7. Struktura, union, enum

# Struktura

Struktury (structures) nám umožňují popsat nový datový typ, který se bude skládat z jednoho či více tzv. členů (members)1. Každému členu musíme určit jeho jméno a datový typ

Pokud vytvoříme proměnnou datového typu struktury, tak překladač naalokuje paměť pro všechny členy této struktury.

Členové struktury budou v paměti uloženi ve stejném pořadí, v jakém byli popsáni při deklaraci struktury. Neznamená to ovšem, že musí ležet hned za sebou! Překladač se může rozhodnout mezi členy struktury v paměti vložit mezery (tzv. padding) kvůli urychlení provádění programu.

## Deklarace

typedef struct <název struktury> {

<datový typ prvního členu> <název prvního členu>;

<datový typ druhého členu> <název druhého členu>;

<datový typ třetího členu> <název třetího členu>;

...

} <datový typ>;

struct {

int vek;

char[] jmeno;

}ZVIRE;

## Inicializace

struct ZVIRE kocka = {1, “Miki“};

struct ZVIRE kocka = {.vek = 1, .jmeno = “Miki“}; //když si nepamatujeme pořadí prvků

## Přístup ke členům

kocka.vek;

Při pointerech : \* má menší prioritu jak . :

(\*kocka).vek; == kocka->vek;

# Union

Jednotlivé prvky se překrývají, nabírá velikost pouze největšího prvku. PO přiřazení hodnoty jednomu prvku lze pracovat už pouze s ním.

typedef union {

char[] spz;

int pocetKol;

} VOZIDLO;

# Enum

Výčtový datový typ. Náhradou za symbolické konstanty. Vnitřně jsou reprezentovány čísly, zvenku pouze názvem. Hodnoty mohou být kladné i záporné.

typedef enum {

MODRA, //0

ZELENA, //1

CERVENA //2

}BARVA;

typedef enum{

MODRA = 2,

ZELENA = -52,

CERVENA = 1

}BARVA;