**ZŁOŻONE TYPY DANYCH**

**STRUCT**

**struct trojkat{**

**int bokA;**

**int bokB;**

**int bokC;**

#include <stdio.h>

#include <limits.h>

#define MAX\_MARKA\_SIZE 100

struct Telewizor {

char marka[MAX\_MARKA\_SIZE];

int cena;

};

int znajdzNajmniejszaCene(struct Telewizor telewizory[], int rozmiar) {

int najmniejszaCena = INT\_MAX;

for (int i = 0; i < rozmiar; i++) {

if (telewizory[i].cena < najmniejszaCena) {

najmniejszaCena = telewizory[i].cena;

}

}

return najmniejszaCena;

}

int main() {

struct Telewizor telewizory[4] = {

{"Samsung", 1500},

{"LG", 1200},

{"Sony", 1800},

{"Panasonic", 1000}

};

int rozmiar = sizeof(telewizory) / sizeof(telewizory[0]);

int najmniejszaCena = znajdzNajmniejszaCene(telewizory, rozmiar);

printf("Najmniejsza cena: %d\n", najmniejszaCena);

return 0;

}

**};**  
Struktury tworzymy tak, aby działały globalnie.

**struct trojkat a = {4, 7, 9};**  
Tworzy obiekt typu struct trojkat i przypisuje do niego wartości.

**a.bokA + a.bokB + a.bokC;**  
Tak odwołujemy się do właściwości poszczególnego obiektu.

**int foo(struct trojkat a){}**  
Deklarujemy obiekt typu struct w funkcji.

struct Trojkat \*troj2 = (struct Trojkat \*)malloc(sizeof(struct Trojkat));  
Alokacja pamięci dla struktury, tworzy wskaźnik.

**troj2->bokA = 2, troj2->bokB = 9, troj2->bokC = 6;**  
Przypisanie wartości dla właściwości obiektu przy użyciu wskaźnika.

**void foo(struct Trojkat a1, struct Trojkat \*a2){}**  
Deklarujemy wskaźnik na obiekt typu struct Trojkat.

**struct zespolone dodaj(){}**  
Funkcja zwracająca obiekt typu stuct zespolone.

**Funkcja przyjmuje dwa argumenty: tytuł i liczbę stron, i zwraca wskaźnik nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.**

struct Ksiazka\* initKsiazka(char\* tytul, int liczba\_stron){

if(strlen(tytul) < 5 || liczba\_stron <= 50){

return NULL;

}

struct Ksiazka\* ksiazka = malloc(sizeof(struct Ksiazka));

strcpy(ksiazka->tytul, tytul);

ksiazka->liczba\_stron = liczba\_stron;

return ksiazka;

}

**UNION**

**union super\_int{**

**int zmienna1;**

**unsigned int zmienna2;**

**};**

Tworzymy unię globalnie. Umożliwia przechowywanie w jednym miejscu danych różnych typów, ale tylko jednego typu w danym momencie.

union Liczba{

double x;

int y;

};

struct Dane{

int tp;

union Liczba zaw;

};

struct Dane foo(){

struct Dane dane;

printf("Jaka liczbe chcesz wczystac?");

scanf("%i", &dane.tp);

if(dane.tp == 0){

printf("Wpisz liczbe calkowita: ");

scanf("%i", &dane.zaw.y);

} else{

printf("Wpisz liczbe wymierna: ");

scanf("%f", &dane.zaw.x);

}

return dane;

}

#include <stdio.h>

union XYZ {

int a;

char b;

};

int main() {

union XYZ tablica[6];

for (int i = 0; i < 6; i++) {

tablica[i].a = i + 1;

tablica[i].b = 'A' + i;

}

for (int i = 0; i < 6; i++) {

printf("Element %d: a = %d, b = %c\n", i+1, tablica[i].a, tablica[i].b);

}

return 0;

}

**ENUM**

**enum Color { RED, GREEN, BLUE };**   
Typ ENUM

**enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};**

**enum week day;**

**day = Wed;**

Przypisuje zmienną typu enum. Po wyprintowaniu day otrzymamy 2.

**Iteracja po ENUM**

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

int i;

for (i=Jan; i<=Dec; i++)

printf("%d ", i); // wypisze liczby od 0 do 11

#include <stdio.h>

typedef enum {

Samsung,

LG,

Sony,

Panasonic,

Philips,

Toshiba

} Telewizor;

int main() {

Telewizor telewizory[6] = {Samsung, LG, Sony, Panasonic, Philips, Toshiba};

for (int i = 0; i < 6; i++) {

printf("Telewizor %d: ", i+1);

switch (telewizory[i]) {

case Samsung:

printf("Samsung\n");

break;

case LG:

printf("LG\n");

break;

case Sony:

printf("Sony\n");

break;

case Panasonic:

printf("Panasonic\n");

break;

case Philips:

printf("Philips\n");

break;

case Toshiba:

printf("Toshiba\n");

break;

default:

printf("Nieznana marka\n");

break;

}

}

return 0;

}