cppreference.com
https://www.cplusplus.com/
Doc Microsoft
C++ ISO/20
https://youtube.com/playlist?list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV

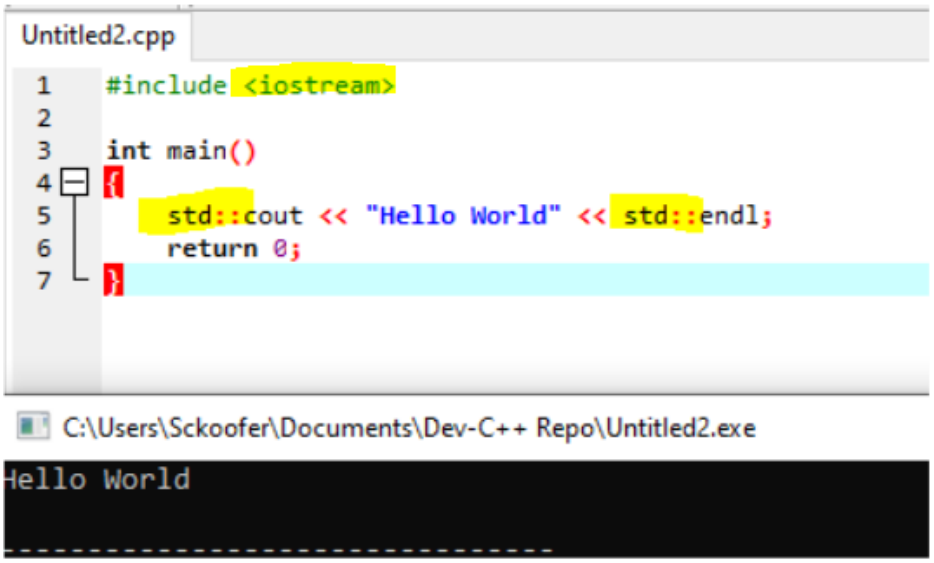
C++

Compilação. O G++ é um componente do GCC (Compilador padrão do GNU) usado para compilar programas em C++.

Curso de C++ - Aula 02 - Hello World

https://www.youtube.com/watch?v=qZziRrj55H4&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=2

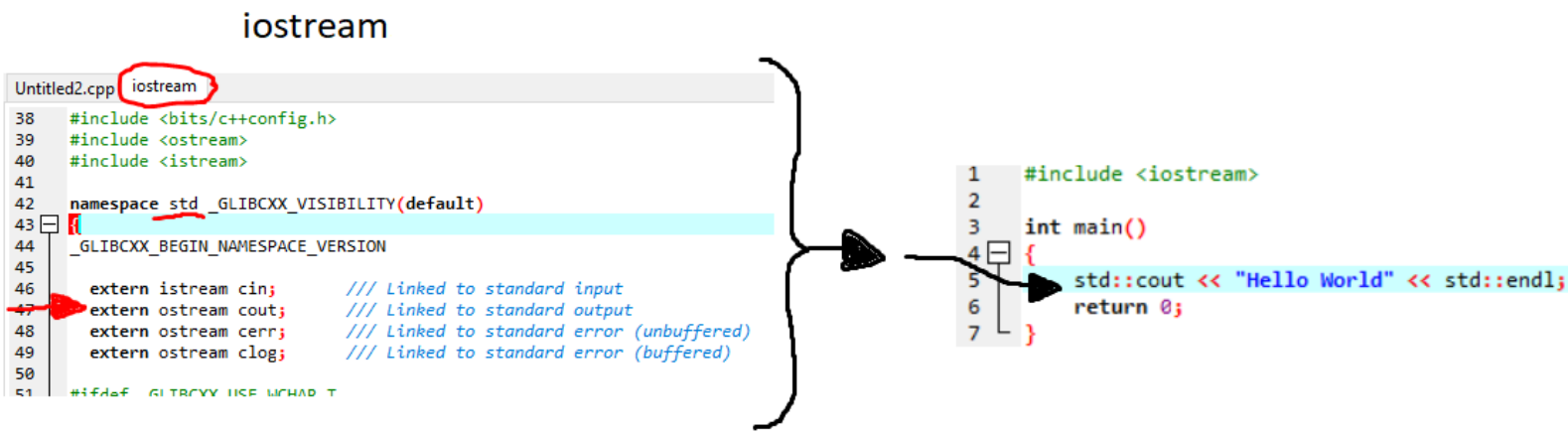
```
#include <iostream>

int main()
{
	std::cout << "Hello World" << std::endl;
	return 0;
}
```



O C++ tem uma característica para evitar conflito de nomes iguais no código. O nome disso é **Namespace.**

Nome_do_namespace::Identificador

Exemplo abaixo do namespace **std** que pertence a biblioteca **iostream**



Não use o comando using namespace std;

O namespace pode agrupar entidades como classes, objetos e funções, em um nome. Podemos acessar os elementos da namespace através do operador **::**

The format of namespaces is:

```
namespace identifier
{
entities
}
```

Where `identifier` is any valid identifier and `entities` is the set of classes, objects and functions that are included within the namespace. For example:

```
namespace myNamespace
{
  int a, b;
}
```

In this case, the variables `a` and `b` are normal variables declared within a namespace called `myNamespace`. In order to access these variables from outside the `myNamespace` namespace we have to use the scope operator `::`. For example, to access the previous variables from outside `myNamespace` we can write:

```
myNamespace::a
myNamespace::b
```

Untitled2.cpp | iostream

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

namespace first
{
  int var = 5;
}

namespace second
{
  double var = 3.1416;
}

int main () {
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\
5
3.1416

--------------------------------
Process exited after 0.115 seconds
Press any key to continue . . .
```

No exemplo acima estamos declarando o namespace no escopo do arquivo. Isso significa que todo o arquivo consegue acessar livremente (pelo menos nesse caso) os membros de **std**.

Untitled2.cpp | iostream

```
#include <bits/c++config.h>
#include <ostream>
#include <istream>

namespace std _GLIBCXX_VISIBILITY(default)
{
_GLIBCXX_BEGIN_NAMESPACE_VERSION

  extern istream cin;       /// Linked to standard input
  extern ostream cout;      /// Linked to standard output
  extern ostream cerr;      /// Linked to standard error (unbuffered)
  extern ostream clog;      /// Linked to standard error (buffered)
```

Namespace std

All the files in the C++ standard library declare all of its entities within the `std` namespace. That is why we have generally included the `using namespace std;` statement in all programs that used any entity defined in `iostream`.

Quando a função está dentro da biblioteca, precisamos incluir no código a nossa biblioteca.

STD faz parte da biblioteca **iostream**, por isso incluímos ela.

Curso de C++ - Aula 03 - Declarando variáveis

https://youtu.be/5nnyNLzBG9o

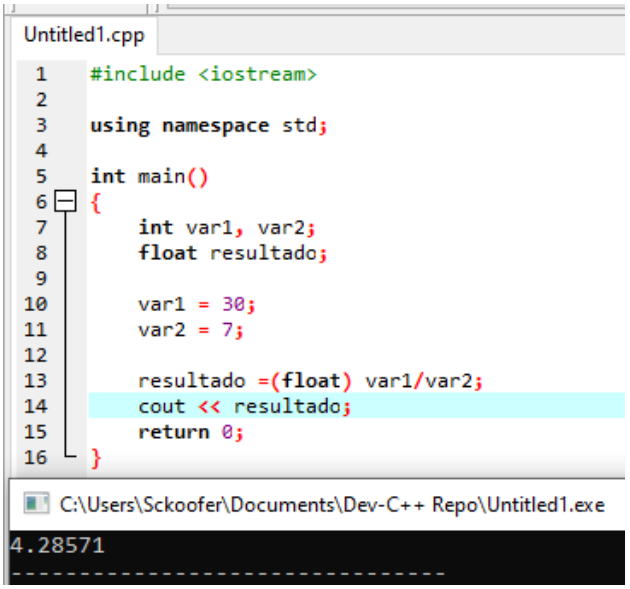
Variáveis com letras maiúsculas são tratadas como variáveis distintas das variáveis com letras minúsculas.

Curso de C++ - Aula 04 - Operadores

https://youtu.be/hQaqOV0O01A

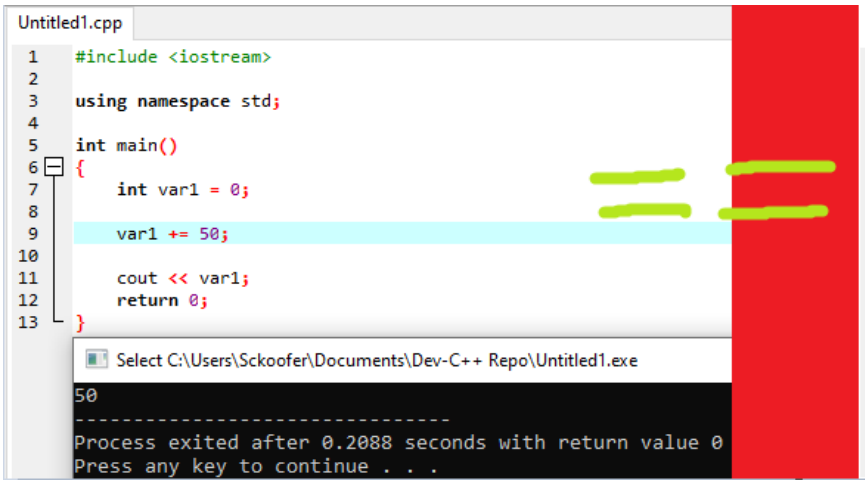
Variáveis não inicializadas são inicializadas automaticamente com lixo (valor inútil), o correto é inicializar manualmente.

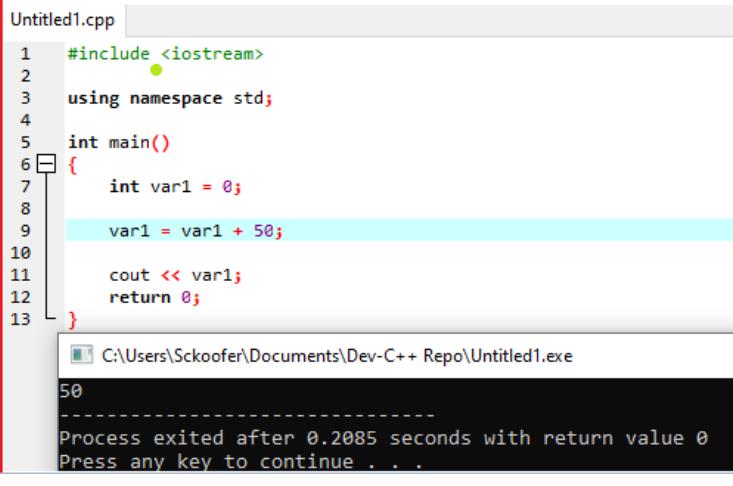
A imagem abaixo mostra como fazer a **conversão** de valores de um tipo para outro (**casting**). Caso a variável receba o valor inteiro, o resultado será truncado. Só apareceria o valor 4, se fosse maior o valor fracionado, o valor seria arredondado.



Abaixo, veja os operadores unários (agem sobre um único valor)

Operador += faz a soma da mesma forma. -= funciona da mesma forma. Multiplicação também. % é o resto da divisão.





Operador de precedência é (). Aquele que vem a ser calculado antes.

(0_0)Curso de C++ - Aula 05 - Variável Caractere char

https://youtu.be/oJc-TUrWWC4

Inverter valores:

Usa-se o valor unário negativo junto a variável (não funciona com o operador **+**)

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int var1 = -500;
    int var2 = -var1;


    cout << var2;
    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.ex
500
--------------------------------
Process exited after 0.1875 seconds with re
Press any key to continue . . .
```

Se pretende garantir que a o operador ++ faça a soma, coloque antes da variável.

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int var1 = 500;
    int var2 = var1++;

    cout << var2;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
500
--------------------------------
```

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int var1 = 500;
    int var2 = ++var1;

    cout << var2;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\U
501
--------------------------------
```

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int var1 = 500;
    int var2 = var1++;

    cout << var2;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.ex
500
```

Podemos somar valores de caracteres (veja a tabela ASCII)

[*] Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    //a == 97
    char var1 = 'a';
    //b == 98
    char var2 = 'b';

    // 97 + 98 = 195
    int soma = var1 + var2;


    cout << soma;

    return 0;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe

```
195
```

0037	0x1f	_	95	0137	0x5f	ƒ
0040	0x20	`	96	0140	0x60	á
0041	0x21	a	97	0141	0x61	í
0042	0x22	b	98	0142	0x62	ó
0043	0x23	c	99	0143	0x63	ú
0044	0x24	d	100	0144	0x64	ñ
0045	0x25	e	101	0145	0x65	Ñ

Podemos descobrir os valores de cada caractere através da conversão (veja a tabela ASCII).

ntitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{

    char var1 = 'a';
    char var2 = 'b';

    cout << (int)var1 << endl;
    cout << (int)var2 << endl;

    return 0;
}
```

CAST

C:\Users\Sckoofer\Documents\Dev-C++ Repo\U

```
97
98

--------------------------------
```

		_	95	0137	0x5f	ƒ
		`	96	0140	0x60	á
		a	97	0141	0x61	í
		b	98	0142	0x62	ó
		c	99	0143	0x63	ú
		d	100	0144	0x64	ñ
		e	101	0145	0x65	Ñ

Dica: Como realizar a demonstração de umas aspas simples na saída? Use a barra invertida.

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{

    char var1 = '\'';


    cout << var1 << endl;

    return 0;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\U

```
'

--------------------------------
Process exited after 0.09496 seconds
Press any key to continue . . .
```

Curso de C++ - Aula 06 - Declaração condicional if

https://youtu.be/p4rOoeZ0j1w

Com o IF é possível receber respostas booleanas.

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{

    int variavel = 10;

    if(!( 5 * 2 == 0))
    {
        cout << "Nao, 5 * 2 nao eh 0";
    }


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Ur
Nao, 5 * 2 nao eh 0
--------------------------------
Process exited after 0.2619 seconds
Press any key to continue . . .
```

Podemos usar variáveis inteiras para receber valores booleanos.

Untitled1.cpp

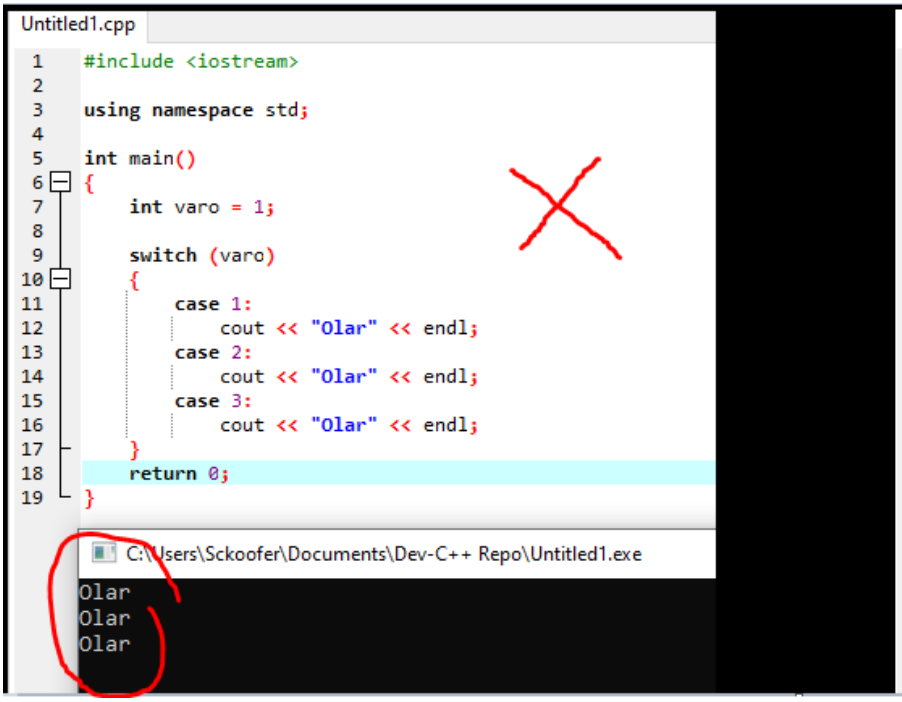
```cpp
#include <iostream>

using namespace std;

int main()
{

    int variavel = 10 < 20;

    if(variavel)
    {
        cout << "verdadeiro" << endl;
    }


    return 0;
}
```

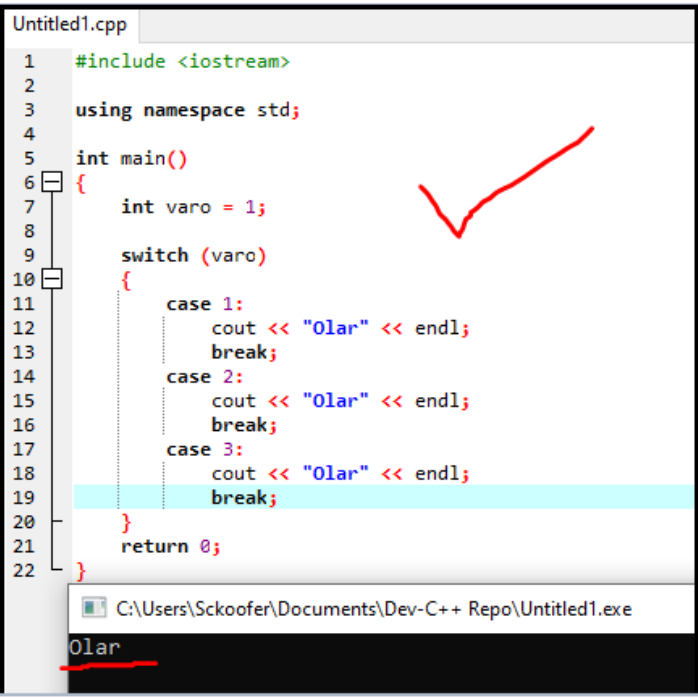
```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
verdadeiro

--------------------------------
```

Curso de C++ - Aula 07 - Comando switch

https://youtu.be/Hp7iJoxgNgM

Se não colocar o break entre os cases, o programa executa tudo de uma vez.



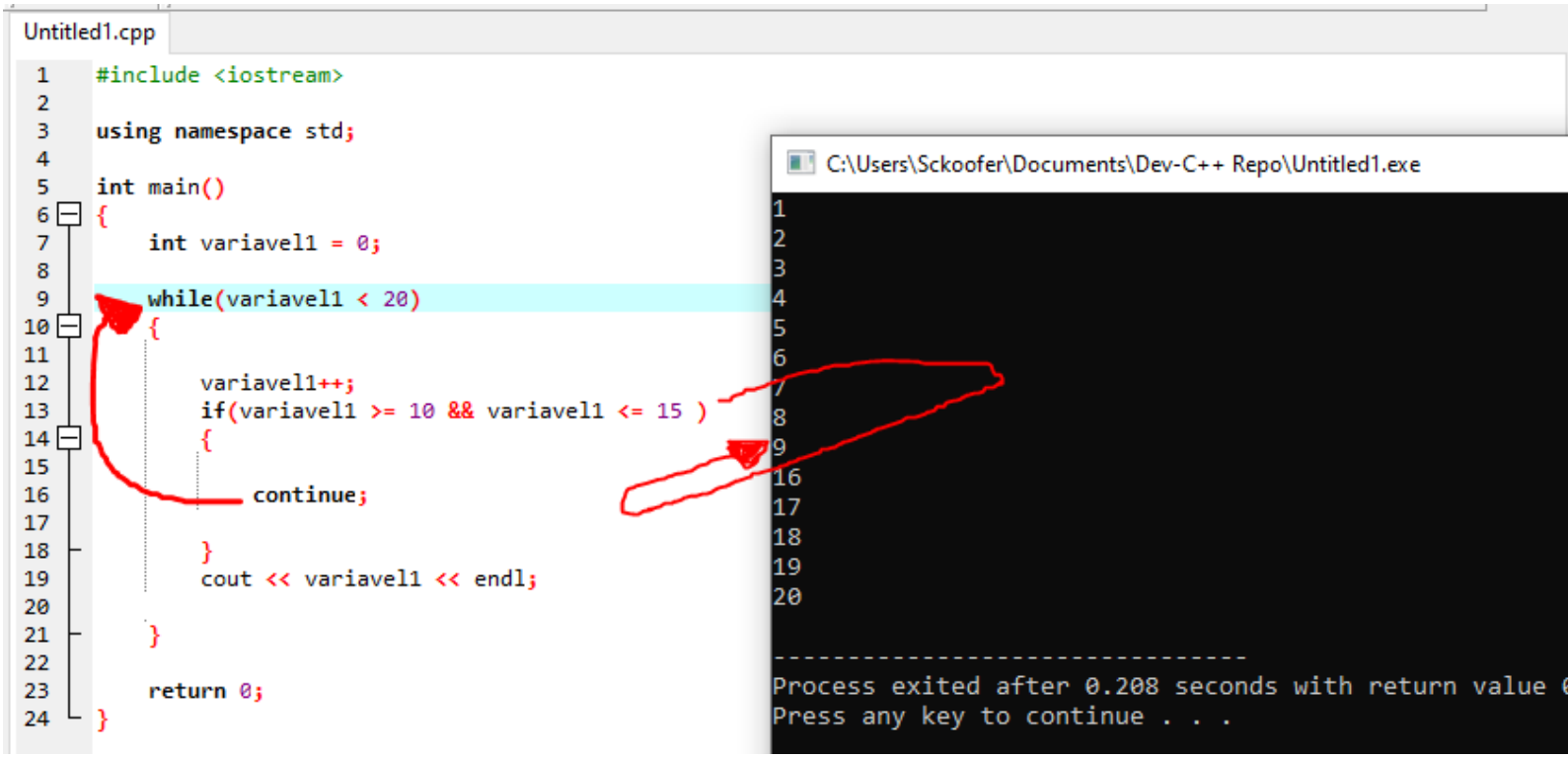


O código à esquerda pode ser útil quando precisar rodar uma sequência de código sem pausar no **BREAK.**

Curso de C++ - Aula 09 - Loop - Comando while

https://youtu.be/ll30yj7Rsy4

É possível usar a palavra-chave **break** para sair do loop.
Com a palavra-chave **continue**, a execução do loop volta **desde o começo**, dessa forma interrompendo o fluxo todo.



Curso de C++ - Aula 10 - Loop - Comando for

for(condição_1; condição_2; condição_3){ }
Podemos omitir quantas condições quisermos, por exemplo **for**(;;)
Podemos colocar quantas condições quisermos através da vírgula, por exemplo

for(condição1,condição1.2,condição1.3**;** condição2, condição2.1, condição2.3**; etc**){ }

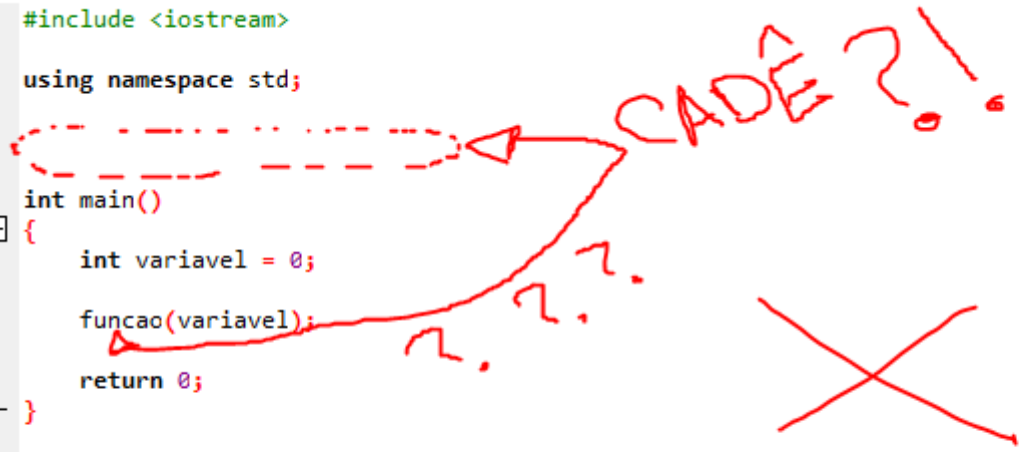
Curso de C++ - Aula 11 - Criando funções

O uso de protótipos das funções deve ser colocado antes da função principal, se não o programa vai rodar a função que nem ainda foi declarada, e vai dar problema de compilação, pois não foi declarada.

Exemplo da falta de protótipo:

Untitled1.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5
6
7   int main()
8   {
9       int variavel = 0;
10
11      funcao(variavel);
12
13      return 0;
14  }
15
16  bool funcao(int copia_variavel)
17  {
18      cout << "O valor eh: " << copia_variavel << endl;
19  }
```



Line	Col	File	Message
11	17	C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1...	[Error] 'funcao' was not declared in this scope

---

Exemplo do USO do protótipo:

Untitled1.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   bool funcao(int copia_variavel);
6
7   int main()
8   {
9       int variavel = 0;
10
11      funcao(variavel);
12
13      return 0;
14  }
15
16  bool funcao(int copia_variavel)
17  {
18      cout << "O valor eh: " << copia_variavel << endl;
19  }
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe

```
O valor eh: 0

--------------------------------
Process exited after 0.1527 seconds with return va
Press any key to continue . . .
```

Curso de C++ - Aula 12 - Sobrecarga de nomes de funções

Funções com o mesmo nome devem ter, pelo menos, **argumentos diferentes**, para não causar erro na hora da compilação. A ordem de execução vai depender **do uso do parâmetro (**ou quantidade de parâmetros**).**

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int funcao();
int funcao(int entrada);

int main()
{
    int variavel = 0;

    funcao(variavel);
    funcao();

    return 0;
}

int funcao()
{
    cout << "Esta usando a primeira funcao" << endl;
}
int funcao(int entrada)
{
    cout << "Esta usando a segunda funcao" << endl;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe

```
Esta usando a segunda funcao
Esta usando a primeira funcao

--------------------------------
Process exited after 0.2362 seconds with return val
Press any key to continue . . .
```

Curso de C++ - Aula 13 - Criando módulos

https://youtu.be/fmM2vpBSZjQ

O que é um módulo? Pense no **iostream**, é um **arquivo** contendo uma série de funções, ou uma única função.

Untitled1.cpp | modulo.h

```cpp
#include <iostream>
#include "modulo.h"

using namespace std;

int main()
{
    int variavel = 0;

    cout << funcao(variavel) << endl;

    return 0;
}
```

Untitled1.cpp | modulo.h

```cpp
int funcao(int valor_variavel)
{
    valor_variavel = 1;
    return valor_variavel;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe

```
1

--------------------------------
Process exited after 0.2837 seconds with return value 0
Press any key to continue . . .
```

Por exemplo, pense na biblioteca <math.h>, ela é um módulo.

Curso de C++ - Aula 14 - Escopo de variável

https://youtu.be/2dO5tZAmS2k

Variável Global x Variável Local
Variável Global pode ser vista e acessada por todas as funções.
Variável Local só funciona dentro de uma função.

A variável Estática não é destruída após a função rodar e finalizar, por isso ela retém o valor. Podemos utilizar a variável local para registrar quantas vezes a função é chamada.

Curso de C++ - Aula 15 - Vetores

https://youtu.be/cdEccuCz34w

O vetor começa do zero. Vetores inicializados já preenchidos não precisam necessariamente de um tamanho declarado em seu parâmetro, exemplo: vetor[] = {1,2,3,4,5,6}; É possível identificar o tamanho do vetor através do **sizeof.**

Curso de C++ - Aula 16 - Vetores de caracteres strings

https://youtu.be/NLTt1k-YXX0

Untitled1.cpp

```
1  #include <iostream>
2  #include "modulo.h"
3
4  using namespace std;
5
6  int main()
7  {
8      char nome[] = {'R','o','b','e','r','t','t','\0'};
9
10     cout << nome;
11
12     return 0;
13 }
14
15
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
Robertt
--------------------------------
Process exited after 0.1036 seconds with return value 0
Press any key to continue . . .
```

Para imprimir até um limite, ou seja, sem imprimir nada além do que queremos (se não acaba imprimindo lixo), utilizamos o caractere nulo, \0;

MAAASS…isso é só em caso de imprimir caractere por caractere, se precisar imprimir uma string inteira, é mais fácil e direto de usar aspas duplas e colocar a palavra completa inteira como valor, por exemplo:

Untitled1.cpp

```
1  #include <iostream>
2  #include "modulo.h"
3
4  using namespace std;
5
6  int main()
7  {
8      char nome[] = {"Robertt"};
9
10     cout << nome;
11
12     return 0;
13 }
14
15
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
Robertt
--------------------------------
Process exited after 0.3527 seconds with return value 0
Press any key to continue . . .
```

Dica: Como identificar caracteres com caixa alta? É através da palavra-chave **isupper.**

A biblioteca **string.h** é usada para manipulação de strings e fornece várias outras funções, por exemplo: https://www.cplusplus.com/reference/cstring/ - Tem MUITA FUNÇÃO CONHECIDA.

Curso de C++ - Aula 17 - Ponteiros

https://youtu.be/U3Y9d5Jz4LQ

Dica 1: O C++ armazena as variáveis locais em uma região chamada **stack (pilha)**.
Dica 2: O ponteiro deve ser do mesmo tipo da variável em que se baseará para a referência.
Dica 3: O C++ armazena os argumentos passados para a função a partir da direita para a esquerda.

Veja a imagem abaixo, a passagem de valores por referência. Isso não ocorre com a passagem por valor, pois é realizado somente a cópia na passagem por valor. Por referência, ocorre a alteração na variável referenciada.

Untitled1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int funcao(int * pointeiro_variavel)
5  {
6      *pointeiro_variavel = 50;
7  }
8  int main()
9  {
10     int variavel = 1;
11     int * pointeiro_variavel = &variavel;
12
13     funcao(pointeiro_variavel);
14
15     cout << variavel << endl;
16
17     return 0;
18 }
19
20
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
50

--------------------------------
Process exited after 0.9236 seconds with return value 0
Press any key to continue . . .
```

Untitled1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int funcao(int * ponteiro_variavel)
5  {
6      *ponteiro_variavel = 50;
7  }
8
9  int main()
10 {
11     int variavel = 1;
12
13     int * ponteiro = &variavel;
14
15     funcao(&variavel);
16
17     cout << *ponteiro << endl;
18
19     return 0;
20 }
21
22
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
50

--------------------------------
Process exited after 0.281 seconds with return value 0
Press any key to continue . . .
```

É a mesma coisa que passar um ponteiro

A palavra chave **new** aloca memória. A palavra chave **delete** desaloca a memória.

Untitled1.cpp

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


int main()
{

    //Aloca o espaço 10 bytes e espaço na memória.
    int * vetor = new int[10];


    //Preenche o espaço reservado número 1
    *(vetor + 0) = 10;
    //Preenche o espaço reservado número 2
    *(vetor + 1) = 20;
    //Preenche o espaço reservado número 3
    *(vetor + 2) = 30;

    //Exibe no prompt
    cout << *(vetor + 0) << endl;
    cout << *(vetor + 1) << endl;
    cout << *(vetor + 2) << endl;


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
10
20
30

--------------------------------
Process exited after 0.2787 seconds with return
Press any key to continue . . .
```

Mas não se acanhe, dá pra fazer a mesma coisa com um vetor comum. Veja a imagem abaixo.

Untitled1.cpp

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


int main()
{

    //Aloca o espaço 10 bytes e espaço na memória.

    int vetor[10];

    vetor[0] = 1;
    vetor[1] = 2;
    vetor[2] = 3;

    //Exibe no prompt
    cout << *(vetor + 0) << endl;
    cout << *(vetor + 1) << endl;
    cout << *(vetor + 2) << endl;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
1
2
3

--------------------------------
Process exited after 0.1957 seconds with return value
Press any key to continue . . .
```

Pergunta…é possível fazer a mesma coisa através do uso de funções? Sim, é, olhe a imagem abaixo (eu q fiz =))

[*] Untitled1.cpp

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

funcao(int * variavel)
{
    variavel[0] = 0;
    variavel[1] = 0;
    variavel[2] = 0;
}

int main()
{
    //Aloca o espaço 10 bytes e espaço na memória.

    int vetor[10];

    vetor[0] = 1;
    vetor[1] = 2;
    vetor[2] = 3;

    //Manda o vetor como parâmetro para a funcao
    funcao(vetor);

    //Exibe no prompt
    cout << *(vetor + 0) << endl;
    cout << *(vetor + 1) << endl;
    cout << *(vetor + 2) << endl;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
0
0
0

--------------------------------
Process exited after 0.1975 seconds with return va
Press any key to continue . . .
```

Mas é possível alterar o valor diretamente do vetor na função, sem usar os ponteiros, pois um vetor é um conjunto de endereços.

Untitled1.cpp

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

void foo(int aux[])
{
    aux[0] = 100;
}

int main()
{
    int aux[10];

    aux[0] = 10;
    aux[1] = 20;
    aux[2] = 30;

    foo(aux);

    cout << aux[0] << endl;
    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Unt
100

--------------------------------
Process exited after 0.197 seconds wi
Press any key to continue . . .
```

No final é a mesma coisa.

Lembrando que podemos usar a penúltima imagem como uma possibilidade para preencher vetores daquela forma.

Untitled1.cpp

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

int main()
{
    int vetor[10];


    printf("Digite um numero: ");
    cin >> *(vetor + 0);

    cout << "Voce digitou: ";
    cout << vetor[0] << "\n";;


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
Digite um numero: 5
Voce digitou: 5

--------------------------------
Process exited after 2.308 seconds with return value 0
Press any key to continue . . .
```

Veja a palavra-chave **delete** em ação:

Untitled1.cpp

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

int main()
{
    int * ponteiro_de_array = new int[10];

    cout << "Vetor alocado e preenchido com 0 na posicao 0" << endl;
    *(ponteiro_de_array + 0) = 10;
    cout << *(ponteiro_de_array + 0) << endl << endl;

    cout << "Vetor desalocado" << endl << endl;
    delete[] ponteiro_de_array;
    cout << ponteiro_de_array;


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
Vetor alocado e preenchido com 0 na posicao 0
10

Vetor desalocado

0xd21860
--------------------------------
Process exited after 0.4074 seconds with return v
Press any key to continue . . .
```

Depois de desalocar a memória, só restam cinzas.
Cada vez que não é liberada a memória, o programa continua a reter a mesma.

Dica: Boa prática. Depois de usar ponteiros, anule eles. Para isso faça: **ponteiro = NULL**;
Mas de qualquer forma, todo o programa retorna a memória após encerrar o programa, porém sempre faça.

O delete com colchetes é somente usado quando precisa desalocar memória em vetor. Caso contrário, apenas a palavra chave já faz o serviço. Por exemplo, se precisa desalocar memória de variável faça: **delete** nome_da_variável
Se for ponteiro, use o delete e depois aponte o ponteiro para NULL;

Curso de C++ - Aula 18 - Continuando com ponteiros

https://youtu.be/CrFq_L-UZ6g

Ponteiros também podem apontar para outras regiões mesmo após a sua declaração.

Untitled1.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int saida   = 1;
8       int saida1  = 2;
9       int saida2  = 3;
10
11      int *ponteiro = &saida;
12      cout << *ponteiro << endl;
13
14      ponteiro = &saida1;
15      cout << *ponteiro << endl;
16
17      ponteiro = &saida2;
18      cout << *ponteiro << endl;
19
20      return 0;
21  }
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++
1
2
3

--------------------------------
Process exited after 0.08775 second
Press any key to continue . . .
```

Alterando a posição para onde o ponteiro aponta. O exemplo abaixo mostra a posição sendo alterada (incrementada).

Untitled1.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int array[] = {1, 2, 3, 4, 5};
8       int * parray = &array[0];
9       int i = 10;
10
11      while(i >= 0)
12      {
13          cout << *parray << endl;
14          parray++;
15          i--;
16      }
17      return 0;
18  }
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitl
1
2
3
4
5
5
7339544
0
13243312
0
4199400

--------------------------------
Process exited after 0.2604 seconds wit
Press any key to continue . . .
```

LIXO

Em azul está o lixo, tudo isso está além do vetor que foi alocado, essa é a região da memória que não deveríamos ter posto as mãos, é lixo, é sujo, é cacáca.

Ponteiro para ponteiro (um **ponteiro** que **aponta** para **outro ponteiro**)

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int array[] = {1, 2, 3, 4, 5};

    int * ponteiro_para_array = &array[4];
    int ** ponteiro_para_ponteiro = &ponteiro_para_array;

    cout << "Para onde o ponteiro para ponteiro está apontando?" << endl;
    cout << *ponteiro_para_ponteiro << endl << endl;

    cout << "E qual é o conteúdo dessa regiao?" << endl;
    cout << **ponteiro_para_ponteiro;


    return 0;
}
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
Para onde o ponteiro para ponteiro estß apontando?
0x6ffe10

E qual 0 o conte·do dessa regiao?
5
--------------------------------
```

O conteúdo do ponteiro para ponteiro é a memória, pois é outro ponteiro que serve como seu conteúdo.
O conteúdo **apontado por ele, pode ser outro ponteiro ou pode ser o conteúdo na memória.

Mas também é possível fazer um ponteiro para ponteiro para ponteiro para ponteiro para ponteiro etc.

Por exemplo:

int **********ponteiro

Curso de C++ - Aula 19 - Ponteiros novamente

https://youtu.be/PGG6jr0uMyQ

```cpp
#include <iostream>

using namespace std;

int my_strlen(char * str)
{
    int tam = 0;

    while(*str != '\0')
    {
        str++;
        tam++;
    }
    return tam;
}

char * my_strcat(char * dest, char * orig)
{
    char * resultado;
    int tam_dest = my_strlen(dest);
    int tam_orig = my_strlen(orig);

    resultado = new char[tam_dest + tam_orig];
    char *p_str = resultado;
    while(*dest != '\0')
    {
        *p_str = *dest;
        p_str++;
        dest++;
    }
    while(*orig != '\0')
    {
        *p_str = *orig;
        p_str++;
        orig++;
    }
    *p_str = '\0';
    return resultado;
}

int main(int argc, char *argv[])
{
    char * nome1 = new char[100];
    char * nome2 = new char[100];
    int vair1;

    cout << "Digite o primeiro nome: ";
    cin >> nome1;
    cout << "Digite o segundo nome: ";
    cin >> nome2;
    vair1 = 6;
    cout << "Resultado: " << my_strcat(nome1, nome2) << "\n";
    return 0;
}
```

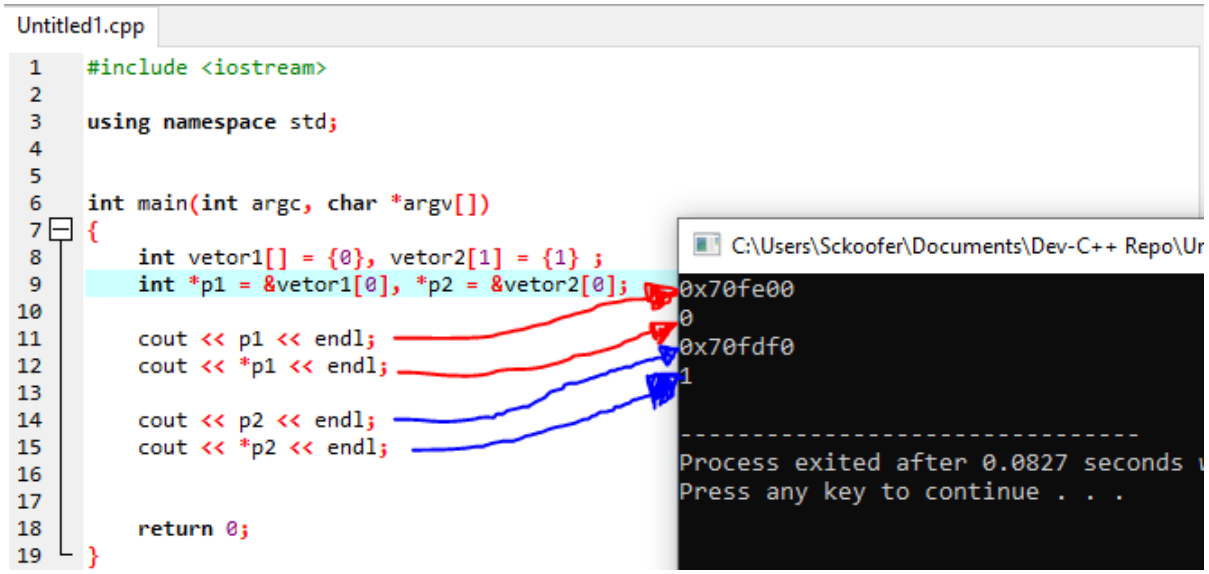
```
#include <iostream>

using namespace std;

int my_strlen(char * str)
{
  int tam = 0;
  while(*str != '\0')
  {
        str++;
        tam++;
  }
  return tam;
}
char * my_strcat(char * dest, char * orig)
{
  char * resultado;
  int tam_dest = my_strlen(dest);
  int tam_orig = my_strlen(orig);

  resultado = new char[tam_dest + tam_orig];
  char *p_str = resultado;
  while(*dest != '\0')
  {
        *p_str = *dest;
        p_str++;
        dest++;
  }
  while(*orig != '\0')
  {
        *p_str = *orig;
        p_str++;
        orig++;
  }
  *p_str = '\0';
  return resultado;
}

int main(int argc, char *argv[])
{
  char * nome1 = new char[100];
  char * nome2 = new char[100];

  cout << "Digite o primeiro nome: ";
  cin >> nome1;
  cout << "Digite o segundo nome: ";
  cin >> nome2;

  cout << "Resultado: " << my_strcat(nome1, nome2) << "\n";
  return 0;
}
```

Dica: é possível realizar a troca de valores entre ponteiros através do *ponteiro = *ponteiro.

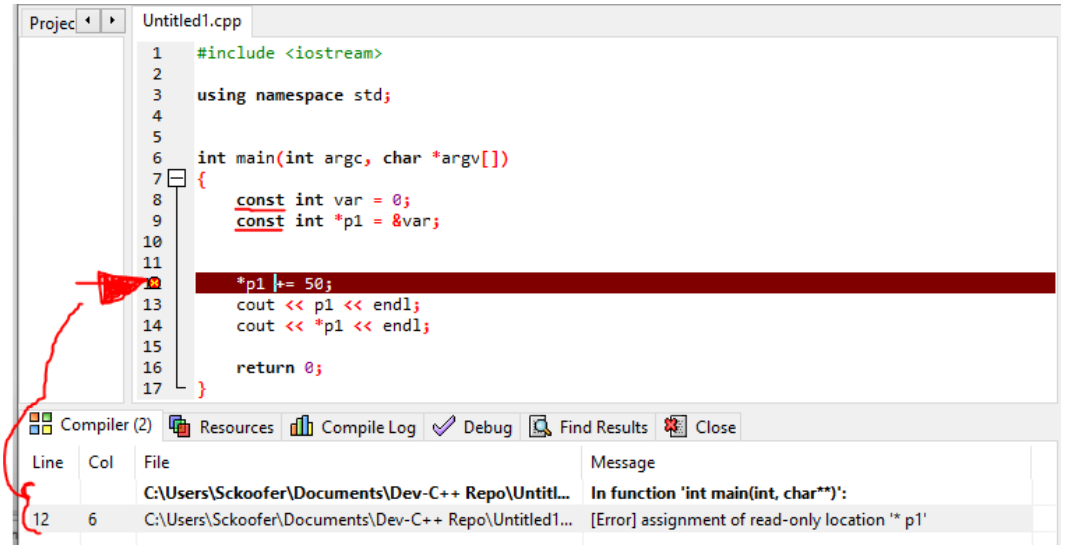
Curso de C++ - Aula 20 - Constantes

https://youtu.be/gvKVnC2wWZA

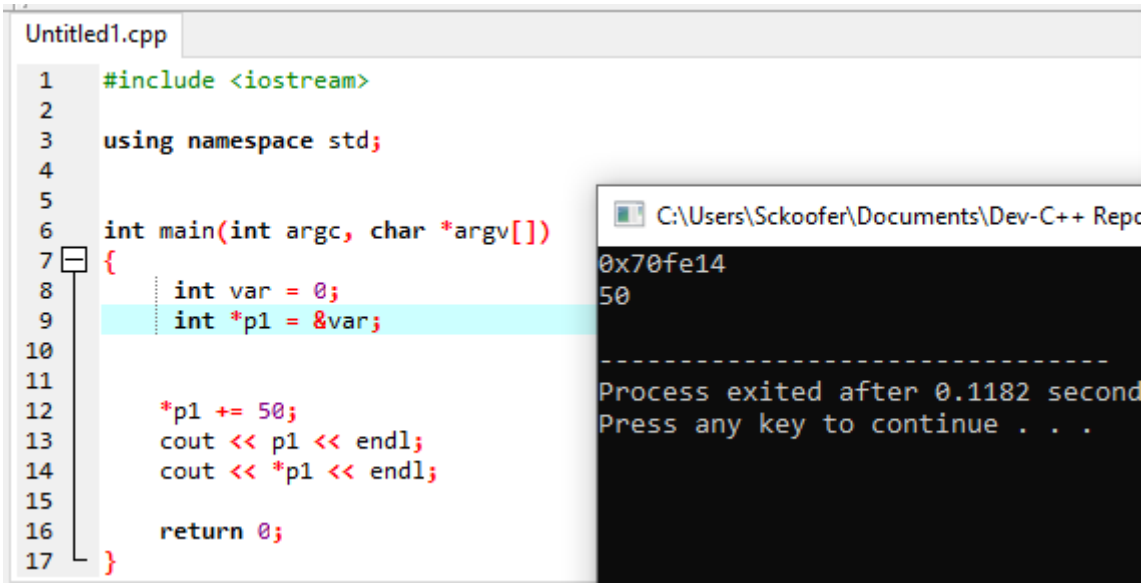
A atribuição de valores de vetores para ponteiros pode ser feita na inicialização do ponteiro, exemplo:
int variavel[] = {1, 2, 3,};



Float é para números quebrados
Double é para precisão dupla.
const é para quando nunca for alterada.



Deu ruim na imagem acima, mas por quê? Porque é tudo constante e você tava tentando alterar valor inalterável (const).



A imagem acima deu boa, porque não é constante.

A posição do * na declaração do ponteiro constante altera seu propósito.

int const * ponteiro
const int * ponteiro
(mesma coisa)

O que não dá pra fazer nesse caso é alterar o conteúdo referenciado pela variável que está sendo apontada pelo ponteiro.

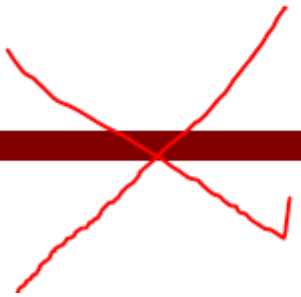
Untitled1.cpp

```cpp
#include <iostream>

using namespace std;


int main(int argc, char *argv[])
{
    //É possível modificar para onde apontam
    //Mas não é possível modificar o valor da variável apontada

    const int variavel = 50;
    const int * p1 = &variavel;
    int const * p2 = &variavel;;

    *p1 = 20;


    return 0;
}
```

File	Message
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitl...	**In function 'int main(int, char**)':**
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1...	[Error] assignment of read-only location '*

Untitled1.cpp

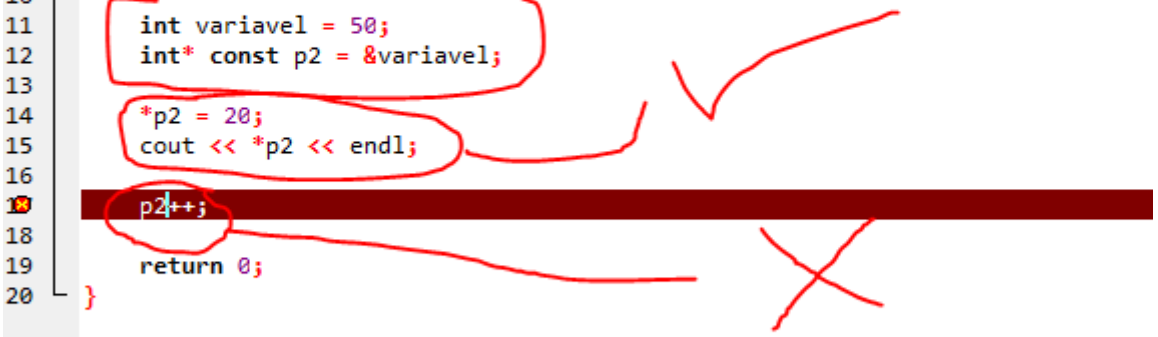
```cpp
#include <iostream>

using namespace std;


int main(int argc, char *argv[])
{
    //É possível modificar para onde apontam
    //Mas não é possível modificar o valor da variável apontada

    const int variavel = 50;
    const int * p1 = &variavel;
    int const * p2 = &variavel;;

    *p2 = 20;


    return 0;
}
```



File	Message
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitl...	**In function 'int main(int, char**)':**
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1...	[Error] assignment of read-only location '*

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;


int main(int argc, char *argv[])
{
    //É possível modificar para onde apontam
    //Mas não é possível modificar o valor da variável apontada

    int variavel = 50;
    int* const p2 = &variavel;

    *p2 = 20;
    cout << *p2 << endl;

    p2++;

    return 0;
}
```



2) Resources | Compile Log | Debug | Find Results | Close

File	Message
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitl...	**In function 'int main(int, char**)':**
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1...	[Error] increment of read-only variable 'p2'

https://youtu.be/gvKVnC2wWZA?t=814

Pergunta: Ok, é possível transformar o valor da variável apontada, em um constante: **int const*/const int*.**
Ok, é possível transformar o endereço da variável apontada, em uma constante: **int* const.**

Mas como fazer para transformar o valor da variável apontada e o endereço da variável apontada, em uma constante?
Você faz assim ó: **const tipo_do_ponteiro* const**

```cpp
    int variavel = 50;

    const int* const p1 = &variavel;

    p1++;

    *p1 = 1;
```

```cpp
    int variavel = 50;

    const int* const p1 = &variavel;

    //p1++;

    *p1 = 1;
```

Line	Col	File	Message
		C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitl...	**In function 'int main(int, char**)':**
17	6	C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1...	[Error] assignment of read-only location '*(const int*)p1'

Recapitulando: O **vetor[tamanho] = new tipo_do_vetor[tamanho]** está alocando espaço na memória para 4 tipos iguais.
Já um ponteiro, ele não aloca espaço para valores(objetos), apenas para endereços.
Não é possível alterar o endereço de um vetor, é como se fosse **int* const**

Curso de C++ - Aula 21 - Argumentos da função main

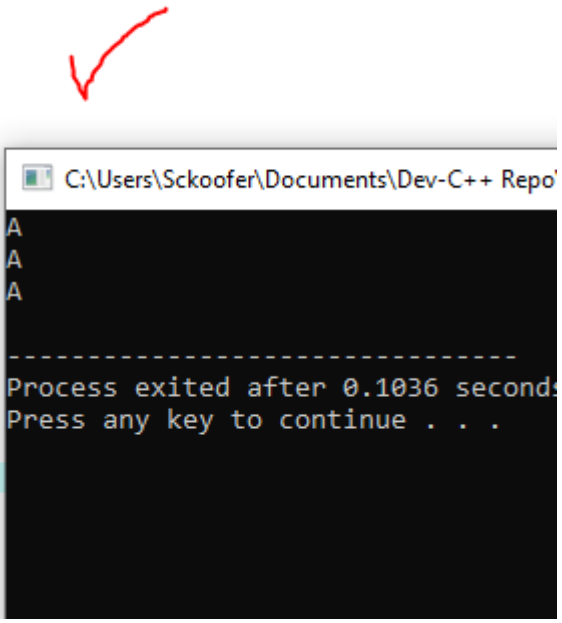
https://youtu.be/9VbAOtmK1rg
https://youtu.be/9VbAOtmK1rg?t=67

Vamos falar um pouco mais sobre **NEW**. Detalhe: array == vetor, ok!
No exemplo abaixo, a palavra chave **new** retorna um ponteiro para o array recém criado (imagem abaixo).

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

int main(int argc, char *argv[])
{
    char* array = new char[100];
    return 0;
}
```

Untitled1.cpp

```cpp
#include <iostream>

using namespace std;


int main(int argc, char *argv[])
{
    char * array = new char[5];

    cout << (array[0] = 'A') << endl;
    cout << (array[1] = 'A') << endl;
    cout << (array[2] = 'A') << endl;

    delete array;

    return 0;
}
```



Quando se usa um **new,** a pilha na memória não será alocada até o momento de sua execução. Se isso é verdade, então o tamanho do array (imagem acima) não será limitado por constantes.

Vamos tratar agora sobre é o: **int argc**, **char *argv[]**

Servem para passarmos argumentos via prompt de comando, até mesmo arquivo txt

Untitled1.cpp

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

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

    cout << "*argv[]: " << argv[0] << endl;

    return 0;
}
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
argc: 1

*argv[]: C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe

--------------------------------
Process exited after 0.09773 seconds with return value 0
Press any key to continue . . .
```

argc (*argument count*) and argv (*argument vector*)

argc: 1 - É a quantidade de argumentos passados pela linha de comando (prompt de comando). Por ele podemos passar um arquivo txt.

***argv[0]** é o caminho do programa, é onde ele está localizado.

Dúvida sobre **argc**, porque o argc está com o valor 1? Esse é o nome do programa, esse é sempre o primeiro argumento.
Dúvida sobre ***argv[0],** é um array de ponteiros para strings de caracteres, então o primeiro parâmetros é sempre o nome do programa, por exemplo:  e daí o caminho completo:

Untitled1.cpp

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

int main(int argc, char *argv[])
{
    int var1 = 0, var2 = 0, var3 = 0;


    //Veja o array de ponteiros abaixo:
    int* array[] = {&var1, &var2, &var3};


    cout << *array[0] << endl;
    cout << *array[1] << endl;
    cout << *array[2] << endl;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\Untitled1.exe
0
0
0


--------------------------------
Process exited after 0.08538 seconds with return value 0
Press any key to continue . . .
```

Veja, na imagem acima, um exemplo de um array de ponteiros/vetor de ponteiros.

Agora veja no exemplo prático abaixo, o uso do **argc** e **argv**

executar.cpp

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

int main(int argc, char *argv[])
{
    cout << "Quantidade de argumentos (argc): " << argc << endl << endl;
    cout << "Argumentos passados (argv): " << endl;

    for(int i = 0; i < argc; i++)
        cout << argv[i] << endl;

    cout << endl << endl;
    system("pause");
    return 0;
}
```

```
Command Prompt

C:\Users\Sckoofer\Documents\Dev-C++ Repo>start executar.exe

C:\Users\Sckoofer\Documents\Dev-C++ Repo>start executar.exe carne carninha carnelinha batata

C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Quantidade de argumentos (argc): 5

Argumentos passados (argv):
executar.exe
carne
carninha
carnelinha
batata


Press any key to continue . . .
```

Curso de C++ - Aula 22 - Agrupando dados com structs

https://youtu.be/YeRebCWzO6E
https://www.youtube.com/watch?v=YeRebCWzO6E&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=22&ab_channel=MarcosCastro

executar.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   typedef struct pessoa{
6       char nome[5] = {'A','n','n','e'};
7       int idade;
8   } t_pessoa;
9
10  int main(int argc, char *argv[])
11  {
12      t_pessoa variavel;
13
14      cout << "O nome da pessoa eh: " << variavel.nome << endl;
15      cout << "A idade da pessoa eh: " << variavel.idade << endl;
16
17      return 0;
18  }
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
O nome da pessoa eh: Anne
A idade da pessoa eh: 54

--------------------------------
Process exited after 0.08352 seconds with return value 0
Press any key to continue . . .
```

typedef define um apelido para a estrutura, no final da estrutura (código acima) é possível ver na linha 8 o apelido, que é t_pessoa e representa a estrutura, ou seja, acabamos de criar um **tipo**.

executar.cpp

```
1   #include <iostream>
2   #include <string.h>
3
4   using namespace std;
5
6   typedef struct pessoa{
7       char nome[100];
8       int idade;
9   } t_pessoa;
10
11  int main(int argc, char *argv[])
12  {
13      t_pessoa p[10];
14
15      p[2].idade;
16
17      strcpy(p[0].nome, "marcos");
18      p[0].idade = 26;
19
20      strcpy(p[1].nome, "joao");
21      p[1].idade = 26;
22
23      cout << "Nome: " << p[0].nome << endl;
24      cout << "Idade: " << p[0].idade << endl << endl;
25
26      cout << "Nome: " << p[1].nome << endl;
27      cout << "Idade: " << p[1].idade << endl << endl;
28
29      return 0;
30  }
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Nome: marcos
Idade: 26

Nome: joao
Idade: 26


--------------------------------
Process exited after 0.2337 seconds with return value 0
Press any key to continue . . .
```

Através do uso da **strcpy** é possível atribuir strings para o vetor da **struct.**
Porém, isso só é possível através da biblioteca **string.h**

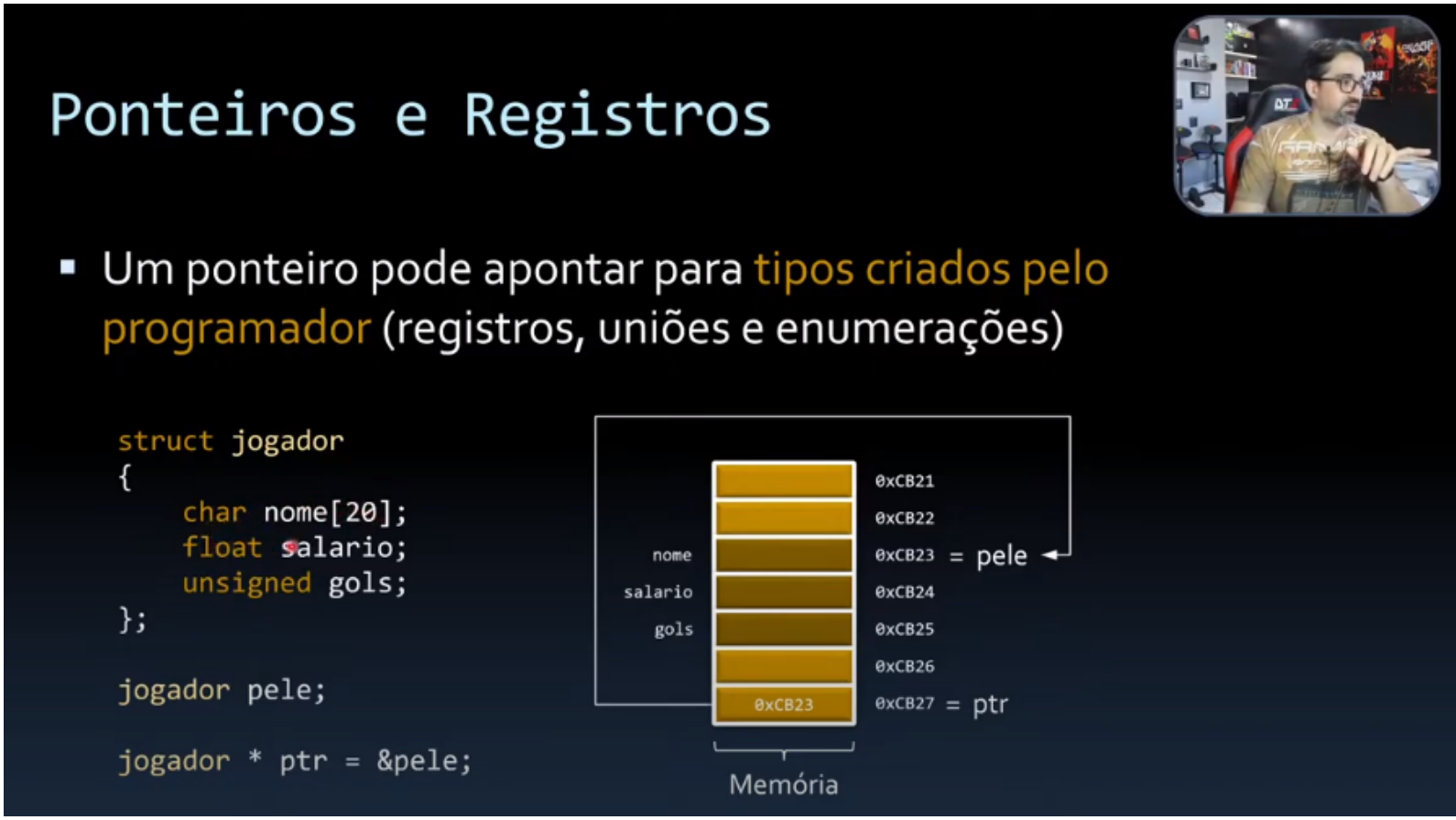
Dúvida: Todas as informações que estão sendo usadas a partir da **struct**, dá pra acessar através de ponteiros? Sim, é possível, porém o conteúdo mostrado na aula a qual se baseia esse material está errada (antigamente poderia estar certa, hoje em dia deu **tilt**).

executar.cpp

```cpp
#include <iostream>

using namespace std;
struct pessoa{

    int idade = 5;

};

int main()
{
    pessoa *ponteiro;

    ponteiro->idade = 10;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe

--------------------------------
Process exited after 0.6171 seconds with return value 3221225477
Press any key to continue . . .
```

Mas a aula do Judson está correta:



E também descrito em:
https://www.cplusplus.com/doc/tutorial/structures/

Pointers to structures

Like any other type, structures can be pointed to by its own type of pointers:

```
struct movies_t {
  string title;
  int year;
};

movies_t amovie;
movies_t * pmovie;
```

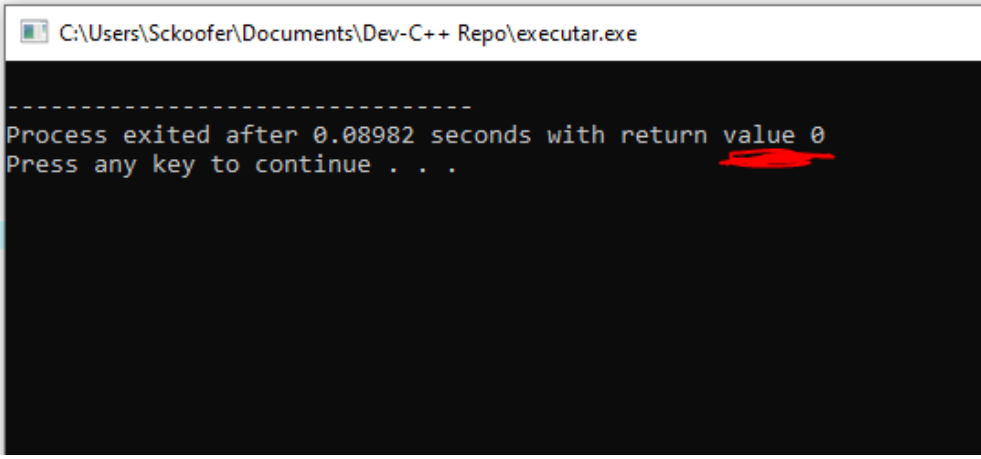
Here `amovie` is an object of structure type `movies_t`, and `pmovie` is a pointer to point to objects of structure type `movies_t`. Therefore, the following code would also be valid:

```
pmovie = &amovie;
```

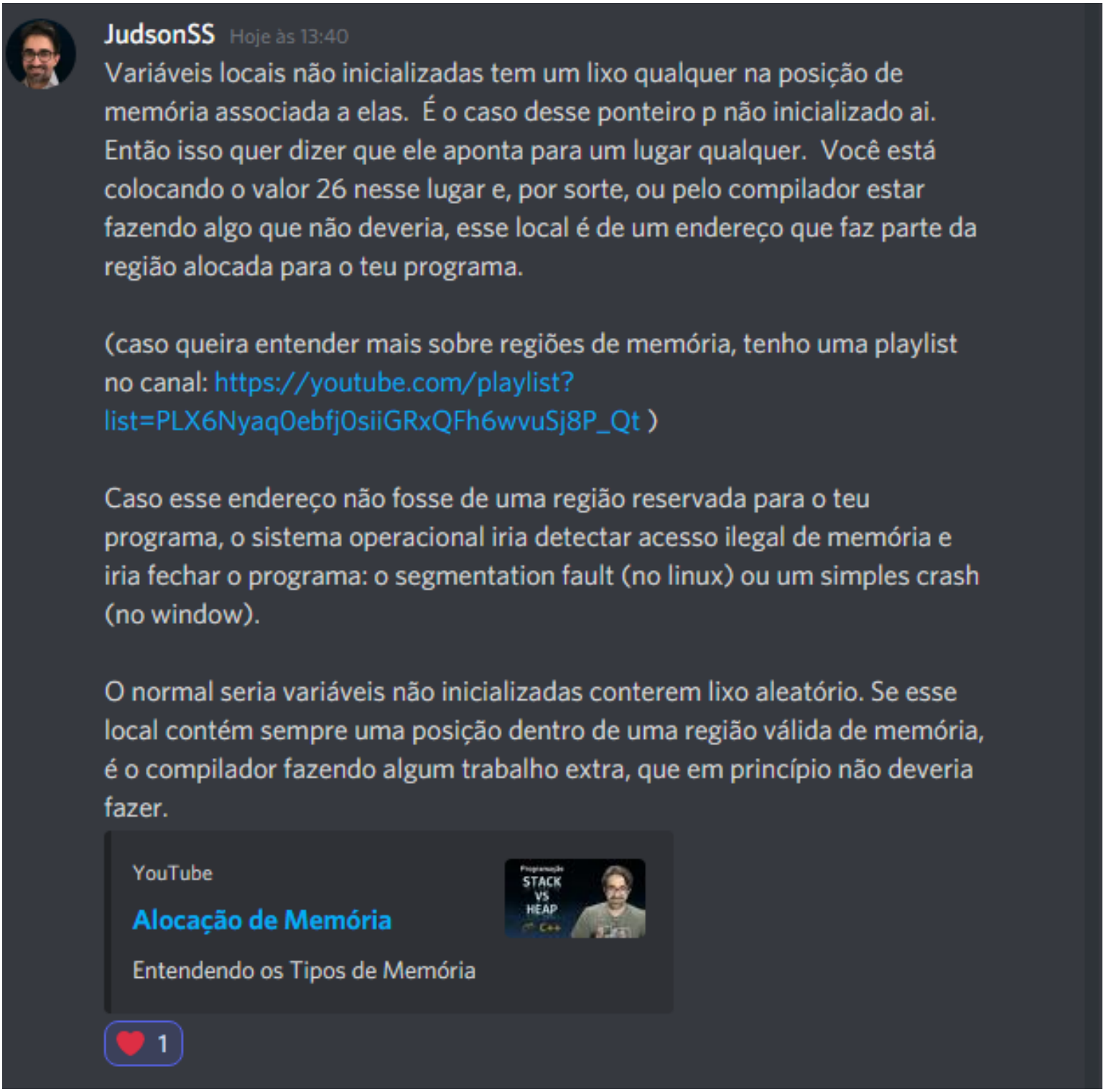
The value of the pointer `pmovie` would be assigned the address of object `amovie`.

executar.cpp

```
#include <iostream>


using namespace std;

struct pessoa{

    int idade = 30;

};

int main()
{
    pessoa variavel;
    pessoa *ponteiro = &variavel;

    ponteiro->idade = 35;


    return 0;
}
```

https://youtu.be/napD2SXz4Fo?t=4898



Informações sobre non-static members (membros não estáticos)
https://en.cppreference.com/w/cpp/language/data_members

Dúvida, é possível criar vetores a partir do tipo da struct? Yes, dá yes.

executar.cpp

```cpp
#include <iostream>


using namespace std;

struct pessoa{

    int idade = 5;

};

int main()
{
    //vetores do tipo da struct podem
    //ser construídos.

    pessoa variavel[100];
    pessoa *ponteiro = &variavel[0];


    return 0;
}
```

Curso de C++ - Aula 23 - Structs novamente

https://youtu.be/MYe6vwhze9Q
https://www.youtube.com/watch?v=MYe6vwhze9Q&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=23&ab_channel=MarcosCastro

The members and base classes of a `struct` are `public` by default, while in `class`, they default to `private`.

https://isocpp.org/wiki/faq/classes-and-objects#struct-vs-class

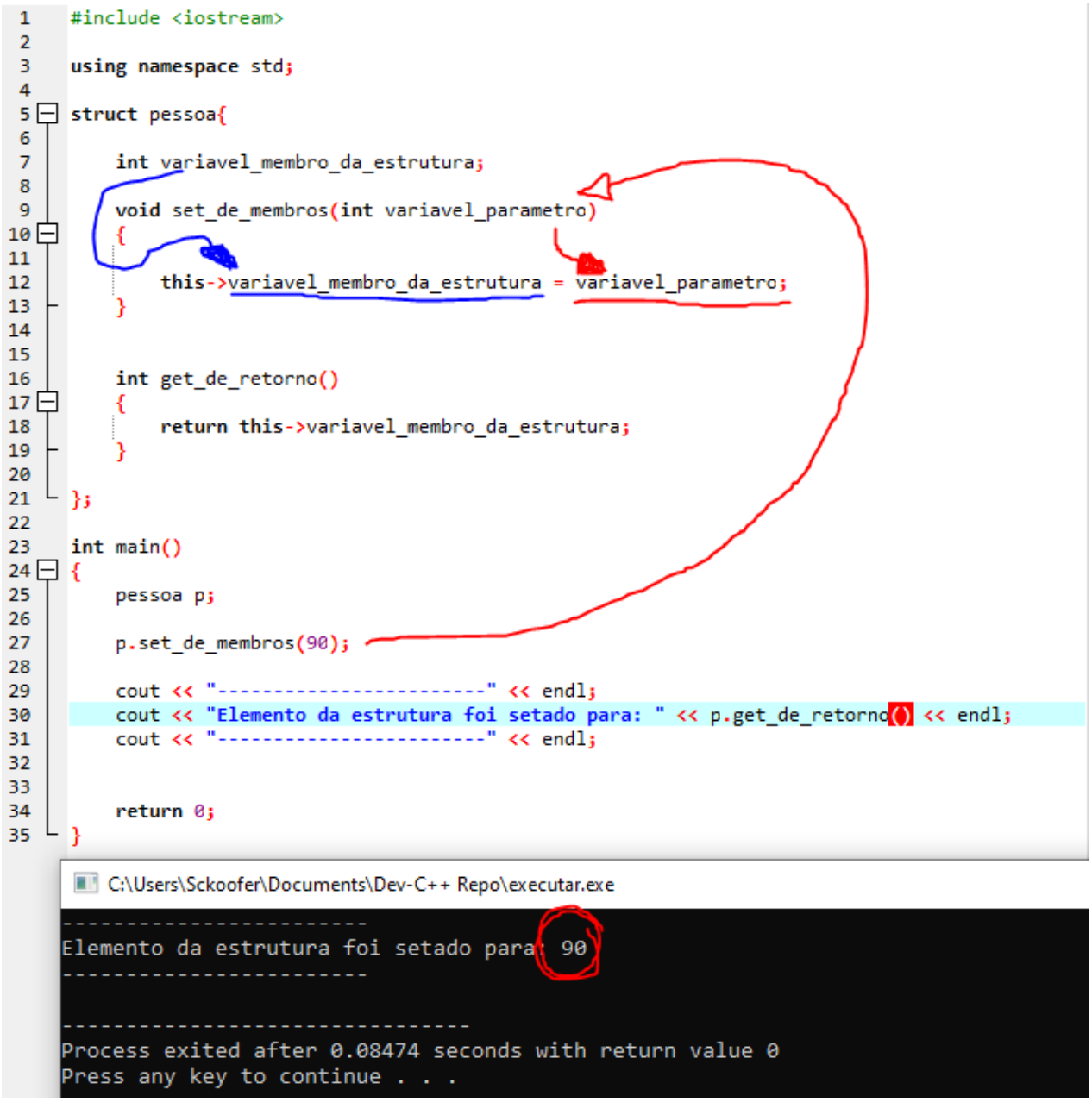
Membros da struct são públicos (é o padrão), enquanto que as classes são privadas (padrão).

As estruturas em C++ agem de forma diferente em comparação com o C.
Dentro de structs, podem existir métodos (funções).

executar.cpp

```cpp
#include <iostream>

using namespace std;

struct pessoa{

    int idade = 5;

    void setIdade(int idade)
    {
        this->idade = idade;
    }
    int getIdade()
    {
        return this->idade;
    }

};

int main()
{
    pessoa p;

    p.setIdade(90);

    cout << p.getIdade() << endl;

    return 0;
}
```

A palavra chave **this** altera o componente da estrutura, não a variável do parâmetro. Às vezes esse método acima pode confundir, pra ficar mais fácil, veja outro exemplo abaixo.




```
#include <iostream>

using namespace std;

struct pessoa{
	int variavel_membro_da_estrutura;
	void set_de_membros(int variavel_parametro)
	{
		this->variavel_membro_da_estrutura = variavel_parametro;
	}
	int get_de_retorno()
	{
		return this->variavel_membro_da_estrutura;
	}
};
int main()
{
	pessoa p;
	p.set_de_membros(90);
	cout << "Elemento da estrutura foi setado para: " << p.get_de_retorno() << endl;
	return 0;
}
```

Note que o **this->** acessa o atributo da estrutura, e não o parâmetro.

E como ficam os construtores dentro das structs? Lembrando que o nome da struct ou classe deve **começar com letra maiúscula**, **sempre**, deixa mais fácil de entender.

Veja na imagem abaixo, o uso de sobrecarga de métodos usando **this** dentro da estrutura.

```cpp
#include <iostream>

using namespace std;

struct Estrutura
{
    int valor;
    int valor2;

    //Construtor
    Estrutura(int inicializador)
    {
        this->valor = inicializador;
    }
    //Sobrecarga de construtor
    Estrutura(int inicializador, int inicializador2)
    {
        this->valor = inicializador;
        this->valor2 = inicializador2;
    }

    int retorno()
    {

        return valor;
        return valor2;
    }
};
int main()
{
    Estrutura estrut(10);
    cout << estrut.valor << endl;

    Estrutura objeto(20, 30);
    cout << objeto.valor << endl;
    cout << objeto.valor2 << endl;

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
10
20
30

--------------------------------
Process exited after 0.08969 seconds with retu
Press any key to continue . . .
```

A de cima não está usando o método **int retorno()**

```cpp
#include <iostream>

using namespace std;

struct Estrutura
{
    int valor;
    int valor2;

    //Construtor
    Estrutura(int inicializador)
    {
        this->valor = inicializador;
    }
    //Sobrecarga de construtor
    Estrutura(int inicializador, int inicializador2)
    {
        this->valor = inicializador;
        this->valor2 = inicializador2;
    }

    int retorno()
    {
        return valor + valor2;
    }
};
int main()
{
    Estrutura estrut(10);
    cout << estrut.valor << endl << endl;

    Estrutura objeto(20,30);
    cout << objeto.valor << endl << endl;
    cout << objeto.valor2 << endl << endl;

    cout << objeto.retorno() << endl << endl;

    return 0;
}
```

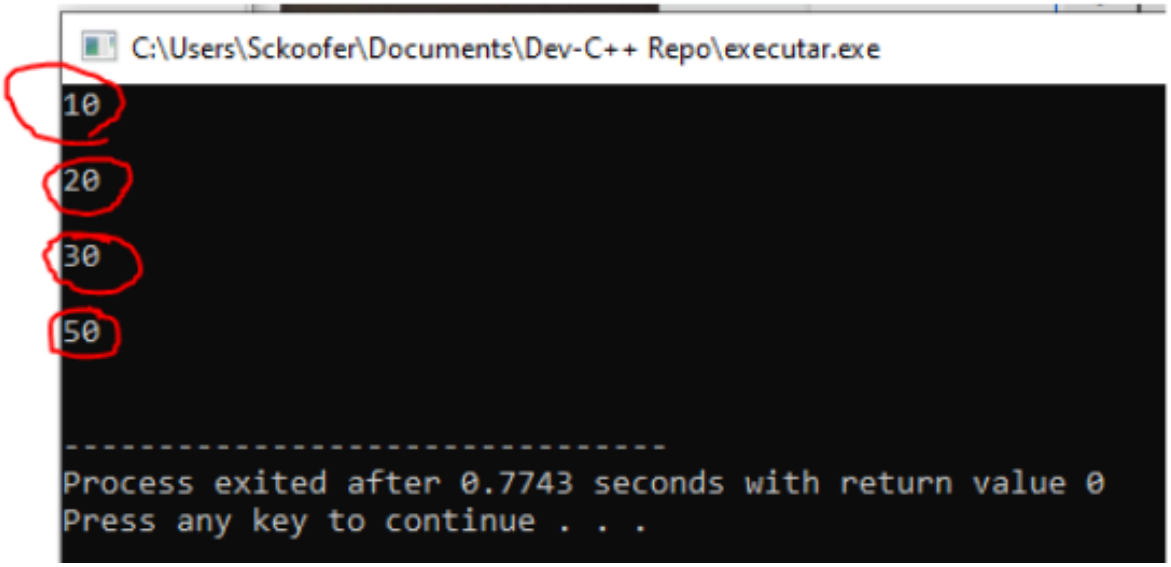
```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\
10

20

30

50


--------------------------------
Process exited after 0.09134 second
Press any key to continue . . .
```

Chegou a hora. Você sabia que é possível separar o construtor para fora da struct? Já vimos algo parecido com classes, mas também dá pra fazer com struct.

[*] executar.cpp

```cpp
#include <iostream>

using namespace std;

struct Estrutura
{
    int valor;
    int valor2;

    //Construtor
    Estrutura(int inicializador);

    //Sobrecarga de construtor
    Estrutura(int inicializador, int inicializador2);

    int retorno()
    {
        return valor + valor2;
    }
};

Estrutura::Estrutura(int inicializador)
{
    this->valor = inicializador;
}
```

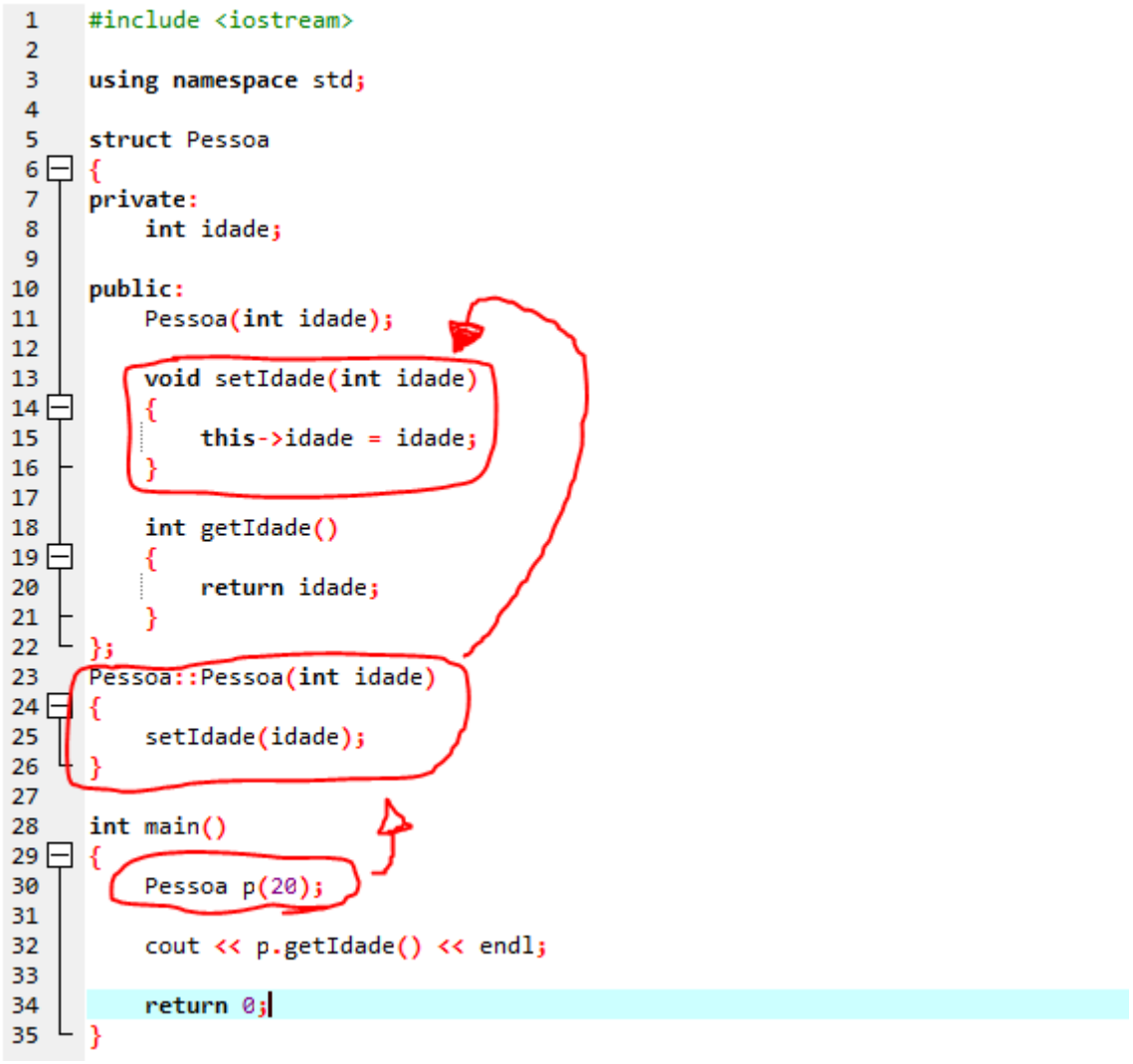
```cpp
Estrutura::Estrutura(int inicializador, int inicializador2)
{
    this->valor = inicializador;
    this->valor2 = inicializador2;
}

int main()
{
    Estrutura estrut(10);
    cout << estrut.valor << endl << endl;

    Estrutura objeto(20,30);
    cout << objeto.valor << endl << endl;
    cout << objeto.valor2 << endl << endl;

    cout << objeto.retorno() << endl << endl;

    return 0;
}
```



Como você percebeu, conseguimos transferir o construtor para fora da **struct**, porém ainda mantendo o protótipo dentro dela. O operador que utilizamos para isso é o **::** (operador de escopo).

```cpp
Estrutura::Estrutura(int inicializador)
{
    this->valor = inicializador;
}
Estrutura::Estrutura(int inicializador, int inicializador2)
{
    this->valor = inicializador;
    this->valor2 = inicializador2;
}
```

Na imagem abaixo temos a primeira demonstração do uso de **private** e **public**
Tudo isso nós aplicamos nas classes, mas também podemos usar da **mesma forma em structs**



```cpp
#include <iostream>

using namespace std;

struct Pessoa
{
private:
    int idade;

public:
    Pessoa(int idade);

    void setIdade(int idade)
    {
        this->idade = idade;
    }

    int getIdade()
    {
        return idade;
    }
};
Pessoa::Pessoa(int idade)
{
    setIdade(idade);
}

int main()
{
    Pessoa p(20);

    cout << p.getIdade() << endl;

    return 0;
}
```

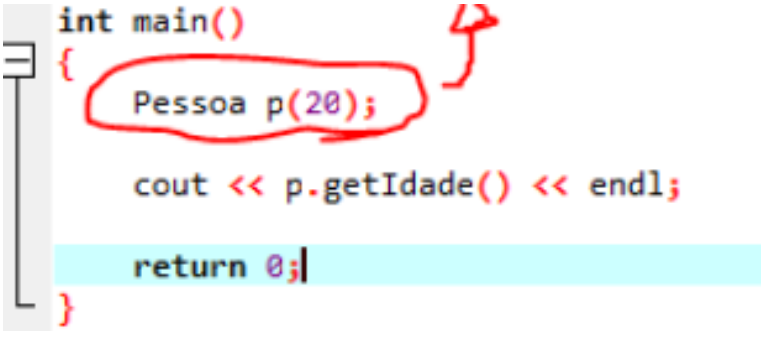
Nota 1: Na linha 15 temos a variável idade (**privada**) sendo preenchida com o valor do parâmetro (linha 30) da **main**.
Nota 2: Linha 18, apenas retorna.

O que fizemos na imagem acima é chamado de **encapsulamento**.
Public está disponível a todo método que tenha acesso ao objeto.
Private é o mais restrito, somente acessado pelos métodos dentro da classe e métodos explicitamente declarados.
Protected somente acessados por métodos da mesma classe, classes herdadas e métodos explicitamente declarados.

Curso de C++ - Aula 24 - Criando classes

https://youtu.be/IN8I5QMocNo
https://www.youtube.com/watch?v=IN8I5QMocNo&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=24&ab_channel=MarcosCastro

É uma convenção usarmos letras maiúsculas no nome das classes.
Um objeto é a instância de uma classe, assim como quando criamos uma instância de uma struct:
nome_da_struct ou **nome_da_classe** **nome_da_instância**



```cpp
int main()
{
    Pessoa p(20);

    cout << p.getIdade() << endl;

    return 0;
}
```

executar.cpp

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Pessoa
{
    public:
        char nome[100];
        char cpf[20];
        int idade;
};

int main()
{
    Pessoa p1;

    strcpy(p1.nome, "joao");
    strcpy(p1.cpf, "00000000000");
    p1.idade = 30;

    cout << "Nome da pessoa: " << p1.nome << endl;
    cout << "CPF da pessoa: " << p1.cpf << endl;
    cout << "Idade da pessoa: " << p1.idade << endl;


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Nome da pessoa: joao
CPF da pessoa: 00000000000
Idade da pessoa: 30

--------------------------------
Process exited after 0.1103 seconds with return value 0
Press any key to continue . . .
```

Mas dá sim para inicializar enquanto cria uma instância da classe ao mesmo tempo.

executar.cpp

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Pessoa
{
    public:
        char nome[100];
        char cpf[20];
        int idade;
};

int main()
{
    Pessoa p1 = {"Joao", "111111111", 30};

    cout << "Nome da pessoa: " << p1.nome << endl;
    cout << "CPF da pessoa: " << p1.cpf << endl;
    cout << "Idade da pessoa: " << p1.idade << endl;


    return 0;
}
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Nome da pessoa: Joao
CPF da pessoa: 111111111
Idade da pessoa: 30

--------------------------------
Process exited after 0.08804 seconds with return value 0
Press any key to continue . . .
```

E também dá para instanciar vetores de objetos (exemplo abaixo)

[*] executar.cpp

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Pessoa
{
    public:
        char nome[100];
        char cpf[20];
        int idade;
};

int main()
{
    int contador = 0;
    Pessoa pessoas[3] = {
                        {"Joao", "111111111", 30},
                        {"Maria", "111111111", 20},
                        {"Maria", "111111111", 25},
                       };

    while (contador <3)
    {
        cout << "Nome da pessoa: " << pessoas[contador].nome << endl;
        cout << "CPF da pessoa: " << pessoas[contador].cpf << endl;
        cout << "Idade da pessoa: " << pessoas[contador].idade << endl;
        contador++;
        cout << "\n\n";
    }

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\execu
Nome da pessoa: Joao
CPF da pessoa: 111111111
Idade da pessoa: 30


Nome da pessoa: Maria
CPF da pessoa: 111111111
Idade da pessoa: 20


Nome da pessoa: Maria
CPF da pessoa: 111111111
Idade da pessoa: 25



--------------------------------
Process exited after 0.1071 seconds wit
Press any key to continue . . .
```

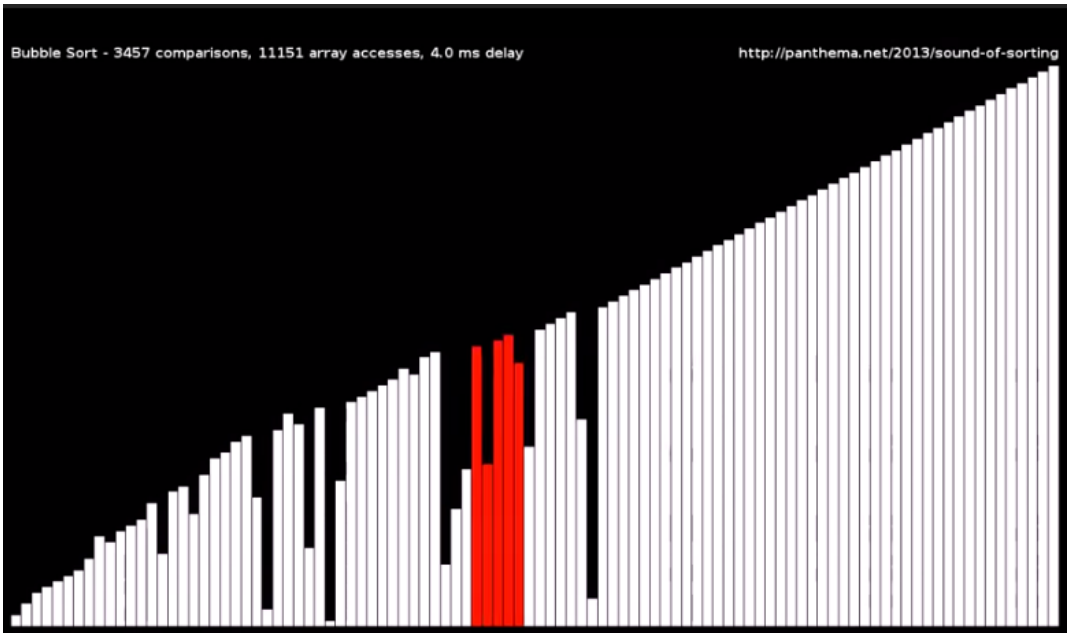
Curso de C++ - Aula 25 - Ordenando carros (bubble sort)

https://youtu.be/dKHqEuaO1y8
https://www.youtube.com/watch?v=dKHqEuaO1y8&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=25&ab_channel=MarcosCastro

Através do **#define** é possível definir constantes no código. Por exemplo:

#define nome_da_constante = valor

Abaixo temos a imagem de como é organizado o **Bubble Sort**:



A organização ocorre de forma crescente.

```cpp
#include <iostream>
#include <string.h>
#define MAX 10

using namespace std;

class Carro
{
public:
    char nome[100];
    char cor[20];
    char placa[20];
    double preco;
};

// ordenação utilizando o algoritmo bubble sort
void ordena(Carro carros[], int tam)
{
    Carro aux;

    for(int i = tam; i >= 0; i--)
    {
        for(int j = 1; j <= i; j++)
        {
            if(strcmp(carros[j - 1].nome, carros[j].nome) == 1)
            {
                aux = carros[j - 1];
                carros[j - 1] = carros[j];
                carros[j] = aux;
            }
        }
    }
}

int main(int argc, char *argv[])
{
    Carro carros[MAX];
    int i = 0;

    while(true)
    {
        char resp;
        cout << "Digite o nome do carro: ";
        cin >> carros[i].nome;
        cout << "Digite o preco: ";
        cin >> carros[i].preco;
        cout << "Voce deseja continuar? <S>SIM ou <N>NAO: ";
        cin >> resp;
        if(resp != 'S')
            break;
        cout << endl;
        i++;
    }

    cout << endl;
    cout << "Exibindo todos os carros...\n\n";

    for(int j = 0; j <= i; j++)
    {
        cout << "Nome do carro: " << carros[j].nome << endl;;
        cout << "Preco: R$" << carros[j].preco << "\n\n";
    }

    ordena(carros, i);

    cout << "Exibindo os carros ordenados pelo nome...\n\n";

    for(int j = 0; j <= i; j++)
    {
        cout << "Nome do carro: " << carros[j].nome << endl;;
        cout << "Preco: R$" << carros[j].preco << "\n\n";
    }
    return 0;
}
```

O **Bubble Sort** é conhecido como algoritmo lento, já que ele compara todos os elementos várias vezes para realizar a organização.
https://gist.github.com/marcoscastro/42e87043d08e0833d4f0

Curso de C++ - Aula 26 - Classes novamente

https://youtu.be/XI952miiyw8
https://www.youtube.com/watch?v=XI952miiyw8&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=26&ab_channel=MarcosCastro

Esse código abaixo não debuga bem no DevC++. No Visual Studio, o debug funciona normalmente.
https://gist.github.com/marcoscastro/9602806ba3d89e769924

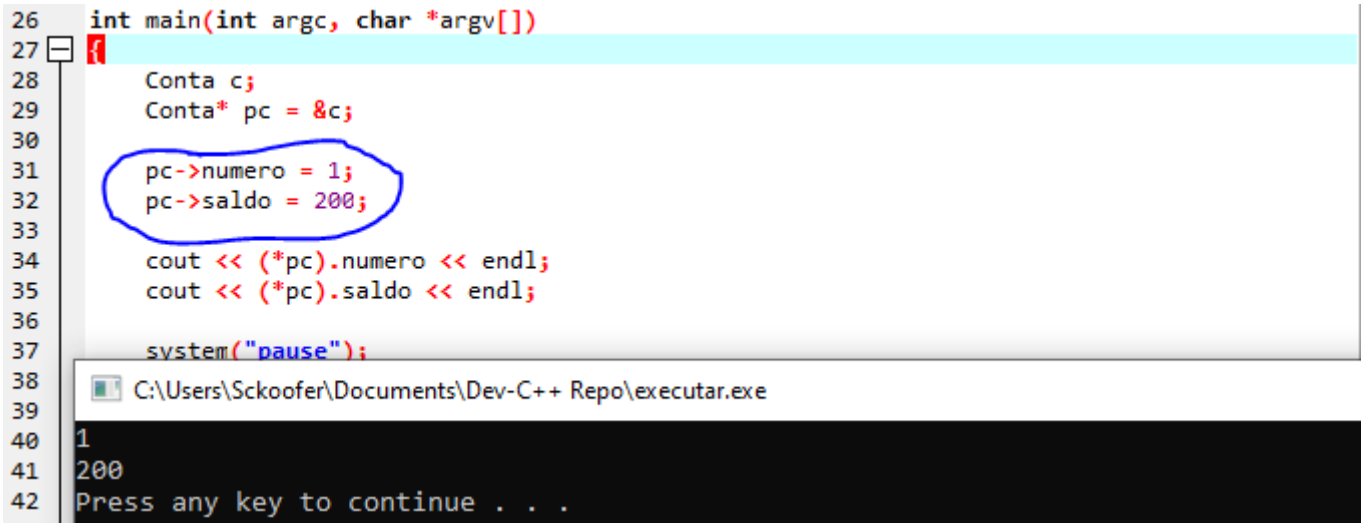
Para usar métodos fora da classe, podemos usar novamente operadores de escopo, como o **::** para indicar um método fora da classe, mas lembrando que construir o seu protótipo ainda dentro da classe.

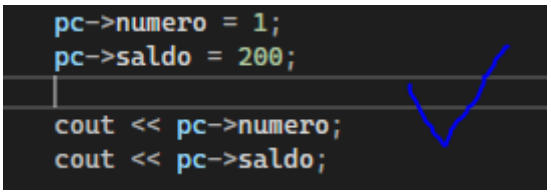
```cpp
#include <iostream>

using namespace std;

class Conta
{
public:
    int numero;
    double saldo;

    double depositar(double quantidade);

    double retirar(double quantidade);
};
double Conta::depositar(double quantidade)
{
    if(quantidade > 0)
        saldo += quantidade;
    return saldo;
}
double Conta::retirar(double quantidade)
{
    if(quantidade > 0 && saldo >= quantidade)
        saldo -= quantidade;
    return saldo;
}
int main(int argc, char *argv[])
{
    Conta c;

    c.numero = 1;
    c.saldo = 100.75;

    cout << "Saldo: " << c.saldo << endl;
    cout << "Saldo depois do deposito: " << c.depositar(100) << endl;
    cout << "Saldo depois do saque: " << c.retirar(-50) << endl;
    return 0;
}
```

Vamos voltar agora os métodos para dentro da classe, vamos fazer outra coisa agora.

```cpp
int main(int argc, char *argv[])
{
    Conta c;
    Conta* pc = &c;

    (*pc).numero = 1;
    (*pc).saldo = 200;

    cout << (*pc).numero << endl;
    cout << (*pc).saldo << endl;

    system("pause");
```

Pergunta: Por que está sendo usado os parênteses no exemplo acima? É por causa da ordem de precedência, o ponto vem antes do *, então pode gerar erro, ao invés do (*ponteiro) podemos usar a ->






```
#include <iostream>
using namespace std;
class Conta
{
public:
	int numero;
	double saldo;
	double depositar(double quantidade)
	{
		if (quantidade > 0)
			saldo += quantidade;
		return saldo;
	}
	double retirar(double quantidade)
	{
		if (quantidade > 0 && saldo >= quantidade)
			saldo -= quantidade;
		return saldo;
	}
	double getSaldo()
	{
		return saldo;
	}
	int getNumero()
	{
		return numero;
	}
};
int main(int argc, char* argv[])
{
	Conta c;
	Conta* pc = &c;
	pc->numero = 1;
	pc->saldo = 200;
	cout << pc->numero;
	cout << pc->saldo;

	return 0;
}
```

Curso de C++ - Aula 28 - Classes e ponteiros

https://youtu.be/iaM7nF6luLc
https://www.youtube.com/watch?v=iaM7nF6luLc&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=28&ab_channel=MarcosCastro

Veja abaixo um exemplo de passagem por valor e passagem por referência.

POR VALOR

```
#include <iostream>
using namespace std;
class Conta
{
public:
    int numero;
    double saldo;
    double depositar(double quantidade) { ... }
    double retirar(double quantidade) { ... }
    double getSaldo()
    {
        return saldo;
    }
    int getNumero()
    {
        return numero;
    }
};
void foo(Conta c)
{
    c.depositar(50);
}
int main(int argc, char* argv[])
{
    Conta c;
    Conta* pc = &c;
    pc->numero = 1;
    pc->saldo = 200;

    foo(c);

    cout << "Saldo: " << pc->getSaldo() << endl;

    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Saldo: 200
C:\Users\Sckoofer\source\repos\ConsoleApplication1\x64\De
```

REFERÊNCIA

```
#include <iostream>
using namespace std;
class Conta
{
public:
    int numero;
    double saldo;
    double depositar(double quantidade) { ... }
    double retirar(double quantidade) { ... }
    double getSaldo()
    {
        return saldo;
    }
    int getNumero()
    {
        return numero;
    }
};
void foo(Conta* pc)
{
    pc->depositar(50);
}
int main(int argc, char* argv[])
{
    Conta c;
    Conta* pc = &c;
    pc->numero = 1;
    pc->saldo = 200;

    foo(pc);

    cout << "Saldo: " << pc->getSaldo() << endl;

    return 0;
}
```

2

1

```
Microsoft Visual Studio Debug Console
Saldo: 250
```

Na passagem por **referência**, a função **void** (somar 50) acrescenta valor à variável ponteiro.
A mesma coisa aconteceria se fizessemos com a referência **&**

```
void foo(Conta& c)
{
    c.depositar(50);
}
int main(int argc, char* argv[])
{
    Conta c;
    Conta* pc = &c;
    pc->numero = 1;
    pc->saldo = 200;

    foo(c);
```

```
Microsoft Visual Studio Debug Console
Saldo: 250
```

Dica: Não precisa utilizar o **this** sempre que for se referenciar ao elemento membro da classe, apenas quando forem nomes iguais, por exemplo:

```
class carro
{
	int valor;
	void metodo(int valor)
	{
		this->valor = valor;
	}
}
```

Vamos falar do operador **new.** É uma palavra chave para alocação dinâmica na memória.

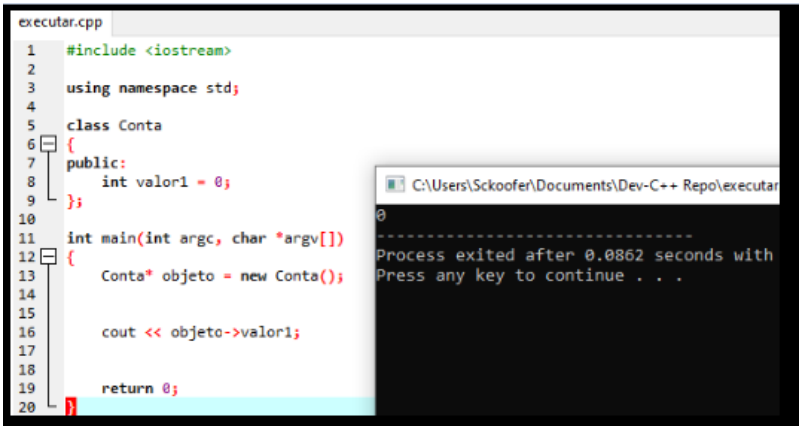
O tamanho da memória alocada será o tamanho criado a partir do tipo em que o **new** é criado.
tipo nome_do_elemento = new **tipo**

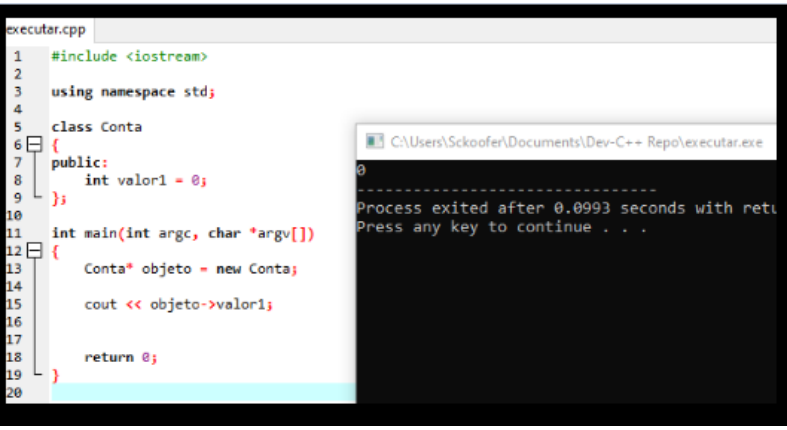
Exemplo:
int variavel = new int;

Exemplo para vetores:
int vetores = new vetores[12];

https://en.cppreference.com/w/cpp/memory/new/operator_new

O operador também é usado para criar instâncias de objetos a partir de classes e estruturas.
Nome_da_classe nome_do_objeto = new Nome_da_classe();






```
double* p = new double[]{1,2,3}; // creates an array of type double[3]
auto p = new auto('c');          // creates a single object of type char. p is a char*
```

Explanation

The new expression attempts to allocate storage and then attempts to construct and initialize either a single unnamed object, or an unnamed array of objects in the allocated storage. The new-expression returns a prvalue pointer to the constructed object or, if an array of objects was constructed, a pointer to the initial element of the array.

Allocation

The new-expression allocates storage by calling the appropriate allocation function. If type is a non-array type, the name of the function is `operator new`. If type is an array type, the name of the function is `operator new[]`.

As described in allocation function, the C++ program may provide global and class-specific replacements for these functions. If the new-expression begins with the optional `::` operator, as in `::new T` or `::new T[n]`, class-specific replacements will be ignored (the function is looked up in global scope). Otherwise, if T is a class type, lookup begins in the class scope of T.

O operador **new** realiza a aloca memória fora da pilha (stack), e armazena na **heap.**

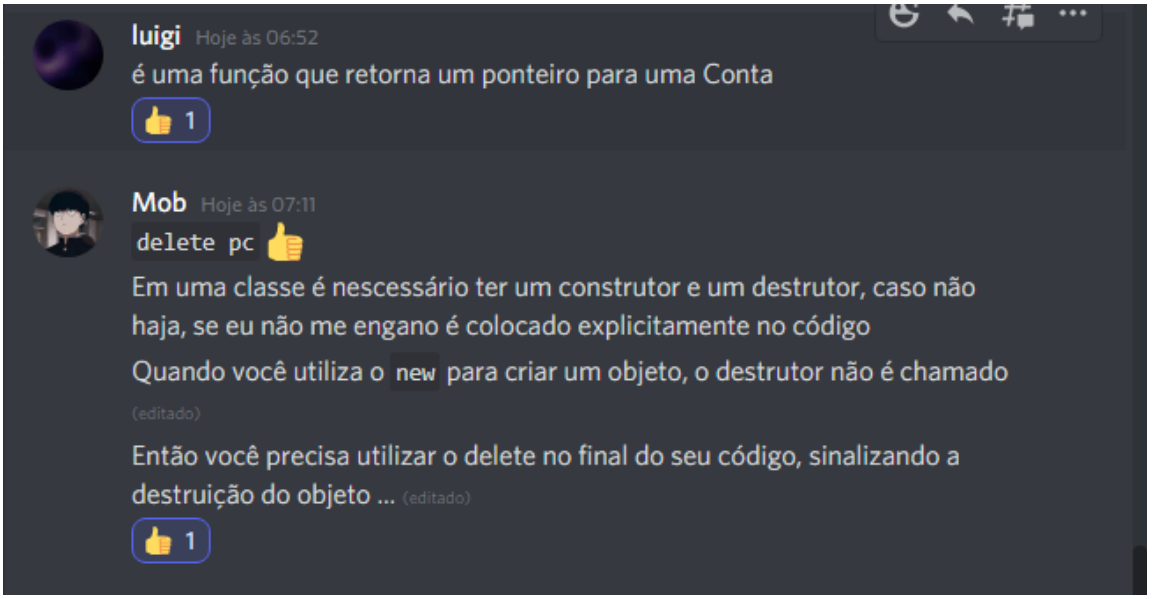
A principal função de **new** é alocar memória, especificamente na seção da memória chamada de **heap.**

Veja um exemplo abaixo:

```
#include <iostream>
using namespace std;
class Conta
{
public:
   int numero;
   double saldo;
   double depositar(double quantidade)
   {
      if(quantidade > 0)
         saldo += quantidade;
      return saldo;
   }
   double retirar(double quantidade)
   {
      if(quantidade > 0 && saldo >= quantidade)
         saldo -= quantidade;
      return saldo;
   }
   double getSaldo()
   {
      return saldo;
   }

   int getNumero()
   {
      return numero;
   }
};
void foo(Conta& c)
{
   c.depositar(50);
}
Conta *novaConta(int numero)
{
   Conta* c = new Conta;
   c->numero = numero;
   c->saldo = 0.0;
   return c;
}
int main(int argc, char *argv[])
{
   Conta* pc = novaConta(1111);
   cout << "Numero: " << pc->getNumero() << endl;
   cout << "Saldo: " << pc->getSaldo() << endl;
   return 0;
}
```

```
Conta *novaConta(int numero)
{
    Conta* c = new Conta;
    c->numero = numero;
    c->saldo = 0.0;
    return c;
}
```

Dúvida, do que se trata Conta* novaConta?



Curso de C++ - Aula 29 - Modificadores de acesso e funções friends

Membros protegidos (protected) são acessíveis para métodos e classes que derivam da mesma classe que os membros protegidos.

Função friend (função amiga), é uma função que não é membro da classe, mas que tem acesso aos membros **protegidos** dentro da classe. Isso serve para expor os membros protegidos. Todas as funções friends terão acesso irrestrito aos membros privados.

executar.cpp

```
1   #include <iostream>
2   #include <string.h>
3
4   using namespace std;
5
6   class Linguagem
7   {
8
9   private:
10      friend void classeAmiga(Linguagem*);
11      char nome[100];
12
13  public:
14      void setNome(char* nome)
15      {
16          strcpy(this->nome, nome);
17      }
18  };
19  void classeAmiga(Linguagem* l)
20  {
21      cout << "Classe amiga diz: " << l->nome << endl;
22  }
23  int main()
24  {
25      Linguagem l;
26      l.setNome("C++");
27      classeAmiga(&l);
28
29      return 0;
30  }
```

Só foi possível utilizar o ponteiro **l** para acessar a classe e mostrar o valor setado para nome(perceba que não se trata de uma função de retorno retornando o valor de nome), pois o método é friend da classe.

executar.cpp

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Linguagem
{

private:
    friend void classeAmiga(Linguagem*);
    char nome[100];

public:
    void setNome(char* nome)
    {
        strcpy(this->nome, nome);
    }
};
void classeAmiga(Linguagem* l)
{
    cout << "Classe amiga diz: " << l->nome << endl;
}
int main()
{
    Linguagem l;
    l.setNome("C++");
    classeAmiga(&l);

    return 0;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe

```
Classe amiga diz: C++

--------------------------------
Process exited after 0.08537 seconds with return value 0
Press any key to continue . . .
```

≠

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Linguagem
{

private:
    char nome[100];

public:
    void setNome(char* nome)
    {
        strcpy(this->nome, nome);
    }
};
void classeAmiga(Linguagem* l)
{
    cout << "Classe amiga diz: " << l->nome << endl;
}
int main()
{
    Linguagem l;
    l.setNome("C++");
    classeAmiga(&l);

    return 0;
}
```

Compile Log | Debug | Find Results | Close

Message
oofer\Documents\Dev-C++ Repo\execu...	In function 'void classeAmiga(Linguagem*)':
oofer\Documents\Dev-C++ Repo\executar....	[Error] 'char Linguagem::nome [100]' is private

https://youtu.be/0OvrreynBKo?t=1254

Uma classe inteira pode ser configurada para ser friend de outra classe.

```cpp
#include <iostream>
#include <string.h>

using namespace std;

//Protótipo da classe
class LinguagemAmiga;

class Linguagem
{
protected:
    LinguagemAmiga* lamiga;

public:
    void mostrarAmiga();
};



class LinguagemAmiga
{
    friend class Linguagem;
    void mostrarLinguagemAmiga()
    {
        cout << "ola, linguagem amiga!\n";
    }
};


//Método usado fora da classe Linguagem
void Linguagem::mostrarAmiga()
{
    lamiga->mostrarLinguagemAmiga();
}


int main(int argc, char *argv[])
{
    Linguagem *variavel = new Linguagem;

    variavel->mostrarAmiga();

    return 0;
}
```

C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe

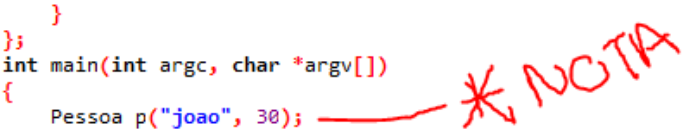
```
ola, linguagem amiga!

--------------------------------
Process exited after 0.09093 seconds with return value 0
Press any key to continue . . .
```

Curso de C++ - Aula 30 - Construtores e destrutores

https://www.youtube.com/watch?v=DPs1kVC_Iqw&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=30&ab_channel=MarcosCastro
https://youtu.be/DPs1kVC_Iqw

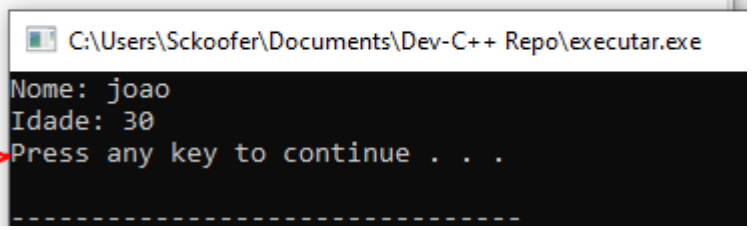
Construtor normalmente não tem retorno, pois não é uma função e nem método.

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Pessoa
{
protected:
    char nome[100];
    int idade;
    int *parentes;
    double * filhos;
public:
    Pessoa(char *nome, int idade)
    {
        strcpy(this->nome, nome);
        this->idade = idade;
        parentes = new int[100];
        filhos = new double[100];

    }
    char * getNome()
    {
        return nome;
    }
    int getIdade()
    {
        return idade;
    }
    ~Pessoa(){
        system("pause");
        delete[] parentes;
        delete[] filhos;

    }
};
int main(int argc, char *argv[])
{
    Pessoa p("joao", 30);

    cout << "Nome: " << p.getNome() << endl;
    cout << "Idade: " << p.getIdade() << endl;

    return 0;
}

```



Nota: Se tivéssemos feito: Pessoa p = **new** Pessoa(“Joao”,30); O destrutor não seria usado automaticamente.

```cpp
    int getIdade()
    {
        return idade;
    }
    ~Pessoa(){
        system("pause");
        delete[] parentes;
        delete[] filhos;

    }
};
```



O problema de instanciar objetos dessa forma, é que todos os métodos, construtores e destrutores, serão acessados automaticamente. Veja o exemplo abaixo:

```
30      ~Pessoa(){
31          cout << "Acessou" << endl;
32      }
33  };
34  int main(int argc, char *argv[])
35  {
36      Pessoa pessoas[3] =
37      {
38          {"joao", 30}, {"maria", 20}, {"statios", 2}
39      };
40
41
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executa
Acessou
Acessou
Acessou

--------------------------------
Process exited after 0.08882 seconds wit
Press any key to continue . . .
```

Os Objetos são destruídos na ordem inversa em relação à ordem em que são construídos.

Curso de C++ - Aula 31 - Sobrecarregando construtores

https://www.youtube.com/watch?v=iSm4tYUQ9LE&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=31&ab_channel=MarcosCastro
https://youtu.be/iSm4tYUQ9LE

```
1   #include <iostream>
2   #include <string.h>
3
4   using namespace std;
5
6   class Carro
7   {
8   public:
9       int ano;
10
11      Carro()
12      {
13          ano = 0;
14      }
15      Carro(int ano)
16      {
17          this->ano = ano;
18      }
19      Carro(int ano, double preco)
20      {
21          this->ano = ano;
22          this->ano = preco;
23      }
24  };
25  int main(int argc, char *argv[])
26  {
27      Carro carro;
28      Carro carro2(2014);
29      Carro carro3(2014, 200000);
30
31      cout << "Ano do carro: " << carro.ano << endl;
32      cout << "Ano do carro2: " << carro2.ano << endl;
33      cout << "Ano do carro3: " << carro3.ano << endl;
34      return 0;
35  }
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo
Ano do carro: 0
Ano do carro2: 2014
Ano do carro3: 200000

--------------------------------
Process exited after 0.09029 seconds w
Press any key to continue . . .
```

Exemplo acima de três construtores sendo usados.
Se não for declarado nenhum construtor em código, o C++ automaticamente implementará um construtor padrão.

Curso de C++ - Aula 32 - Construtor de cópias

https://www.youtube.com/watch?v=t6av9Squ51g&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=32&ab_channel=MarcosCastro
https://youtu.be/t6av9Squ51g

Copiar construtores é chamado de “cópia superficial”. Os membros da classe serão copiados para outra.

```cpp
#include <iostream>
#include <string.h>
using namespace std;

class Estudante
{
protected:
    char* nome;

public:
    Estudante(const char* nome)
    {
        cout << "Construindo objeto: " << nome << endl;
        int tam = strlen(nome) + 1;
        this->nome = new char[tam];
        //strcpy(this->nome, nome);
    }

    Estudante(const Estudante& e)
    {
        cout << "Construindo copia de " << e.nome << endl;

        int tam = strlen(e.nome) + strlen("Copia de ") + 1;
        this->nome = new char[tam];
        //strcpy(this->nome, "Copia de ");
        //strcat(this->nome, e.nome);
    }

    ~Estudante()
    {
        cout << "Destruindo... " << nome << endl;
        delete[] nome;
        nome = 0;
    }

    const char* getNome()
    {
        return nome;
    }
};
void foo2(Estudante e)
{

}
void foo()
{
    Estudante estudante("joao");
    foo2(estudante);
    cout << "Estudante " << estudante.getNome() << endl;
}
int main(int argc, char* argv[])
{
    foo();

    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Construindo objeto: joao
Construindo copia de ═══²²²²
Destruindo... ═══════════²²²=]~
Estudante ═══²²²²
Destruindo... ═══²²²²

C:\Users\Sckoofer\source\repos\ConsoleAppl
process 324) exited with code 0.
To automatically close the console when de
ing->Automatically close the console when
Press any key to close this window . . .
```

Curso de C++ - Aula 33 - Herança

https://www.youtube.com/watch?v=gysAI7JDYRY&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=33&ab_channel=MarcosCastro

https://youtu.be/gysAI7JDYRY

```cpp
#include <iostream>

using namespace std;

class ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe mae" << endl;
    }
};


class ClasseFilha : public ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe filha" << endl;
    }
};

int main()
{
    ClasseMae mae;
    ClasseFilha filha;

    mae.mostrarMensagem();
    filha.mostrarMensagem();
    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Ola, sou a classe mae
Ola, sou a classe filha

C:\Users\Sckoofer\source\repos\ConsoleApplication5
To automatically close the console when debugging
le when debugging stops.
Press any key to close this window . . .
```

Exemplo 1 acima, exemplo 2 abaixo:

```cpp
#include <iostream>
#include <string.h>

 using namespace std;

class Animal
{
protected:
    char* nome;

public:
    Animal(const char* nome)
    {
        cout << "Construindo animal..." << endl;
        this->nome = new char[strlen(nome) + 1];

    }
    ~Animal()
    {
        delete[] nome;
        nome = 0;
    }
    const char* getNome()
    {
        return nome;
    }
};

class Cachorro : public Animal
{
protected:
    int idade;

public:
    Cachorro(const char* nome) : Animal(nome)
    {
        cout << "Construindo cachorro..." << endl;
        idade = 0;
    }
    int getIdade()
    {
        return idade;
    }
    void setIdade(int idade)
    {
        this->idade = idade;
    }
};


int main(int argc, char* argv[])
{
    Cachorro c("yankee");
    c.setIdade(5);

    cout << "Nome: " << c.getNome() << endl;
    cout << "Idade: " << c.getIdade();
    return 0; ≤ 1ms elapsed
}
```

```
C:\Users\Sckoofer\source\rep
Construindo animal...
Construindo cachorro...
Nome: ========2222
Idade: 5
```

Curso de C++ - Aula 34 - Funções virtuais

https://www.youtube.com/watch?v=0ZH5-bUlXK8&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=34&ab_channel=MarcosCastro
https://youtu.be/0ZH5-bUlXK8

```cpp
#include <iostream>

using namespace std;

class ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe mae" << endl;
    }
};


class ClasseFilha : public ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe filha" << endl;
    }
};

int main()
{
    ClasseMae mae;
    ClasseFilha filha;

    mae.mostrarMensagem();
    filha.mostrarMensagem();
    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Ola, sou a classe mae
Ola, sou a classe filha

C:\Users\Sckoofer\source\repos\ConsoleApplication5
To automatically close the console when debugging
le when debugging stops.
Press any key to close this window . . .
```

O **C++** decide, em tempo de compilação, qual função ele vai chamar. Isso se chama “**Ligação prematura**”.

Vamos falar agora sobre a palavra chave “**virtual**”.

SEM VIRTUAL

```cpp
#include <iostream>

using namespace std;

class ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe mae" << endl;
    }
};

class ClasseFilha : public ClasseMae
{
public:
    void mostrarMensagem()
    {
        cout << "Ola, sou a classe filha" << endl;
    }
};
void foo(ClasseMae* p)
{
    p->mostrarMensagem();
}
int main(int argc, char *argv[])
{
    ClasseMae mae;
    ClasseFilha filha;

    foo(&mae);
    foo(&filha);
    return 0;
}
```

```
Select C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Ola, sou a classe mae
Ola, sou a classe mae

--------------------------------
```

COM VIRTUAL

```cpp
#include <iostream>

using namespace std;

class ClasseMae
{
public:
    virtual void mostrarMensagem()
    {
        cout << "Ola, sou a classe mae" << endl;
    }
};

class ClasseFilha : public ClasseMae
{
public:
    virtual void mostrarMensagem()
    {
        cout << "Ola, sou a classe filha" << endl;
    }
};
void foo(ClasseMae* p)
{
    p->mostrarMensagem();
}
int main(int argc, char *argv[])
{
    ClasseMae mae;
    ClasseFilha filha;

    foo(&mae);
    foo(&filha);
    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
Ola, sou a classe mae
Ola, sou a classe filha

--------------------------------
```

No primeiro exemplo, a chamada do objeto cai diretamente para a classe original. Quando queremos fazer o mesmo uso, porém da classe herdada, nós usamos o método **virtual.**
Isso que a palavra chave **virtual** realiza é chamado de “ligação tardia”

Curso de C++ - Aula 36 - Tipos de dados abstratos parametrizados (classes template)

https://www.youtube.com/watch?v=gmnfY7kEz28&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=36&ab_channel=MarcosCastro
https://youtu.be/gmnfY7kEz28

Não finalizada. Veja um exemplo de um template:

```cpp
using namespace std;

template <class Type>
class Pilha
{
private:
    Type* vet;
    int max_tam;
    int topo;
public:
```

Curso de C++ - Aula 37 - Classe String

https://www.youtube.com/watch?v=h98GWpCejL0&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=37&ab_channel=MarcosCastro
https://youtu.be/h98GWpCejL0

Apenas várias funcionalidade de string

Curso de C++ - Aula 38 - Parâmetros opcionais

São usados para inicializar variáveis de parâmetros quando não for passado nenhum valor para as funções.

```cpp
#include <iostream>

using namespace std;

int funcao(int variavel = 20){

    return variavel;
}

int main(int argc, char *argv[])
{
    cout << funcao();


    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\executar.exe
20
--------------------------------
Process exited after 0.3078 seconds with retur
Press any key to continue . . .
```

Curso de C++ - Aula 39 - Operador ternário

https://www.youtube.com/watch?v=LUAU1_MWbIs&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=39&ab_channel=MarcosCastro
https://youtu.be/LUAU1_MWbIs

Operador ternário é um operador de operação

```
<condicao> ? <operacao 1> : <operacao 2>
```

```cpp
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int variavel = 10;


    variavel == 10 ? cout<<"eh igual":cout<<"nao eh igual";
    cout << "\n\n\n";


    variavel < 10 ? cout<<"eh menor":cout<<"nao eh menor";
    cout << "\n\n\n";


    variavel > 10 ? cout<<"eh maior":cout<<"nao eh maior";
    cout << "\n\n\n";

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C
eh igual


nao eh menor


nao eh maior



--------------------------------
Process exited after 0.1015 s
Press any key to continue . .
```

Curso de C++ - Aula 40 - Revisando alguns conceitos

https://www.youtube.com/watch?v=H6hAJWp1Erk&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=40&ab_channel=MarcosCastro
https://youtu.be/H6hAJWp1Erk

Herança Múltipla

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

class Pai1
{
public:
    void foo()
    {
        cout << "Oi, sou o Pai1." << endl;
    }
};

class Pai2
{
public:
    void foo()
    {
        cout << "Oi, sou o Pai2." << endl;
    }
};

// a classe Filha herda todos os membros de Pai1 e Pai2
class Filha : public Pai1, public Pai2
{

};

int main(int argc, char *argv[])
{
    Filha f;


    // chamando foo() de Pai1
    f.Pai1::foo();

    // chamando foo() de Pai2
    f.Pai2::foo();

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo
Oi, sou o Pai1.
Oi, sou o Pai2.

--------------------------------
Process exited after 0.08513 secon
Press any key to continue . . .
```

Curso de C++ - Aula 41 - Sobrecarga de operadores

https://www.youtube.com/watch?v=mD5FRTpvZKM&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=41&ab_channel=MarcosCastro
https://youtu.be/mD5FRTpvZKM

Não é possível utilizar operadores elementares para calcularmos objetos de classes.

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

int main(int argc, char *argv[])
{
    Pessoa p1(20), p2(30), p3;

    p3 = p1 + p2;

    return 0;
}
```

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

class Complexo
{
    public:
        int real, imag;

        Complexo(int real, int imag)
        {
            this->real = real;
            this->imag = imag;
        }

    Complexo operator+(Complexo &c)
    {
        return Complexo(this->real + c.real, this->imag + c.imag);
    }
};

int main(int argc, char *argv[])
{
    Complexo c1(1, 2), c2(3, 4);
    Complexo c3 = c1 + c2;

    cout << "Parte real: " << c3.real << endl;
    cout << "Parte imaginaria: " << c3.imag << endl;



    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\exe
Parte real: 4
Parte imaginaria: 6

--------------------------------
Process exited after 0.1003 seconds w
Press any key to continue . . .
```

+ é uma abreviação do operador. Dentro da classe estamos declarando uma redefinição do mesmo operador(**operator+**).
Está sendo definido o operador para calcular objetos.

Curso de C++ - Aula 42 - Funções inline

https://www.youtube.com/watch?v=0oSZ4vnQDv8&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=42&ab_channel=MarcosCastro
https://youtu.be/0oSZ4vnQDv8

O operador inline troca a chamada da função em código, em troca recebe a função.

executar.cpp

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

void foo()
{
    cout << "Aprendendo C++\n";
}

int main(int argc, char *argv[])
{
    foo();

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo\exec
Aprendendo C++

--------------------------------
```

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

inline void foo()
{
    cout << "Aprendendo C++\n";
}

int main(int argc, char *argv[])
{
    foo();

    return 0;
}
```

```
C:\Users\Sckoofer\Documents\Dev-C++ Repo
Aprendendo C++

--------------------------------
Process exited after 0.1373 second
```

Curso de C++ - Aula 43 - Fluxo IO - Arquivos

https://www.youtube.com/watch?v=i-SGS8XbaBE&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=43&ab_channel=MarcosCastro

https://youtu.be/1LqyItkf8Ew

Atividades propostas:

Lição: 1 - Fazer uma calculadora

bubble sort

Curso de C++ - Aula 35 - TAD Pilha (Stack)

https://www.youtube.com/watch?v=O2QqOGGXpts&list=PL8eBmR3QtPL13Dkn5eEfmG9TmzPpTp0cV&index=35&ab_channel=MarcosCastro

https://youtu.be/O2QqOGGXpts