Treinamento Maratona de Prog. 2015

Dia 2

C++

Abril de 2015

unipampa

Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Diferenças entre C e C++;
- Declaração de Classes;
- Ponteiros;
- Templates;
- Namespaces;
- Standard Template Library (STL).

Conceitos Básicos

- **Entrada e saída:**
  - *stdin* → Dispositivo de entrada padrão. (teclado)
  - *stdout* → Dispositivo de saída padrão. (monitor)
  - *stderr* → Dispositivo de saída de erro padrão. (monitor)

- *int **printf**(char *str, …); int **sprintf**(char *destino, char *str, ...);*
- *int **scanf**(char *str, …); int **sscanf**(char *origem, char *str, ...);*
- *int **fprintf**(FILE *pf, char *str, ...);*
- *int **fscanf**(FILE *pf, char *str, ...);*

Conceitos Básicos

- **Exemplos *printf***

CÓDIGO	FORMATAÇÃO	SAÍDA
printf(“%4d”, 45);	No mínimo 4 caracteres	| 45|
printf(“%-4d”, 45);	Anterior, mas alinhado a esq.	|45 |
printf(“%04d”, 45);	Completa com zeros	|0045|
printf(“%.2f”, 75.777);	2 casas decimais	|75,78|
printf("%8.2f\n", 75.777);	Anterior, mas com 8 carac.	| 75,78|
printf("%-8.2f\n", 75.777);	Anterior, mas alinhado a esq.	|75,78 |

Conceitos Básicos

- **Exemplos *sprintf, scanf, sscanf***

CÓDIGO	DESCRIÇÃO
sprintf(s, "%d", x);	converte x para uma string
sprintf(s, "%d%f", a, f);	concatenar a e f, armazenando em s
scanf(“%d\n”, &n);	lê um inteiro e pula o \n
scanf(“%[^\n]\n”, str);	lê uma linha e pula o \n
scanf(“%s%n”, str, &n);	lê uma str e a qtd de char lidos
sscanf(s, "%f", &f);	Converte s para um float

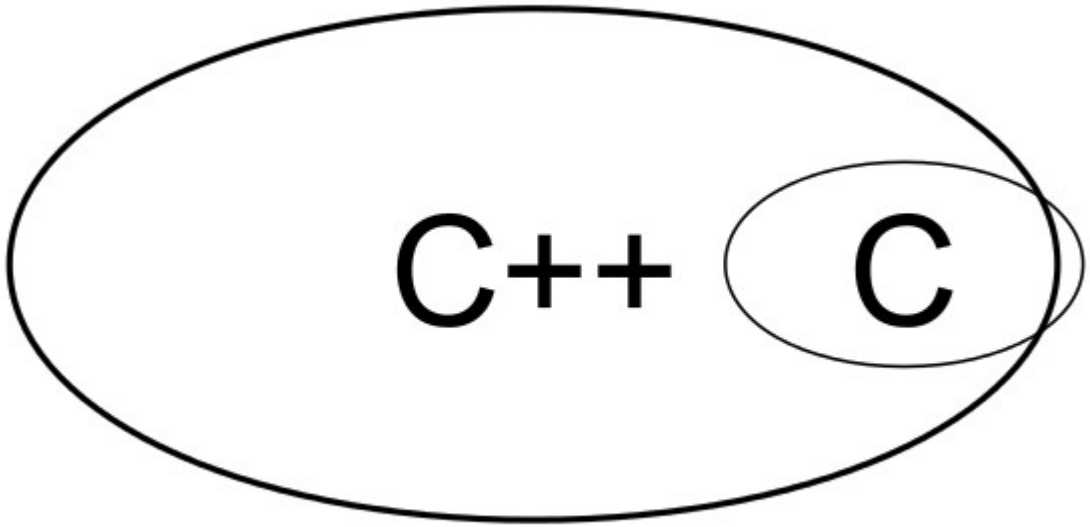
Diferenças entre C e C++

- **Bibliotecas padrões do C.** *#include*

EM C	DESCRIÇÃO	EM C++
<stdio.h>	Funções de E/S padrão	<cstdio>
<stdlib.h>	Funções diversas	<cstdlib>
<string.h>	Funções para manipulação de strings	<cstring>
<math.h>	Funções matemáticas	<cmath>
<time.h>	Manipulação de tempo/datas	<ctime>
<limits.h>	Constantes com os limites para tipos inteiros	<climits>

Diferenças entre C e C++

- **Comparativo**
  - C++ possui performance muito semelhante a do C.
  - C++ foi criada para ser uma extensão do C.
    - Mas não é 100% compatível.



Diferenças entre C e C++

- **Comparativo**

C	C++
Estruturada	Orientada a Objetos
malloc e calloc	**new**
free	**delete**
Passagem por valor	Passagem por referência
stdio	iostream
Variáveis declarada no início de um bloco	Variáveis declaradas em qualquer parte do bloco

Diferenças entre C e C++

- **Comparativo**

C	C++
Inteiro como valor booleano	Tipo **bool**
Duas funções não podem ter o mesmo nome	**new**
Argumentos são sempre necessários	Duas funções não podem ter o mesmo protótipo
Casts simples	Valor defaut para os argumentos
String como array de caracteres	Tipo string

Declaração de Classes

- São definições a partir das quais os objetos podem ser criados.

- As classes determinam quais são os atributos e métodos de um objeto.

- Sintaxe:

```
class nomeDaClasse {
    corpoDaClasse;
};
```

Declaração de Classes

- Exemplo:

```
 1  class Retangulo {
 2
 3      int largura;
 4      int altura;
 5
 6      int area() {
 7          return largura * altura;
 8      }
 9
10      int perimetro() {
11          return 2 *(largura + altura);
12      }
13  };
```

Declaração de Classes

- Membros de uma classe podem ser:
  - **public**

    Podem ser acessados em qualquer lugar
  - **private**

    Só podem ser acessados pelos membros da própria classe
  - **protected**

    Podem ser acessados apenas por membros da própria classe ou das suas sub-classes
- Obs.: Por default todo membro de uma classe é considerado **private.**

Declaração de Classes

- Exemplo:

```
1  class Retangulo {
2
3      int largura;
4
5      private:
6          int altura;
7
8      public:
9          int area() {
10             return largura * altura;
11         }
12
13     protected:
14         int perimetro() {
15             return 2 *(largura + altura);
16         }
17 };
18
19
20 // Exemplo de acesso:
21 int main() {
22     Retangulo r;
23     r.altura = 10; // Erro!
24     r.largura = 40; // Erro!
25     int a = r.area();
26     a = r.perimetro(); // Erro!
27 }
```

Declaração de Classes

- Quando implementamos um método dentro de uma classe o compilador copia e cola o código toda vez que o método é chamado!
  - O método é dito **inline**
  - Isto torna o executável mais rápido
  - Mas deixa o executável bem maior
  - Só é bom para métodos muito curtos

- Qual a solução?
  - Utilizar o operador **::**

Declaração de Classes

- Exemplo:

```
 1  class Retangulo {
 2
 3      private:
 4          int largura;
 5          int altura;
 6
 7      public:
 8          int area();
 9          int perimetro();
10  };
11
12  // força o método a ser inline
13  int inline Retangulo::area(){
14      return largura * altura;
15  }
16
17  int Retangulo::perimetro(){
18      return 2 *(largura + altura);
19  }
20
```

Declaração de Classes

- **Construtor e Destrutor**:
  - **Construtor**:
    - É um método especial que é chamado quando criamos um novo objeto
    - Deve possuir o mesmo nome da classe
    - É utilizado para inicializar os atributos da classe
    - Não possui retorno
  - **Destrutor**:
    - Método especial que é chamado automaticamente quando um objeto está prestes a ser apagado da memória
    - Deve ter o mesmo nome da classe mas precedido por um **~**
    - Ele não pode ter parâmetros e não possui retorno

Declaração de Classes

- Exemplo:

```
 1  class Retangulo {
 2
 3      private:
 4          int largura;
 5          int altura;
 6
 7      public:
 8          Retangulo(int a, int l);
 9          ~Retangulo(){} // dest. padrao
10  };
11
12  Retangulo::Retangulo(int a, int l) {
13      altura = a;
14      largura = l;
15  };
```

Declaração de Classes

- **Alocação de Memória**:
  - new
    - Aloca memória para um objeto
    - Retorna um ponteiro para a posição alocada
    - Exemplo:
      - Retangulo *r = new Retangulo(10, 15);
      - Retangulo *array = new Retangulo[10];
  - delete
    - Libera um região de memória alocada previamente
    - Exemplo:
      - delete r;
      - delete[] array;

Ponteiros

- Como é em C, porém para passar parâmetros por referências, utilizamos o símbolo **&**

- Exemplo:

```
void swap(int *a, int *b){} // em C

void swap(int &a, int &b){} // em C++

void soma(Retangulo &a, Retangulo &b){}
```

Templates

- Definem um algoritmo genérico e independente de tipo.

- Exemplo:

  - Busca em um array
  - Maximo entre elementos
  - Ordenação de um array
  - Mínimo entre elementos

Templates

- Exemplo:

```
1  template<class T>
2  T maximo(T a, T b){
3      return (a > b) ? a : b;
4  }
5
6  template<class T>
7  class Stack{...};
8
9  int main(){
10
11     int a = maximo(10, 45) ;
12     double d = maximo(10.5, 5.06);
13
14     // como a e b devem ter o mesmo
15     // tipo devemos usar o <double>
16     double c = maximo <double> (5, 5.06);
17
18     return 0;
19 }
```

Templates

- Exemplo:

```
1  template<class T>
2  T maximo(T a, T b){
3      return (a > b) ? a : b;
4  }
5
6  template<class T>
7  class Stack{...};
8
9  int main(){
10
11     int a = maximo(10, 45) ;
12     double d = maximo(10.5, 5.06);
13
14     // como a e b devem ter o mesmo
15     // tipo devemos usar o <double>
16     double c = maximo <double> (5, 5.06);
17
18     return 0;
19 }
```

Templates

- Permitem a criação de funções genéricas
  - Recebendo qualquer tipo de dado como parâmetro
  - Retornando qualquer tipo de dado

- Uma única função criada pode ser aplicada a qualquer tipo
- Sintaxe:
  - template <class identificador> funcao;
  - template <tipo identificador> funcao;

Namespaces

- Um namespace é um mecanismo para expressar um agrupamento lógico.
- Sintaxe:

```
namespace nomeDoNamespace {
    corpoDoNamespace
}
```

- Podemos utilizar um namespace para agrupar diversas funções, classes, variáveis ....

- Por exemplo, se tivermos muitas funções para realização de operações matemáticas podemos criar um namespace para todas

Namespaces

- Podemos acessar os membros de um namespace de duas maneiras diferentes:
  - Usando o operador **::**
  - Através do comando using namespace

```
1  #include <iostream>
2  #include "exemploNamespace.h"
3
4  using namespace Mat;
5
6  int main() {
7
8      std::cout << maximo(10, 56);
9      std::cout << minimo(10, 56);
10     std::cout << PI << std::endl;
11     NumeroComplexo c;
12     BigInteger b;
13
14     return 0;
15 }
```

Standard Template Library (STL)

- Olhar aula 8 curso C++ avançado realizado pelo Allan Lima.

Referências

[1] LIMA, A. Curso de C e C++ avançado . Disponível em: https://allanlima.wordpress.com/category/curso-de-c-e-c-avancado/

[2] CORMEN, T.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introdução à Algoritmos (Terceira Edição). MIT Press and McGraw-Hill. 2009.

Fim

Próximo dia:

Abstração dos problemas;

Estruturas de dados básicas;

Breve introdução sobre análise de complexidade;

Resolver problemas.

Dúvidas?

marcosvtreviso@gmail.com
facebook.com/marcos.treviso

Obrigado pela atenção!