Pracownia programowania - Wykład 11/12

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 */
```

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

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

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

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

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

struct Struktura{
   int pole;
 } abc;

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

```c
#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'};
}
```

```c
#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);
```

Co znaczy nazwa struktury?

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

struct Test{
    int p1;
    char p2;
    //double p3; //porównac po odkomentowaniu poczatku linijki
};

int main()
{
    struct Test a={22,'w'};
    printf("%d\n",a); //unexptected
    printf("%p\n",&a); // adres struktury/pierwszego pola
    printf("%p\n",&a.p1); //adres pierwszego pola
    //printf("%p\n",(&a).p1); //blad kompilacji
    printf("%p\n",&a.p2); //adres drugiego pola
    //printf("%d\n",*a); //brak kompilacji
```

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

struct Punkt2D {
    float x;
    float y;
};


int main() {
    struct Punkt2D p1 = {3.0, 4.0};
    struct Punkt2D p2 = {6.0, 8.0};
    return 0;
}
```

“structure padding”

```c
#include <stdio.h>

struct Przyklad {
    char x;
    int y;
};

int main() {
    printf("Rozmiar struktury: %Iu bajty\n", sizeof(struct Przyklad));
    return 0;
}
```

```c
#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);
}
```

```c
#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);
}
```

Parę zasad dla GCC:

1. **Pola są wyrównane do swojego naturalnego granicznika**: Typy danych o wielkości 1 bajta (np. `char`) są wyrównane do granicy 1 bajta, typy danych o wielkości 2 bajty (np. `short`) są wyrównane do granicy 2 bajtów, a typy danych o wielkości 4 bajty (np. `int`, `float`) są wyrównane do granicy 4 bajtów, itd. Oznacza to, że adres w pamięci dla danego pola jest zawsze podzielny przez wielkość tego pola.
2. **Całkowity rozmiar struktury jest wyrównany do największego granicznika używanego przez jakiekolwiek z jej pól**: Na przykład, jeśli struktura ma największe pole typu `int`, to cały rozmiar struktury będzie wielokrotnością wielkości `int`, czyli 4 bajtów.
3. **Kolejność pól ma znaczenie**: Pisanie pól w strukturze od największego do najmniejszego rozmiaru może zminimalizować ilość paddingu.

Wskaźnik do struktury

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

// Definicja struktury
struct Punkt2D {
    float x;
    float y;
};

int main() {
    // Alokacja pamięci dla struktury Punkt2D
    struct Punkt2D *punkt = (struct Punkt2D *)malloc(sizeof(struct Punkt2D));

    // Przypisanie wartości do pól struktury poprzez wskaźnik
    punkt->x = 3.0;
    punkt->y = 4.0;

    printf("Punkt przed przesunieciem: (%.1f, %.1f)\n", punkt->x, punkt->y);
```

Tablice struktur

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

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

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

```c
#include <stdio.h>

struct ksiazka {
    char tytul[50];
    char autor[50];
    int liczba_stron;
    float ocena;
};

void wyswietlKsiazki(struct ksiazka* tablica, int rozmiar) {
    for (int i = 0; i < rozmiar; i++) {
        printf("Tytul: %s\n", tablica[i].tytul);
        printf("Autor: %s\n", tablica[i].autor);
        printf("Liczba stron: %d\n", tablica[i].liczba_stron);
        printf("Ocena: %.2f\n", tablica[i].ocena);
        printf("\n");
    }
}
```

```c
#include <stdio.h>

struct ksiazka {
    char tytul[50];
    char autor[50];
    int liczba_stron;
    float ocena;
};

struct ksiazka znajdzKsiazkeZNajwiekszaLiczbaStron(struct ksiazka* tablica, int ro
    struct ksiazka najwieksza = tablica[0];

    for (int i = 1; i < rozmiar; i++) {
        if (tablica[i].liczba_stron > najwieksza.liczba_stron) {
            najwieksza = tablica[i];
        }
    }

```

typedef

`typedef` to słowo kluczowe w języku C, które pozwala na definiowanie aliasów dla typów danych. W kontekście struktur, `typedef` jest często używane w celu uproszczenia deklaracji zmiennych strukturalnych i funkcji.

Wariant 1 - Użycie `typedef` podczas definiowania struktury:

```c
#include <stdio.h>

typedef struct {
    float x;
    float y;
} Punkt2D;

int main() {
    Punkt2D punkt = {3.0, 4.0};
    printf("Punkt: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
}
```

Wariant 2 - Użycie `typedef` po definiowaniu struktury:

```c
#include <stdio.h>

struct Punkt {
    float x;
    float y;
};

typedef struct Punkt Punkt2D;

int main() {
    Punkt2D punkt = {3.0, 4.0};
    printf("Punkt: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
}
```

Wariant 3 - Użycie `typedef` w kombinacji ze wskaźnikami na strukturę:

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

typedef struct {
    float x;
    float y;
} Punkt2D;

int main() {
    Punkt2D *punkt = (Punkt2D *)malloc(sizeof(Punkt2D));
    punkt->x = 3.0;
    punkt->y = 4.0;
    printf("Punkt: (%.1f, %.1f)\n", punkt->x, punkt->y);
    free(punkt);
    return 0;
}
```

Wariant 4 - Użycie `typedef` w funkcjach przyjmujących struktury jako argumenty:

```c
#include <stdio.h>

typedef struct {
    float x;
    float y;
} Punkt2D;

void wypisz_punkt(Punkt2D p) {
    printf("Punkt: (%.1f, %.1f)\n", p.x, p.y);
}

int main() {
    Punkt2D punkt = {3.0, 4.0};
    wypisz_punkt(punkt);
    return 0;
}
```

Przekazywanie struktur do funkcji

1. Przekazywanie struktury przez wartość:

```c
#include <stdio.h>

struct Punkt2D {
    float x;
    float y;
};

void wypisz_punkt(struct Punkt2D p) {
    printf("Punkt: (%.1f, %.1f)\n", p.x, p.y);
}

int main() {
    struct Punkt2D punkt = {3.0, 4.0};
    wypisz_punkt(punkt); // przekazanie struktury przez wartość
    return 0;
}
```

2. Przekazywanie struktury przez wskaźnik:

```c
#include <stdio.h>

struct Punkt2D {
    float x;
    float y;
};

void przesun_punkt(struct Punkt2D *p, float dx, float dy) {
    p->x += dx;
    p->y += dy;
}

int main() {
    struct Punkt2D punkt = {3.0, 4.0};
    printf("Punkt przed przesunieciem: (%.1f, %.1f)\n", punkt.x, punkt.y);
    przesun_punkt(&punkt, 2.0, 3.0); // przekazanie struktury przez wskaźnik
    printf("Punkt po przesunieciu: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
```

Zwracanie przez funkcję struktury

Przykład zwracania struktury jako zwykłej zmiennej:

```c
#include <stdio.h>

struct Point {
  int x;
  int y;
};

struct Point add_points(struct Point p1, struct Point p2) {
  struct Point result;
  result.x = p1.x + p2.x;
  result.y = p1.y + p2.y;
  return result;
}

int main() {
  struct Point p1 = {1, 2};
  struct Point p2 = {3, 4};
  struct Point sum = add_points(p1, p2);
```

Przykład zwracania struktury przez wskaźnik przekazany jako argument:

```c
#include <stdio.h>

struct Point {
  int x;
  int y;
};

void add_points(struct Point p1, struct Point p2, struct Point *result) {
  result->x = p1.x + p2.x;
  result->y = p1.y + p2.y;
}

int main() {
  struct Point p1 = {1, 2};
  struct Point p2 = {3, 4};
  struct Point sum;
  add_points(p1, p2, &sum);
  printf("sum: (%d, %d)\n", sum.x, sum.y);
```

Przykład zwracania struktury przez wskaźnik przekazany (bezpośrednio):

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

struct Person {
    char *name;
    int age;
};

struct Person *create_person(char *name, int age) {
    struct Person *new_person = malloc(sizeof(struct Person));
    new_person->name = name;
    new_person->age = age;
    return new_person;
}

int main() {
    struct Person *person1 = create_person("John Doe", 30);
    printf("Name: %s, Age: %d\n", person1->name, person1->age);
```

“Wartościowość struktur”

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

struct Punkt3D
{
    int x,y,z;
};

void przepisz(struct Punkt3D tab1[], struct Punkt3D tab2[], int n)
{
    for(int i = 0; i < n; i++)
    {
        tab2[i] = tab1[i];
    }
}

void wyswietl(struct Punkt3D tab[], int n)
{
```

Dziwny warning?

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

struct Punkt3D
{
    int tab[3];
};

void przepisz(struct Punkt3D tab1[], struct Punkt3D tab2[], int n)
{
    for(int i = 0; i < n; i++)
    {
        tab2[i] = tab1[i];
    }
}

void wyswietl(struct Punkt3D tab[], int n)
{
```

Wersja poprawiona

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

struct Punkt3D
{
    int wspolrzedne[3];
};

void przepisz(struct Punkt3D tab1[], struct Punkt3D tab2[], int n)
{
    for(int i = 0; i < n; i++)
    {
        tab2[i] = tab1[i];
    }
}

void wyswietl(struct Punkt3D tab[], int n)
{
```

A gdy mamy wskaźnik?

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

struct Punkt3D
{
    int * wspolrzedne;
};

void przepisz(struct Punkt3D tab1[], struct Punkt3D tab2[], int n)
{
    for(int i = 0; i < n; i++)
    {
        tab2[i] = tab1[i];
    }
}

void wyswietl(struct Punkt3D tab[], int n)
{
```

Trzeba zrobić “głębokie kopiowanie”:

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

struct Punkt3D
{
    int * wspolrzedne;
};

void przepisz(struct Punkt3D tab1[], struct Punkt3D tab2[], int n)
{
    for(int i = 0; i < n; i++)
    {
        tab2[i].wspolrzedne[0] = tab1[i].wspolrzedne[0];
        tab2[i].wspolrzedne[1] = tab1[i].wspolrzedne[1];
        tab2[i].wspolrzedne[2] = tab1[i].wspolrzedne[2];
    }
}

```

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;
  /* ... */
};
```

```c
#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;
}
```

```c
#include <stdio.h>

union Number {
    int i;
    float f;
    double d;
};

int main() {
    union Number num;
    num.i = 10;
    printf("int: %d\n", num.i);
    num.f = 3.14;
    printf("float: %f\n", num.f);
    num.d = 2.718;
    printf("double: %lf\n", num.d);
    printf("int: %d\n", num.i);
    return 0;
```

```c
#include <stdio.h>
#include <stdlib.h>
union Liczba
{
    int a;
    float b;
};
struct Dane
{
    int tp;
    union Liczba zaw;
};
struct Dane wczytaj()
{
    struct Dane temp;
    printf("Jesli chcesz wpisac liczbe calk to wpisz 0,a jesli wymierna to wpisz 1
    scanf("%d", &temp.tp);
    if (temp.tp ==0)
```

Typ wyliczeniowy

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

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

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

enum miasta {OLSZTYN, GDANSK, KRAKOW, WARSZAWA, BYDGOSZCZ};

int main()
{
    enum miasta m1 = OLSZTYN;
    printf("%s\n",m1);
    printf("%d\n",m1);
    printf("%u\n",m1);
    return 0;
}
```

Struktury - prekursor do obiektowości?

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

struct Laptop {
    char model[30];
    float cena;
};

struct Laptop initLaptop(char* model, float cena) {
    struct Laptop nowyLaptop;
    strncpy(nowyLaptop.model, model, sizeof(nowyLaptop.model) - 1);
    nowyLaptop.model[sizeof(nowyLaptop.model) - 1] = '\0';
    nowyLaptop.cena = cena;
    return nowyLaptop;
}

void pokazLaptop(struct Laptop laptop) {
    printf("Model: %s\n", laptop.model);
```

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

typedef struct Ksiazka {
    char tytul[50];
    int liczba_stron;
} Ksiazka;

Ksiazka* initKsiazka(const char *tytul, int liczba_stron) {
    if(strlen(tytul) < 5 || liczba_stron <= 50)
        return NULL;

    Ksiazka* nowa_ksiazka = (Ksiazka*)malloc(sizeof(Ksiazka));
    strcpy(nowa_ksiazka->tytul, tytul);
    nowa_ksiazka->liczba_stron = liczba_stron;

    return nowa_ksiazka;
```

Bibliografia

- Stephen Prata, Jezyk C. Szkoła programowania. Wydanie VI, Wyd. Helion, 2016.
- https://cybersecurity.umcs.lublin.pl/wp-content/uploads/kmazur/PP2017/, dostęp online 1(
- Richard Reese, Wskaźniki w języku C, Wydawnictwo Helion 2014.
- http://marek.piasecki.staff.iiar.pwr.wroc.pl/dydaktyka/skp/W11_wskazniki_na_tablice_wiel(dostęp online 15.04.2023.
- https://pl.wikibooks.org/wiki/C/Typy_z%C5%82o%C5%BCone#Struktury, dostęp online 20.(