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

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 proceso 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 entando, não existiria como modifica-lo.

1.4 Linker

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

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

```
//No caso o 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 intuide "guardar" algo, no entando 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 = sr1;
```

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 contrução 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 retono, 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 falor 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 indentificador 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 hypotenusa, a partir da seguinte sitaxe:

```
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 bilbioteca cmath tabém possui funções que realizam o cálculo de funções trignométricas, 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 confito 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 procência entre eles, na qual é a mesma em matemática.

> Caso seja necessário que siga uma precedencia 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
- (>=) - Maior ou igual que
- (<) - Menor que
- (<=) - Menor ou igual que
- (==) - Igual
- (!=) - 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
=	x = 6;	Atribui 6 a x
*=	x *= 6;	x = x * 6;
/=	x /= 6;	x = x / 6;
%=	x %= 6;	x = x % 6;
+=	x += 6;	x = x + 6;
-=	x -= 6;	x = x - 6;
<<=	x <<= 6;	x = x << 6;
>>=	x >>= 6;	x = x >> 6;
&=	x &= 6;	x = x & 6;
^=	x ^= 6;	x = x ^ 6;
|=	x |= 6;	x = x | 6;

5.2.3 - Procedência

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

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

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

5.2.2.4 - Operadores de Deslocamento

Como todo programa que possua interação com uma máquina sempre terá numeros convertidos para binário, é possível utilizar operadores de deslocamento bínario para se fazer determinadas operações, exemplo:

```
X = 2;
X<<=1;
//Isso implica dizer que o número 2 (0010) será deslocado para a esquerda 1 bit, ficando igu
```

Tipos de operadores de deslocamento:

- (<<) - Irá mover os bits para a esquerda
- (>>) - Irá mover os bits para a direita

5.2.2.5 - Operadores de Incremento e Decremento

Os operadores de Incremento e Decremento podem auxiliar de maneira a simplificar o código e também em momentos que necessita de situações específicas, a seguir será mostrado os operadores:

```
++X //Significa que o X = X + 1;
--X // Significa que o X = X - 1;
```

No entando existem difenrenças na utilização destes operadores, pois a maneira que se escreva ele, pode mudar a maneira de que seja atribuida a variável.

```
#include <iostream>
using namespace std;
int main () {
    int y, x = 3;
    y = ++x; // x será 4, y será 4
    cout << x << endl;
    cout << y << endl;
}
```

Diferente de:

```
#include <iostream>
using namespace std;
int main () {
    int y, x = 3;
    y = x++; // x será 4, y será 3
    cout << x << endl;
    cout << y << endl;
}
```

5.2.2.6 - Operadores de Manipulação Bit-a-Bit (bitwise)

Quando se fala em operadores bit a bit, entende-se que as operações relacionadas aos números não ocorrerão de maneira simples, ou seja, o numero sera visto de maneira mais aprofundada, temos como operadores de manipulação bit-a-bit:

- (&) - AND bit-a-bit
- (|) - OR bit-a-bit

- (^) - XOR bit-a-bit
- (~) - INVERSOR bit-a-bit
- (<<) - Deslocamento de bits à esquerda
- (>>) - Deslocamento de bits à direita

Pode ser utilizado a mesma sitaxe do verilog:

```
0b0000001 // isso é a mesma coisa que 1 em decimal.
```

5.2.2.7 - Operador condicional ternário (Parecido com o IF)

O operador condicional é bastante parecido com o IF, no entando ele necessita de apenas uma linhan para ser escrito (contém limitações).

```
y = (x<20) ? 50 : 70;
//Y = 50 se x<20 senão Y = 70.
```

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`

Capítulo 6 - Conversão de tipos

6.1 - Conversões Implícitas

São aquelas executadas automaticamente e tem como regra a promoção de tipos:

- Tipo de precisão mais baixa --> Tipo de precisão mais alta.

Exemplo:

```
float pi = 3.14 + 3;
```

O valor de 3 que é do tipo inteiro, é convertido para o tipo float, concluindo a conta e chegando a 6.14.

No entanto, caso tenhamos definido que pi é do tipo int, a sequência tomorá um destino diferente do 1° exemplo.

```
int pi = 3.14 + 3;
```

O valor 3 será convertido para float, para que haja a soma do valor float (3.14) mais o valor de 3 convertido para float. No entando, logo após realizada a soma o valor deixa de ser decimal e é convertido para inteiro novamente, visto que o tipo de variável declarada no do código seria do tipo inteiro, obtendo-se como valor final 6.

6.2 - Conversões Explícitas

A conversão explícita ocorre quando o programador força a conversão de um tipo de dado para outro, informando isso diretamente no código.
Em C++, esse processo é chamado de type casting.

Ela é utilizada quando se deseja controlar exatamente como a conversão deve acontecer, evitando comportamentos inesperados.

- Sintaxe

```
tipo_de_dado(expressão);
(tipo_de_dado) expressão;
```

- Exemplo

```
double a=3.27;
int b;
b = int (a); //casting de double para int
b = (int) a; //casting de double para int
```

Capítulo 7 - Programação Estruturada

A base da programação estruturada é que um programa dever ser composto por **blocos de código** (procedimentos) que se interligam através de três estruturas de controle:

- Sequência

- Seleção
- Iteração

7.1 - Sequência

Passos de procedimento ordenados de um programa.

- Exemplos

```
"primeiro faça a Tarefa A e depois a Tarefa B"
```

Pode ser reprensentada em pseudocódigo ou em fluxograma.

7.2 - Estrutura de Controle de Seleção

A estrutura de controle de seleção é um mecanismo da programação que permite ao programa tomar decisões, escolhendo diferentes caminhos de execução de acordo com o resultado de uma condição lógica.

Por meio dessa estrutura, o fluxo do programa deixa de ser sempre sequencial, possibilitando a execução de blocos distintos de código, conforme uma condição seja verdadeira ou falsa.

Em C++, as principais estruturas de seleção são o if, if–else e o switch.

7.2.1 - Comando IF-ELSE

O comando de controle de seleção mais simples é o IF, onde o usuário irá fornecer a condição e logo após será executado o bloco de comandos, segue a sintaxe da estrutura:

```
if(expressão) comando;

//ou

if(expressão){
    bloco de comandos;
}else{
    bloco de comandos;
}
```

- Exemplo de um código utilizando a estrutura de condição IF.

```
#include <iostream>
using namespace std;
int main(){
    float nota;
    cout << "Digite a nota do aluno" << endl;
    cin >> nota;
    if (nota >= 7)
    cout << "Aprovado" << endl;
    if (nota < 7)
    cout << "Não aprovado" << endl;
    if (nota > 4)
    cout << "Final" << endl;
    if (nota <= 4)
    cout << "Reprovado" << endl;
}
return 0;
```

Durante a estruturação do if, pode se utilizar estruturas aninhadas, ou seja, você poderá escrever um if logo após outro, exemplo:

```
if (x > 0)
    cout << "x eh positivo";
else if (x < 0)
    cout << "x eh negativo";
else
    cout << "x eh 0";
```

7.2.2 - Comando Swith-case

A estrutura switch-case é um pouco diferente da estrutura if, no entanto segue o mesmo propósito de construir uma estrutura de condição mas o case possui peculiaridades nas quais são bem aproveitadas, como na utilização para a criação de menus.

- Sintaxe da estrtura switch-case:

```
switch(expressão_de_condição){
    case constante1:
        grupo_de_comandos_1;
        break;
    case constante2:
        grupo_de_comandos_2;
        break;
    default:
        grupo_de_comandos_padrões;
}
```

7.3 - Estrutura de Controle de Iteração

A parte de estrutura de controle de iteração funciona como a execução repetitiva de bloco de códigos do programa.
A forma básica de iteração é pela construção do **While/Do** ou **Do/While**.

- Sintaxe do While:

```
while(expressão){

}
```

- Sintaxe do Do-While:

```
do{

}while();
```

> Pode se criar uma estrutura de repetição "infinita" apenas inserindo **true** dentro da condição do while.

Na estrutura de repetição é possivel fazer uso do mesmo comando visto anteriormente, o comando **break**, o qual encerra o bloco de código saindo assim da estrutura de repetição.

Na estrutura de repetição também temos a adição de uma ferramenta muito útil durante o processo de programação, o código **continue**, este comando faz com que após ele, nada seja executado no bloco da estrutura de condição e acabe voltando para o inicio da estrutura de condição.