UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação 2

ET43G

Fundamentos de C++: origens, padrões ISO, diferenças para C, comentários, cabeçalhos, tipos primitivos e compostos, operador de escopo, entrada e saída, definição de variáveis, variáveis referência, classes *string* e *vector*, *namespaces*.

Aula 4

Prof. Daniel Cavalcanti Jeronymo

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia Eletrônica – 3º Período
2016.1

Plano de Aula

- História do C++ – Padrões ISO
- C vs C++
- Fundamentos
- STL básica – string e vector

História do C++

- *C++ é um superconjunto da linguagem C*
  - *Uma linguagem B é dita superconjunto de A quando todo programa válido na linguagem A é também válido em B*

- "C com Classes" – Bjarne Stroustrup, 1979

História do C++

- C com características de Simula (superconjunto de ALGOL)
  - Objetos
  - Classes
  - Herança
  - Funções virtuais
  - Etc…

- *"the name signifies the evolutionary nature of the changes from C“*

  – Bjarne Stroustrup

História do C++

- “C with Classes” -> C++ – 1983

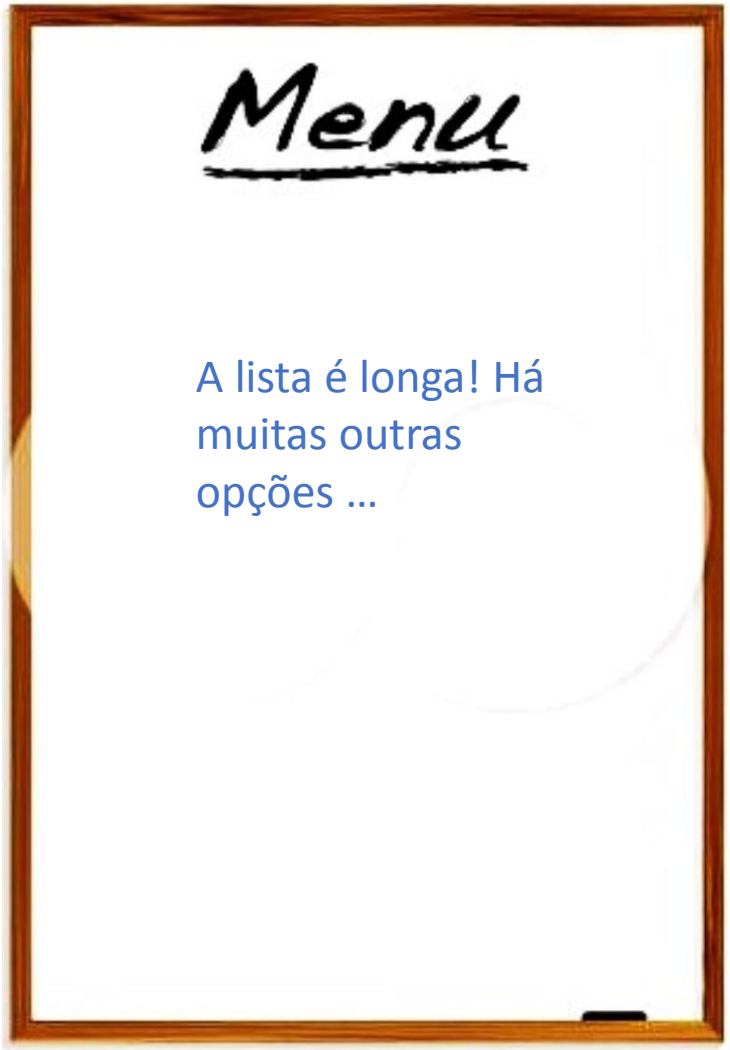
- Novas regras sintáticas e semânticas para:
  - *Classes, funções inline, argumento padrão, funções virtuais, sobrecarga de operadores e funções, referências, constantes, new/delete, verificação de tipos, comentários de uma linha, funções virtuais puras (classes abstratas), herança múltipla, funções estáticas e constantes, templates, exceções, namespaces, casts (static, dynamic, reinterpret), tipo booleano, etc …*

História do C++

- Padronizações
  - 1998 – ISO/IEC 14882:1998 – C++98 (1o padrão)
  - 2003 – ISO/IEC 14882:2003 – C++03 (correções)
  - 2007 – ISO/IEC TR 19768:2007 – C++07/TR1 (extensões STL)
  - 2011 – ISO/IEC 14882:2011 – C++11 (extensões)
  - 2014 – ISO/IEC 14882:2014 – C++14 (extensões)
  - 2017 – ? – C++17

História do C++

- Compiladores
  - GCC / G++
  - Clang / LLVM
  - Intel C++
  - Microsoft Visual C++
- IDEs
  - Code::Blocks
  - Vim
  - Geany



C vs C++

- Programa básico para saída de dados em C

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Ola %s!\n", "mundo");

    return 0;
}
```

C vs C++

- Programa básico para saída de dados em C++

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "Ola " << "mundo" << "!" <<
std::endl;

    return 0;
}
```

C vs C++

- Programa básico para saída de dados em C++ (alternativo)

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    cout << "Ola " << "mundo" << "!" << endl;

    return 0;
}
```

C vs C++

- C++ foi desenvolvido para ser compatível com C
  - Intenção original: C++ deveria ser um superconjunto de C
  - Na prática …

C vs C++

- Tipagem
  - C – estática, fracamente enforçada
  - C++ – estática, mais fortemente enforçada

- Correto em C

```
void *pvoid;
int *pint = pvoid;
```

- Correto em C++

```
void *pvoid;
int *pint = (int*)pvoid;
```

C vs C++

- Tipagem – fortemente enforçada

- Sem erro em C

```
char A = 10;
int *B = (int*) &A;
```

- Sem erro em C++

```
char A = 10;
int *B = (int*) &A;
```

- Aponta o erro em C++

```
char A = 10;
int *B = static_cast<int*>(&A);
```

C vs C++

- Reinterpretações de tipo em C++
  - Estilo C: **(**tipo**)**valor
  - Estilo C – notação funcional: tipo**(**valor**)**
  - Estilo C++: **static_cast<**tipo**>(**valor**)**

- Outros reinterpretadores:
  - const_cast, reinterpret_cast, dynamic_cast

C vs C++

- reinterpret_cast
  - Cópia de ponteiros
  - Não há verificação de conteúdo nem dos ponteiros

- dynamic_cast
  - Permite reinterpretação entre classes bases e classes derivadas
  - Exige RTTI (*Run Time Type Information*)

C vs C++

- const_cast
  - Permite adicionar e remover o modificador const
  - **Cuidado!**
  - Teste o código abaixo em C e C++, adapte a sintaxe conforme necessário
  - **O resultado é o esperado?**

```
const int A = 10;

int *B = const_cast<int*>(&A);
*B = 20;


cout << &A << " " << B << endl;
cout << A << " " << *B << endl;
```

OBS: O resultado desse código é dependente do compilador

*4 Certain other operations are described in this International Standard as undefined (for example, **the effect of attempting to modify a const object**). [Note: This International Standard imposes no requirements on the behavior of programs that contain undefined behavior. —end note]*

C vs C++

- const_cast
  - A remoção de const é sintática, não semântica!
  - **A remoção de const permite a remoção de *constness* de um *caminho de acesso* (ponteiro ou referência) que leva a um objeto não-constante.**
  - Compare com o resultado anterior

```
int A = 10;
const int *pA = &A;

int *B = const_cast<int*>(pA);
*B = 20;


cout << &A << " " << B << endl;
cout << A << " " << *B << endl;
```

C vs C++

- Enumeradores
  - C – inteiros
  - C++ - tipos próprios
- Struct, union e enum recebem *typedef* implícito

- Código correto em C

```
struct MinhaStruct
{
    int a;
    struct MinhaStruct *b;
};

struct MinhaStruct var;
```

- Código correto em C++

```
struct MinhaStruct
{
    int a;
    MinhaStruct *b;
};

MinhaStruct var;
```

C vs C++

- C++ introduz novas palavras-chaves que não existem em C

  - Código válido em C mas não em C++

```
struct template
{
    int new;
    struct template* class;
};
```

C vs C++

- C++ introduz novas palavras-chaves que não existem em C

alignas (C++11) alignof (C++11) and and_eq asm auto(1) bitand bitor bool break case catch char char16_t (C++11) char32_t (C++11) class compl concept (TS) const constexpr (C++11) const_cast continue decltype (C++11) default(1) delete(1) do double dynamic_cast	else enum explicit export(1) extern false float for friend goto if inline int long mutable namespace new noexcept (C++11) not not_eq nullptr (C++11) operator or or_eq private protected public register reinterpret_cast	requires (TS) return short signed sizeof static static_assert (C++11) static_cast struct switch template this thread_local (C++11) throw true try typedef typeid typename union unsigned using(1) virtual void volatile wchar_t while xor xor_eq

C vs C++

- Argumentos vazios em funções
  - C – func() é equivalente a func(...)
  - C++ – func() é equivalente a func(void)

- Literais de strings
  - C – literais são inteiros
  - C++ – literais são char
  - Verifique com sizeof('a')

C vs C++

- Gerenciamento de memória
  - C – malloc, calloc, free e realloc
  - C++ – **new, delete, new[], delete[]**
  - **Teste a versão com e sem nothrow para um valor grande**

- Código em C

```
int *vec = malloc(sizeof(int)*10);
free(vec);
```

- Código em C++

```
int *vec = new int[10];
delete [] vec;

int *vec = new(nothrow) int[10];
delete [] vec;
```

C vs C++

- Comentários
  - C – Múltiplas linhas com /* */
  - C++ – Múltiplas linhas com /* */ e uma linha com //

- Passagem por referência
  - C – Comportamento emulado usando ponteiros
  - C++ – Comportamento real utilizando o operador &

```
int A = 10;
int &B = A;

cout << A << " " << B << endl;

B = 5;
cout << A << " " << B << endl;

A = 1;
cout << A << " " << B << endl;
```

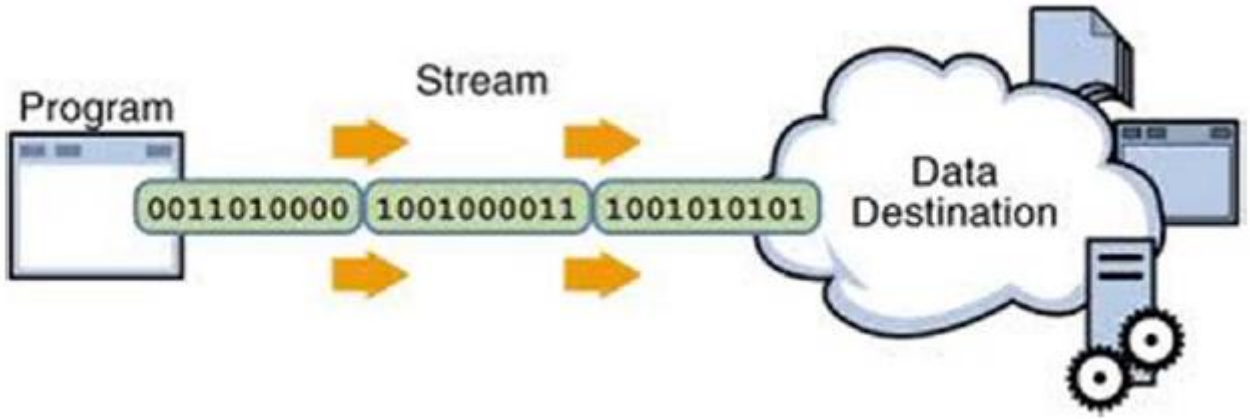
C vs C++

- Arquivos de cabeçalhos
  - C – terminam em .h
  - C++ – não terminam em .h e são diferentes dos arquivos de C
  - Arquivos de cabeçalho normalmente utilizados em C, em C++ recebem **c** precedendo o nome

```
#include <iostream>
#include <cmath>
```

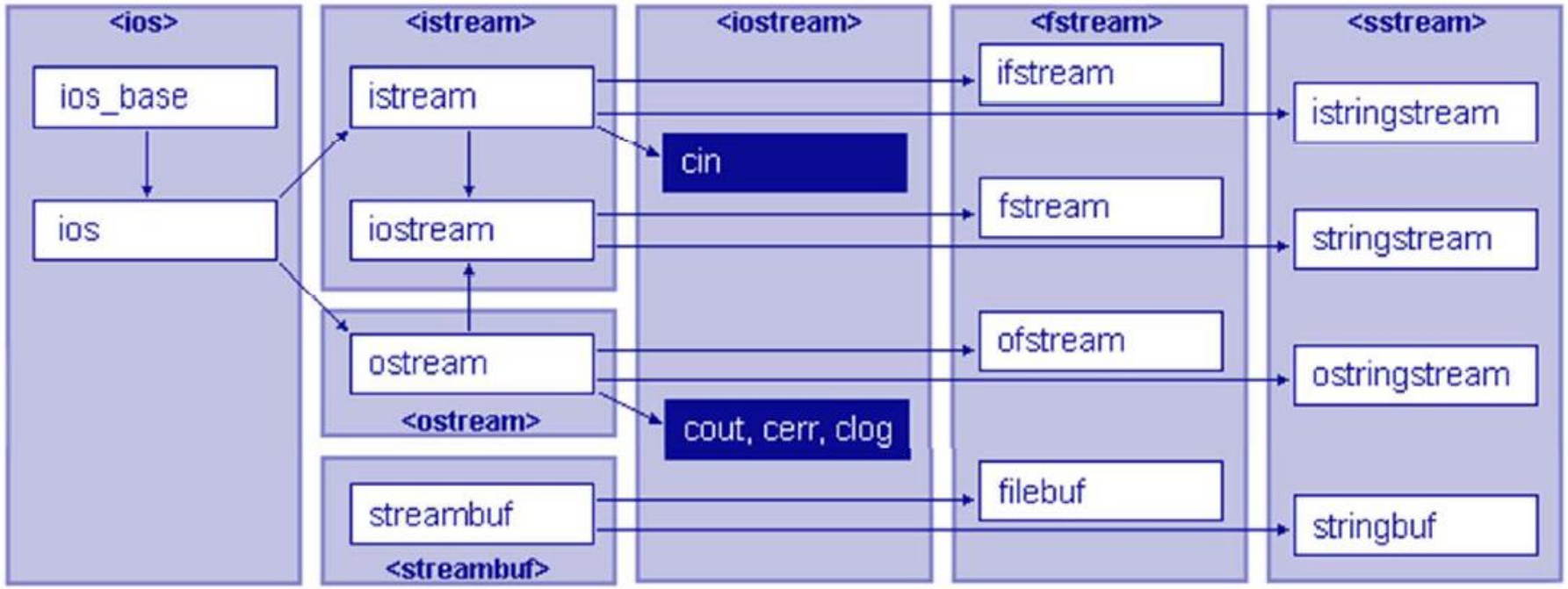
Fundamentos

- E/S básica
  - Baseada em streams
  - Abstração que representa fluxo de dados entre objetos



Fundamentos

- E/S básica
  - <cstdio> disponibiliza as funções padrões do C
  - <iostream> disponibiliza a funcionalidade padrão do C++



Fundamentos

- E/S básica
  - Objeto cout
  - Saída de dados na tela
  - Quebra de linha é gerada por std::endl

```
int A = 10;
char str[] = "uma string em array";

cout << "Imprimindo uma variavel: " << A << endl;
cout << "Imprimindo " << str << endl;
```

Fundamentos

- E/S básica
  - Saída pode ser formatada
  - Incluir <iomanip>

```
int A = 10;

cout << "Imprimindo em decimal: " << dec << A << endl;
cout << "Imprimindo em hexadecimal: " << hex << uppercase << setw(2) << setfill('0') << A << endl;
cout << "Imprimindo em octal: " << oct << A << endl;
```

Fundamentos

- E/S básica
  - Objeto cin
  - Entrada de dados na tela

```
int A;

cout << "Entre com um valor: ";
cin >> A;
cout << endl << "Seu valor: " << A << endl;
```

Fundamentos

- E/S básica
  - Objeto cin
  - Leitura de texto

```
char A[100];

cout << "Entre com um texto: ";
cin >> A;
cout << endl << "Seu texto: " << A << endl;
```

Fundamentos

- E/S básica
  - Leitura de texto **com string!**
  - Incluir **<string>**

```
string A;

cout << "Entre com um texto: ";
cin >> A;
cout << endl << "Seu texto: " << A << endl;
```

Fundamentos

- E/S básica
  - Leitura de linha inteira

```
string A;

cout << "Entre com um texto: ";
getline(cin, A );
cout << endl << "Seu texto: " << A << endl;
```

Fundamentos

- E/S básica
  - Arquivos -Funcionamento similar ao de cin e cout!

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
 ofstream arq("exemplo.txt");
 if (arq.is_open())
 {
  arq << "Uma linha.\n" << endl;
  arq << "Outra linha." << endl;
  arq.close();
 }
 else
  cout << "Erro ao abrir o arquivo!";
 return 0;
}
```

Fundamentos

- Operador de resolução de escopo e namespace
  - **Namespace** realiza o agrupamento de variáveis, classes e funções sob um nome

```
#include <iostream>
using namespace std;
namespace MeuNamespace
{
    int x = 10;
}
int x = -1;
int main()
{
    int x = 0;

    cout << "Local: " << x << endl;
    cout << "Global: " << ::x << endl;
    cout << "Namespace: " << MeuNamespace::x << endl;
}
```

Fundamentos

- Namespace
  - Declarações podem estar separadas de suas definições

```
namespace MeuNamespace{
    int x = 10;
    int minhafunc();
}

int MeuNamespace::minhafunc(){
    /* algo */
}
```

  - using define o conteúdo daquele namespace no escopo global

```
using MeuNamespace;

using MeuNamespace::minhafunc;
```

Fundamentos

- Conversão de / para string utilizando stringstream

```
#include <iostream>
#include <sstream>
using namespace std;


int main()
{
    stringstream ss;

    int i = 10;
    float f;

    ss << i;
    cout << "Inteiro para string: " << ss.str() << endl;

    ss.str("-9.4"); // redefine a string para conter o valor "9.4"
    ss >> f;
    cout << "String para float: " << f << endl;
}
```

STL

- Para casa – procurar material sobre
- A biblioteca padrão (Standard Template Library)
- A classe string
- A classe vector

STL

- Para casa – entender o seguinte código

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void minhafunc(int i)
{
  cout << ' ' << i;
}

struct functor
{
   inline void operator()(int element)
   {
      cout << ' ' << element;
   }
};

int main()
{
   int vec[] = {-1, 0, 4, -10, 5, 100};
   vector<int> meuvec(vec, vec + sizeof(vec)/sizeof(vec[0]));

   for_each(meuvec.begin(), meuvec.end(), minhafunc);

   cout << endl;

   functor minhafunc2;
   for_each(meuvec.begin(), meuvec.end(), minhafunc2);
}
```