Wstęp do informatyki- wykład 9

Funkcje cd- przekazywanie parametrów przez wartość i referencję

Treści prezentowane w wykładzie zostały oparte o:

- S. Prata, Język C++. Szkoła programowania. Wydanie VI, Helion, 2012
- *www.cplusplus.com*
- Jerzy Grębosz, Opus magnum C++11, Helion, 2017
- B. Stroustrup, Język C++. Kompendium wiedzy. Wydanie IV, Helion, 2014
- S. B. Lippman, J. Lajoie, Podstawy języka C++, WNT, Warszawa 2003.

Funkcje – deklaracja, definicja i wywołanie

Prosty przykładowy program zawierający funkcję:

```
#include < iostream>
using namespace std;
int suma (int x, int y);  //deklaracja funkcji
int main()
{
      int a = 10, b = 20;
      int c = suma (a, b);   //wywołanie funkcji
      cout << c;
}
int suma (int x, int y)   //definicja funkcji
{
    return x + y; //funkcja zwraca sumę argumentów
}
```

Tworzenie funkcji - 1 sposób – definicja przed main()

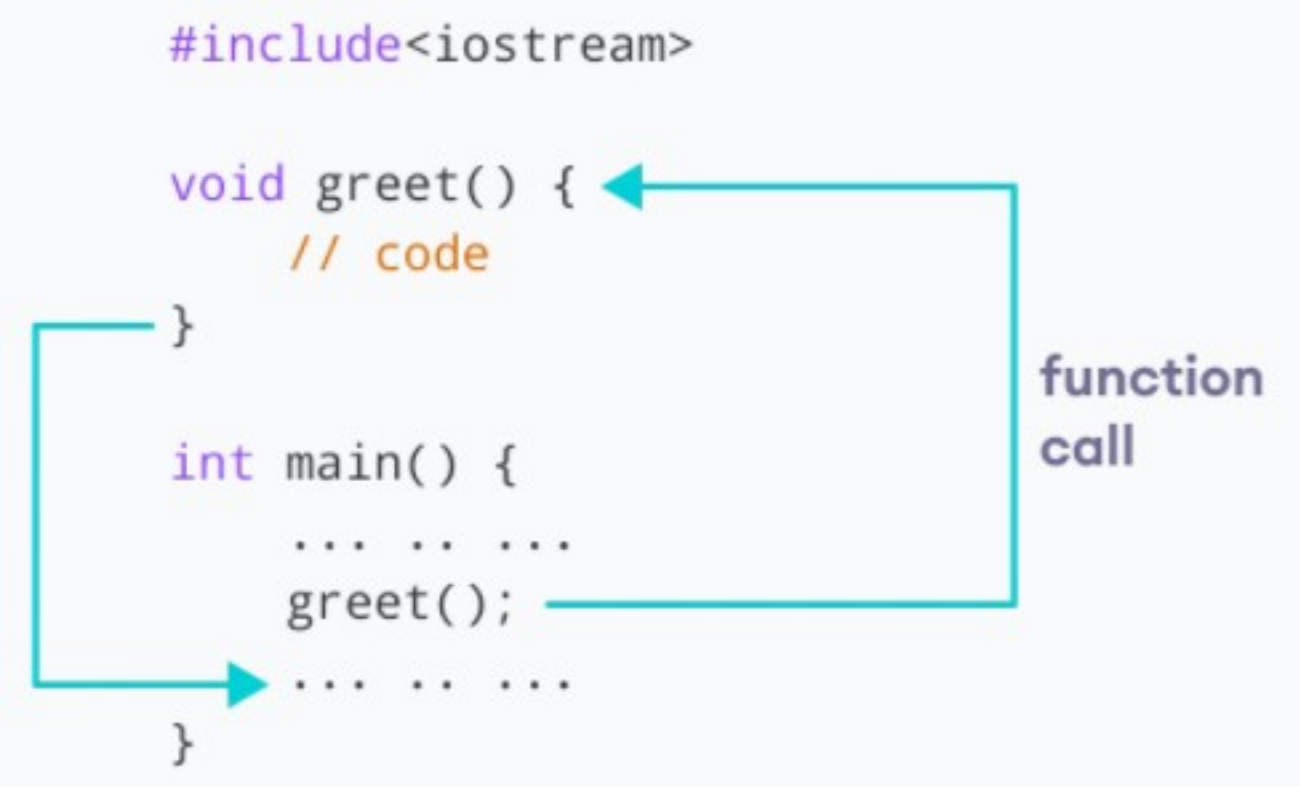
```
#include <iostream>

using namespace std;

//definicja funkcji wraz z jej ciałem
typ_funkcji nazwa_funkcji(lista argumentów)
{
   //ciało funkcji
     return wartość;
}


int main()
{
//instrukcje funkcji main

//m.in. wywoływanie funkcji

     return 0;
}
```

Tworzenie funkcji - 2 sposób – deklaracja przed main()

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

//deklaracje(prototypy) funkcji
typ_funkcji1 nazwa_funkcji(lista argumentów);
//przy tworzeniu prototypów wstawiamy średnik na końcu
typ_funkcji2 nazwa_funkcji_drugiej(lista argumentów);
...
int main()
{
//instrukcje funkcji main, m.in.. wywołanie funkcji
    return 0;
}

//DEFINICJE - implementacje funkcji zadeklarowanych powyżej
typ_funkcji1 nazwa_funkcji(lista argumentów)
{
    //treść - instrukcje funkcji
    return wartość;
}

//implementacja funkcji zadeklarowanej powyżej
typ_funkcji2 nazwa_funkcji_drugiej(lista argumentów)
{
    //treść -instrukcje funkcji
    return wartość;
}
```

Funkcje- powtórzenie – funkcje typu void

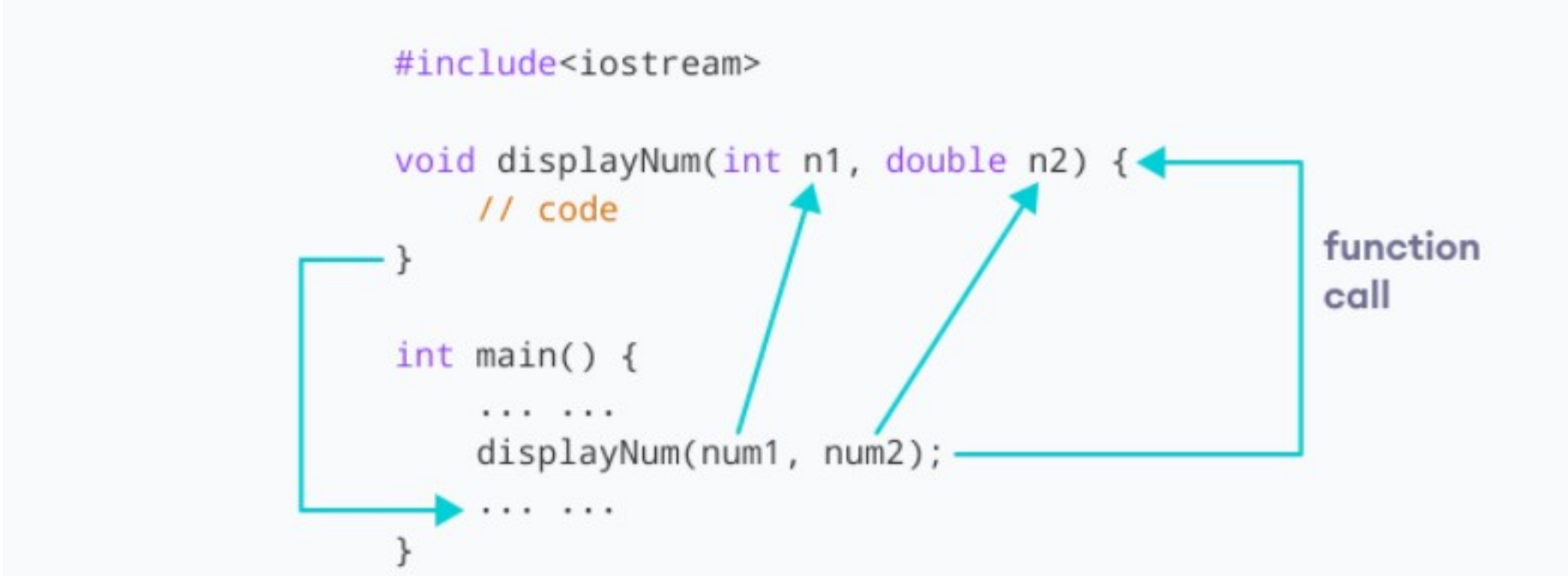
```
void greet() {//definicja funkcji

    cout << "Hello World";
}

int main() {

    // wywołanie funkcji typu void
    greet();
}
```



Funkcje- powtórzenie- funkcje typu void

```
// funkcja wyświetla przekazane parametry liczbę całkowitą
// i rzeczywista
#include <iostream>
using namespace std;

// definicja funkcji
void displayNum(int n1, double n2) {
    cout << "The int number is " << n1;
    cout << "The double number is " << n2;
}


int main() {
    int num1 = 5;
    double num2 = 5.5;


    // wywołanie funkcji
    displayNum(num1, num2);


    return 0;
}
```

Funkcje- powtórzenie- funkcje typu void



Typy argumentów używanych podczas wywołania funkcji muszą odpowiadać odpowiadającym im parametrom w deklaracji/definicji funkcji.

Parametry funkcji i przekazywanie przez wartość

```
double cube(double x); //prototyp
```

C++ normalnie przekazuje parametry do funkcji **przez wartość,** oznacza to, że jeśli do funkcji przekazywana jest liczba, to tworzona jest nowa zmienna. Argumenty przesłane do funkcji są zatem tylko **kopiami**.

Jakiekolwiek działanie na argumentach przekazanych przez wartość nie dotyczy oryginału.

```
double side = 5;
```

```
double vol = cube(side);
```

Kiedy funkcja **cube** jest wywoływana tworzona jest nowa zmienna `x` typu `double` inicjalizowana wartością 5, `cube()` działa na **kopii** `side`, a nie na oryginale.

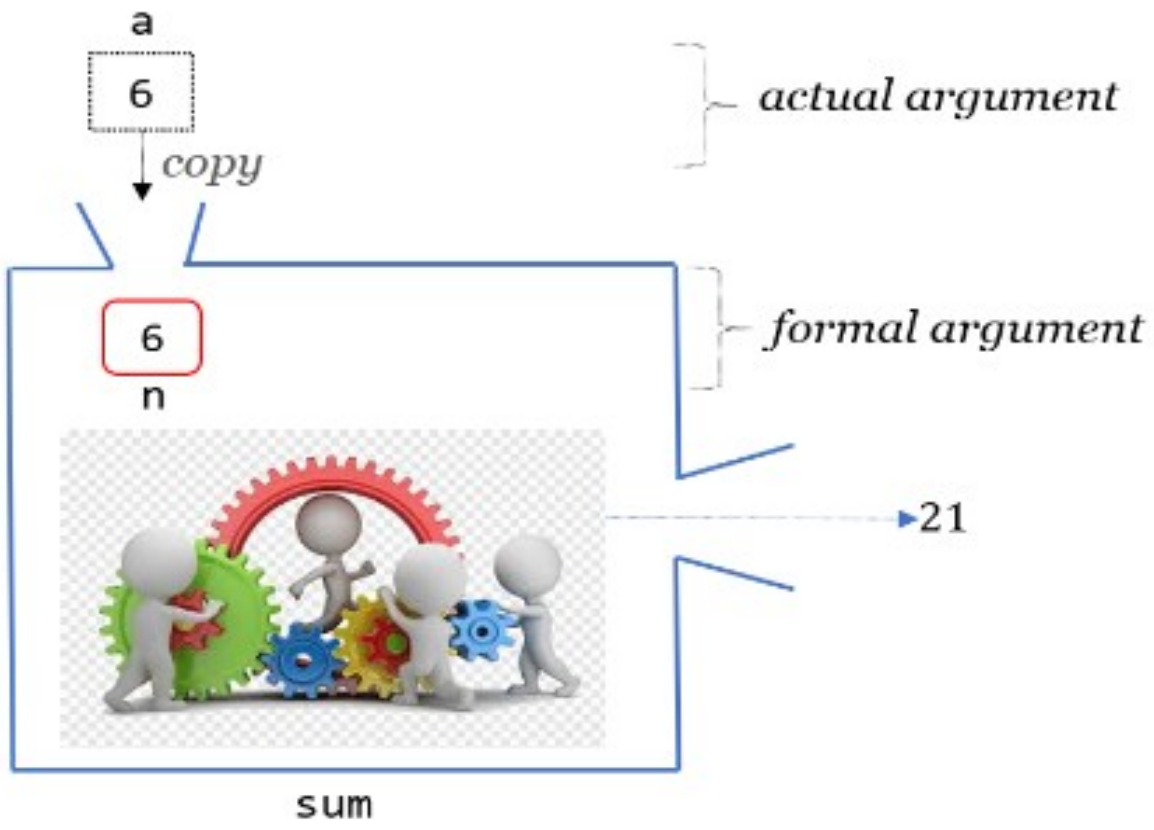
Parametry funkcji i przekazywanie przez wartość

Ponieważ przy przekazywaniu parametrów przez wartość do funkcji przesyłamy tylko wartość liczbową argumentu aktualnego, to **przy wywołaniu funkcji możemy w miejsce argumentów podstawiać: zainicjalizowane wcześniej zmienne albo stałe, lub wyrażenia,** np.:

```
double q = cube(1.2); //wywołanie funkcji
double side = 4;
cout << cube(side);
```

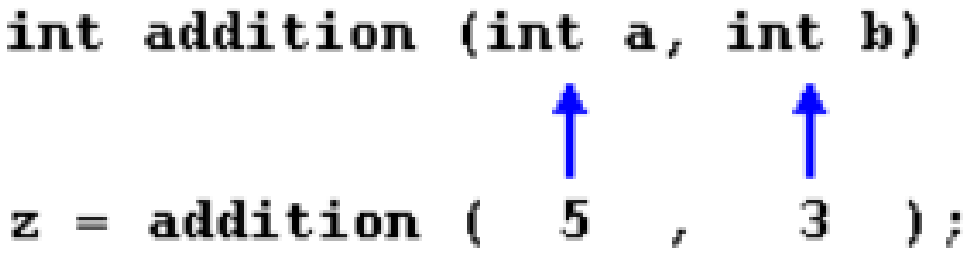
Parametry funkcji i przekazywanie przez wartość

```
// Return the sum of integers from 1 to n
int sum(int n)
{
    return (n * (n + 1)) / 2;
}

int main() {
    int a = 0;
    cout << "Enter a:\n";
    cin >> a;

    if (a > 0) {
        cout << sum( n: a); //call function sum for argument a
    }
}
```



Funkcje- parametry przez wartośc

```
// funkcja suma 2-ch licz całkowitych
#include <iostream>
using namespace std;

int addition (int a, int b)//definicja
{
   return a+b;
}

int main ()
{
    int z;
    z = addition (5,3);//wywołanie funkcji
    cout << "Suma " << z;
}
```



Funkcje- parametry przez wartośc

```
int addition (int a, int b)
                 ↑     ↑
z = addition (   5   ,   3   );
```

Argumenty wywołania funkcji odpowiadają parametrom z deklaracji funkcji. Wywołanie przesyła wartości 5 i 3, do funkcji; odpowiadają one parametrom *a* i *b*.

W momencie wywołania funkcji kontrona jest przekazywana do funkcji *addition*: wykonywanie *main* jest zatrzymane, i będzie kontynuowane po zakończeniu wykonania funkcji *addition* . W momencie wywołania funkcji , wartości obu argumentów (5 i 3)są są kopiowane do zmiennych lokalnych funkcji *int a , int b* .

Następnie, zwracana jest wartość wyrażenie *a*+*b*, tj. 8 dla 5 i 3.
Return kończy funkcję *addition*, program powraca do punktu wywołania funkcji; u nas do funkcji *main* i dodatkowo, ponieważ *addition* zwraca wynik, wywołanie jest wyrażeniem o wartości określonej w instrukcji *return*: w tym konkretnym przypadku 8.

```
int addition (int a, int b)
 ↓8
z = addition (   5   ,   3   );
```

Parametry funkcji i przekazywanie przez wartość - przykład

```
void zwieksz(int formalny)
{
    formalny += 1000; // zwiekszenie liczby o 1000
    cout << "W funkcji modyfikuje arg formalny\n\t"
         << " i teraz arg formalny = "<< formalny
        <<  endl;
}
```

Jak widać, w tej funkcji zwiększa się wartość parametru formalnego funkcji. Funkcję tę wywołujemy na przykład w takim fragmencie programu:

```
int main() {
    int aktu = 2;
    cout << "Przed wywolaniem, aktu = " << aktu
         << endl;
    zwieksz(aktu);
    cout << "Po wywolaniu, aktu = " << aktu << endl;
}
```

Parametry funkcji i przekazywanie przez wartość

Jeżeli wykonamy taki fragment programu, to otrzymamy:

```
Przed wywolaniem, aktu = 2
W funkcji modyfikuje arg formalny
i teraz arg formalny = 1002
Po wywolaniu, aktu = 2
```

Do funkcji przesyłamy tylko wartość liczbową zmiennej `aktu` (czyli argumentu aktualnego).

Wartość ta służy do inicjalizacji parametru formalnego, czyli zmiennej lokalnej tworzonej przez funkcję na stosie. Jest to więc jakby zrobienie kopii w obrębie funkcji.

Funkcja pracuje na tej kopii. Czyli w naszym przykładzie dodanie 1000 nie nastąpiło do komórki pamięci, gdzie tkwi `aktu`, ale do tej zmiennej lokalnej na stosie, gdzie mieści się kopia (o nazwie `formalny`). Po opuszczeniu funkcji ten fragment stosu jest niszczony, znika więc też kopia.

Parametry funkcji i przekazywanie przez wartość - przykład

```
//f-cja oblicza ilość cyfr danej liczby całkowitej
int ileCyfr(int n)
{
    n = abs(n);
    int ile = 0; //licznik cyfr liczby
    //do while bo każda liczba ma przynajmniej
    //1 cyfrę
    do
    {
        ile++; //zwiekszamy licznik
        n /= 10;  //pozbywamy się policzonej cyfry
    }while(n! = 0);
    return ile;
}
```

Parametry funkcji i przekazywanie przez wartość - przykład

```
int main()
{
    cout<<"liczba 834 ma "<< ileCyfr(834)
        <<" cyfr "<< endl;

    int liczba;
    cout << "podaj liczbe calkowita ";

    cin >> liczba;

    cout <<"liczba "<< liczba <<" ma "
         << ileCyfr(liczba)
         <<" cyfr "<< endl;

}
```

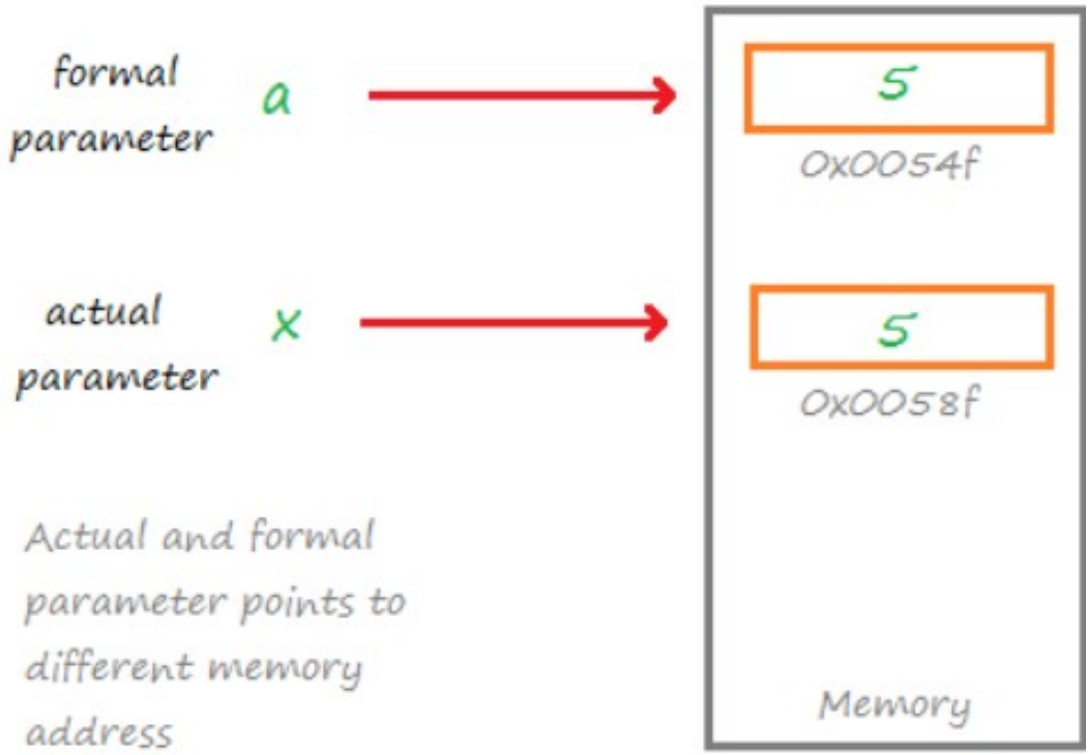
Funkcja obliczająca n!- przykład

```
//n!=1*2*...*n
//0!=1
unsigned long long int silnia(unsigned int n)
{
    unsigned long long s=1;
    for(int k=1; k<=n; k++)
    {
        s *= k;
    }
    return s;
}
int main()
{
    cout <<5<<"!="<< silnia(5) << endl;
}
```

Funkcje przekazywanie parametrów przez wartość

```
void increment(int a)
{
    a++;
}
```

Formal

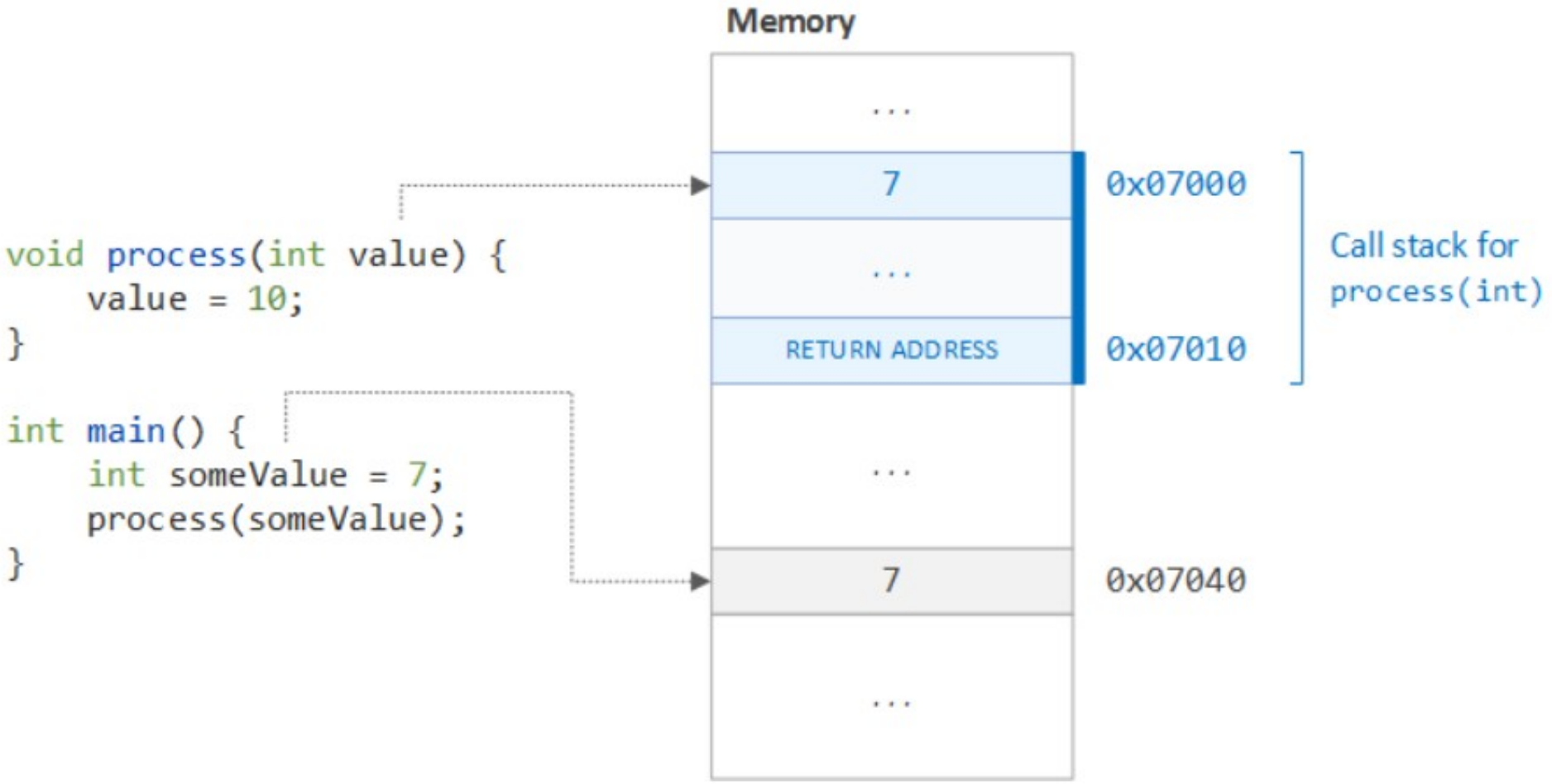



```
int main()
{
    int x = 5;
    increment(x);
}
```

Actual Parameter

Call by Value in C++

Funkcje przekazywanie parametrów przez wartość



Sposoby przekazywania parametrów

Przekazywanie argumentów przez wartość

W momencie wywołania funkcji tworzona jest na jej potrzeby zmienna lokalna o podanej nazwie (parametru funkcji) i do niej jest kopiowana wartość przekazana do funkcji (argumentu).

Po zakończeniu działania funkcji wszystkie zmienne powiązane z parametrami przekazywanymi do funkcji przestają istnieć. Po wyjściu z funkcji znów odwołujemy się do oryginalnej zmiennej, która nie została zmodyfikowana.

Wywołanie funkcji – argumenty: zmienne, stałe, wyrażenie

Przekazywanie argumentów do funkcji przez referencję &

W momencie wywołania wszelkie operacje są wykonywanie na oryginalnych zmiennych. Po zakończeniu działania funkcji odwołujemy się do zmiennej zmodyfikowanej.
Wywołanie funkcji – argumenty: tylko zmienne

Przekazywanie argumentów przez wartość i &

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

//pass by Value- przez wartość

void fun(int x) {
    x = 30;
}

int main() {
    int x = 20;
    fun(x);
    cout << "x = " << x;
    return 0;
}
Output:
x = 20
```

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

//przez referencję

void fun(int &x) {
    x = 30;
}

int main() {
    int x = 20;
    fun(x);
    cout << "x = " << x;
    return 0;
}

Output:
x = 30
```

Przesyłanie argumentów przez referencję

W C++ argumenty możemy przesyłać do funkcji nie tylko przez wartość ale i przez referencję, czyli „przez przezwisko”.

```
#include <iostream>
using namespace std;
void zer(int wart, int &ref);//deklaracja
int main()
{
    int a = 44, b = 77;
    cout << "Przed wywolaniem funkcji: a = " << a
         << ", b = " << b << endl;
    zer(a, b); //wywołanie
    cout << "Po powrocie z funkcji:  a = " << a
         << ", b = " << b << endl;
}
```

Przesyłanie argumentów przez referencję

```
void zer(int wart, int &ref)
{
    cout << "\tW funkcji  przed zerowaniem \n"
         << "\twart = " << wart << ", ref = "
         << ref   << endl;
    wart = 0;
    ref = 0;
    cout << "\tW funkcji  po zerowaniu \n"
         << "\twart = " << wart << ", ref = "
         << ref << endl;
}
```

W rezultacie działania tego programu na ekranie pojawi się:

```
Przed wywolaniem funkcji: a = 44, b = 77
   W funkcji przed zerowaniem
   wart = 44, ref = 77
   W funkcji po zerowaniu
   wart = 0, ref = 0
Po powrocie z funkcji: a = 44, b = 0
```

Przesyłanie argumentów przez referencję

```
void zer(int wart, int &ref);//deklaracja
```

Funkcja `zer`, przyjmuje dwa argumenty: pierwszy z nich jest przesyłany przez wartość, drugi natomiast przez referencję `&`.

Widać, że argument, który funkcja przyjmowała przez wartość, nie został zmodyfikowany. Natomiast zmienna, którą funkcja odebrała przez referencję została zmodyfikowana.

Przesyłanie argumentów przez referencję

```
void zer(int wart, int &ref);//deklaracja
zer(a, b); //wywołanie dla a = 44, b = 77;
```

W tym przypadku do funkcji zamiast liczby 77 (wartość zmiennej b) został wysłany adres zmiennej b w pamięci komputera.

Ten adres funkcja sobie odebrała i (na stosie) stworzyła sobie referencję, czyli komórce pamięci o przysłanym adresie nadała pseudonim (przezwisko, alias) `ref`.

Referencja jest inną nazwą danej zmiennej.

Zatem ta sama komórka, na którą w `main` mówiło się `b`, stała się teraz w funkcji `zer` znana pod przezwiskiem `ref`. Są to dwie różne nazwy, ale określają ten sam obiekt.

Zatem gdy do obiektu o przezwisku `ref` wpisano zero to znaczy, że odbyło się to faktycznie na obiekcie `b`.

Przesyłanie argumentów przez referencję

Po zakończeniu działania funkcji likwiduje się śmieci:

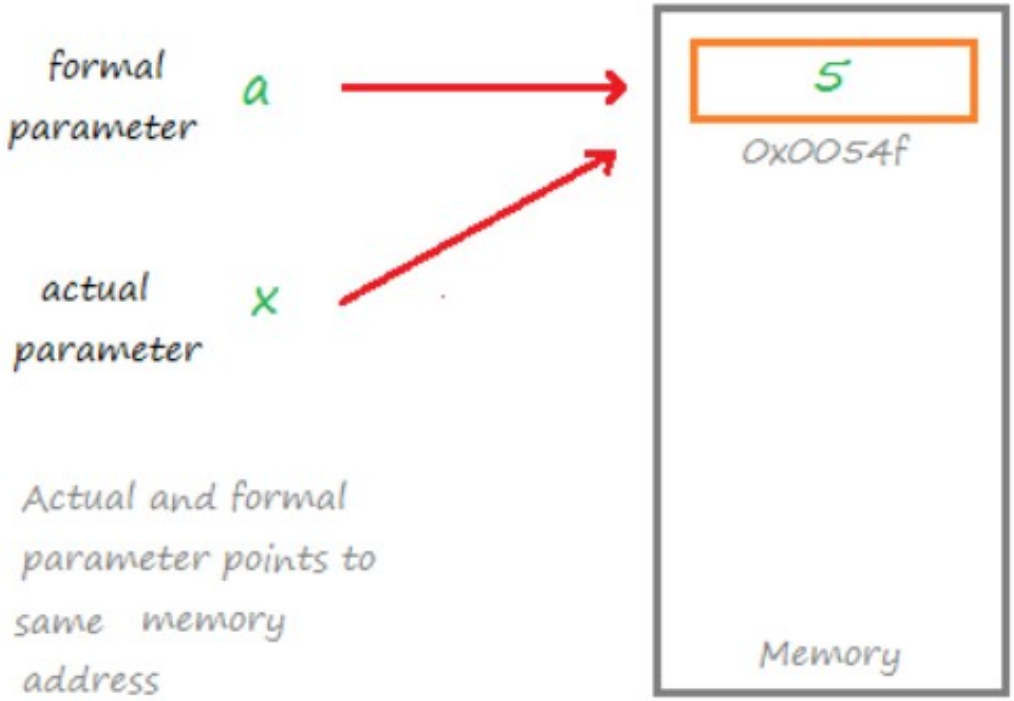
- kopię zmiennej a.
- adres obiektu b, który to obiekt wewnątrz funkcji przezywaliśmy `ref`. Ten adres został zlikwidowany. (My tracimy adres, ale np. funkcja `main` – ma ten adres u siebie zanotowany).

Wniosek: przesłanie argumentów funkcji przez referencję pozwala tej funkcji na modyfikowanie zmiennych znajdujących się poza tą funkcją.

Ten sposób przesyłania stosuje się m.in. do dużych obiektów, gdyż przesłanie ich przez wartość (wymagające zrobienia kopii) powodowałoby spowolnienie wywoływania takiej funkcji. W przypadku gdy taka funkcja jest wywoływana bardzo wiele razy, może to być ważnym czynnikiem.

Jeszcze innym sposobem przesłania argumentu może być **przekazywanie przez wskaźnik**.

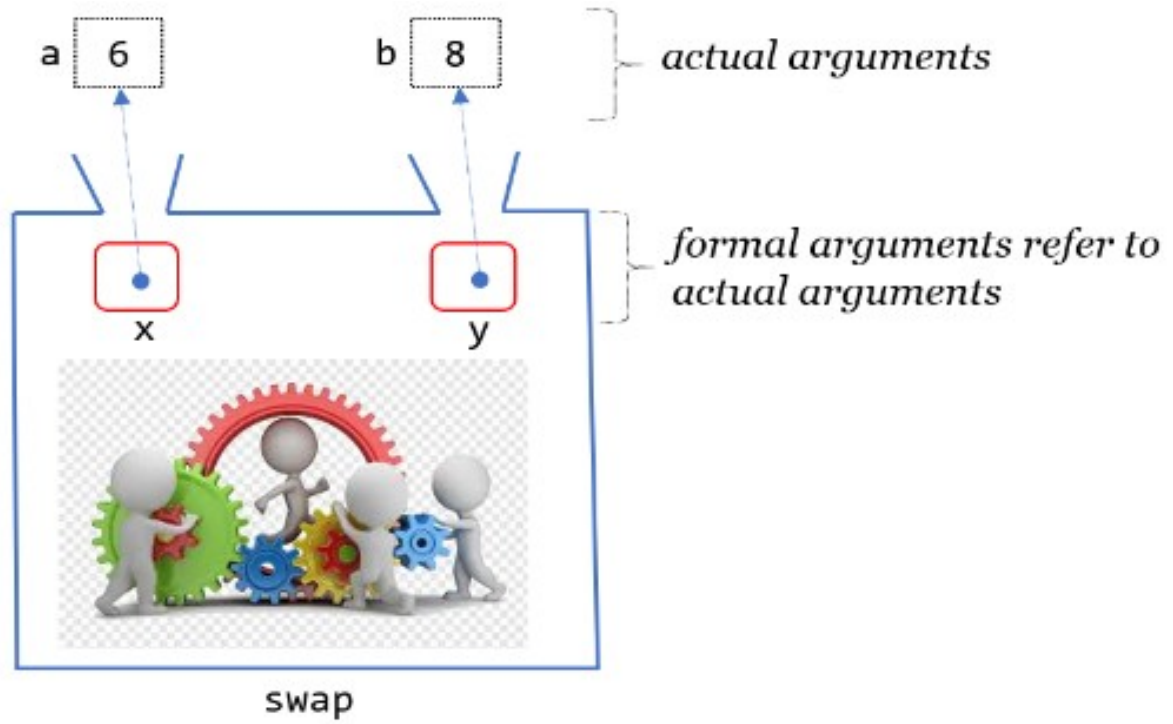
Przesyłanie argumentów przez referencję




```
//Value of x is shared with a
void increment(int &a){
    a++;
}
int main()
{
    int x = 5;
    increment( &: x);
    cout << "Value in Function main: "<< x <<endl;
    return 0;
}
```

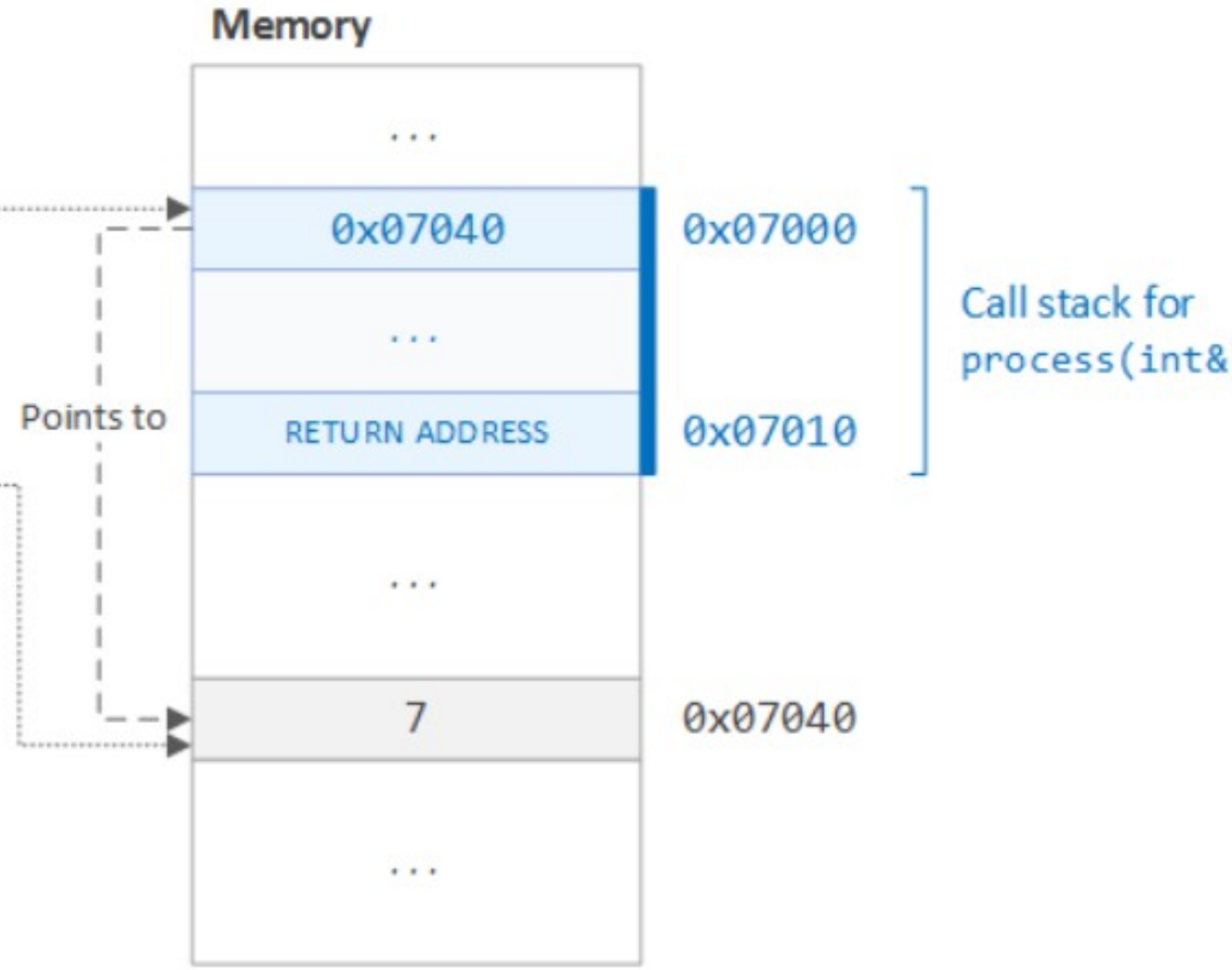
Przesyłanie argumentów przez referencję - przykład

```
void swap(int &x, int &y) {
    int temp = x;

    x = y;
    y = temp;
}

int main() {
    int a = 6;
    int b = 8;

    swap( &: a,  &: b);  // variables a and b are modified:
    // a stores 8 and b stores 6

    cout << "a = " << a
         << "  b = " << b << "\n";
}
```



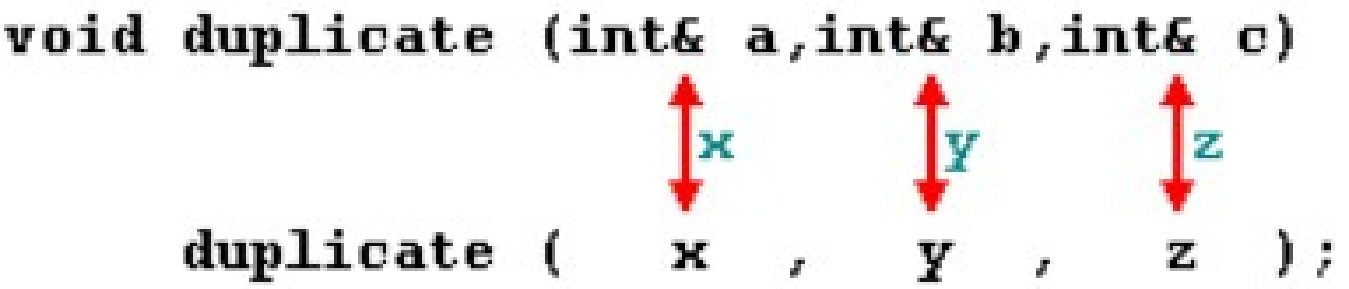
Przesyłanie argumentów przez referencję

```
void process(int& value) {
    value = 10;
}

int main() {
    int someValue = 7;
    process(someValue);
}
```



Przekazywanie parametrów przez &

```
// Funkcja podwaja wartość 3 parametrów
#include <iostream>
using namespace std;

void duplicate (int& a, int & b, int &c)
{
    a*=2;
    b*=2;
    c*=2;
}

int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
    return 0;
}
```



Funkcja zwracająca wiele wartości

return pozwala zwrócić 1 wartość , korzystając z przekazania przez & możemy zwrócić kolejne wartości. Funkcja podziel zwraca wynik dzielenia całkowitego, a resztę przekazujemy przez &

```
int podziel(int licznik, int mian, int &reszta) {
    reszta = licznik % mianownik;
    return licznik / mianownik;
}

int main() {
    int licz = 14;
    int mian = 4;
    int reszta;
    int wynik = podziel(licz, mian, reszta);
    cout << wynik << "*" << mian << "+" << reszta << "="
         << licz  << endl;
    // 3*4+2=14
}
```

Parametry funkcji przekazywanie przez wartość – przykład1

```
#include <iostream>
using namespace std;
//funkcja dla danego numeru dnia tygodnia
//zwraca jego nazwę lub ? w przypadku błędnej wartości

string dzienTygSlownie(int n)
{
   switch(n)

    {   case 1: return "poniedzialek";
        case 2: return "wtorek";
        case 3: return "sroda";
        case 4: return "czwartek";
        case 5: return "piatek";
        case 6: return "sobota";
        case 7: return "niedziela";
        default: return "?";
    }
}

int main() {
    int dn;
    cout<<"podaj numer dnia tygodnia (1-7)"<<endl;
    cin>>dn;
    cout<<"Dzien tygodnia numer "<<dn<<" to "
        <<dzienTygSlownie(dn)<<endl;

}
```