

Programowanie strukturalne

- wykład 7

dr Piotr Jastrzębski

Tablice “wielowymiarowe” - cd.

Misz-masz definicji

Tablice wielowymiarowe

- ▶ tablice wielowymiarowe o stałym rozmiarze
- ▶ tablice wielowymiarowe statyczne

Tablice tablic

- ▶ tablice wielowymiarowe dynamiczne
- ▶ tablice wielowymiarowe o zmiennym rozmiarze (?)

Kod do analizy

https:

[//gist.github.com/pjastr/4ac29959e212436807c626610cb1b964](https://gist.github.com/pjastr/4ac29959e212436807c626610cb1b964)

Tablice postrzępione

Tablica postrzępiona jest to taka dwuwymiarowa tablica, która posiada różną liczbę kolumn w każdym rzędzie.

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

int main()
{
    int (*(arr2[])) = {
        (int[]) {0, 1, 4, 3},
        (int[]) {4, -1},
        (int[]) {6, -2, 8}
    };
    return 0;
}
```

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

int main()
{
    int **tab = (int**) malloc(sizeof(int*)*2);
    tab[0]=(int*) malloc(sizeof(int)*4);
    tab[1]=(int*) malloc(sizeof(int)*3);
    return 0;
}
```

Złożone typy danych

Struktury

Struktury to specjalny typ danych mogący przechowywać wiele wartości w jednej zmiennej. Od tablic jednakże różni się tym, iż te wartości mogą być różnych typów.

Deklaracja struktury:

```
struct Struktura {  
    int pole1;  
    int pole2;  
    char pole3;  
};
```

Deklaracja zmiennej strukturalnej:

```
struct Struktura zmiennaS;
```

Dostęp do pól - “kropka” - operator wyboru składnika:

```
zmiennaS.pole1 = 60;    /* przypisanie liczb do pól */  
zmiennaS.pole2 = 2;  
zmiennaS.pole3 = 'a'; /* a teraz znaku */
```

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

struct Struktura {
    int pole1;
    int pole2;
    char pole3;
};

int main()
{
    struct Struktura zmiennaS = {60, 2, 'a'};
}
```

```
struct moja_struct {  
    int a;  
    char b;  
} moja = {1, 'c'};
```

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

struct Struktura{
    int pole;
} abc;

int main()
{
    abc.pole=4;
    printf("%d",abc.pole);
    return 0;
}
```

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

struct Struktura {
    int pole1;
    int pole2;
    char pole3;
};

int main()
{
    struct Struktura zmiennaS =
        { .pole1=60, .pole2=2, .pole3='a'};
}
```

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

```
struct Struktura {
    int pole1;
    int pole2;
    char pole3;
};
```

```
int main()
{
    struct Struktura zmiennaS =
        { .pole1=60, .pole2=0.2, .pole3='a'};
    printf("%d\n",zmiennaS);
    printf("%p\n",&zmiennaS);
    printf("%p\n",&zmiennaS.pole1);
    printf("%p\n",&zmiennaS.pole2);
    printf("%p\n",&zmiennaS.pole3);
}
```

“structure padding”

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

struct Struktura {
    int pole1;
    double pole2;
    char pole3;
};

int main()
{
    struct Struktura zmiennaS =
        { .pole1=60, .pole2=0.2, .pole3='a'};
    printf("%p\n",&zmiennaS.pole1);
    printf("%p\n",&zmiennaS.pole2);
    printf("%p\n",&zmiennaS.pole3);
}
```



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

#pragma pack ( 1 )
struct Struktura {
    int pole1;
    double pole2;
    char pole3;
};

int main()
{
    struct Struktura zmiennaS =
        { .pole1=60, .pole2=0.2, .pole3='a'};
    printf("%p\n",&zmiennaS.pole1);
    printf("%p\n",&zmiennaS.pole2);
    printf("%p\n",&zmiennaS.pole3);
}
```

Tablice struktur

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

#pragma pack ( 1 )
struct Struktura {
    int pole1;
    double pole2;
    char pole3;
};

int main()
{
    struct Struktura tabS[5];
    struct Struktura zm;
    tabS[1] = zm;
}
```

Wskaźnik do struktury

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

struct Struktura {
    int pole1;
    double pole2;
    char pole3;
};

int main()
{
    struct Struktura * wsk;
    struct Struktura zm = {20, 3.4, 'w'};
    wsk = &zm;
    printf("%d\n", (*wsk).pole1);
    printf("%d\n", wsk->pole1);
    return 0;
}
```

Struktury a funkcje

- ▶ Składnik (pola) struktur przekazujemy tak jak typ.

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

struct Struktura {
    int pole1;
    double pole2;
    char pole3;
};

int main()
{
    struct Struktura * wsk;
    struct Struktura zm ={-20,3.4,'w'};
    wsk=&zm;
    printf("%d\n",abs(zm.pole1));
    return 0;
}
```

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

```
int suma(int a, int b)
{
    return a+b;
}
```

```
struct Struktura {
    int pole1;
    int pole2;
};
```

```
int main()
{
    struct Struktura zm = {4,5};
    int w= suma(zm.pole1,zm.pole2);
    printf("%d",w);
    return 0;
}
```

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

struct Struktura {
    int pole1;
    int pole2;
};

int suma(struct Struktura zm)
{
    return (zm.pole1+zm.pole2);
}

int main()
{
    struct Struktura zm = {4,5};
    int w= suma(zm);
    printf("%d",w);
    return 0;
}
```

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

struct Struktura {
    int pole1;
    int pole2;
};

int suma(struct Struktura *zm)
{
    return (zm->pole1+zm->pole2);
}

int main()
{
    struct Struktura zm = {4,5};
    int w= suma(&zm);
    printf("%d",w);
    return 0;
}
```

Tak nie działa!

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

struct Struktura {
    int pole1;
    int pole2;
};

void foo(struct Struktura zm)
{
    zm.pole1=43;
}

int main()
{
    struct Struktura zm = {4,5};
    foo(zm);
    printf("%d",zm.pole1);
    return 0;
}
```



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

```
struct Struktura {
    int pole1;
    int pole2;
};
```

```
void foo(struct Struktura *zm)
{
    zm->pole1=43;
}
```

```
int main()
{
    struct Struktura zm = {4,5};
    foo(&zm);
    printf("%d",zm.pole1);
    return 0;
}
```

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

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

```
struct Struktura {  
    int pole1;  
    int pole2;  
};
```

```
struct Struktura foo(struct Struktura zm)  
{  
    zm.pole1=43;  
    return zm;  
}
```

```
int main()  
{  
    struct Struktura zm = {4,5};  
    zm=foo(zm);  
    printf("%d",zm.pole1);  
    return 0;  
}
```

Unie

Unia (ang. union) jest typem, który pozwala przechowywać różne rodzaje danych w tym samym obszarze pamięci (jednak nie równocześnie).

```
union Nazwa {  
    typ1 nazwa1;  
    typ2 nazwa2;  
    /* ... */  
};
```

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

```
union Unia {
    int pole1;
    char pole2;
};
```

```
int main()
{
    union Unia zm;
    zm.pole1=67;
    printf("%c\n",zm.pole1);
    printf("%d\n",zm.pole1);
    printf("%c\n",zm.pole2);
    printf("%d\n",zm.pole2);
    return 0;
}
```

Typ wyliczeniowy

Służy do tworzenia zmiennych, które mogą przyjmować tylko pewne z góry ustalone wartości:

```
enum Nazwa {WARTOSC_1, WARTOSC_2, WARTOSC_N };
```

typedef

Słowo kluczowe typedef jest zaawansowanym elementem języka C, który pozwala tworzyć nowe nazwy typów.

```
typedef struct {  
    int pole;  
} Struktura;  
  
Struktura s1;
```

Bibliografia

- ▶ Stephen Prata, Język C. Szkoła programowania. Wydanie VI, Wyd. Helion, 2016.
- ▶ <https://cybersecurity.umcs.lublin.pl/wp-content/uploads/kmazur/PP2017/>, dostęp online 10.04.2020.
- ▶ https://pl.wikibooks.org/wiki/C/Typy_z%C5%82o%C5%BCone#Struktury, dostęp online 20.04.2020.