



Podstawy Programowania

dr inż. Tomasz Marciniak

• ??



Wykład 6

- Wstęp do C++
- C a C++

Co nowego w C++



- Nowe mechanizmy wejścia/wyjścia;
- Dowolne rozmieszczenie deklaracji;
- Przekazywanie argumentów przez referencje;
- Przeciążanie funkcji;
- Dynamiczne zarządzanie pamięcią;
- Funkcje *inline*.

Strumienie wejścia/wyjścia

- Standardowe z C:

- stdin
- stdout
- stderr
- Stdlog

- Dodatkowe

- cin
- cout
- cerr
- clog

Mogą zastąpić
printf() i scanf()

Strumienie <iostream>

- część biblioteki standardowej,
- strumień – obiekt, do którego można wysyłać i odbierać dane,
- biblioteka ukrywa sposób zapisu i odczytu,
- programista posługuje się głównie operatorem strumieniowym wejściowym (>>) i wyjściowym (<<),
- operator strumieniowy jest określony dla wszystkich podstawowych typów danych,
- strumień może być wejściowy (klasa podstawowa *istream*) lub wyjściowy (klasa podstawowa *ostream*).

Strumienie <iostream>

- strumień może być rozumiany jako ciąg znaków, który można odczytywać lub do którego można dołączać kolejne znaki,
- operator strumieniowy odpowiada za automatyczną konwersję pomiędzy wartościami zmiennymi i ich reprezentacją znakową,
- strumień może być powiązany z:
 - ekranem (*cout*, *cerr*)
 - klawiaturą (*cin*)
 - plikiem (*fstream*)
 - obszarem pamięci (*stringstream*)

Przykład

```
#include <stdio.h>
#include <conio.h>
```

```
int main ( )
{
    int x ;
    float y ;
    printf("\nPodaj liczbe calkowita : " ) ;
    scanf("%d" ,&x ) ;
    printf("\nPodaj liczbe rzeczywista:" ) ;
    scanf("%f" ,&y ) ;
    printf("\n%3d razy %3.2f = %f " ,x , y ,
        x*y ) ;
    getch();
    return 0 ;
}
```

```
#include <iostream>
#include <conio.h>
```

```
using namespace std;
```

```
int main ( )
{
    int x ;
    float y ;
    cout << "Podaj liczbe calkowita: ";
    cin >> x;
    cout << "Podaj liczbe rzeczywista:";
    cin >> y;
    cout << x << " razy " << y << " = "
        << x*y ;
    getch();
    return 0 ;
}
```


Przykład

```
1 #include <stdio.h>
2 #include <conio.h>
3
4 int main (){
5     int x ;
6     float y ;
7     printf("\nPodaj liczbe calkowita: " ) ;
8     scanf("%d" ,&x ) ;
9     printf("\nPodaj liczbe rzeczywista: " ) ;
10    scanf("%f" ,&y ) ;
11    printf("\n%3d razy %3.2f = %f " ,x , y , x*y ) ;
12    getch();
13    return 0 ;
14 }
```

```
Podaj liczbe calkowita: 3
Podaj liczbe rzeczywista: 5.5
 3 razy 5.50 = 16.500000
```

```
1 #include <iostream>
2 #include <conio.h>
3 using namespace std;
4 int main(){
5     int x;
6     float y;
7     cout << "Podaj liczbe calkowita: ";
8     cin >> x;
9     cout << "Podaj liczbe rzeczywista: ";
10    cin >> y;
11    cout << x << " razy " << y << " = " << x*y ;
12    getch();
13    return 0 ;
14 }
```

```
Podaj liczbe calkowita: 4
Podaj liczbe rzeczywista: 5.5
4 razy 5.5 = 22
```

using namespace std; przestrzeń nazw std

```
1 #include <iostream>
2 //using namespace std;
3 int main(){
4     cout << "Na ekran";
5 }
```

[Error] 'cout' was not declared in this scope

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     cout << "Na ekran";
5 }
```

Na ekran

```
1 #include <iostream>
2 //using namespace std;
3 int main(){
4     std::cout << "Na ekran";
5 }
```

Na ekran

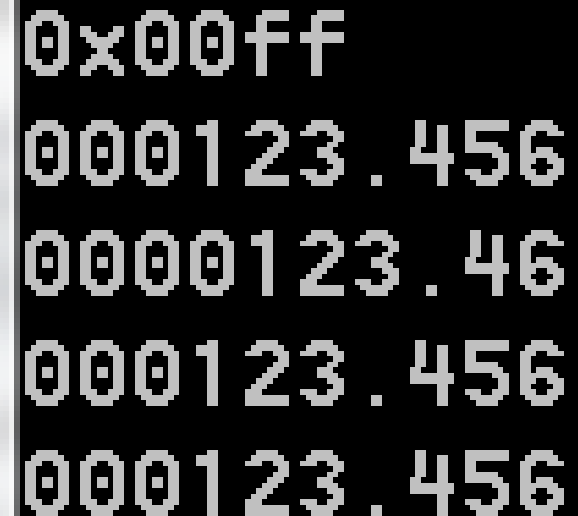
Znaki specjalne

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << "Na ekran" << endl;
5      cout << "1\t2\t3\t4" << endl;
6      cout << "-----" << endl;
7      cout << "\\ - backslash \n";
8      cout << "\\n - znak końca linii" << endl;
9      cout << "\\t - tabulator";
10 }
```

```
Na ekran
1          2          3          4
-----
\ - backslash
\n - znak końca linii
\t - tabulator
```

Formatowanie wydruku

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main (){
5      int a=255;
6      float z=123.456;
7      cout << "0x";
8      cout.width(4);
9      cout.fill('0');
10     cout << hex << a << endl;
11     cout.width(10);
12     cout.fill('0');
13     cout << z << endl;
14     cout.width(10);
15     cout.fill('0');
16     cout.precision(5);
17     cout << z << endl;
18     cout.width(10);
19     cout.fill('0');
20     cout.precision(6);
21     cout << z << endl;
22     cout.width(10);
23     cout.fill('0');
24     cout.precision(7);
25     cout << z << endl;
26 }
```



0x00ff
000123.456
0000123.46
000123.456
000123.456

Poprawność wprowadzonych danych

Mamy do dyspozycji dwie metody strumienia cin:

- `std:cin.good()`,
 - `std::cin.fail()`.
-
- Czyszczenie zawartości strumienia:
 - `std::cin.clear()` – czyści flagi błędów,
 - `std::cin.sync()` – czyści bufor strumienia.

Poprawność wprowadzonych danych

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main(){
5      int x;
6      float y;
7      cout << "Podaj liczbe: ";
8      cin >> x;
9      cout << "Czy dane poprawne ? " << cin.good() << endl;
10     cout << "Czy dane niepoprawne ? " << cin.fail() << endl;
11     getch();
12     return 0 ;
13 }
```

```
Podaj liczbe: 2
Czy dane poprawne ? 1
Czy dane niepoprawne ? 0
```

Poprawność wprowadzonych danych

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main(){
5      int x;
6      float y;
7      cout << "Podaj liczbe: ";
8      cin >> x;
9      cout << "Czy dane poprawne ? " << cin.good() << endl;
10     cout << "Czy dane niepoprawne ? " << cin.fail() << endl;
11     getch();
12     return 0 ;
13 }
```

```
Podaj liczbe: a
Czy dane poprawne ? 0
Czy dane niepoprawne ? 1
```


Typy danych

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      cout << "sizeof(bool) = " << sizeof( bool ) << endl;
5      cout << "sizeof(char) = " << sizeof( char ) << endl;
6      cout << "sizeof(unsigned char) = " << sizeof( unsigned char ) << endl;
7      cout << "sizeof(wchar_t) = " << sizeof( wchar_t ) << endl;
8      cout << "sizeof(short) = " << sizeof( short ) << endl;
9      cout << "sizeof(unsigned short) = " << sizeof( unsigned short ) << endl;
10     cout << "sizeof(int) = " << sizeof( int ) << endl;
11     cout << "sizeof(unsigned int) = " << sizeof( unsigned int ) << endl;
12     cout << "sizeof(long) = " << sizeof( long ) << endl;
13     cout << "sizeof(unsigned long) = " << sizeof( unsigned long ) << endl;
14     cout << "sizeof(long long) = " << sizeof( long long ) << endl;
15     cout << "sizeof(float) = " << sizeof( float ) << endl;
16     cout << "sizeof(double) = " << sizeof( double ) << endl;
17     cout << "sizeof(long double) = " << sizeof( long double ) << endl;
18 }
```


Typy danych

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     cout << "sizeof(bool) = " << sizeof( bool ) << endl;
5     cout << "sizeof(char) = " << sizeof( char ) << endl;
6     cout << "sizeof(unsigned char) = " << sizeof( unsigned char ) << endl;
7     cout << "sizeof(wchar_t) = " << sizeof( wchar_t ) << endl;
8     cout << "sizeof(short) = " << sizeof( short ) << endl;
9     cout << "sizeof(unsigned short) = " << sizeof( unsigned short ) << endl;
10    cout << "sizeof(int) = " << sizeof( int ) << endl;
11    cout << "sizeof(unsigned int) = " << sizeof( unsigned int ) << endl;
12    cout << "sizeof(long) = " << sizeof( long ) << endl;
13    cout << "sizeof(unsigned long) = " << sizeof( unsigned long ) << endl;
14    cout << "sizeof(long long) = " << sizeof( long long ) << endl;
15    cout << "sizeof(float) = " << sizeof( float ) << endl;
16    cout << "sizeof(double) = " << sizeof( double ) << endl;
17    cout << "sizeof(long double) = " << sizeof( long double ) << endl;
18 }
```

Czytanie ze strumienia

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      char text[10];
6      cout << "text: ";
7      cin >> text;           //czyta słowo
8      cout << "słowo-" << text;
9      cout << "text: ";
10     cin.getline(text,10);   //czyta linię bez znaku \n
11     cout << "linia bez-" << text;
12     cout << "text: ";
13     cin.get(text,10);       //czyta linię ze znakiem \n
14     cout << "linia z- " << text;
15
16     string str;
17     cout << "str: ";
18     cin >> str;             //czyta słowo
19     cout << "słowo-" << str;
20     cout << "str: ";
21     getline(cin,str);       //czyta linię i pomija \n
22     cout << "linia bez-" << str;
23     cout << "str: ";
24     getline(cin,str,'4');    //czyta do znaku 4
25     cout << "linia do 4-" << str;
26 }
```

Czytanie ze strumienia

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main(){
5     char text[10];
6     cout << "text: ";
7     cin >> text;
8     cout << "slowo-" << text;
9     cout << "text: ";
10    cin.getline(text,10); //czyta linię bez znaku \n
11    cout << "linia bez-" << text;
12    cout << "text: ";
13    cin.get(text,10); //czyta linię ze znakiem \n
14    cout << "linia z- " << text;
15
16    string str;
17    cout << "str: ";
18    cin >> str; //czyta słowo
19    cout << "slowo-" << str;
20    cout << "str: ";
21    getline(cin,str); //czyta linię i pomija \n
22    cout << "linia bez-" << str;
23    cout << "str: ";
24    getline(cin,str,'4'); //czyta do znaku 4
25    cout << "linia do 4-" << str;
26 }
```

```
text: 12345
slowo-12345text: linia bez-text: 12345
linia z- 12345str: 12345
slowo-12345str: linia bez-str: 12345
linia do 4-123
```

Uwaga na liczbę
wprowadzanych
znaków

Porównywanie łańcuchów

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string str1("CCCCC");
6      string str2("AAAAAA");
7      char text[]="BBBBBBB";
8      if (str1 < str2) cout << "str1 < str2" << endl;
9      if (str1 > str2) cout << "str1 > str2" << endl;
10     if (str1 == str2) cout << "str1 = str2" << endl;
11     if (str1 < text) cout << "str1 < text" << endl;
12     if (str1 > text) cout << "str1 > text" << endl;
13     if (str1 == text) cout << "str1 = text" << endl;
14 }
```

Porównywanie łańcuchów

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string str1("CCCCC");
6      string str2("AAAAAA");
7      char text[]="BBBBBBB";
8      if (str1 < str2) cout << "str1 < str2" << endl;
9      if (str1 > str2) cout << "str1 > str2" << endl;
10     if (str1 == str2) cout << "str1 = str2" << endl;
11     if (str1 < text) cout << "str1 < text" << endl;
12     if (str1 > text) cout << "str1 > text" << endl;
13     if (str1 == text) cout << "str1 = text" << endl;
14 }
```

```
str1 > str2
str1 > text
```

Przeszukiwanie łańcuchów

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string str("to jest przyklad");
6      cout << str << endl;
7      cout << "0123456789012345" << endl;
8      cout << "-----" << endl;
9      cout << str.find("jest") << endl;
10     cout << str.find_first_of("akt") << endl;
11     cout << str.find_last_of("akt") << endl;
12     cout << str.find_first_not_of("akt") << endl;
13     cout << str.find_last_not_of("akt") << endl;
14 }
```

Przeszukiwanie łańcuchów

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main(){
5      string str("to jest przyk
6      cout << str << endl;
7      cout << "0123456789012345
8      cout << "-----
9      cout << str.find("jest") << endl;
10     cout << str.find_first_of("akt") << endl;
11     cout << str.find_last_of("akt") << endl;
12     cout << str.find_first_not_of("akt") << endl;
13     cout << str.find_last_not_of("akt") << endl;
14 }
```

```
to jest przyklad
0123456789012345
-----
3
0
14
1
15
```

Dowolne rozmieszczanie deklaracji

- „C” wymaga aby deklaracje umieszczane były na początku funkcji lub bloku,
- „C++” pozwala na umieszczenie deklaracji w dowolnym miejscu, tak aby deklaracja nastąpiła przed użyciem zmiennej.

```
int a;  
a = 10;  
int b;  
b = 100;  
int y = a * b;
```


Przekazywanie wartości do funkcji

- Przez wartość,
- Przez wskaźnik,
- Przez referencję.

Przekazywanie przez wartość

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int s);
5  int main (){
6      int a=10;    int b=20;    int c=0;
7      suma(a,b,c);
8      cout << "a=" << a << ", b=" << b << ", c=" << c;
9      getch();
10 }
11 void suma(int x, int y, int s){
12     s=x+y;
13 }
```

Przekazywanie przez wartość

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int s);
5  int main (){
6      int a=10;
7      suma(a,b,c);
8      cout << "a="
9      getch();
10 }
11 void suma(int x, int y, int s){
12     s=x+y;
13 }
```

a=10, b=20, c=0

Dlaczego ?

Przekazywanie przez wskaźnik

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int *s);
5  int main (){
6      int a=10;    int b=20;    int c=0;
7      suma(a,b,&c);
8      cout << "a=" << a << ", b=" << b << ", c=" << c;
9      getch();
10 }
11 void suma(int x, int y, int *s){
12     *s=x+y;
13 }
```

The diagram illustrates the concept of passing a pointer to a function. In the `main` function (lines 5-10), three integer variables are declared: `a=10`, `b=20`, and `c=0`. The `suma` function is called with `a`, `b`, and the address of `c` (`&c`) as arguments. The `suma` function (lines 11-13) takes three parameters: two integers (`x` and `y`) and a pointer to an integer (`*s`). Inside the function, the value stored at the memory address pointed to by `*s` is updated to the sum of `x` and `y` (`*s=x+y`). Red arrows highlight the flow of data: one arrow points from `*s` in the function signature to `*s=x+y` in the function body, and another arrow points from `&c` in the `main` function call to the `*s` parameter in the `suma` function signature.

Przekazywanie przez wskaźnik

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int *s);
5  int main (){
6      int a=10;    int b=20;    int c=0;
7      suma(a,b,&c);
8      cout << "a=" << a << ", b=" << b << ", c=" << c;
9      getch();
10 }
11 void suma(int x, int y, int *s){
12     *s=x+y;
13 }
```

Dlaczego ?

a=10, b=20, c=30

Przekazywanie przez referencję

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int &s);
5  int main (){
6      int a=10;    int b=20;    int c=0;
7      suma(a,b,c);
8      cout << "a=" << a << ", b=" << b << ", c=" << c;
9      getch();
10 }
11 void suma(int x, int y, int &s){
12     s=x+y;
13 }
```

Przekazywanie przez referencję

a=10, b=20, c=30

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void suma(int x, int y, int &s);
5  int main (){
6      int a=10;    int b=20;    int c=0;
7      suma(a,b,c);
8      cout << "a=" << a << ", b=" << b << ", c=" << c;
9      getch();
10 }
11 void suma(int x, int y, int &s){
12     s=x+y;
13 }
```

Osiągamy to samo co poprzednio ale w bardziej „ludzkiej postaci” ?

Przeciążanie funkcji

- C++ umożliwia tworzenie większej ilości funkcji o tej samej nazwie. Nazywa się to *przeciążaniem lub przeładowaniem funkcji* (ang. *function overloading*).
- Listy parametrów funkcji przeciążonych muszą się różnić od siebie albo typami parametrów, albo ich ilością, albo jednocześnie typami i ilością.
- Przykłady:

```
int myFunction (int, int);  
int myFunction (long, long);  
int myFunction (long);
```


Przeciążanie funkcji cd.

- Funkcja `myFunction()` jest przeciążona trzema listami parametrów. Pierwsza i druga wersja różnią się od siebie typem parametrów, zaś trzecia wersja różni się od nich ilością parametrów.
- Typy wartości zwracanych przez funkcje przeciążone mogą być takie same lub różne.

Przeciążanie funkcji cd.

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

```
#include <conio.h>
```

```
int funkcja(int x);
```

```
double funkcja(double x);
```

```
int funkcja(int x, int y);
```

```
int main ( )
```

```
{
```

```
    cout << "\nf1=" << funkcja(3);
```

```
    cout << "\nf2=" << funkcja(1.5);
```

```
    cout << "\nf3=" << funkcja(3,5);
```

```
    getch();
```

```
}
```

```
int funkcja(int x){ return x*x; }
```

```
double funkcja(double x){ return x*x; }
```

```
int funkcja(int x, int y){ return x*y; }
```

Przeciążanie funkcji cd.

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

```
#include <conio.h>
```

```
int funkcja(int x);
```



```
double funkcja(double x);
```



```
int funkcja(int x, int y);
```



```
int main ( )
```

```
{
```

```
    cout << "\nf1=" << funkcja(3);
```



```
    cout << "\nf2=" << funkcja(1.5);
```



```
    cout << "\nf3=" << funkcja(3,5);
```



```
    getch();
```

```
}
```

```
int funkcja(int x){ return x*x; }
```

```
double funkcja(double x){ return x*x; }
```

```
int funkcja(int x, int y){ return x*y; }
```

Wzorce - template

- Jeszcze jeden mechanizm pozwalający na operowanie przez funkcję na różnych typach zmiennych.

Definicja funkcji wzorcowej

```
template <class T>
T wieksza ( T zm1 ,T zm2 )
{
    T zm ;
    zm = (zm1 > zm2 ) ? zm1 :zm2 ;
    return zm ;
}
```

Definicja funkcji wzorcowej

Słowo kluczowe



```
template <class T>  
T wieksza ( T zm1 ,T zm2 )  
{  
    T zm ;  
    zm = (zm1 > zm2 ) ? zm1 :zm2 ;  
    return zm ;  
}
```

Definicja funkcji wzorcowej

Lista formalnych typów parametrów

Zawsze w < >

template <class **T**>

T wieksza (T zm1 , T zm2)

{

T zm ;

zm = (zm1 > zm2) ? zm1 :zm2 ;

return zm ;

}

Definicja funkcji wzorcowej

**Każdy parametr poprzedzony
słowem kluczowym *class***

template <class T>

T wieksza (T zm1 , T zm2)

{

T zm ;


zm = (zm1 > zm2) ? zm1 :zm2 ;

return zm ;

}

Definicja funkcji wzorcowej

Typ formalny T (nazwa dowolna, podczas kompilacji zastępowany typem parametrów)



```
template <class T>
T wieksza ( T zm1 ,T zm2 )
{
    T zm ;
    zm = (zm1 > zm2 ) ? zm1 :zm2 ;
    return zm ;
}
```

Dla parametru typu int

T_wieksza (T_zm1 , T_zm2)

{

 T_zm ;

 zm = (zm1 > zm2) ? zm1 :zm2 ;

 return zm ;

}



int

Dla parametru typu int

```
int wieksza ( int zm1 , int zm2 )  
{  
    int zm ;  
    zm = (zm1 > zm2 ) ? zm1 :zm2 ;  
    return zm ;  
}
```

Przykład

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

```
#include <conio.h>
```

```
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
```

```
int main ( )
```

```
{
```

```
    int il1,il2;
```

```
    cout << "Podaj dwie liczby\n";
```

```
    cin >> il1 >> il2;
```

```
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
```

```
    getch();
```

```
}
```

Przykład

Czy wszystko zadziała poprawnie?

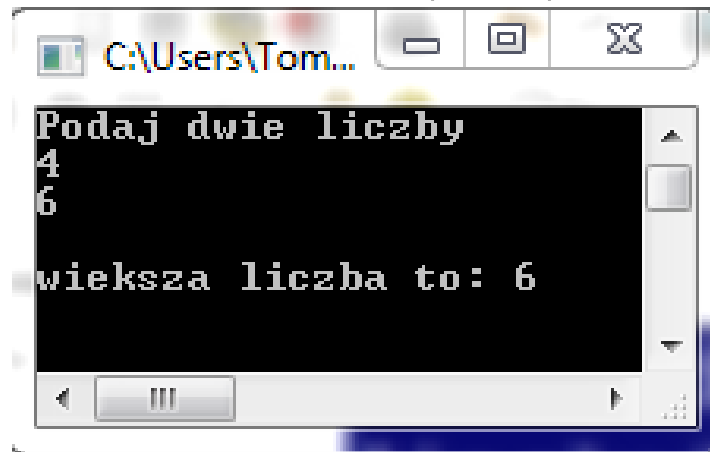
```
#include <iostream>
#include <conio.h>
using namespace std;
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
int main ( )
{
    int il1,il2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    getch();
}
```

Przykład

Czy wszystko zadziała poprawnie?
TAK

```
#include <iostream.h>
#include <conio.h>
```

```
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
int main ( )
{
    int il1,il2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    getch();
}
```



Przykład

A co z tym kodem?

```
#include <iostream>
#include <conio.h>
using namespace std;
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
int main ( )
{
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

Przykład

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

```
#include <conio.h>
```

```
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
```

```
int main ( )
```

```
{
```

```
    int il1,il2;
```

```
    double dl1,dl2;
```

```
    cout << "Podaj dwie liczby\n";
```

```
    cin >> il1 >> il2;
```

```
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
```

```
    cout << "\nPodaj dwie liczby\n";
```

```
    cin >> dl1 >> dl2;
```

```
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
```

```
    getch();
```

```
}
```

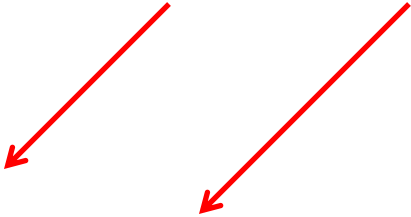
BŁĄD !! – zły parametr funkcji



Jak to możemy poprawić ?

Przykład – poprawiony sposób I

Przeciążanie funkcji



```
#include <iostream>
#include <conio.h>
using namespace std;
int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
double wieksza(double l1, double l2){ return (l1>l2) ? l1 : l2;}
int main ( )
{
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

Przykład – poprawiony sposób 2

```
#include <iostream>
#include <conio.h>
using namespace std;
template <class T>
T wieksza ( T zm1 ,T zm2 ){ return (zm1 > zm2 ) ? zm1 :zm2;}

int main ( ){
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

Przykład – poprawiony sposób 2

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

```
#include <conio.h>
```

```
template <class T>
```

```
T wieksza ( T zm1 ,T zm2 ){ return (zm1 > zm2 ) ? zm1 : zm2;}
```

```
int main ( )
```

```
{
```

```
    int il1,il2;
```

```
    double dl1,dl2;
```

```
    cout << "Podaj dwie liczby\n";
```

```
    cin >> il1 >> il2;
```

```
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
```

```
    cout << "\nPodaj dwie liczby\n";
```

```
    cin >> dl1 >> dl2;
```

```
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
```

```
    getch();
```

```
}
```

Przykład – poprawiony sposób 2

```
#include <iostream.h>
#include <conio.h>
template <class T>
T wieksza ( T zm1 ,T zm2 ){ return (zm1 > zm2 ) ? zm1 :zm2;}
```

```
int main ( )
{
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

Przykład – poprawiony sposób 2

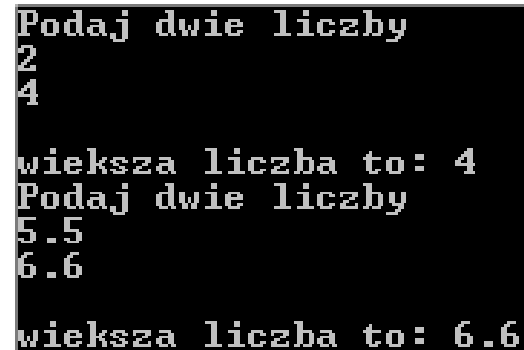
```
#include <iostream.h>
#include <conio.h>
template <class T>
T wieksza ( T zm1 ,T zm2 ){ return (zm1 > zm2 ) ? zm1 :zm2;}

int main ( )
{
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

Przykład – poprawiony sposób 2

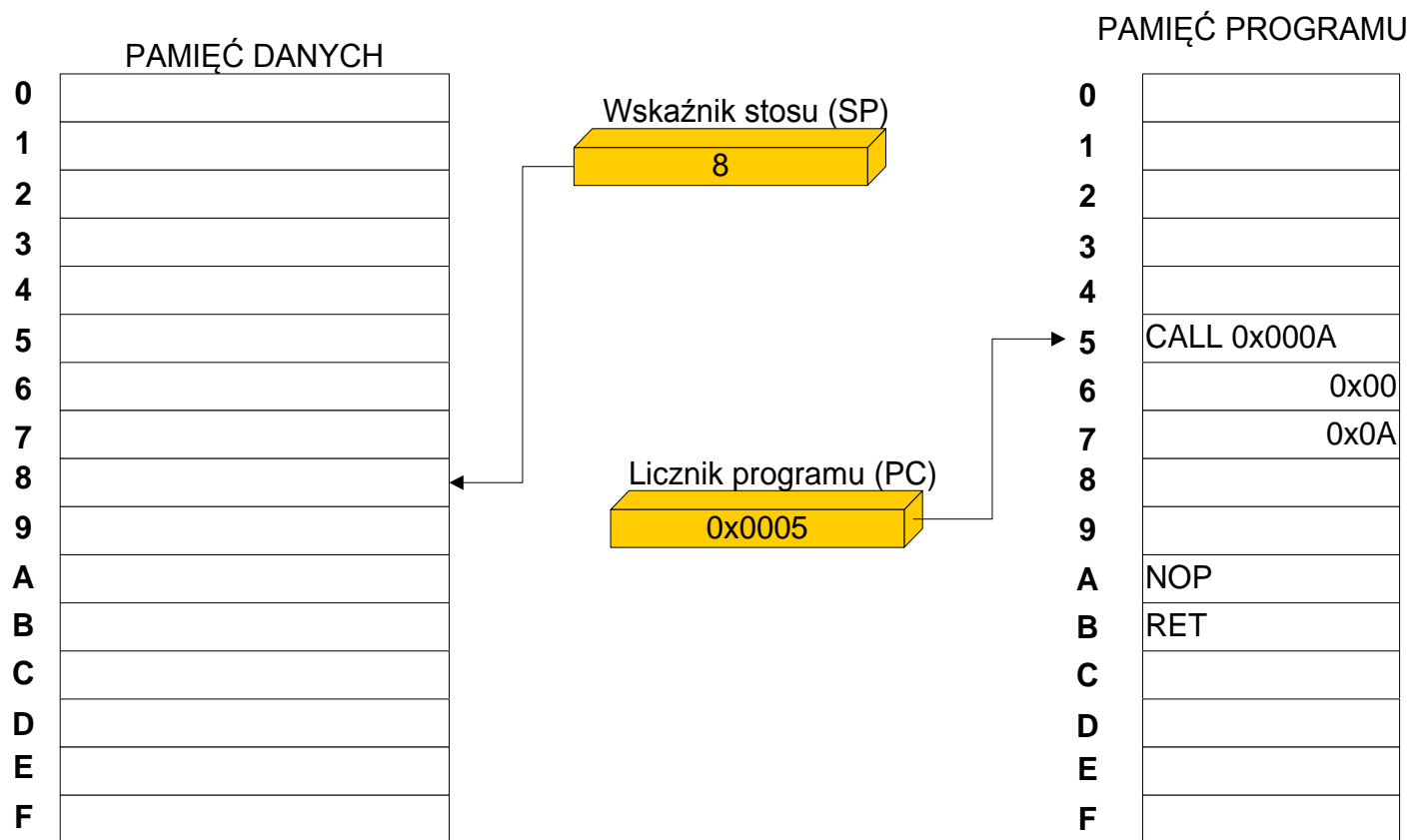
```
#include <iostream.h>
#include <conio.h>
template <class T>
T wieksza ( T zm1 ,T zm2 ){ return (zm1 > zm2 ) ? zm1 :zm2;}

int main ( )
{
    int il1,il2;
    double dl1,dl2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    cout << "\nwieksza liczba to: " << wieksza(il1,il2);
    cout << "\nPodaj dwie liczby\n";
    cin >> dl1 >> dl2;
    cout << "\nwieksza liczba to: " << wieksza(dl1,dl2);
    getch();
}
```

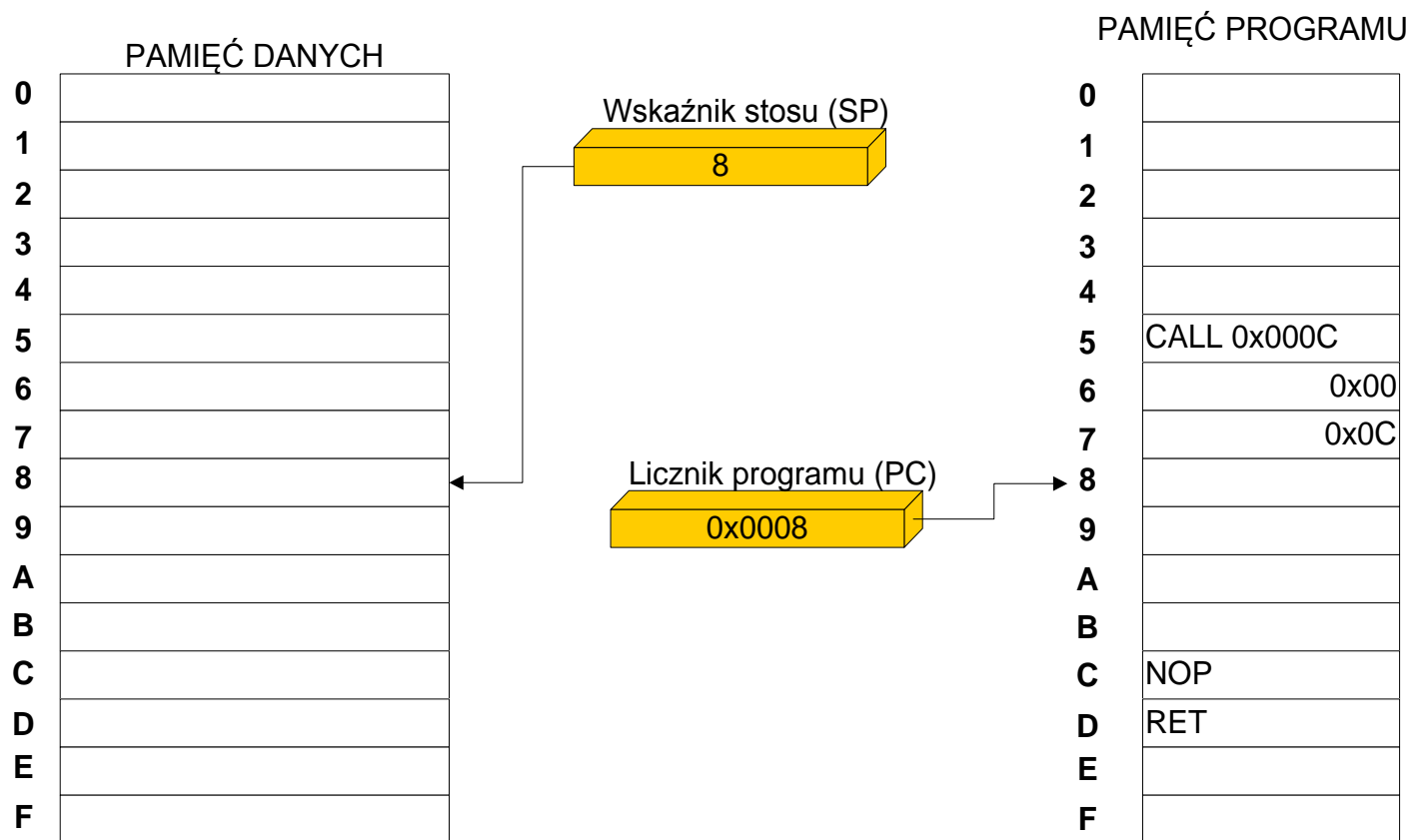


```
Podaj dwie liczby
2
4
wiesza liczba to: 4
Podaj dwie liczby
5.5
6.6
wiesza liczba to: 6.6
```

Wywołanie funkcji

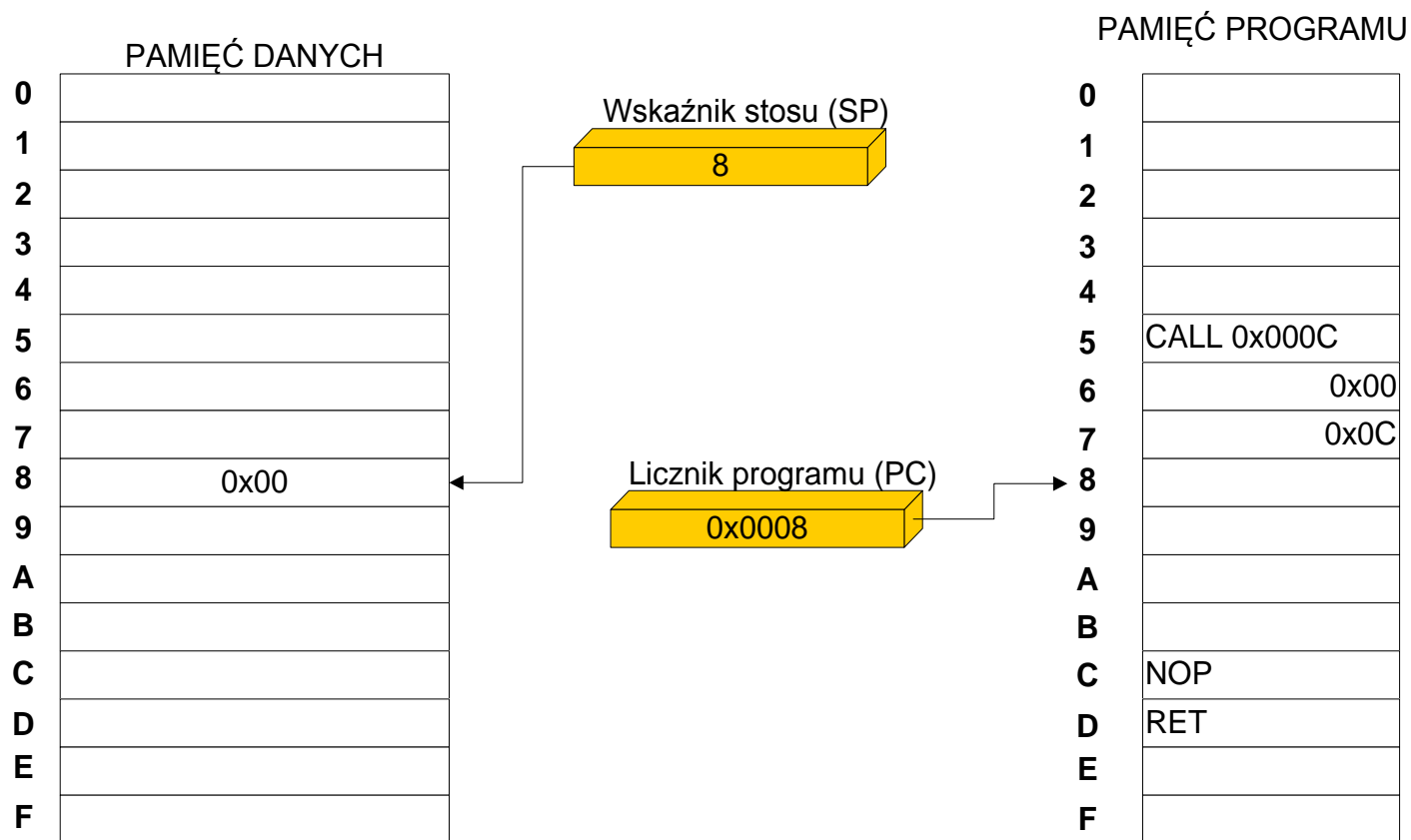


Wywołanie funkcji pobranie rozkazu

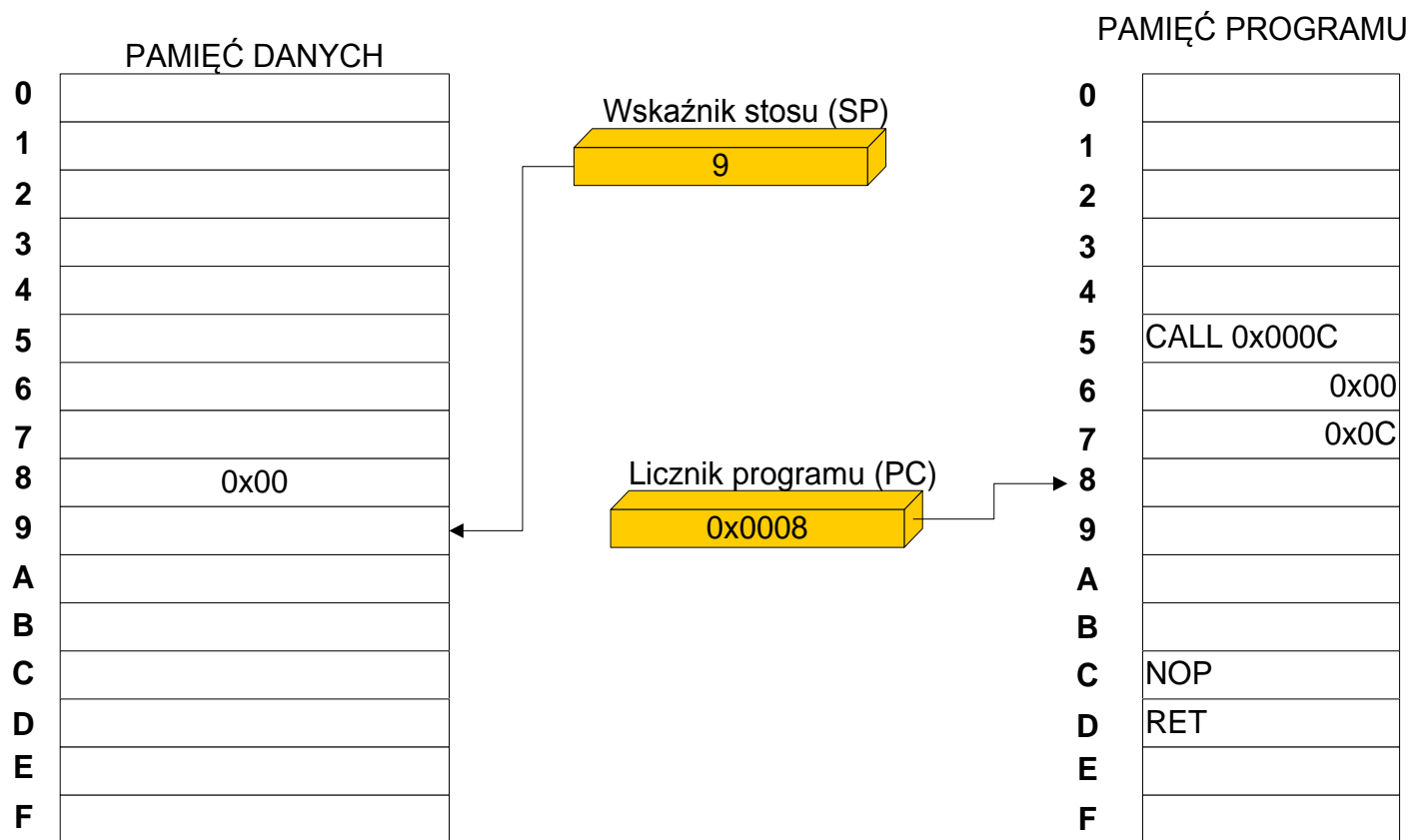


Wywołanie funkcji

odłożenie starszej części adresu

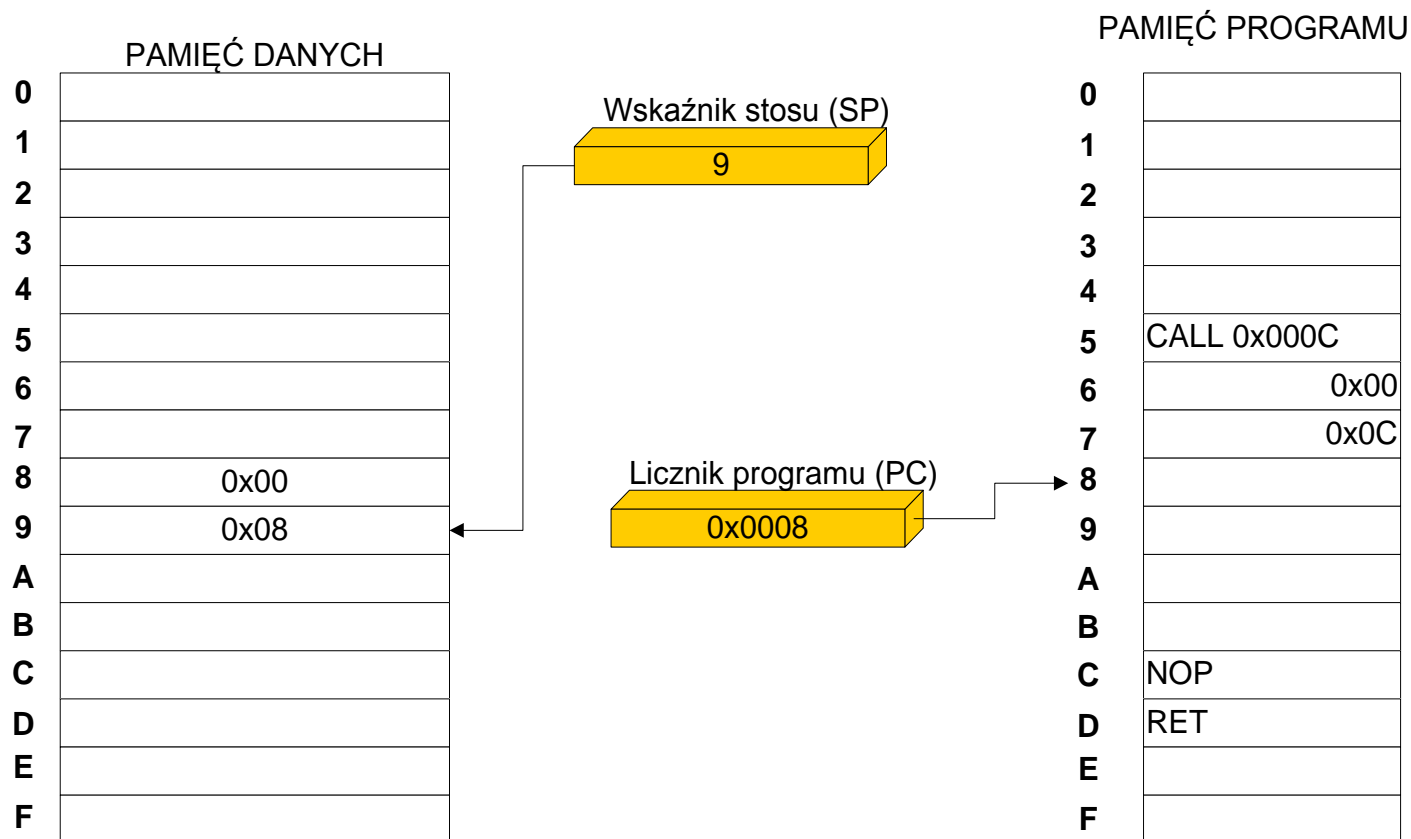


Wywołanie funkcji inkrementacja wskaźnika



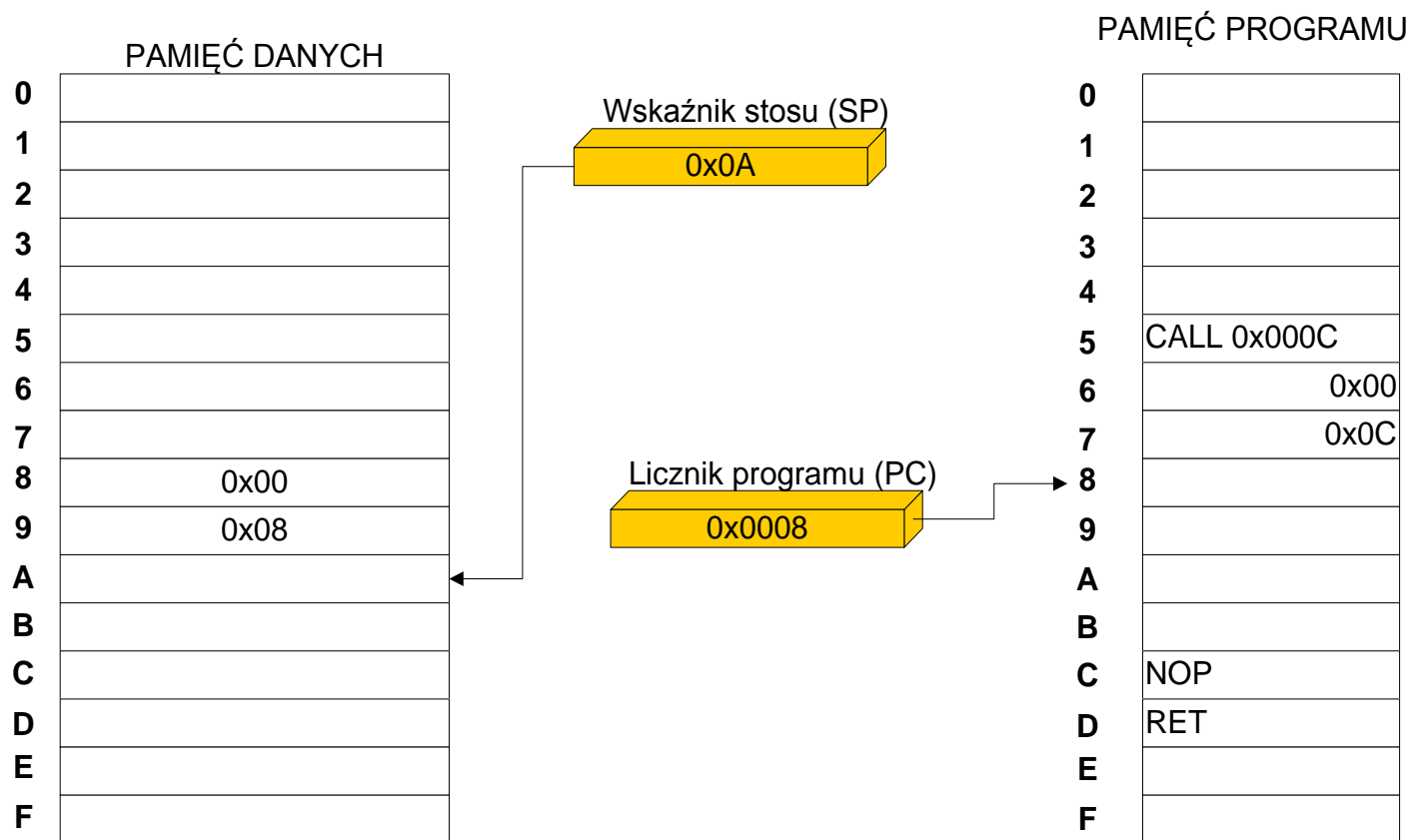
Wywołanie funkcji

odłożenie młodszej części adresu

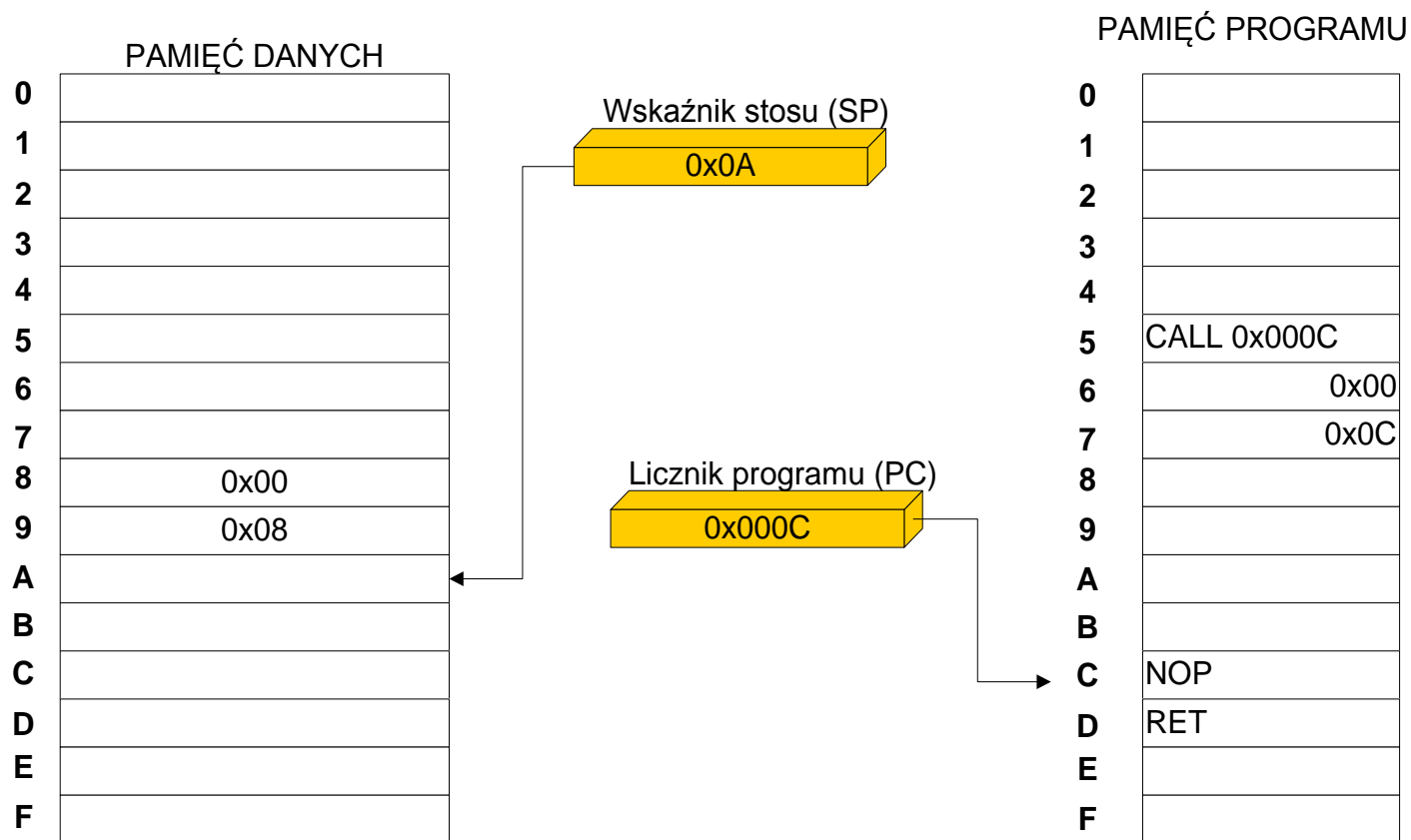


Wywołanie funkcji

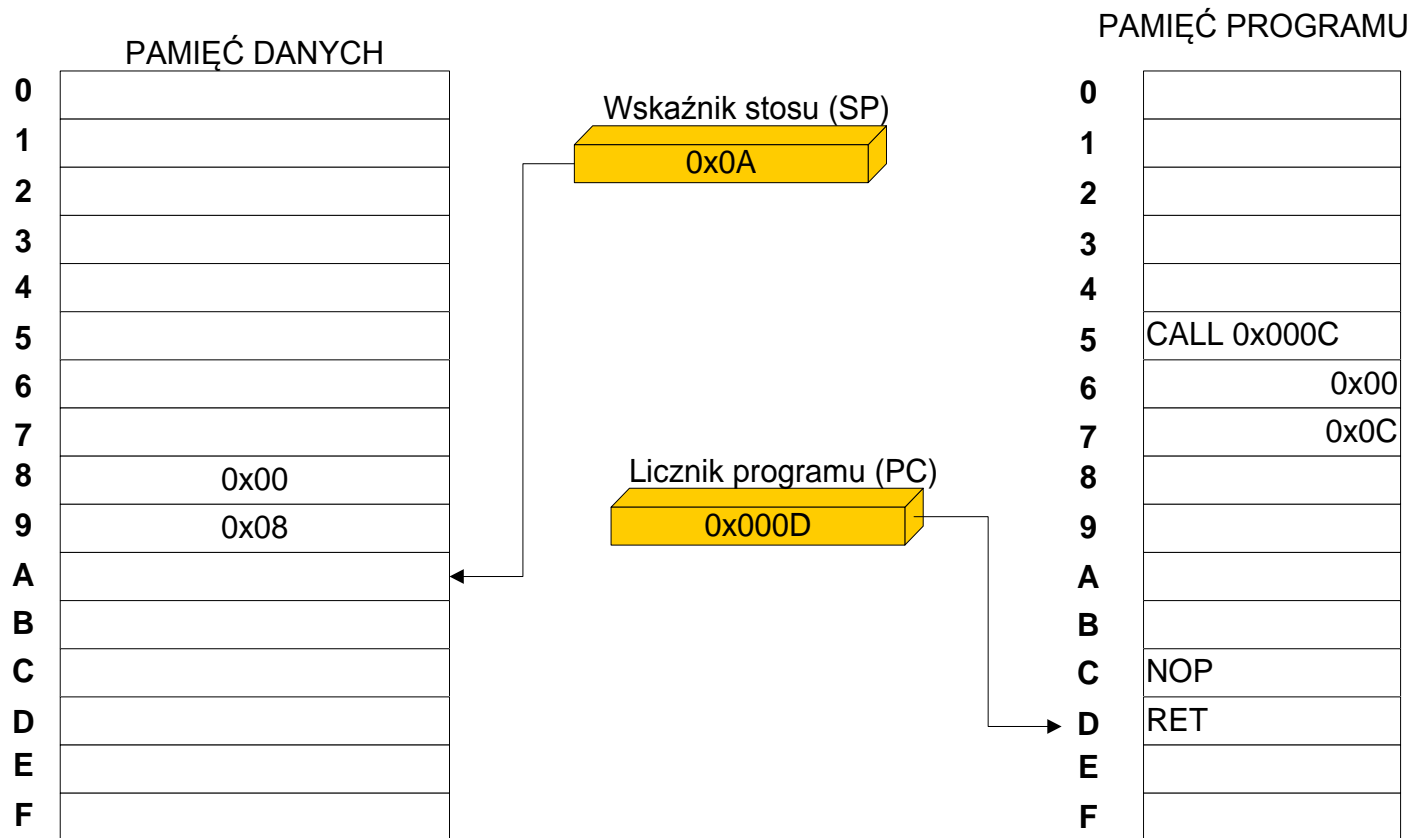
inkrementacja wskaźnika



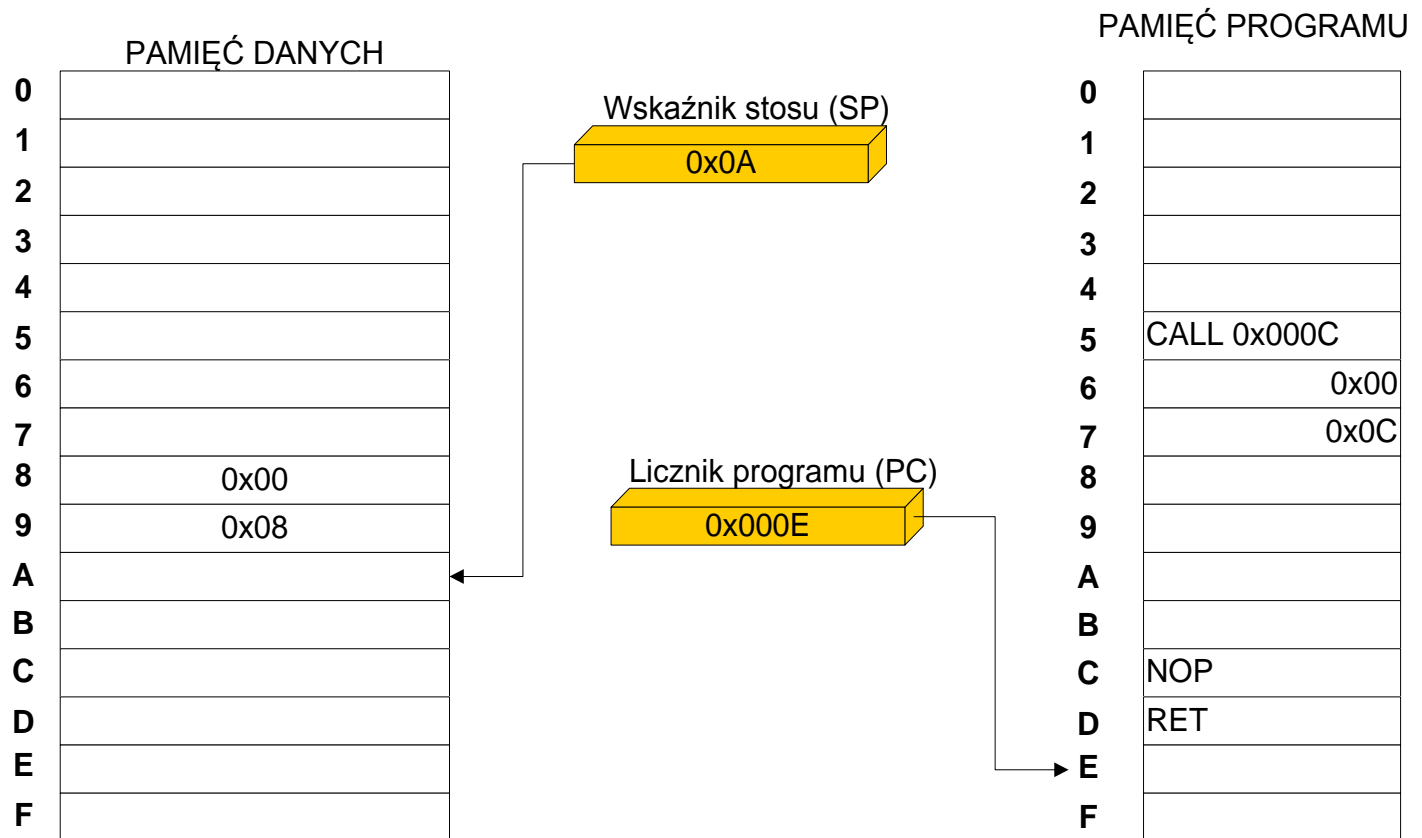
Wywołanie funkcji wykonanie rozkazu



Powrót z funkcji

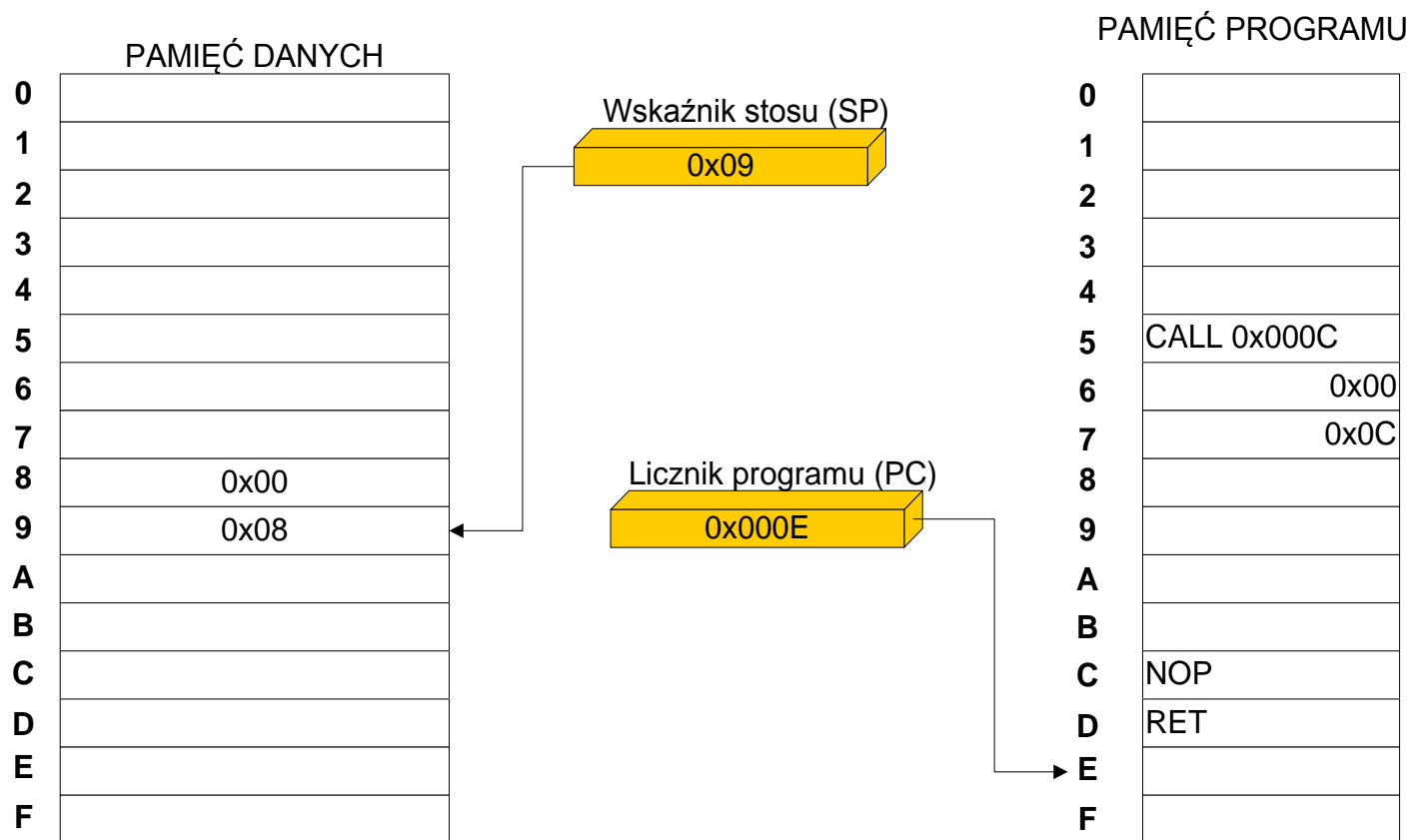


Powrót z funkcji pobranie rozkazu



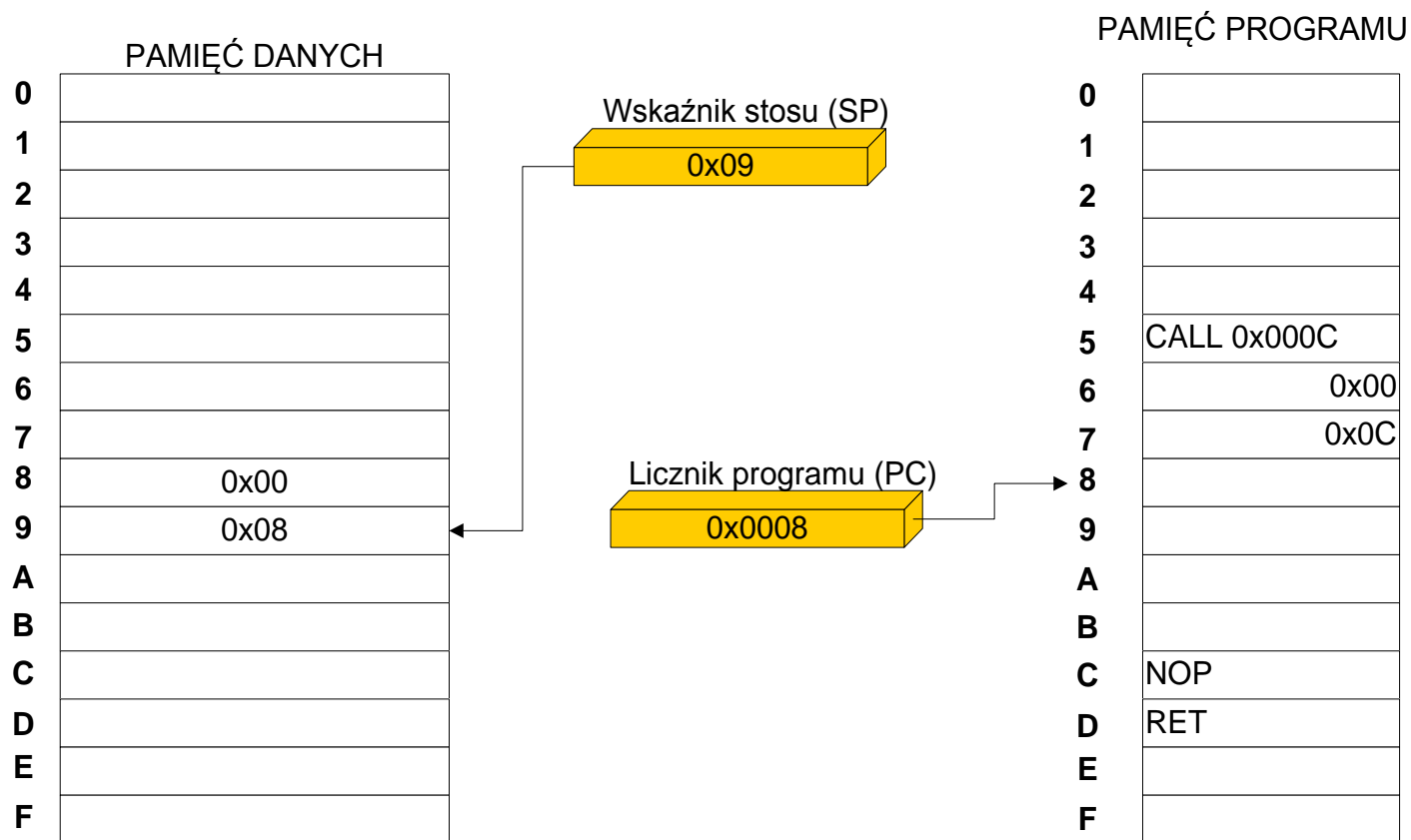
Powrót z funkcji

dekrementacja wskaźnika



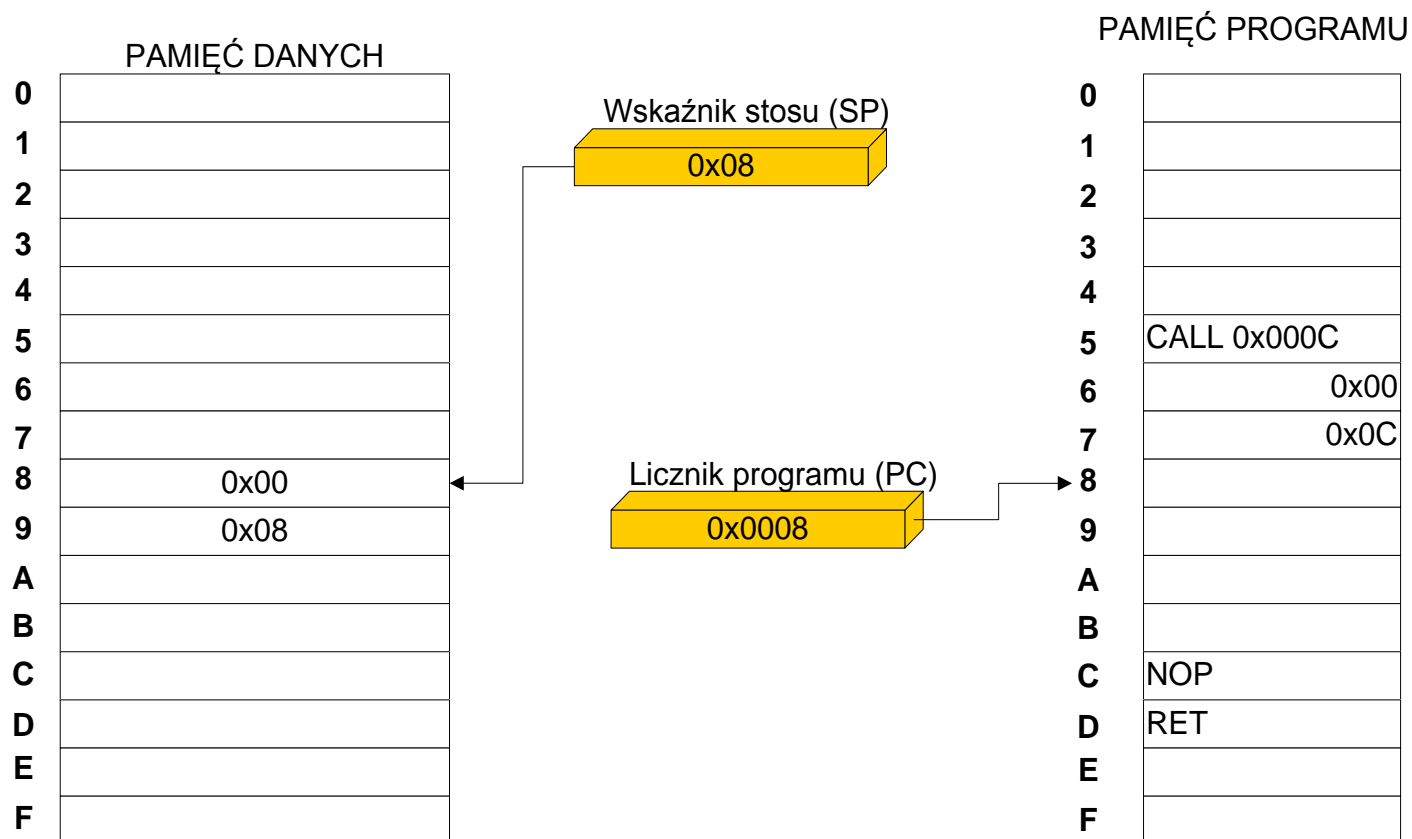
Powrót z funkcji

pobranie młodszej części adresu



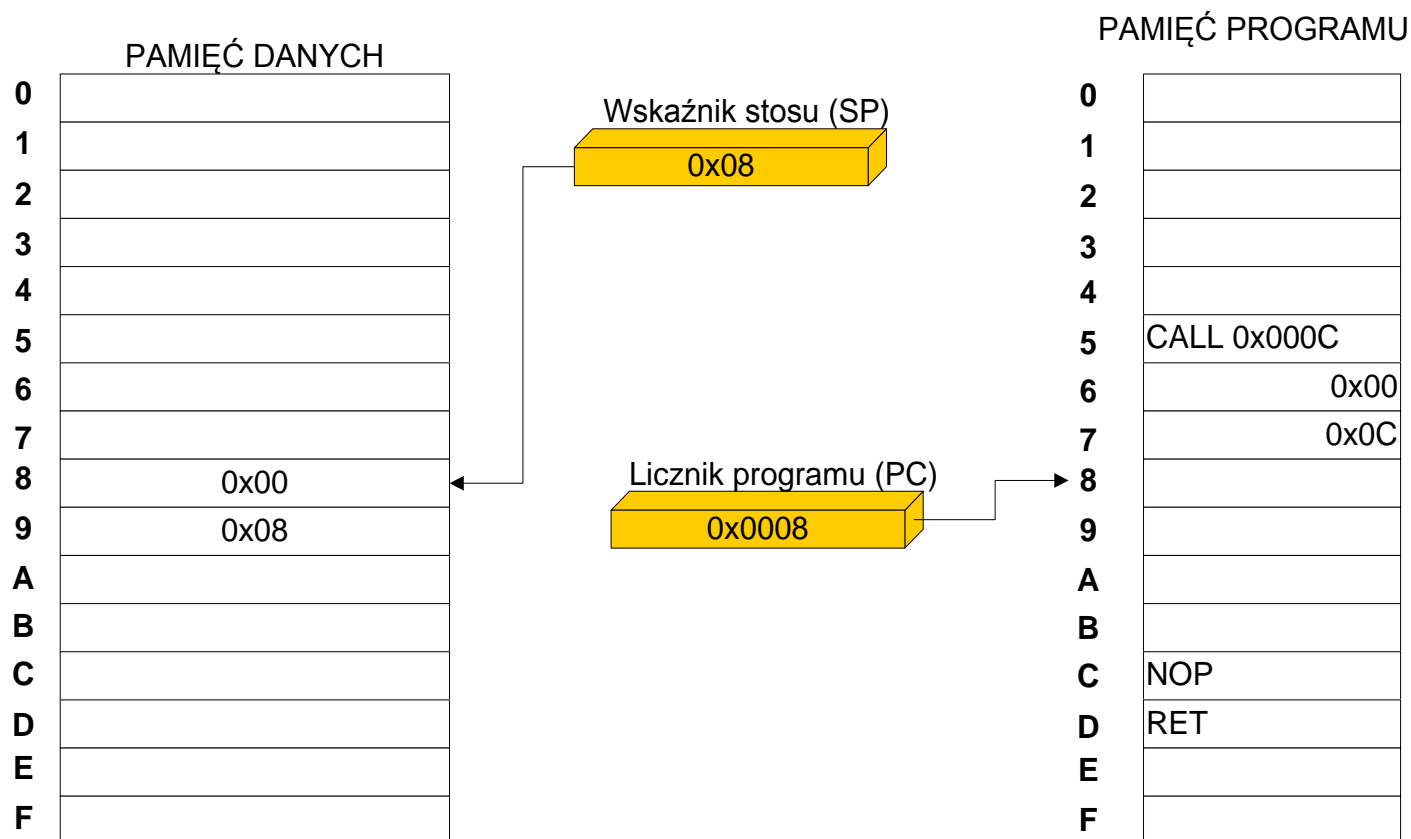
Powrót z funkcji

dekrementacja wskaźnika



Powrót z funkcji

pobranie starszej części adresu



Funkcje inline

- Standardowe wywołanie funkcji powoduje wykonanie dużej ilości operacji, co może spowalniać program;
- Funkcje typu ***inline*** z reguły działają szybciej;
- Kompilator nie tworzy wywołania funkcji ale w miejscu jej wywołania wstawia kod zawarty w funkcji.

Funkcje inline -przykład

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

```
#include <conio.h>
```

```
inline int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
```

```
int main ( )
```

```
{
```

```
    int il1,il2;
```

```
    cout << "Podaj dwie liczby\n";
```

```
    cin >> il1 >> il2;
```

```
    int il=wieksza(il1,il2);
```

```
    cout << "\nwieksza liczba to:" << il;
```


```
    getch();
```

```
}
```

Funkcje inline -przykład

```
#include <iostream.h>
#include <conio.h>
```

Deklaracja funkcji poprzedzona
słowem kluczowym *inline*



```
inline int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
int main ( )
{
    int il1,il2;
    cout << "Podaj dwie liczby\n";
    cin >> il1 >> il2;
    int il=wieksza(il1,il2);
    cout << "\nwieksza liczba to: " << il;
    getch();
}
```

Funkcje inline -przykład

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

```
#include <conio.h>
```

```
inline int wieksza(int l1, int l2){ return (l1>l2) ? l1 : l2;}
```

```
int main ( )
```

```
{
```

```
    int il1,il2;
```

```
    cout << "Podaj dwie liczby\
```

```
    cin >> il1 >> il2;
```

```
    int il=wieksza(il1,il2);
```


```
    cout << "\nwieksza liczba to: " << il;
```

```
    getch();
```

```
}
```

Wywołanie równoważne jest sytuacji jakby w tym miejscu był kod

```
int il= (il1>il2) ? il1 : il2;
```



Dynamiczne zarządzanie pamięcią

- `int a;`
- `char nazw[]="nazwa";`
- `int tab[100];`

Automatyczne
przydzielanie pamięci

Dynamiczny przydział
pamięci

Np. w „C”

- `int* wsk;`
- `wsk = (int*) malloc (100*sizeof(int));`

Przykład

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

```
#include <conio.h>
```

Rozmiar tablicy



```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```

Przykład

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

```
#include <conio.h>
```

```
int main ( )  
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy :";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

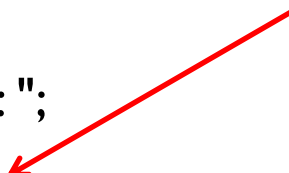
```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```

Zarezerwowanie miejsca w pamięci
dla tablicy o ilości elementów *rozmiar*



Przykład

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

```
#include <conio.h>
```

```
int main ()  
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy :";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

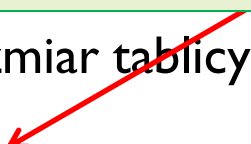
```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```

Wynikiem funkcji *malloc* jest wskaźnik na *void**, czyli musimy pamiętać „zrzutować go na odpowiedni typ”



Przykład

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

```
#include <conio.h>
```

```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```


```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```



Sprawdź czy została przydzielona pamięć

Przykład

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

```
#include <conio.h>
```

```
int main ( )  
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

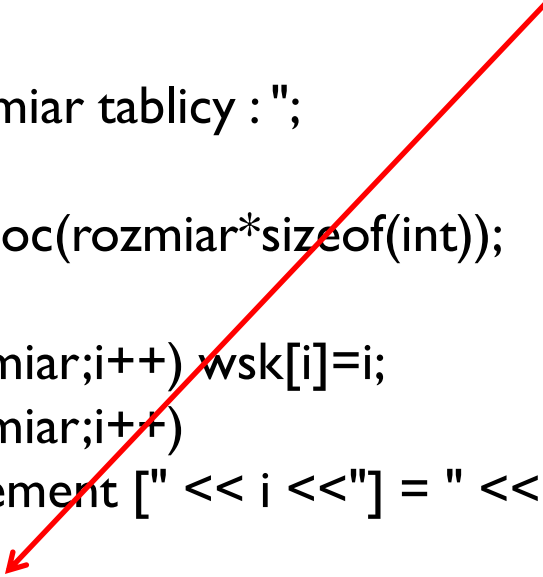
```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```

Jeśli zmienna nie jest już potrzebna
nie zapomnij o zwolnieniu pamięci



Przykład

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

```
#include <conio.h>
```

```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy :";
```

```
    cin >> rozmiar;
```

```
    int *wsk= (int*)malloc(rozmiar*sizeof(int));
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

```
        free(wsk);
```

```
    }
```

```
    getch();
```

```
}
```

```
Podaj rozmiar tablicy : 3
```

```
element [0] = 0
```

```
element [1] = 1
```

```
element [2] = 2
```

Dlaczego stosujemy dynamiczny przydział pamięci?

- Umożliwia optymalne wykorzystanie pamięci, używamy tyle ile potrzebujemy w danej chwili;
- W „C++” mamy dodatkowe funkcje
new
delete

Zmienne dynamiczne

```
int a = 10;
```

```
int *wsk = &a;
```

Tak robiliśmy dotychczas

A co jeśli nie mamy zmiennej statycznej?

```
int *wsk;
```

```
wsk = new int;
```

```
....
```

```
delete wsk;
```

Deklaracja wskaźnika

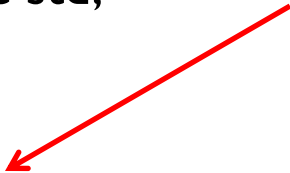
Przydził pamięci dla zmiennej

Teraz można wykonywać operacje na zmiennej

Jeśli zmienna nie jest już potrzebna to ją kasujemy

Przykład

```
#include <iostream>
#include <conio.h>
using namespace std;
int main ( )
{
    int rozmiar;
    cout << "Podaj rozmiar tablicy : ";
    cin >> rozmiar;
    int *wsk= new int[rozmiar];
    if (wsk != NULL){
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
        for (int i=0;i<rozmiar;i++)
            cout << "\nelement [" << i <<"] = " << wsk[i];
        delete [] wsk;
    }
    getch();
}
```



Rozmiar tablicy

Przykład

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

```
#include <conio.h>
```

```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= new int[rozmiar];
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```


```
        delete [] wsk;
```

```
    }
```

```
    getch();
```

```
}
```

Zarezerwowanie miejsca w pamięci dla tablicy o ilości elementów *rozmiar*, Nie musimy jak przy *malloc* rzutować typu *void**



Przykład

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

```
#include <conio.h>
```

```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= new int[rozmiar];
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```


```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

```
        delete [] wsk;
```

```
    }
```

```
    getch();
```

```
}
```



Sprawdź czy została przydzielona pamięć

Przykład

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

```
#include <conio.h>
```

```
int main ( )
```

```
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= new int[rozmiar];
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

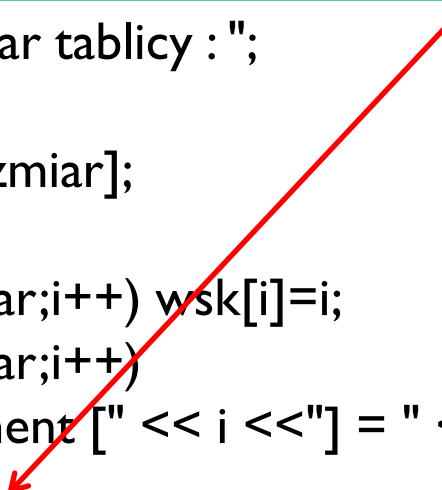
```
            cout << "\nelement [" << i <<"] = " << wsk[i];
```

```
        delete [] wsk;
```

```
    }
```

```
    getch();
```

```
}
```

- Jeśli zmienna nie jest już potrzebna nie zapomnij o zwolnieniu pamięci,
 - Jeśli zmienna była tablicą nie zapomnij o [].
- 

Przykład

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

```
#include <conio.h>
```

```
int main ( )  
{
```

```
    int rozmiar;
```

```
    cout << "Podaj rozmiar tablicy : ";
```

```
    cin >> rozmiar;
```

```
    int *wsk= new int[rozmiar];
```

```
    if (wsk != NULL){
```

```
        for (int i=0;i<rozmiar;i++) wsk[i]=i;
```

```
        for (int i=0;i<rozmiar;i++)
```

```
            cout << "\nelement [" << i << "] = " << wsk[i];
```

```
        delete [] wsk;
```

```
        wsk=NULL;
```

```
    }
```

```
    getch();
```

```
}
```

- Dobrym zwyczajem jest podstawienie za wskaźnik NULL po jego zwolnieniu,
- wywołanie delete jeśli wskaźnik ma wartość NULL nie powoduje błędu.

Nowy typ w C++

- Typ *bool*

Każda inna wartość różna od „0”

- `int zmienna = true;`
- `int zmienna = false;`

Wartość „0”



KONIEC