double cube (const double side) {
       return side * side * side;
}

int main() {
     double sideValue;
     cout << "Entre com o tamanho do lado: ";
     cin >> sideValue;

     cout << "O volume do cube de lado " << sideValue
          << "eh: " << cube(sideValue) << endl;

     return 0;
}
```


Exemplo utilizando Funções Inline em C++

```
/*
 * Aula 6 - Exemplo 7
 * Autor: Miguel Campista
*/
```

```
shell>$ g++ exemplo.cpp –o ex7
shell>$ ./ex7
Entre com o tamanho do lado: 2
O volume do cubo de lado 2 eh: 8
shell>$
```

```
int main() {
     double sideValue;
     cout << "Entre com o tamanho do lado: ";
     cin >> sideValue;

     cout << "O volume do cube de lado " << sideValue
          << "eh: " << cube(sideValue) << endl;

     return 0;
}
```


Referências e Parâmetros de Referências

- Duas formas de passar argumentos a funções
  - Passagem por valor
    - Uma *cópia* do valor do argumento é passada à função chamada
    - As mudanças na cópia não afetam o valor original da variável no chamador
      - Isso evita efeitos colaterais acidentais das funções
  - Passagem por referência
    - Permite que a função chamada acesse e modifique diretamente dados do argumento do chamador

Referências e Parâmetros de Referências

- Duas formas de passar argumentos a funções
  - Passagem por valor
    - Uma *cópia* do valor do argumento é passada à função chamada
    - As mudanças na cópia não afetam o valor original da variável no chamador
      - Isso evita efeitos colaterais acidentais das funções
  - Passagem por referência
    - Permite que a função chamada acesse e modifique diretamente dados do argumento do chamador

Passagem por valor não é vantajosa se um item de dados passado for grande. Copiar esses dados pode exigir uma quantidade considerável de tempo de execução e memória!

Referências e Parâmetros de Referências

- Parâmetro de referência
  - Uma referência para seu argumento correspondente em uma chamada de função
  - & colocado após o tipo de parâmetro no protótipo de função e cabeçalho de função
    - Ex.: `int &count` em um cabeçalho de função
      - Pronuncia-se "`count` é uma referência a um `int`"
  - O nome do parâmetro no corpo da função chamada na verdade refere-se à variável original na função chamadora

Exemplo utilizando Referências em C++

```
#include <iostream>

using namespace std;

int squareByValue(int);
void squareByReference(int &);

int main() {
    int v = 3, r = 4;

    cout << "v = " << v << " antes da funcao squareByValue\n";
    cout << "Valor retornado pela funcao squareByValue: "
         << squareByValue(v) << endl;
    cout << "v = " << v << " depois da funcao squareByValue\n\n";

    cout << "r = " << r << " antes da funcao squareByReference\n";
    squareByReference(r);
    cout << "r = " << r << " depois da funcao squareByReference\n";

    return 0;
}

int squareByValue(int n) {
    return n *= n;
}
void squareByReference(int &n) {
    n *= n;
}
```


Exemplo utilizando Referências em C++

```
#include <iostream>

using namespace std;

int squareByValue(int);
void squareByReference(int &);

int main() {
    int v = 3, r = 4;

    cout << "v = " << v << " antes da funcao squareByValue\n";
    cout << "Valor retornado pela funcao squareByValue: "
         << squareByValue(v) << endl;
    cout << "v = " << v << " depois da funcao squareByValue\n\n";

    cout << "r = " << r << " antes da funcao squareByReference\n";
    squareByReference(r);
    cout << "r = " << r << " depois da funcao squareByReference\n";

    return 0;
}

int squareByValue(int n) {
    return n *= n;
}
void squareByReference(int &n) {
    n *= n;
}
```

Função com passagem de parâmetro por valor

Função com passagem de parâmetro por referência

As variáveis são sempre passadas através dos identificadores

Exemplo utilizando Referências em C++

```
#include <iostream>

using namespace std;

int squareByValue(int);
void squareByReference(int &);

int main() {
    int v = 3, r = 4;

    cout << "v = " << v << " antes da funcao squareByValue\n";
    cout << "Valor retornado pela funcao squareByValue: "
         << squareByValue(v) << endl;
    cout << "v = " << v << " depois da funcao squareByValue\n\n";

    cout << "r = " << r << " antes da funcao squareByReference\n";
    squareByReference(r);
    cout << "r = " << r << " depois da funcao squareByReference\n";

    return 0;
}

int squareByValue(int n) {
    return n *= n;
}
void squareByReference(int &n) {
    n *= n;
}
```

Recebe cópia de argumento

Recebe referência de argumento

Exemplo utilizando Referências em C++

```
#include <iostream>

using namespace std;

int squareByValue(int);
void squareByReference(int &);

int main() {
```

```
shell>$ g++ exemplo.cpp –o ex8
shell>$ ./ex8
v = 3 antes da funcao squareByValue
Valor retornado pela funcao squareByValue: 9
v = 3 depois da funcao squareByValue

r = 4 antes da funcao squareByReference
r = 16 depois da funcao squareByReference
shell>$
```

```
}

int squareByValue(int n) {
    return n *= n;
}
void squareByReference(int &n) {
    n *= n;
}
```

Referências e Parâmetros de Referências

- Parâmetros por referência podem ser inadvertidamente tratados como parâmetros por valor já que em ambos os casos eles são mencionados apenas pelo nome
- Para passar objetos grandes, utilize um parâmetro de referência constante a fim de simular a aparência e a segurança da passagem por valor e evitar o overhead de passar uma cópia do objeto grande

Referências e Parâmetros de Referências

- Referências
  - Podem ser também utilizadas por outras variáveis dentro de uma função
    - Todas as operações supostamente executadas na referência são na verdade executadas na variável original
    - Devem ser inicializadas em suas declarações
      - Não podem ser reatribuídas posteriormente

```
int count = 1;
int &cRef = count;
cRef++;
```

Incrementa `count` por meio da referência `cRef`

Exemplo utilizando Referências em C++

```
/*
 * Aula 6 - Exemplo 9
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int main() {
    int x = 3;
    int &y = x; // y é uma referência

    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "\nx = " << x << endl << "y = " << y << endl;

    return 0;
}
```


Exemplo utilizando Referências em C++

```
/*
 * Aula 6 - Exemplo 9
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int main() {
    int x = 3;
    int &y = x; // y é uma referência

    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "\nx = " << x << endl << "y = " << y << endl;

    return 0;
}
```

Criação de uma referência para **x**

Atribuição de um valor a **x** através da sua referência **y**

Exemplo utilizando Referências em C++

```
shell>$ g++ exemplo.cpp –o ex9
shell>$ ./ex9
x = 3
y = 3

x = 7
y = 7
shell>$
```

```
    int x = 3;
    int &y = x; // y é uma referência

    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "\nx = " << x << endl << "y = " << y << endl;

    return 0;
}
```


Exemplo utilizando Referências em C++

```
/*
 * Aula 6 - Exemplo 9
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int main() {
    int x = 3;
    int &y;

    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "\nx = " << x << endl << "y = " << y << endl;

    return 0;
}
```

O que acontece nesse caso?

Referências e Parâmetros de Referências

- Retornando uma referência de uma função
  - As funções podem retornar referências a variáveis
    - Isso só pode ser usado quando a variável cuja referência foi retornada é estática à função chamada
      - Retornar uma referência a uma variável automática causa problema pois essa variável deixa de existir depois que a função termina → Referência é perdida!

Argumentos Padrão

- Valor-padrão a ser passado a um parâmetro
  - Argumento passado comumente a um parâmetro de uma função
    - Chamada da função não especifica o argumento para esse parâmetro
- Deve ser especificado com a primeira ocorrência do nome da função
  - Em geral, o protótipo da função
- Deve(m) ser o(s) argumento(s) mais à direita na lista de parâmetros de uma função
  - Padronização necessária caso a função receba outros argumentos

Exemplo utilizando Argumentos Padrão em C++

```
/*
 * Aula 6 - Exemplo 10
 * Autor: Miguel Campista
*/

#include <iostream>

int volume (int length = 1, int width = 1, int height = 1);

using namespace std;

int main() {
    cout << "O volume padrao eh: " << volume() << endl;
    cout << "\nO volume com comprimento 10 eh: "
         << volume(10) << endl;
    cout << "\nO volume com comprimento 10 e largura 5 eh: "
         << volume(10, 5) << endl;
    cout << "\nO volume com comprimento 10, largura 5 e altura 2 eh: "
         << volume(10, 5, 2) << endl;

   return 0;
}

int volume (int length, int width, int height) {
    return length * width * height;
}
```


Exemplo utilizando Argumentos Padrão em C++

```
/*
 * Aula 6 - Exemplo 10
 * Autor: Miguel Campista
*/

#include <iostream>

int volume (int length = 1, int width = 1, int height = 1);

using namespace std;

int main() {
    cout << "O volume padrao eh: " << volume() << endl;
    cout << "\nO volume com comprimento 10 eh: "
         << volume(10) << endl;
    cout << "\nO volume com comprimento 10 e largura 5 eh: "
         << volume(10, 5) << endl;
    cout << "\nO volume com comprimento 10, largura 5 e altura 2 eh: "
         << volume(10, 5, 2) << endl;

   return 0;
}

int volume (int length, int width, int height) {
    return length * width * height;
}
```

Argumentos padrão

Função chamadora sem argumento

Exemplo utilizando Argumentos Padrão em C++

```
/*
 * Aula 6 - Exemplo 10
 * Autor: Miguel Campista
*/

#include <iostream>

int volume (int length = 1, int width = 1, int height = 1);

using namespace std;

int main() {
    cout << "O volume padrao eh: " << volume() << endl;
    cout << "\nO volume com comprimento 10 eh: "
         << volume(10) << endl;
    cout << "\nO volume com comprimento 10 e largura 5 eh: "
         << volume(10, 5) << endl;
    cout << "\nO volume com comprimento 10, largura 5 e altura 2 eh: "
         << volume(10, 5, 2) << endl;

   return 0;
}

int volume (int length, int width, int height) {
    return length * width * height;
}
```

Argumentos padrão

Função chamadora com todos os argumentos

Exemplo utilizando Argumentos Padrão em C++

```
/*
 * Aula 6 - Exemplo 10
 * Autor: Miguel Campista
*/

#include <iostream>

int volume (int length = 1, int width = 1, int height = 1);
```

```
shell>$ g++ exemplo.cpp –o ex10
shell>$ ./ex10
O volume padrão eh: 1

O volume com comprimento 10 eh: 10

O volume com comprimento 10 e largura 5 eh: 50

O volume com comprimento 10, largura 5 e altura 2 eh: 100
shell>$
```

```
int volume (int length, int width, int height) {
    return length * width * height;
}
```

Argumentos Padrão

- Utilizar argumentos-padrão pode simplificar a escrita de chamadas de função
  - Entretanto, pode ser mais claro especificar todos os argumentos explicitamente
- Se os valores-padrão de uma função mudam...
  - Todos os códigos-cliente que estiverem utilizando a função devem ser recompilados

Operador de Solução de Escopo Unário (: :)

- Usado para acessar uma variável global quando uma variável local com o mesmo nome estiver no escopo
  - Ex.: `cout << ::x;`
- Não pode ser usado para acessar uma variável local com o mesmo nome em um bloco externo
- Sempre utilizar o operador unário de resolução de escopo (::) para referenciar as variáveis globais torna os programas mais fáceis de ler e entender
  - Variáveis globais são explicitadas no código

Exemplo utilizando Escopo Unário em C++

```
/*
 * Aula 6 - Exemplo 11
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

int x = 1; // Variável global

int main() {
    double x = 1.5;

    cout << "Valor double local eh: " << x << endl
         << "\nValor int global eh: " << ::x << endl;

    return 0;
}
```


Exemplo utilizando Escopo Unário em C++

```
/*
 * Aula 6 - Exemplo 11
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

int x = 1; // Variável global

int main() {
    double x = 1.5;

    cout << "Valor double local eh: " << x << endl
         << "\nValor int global eh: " << ::x << endl;

    return 0;
}
```

Operador unário para definição de escopo

Exemplo utilizando Escopo Unário em C++

```
shell>$ g++ exemplo.cpp –o exemplo
shell>$ ./exemplo
Valor double local eh: 1.5

Valor int global eh: 1
shell>$
```

```
using namespace std;

int x = 1; // Variável global

int main() {
    double x = 1.5;

    cout << "Valor double local eh: " << x << endl
         << "\nValor int global eh: " << ::x << endl;

    return 0;
}
```


Funções Sobrecarregadas

- As funções sobrecarregadas têm:
  - O mesmo nome e diferentes conjuntos de parâmetros
- O compilador seleciona a função apropriada
  - Baseado no nome, tipo e ordem dos argumentos na chamada de função
- A sobrecarga cria várias funções do mesmo nome
  - Executam tarefas semelhantes, mas em tipos de dados diferentes
- Sobrecarregar funções que realizam tarefas intimamente relacionadas pode tornar os programas mais legíveis e compreensíveis

Exemplo utilizando Funções Sobrecarregadas em C++

```
/*
 * Aula 6 - Exemplo 12
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int square(int x) {
    cout << "Quadrado do inteiro " << x << " eh: ";
    return x * x;
}

double square(double x) {
    cout << "Quadrado do double " << x << " eh: ";
    return x * x;
}

int main() {
    cout << square(2) << endl;
    cout << square(2.5) << endl;

    return 0;
}
```


Exemplo utilizando Funções Sobrecarregadas em C++

```
/*
 * Aula 6 - Exemplo 12
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int square(int x) {
    cout << "Quadrado do inteiro " << x << " eh: ";
    return x * x;
}

double square(double x) {
    cout << "Quadrado do double " << x << " eh: ";
    return x * x;
}

int main() {
    cout << square(2) << endl;
    cout << square(2.5) << endl;

    return 0;
}
```

Função `square` para `int`

Função `square` para `double`

Exemplo utilizando Funções Sobrecarregadas em C++

```
shell>$ g++ exemplo.cpp –o ex12
shell>$ ./ex12
Quadrado do inteiro 2 eh: 4
Quadrado do double 2.5 eh: 6.25
shell>$
```

```
int square(int x) {
    cout << "Quadrado do inteiro " << x << " eh: ";
    return x * x;
}

double square(double x) {
    cout << "Quadrado do double " << x << " eh: ";
    return x * x;
}

int main() {
    cout << square(2) << endl;
    cout << square(2.5) << endl;

    return 0;
}
```


Funções Sobrecarregadas

- Como o compilador diferencia as funções sobrecarregadas?
  - As funções sobrecarregadas são diferenciadas pela respectiva assinatura
  - Desfiguração de nome ou decoração de nome
    - O compilador codifica cada identificador de função com o número e o tipo de parâmetro para permitir a ligação segura para tipos
  - A ligação segura para tipos garante que
    - Seja chamada a função sobrecarregada apropriada
    - Os tipos de argumento correspondam aos tipos de parâmetro

Exemplo utilizando Funções Sobrecarregadas em C++

```
/*
 * Aula 6 - Exemplo 13
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int square (int x)
    return x * x;
}
double square (double x)
    return x * x;
}

int main() {
    return 0;
}
```

Função `square` sobrecarregada

Exemplo utilizando Funções Sobrecarregadas em C++

```
/*
 * Aula 6 - Exemplo 13
 * Autor: Miguel Campista
 */

#include <iostream>

using namespace std;

int square (int x)
    return x * x;
}
double square (double x)
    return x * x;
}

int main() {
    return 0;
}
```

Função `square` sobrecarregada

```
shell>$ g++ -S –o ex13 exemplo.cpp
```

```
.type   _GLOBAL__I__Z6squarei, @function
.type   _GLOBAL__I__Z6squared, @function
```

Funções Sobrecarregadas

- Uma função com argumentos padrão omitidos pode ser chamada de modo idêntico a outra função sobrecarregada
  - Isso constitui um erro de compilação!
    - Ex.: Uma função que não aceita explicitamente nenhum argumento e uma função de mesmo nome que contém todos os argumentos como padrão provoca um erro de compilação...

O compilador não consegue identificar qual função deve utilizar

Funções Sobrecarregadas

- Uma função com argumentos padrão omitidos pode ser chamada de modo idêntico a outra função sobrecarregada
  - Isso constitui um erro de compilação!
    - Ex.: Uma função que não aceita explicitamente nenhum argumento e uma função de mesmo nome que contém todos os argumentos como padrão provoca um erro de compilação...

```
int funcao(int a = 1, int b = 2);

double funcao(int x);
```


Templates de Funções

- Forma mais compacta e conveniente de sobrecarga
  - Lógica e operações de programação idênticas para cada tipo de dados

Templates de Funções

- Definição de template de função
  - É escrita por programadores uma única vez
  - Define toda a família de funções sobrecarregadas
  - Começa com a palavra-chave **`template`**
  - Contém uma lista de parâmetros template de parâmetros de tipo formal para a função template entre colchetes angulares (<>)
  - Parâmetros de tipo formal
    - Precedido pela palavra-chave **`typename`** ou **`class`**
    - São marcadores de lugar para tipos fundamentais ou tipos definidos pelo usuário

Templates de Funções

- Especializações de template de função
  - Geradas automaticamente pelo compilador para lidar com cada tipo de chamada para o template de função
  - Exemplo para o template de função **`max`** com o tipo de parâmetro **`T`** chamado com argumentos **`int`**
    - O compilador detecta uma invocação **`max`** no código do programa
    - **`int`** substitui **`T`** em toda a definição do template
    - Isso gera a especialização do template de função **`max< int >`**

Templates de Funções

- Não colocar a palavra-chave **`class`** ou **`typename`** antes de cada parâmetro de tipo formal de um template de função é um erro de sintaxe
  - Ex.: Escrever `< class S, T >` em vez de `< class S, class T >`) é um erro

Templates com tipos diferentes de dados...

Exemplo utilizando Funções Templates em C++

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo template.h
 * Autor: Miguel Campista
 */

template <class T>
T maximo(T v1, T v2, T v3) {
        T vMax = v1;
        if (v2 > vMax)
                vMax = v2;
        if (v3 > vMax)
                vMax = v3;
        return vMax;
}
```


Exemplo utilizando Funções Templates em C++

Usando o parâmetro de tipo formal T no lugar do tipo de dados

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo template.h
 * Autor: Miguel Campista
 */

template <class T>
T maximo(T v1, T v2, T v3) {
        T vMax = v1;
        if (v2 > vMax)
                vMax = v2;
        if (v3 > vMax)
                vMax = v3;
        return vMax;
}
```


Exemplo utilizando Funções Templates em C++

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include "template.h"

using namespace std;

int main() {
    int i1, i2, i3;
    cout << "Entre com os valores de tres inteiros: " << endl;
    cin >> i1 >> i2 >> i3;
    cout << "O valor maximo eh: " << maximo(i1, i2, i3) << endl;

    double d1, d2, d3;
    cout << "\nEntre com os valores de tres doubles: " << endl;
    cin >> d1 >> d2 >> d3;
    cout << "O valor maximo eh: " << maximo(d1, d2, d3) << endl;

    char c1, c2, c3;
    cout << "\nEntre com os valores de tres chars: " << endl;
    cin >> c1 >> c2 >> c3;
    cout << "O valor maximo eh: " << maximo(c1, c2, c3) << endl;

    return 0;
}
```


Exemplo utilizando Funções Templates em C++

Função **maximo** com argumentos **int**

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include "template.h"

using namespace std;

int main() {
    int i1, i2, i3;
    cout << "Entre com os valores de tres inteiros: " << endl;
    cin >> i1 >> i2 >> i3;
    cout << "O valor maximo eh: " << maximo(i1, i2, i3) << endl;

    double d1, d2, d3;
    cout << "\nEntre com os valores de tres doubles: " << endl;
    cin >> d1 >> d2 >> d3;
    cout << "O valor maximo eh: " << maximo(d1, d2, d3) << endl;

    char c1, c2, c3;
    cout << "\nEntre com os valores de tres chars: " << endl;
    cin >> c1 >> c2 >> c3;
    cout << "O valor maximo eh: " << maximo(c1, c2, c3) << endl;

    return 0;
}
```


Exemplo utilizando Funções Templates em C++

Função **maximo** com argumentos **double**

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include "template.h"

using namespace std;

int main() {
    int i1, i2, i3;
    cout << "Entre com os valores de tres inteiros: " << endl;
    cin >> i1 >> i2 >> i3;
    cout << "O valor maximo eh: " << maximo(i1, i2, i3) << endl;

    double d1, d2, d3;
    cout << "\nEntre com os valores de tres doubles: " << endl;
    cin >> d1 >> d2 >> d3;
    cout << "O valor maximo eh: " << maximo(d1, d2, d3) << endl;

    char c1, c2, c3;
    cout << "\nEntre com os valores de tres chars: " << endl;
    cin >> c1 >> c2 >> c3;
    cout << "O valor maximo eh: " << maximo(c1, c2, c3) << endl;

    return 0;
}
```


Exemplo utilizando Funções Templates em C++

Função **maximo** com argumentos **char**

```
/*
 * Aula 6 - Exemplo 14
 * Arquivo principal
 * Autor: Miguel Campista
 */

#include <iostream>
#include "template.h"

using namespace std;

int main() {
    int i1, i2, i3;
    cout << "Entre com os valores de tres inteiros: " << endl;
    cin >> i1 >> i2 >> i3;
    cout << "O valor maximo eh: " << maximo(i1, i2, i3) << endl;

    double d1, d2, d3;
    cout << "\nEntre com os valores de tres doubles: " << endl;
    cin >> d1 >> d2 >> d3;
    cout << "O valor maximo eh: " << maximo(d1, d2, d3) << endl;

    char c1, c2, c3;
    cout << "\nEntre com os valores de tres chars: " << endl;
    cin >> c1 >> c2 >> c3;
    cout << "O valor maximo eh: " << maximo(c1, c2, c3) << endl;

    return 0;
}
```

Exemplo utilizando Funções Templates em C++

```
shell>$ g++ exemplo.cpp –o ex14
shell>$ ./ex14
Entre com os valores de tres inteiros:
1 2 3
O valor maximo eh: 3

Entre com os valores de tres doubles:
1.2 2.3 3.4
O valor maximo eh: 3.4

Entre com os valores de tres chars:
a b c
O valor maximo eh: c
shell>$
```

```
    char c1, c2, c3;
    cout << "\nEntre com os valores de tres chars: " << endl;
    cin >> c1 >> c2 >> c3;
    cout << "O valor maximo eh: " << maximo(c1, c2, c3) << endl;

    return 0;
}
```


Exemplo: Máquina para Testes de Multiplicação

- Escreva um programa em C++ para tomar a tabuada de alunos de primário
  - Cada acerto e erro gera uma mensagem aleatória de incentivo
  - Após 10 rodadas, se o desempenho tiver sido abaixo do mínimo o programa termina e avisa ao usuário o motivo

?

Exemplo: Máquina para Testes de Multiplicação

```
/*
 * Aula 6 - Exemplo 17
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include "machine.h"

using namespace std;

int main() {
    int exit;
    string name;

    cout << "Entre com o seu nome: ";
    getline(cin, name);

    Machine machine(name);

    while (1) {

        exit = machine.multiplicationTests();

        if(exit == 1) {
            cout << "\n\nTchau!\n\n";
            break;
        }
```



```
        if ((machine.getRuns() % 10) == 0) {

            if (machine.getPerf() <= 0.75) {
                cout << name << ", seu desempenho na rodada "
                     << machine.getRound()
                     << "foi abaixo do esperado: "
                     << machine.getPerf() << endl;
                cout << "\nPor favor, ";
                cout << "procure o seu professor para ajuda extra."
                     << endl;
                break;
            } else {
                string cont;
                cout << name << ", seu desempenho na rodada "
                     << machine.getRound()
                     << " foi acima do esperado: "
                     << setprecision(2) << fixed
                     << 100*machine.getPerf() << "%." << endl;
                cout << "\nParabens!!!\n" << endl;

                cin.ignore(100, '\n');

                cout << "Deseja continuar? (S/N)" << endl;
                getline(cin, cont);

                if (cont.compare("S") == 0 || cont.compare("SIM") == 0) {
                    machine.setRound();
                    cout << "\n== Proxima rodada: Rodada "
                         << machine.getRound() << " ==\n" << endl;
                } else if (cont.compare("N") == 0 || cont.compare("NAO") == 0) {
                    cout << "\n\nTchau!\n\n";
                    break;
                } else {
                    cout << "\nResposta desconhecida\nTerminando."
                         << endl;
                    break;
                }
            }
        }
    }

    return 0;
}
```

```
/*
 * Aula 6 - Exemplo 17
 * Arquivo machine.h
 * Autor: Miguel Campista
 */
#include <string>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "machine-template.h"

using namespace std;

class Machine {
    public:
        // Construtor
        Machine(string);
        // Função para lançar novo desafio matemático
        int multiplicationTests();
        // Função para obter o desempenho nas últimas 10 rodadas
        double getPerf();
        // Função para obter o número de rodadas
        int getRuns();
        // Função para obter o número conjunto de 10 rodadas
        int getRound();
        // Função para ajustar o número conjunto de 10 rodadas
        void setRound();
    private:
        // Variáveis privadas
        enum Performance { SUCCESS, FAILURE };
        int successes, failures, runs, round;
        string nameStudent;
        // Função para checar se a resposta está correta
        void checkResp(int, int, int);
        // Funções para uma respostas após o desafio
        void correctAnswer();
        void WrongAnswer();
        // Função para contabilizar sucessos e falhas
        void setPerf(Performance);

};
```

Exemplo: Máquina para Testes de Multiplicação

```
/*
 * Aula 6 - Exemplo 17
 * Arquivo machine.cpp
 * Autor: Miguel Campista
 */
#include "machine.h"

Machine::Machine(string name) {
        successes = 0;
        failures = 0;
        runs = 0;
        round = 1;
        srand(time(0));
        nameStudent = name;
        cout << "\nAprendendo de matematica\n\n******* "
             << nameStudent << ", Bem-vindo! *******\n\n"
             << "Digite -1 para sair.\n" << endl;
}

int Machine::multiplicationTests() {
    int a = rand() % 10;
    int b = rand() % 10;
    int resp, perf;

    cout << "Quanto eh " << a << " x " << b << " ?\nResposta: ";
    cin >> resp;

    if(resp == -1)
            return 1;
    else
            checkResp(a, b, resp);

    return 0;
}
```

Exemplo: Máquina para Testes de Multiplicação

```
void Machine::checkResp(int a, int b, int resp) {
    Performance perf;
    if(multiply(a, b) == resp) {
        perf = SUCCESS;
        correctAnswer();
    } else {
        perf = FAILURE;
        wrongAnswer();
    }
    setPerf(perf);
}

void Machine::correctAnswer() {
    int randAnswer = rand() % 4;

    switch (randAnswer) {
        case 0:
            cout << "\nMuito bem!\n" << endl;
            break;
        case 1:
            cout << "\nExcelente!\n" << endl;
            break;
        case 2:
            cout << "\nBom trabalho!\n" << endl;
            break;
        case 3:
            cout << "\nContinue com o bom trabalho!\n" << endl;
            break;
    }
}
```



```
void Machine::wrongAnswer() {
    int randAnswer = rand() % 4;

    switch (randAnswer) {
        case 0:
            cout << "\nNum, por favor, tente novamente.\n" << endl;
            break;
        case 1:
            cout << "\nErrado. Tente mais uma vez.\n" << endl;
            break;
        case 2:
            cout << "\nNao desista!\n" << endl;
            break;
        case 3:
            cout << "\nNao. Continue tentando.\n" << endl;
            break;
    }
}

void Machine::setPerf(Performance perf) {
    if(perf == SUCCESS)
        successes++;
    else
        failures++;
    runs++;
}

double Machine::getPerf() {
    return static_cast <double> (successes) / runs;
}

int Machine::getRuns() {
    return runs;
}

int Machine::getRound() {
    return round;
}
void Machine::setRound() {
    round++;
    runs = 0;
    successes = 0;
    failures = 0;
}
```

Exemplo: Máquina para Testes de Multiplicação

```
/*
 * Aula 6 - Exemplo 17
 * Arquivo machine-template.h
 * Autor: Miguel Campista
 */

template <class T>
T multiply (T a, T b) {
    return a * b;
}
```


Leitura Recomendada

- Capítulos 4, 5 e 6 do livro
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005