Język ANSI C

część 8

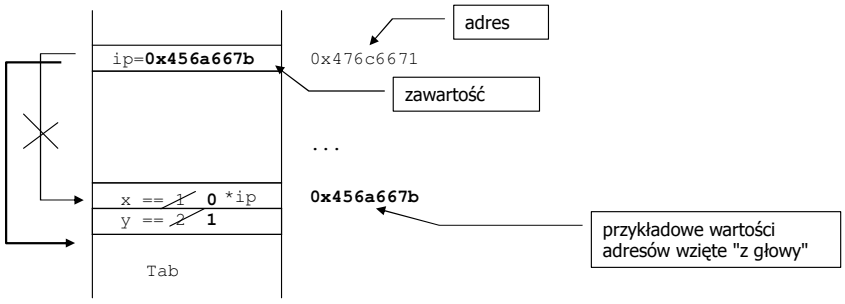
wskaźniki - podstawy

Jarosław Gramacki
Instytut Informatyki i Elektroniki

Podstawowe pojęcia

– najbardziej podstawowe operacje na wskaźnikach

```
int x = 1, y = 2, Tab[10];
int *ip;                  // czy lepiej pisać int* ip; ???

// kilka operacji
ip  = &x;                 // ip wskazuje na x
y   = *ip;                // y ma wartość 1
*ip = 0;                  // x ma wartość 0
ip  = &Tab[0];            // ip wskazuje na element Tab[0]
ip  = Tab;                // j.w.

ip  = &Tab[2];            // wskaźnik na element tablicy o indeksie 2
                          // również: ip = Tab + 2;
```



```
void main()  {
int x = 1, y = 2, Tab[10];
int *ip;  int i;

ip  = &x;                     // ip wskazuje na x
printf("0x%p 0x%p [Dziesietnie: %lu]\n", ip, &x, ip);

y   = *ip;                    // y ma wartość 1
*ip = 0;                      // x ma wartość 0

ip  = &Tab[0];                // ip wskazuje na element Tab[0]
ip  = Tab;                    // j.w.
printf("0x%p 0x%p\n", ip, Tab);

ip  = &Tab[2];
printf("0x%p 0x%p\n\n", ip, &Tab[2]);

for(i=0; i<10; i++)  printf("[%d]: 0x%p \n",  i, &Tab[i]);

ip = NULL;
printf("0x%p" ip);
}
```

```
c:\Temp>Project1.exe
0x0022FF6C 0x0022FF6C [Dziesietnie: 2293612]    // wartość wskaźnika i adres zmiennej
0x0022FF30 0x0022FF30                           // zmiana wartości wskaźnika
0x0022FF38 0x0022FF38

[0]: 0x0022FF30
[1]: 0x0022FF34
[2]: 0x0022FF38
[3]: 0x0022FF3C
[4]: 0x0022FF40
...
0x00000000              // wskaźnik pusty
```


Podstawowe pojęcia

– najbardziej podstawowe operacje na wskaźnikach

```
// zwiększ obiekt wskazywany przez ip o jeden

*ip  = *ip + 1;
*ip += 1;
++*ip;
(*ip)++;
*ip++;            // zwiększa wskaźnik a NIE wskazywany obiekt
```

```
// uawga na priorytet i łączność operatorów

++, *             // łączność prawostronna, ten sam priorytet

*(ip++)           // tak wyrażenie *ip++ opracowuje kompilator
                  // (nawiasy wymuszone przez kompilator)
```

Wskaźniki jako argumenty funkcji

```
// przekazywanie przez wartość

#include <stdio.h>

void Get2Int (int, int);

int main (void)
{
 int x = 4, y = 5;

 Get2Int (x, y);
 printf("First: %d, Second: %d\n, x, y);            // 4, 5
 return 0;
}

void Get2Int (int xx, int yy)
{
  xx = 10;
  yy = 20;
  printf("First: %d, Second: %d\n, xx, yy);         // 10, 20
}
```


Wskaźniki jako argumenty funkcji

```
// przekazywanie przez zmienną (przez wskaźnik, przez adres)

#include <stdio.h>
void Get2Int (int*, int*);

int main (void)
{
 int x = 4, y = 5;

 Get2Int (&x, &y);
 printf("First: %d, Second: %d\n, x, y);          // 5, -5

 return 0; }

void Get2Int (int *px, int *py)  {
  *px = *px + 6 - *py;
  *py = *py - 6 - *px
   printf("First: %d, Second: %d\n, *px, *py);    // 5, -5
}
```

- w wywołaniu `Get2Int (&x, &y);` przekazano adresy zmiennych `x i y` gdzie zostaną umieszczone wyniki działania funkcji `Get2Int()`
- przekazanie wskaźnika do funkcji pozwala na ingerencję w zawartość zmiennych utworzonych poza funkcją.
- zmienne `x` i `y` zostały (poprzez ich zadeklarowanie) utworzone i przydzielono im pamięć w funkcji `main()`

Zwracanie z funkcji wyników obliczeń

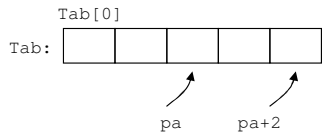
```
// zwracanie z funkcji jednej wartości

#include <stdio.h>

int Get2Int (int, int);            // wynik jako wartość funkcyjna

int main (void)
{
 int x = 4, y = 5, result;

 result = Get2Int (x, y);
 printf("Result: %d\n, result);

 return 0;
}

int Get2Int (int x, int y);
{
  int result;

  result = x + y + 1;
  return (result);
}
```


Zwracanie z funkcji wyników obliczeń

```
// zwracanie z funkcji więcej niż jednej wartości
// użycie np. 3 x return ???

int Get2Int (int, int, int*, int*);// status jako wartość funkcyjna
                                   // wyniki przez nagłówek funkcji

int main (void)
{
 int x = 4, y = 5, result1, result2, status;

 status = Get2Int (x, y, &result1, &result2);
 printf("Results: %d %d\n, result1, result2);

 return 0;
}
                    ↙   ↙       ↗       ↗
int Get2Int (int x, int y, int *r1, int *r2);
{
  int status ;

  *r1 = x + y;
  *r2 = x - y;
  if (*r1 + *r2) > 100 status = 1; else status = 2;

  return (status);
}
```

Wskaźniki i tablice

```
int Tab[5];                          // int Tab[]; - Błąd !!!
int Tab2[]    = {1, 2, 3};           // deklaracja i inicjalizacja
int Tab3[100] = {1, 2, 3};           // ???
int *pa;                             // po alokacji pamięci może
                                     // "być" tablicą 5-cio elementową

pa = &Tab[0];     // lub pa = Tab;
pa = &Tab[2];     //     pa = Tab[2]. Błąd !!!

// teraz do elementów Tab możemy odwoływać się z użyciem pa
MyVar = *pa;      // trzeci element Tab
MyVar = *pa++;    // czwarty element Tab
```




```
pa = Tab;
Tab[2] = 7;       // wpis do elementu o indeksie 2 wartości 7
*(pa + 2) = 7;    // j.w.
2[Tab] = 7;       // poprawne !!! gdyż *(Tab + 2) == *(2 + Tab)
pa[2] = 7;        // poprawne
```


Wskaźniki i tablice

- czy jednak `Tab` (nazwa tablicy) i `pa` (wskaźnik do tablicy `Tab`) to dokładnie to samo? - NIE !
- wskaźnik jest zmienną (wskaźnikową)
- nazwa tablicy jest stałą (wskaźnikową)

```
// ... gdzieś w programie

pa++;             // OK
pa = Tab;         // OK
Tab++;            // źle
Tab = pa;         // źle

// inny sposób zapamiętania
int b = 7;

b++;              // OK
7++;              // ???
7 = b;            // ???
```

Specyfikator const

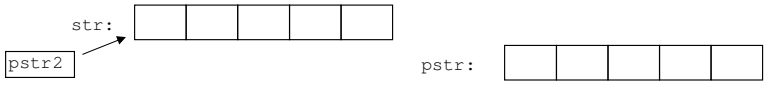
- specyfikator `const` NIE zmienia charakteru zmiennej ale uniemożliwia modyfikacje wartości tej zmiennej
- poprzednio mowa była o różnych charakterach zmiennej (stała i zmienna wskaźnikowa)
- użycie `const` w deklaracji tablicy oznacza, że żaden jej element nie może się zmienić

```
const int Tab[] = {1, 4 , -7, 88};
Tab[2] = 100;      // źle ?
```

- uwaga: wg. normy ANSI C próba zmiany wartości zmiennej zadeklarowanej jako `const` kończy się w sposób zależny od implementacji !

Tablice znakowe

```
char str[] = "ABCD";                        // pięć znaków; ABCD + '\0'
char str[];                                 // źle; ile zaalokować pamięci ?
char str[] = {'A', 'B', 'C', 'D', '\0'};

char *pstr  = "ABCD";              // wskaźnik wskazuje na stałą napisową !
char *pstr2 = str;                 // wskaźnik wskazuje na zmienną napisową

char *bufor;
gets (bufor);                      // źle !!!
```



- `str` wskazuje zawsze na ten sam obszar pamięci (jest tzw. stała wskaźnikową). Nie można zmienić wartości `str` (tak jak nie można na przykład napisać `7++`). Można oczywiście napisać : `str[2]='X';`
- `pstr` (zmienna wskaźnikowa) może zmieniać wartość jak każda zmienna. Może więc wskazywać na cokolwiek
- Ale uwaga!
  `pstr` jest wskaźnikiem zainicjowanym tak, aby wskazywał stałą napisową. Próba zmiany stałej napisowej zakończy się błędem wykonania (`pstr[2]='X';` run-time error) (Według K&R - ma skutek nieokreślony)

Tablice jako parametry funkcji

```
// przekaż do funkcji tablicę

#include <stdio.h>

int GetTab ( int* );
// lub
// int GetTab ( int[] );
// int GetaTab ( int A[] );   identyfikator zbędny w prototypie
// int GetaTab ( int A[7] );  rozmiar jest tu zbędny

int main (void)
{
 int Tab[]={1,2,3,4,5,6,7}, suma;

 suma = GetTab ( Tab );        // źle:    suma = GetTab( &Tab );
                               // dobrze: suma = GetTab( &Tab[0]);
                               // dobrze: suma = GetTab( &Tab[-3]);  ???
                               // dobrze: suma = GetTab( &Tab[2]);   podtablica
                               // dobrze: suma = GetTab( Tab + 2 );  podtablica
 return 0;
}

int GetTab (int *ptr);
{
  int suma;

  for (i=0; i < 6; i++)                // lepiej wymiar przekazać z miejsca wywołania
  // for (i=0; i < sizeof(ptr); i++)   // źle: sizeof(ptr) == 4; zawsze!
    suma += ptr[i];                    // lub: suma += *(ptr + i);
  return (suma);  }
```

to nawet sugeruje, że tablica jest przekazywana przez wartość !!!

sizeof(Tab) == 28 (7 x 4)
Tab jest typu **int (*)[7]**
(wskaźnik do tablicy 7-miu int-ów)

???

Pointer / offset notation

```
#include <stdio.h>

int main()
{
   int b[] = { 10, 20, 30, 40 };
   int *bPtr = b;
   int i;
   int offset;

   printf( "\nPointer / offset notation\n" );

   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   }

   return 0;
}
```

```
c:\Temp>Project1.exe

Pointer / offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

Pointer subscript notation

```
Pointer subscript notation
#include <stdio.h>

int main()
{
   int b[] = { 10, 20, 30, 40 };
   int *bPtr = b;
   int i;
   int offset;

   printf( "\nPointer subscript notation\n" );

   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   }

   return 0;
}
```

```
Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40
```


Zwracanie z funkcji wskaźnika

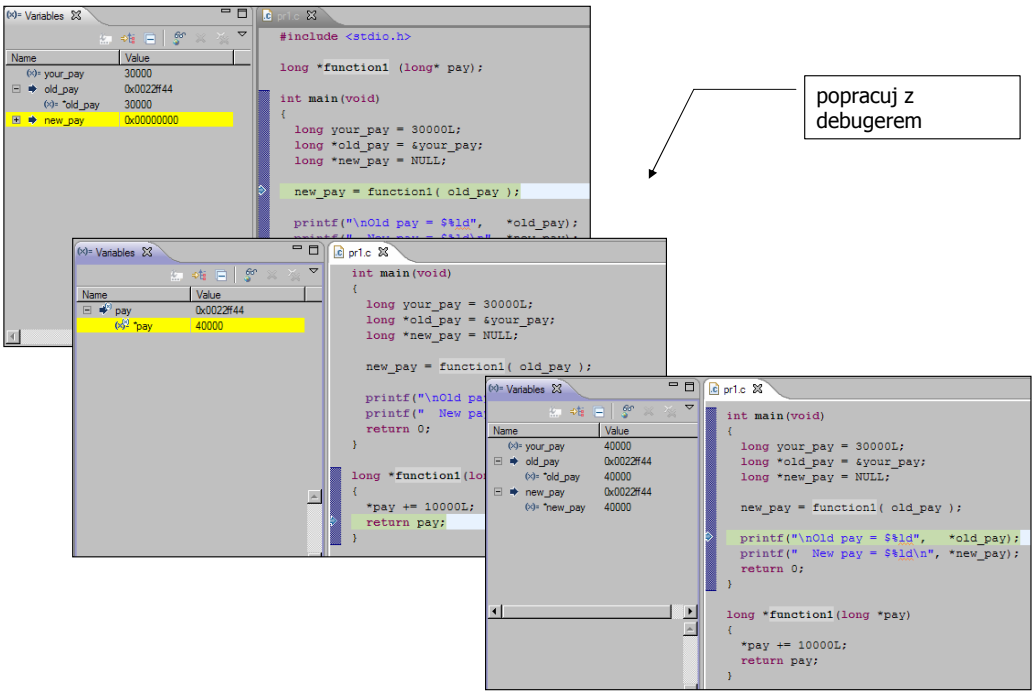
```
#include <stdio.h>

long *function1 (long* pay);

int main(void)
{
  long your_pay = 30000L;
  long *old_pay = &your_pay;
  long *new_pay = NULL;

  new_pay = function1( old_pay );

  printf("\nOld pay = $%ld",   *old_pay);
  printf("  New pay = $%ld\n", *new_pay);
  return 0;
}

long *function1(long *pay)
{
  *pay += 10000L;
  return pay;
}
```

przykład dosyć sztuczny

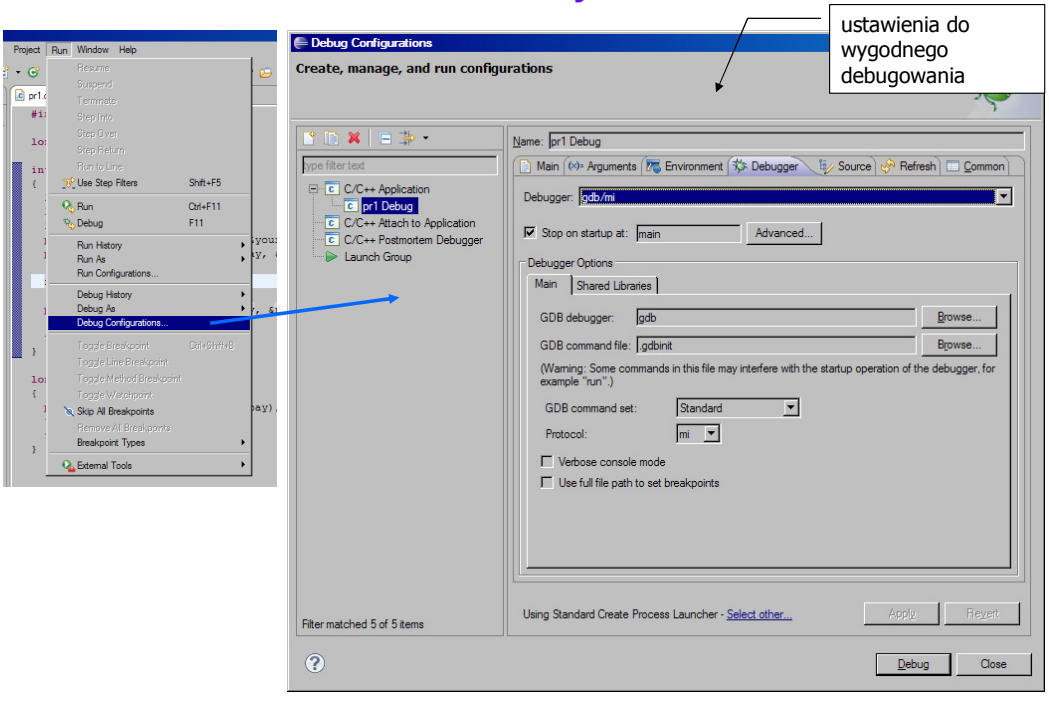
```
c:\Temp>Project1.exe

Old pay = $40000  New pay = $40000
```

wyjaśnij to

Zwracanie z funkcji wskaźnika



Zwracanie z funkcji wskaźnika



Zwracanie z funkcji wskaźnika

```
...
int main(void)
{
  long your_pay = 30000L;
  long *old_pay = &your_pay;
  long *new_pay = NULL;
  printf("0x%p, 0x%p, 0x%p\n\n", &your_pay, old_pay, new_pay);
  printf("0x%p, 0x%p\n\n", &old_pay, &new_pay);

  new_pay = function1( old_pay );

  printf("0x%p, 0x%p\n\n", new_pay, &new_pay);

  return 0;
}

long *function1(long *pay)
{
  printf("0x%p, 0x%p\n\n", pay, &pay);
  *pay += 10000L;
  return pay;
}
```

Tylko czy potrzebujemy znać adresy wskaźników? NIE, wystarczą nam ich WARTOŚCI !!!

	Contents	Address
your_pay	30000	0x0022FF4C
*old_pay	0x0022FF4C	
*new_pay		
*pay		

```
0x0022FF4C, 0x0022FF4C, 0x00000000

0x0022FF48, 0x0022FF44

0x0022FF4C, 0x0022FF30

0x0022FF4C, 0x0022FF44
```

wypełnij pozostałe wartości