# ALGORYTMY I STRUKTURY DANYCH

**Temat 3:**

## Tablice, struktury, wskaźniki - przypomnienie

**Wykładowca:** dr inż. Zbigniew TARAPATA

e-mail: Zbigniew.Tarapata@isi.wat.edu.pl

http://www.tarapata.strefa.pl/p_algorytmy_i_struktury_danych/

Współautorami wykładu są: G.Bliźniuk, A.Najgebauer, D.Pierzchała

## Tablice

**Tablica to zbiór elementów tego samego typu, do których odwołuje się za pośrednictwem wspólnej nazwy.**

**Dostęp do konkretnego elementu tablicy uzyskuje się za pomocą indeksu.**

**W języku C wszystkie tablice składają się z ciągłych obszarów pamięci.**
**Najniższy adres odpowiada pierwszemu elementowi tablicy, najwyższy zaś ostatniemu.**

## Tablice

**Tablice mogą mieć od jednego do kilku wymiarów.**

**Najczęściej spotykana tablica to łańcuch zakończony znakiem ‘/0’, który jest po prostu tablicą znaków zakończoną znakiem ‘/0’.**

**Ogólna postać deklaracji tablicy jednowymiarowej:**
**typ nazwa_zmiennej[rozmiar];**

**Przykład:**
**double saldo[100];**



## Tablice

**Dostęp do elementu tablicy uzyskuje się, indeksując nazwę tablicy. Indeksowanie wykonuje się, umieszczając indeks w nawiasach kwadratowych za nazwą tablicy.**

**Przykład:**
**saldo[3] = 12.23;**

## Tablice

**W języku C pierwszy element wszystkich tablic to element o indeksie 0 (zero).**

**Tak więc:**

```
char p[10];
```

**deklaruje tablicę znakową, składającą się z elementów od p[0] do p[9].**



## Tablice

**Można utworzyć wskaźnik do pierwszego elementu tablicy, podając po prostu jej nazwę bez indeksu.**

```
int sample[10];
```

**Można utworzyć wskźnik do pierwszego elementu, używając nazwy sample.**

**Tak więc poniższy fragment programu przypisuje zmiennej p adres pierwszego elementu tablicy sample:**

```
int *p;
int sample[10];

p = sample;
```

## Łańcuchy znaków

**Najczęstsze zastosowanie tablic jednowymiarowych to łańcuchy znaków.**

**W języku C łańcuch to tablica znaków zakończona znakiem '/0'.**

**Deklarując tablicę znakową, w której mają być przechowywane łańcuchy,**
**należy określić jej długość o jeden znak większą niż najdłuższy łańcuch,**
**jaki może być w niej umieszczony.**

**Przykład:**
**char str[11] - deklaracja tablicy str dla łańcucha 10-znakowego.**



## Łańcuchy znaków

**W języku C istnieje szeroki zakres funkcji służących do manipulowania łańcuchami.**

**Najczęściej używane:**
**strcpy(s1, s2) – kopiuje s2 do s1,**
**strcat(s1, s2) – dołącza s2 do s1,**
**strlen(s1) – zwraca długość s1,**
**strcmp(s1, s2) – zwraca 0, jeśli s1 oraz s2 są identyczne; wartość mniejszą od zera, jeśli s1<s2, większą od zera, jeśli s1>s2,**
**strchr(s1, ch) – zwraca wskaźnik do pierwszego wystąpienia znaku ch w łańcuchu s1,**
**strstr(s1, s2) – zwraca wskaźnik do pierwszego wystąpienia łańcucha s2 w łańcuchu s1.**

## Tablice wielowymiarowe

**Język C pozwala operować na tablicach wielowymiarowych.**

**Aby zadeklarować dwuwymiarową tablicę d liczb całkowitych o wymiarach 10 na 20, należy napisać:**

```
int d[10][20];
```

**Odwołanie do elementu 1,2 tablicy d :**

```
d[1][2];
```

**Tablice dwuwymiarowe przechowywane są w macierzy wierszowo-kolumnowej, w której pierwszy indeks oznacza wiersz, drugi zaś kolumnę.**



## Struktury

**Struktura to zbiór zmiennych, do których można się odwoływać za pośrednictwem jednej nazwy, dzięki czemu w wygodny sposób można razem przechowywać pokrewne informacje.**

**Deklaracja struktury to szablon, którego można używać do tworzenia egzemplarzy struktury (czyli jej instancji).**

**Zmienne, z których zbudowana jest struktura, nazywają się składowymi (polami).**

**Struktury nazywane są także jako rekordy.**

## Struktury

**Z reguły wszystkie składowe struktury są ze sobą logicznie powiązane.**

**Słowo kluczowe struct informuje kompilator, że ma miejsce deklaracja struktury.**

```
struct addr
{
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
};
```



## Struktury

**Deklaracja zmiennej:**

```
struct addr dane_adresowe;
```

**Deklaracja struktury i zmiennych:**

```
struct addr {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} dane_adresowe, bdane, cdane;
```

## Struktury

**Odwołanie do składowych struktury:**

```
dane_adresowe.zip = 12345;
gets(dane_adresowe.name);
```

**UWAGA!**

**Dopuszczalne jest następujące przypisanie:**

```
dane_adresowe = bdane;  /* obie zmienne mają taką samą zawartość */
```

**ale niedopuszczalne jest porównanie** `dane_adresowe = = bdane`

**Aby porównać dwie instancje struktury tego samego typu należy np. napisać funkcję realizującą porównanie.**



## Struktury

**Tablice struktur**

**Aby zadeklarować tablicę struktur, należy najpierw zdefiniować strukturę, a następnie zadeklarować zmienną tablicową tego typu.**

```
struct addr dane_adresowe[100];
```

**Odwołanie się do elementu struktury w tablicy struktur:**

```
dane_adresowe[5].name[0] = 'Z';
strcmp(dane_adresowe[2].name, "Kowalski");
```

## Wskaźniki - podstawy

**Wskaźnik to zmienna przechowująca adres pamięci.**

**Ten adres to lokalizacja innej zmiennej w pamięci (jej adres).**

**Jeśli jedna zmienna zawiera adres innej zmiennej, to mówi się, że ta pierwsza zmienna to wskaźnik do drugiej zmiennej.**




## Wskaźniki - podstawy

**Deklaracja wskaźnika:**

**typ *nazwa;**

## Wskaźniki - podstawy

**Operatory wskaźnikowe: * oraz &.**

**Jednoargumentowy operator & zwraca adres w pamięci.**

**Operacja:**

**m = &count;**

**umieszcza w zmiennej m adres zmiennej count.**

**Załóżmy, że zmienna count zajmuje komórkę pamięci o adresie 2000.**

**Załóżmy, też że jej wartość to 100.**

**Tak więc w wyniku powyższego przypisania zmienna m otrzyma wartość 2000.**

**„m otrzymuje adres zmiennej count”**



## Wskaźniki - podstawy

**Jednoargumentowy operator * zwraca wartość zmiennej, znajdującej się pod adresem następującym po operatorze.**

**Jeśli zmienna m zawiera adres zmiennej count to:**

**q = *m;**

**umieszcza wartość zmiennej count w zmiennej q.**

**Tak więc zmienna q będzie miała wartość 100, gdyż 100 znajduje się pod adresem 2000, który jest adresem zapisanym w m.**

**„q otrzymuje wartość, znajdującą się pod adresem m”**

# Wskaźniki - podstawy

Przykład niepoprawnego wykorzystania wskaźnika.

```
int main(void)
{
  int *p;
  *p = 10;  /* niepoprawne - p jeszcze na nic nie wskazuje */
}
```



# Wskaźniki – działania na wskaźnikach

**Oprócz operatorów * i & istnieją tylko cztery inne operatory, które można stosować ze wskaźnikami: +, ++, -, --.**

**Operator ++ zwiększa adres, na który wskazuje wskaźnik o jedną długość typu wskaźnika.**
**Np. jeżeli p zawiera adres 200 do liczby całkowitej (o długości 2 bajtów), to instrukcja p++ powoduje, że p będzie miało adres 202, tzn. 200+(1*2).**
**Równoważnie można to zapisać: p+=1;**

**Jeżeli mamy:**

```
int *p;
p=p+200;
```

**to p będzie wskazywało na 200. liczbę całkowitą występująca za liczbą wskazywaną poprzednio.**
**Podobnie działają operatory odejmowania.**

# Wskaźniki – działania na wskaźnikach

**Aby otrzymać wartość o jeden większą od wskazywanej przez wskaźnik, należy posłużyć się konstrukcją:**

```
(*p)++;
```

**(Jeżeli założymy, że p wskazuje na adres 200, pod którym znajduje się liczba 100 typu int, to po wykonaniu instrukcji p wskazuje na liczbę 101 nadal pod adresem 200).**
**UWAGA!**
**Instrukcja:**

```
*p++;
```

**powoduje zwiększenia wskaźnika p o 1 (a nie wartości obiektu, na który wskazuje!!!), tzn. p wskazuje na adres 202 (zakładamy, że liczba typu int zajmuje 2 bajty).**
**Dzieje się tak dlatego, że operator * ma wyższy priorytet niż ++.**



# Wskaźniki – rzutowanie typów

**Przypomnijmy, że aby rzutować zmienną *t* typu *T* na inny typ *S*, poprzedzamy *t* nazwą typu *S* ujętą w nawias.**

**Ta sama zasada dotyczy rzutowania wskaźników!!!**
**Załóżmy, że mamy:**

```
int *iptr;
float *fptr;
```

**Aby przypisać wskaźnik do liczby całkowitej iptr wskaźnikowi do liczby zmiennoprzecinkowej fptr, piszemy:**

```
fptr=(float *) iptr;
```

# Wskaźniki i tablice

**Niech:**

**int sample[10];**

**Można utworzyć wskaźnik do pierwszego elementu, używając nazwy sample. Tak więc poniższy fragment programu przypisuje zmiennej p adres pierwszego elementu tablicy sample:**

**int *p;**
**int sample[10];**

**p = sample;**

**Równoważne są również odwołania:**
**sample[i] oraz *(sample+i)**

**Jeżeli sample1 byłoby tablicą dwuwymiarową (int sample1[10][20]; ), to równoważne są odwołania:**
**sample1[i][j] oraz *(*(sample1+i)+j) (sprawdzić !!!)**



# Wskaźniki i tablice

- Nazwa tablicy bez indeksu jest wskaźnikiem do początku tablicy. Przeanalizuj przykład poniżej!!!

```
#include <stdio.h>
int main(void)
{
  int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
  int *p;
  p = a; /* przypisanie p adresu początkowego tablicy a */
  /* wypisanie elementu pierwszego, drugiego i trzeciego tablicy a  z
     użyciem p*/
  printf("%d %d %d\n", *p, *(p+1), *(p+2));
  /* to samo z wykorzystaniem a */
  printf("%d %d %d", a[0], a[1], a[2]);
  return 0;}
```

# Wskaźniki i tablice

Zadanie 2

Jaka wartość zostanie wypisana na ekranie?

```
int temp[5] = {10, 19, 23, 8, 9};
int *p;
p = temp;
printf("%d", *(p+3));
```

**Odpowiedź: 8**



# Wskaźniki i tablice

Wskaźniki mogą być organizowane, jak każdy inny typ danych. Podana instrukcja deklaruje 10-elementową tablicę wskaźników do liczb całkowitych:

```
int *pa[10];
```

Aby przypisać adres zmiennej całkowitej zmienna dziewiątemu elementowi tablicy, należy użyć instrukcji:

```
pa[8]=&zmienna;
```

Aby przypisać zmiennej wskazywanej przez trzeci element tablicy pa wartość 100, należy użyć instrukcji:

```
*pa[2]=100;
```

# Wskaźniki, tablice i funkcje

Jeżeli przekazujemy dwuwymiarową tablicę do funkcji:

```
int array[LWierszy][LKolumn];

f(array);
```

jej deklaracja musi mieć postać:

```
f(int a[][LKolumn])
{. . .}
```

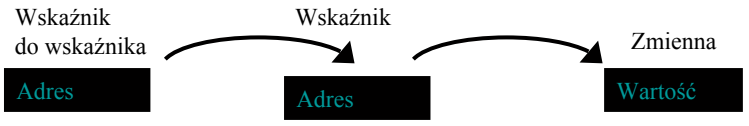
lub

```
f(int (*ap)[LKolumn])  /* ap to wskaźnik na tablicę */
{. . .}
```



# Wskaźniki wielokrotne

Możliwe jest, aby wskaźnik wskazywał na inny wskaźnik (tzw. wielokrotne odwołanie niejawne).




Aby zadeklarować wskaźnik do wskaźnika, należy przed nazwą wskaźnika umieścić dodatkową gwiazdkę:

```
char **mp;
```

# Wskaźniki wielokrotne

Aby dostać się do wartości wskazywanej niejawnie przez wskaźnik do wskaźnika, należy zastosować operator * dwukrotnie:

```
char **mp, *p, ch;

p=&ch;        /* pobranie adresu ch */
mp=&p;        /* pobranie adresu p */

/* przypisanie ch wartości A z wykorzystaniem wielokrotnego odwołania niejawnego */
**mp='A';
```

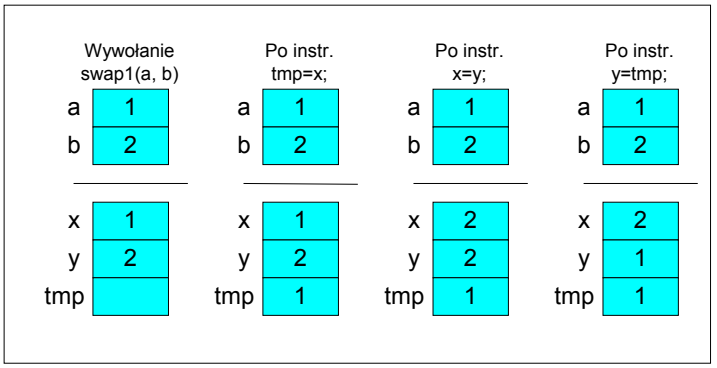


# Wskaźniki – wykorzystanie

**Przekazywanie parametrów przez wartość (bez wskaźników)**

```
void swap1 (int x, int y)
{
int tmp;
tmp=x; x=y; y=tmp;
return; }
```

**Zamiana wartości liczb x i y nie działa !!!**

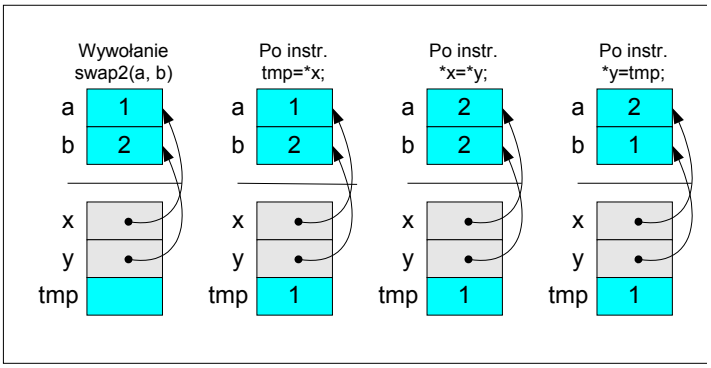


# Wskaźniki – wykorzystanie

**Przekazywanie parametrów przez adres (ze wskaźnikami)**

```
void swap2 (int *x, int *y)
{
int tmp;
tmp=*x; *x=*y; *y=tmp;
return; }
```

**Zamiana wartości liczb x i y działa !!!**





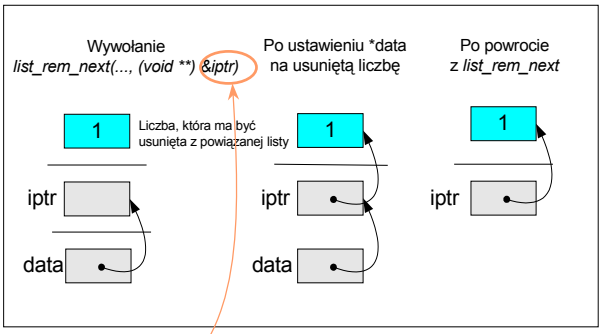

# Wskaźniki – wykorzystanie

**Wskaźniki do wskaźników jako parametry**

Aby móc usuwać elementy za pomocą funkcji:
*int list_rem_next (List *list, ListElem *element, void **data)*

z listy następującej:

```
typedef  struct ListElem_
{
void *data;
struct ListElem_ *next;
} ListElem;
```




**musimy użyć wskaźnika do wskaźnika na dane data, gdyż po powrocie z funkcji data ma wskazywać usunięte dane. Przekazujemy adres wskaźnika iptr, gdyż operacja zmodyfikuje wartość samego wskaźnika tak, aby wskazywał usuniętą daną.**

# Wskaźniki – wykorzystanie

## Wskaźniki do funkcji

Wskaźniki do funkcji to wskaźniki, które nie wskazują danych, lecz fragmenty kodu wykonywalnego lub informacje potrzebne do wywołania takiego kodu.

Deklaracje wskaźników do funkcji są bardzo podobne do deklaracji zwykłych funkcji, ale przed nazwą tej funkcji pojawia się gwiazdka (*), która wraz z nazwą ujęta jest w nawias.

Przykład:
int (*porownaj) (void *klucz1, void *klucz2);

Deklaracja ta oznacza, że porownaj może wskazywać dowolną funkcję o dwóch parametrach (klucz1, klucz2) będących wskaźnikami ogólnymi (void) i zwracającą liczbę całkowitą (int).



# Wskaźniki – dynamiczna alokacja pamięci

- Podstawowe funkcje dynamicznej alokacji pamięci, to *malloc*() , przydzielająca pamięć oraz *free*() , zwalniająca pamięć uprzednio przydzieloną.
- Prototypy funkcji:

  *void *malloc(size_t liczba_bajtów)*

  *void free(void *wsk)*

Funkcja *malloc*() zwraca wskaźnik do początku przydzielonego fragmentu pamięci o rozmiarze liczba_bajtów. Funkcja *free*() zwalnia obszar pamięci, na który wskazuje *wsk*.

## Wskaźniki – dynamiczna alokacja pamięci

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

int main(void)
{
  char *p;

  p = malloc(80); /* przydział pamięci ; równoważnie : p=malloc(80*sizeof(char); ) */

  if(!p) {
    printf(„Błąd przydziału pamięci");
    exit(1);
  }

  printf(„Wprowadź napis: ");
  gets(p);
  printf(p);
  free(p); /* to wyłącznie w celach demonstracyjnych, po wyjściu z programu pamięc
     jest zwalaniana */            /* automatycznie */

  return 0;
}
```



## Wskaźniki – dynamiczna alokacja tablic dwuwymiarowych

- Pierwszy sposób:

```
#include <stdlib.h>
array1 = (int **)malloc(nrows * sizeof(int *));
for(i = 0; i < nrows; i++)
    array1[i] = (int *)malloc(ncolumns * sizeof(int));
```

- Drugi sposób:

```
#include <stdlib.h>
array2 = (int **)malloc(nrows * sizeof(int *));
array2[0] = (int *)malloc(nrows * ncolumns *
    sizeof(int));
for(i = 1; i < nrows; i++)
    array2[i] = array2[0] + i * ncolumns;
```

**Dziękuję za uwagę**