Programowanie strukturalne - Wykład 4

Wstęp do testowania funkcji

Jak sprawdzać poprawność funkcji?

- Testowanie efektów ubocznych (side effects): Jeśli funkcja wpływa na stan systemu lub zmienia stan obiektów/innych zmiennych, to można przetestować, czy te zmiany są poprawne i oczekiwane. Na przykład, jeśli funkcja zapisuje dane do bazy danych, możesz sprawdzić, czy te dane zostały zapisane w oczekiwany sposób.
- Testowanie wyjątków (exceptions): Można przetestować, czy funkcja rzuca wyjątki (błędy w trakcie wykonania), gdy powinna. W ten sposób można zweryfikować, czy funkcja zachowuje się zgodnie z oczekiwaniami w sytuacjach wyjątkowych. - to dostarcza programownie obiektowe (!!!).

- Testowanie typów zwracanych przez funkcję: Jeśli funkcja zwraca wartość, można przetestować, czy jest ona poprawnego typu. Na przykład, jeśli funkcja powinna zwracać liczbę całkowitą, można sprawdzić, czy zwraca wartość typu `int`. Zwykle powinna być to “najmniejsza grupa typów”.
- Testowanie czy funkcja zwraca wartości niemodyfikujące stanu: Jeśli funkcja nie wpływa na stan systemu lub zmienia stan obiektów/innych zmiennych, można przetestować, czy zwraca poprawne wartości dla różnych argumentów wejściowych.

- Testowanie funkcji za pomocą przypadków brzegowych: Można przetestować, jak funkcja zachowuje się w skrajnych przypadkach, takich jak podanie nieprawidłowych danych wejściowych lub granicznych wartości. To pozwala na upewnienie się, że funkcja działa poprawnie w różnych warunkach.

Pamięć w języku C

Pamięć w języku C

Program w języku C po skompilowaniu wykorzystuje trzy rodzaje pamięci:

- statyczną (globalną) - korzystają z tego zmienne statyczne i globalne.
- automatyczną - zmienne zadeklarowane wewnątrz funkcji
- dynamiczna - alkowana na stercie i może być zwolniona, gdy będzie to konieczne.

Przypadek do analizy:

- Napisz funkcję bez argumentu, która zlicza liczbę swoich wywołań, a następnie ją wyświetla.

Zmienne globalne:

```c
#include <stdio.h>
#include <stdlib.h>

int x=0;

void foo()
{
    x++;
    printf("liczba wywolan: %d\n",x);
}

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

Problem?

```c
#include <stdio.h>
#include <stdlib.h>

int x=0;

void foo()
{
    x++;
    printf("liczba wywolan: %d\n",x);
}

int main()
{
    foo();
    x--;
    foo();
    return 0;
}
```

Zmienne statyczne:

```c
#include <stdio.h>
#include <stdlib.h>

void foo()
{
    static int x=0;
    x++;
    printf("liczba wywolan: %d\n",x);
}

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

Zasięg zmiennych

Rodzaj pamięci	Zasięg	Okres istnienia
globalna	cały plik	cały okres działania aplikacji
statyczna	funkcja, w której została zadeklarowana	cały okres działania aplikacji
automatyczna (lokalna)	funkcja, w której została zadeklarowana	czas wykonywania funkcji
dynamiczna	określony przez wskaźniki odnoszące się do tej pamięci	do momentu zwolnienia pamięci

Wskaźniki

Definicja

Wskaźnik (ang. pointer) to specjalny rodzaj zmiennej, w której zapisany jest adres (innej zmiennej lub funkcji) w pamięci komputera.

Symbol	znaczenie	użycie
`*`	weź wartość `x`	`*x`
`*`	deklaracja wskaźnika do wartości	`int *x;`
`&`	weź adres	`&x`

```c
#include <stdio.h>

 int main (void)
 {
   int liczba = 5;
   printf("Wartość zmiennej: %d\n", liczba );
   printf("Adres zmiennej: %p\n", &liczba );
   printf("Adres zmiennej: %#010x\n", &liczba );
   return 0;
 }
```

```c
#include <stdio.h>

int main ()
{
    int liczba = 5;
    int * wsk = &liczba;
    printf("Adres zmiennej: %p\n", wsk );
    printf("Adres zmiennej przechowujacej wsk.: %p\n", &wsk );
    printf("Wyluskiwanie wskaznika: %d\n", *wsk);
    return 0;
}
```

Przeanalizujmy kod:

```
1  void idzPrawoDol(int x, int y)
2  {
3      x=x+1;
4      y=y-1;
5  }
6
7  int main()
8  {
9      int x=20, y=15;
10     idzPrawoDol(x,y);
11     printf("Aktualna pozycja: [ %d, %d ] \n",x,y);
12     return 0;
13 }
```

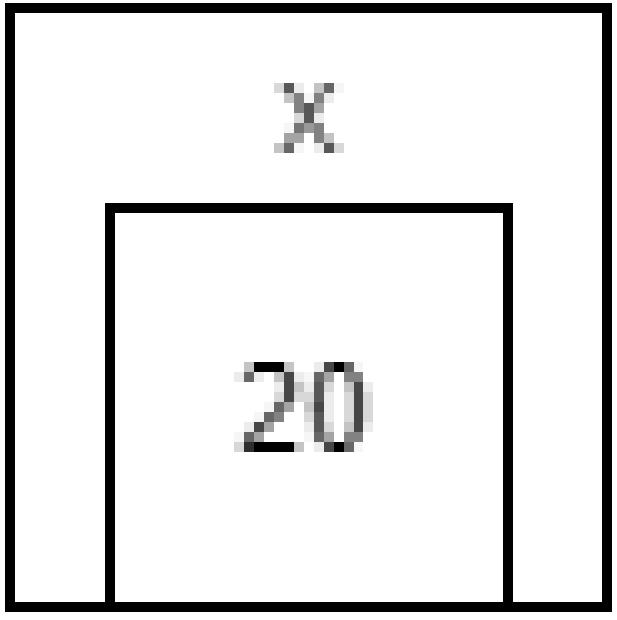
Na wyjściu otrzymujemy:

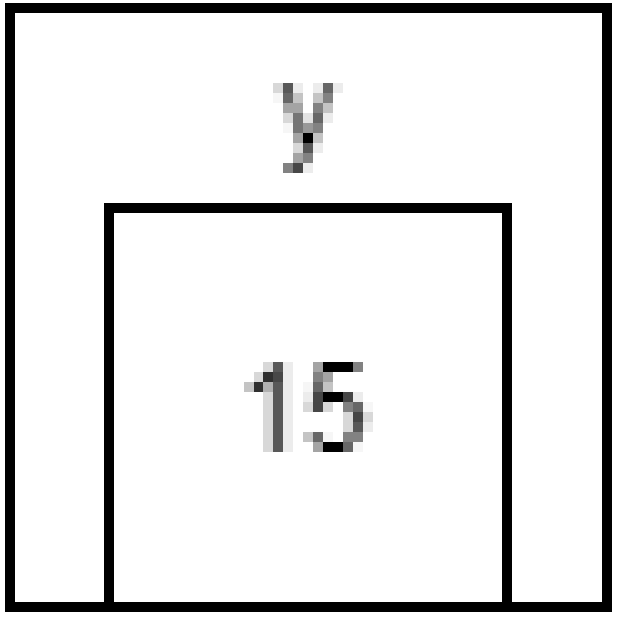
```
Aktualna pozycja: [ 20, 15 ]
```

Dlaczego tak się dzieje?

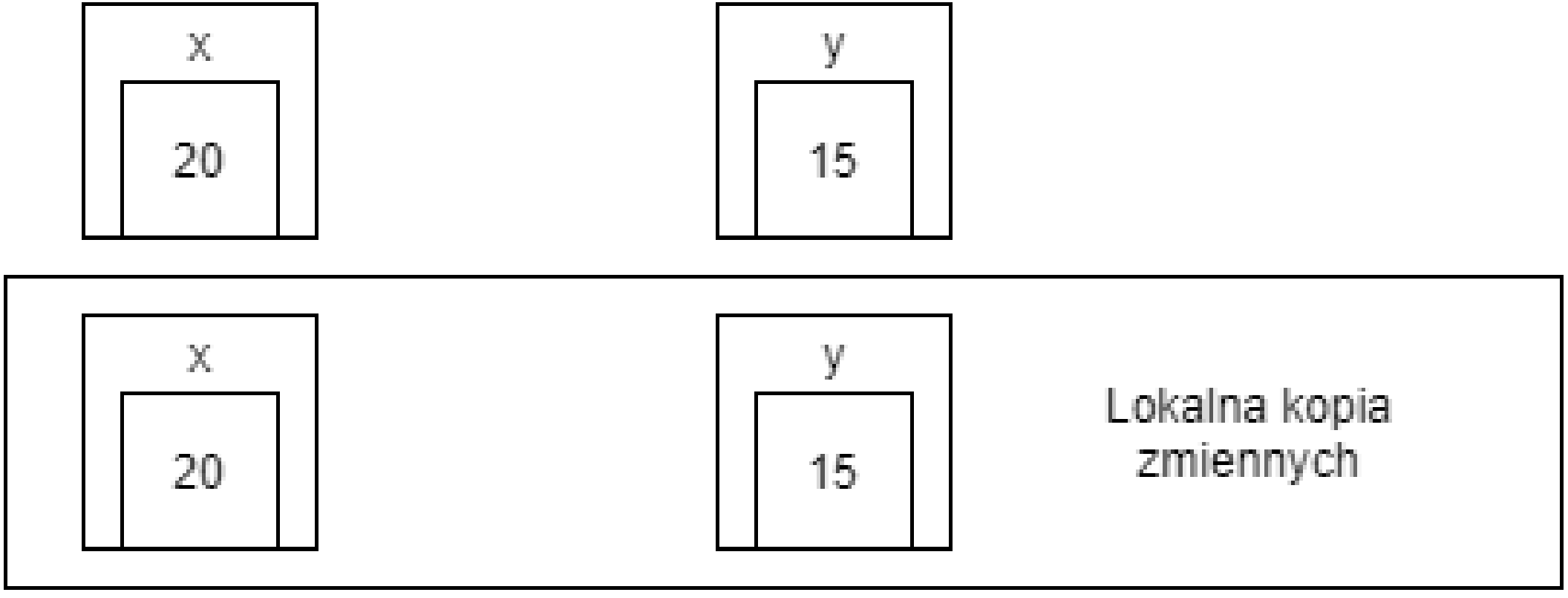
W języku C argumenty przekazywane są przez wartość.

Na początku funkcji `main` mamy dwie zmienne:

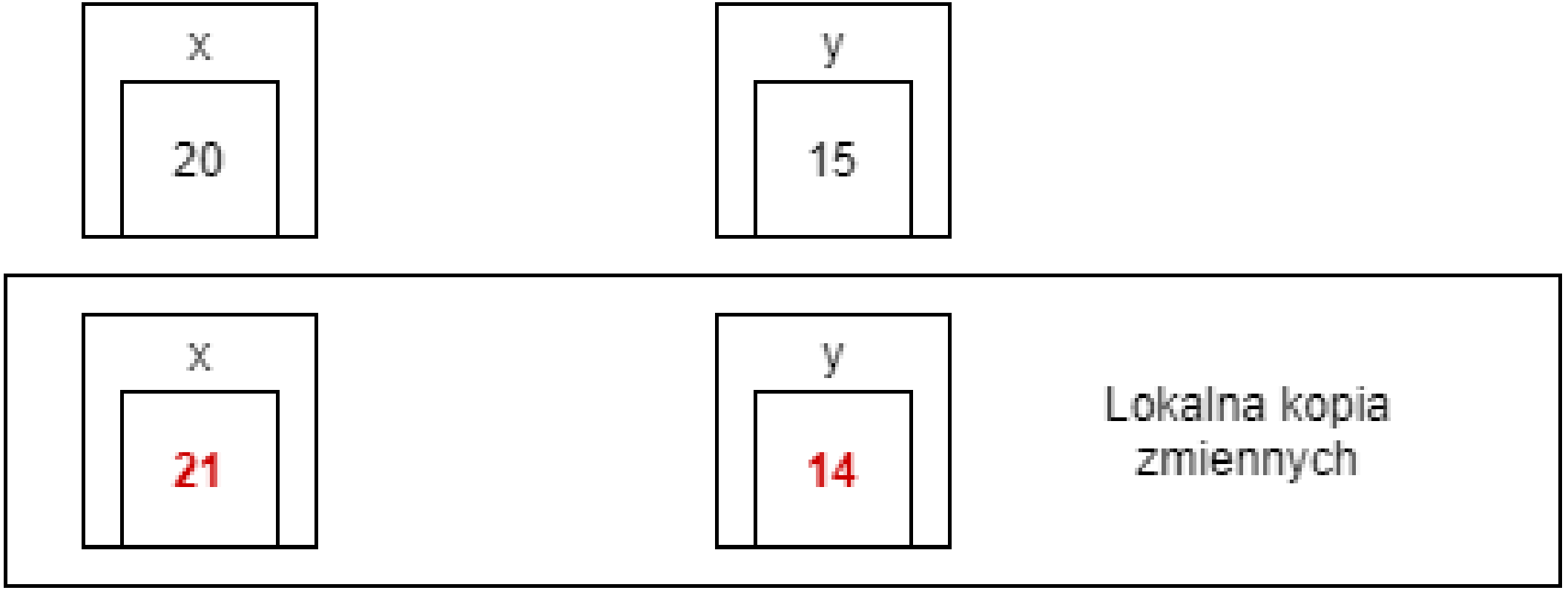




Podczas wykonywania funkcji `idzPrawoDol` następuje kopiowanie wartości do zmiennych lokalnych:



Potem zmiana zmiennych lokalnych:



Jak to naprawić? Użyjemy wskaźników.

```
 1  void idzPrawoDol(int *x, int*y)
 2  {
 3      *x=*x+1;
 4      *y=*y-1;
 5  }
 6
 7  int main()
 8  {
 9      int x=20, y=15;
10      idzPrawoDol(&x,&y);
11      printf("Aktualna pozycja: [ %d, %d ] \n",x,y);
12      return 0;
13  }
```

Na wyjściu będzie:

```
Aktualna pozycja: [ 21, 14 ]
```

Ważne, że poniższe zapisy są równoważne:

```
1  int* x;
2  int * x;
3  int *x;
4  int*x;
```

Zastosowania wskaźników

- tworzenie szybkiego i wydajnego kodu,
- rozwiązywanie w prosty sposób różnego typu problemów,
- obsługa dynamicznej alokacji pamięci,
- tworzenie zwięzłych wyrażeń,
- przekazywanie struktur danych bez ponoszenia kosztów w postaci narzutu,
- ochrona danych przekazywanych do funkcji jako parametry.

Referencje w C?

https://pl.wikipedia.org/wiki/Referencja_(informatyka)

System szesnastkowy

https://pl.wikipedia.org/wiki/Szesnastkowy_system_liczbowy

Pamięć wirtualna

Pamięć wirtualna – mechanizm zarządzania pamięcią komputera zapewniający procesowi wrażenie pracy w jednym, dużym, ciągłym obszarze pamięci operacyjnej podczas, gdy fizycznie może być ona pofragmentowana, nieciągła i częściowo przechowywana na urządzeniach pamięci masowej. Systemy korzystające z tej techniki ułatwiają tworzenie rozbudowanych aplikacji oraz poprawiają wykorzystanie fizycznej pamięci RAM w systemach wielozadaniowych.

ASLR

ASLR (Address Space Layout Randomization) tłumaczony jest jako mechanizm losowego generowania lokalizacji alokacji pamięci wirtualnej.

Wykonywanie czynności przedstawionych na dalszych slajdach związanych z ASLR może oznaczać narażenie komputera na niebezpieczeństwo i podatność na ataki. Wykonanie tych działań nie jest zalecane i robione tylko na własną odpowiedzialność.

Exploit Protection

Funkcja Exploit Protection jest wbudowana w system Windows 10 w celu zabezpieczenia urządzenia przed atakami. Twoje urządzenie jest od razu skonfigurowane za pomocą ustawień ochrony, które są odpowiednie dla większości użytkowników.

Ustawienia funkcji Exploit Protection

Dowiedz się więcej

Wymuś losowe generowanie obrazów (obowiązkowa funkcja ASLR)
Wymuś relokację obrazów, które nie zostały skompilowane z użyciem przełącznika /DYNAMICBASE

Włączone domyślnie

Generuj losowo alokacje pamięci (funkcja ASLR „od dołu do góry")
Generuj losowo lokalizacje alokacji pamięci wirtualnej.

Włączone domyślnie

Funkcja ASLR o wysokiej entropii
Zwiększ zmienność podczas używania ustawienia Generuj losowo alokacje pamięci (funkcja ASLR „od dołu do góry").

Włączone domyślnie

Poprawnie:

```
1  int num=0;
2  int *pi = &num;
```

lub

```
1  int num = 0;
2  int*pi;
3  pi=&num;
```

Wątpliwe dla niektórych kompilatorów

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num = 0;
    int*pi;
    pi= num;
    return 0;
}
```

```c
int num = 0;
int *pi = &num;
printf("Adres pi: %d Wartosc: %d\n",&pi, pi);
printf("Adres pi: %x Wartosc: %x\n",&pi, pi);
printf("Adres pi: %o Wartosc: %o\n",&pi, pi);
printf("Adres pi: %p Wartosc: %p\n",&pi, pi);
```

Wskaźnik na stałą wartość, a stały wskaźnik

Wskaźnik na stałą wartość:

```
1   const int *a;
2   int const * a;
```

Stały wskaźnik:

```
1  int * const b;
```

Stały wskaźnik na stałą wartość:

```
1  int const * const c
```

```c
int i=0;
 const int *a=&i;
 int * const b=&i;
 int const * const c=&i;
 *a = 1;  /* kompilator zaprotestuje */
 *b = 2;  /* ok */
 *c = 3;   /* kompilator zaprotestuje */
 a = b;   /* ok */
 b = a;   /* kompilator zaprotestuje */
 c = a;   /* kompilator zaprotestuje */
```

Funkcja `malloc`

```
1  void *malloc(size_t size);
```

Funkcja służy do dynamicznego rezerwowania miejsca w pamięci. Gdy funkcja zostanie wywołana, w przypadku sukcesu zwróci wskaźnik do nowo zarezerwowanego miejsca w pamięci; w przypadku błędu zwraca wartość NULL.

Funkcja `free`

```
1  void free(void *ptr);
```

Funkcja free zwalnia blok pamięci wskazywany przez ptr wcześniej przydzielony przez `malloc`. Jeżeli ptr ma wartość NULL funkcja nie robi nic.

Rozmiar `int`

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("%Iu\n",sizeof(int)); //Windows
    //printf("%zu\n",sizeof(int)); //linux, os x
    printf("%Iu\n",sizeof(int*));
    printf("%Iu\n",sizeof(int**));
    return 0;
}
```

Wyłuskiwanie (dereferencja) wskaźnika

```
1  int num = 5;
2  int *pi = &num;
3  printf("%p\n",*pi);
```

Wskaźniki na funkcję

```
1  typ_zwracany (*nazwa_wsk)(typ1 arg1, typ2 arg2);
```

```c
#include <stdio.h>

 int suma (int lhs, int rhs)
 {
   return lhs+rhs;
 }

 int main ()
 {
   int (*wsk_suma)(int a, int b);
   wsk_suma = suma;
   printf("4+5=%d\n", wsk_suma(4,5));
   return 0;
 }
```

Jaka różnica?

```
1  int * wsk1();
2  int (*wsk2)();
3  int *(*wsk3)();
```

1. Funkcja zwracająca wskaźnik.
2. Wskaźnik na funkcję.
3. Wskaźnik na funkcję, zwracającą wskaźnik.

Inne typy liczbowe?

Pełna analogia.

Duży błąd merytoryczny - 2 na egzaminie(!)

Dereferencja niezainicjalizowanych wskaźników:

```c
#include <stdio.h>
int main(void)
{
    int *wsk; // niezainicjalizowany wskaznik
    *wsk = 5;
    return 0;
}
```

Bibliografia

- Richard Reese, Wskaźniki w języku C, Wydawnictwo Helion 2014.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki, dostęp online 15.03.2020.
- http://wazniak.mimuw.edu.pl/index.php?title=Wst%C4%99p_do_programowania_w_j%C4%99zyku_C/Wska%C5%BAniki, dostęp online 15.03.2020.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki_-_wi%C4%99cej, dostęp online 15.03.2020.