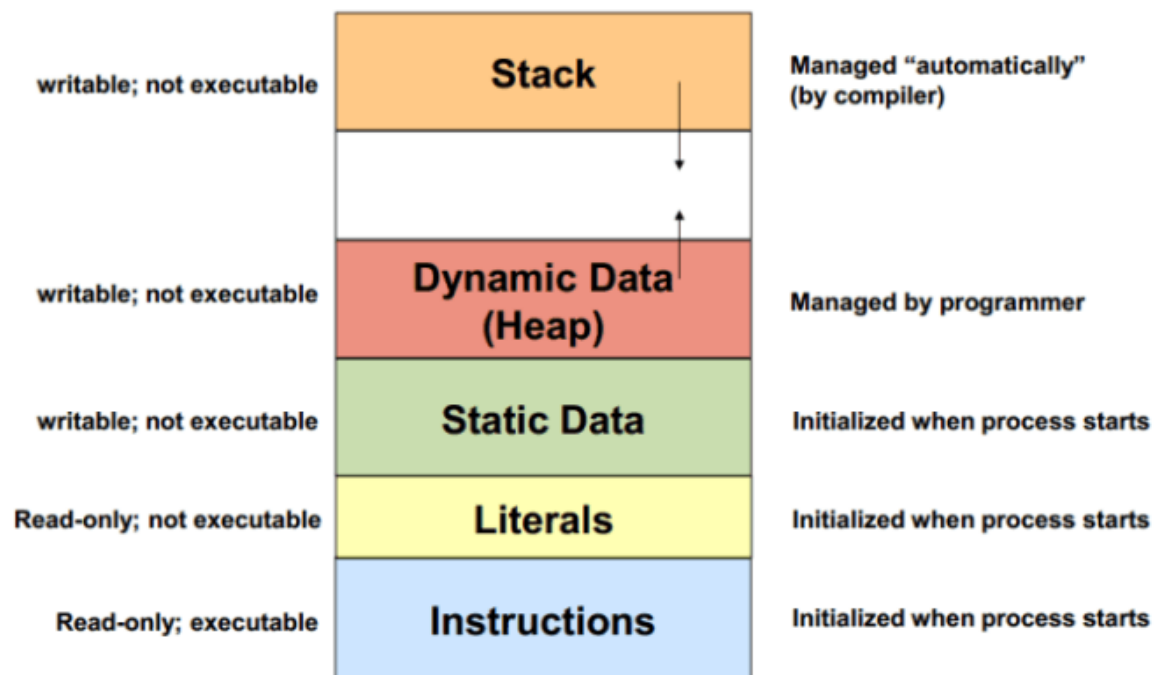


# Указатели, референции и работа с паметта

## Памет



Паметта в C++ програмите е разделена на 5 части (т.нар. сегменти)

### Кодов сегмент (Instructions)

В него се съдържа кода и инструкциите към операционната система и хардуера. От там се изпълнява самата програма

### Сегмент на данните (Data segment, Literals)

Тук се намират инициализираните предварително променливи. Например ако имаме глобална променлива декларирана по следният начин:

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

int global_var = 5;

int main()
{
    return 0;
}
```

тя ще бъде дефинирана в този сегмент

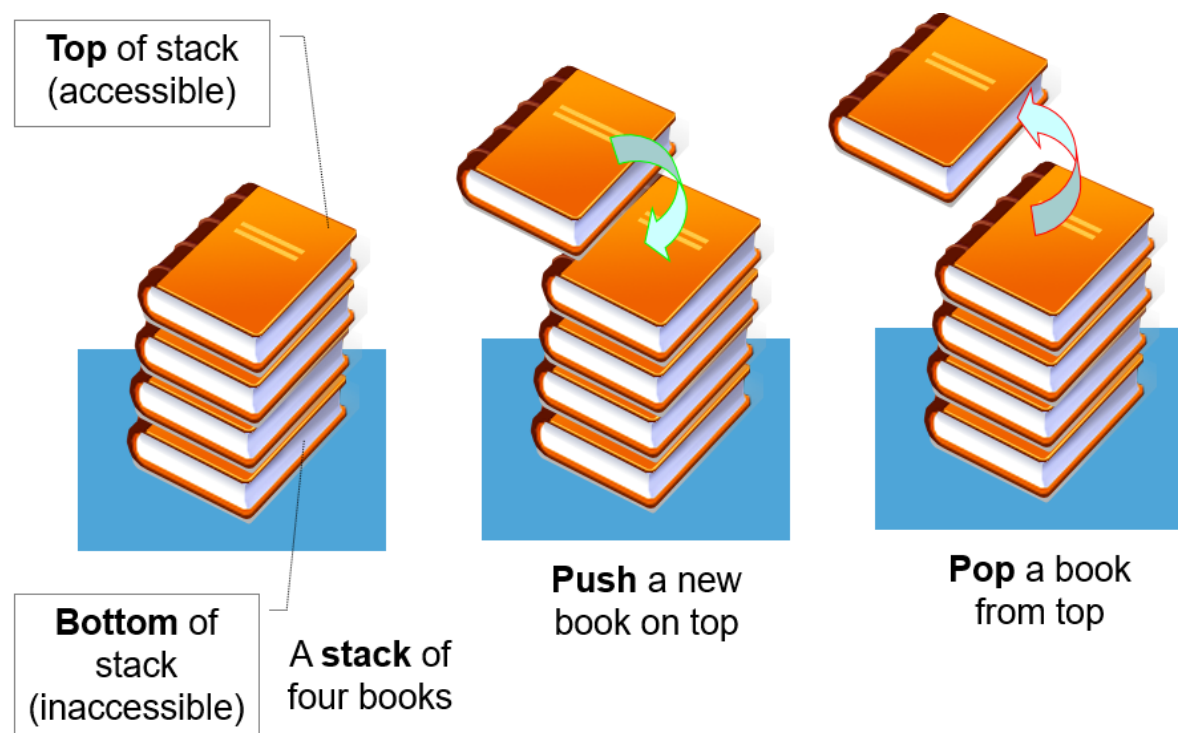
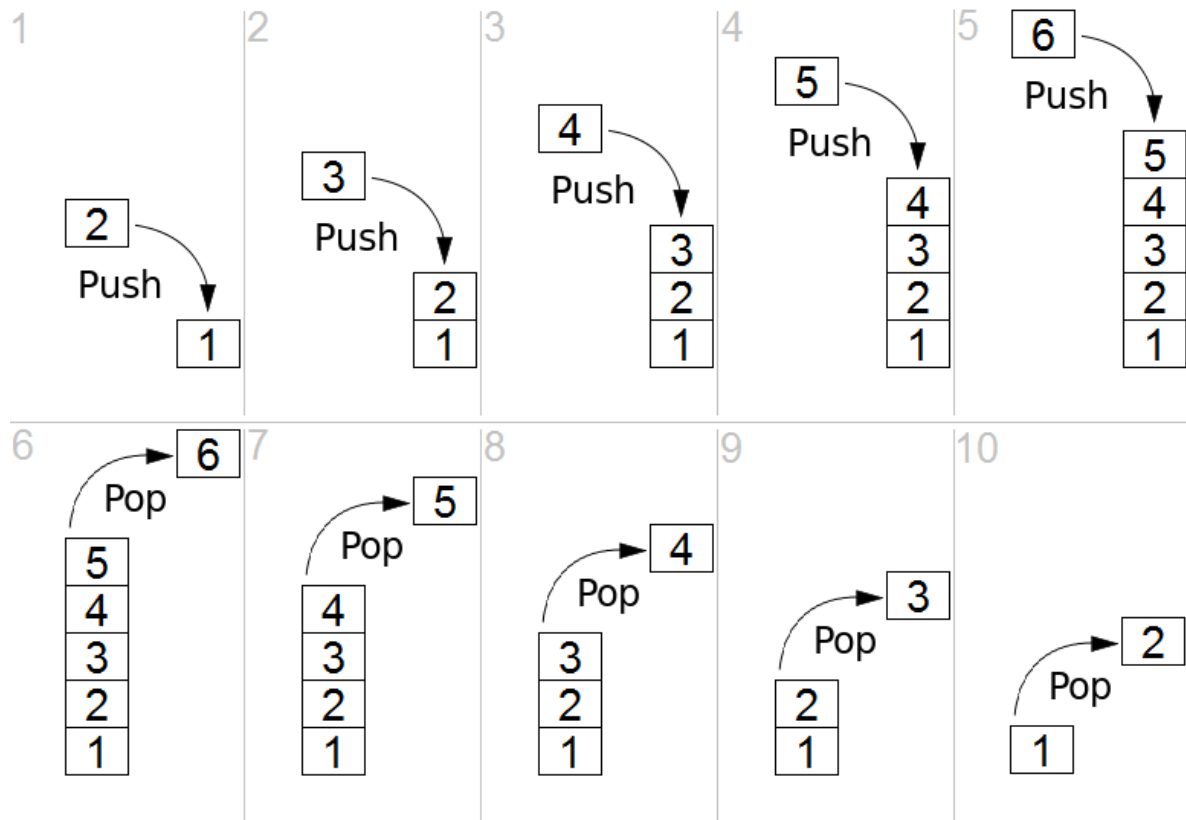
### Сегмент на статичните данни (Static data)

Ако имаме глобална променлива, която не е инициализирана или пък статична променлива (затова ще говорим по-късно), тя ще е в този сегмент.

## Стеков сегмент (Stack)

Стекът е структура от данни, която имплементира следните две операции - **вмъкване** над последният елемент и **изваждане** на последният вмъкнат елемент.

Човешки пример - пакет с бисквити - можем да вземем само най-горната бисквита. А за да поставим нещо в "пакета", можем да го поставим само най-горе в пакета.



## Динамична памет (Heap)

Там се държи т.нар. динамична памет. За нея ще говори малко по-късно в курса. Само ще кажем, че тук програмиста се грижи за заделянето и освобождаването ѝ.

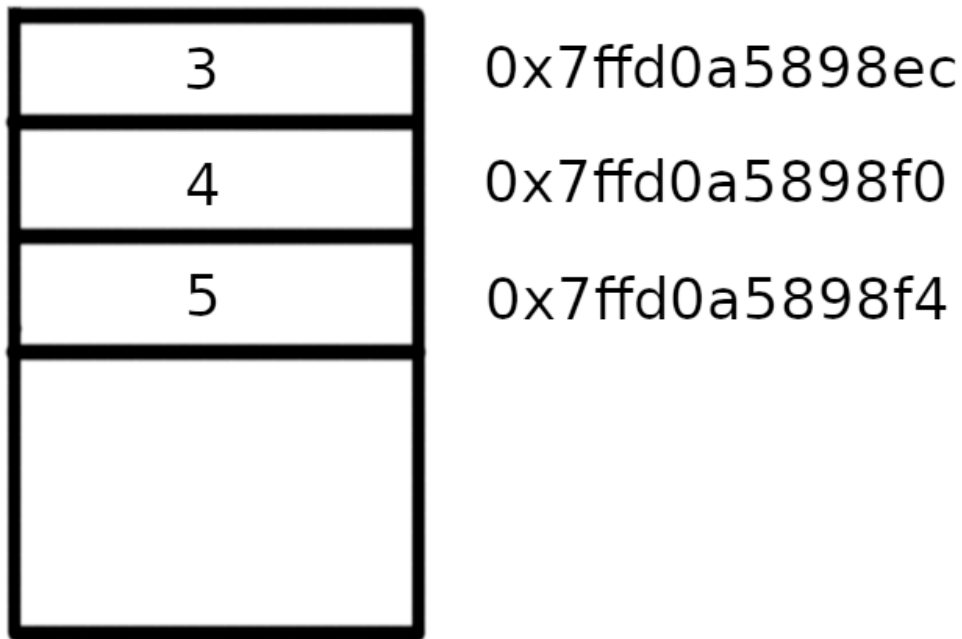
## Представяне на паметта

Когато декларираме някаква променлива, тя се записва в паметта. На всяка клетка в паметта се съпоставя адрес. Съответно всяка променлива си има адрес в паметта.

```
#include <iostream>

using namespace std;

int main()
{
    int a = 3, b = 4, c = 5;
    return 0;
}
```



*Забелязваме, че адресите са през 4 байта*

Ако искаме да видим адреса на дадена променлива, използваме оператора `&`.

```
#include <iostream>

using namespace std;

int main()
{
    int a = 3, b = 4, c = 5;
    cout << &a << endl << &b << endl << &c << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
0x7ffd72033abc
0x7ffd72033ac0
0x7ffd72033ac4
03:41:42 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

## Указатели

В C++ има специални типове променливи, чиито стойности са адреси към паметта. Това са т.нар. указатели. Нека разгледаме следният пример:

```
#include <iostream>

using namespace std;

int main()
{
    int variable = 5;
    int* pointer_to_variable = &variable;

    cout << "Variable is equal to " << variable << endl;
    cout << "The address of the variable is " << &variable << endl;
    cout << "The pointer_to_variable is equal to " << pointer_to_variable <<
endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Variable is equal to 5
The address of the variable is 0x7ffdb558b1ec
The pointer_to_variable is equal to 0x7ffdb558b1ec
03:51:54 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Променивата `variable` има за стойност 5, и се намира на адрес `0x7ffdb558b1ec`.

Синтаксисът на указателите е следният:

```
тип_на_променливата * име_на_променливата
```

т.е. в нашия случай, променливата `pointer_to_variable` е от тип **указател към int** и нейното съдържание е адреса на променливата `variable`.

Ако искаме да видим стойността в даден адрес (т.е. указател), използваме оператор `*` пред името на указателя.

```
#include <iostream>

using namespace std;

int main()
{
    int variable = 5;
    int* pointer_to_variable = &variable;

    cout << "Variable is equal to " << variable << endl;
    cout << "The pointer_to_variable is equal to " << pointer_to_variable <<
endl;
    cout << "The content of pointer_to_variable is equal to " <<
*pointer_to_variable << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Variable is equal to 5
The pointer_to_variable is equal to 0x7ffcc8dd505c
The content of pointer_to_variable is equal to 5
04:36:45 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Още примери:

```
#include <iostream>

using namespace std;

int main()
{
    int integer_var = 5;
    double double_var = 3.4;

    int* integer_ptr = &integer_var;
    double* double_ptr = &double_var;

    cout << "integer_var's address is " << &integer_var << " and the value
inside the address in integer_ptr is " << *integer_ptr << endl;
    cout << "double_var's address is " << &double_var << " and the value inside
the address in double_ptr is " << *double_ptr << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
integer_var's address is 0x7ffc47baabfc and the value inside the address in integer_ptr is 5
double_var's address is 0x7ffc47baac00 and the value inside the address in double_ptr is 3.4
05:35:24 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Когато искаме да дадем стойност по подразбиране на указател, може да използваме `nullptr`.

## Void указатели

Съществуват и т.нар. `void` указатели. Те нямат конкретен тип, при тях приоритета е върху самия адрес, който се пази.

Чрез променяне на типа (cast-ване) на този `void` указател, можем да достъпваме различни типове данни.

```
#include <iostream>

using namespace std;

int main()
{
    int integer_var = 5;
    double double_var = 3.4;

    void* void_ptr = &integer_var;
    cout << "integer_var's address is " << &integer_var << " and the value inside the address in void_ptr as int* is " << *(int*)void_ptr << endl;

    void_ptr = &double_var;
    cout << "double_var's address is " << &double_var << " and the value inside the address in void_ptr as double* is " << *(double*)void_ptr << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
integer_var's address is 0x7ffdb38fb5f4 and the value inside the address in void_ptr as int*
is 5
double_var's address is 0x7ffdb38fb5f8 and the value inside the address in void_ptr as double
* is 3.4
05:35:56 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

## Константни указатели и указатели към константи

Когато си говорехме за променливи, споменахме за т.нар. константни променливи (променливи, които не могат да си променят стойностите). Това става като пред типа на променливата напишем ключовата дума `const`.

Ще разгледаме следните две ситуации:

### Когато искаме да дефинираме указател към константна променлива:

Синтаксисът е следният: `const тип_на_променливата* име_на_променливата`

### Когато искаме да дефинираме константен указател:

Синтаксисът е следният: `тип_на_променливата* const име_на_променливата`

```
#include <iostream>

using namespace std;

int main()
{
    const int PI = 3.14;
    const int* pointer_to_PI = &PI;

    cout << "The value inside the address in pointer_to_PI is constant. PI
cannot be changed " << *pointer_to_PI << endl;

    int variable = 5;
    int* const const_ptr_to_variable = &variable;
    variable++;

    cout << "The value of the pointer is not changable, but variable itself is "
<< variable << endl;
    return 0;
}
```



```
cpp_playground : bash — Konsole
The value inside the address in pointer_to_PI is constant. PI cannot be changed 3
The value of the pointer is not changable, but variable itself is 6
05:44:25 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

## Промяна на променливи, чрез използване на указатели

Можем да променяме променливите чрез указателите към тях. Това става по следният начин:

```
#include <iostream>

using namespace std;

int main()
{
    int variable = 5;

    cout << "Variable value: " << variable << endl;

    int* ptr_to_variable = &variable;
    (*ptr_to_variable)++;

    cout << "Variable value: " << variable << endl;

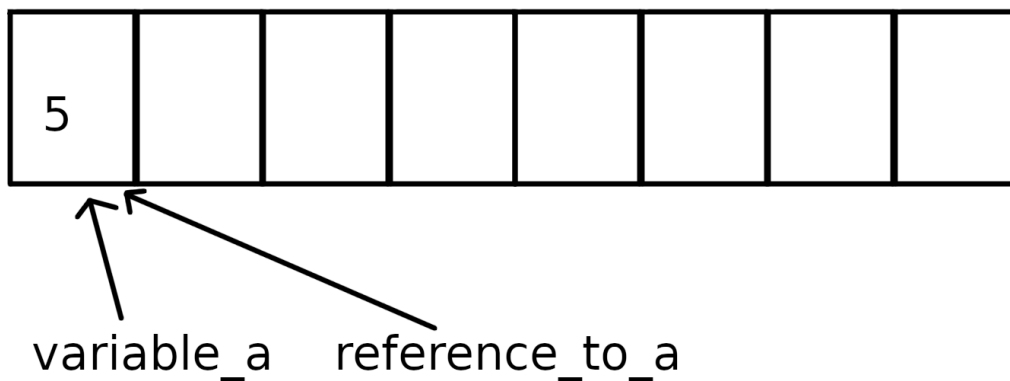
    return 0;
}
```

```
cpp_playground : bash — Konsole
Variable value: 5
Variable value: 6
05:46:17 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Когато извикаме оператора `*` върху някоя променлива, той "връща" обект. Ако оградим този обект със скоби, можем да го използваме все едно е обикновена променлива.

## Референции

Освен указатели, съществуват друг вид променливи - т.нар. референции (псевдоними). Това е начин, две "променливи" да имат една и съща стойност.



```
#include <iostream>

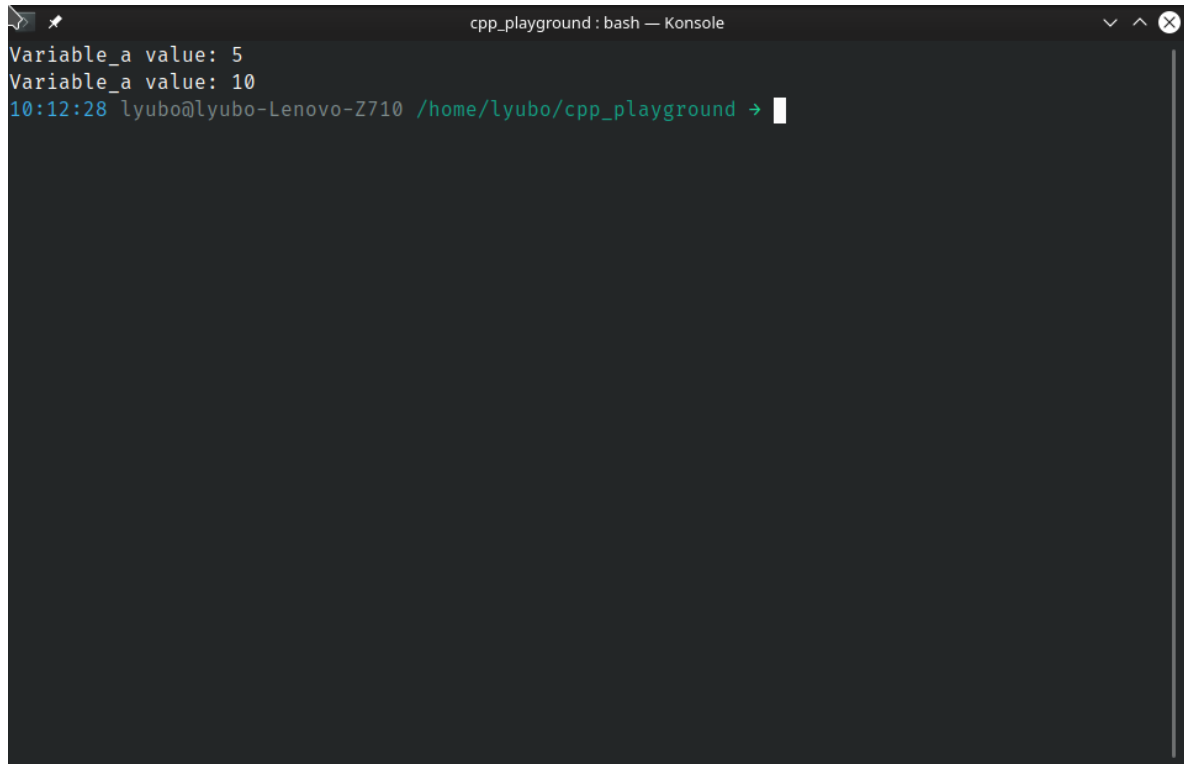
using namespace std;

int main()
{
    int variable_a = 5;
    int& reference_to_a = variable_a;

    cout << "Variable_a value: " << variable_a << endl;

    reference_to_a = 10;
    cout << "Variable_a value: " << variable_a << endl;
}
```

```
    return 0;
}
```



```
cpp_playground : bash — Konsole
Variable_a value: 5
Variable_a value: 10
10:12:28 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

## Използване на указатели и референции при функции

Една от основните употребите на референциите и указателите са при функциите. Нека разгледаме следния сценарий.

Нека имаме функция, която променя дадено число, като го увеличава с единица, само ако числото е четно.

```
#include <iostream>

using namespace std;

void change_var_if_even(int var)
{
    if(var % 2 == 0)
    {
        var += 1;
    }
}

int main()
{
    int some_value = 4;
    cout << "Value before function: " << some_value << endl;
    change_var_if_even(some_value);
    cout << "Value after function: " << some_value << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Value before function: 4
Value after function: 4
11:31:09 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Забелязваме, че променливата в `main` не се е променила. Това се случва, защото когато подадем аргументи към функция, функцията работи с **копие** на тези аргументи. Можем да проверим това, като принтираме адресите на променливите.

```
#include <iostream>

using namespace std;

void change_var_if_even(int var)
{
    cout << "Address of the variable in the function: " << &var << endl;
    if(var % 2 == 0)
    {
        var += 1;
    }
}

int main()
{
    int some_value = 4;
    cout << "Value before function: " << some_value << endl;
    cout << "Address of the value in main(): " << &some_value << endl;
    change_var_if_even(some_value);
    cout << "Value after function: " << some_value << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Value before function: 4
Address of the value in main(): 0x7fff0cb02bc4
Address of the variable in the function: 0x7fff0cb02bac
Value after function: 4
11:36:02 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Един начин да се справим с този проблем е да връщаме променената стойност във функцията, а в `main()` да присвояваме новата стойност на старата променлива

```
#include <iostream>

using namespace std;

int change_var_if_even(int var)
{
    if(var % 2 == 0)
    {
        var += 1;
    }

    return var;
}

int main()
{
    int some_value = 4;
    cout << "Value before function: " << some_value << endl;
    some_value = change_var_if_even(some_value);
    cout << "Value after function: " << some_value << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Value before function: 4
Value after function: 5
11:37:47 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Друг вариант, без да връщаме променливата, запазвайки типа на функцията `void`, е чрез използването на референции - в този случай, функцията не приема копие на променливата, а референция към нея - друго име, сочещо към същото място в паметта.

```
#include <iostream>

using namespace std;

void change_var_if_even(int& var)
{
    if(var % 2 == 0)
    {
        var += 1;
    }
}

int main()
{
    int some_value = 4;
    cout << "Value before function: " << some_value << endl;
    change_var_if_even(some_value);
    cout << "Value after function: " << some_value << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Value before function: 4
Value after function: 5
11:39:53 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

Освен референция, можем да подаваме и самият адрес на променливата, чрез указатели

```
#include <iostream>

using namespace std;

void change_var_if_even(int* var)
{
    if(*var % 2 == 0)
    {
        *var += 1;
    }
}

int main()
{
    int some_value = 4;
    cout << "Value before function: " << some_value << endl;
    change_var_if_even(&some_value);
    cout << "Value after function: " << some_value << endl;
    return 0;
}
```

```
cpp_playground : bash — Konsole
Value before function: 4
Value after function: 5
11:41:20 lyubo@lyubo-Lenovo-Z710 /home/lyubo/cpp_playground →
```

## Задачи

---

1. Напишете функция, която разменя стойностите на две числа
-