Programowanie w języku C (EAR1S02005)

Politechnika Białostocka - Wydział Elektryczny

Automatyka i Robotyka, semestr II, studia stacjonarne I stopnia

Rok akademicki 2020/2021

Zajęcia nr 7 (16.03.2021)

dr inż. Jarosław Forenc

Struktury w języku C

- Tablica - ciągły obszar pamięci zawierający elementy tego samego typu

- Struktura - zestaw elementów różnych typów, zgrupowanych pod jedną nazwą

- Deklaracja struktury

```
struct nazwa
{
    opis_pola_1;
    opis_pola_2;
    ...
};
```

```
struct punkt
{
    int x;
    int y;
};
```

- Elementy struktury to pola (dane, komponenty, składowe) struktury

Deklaracja struktury

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
};
```

```
struct zesp
{
    float Re, Im;
};
```

- Deklarując strukturę tworzymy nowy typ danych (struct osoba, struct zesp), którym można posługiwać się tak samo jak każdym innym typem standardowym
- Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci)
- Zapisanie danych do struktury wymaga zdefiniowania zmiennej strukturalnej

Deklaracja zmiennej strukturalnej

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek, waga;
} Kowal ;

int main(void)
{
    struct osoba Nowak ;
    ...
}
```

- Kowal, Nowak - zmienne typu struct osoba

Odwołania do pól struktury

- Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

- Operator . nazywany jest operatorem bezpośredniego wyboru pola
- Zapisanie wartości do pól zmiennej Nowak ma postać

```
Nowak.wiek = 25;
strcpy(Nowak.imie,"Jan");
```

- Wyrażenie Nowak.wiek traktowane jest jak zmienna typu int, natomiast wyrażenie Nowak.imie traktowane jest jak łańcuch znaków

```
printf("%s - wiek %d\n", Nowak.imie, Nowak.wiek);
scanf("%d", &Nowak.wiek);
gets(Nowak.imie);
```


Odwołania do pól struktury

- Gdy zmienna strukturalna jest wskaźnikiem, to do odwołania do pola struktury używamy operatora pośredniego wyboru pola (->)

```
wskaźnik_do_struktury -> nazwa_pola
```

```
struct osoba Nowak, *Nowak1;
Nowak1 = &Nowak;
Nowak1 -> wiek = 25;
/* lub */
(*Nowak1).wiek = 25;
```

- W ostatnim zapisie nawiasy są konieczne, gdyż operator . ma wyższy priorytet niż operator *

Struktury w języku C

- Inicjalizacja może dotyczyć tylko zmiennych strukturalnych, nie można inicjalizować pól w deklaracji struktury

```
struct osoba
{
    char imie[15], nazwisko[20];
    int wiek, waga;
};
```

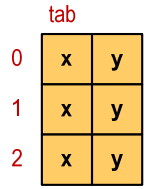
```
struct osoba Nowak = {"Jan","Nowak",25,74};
```

- Do zmiennych strukturalnych można stosować operator przypisania (=)

```
struct osoba Kowal = {"Ewa","Kowal",21,54};
struct osoba Kowal1;
Kowal1 = Kowal;
```

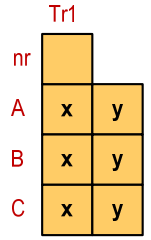

Złożone deklaracje struktur

```
struct punkt
{
   int x;
   int y;
} tab[3];
```




```
tab[0].x = 10;
tab[0].y = 20;
tab[1].x = 15;
…
```

```
struct trojkat
{
   int nr;
   struct punkt A, B, C;
} Tr1;
```



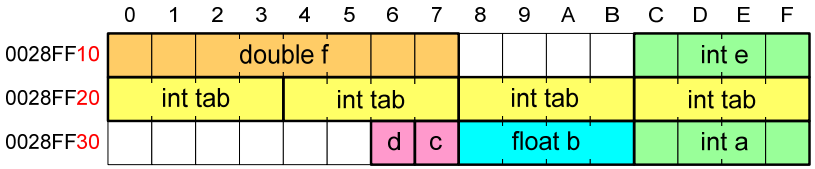

```
Tr1.nr = 1;
Tr1.A.x = 10;
Tr1.A.y = 20;
Tr1.B.x = 15;
…
```

Co to jest wskaźnik?

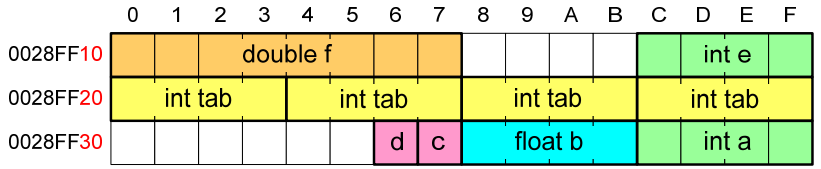
- Wskaźnik - zmienna mogącą zawierać adres obszaru pamięci - najczęściej adres innej zmiennej (obiektu)

```
int a;
float b;
char c, d;
int tab[4], e;
double f;
```

- Zmienne przechowywane są w pamięci komputera



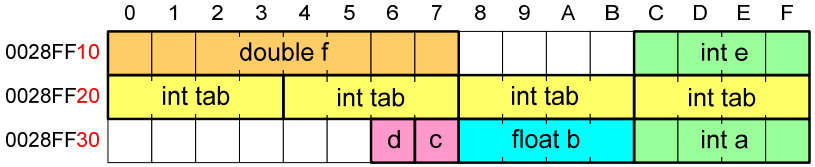
Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
printf("Adres zmiennej a:  %p\n", &a);
printf("Adres tablicy tab: %p\n", tab);
```


Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
printf("Adres zmiennej a:
printf("Adres tablicy tab: %p\n", tab);
```

```
Adres zmiennej a:  0028FF3C
Adres tablicy tab: 0028FF20
```


Deklaracja wskaźnika

- Deklarując wskaźnik (zmienną wskazującą) należy podać typ obiektu na jaki on wskazuje, a jego nazwę poprzedzić symbolem gwiazdki (*)

`typ *nazwa;`	`typ* nazwa;`	`typ * nazwa;`	`typ*nazwa;`

- Deklaracja zmiennej wskaźnikowej do typu int

```
int *ptr;
```

- Mówimy, że zmienna ptr jest typu: wskaźnik do zmiennej typu int
- Do przechowywania adresu zmiennej typu double trzeba zadeklarować zmienną typu: wskaźnik do zmiennej typu double

```
double *ptrd;
```

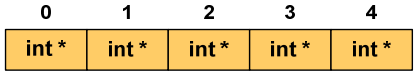
Deklaracja wskaźnika

- Można konstruować wskaźniki do danych dowolnego typu łącznie z typami wskaźnik do wskaźnika do...

```
char **wsk;
```

- Można deklarować tablice wskaźników - zmienna tab_ptr jest tablicą zawierającą 5 wskaźników do typu int

```
int *tab_ptr[5];
```



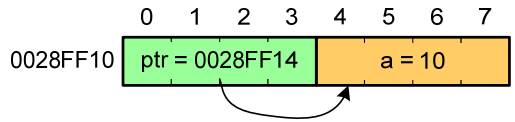
- Natomiast zmienna ptr_tab jest wskaźnikiem do 5-elementowej tablicy liczb int

```
int (*ptr_tab)[5];
```


Przypisywanie wartości wskaźnikom

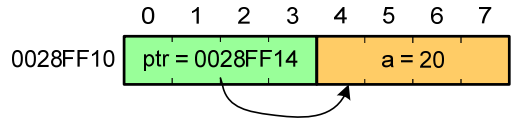
- Wskaźnikom można przypisywać adresy zmiennych
- Adresy takie tworzy się za pomocą operatora pobierania adresu &

```
int a = 10, *ptr;
ptr = &a;
```



- Mając adres zmiennej można „dostać się" do jej wartości używając tzw. operatora wyłuskania (odwołania pośredniego) - gwiazdki (*)

```
*ptr = 20;
```



- Wskaźnik pusty:

```
int *ptr = 0;
```

```
int *ptr = NULL;
```


Przykład: przypisywanie wartości wskaźnikom

```
#include <stdio.h>

int main(void)
{
   int x = 15;
   int *ptri = NULL;

   printf("x =    %d\n",x);
   printf("ptri = %p\n",ptri);

   ptri = &x;            // przypisanie adresu
   printf("ptri = %p\n",ptri);

   *ptri = *ptri + 10;  // x = x + 10
   printf("x =    %d\n",x);
   printf("x =    %d\n",*ptri);

   return 0;
}
```

```
x =    15
ptri = 0000000000000000
ptri = 00000000010FF960
x =    25
x =    25
```

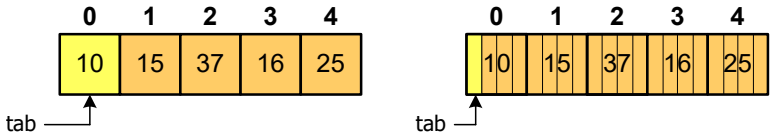

Arytmetyka wskaźnikowa

- Dodanie liczby całkowitej do wskaźnika - przed dodaniem liczby całkowitej jest ona mnożona przez liczbę bajtów zajmowanych przez wartość wskazywanego typu
- Zwiększenie wskaźnika (inkrementacja) - do wskaźnika można dodać 1 lub zastosować operator ++
- Odjęcie liczby całkowitej od wskaźnika - działa analogicznie jak dodanie liczby całkowitej do wskaźnika, ale wskaźnik musi być lewym operandem odejmowania
- Zmniejszenie wskaźnika (dekrementacja) - działa analogicznie jak inkrementacja

Wskaźniki a tablice

- Nazwa tablicy jest jej adresem (dokładniej - adresem elementu o indeksie 0)

```
int tab[5] = {10,15,37,16,25};
```

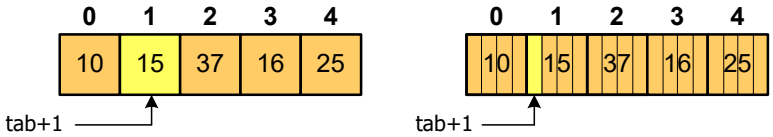


- Zastosowanie operatora * przed nazwą tablicy pozwala „dostać się" do zawartości elementu o indeksie 0

*tab jest równoważne tab[0]

Wskaźniki a tablice

- Dodanie 1 do adresu tablicy przenosi nas do elementu tablicy o indeksie 1 (przesunięcie o 4 bajty, gdyż int zajmuje tyle bajtów)



zatem: *(tab+1) jest równoważne tab[1]
ogólnie: *(tab+i) jest równoważne tab[i]

- W zapisie *(tab+i) nawiasy są konieczne, gdyż operator * ma bardzo wysoki priorytet

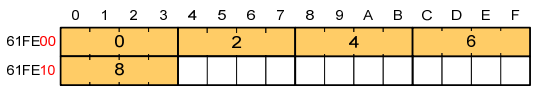
x = *tab+1; jest równoważne x = tab[0]+1;

Przykład: operacje na wskaźnikach

```
int tab[5] = {0,2,4,6,8}, *ptr;

printf("Adres: %p, wartosc: %d\n",tab,*tab);
printf("Adres: %p, wartosc: %d\n",tab+2,*(tab+2));
ptr = tab;
printf("Adres: %p, wartosc: %d\n",ptr,*ptr);
ptr = ptr + 4;
printf("Adres: %p, wartosc: %d\n",ptr,*ptr);
ptr--;
printf("Adres: %p, wartosc: %d\n",ptr,*ptr);
```

```
Adres: 000000000061FE00, wartosc: 0
Adres: 000000000061FE08, wartosc: 4
Adres: 000000000061FE00, wartosc: 0
Adres: 000000000061FE10, wartosc: 8
Adres: 000000000061FE0C, wartosc: 6
```



Dynamiczny przydział pamięci w języku C

- Kiedy stosuje się dynamiczny przydział pamięci?
  - gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu a nie podczas jego kompilacji
  - gdy rozmiar tablicy jest bardzo duży
- Do dynamicznego przydziału pamięci stosowane są funkcje:
  - calloc()
  - malloc()
- Przydział pamięci następuje w obszarze sterty (stosu zmiennych dynamicznych)
- Przydzieloną pamięć należy zwolnić wywołując funkcję:
  - free()

Dynamiczny przydział pamięci w języku C

CALLOC stdlib.h

```
void *calloc(size_t num, size_t size);
```

- Przydziela blok pamięci o rozmiarze num*size (mogący pomieścić tablicę num-elementów, każdy rozmiaru size)
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć jest inicjowana zerami (bitowo)
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
```


Dynamiczny przydział pamięci w języku C

MALLOC stdlib.h

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

- Przydziela blok pamięci o rozmiarze określonym parametrem size
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć nie jest inicjowana
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) malloc(10*sizeof(int));
```


Dynamiczny przydział pamięci w języku C

FREE stdlib.h

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

- Zwalnia blok pamięci wskazywany parametrem ptr
- Wartość ptr musi być wynikiem wywołania funkcji calloc() lub malloc()

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
/* ... */
free(tab);
```


Przykład: przydział pamięci na jedną zmienną

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

int main(void)
{
   float *wsk;

   wsk = (float *) calloc(1,sizeof(float));
   if (wsk == NULL)
   {
      printf("Blad przydzialu pamieci\n");
      return 0;
   }

   *wsk = 123.45f;
   printf("wartosc = %g\n",*wsk);

   free(wsk);
   return 0;
}
```

```
wartosc = 123.45
```

Przykład: przydział pamięci na tablicę

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

int main(void)
{
   int *tab, n = 10;

   tab = (int *) calloc(n,sizeof(int));

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

   free(tab);

   return 0;
}
```

```
tab[0] = 0
tab[1] = 1
tab[2] = 4
tab[3] = 9
tab[4] = 16
tab[5] = 25
tab[6] = 36
tab[7] = 49
tab[8] = 64
tab[9] = 81
```


Przykład: przydział pamięci na strukturę

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

struct punkt
{
   int x, y;
};

int main(void)
{
   struct punkt p, *wsk_p;

   wsk_p = (struct punkt*) malloc(sizeof(struct punkt));

   p.x = 10; p.y = 20;
   wsk_p->x = 30; wsk_p->y = 40;
   printf("%d,%d - %d,%d\n",p.x,p.y,wsk_p->x,wsk_p->y);

   free(wsk_p);
   return 0;
}
```

```
10,20 - 30,40
```