SH12C++

Uma introdução de C++

Programação do "Hello World" a Arrays

27 de Abril de 2023 às 02:02

Introdução

Esse é um tutorial rápido de **programação em C++**, espero que ele sirva a você que está **começando a aprender programação** agora, assim como a você que **já possui algum conhecimento de programação**, mas em outra linguagem, e agora está migrando para o C++. Como este é um resumo, podem existir, e existem, muitas informação que acabaram ficando de fora, espero que apenas com o que você vir aqui já seja capaz de **criar seus primeiros códigos e iniciar sua jornada na programação**, mas saiba que **há muito mais para descobrir** e você pode, e deve, **buscar outras fontes** e tentar se aprofundar mais.

> O que vamos ver agora?

O que é Programação? E C++? E Algoritmos???

Na programação o que fazemos é escrever **algoritmos** através de **linguagens de programação**, os códigos. Esses algoritmos então são transformados em programas de computador.

Mas **o que é um algoritmo**? Algoritmo é uma **sequência de ações executáveis** que visam obter uma solução para um problema, sejam essas ações de qualquer tipo. Pode-ser dizer, por exemplo, que uma receita de bolo é um algoritmo! Outra definição pode dizer "**Algoritmos são procedimentos precisos, não ambíguos, padronizados, eficientes e corretos**." Essa definição se aproxima mais do que chamamos de algoritmo na programação, porque ainda que uma receita de bolo possa ser considerada uma *sequência de ações*, ela não possui o rigor necessário de um programa de computador. O computador não é capaz de entender sutilezas que estão implícitas na nossa comunicação humana (salvo ChatGPT, mas isso não vem ao caso), assim a maioria, se não todas, as ações de uma receita de bolo são ambíguas e não precisas.

Na programação **utilizamos algoritmos escritos em códigos**, através das **linguagens de programação**. Exemplos: **Python**, **C++**, **Java**, **PHP**, entre outros. Mas **por que temos de usar uma linguagem de programação?** Como dito no parágrafo anterior, nossa linguagem humana está cheia de ambiguidades e não é entendida pelo computador. Enquanto usamos palavras e frases para nos comunicarmos, as máquinas utilizam 0s e 1s, a linguagem binária. Mas escrever algoritmos com 0s e 1s também é impraticável para nós. Assim **as linguagens de programação funcionam como uma ponte** entre esses dois extremos, com elas somos capazes de escrever códigos **precisos** e **livres de ambiguidades**, mas ainda assim **entendíveis** para nós. O C++ é uma dessas linguagens e a que vamos utilizar aqui.

Onde programar?

Normalmente utilizamos programas como o **Visual Studio Code**, o **Sublime**, o **Vim**, entre outros. Esses são **editores de texto**, eles nos ajudam a organizar o código, detectar erros de sintaxe e agilizam a tarefa de codificar mostrando dicas. Mas eles servem apenas pra isso, pra editar o texto do seu código, para realmente se obter um programa funcional a partir das suas linhas de código é necessário um **Compilador**, que vai pegar seu código e transformá-lo em um programa propriamente dito, um arquivo executável.

Mas a instalação de um compilador pode ser um pouco trabalhosa, então se você quiser já começar a testar seus primeiros códigos, existem algumas plataformas online que podem fazer o trabalho pra gente. Uma plataforma exelente pra se começar é o Replit, nele podemos escrever nossos códigos e executar logo em seguida. Basta entrar na plataforma, se cadastrar e ir na opção "*+ Create Repl*", selecionar o C++ como linguagem e pronto! Uma aba vai se abrir já com algumas linhas de código e um ambiente para você começar a programar. Se você quiser começar a treinar seus códigos agora, enquanto segue aqui a aula, já é uma exelente forma de começar a praticar!

Durante o tutorial vou também recomendar algumas questões de programação para você resolver, essas questões estarão no site Beecrowd, então recomendo que já faça seu cadastro no site e tente resolver as questões conforme forem apresentadas, isso irá lhe ajudar a fixar melhor o conteúdo visto.

Mas você eventualmente precisará de um Compilador, não há como escapar. Para **instalar um compilador de C++** recomento que siga esse tutorial. Ele está em inglês, mas é fácil de entender e muito claro quanto ao passo a passo. Como editor de texto recomendo o **Visual Studio Code**, que pode ser instalado por aqui.

Estrutura básica

No C++ temos que respeitar um certo padrão, chamado de **Sintaxe**. A sintaxe dita quais palavras possuem quais funções e como devemos estruturar nosso código para que ele possa ser entendido pelo computador. Para iniciar nosso código temos de utilizar uma estrutura como essa aqui:

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

int main() {

    //O seu código vai aqui

    return 0;
}
```

Mas, o que cada uma dessas coisas faz? Pra que isso tudo??? Vamos por partes.

Primeiro, o `#include <iostream>`: aqui estamos "incluindo", ou **importando**, a biblioteca IO (input e output), isso é necessário para que o programa saiba que desejamos utilizar os recursos de entrada e saída, além de mais alguns outros. Mas essa parte do código é padrão e não precisaremos nos preocupar com isso.

O `using namespace std;` utilizamos para declarar que estamos utilizando coisas da biblioteca *standart*, que significa "padrão". Essa linha não é necessária, podemos escrever nosso código sem ela, mas se o fizermos teremos de escrever algumas coisas do nosso código com um `std::`, então, apenas por comodidade, podemos adicionar ela para facilitar a leitura e escrita do código.

Agora a parte mais importante da base do nosso código! O `int main( )` é nossa função ou "método" principal (*main* do inglês *principal)*, e dentro dele, ou seja, **entre as duas *chaves*** `{}` que indicam o início e o fim da função **estará contido todo o código**. Essa função será chamada quando o programa iniciar sua execução e então **todos os comandos dentro dela serão chamados sequencialmente**, linha por linha. Isso até chegar no `return 0;`, ele indica o **fim do programa** e também que o programa foi executado sem nenhum erro.

Ainda falta uma última coisa antes de prosseguirmos: O que é `int`, em `int main( )`? Apesar de ser a parte principal do código o método *main* nada mais é que uma função como qualquer outra, nesse caso uma função do tipo `int` (os tipos serão explicados mais adiante), e por isso também é necessário que haja um *return*, como toda função. Mas **por enquanto, não se preocupe com isso**, apenas tenha em mente que a **parte principal de seu código** deve ser escrita dentro da função main!

Saída

A saída é a forma que o nosso programa se comunica com o mundo. No nosso programa a saída padrão vai consistir em textos que nosso código vai escrever no console. O console vai ser a principal forma de comunicação entre nós e nosso programa.

Mas lembra sobre a sintaxe? Que nosso programa deve respeitar algumas estruturas pré-definidas, isso inclui algumas palavras que são reservadas pela linguagem de programação e que têm algum significado e função, como a saída, que será determinada por uma palavra e alguns símbolos. No C++ a saída padrão é o `cout`, cuja escrita é *C* (C++) mais *Out* (Output). Podemos escrever nosso primeiro código assim:

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

int main() {

    cout << "Ola Mundo";

    return 0;
}
```

Primeiro escrevemos **cout** para indicar que queremos usar a **saída** e logo na frente, **separado por dois "*menor quê*"** `<<` escrevemos a **mensagem** que queremos pôr na saída. Lembrando que **textos devem ser sempre escritos entre aspas duplas** `"texto"` , para não serem confundidos com o código pelo nosso programa. Outro lembrete é que devemos **sempre finalizar cada comando com um ponto-e-vírgula** `;`.

Se quisermos deixar esse código ainda melhor, podemos adicionar também um `endl` para sinalizar uma **quebra de linha** na saída, como se tivéssemos apertado um *enter* após escrever a mensagem e pulado para a próxima linha, assim:

```
cout << "Hello World" << endl;
```

`endl` significa *end line*, e indica justamente isso, o fim da linha. Veja também que podemos colocar várias mensagens uma após a outra apenas separando elas com `<<`.

Recomendo que resolva a questão Hello World! para treinar um pouco a saída.

Variáveis e Tipos

Certo, mas não queremos apenas escrever coisas, podemos também armazenar informações em variáveis e acessar essas informações mais tarde durante a execução do nosso programa. Essas variáveis têm tipos, que indicam que tipo de dado elas armazenam, seja um texto, um número ou um valor booleano.

Os tipos primitivos que o C++ possui são:

- `int` - armazena números **inteiros**, como `1`, `2`, `3`, `4`, `1000` ou `1000000007`.
- `float` - armazena números de "**ponto flutuante**", ou seja, **decimais**, como `1.3`, `0.004`, `3.33333` e assim por diante.
- `double` - também é "**ponto flutuante**", contudo possui o **dobro de precisão** do float e é mais recomendado na programação competitiva.
- `bool` - guarda valores **booleanos**, ou seja, **verdadeiro** ou **falso**, 0 ou 1, `true` ou `false`.
- `char` - guarda **caracteres** ASCII, como letras `'a'`, `'s'`, `'h'`, números `'1'`, `'2'`, ou mesmo caracteres como `'!'`, `')'` e etc.
- `string` - guarda **textos**, como `"Isso e um texto"`. (Contudo, strings não são tipos primitivos, elas são na verdade *cadeias de caracteres*, como uma lista de *char*)

No C++ devemos declarar variáveis com o seguinte formato: `tipo nome = valor;`, sendo o *valor* um atributo opcional para se declarar a variável. Veja um exemplo de como podemos utilizar variáveis:

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

int main() {

    string nome = "Samuell";
    int idade;

    idade = 18;

    cout << "Ola, meu nome é " << nome << endl;
    cout << "E eu tenho " << idade << " anos!" << endl;

    //Outros exemplos
    float altura = 1.83;
    double saldo = 2.50;
    bool aluno = true;
    char inicial = 'S';

    return 0;
}
```

Observe que na primeira variável atribuí um valor para ela na declaração, já na segunda variável declarei a variável e só depois atribuí um valor a ela. Outra informação importante: no C++ devemos utilizar **aspas simples** `'a'` para caracteres e **aspas duplas** `"texto"` para strings.

Para realizar operações matemáticas podemos utilizar os seguintes operadores:

- `+` - **adição**
- `-` - **subtração**
- `*` - **multiplicação**
- `/` - **divisão**
- `%` - **resto da divisão**

Você pode utilizar esses operadores para realizar operações entre dois números `int soma = 2 + 5;`, entre uma variável e um número `int soma = valorA + 3;`, entre duas variáveis `int soma = valorA + valorB;` e até atribuir o valor de uma variável somada com algum valor a ela mesma! `soma = soma + 1;` Mas nesse caso a variável precisa já ter sido declarada com algum valor anteriormente.

Antes de tudo, tente **praticar alguns códigos** realizando **operações matemáticas** com alguns números e veja o resultado dessas operações. Outra dica: É possível somar strings! Então se você fizer `string texto = "Texto 1" + " " + "Texto 2";`, o resultado será `"Texto 1 Texto 2"`.

Entrada

Já aprendemos como escrever e como armazenar valores em variáveis, o próximo passo agora é aprender a ler, como acessar a entrada.

No C++ a entrada padrão é o `cin`, que é formado pelas palavras C++ e Input. O valor da entrada sempre será armazenado em uma variável, assim a grafia que devemos utlizar no *cin* é `cin >> variavel;`, observe que diferente do *cout o cin* deve ser escrito com dois "*maior quê*" `>>` para separar as variáveis. Veja um exemplo como o acima:

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

int main() {

    string nome;
    int idade;

    cout << "insira seu nome: ";
    cin >> nome;

    cout << "insira sua idade: ";
    cin >> idade;

    cout << "Ola, meu nome e " << nome << endl;
    cout << "E eu tenho " << idade << " anos!" << endl;

    return 0;
}
```

Experimente criar alguns códigos utilizando a etrada. Para praticar tanto entrada como variáveis, resolva esses exercícios: Extremamente Básico, Produto Simples, Média 1, Tomadas e Pedágio. Esse último exigirá também um pouco de lógica!

Estruturas Condicionais

Já havendo praticado a entrada e as variáveis, vamos para o para o próximo passo: estruturas condicionais!

As estruturas condicionais, ou de decisão, servem para que possamos executar nosso código com base em decisões lógicas, algo como "*se essa condição for verdadeira faça isso. Se for falsa faça aquilo*". E aqui utilizamos duas palavrinhas importantes: `if` e `else`. O *if* é basicamente a tradução de inglês para "*SE*" e o *else* é igualmente apenas um "*SENÃO*". *Vamos utilizá-las assim:*

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

int main() {

    bool condicao = true;

    if( condicao == true ){

        cout << "Condicao verdadeira" << endl;

    } else {

       cout << "Condicao falsa" << endl;

    }

    return 0;
}
```

Veja, ***SE*** nossa condição entre os parênteses `( )` após o `if` for verdadeira nosso código executará o primeiro *cout*, que está contido entre as chaves `{}` do *if* , ***SE NÃO*** ele executará o segundo *cout*, contido entre as *chaves* `{}` do `else`. Observe também que ao "perguntarmos" se a condição é verdadeira utilizamos dois *iguais* `==`, isso é importante para o código não confundir nossa "pergunta" com uma "afirmação"', então tome cuidado com isso. Mas claro que existem outros *operadores lógicos:*

- `==` - **Igual**: retorna verdadeiro se os objetos da comparação forem **iguais** e falso caso o contrário.
- `!=` - **Diferente**: retorna verdadeiro se os objetos comparados forem **diferentes**.
- `<` - **Menor quê**: retorna verdadeiro se o objeto da esquerda for **menor que** o da direita.
- `>` - **Maior quê**: retorna verdadeiro se o objeto da esquerda for **maior que** o da direita.
- `<=` - **Menor ou igual quê**: retorna verdadeiro se o objeto da esquerda for **menor ou igual** que o da direita.
- `>=` - **Maior ou igual quê**: retorna verdadeiro se o objeto da esquerda for **maior ou igual** que o da direita.

Assim podemos fazer algo dessa forma:

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

int main() {

    int idade = 18;

    if( idade >= 18 ){
        cout << "Você é maior de idade" << endl;
    } else {
       cout << "Você é menor de idade" << endl;
    }

    return 0;
}
```

Mas ainda tem mais, podemos ainda colocar mais de uma condição no mesmo *if* e executar as ações "*se essa condição for verdadeira* ***E*** *essa outra condição for verdadeira também*" ou "*se essa condição for verdadeira* ***OU*** *essa outra condição for verdadeira*", *para isso podemos utilizar:*

- `&&`, `and` - São o ***E***, ele exige que as duas condições sejam verdadeira para retornar verdadeiro.
- `||`, `or` - São o ***OU***, ele exige que apenas uma das condições seja verdadeira retornar verdadeiro.

Você pode uni-los de diferentes formas, utilizando mais parênteses se quiser utilizar combinar mais de um deles e utilizar como se fossem portas lógicas.

Por último, ainda podemos concatenar vários *Ifs* e *elses* para criar estruturas ainda mais complexas, como no exemplo abaixo:

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

int main() {

    float nota = 5.5;

    if( nota >= 7 ){
        cout << "Você esta aprovado" << endl;
    } else if( nota < 7 && nota >=4 ){
        cout << "Você esta de recuperacao" << endl;
    } else {
       cout << "Você esta reprovado" << endl;
    }

    return 0;
```

```
}
```

Tente fazer primeiro um código semelhante ao de cima, mas recebendo duas notas do usúário, calculando a média e então imprima "Aprovado" se a nota for maior ou igual a 7, "Recuperacao" se a nota for menor que 7 mas maior ou igual a 4, ou "Reprovado" se a nota do aluno for menor que 4.

Tente agora resolver essas três questões: Identificando o Chá, Tipos de Triângulos e O Maior. Neste último é informada uma fórmula, ignore e utilize apenas `if` e `else`, se seu código não funcionar de primeira analise o que ocorre se os três números forem iguais e como seu código está se comportando quando isso ocorre.

Estruturas de Repetição

Agora, pressupondo que você já conseguiu entender os tópicos anteriores e resolver as questões propostas, vamos para o próximo passo, as estruturas de repetição!

Pense no seguinte problema: Você deverá receber um número inteiro da entrada, que chamarei de N, e então deverá escrever na saída todos os números entre 0 e N. Além disso N é um número entre 1 e 1000000 (1 <= N <= 1000000). Como você poderia fazer isso? Não é viável escrever centenas de milhares de `if`'s e `cout`'s no seu código. Podemos então repetir um mesmo pedaço de código várias vezes, ou exatamente N vezes nesse caso.

No C++ temos duas estruturas de repetição, o `for` e o `while`, algo como "***para***" *e* "***enquanto***" traduzindo do inglês.

O **while** recebe uma condição entre parênteses e então repetirá o código dentro dele **enquanto** essa condição for verdadeira. Poderíamos resolver o problema anterior com um código mais ou menos assim:

```
int n;
cin >> n;

int contador = 0;

while( contador <= n ){
        cout << contador << endl;
        contador = contador + 1;
}

cout << "Fim do código" << endl;
```

No código acima temos uma variável `contador`, que guardará o número que queremos imprimir, nesse caso começaremos pelo número 0. O laço de repetição `while` agora irá "testar" se a condição é verdadeira, sendo verdadeira ele executará o código essa vez, imprimindo a variável `contador`, então aumentamos o valor de contador em `+1`. Agora que a execução desse pedaço de código terminou o `while` vai mais uma vez testar se a condição é verdadeira, e assim, enquanto a condição continuar verdadeira o código dentro do `while` continuará a executar. No nosso código, assim que a variável `contador` alcançar um número maior que N o `while` deixará de repetir esse pedaço de código.

Já o **for** funciona de uma forma um pouco mais prática, mas com mais parâmetros também. O `for` vai assumir o seguinte "formato": Para(*variável ; condição ; ação final*), assim: `for(int contador = 0; contador <= n; contador++)`. Seria algo como "***Para*** *a variável contador; enquanto contador for menor que N; e ao fim de cada iteração aumente o valor de contador*". Veja como ficaria o código acima com o `for` em vez do *while*:

```
int n;
cin >> n;

for( int contador=0;  contador <= n ;  contador++ ){
        cout << contador << endl;
}

cout << "Fim do programa" << endl;
```

Veja que o código ficou um pouco mais "*compacto*". Claro que tudo que você pode fazer com o `for` você pode fazer com o `while` e vice-versa, mas para diferentes problemas um pode ser mais prático que o outro, então é importante ter domínio dos dois. Observe que o funcionamento do `for` segue a mesma lógica do `while`.

Tente resolver os seguintes problemas: Números Pares, Ho Ho Ho, Fibonacci Fácil e Mjölnir.

Arrays

Imagine agora um problema semelhante ao anterior: Você recebe um inteiro N e então terá de ler N números em seguida. Semelhante ao problema anterior, não podemos criar incontáveis variáveis no código para armazenar cada número em uma variável diferente, o melhor seria se pudéssemos armazenar todos os valores numa mesma variável, algo como uma **lista**, uma variável capaz de armazenar vários valores. No C++ podemos criar essa "lista" com um **Array**.

O array é declarado como uma variável, mas com a seguinte semântica: `tipo nome [tamanho]`, em que *tamanho* é o tamanho da lista. O array possui tamanho estático, então ele será criado com um tamanho e permanecerá com o mesmo tamanho durante toda a execução do código. Você deve acessar o elemento pelo índice dele, `nome[indice]`, assim o *primeiro* elemento é o elemento `[0]`, o segundo é o `[1]`, o terceiro é o `[2]`, o quarto o `[3]` e assim por diante. Pode parecer meio estranho no começo, mas no C++ e na maioria das linguagens de programação os elementos começam a ser contados a partir do 0. Assim o N-ésimo elemento é na verdade o [N-1].

Veja um exemplo de como criar um array:

```
int tamanho = 10;

int lista [ tamanho ];

for(int i=0; i < tamanho; i++){
        cin >> lista[i];
}

cout << "O primeiro elemento da lista é : " << lista[0] << endl;

int lista2 [4] = {0, 0, 0, 0};
```

Também é possível inicializar um array colocando os valores que você deseja entre chaves `{}`, listando todos os valores entre vírgula.

Quando você for acessar um array é preciso tomar cuidado para não acessar posições inválidas, por exemplo, se você tiver um array de tamanho 10 e tentar acessar a posição 12 você receberá um erro, pois essa posição não existe, da mesma forma, se você acessar a posição 10 também receberá um erro, pois lembre que, como o array começa do índice 0 e assim o elemento 9 desse array já é o último. Existem outros tipos de lista em C++, mas essa é a principal delas.

Pratique com as questões: Troca em Vetor I, Substituição em Vetor I e Saltos Ornamentais.

Considerações Finais

Isso é tudo que você precisa saber para começar a praticar seus códigos. Ainda há bastante coisa pra aprender, e, principalmente, para praticar. Quanto mais você praticar, melhor vai ficar, então tente resolver questões e procurar mais fontes para aprender. As plataformas **Beecrowd**, **Neps Academy** e **NOIC** são um ótimo ponto de partida, mas a partir daí procure expandir ainda mais suas fontes! Para qualquer dúvida, sugestão ou crítica, entre em contato!

Boa sorte e até a próxima!

Comentários