

Técnicas de Programação

Anotações referentes a linguagem c++

Autor: Arthur Emanuell de Sousa Firmino

Data: 2026

Prefácio

Nesta anotação será falado um pouco sobre C++.

Sumário

- [Capítulo 1 - Criação de um Código](#)
- [Capítulo 2 - Variáveis](#)
- [Capítulo 3 - Funções](#)
- [Capítulo 4 - Espaço de Nomes](#)

Capítulo 1 - Criação de um Código

Para o que ocorra o processo de criação de um código em c++, é necessário que haja a seguinte estrutura:

- 1° Código Fonte em C++
- 2° Compilador C++
- 3° Código-Objeto
- 4° O linker
- 5° Programa executável

A seguir será exemplificado o passo a passo do processo de criação de um programa em C++.

1.1 Código Fonte em C++

O código fonte é a parte inicial de todo o projeto, na qual é necessário para que ocorra todo o decorrer da criação de um programa. Dentro dele é onde haverá a maior ênfase dentro deste PDF.

- Exemplo de um código fonte em C++

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

int main(){
    cout << "Hello Word";
    return 0;
}
```

1.2 Compilador C++

Para que ocorra o processo de compilação do código fonte necessitamos de um compilador chamado **g++**, com ele podemos criar o código objeto, apartir do seguinte comando:

```
g++ -c codigo.cpp
```

Observação: Onde está escrito **codigo** é o nome do arquivo.

Pórem, desta maneira não criamos o código executável por conseguinte não fazemos a linkedeção, apenas criamos o código objeto.

1.3 Código Objeto

Desta maneira já possui-se uma compilação para a criação de um arquivo **.o** que seria o arquivo que o processador do computador leria, no entanto, não existiria como modifica-lo.

1.4 Linker

O linker é o responsável pela linkagem, ou seja, é ele que irá compilar o código .o para o .exe, tornando assim o arquivo executável.

- É possível compilar o arquivo direto de .cpp para .exe, utilizando o seguinte comando.

```
//No caso do linux  
g++ -o hello.cpp hello
```

```
//No caso do Windows  
g++ hello.cpp -o hello
```

Caítulos 2 - Variáveis

Para que se possa construir um código em C++, geralmente há a necessidade de utilizar variáveis que possuem o intuito "guardar" algo, no entanto existe uma gama de variáveis a depender do meio em que será utilizada. Dentre os tipos de variáveis, temos: **int**, **float** e **char**.

2.1 - Int

O tipo de variável **int** (integer) é utilizado em C++ para armazenar valores numéricos inteiros, ou seja, números sem parte decimal, como valores positivos, negativos e o zero.

- Ocupa, normalmente, 4 bytes (32 bits) na maioria dos sistemas modernos
- Suporta operações aritméticas como: adição, subtração, etc.
- Exemplo:

```
int numero = 1;
```

A variável **int** também pode assumir dois tipos de divisões, números com sinais (signed) e números sem sinal (unsigned).

- Exemplo:

```
//Versão com sinal  
int numero_com_sinal = -12;  
//Versão sem considerar o sinal  
uint numero_sem_considerar-o-sinal = -12;
```

2.2 - Char

2.2.1 - Strings

São sequências de caracteres utilizadas para representar palavras, frases e textos completos.

- O C++ suporta strings no padrão C: **Array de caracteres**.
- Seu funcionamento é idêntico ao da linguagem C.

```
char str[6] = "OI";
```

No entando existe uma biblioteca chamada < string > que manipula sequências de caracteres de forma fácil e rápida.

```
string str
```

A incialização pode ser feita usando o operador de atribuição (=), por exemplo:

```
string str = "texto";
```

Podemos usar o operador (=), para atribuir o contúdo de uma string a outra.

```
string str1 = "Linguagem c++";  
string str2,str3;  
str2 = "Texto";  
str3 = str1;
```

Um modo mais adequado para ler uma string é usando o método `getline()`, pois com ele você consegue adicionar espaços, diferente se lessemos apenas com o **CIN**, e apenas deixará de ser lida quando for pressionado ENTER pelo o usuário, podemos usa-lo da senguinte maneira:

```
string str;  
getline(cin,str);  
cout << "A string lida foi" << str;
```

2.3 - Bool

A variável do tipo **Bool** não existe na linguagem ANSI-C e pode assumir dois tipos de valores: verdade (`true`) e falso (`false`).

- Exemplo:

```
bool v1 = false;  
bool v2 = true;
```

2.4 - Float

A variável do tipo **float** permite a utilização de casas decimais quando declarado algum valor, por exemplo:

```
float number = 0.5;
```

Capítulo 3 - Funções

Uma função é um bloco de código nomeado que executa uma tarefa específica. Ela pode receber valores de entrada (parâmetros), processar esses valores e retornar um resultado. O qual seu objetivo principal é:

- Organizar o Código.
- Evitar repetição.
- Facilitar a manutenção e reutilização.

3.1 Sintaxe

Para a construção de uma função é necessário que siga a sintaxe de escrita de uma função, na qual será estudada subdividida.

- Exemplo de uma função:

```
int somatorio(int a, int b){  
    return a+b;  
}
```

3.1.1 - Tipo de Retorno

Toda função deve ser especificado o tipo de retorno, pois é com ela que será baseada a saída da função. Dentre os tipos de retorno, temos eles: **int**, **float**, **char**, **void**, etc.

A seguir será demonstrado a declaração dos tipos de retorno e suas saídas:

- Int:

```
int somatorio(int a, int b){  
    return a+b;  
}
```

| A saída da função int, será um valor inteiro.

- Float:

```
float soma_medidas(float a, float b){  
    return a+b;  
}
```

| A saída da função float, será um valor decimal.

- Char:

```
char maiuscula(char letra) {  
    if (letra >= 'a' && letra <= 'z') {  
        return letra - 32; // converte para maiúscula  
    }  
    return letra;  
}
```

| A saída da função char, será uma variável char.

- Void

```
void hello(){
    cout << "Hello Word";
}
```

Pelo fato do void não retornar nada, não há a necessidade de colocar "return".

3.1.2 - Nome da função

O nome da função nada mais é que o identificador da função para auxiliar o programador.

3.1.3 - Lista de Parâmetros

Dentro da lista de parâmetros é onde iremos indicar a quantidade e o tipo das variáveis de entrada, por exemplo:

```
int somatorio(int a, int b){
    return a+b;
}
```

Note que foi especificado quantas varáveis e quais os tipos de variáveis entraram na função somatorio.

3.2 - Funções de Biblioteca

Além das funções definidas pelo próprio programador, o C++ disponibiliza um amplo conjunto de funções de biblioteca, prontas para uso.

A linguagem C++ possui sua biblioteca padrão, que oferece recursos para:

- Entrada e saída de dados
- Manipulação de strings
- Operações matemáticas
- Controle de erros
- Manipulação de tempo, caracteres, memória, entre outros

Além disso, o C++ mantém compatibilidade com a biblioteca padrão da linguagem C, disponibilizando 18 cabeçalhos herdados.

3.2.1 - Biblioteca <cmath>

- Função pow()

Na Função pow() temos a implementação de uma função que elevava o numero dependendo do que se colocar na condição, o qual possui a seguinte sitaxe:

```
float pow(float base, float exp);
```

Exemplo de código usando a função.

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

int main(){
    // typical usage
    std::cout << "pow(2, 10) = " << std::pow(2,10) << '\n'
    << "pow(2, 0.5) = " << std::pow(2,0.5) << '\n'
    << "pow(-2, -3) = " << std::pow(-2,-3) << '\n';

    // special values
    std::cout << "pow(-1, NAN) = " << std::pow(-1,NAN) << '\n'
    << "pow(+1, NAN) = " << std::pow(+1,NAN) << '\n'
    << "pow(INFINITY, 2) = " << std::pow(INFINITY,2) << '\n'
    << "pow(2,-INFINITY) = " << std::pow(2,-INFINITY) << '\n'
    << "pow(2,INFINITY) = " << std::pow(2,INFINITY) << '\n'
    << "pow(-INFINITY,1) = " << std::pow(-INFINITY,1) << '\n'
    << "pow(INFINITY, -1) = " << std::pow(INFINITY, -1) << '\n';

    return 0;
}
```

- Função sqrt()

Na função sqrt(), temos a implementação de uma função que realiza a raiz quadrada de um número, a partir da seguinte sintaxe:

```
float sqrt(float arg);
```

Exemplo de código usando a função.

```

#include <iostream>
#include <cmath>
int main(){

    // normal use
    std::cout << "sqrt(100) = " << sqrt(100) << '\n'
    << "sqrt(2) = " << sqrt(2) << '\n'
    << "golden ratio = " << (1+sqrt(5))/2 << '\n';

    // special values
    std::cout << "sqrt(-0) = " << sqrt(-0.0) << '\n';

    return 0;
}

```

- Função cbrt()

Na função cbrt(), temos a implementação de uma função que realiza a raiz cúbica de um número, a partir da seguinte sintaxe:

```
float cbrt(float arg);
```

Exemplo de código usando a função.

```

#include <iostream>
#include <cmath>
int main(){

    // normal use
    std::cout << "cbrt(729) = " << cbrt(729) << '\n'
    << "cbrt(-0.125) = " << cbrt(-0.125) << '\n';

    // special values
    std::cout << "cbrt(-0) = " << cbrt(-0.0) << '\n'
    << "cbrt(+inf) = " << cbrt(INFINITY) << '\n';

    return 0;
}

```

- Função hypot()

Na função hypot(), temos a implementação de uma função que realiza o cálculo da hipotenusa, a partir da seguinte sintaxe:

```
float hypot(float a, float b);
```

Exemplo de utilização da função:

```
#include <iostream>
#include <cmath>
int main(){
    float x,y,h;
    x = 3;
    y = 4;
    h = hypot(x,y);
    std::cout << "Hipotenusa igual a " << h << '\n';
    return 0;
}
```

- Funções Trigonométricas

A biblioteca cmath também possui funções que realizam o cálculo de funções trigonométricas, como por exemplo:

```
float sin(float theta);
float cos(float theta);
float tan(float theta);
```

Também é possível designar funções inversas, ou seja, o arco seno, o arco cosseno e o arco tangente.

```
float asin(float theta);
float acos(float theta);
float atan(float theta);
```

Capítulo 4 - Espaço de Nomes

É uma região declarativa que vincula um identificador a um grupo de variáveis, funções, subespaços, estruturas, constantes, etc.

- Exemplo:

```
namespace maria{
    int idade = 12;
    int cra = 8.3;
}
```

```
namespace jose{
    int idade = 21;
    int cra = 4.3;
}
```

No exemplo, idade e cra fazem parte do namespace **maria** e **jose**, ou seja, podemos criar variáveis de nomes iguais dentro de namespaces diferentes que mesmo assim não existirá conflito entre as variáveis.

- Exemplo 2: Qual seria a saída deste código?

```
#include <iostream>
namespace maria {
int idade = 18;
float cre = 8.3;
}
namespace jose {
int idade = 21;
float cre = 4.3;
}
int main(){
    int idade = 12;
    std::cout << idade << " " << maria::idade << std::endl;
    std::cout << jose::idade << std::endl;
return 0;
}
```

A saída deste código será:

```
12 18
21
```

Capítulo 5 - Operadores e Expressões

Neste capítulo será retratado sobre a utilização de expressões e operados, os quais fazem parte de quase todos os códigos feitos em c++.

5.1 - Expressões

É uma sequência de operandos e operadores que especifica uma operação.

- Exemplo

X = Y + 15

Expressão: Y + 15;

Operador: +;

Operandos: Y e 15.

5.2 - Operadores

5.2.1 - Definição

O significado de um **operador** depende dos tipos de seus operandos: M + N.

O operador + pode ser:

- Soma de inteiros
- Soma de pontos flutuantes
- Concatenação de strings
- Soma de ponteiros

5.2.2 - Tipos de Operadores

5.2.2.1 - Aritméticos

Dentre os tipos de operadores aritméticos temos:

- Subtração
- Adição

- Multiplicação
- Divisão
- Resto da Divisão

Sempre existe uma ordem de precedência entre eles, na qual é a mesma em matemática.

Caso seja necessário que siga uma precedência forçada, basta apenas utilizar os parênteses () .

5.2.2.2 - Operadores Lógicos e Relacionais

As expressões que usam operadores relacionais ou lógicos devolvem zero para "falso" e 1 para "verdadeiro" (true ou false). Dentre os tipos de operadores logicos e relacionais temos:

- ($>$) - Maior que
- (\geq) - Maior ou igual que
- ($<$) - Menor que
- (\leq) - Menor ou igual que
- ($=$) - Igual
- (\neq) - Diferente
- ($\&\&$) - AND
- ($\|$) - OR
- ($!$) - NOT

5.2.2.3 - Operadores de Atribuição

A fim de simplificar a maneira de atribuir valores à variáveis, pode ser utilizados diferentes tipos de operadores de atribuição, são eles:

Símbolo	Uso	Operação
=	<code>x = 6;</code>	Atribui 6 a x
*=	<code>x *= 6;</code>	<code>x = x * 6;</code>
/=	<code>x /= 6;</code>	<code>x = x / 6;</code>
%=	<code>x %= 6;</code>	<code>x = x % 6;</code>
+=	<code>x += 6;</code>	<code>x = x + 6;</code>
-=	<code>x -= 6;</code>	<code>x = x - 6;</code>
<<=	<code>x <<= 6;</code>	<code>x = x << 6;</code>
>>=	<code>x >>= 6;</code>	<code>x = x >> 6;</code>
&=	<code>x &= 6;</code>	<code>x = x & 6;</code>
^=	<code>x ^= 6;</code>	<code>x = x ^ 6;</code>
=	<code>x = 6;</code>	<code>x = x 6;</code>

5.2.3 - Procedência

Todo operador possui uma precedência, na qual podemos visualizar da seguinte maneira:

- 1º Operadores Aritméticos (Seguindo a ordem da matemática)
- 2º Operadores Lógicos e Relacionais.

Dentro do conceito, temos a associatividade, na qual segue apartir da esquerda para a direita (L-R).

5.2.4 - Classificação

- Unário: Atuam sobre um operando: &P
- Binário: Atuam sobre dois operandos: X * Y
- Ternário: Atuam sobre três operandos: a?b:c