

Procedurálne programovanie

slido

slido.com
4198161
PrPr – P8

Ján Zelenka
Ústav Informatiky
Slovenská akadémia vied



Obsah prednášky

- 1. Vlastný dátový typ**
- 2. Štruktúry**

Spätná väzba: <https://forms.gle/6q5D2G6UwrtimXEx9>

Dátové typy

- jednoduché dátové typy (char, int, double...)
- pole jednoduchých dátových typov
- vytváranie vlastných dátových typov:
 - typedef
 - struct
 - enum
 - union

Vlastný dátový typ

slido

slido.com
4198161
PrPr – P8

Operátor typedef

- zavedenie nového dátového typu
- skrátenie zápisu komplikovaných typov

```
typedef float *P_FLOAT;
```

P_FLOAT je ukazovateľ na typ **float**

Operátor typedef

```
int *p_i, **p_p_i;
```

`p_i` - ukazvateľ na `int`
`p_p_i` - ukazvateľ na
ukazovateľ na `int`

je ekvivalentné

```
typedef int *P_INT;  
typedef P_INT *P_P_INT;
```

```
P_INT p_i;  
P_P_INT p_p_i;
```

`pole` - ukazvateľ na
celočíselné pole veľkosti 10

```
P_INT pole;  
pole = (P_INT) malloc(10*sizeof(int));
```

Operátor typedef

Príklady definícií

```
int i;          - i je typu int
float *y;       - y je ukazovateľ na typ float
double *z();    - z je funkcia vracajúca
                  ukazovateľ na double
int (*v)();     - ukazovateľ na funkciu
                  vracajúcu int
int *(*v)();    - ukazovateľ na funkciu
                  vracajúcu ukazovateľ na int
```

ukazovateľ na funkciu
vracajúcu **double**

```
typedef double (*P_FD)();
P_FNC  funkcie[10];
```

pole 10
ukazovateľov

Štruktúry



slido.com
4198161
PrPr – P8

Štruktúra: Motivácia

- Komplikovanejší dátový typ s atribútmi rôznych typov
 - bod (súradnice), zamestnanec, prvý projekt (modul)...
- Komplikovaná implementácia
 - premenná pre každý atribút
- Predávanie pomocou funkcie
 - narastá počet parametrov funkcie
 - globálne premenné
- Viacero inštancií
 - pole
- Priebežné pridávanie nových inštancií

Štruktúra

- heterogénny dátový typ (zložená z prvkov rôznych typov), ku ktorému je možné pristupovať ako k jednému objektu.
 - pole je homogénny dátový typ
- veľkosť premennej typu struct zodpovedá súčtu veľkostí všetkých položiek (+ zarovnanie)

```
struct {  
    polozka_1;  
    ...  
    polozka_n;  
};
```

dá sa definovať
rôznymi spôsobmi

Definícia štruktúry

Základný spôsob

- štruktúra nie je pomenovaná,
- nedá sa inde v programe použiť
- dajú sa použiť len definované premenné

```
struct {  
    int vyska;  
    float vaha;  
} pavol, jan, karol;
```

Definícia štruktúry

modifikácia základného spôsobu

- štruktúra je pomenovaná
- dá sa využiť aj inde v programe

```
struct miery {  
    int vyska;  
    float vaha;  
} pavol, jan, karol;
```

Definícia štruktúry

podobne ako predchádzajúci spôsob

- definícia štruktúry a premenných je oddelená od definície premenných (ktoré sa môžu robiť viackrát)

```
struct miery {  
    int vyska;  
    float vaha;  
};  
struct miery pavol;  
struct miery jan, karol;
```

Definícia štruktúry

Definícia nového typu (typedef)

- štruktúra nie je pomenovaná, pomenovaný je typ
- typ sa dá použiť na definíciu premenných, pretypovanie...

```
typedef struct {  
    int vyska;  
    float vaha;  
} MIERY;  
MIERY pavol, jan, karol;
```

nebolo použité "**struct**" pri
deklarácii premennej

Definícia štruktúry

modifikácia predchádzajúceho spôsobu

- štruktúry aj typ je pomenovaná (v tomto prípade to nie je potrebné, ale neskôr to budeme potrebovať)

```
typedef struct miery {  
    int vyska;  
    float vaha;  
} MIERY;  
MIERY pavol, jan, karol;
```

odporúča sa pomenovať typ aj štruktúru rovnako, odlíšiť ich len veľkosťou písma

Prístup k položkám štruktúry

bodková notácia

```
typedef struct miery {  
    int vyska;  
    float vaha;  
} MIERY;  
MIERY pavol, jan, karol;  
  
pavol.vyska = 182;  
karol.vaha = 62.5;  
jan.vyska = pavol.vyska;
```

často sa používa pole štruktúr:

```
MIERY ludia[100];  
  
ludia[50].vyska = 156;  
  
ludia[0] = ludia[50];
```

v ANSI C sa dá urobiť

Pole v štruktúre

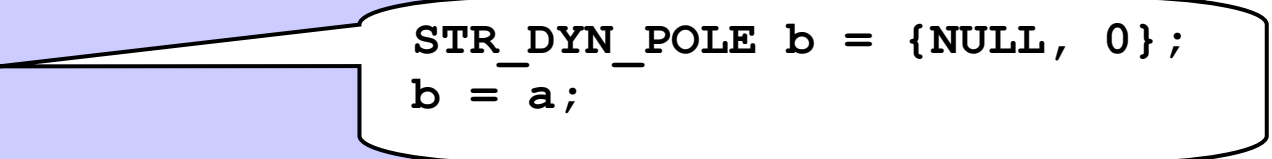
```
typedef struct str_pole{  
    int pole[10];  
} STR_POLE;  
  
void main()  
{  
    STR_POLE a, b;  
    a.pole[0] = 5;  
    b = a;  
}
```

takto sa dá použiť štruktúra
na to, aby sa dalo naraz
skopírovať celé pole

Pole v štruktúre

```
typedef struct str_dyn_pole{
    int *p_pole;
    int velkost;
} STR_DYN_POLE;

void main()
{
    STR_DYN_POLE a = {NULL, 0};
    a.velkost = 10;
    a.p_pole = (int*)malloc(a.velkost*sizeof(int));
    for(int i=0; i<a.velkost; i++)
        a.p_pole[i] = i;
    free(a.p_pole);
}
```



```
STR_DYN_POLE b = {NULL, 0};
b = a;
```

Kopírovanie štruktúr

- obsah štruktúry je možné kopírovať pomocou operátora priradenia
- kopíruje sa obsah jednotlivých položiek
 - primitívny dátový typ (int, double, char...)
 - skopíruje sa hodnota
 - pole (float hodnoty[10])
 - skopíruje sa hodnota jednotlivých prvkov poľa
 - ukazovateľ
 - skopíruje sa adresa (uložená v ukazovateli)
- kopírovanie celej pamäti so štruktúrou
 - napr. memcpy()

Plytká vs hlboká kópia

- Kopírovaná položka je ukazovateľ
 - skopíruje sa hodnota ukazovateľa (nie obsah odkazovanej pamäte)
 - pôvodná aj nová štruktúra ukazujú na to isté miesto
 - navzájom si prepisujú hodnoty
 - ak jedna z nich bude uvoľnená, druhá ukazuje na neplatné miesto v pamäti
 - **Kópia je plytká**
- **Hlboká kópia**
 - vytvorenie samostatného pamäťového bloku
 - explicitne cez príslušné funkcie
 - `malloc()` + `memcpy()`

Štruktúry a ukazovatele

Použitie:

- štruktúra v dynamickej pamäti
- štruktúra vo funkcii

```
typedef struct {  
    char meno[30];  
    int rocnik;  
} STUDENT;  
  
STUDENT s, *p_s;
```

***p_s** - ukazovateľ na štruktúru, nemá pridelené miesto v pamäti, preto je potrebné alokovať mu pamäť.

```
p_s = (STUDENT *) malloc(sizeof(STUDENT));
```

alebo nastaviť ukazovateľ na inú pamäť

```
p_s = &s;
```

Štruktúra vo funkcii

```
STUDENT *vytvor(void) {  
    STUDENT *p_student;  
    p_student = (STUDENT *) malloc(sizeof(STUDENT));  
    return p_student;  
}  
  
void zapis(STUDENT student, int r) {  
    student.rocnik = r;  
}  
  
void zapis2(STUDENT *student, int r) {  
    (*student).rocnik = r;  
}  
  
int main(void) {  
    STUDENT s = {"Peter Kovac", 0};  
    zapis(s, 1);  
    zapis2(&s, 1);  
    return(0);}
```

```
typedef struct {  
    char meno[30];  
    int rocnik;  
} STUDENT;
```

Hodnotou -> zmena sa
nepropaguje

Neefektívne -> vytvorí sa
kópia štruktúry na
zásobníku (kopírovanie je
časovo náročné, a
štruktúra zaberá zbytočne
pamäť na zásobníku)

Ukazovateľom

Štruktúry a ukazovatele

```
typedef struct {  
    char meno[30];  
    int rocnik;  
} STUDENT, *P_STUDENT;
```

```
typedef struct {  
    char meno[30];  
    int rocnik;  
} STUDENT;
```

definícia typu ukazovateľa
na štruktúru

```
STUDENT s;  
P_STUDENT p_s;  
p_s = (P_STUDENT) malloc(sizeof(STUDENT));
```

```
s.rocnik = 2;  
(*p_s).rocnik = 3;  
p_s->rocnik = 4;
```

prístup k štruktúre
pomocou ukazovateľa

Prístup do štruktúry pomocou ukazovateľa

```
typedef struct {  
    char meno[30];  
    int rocnik;  
} STUDENT;
```

```
STUDENT s, *p_s=&s;  
  
s.rocnik = 2;  
  
(*p_s).rocnik = 3;  
p_s->rocnik = 5;
```

chybné, lebo . má väčšiu
prioritu ako *

teda tam, kam ukazuje
`p_s.rocnik` (adesa 3) priradiť
hodnotu 4

Dynamická alokácia štruktúr

- pomocou funkcií pre dynamickú alokáciu pamäte (napr. malloc)

Jedna premenná

```
STUDENT *p_student = NULL;  
p_student = malloc(sizeof(STUDENT));
```

Pole ukazovateľov

```
STUDENT *students[10];  
students[0] = malloc(sizeof(STUDENT));
```

Statická a dynamická štruktúra

Statická štruktúra:

- lokálna premenná
- alokovaná staticky (na zásobník)

```
KOMP scitaj(KOMP a, KOMP b);  
c.re = a.re + b.re;  
z = scitaj(x, y);
```

```
typedef struct {  
    double re, im;  
} KOMP;  
  
KOMP x, y, z;
```

vhodné pre malé štruktúry

```
void scitaj(KOMP *p_a, KOMP *p_b, KOMP *p_c);  
p_c->re = p_a->re + p_b->re;  
scitaj(&x, &y, &z);
```

Prístupy je možné kombinovať.

Statická a dynamická štruktúra

Dynamická
štruktúra:

```
typedef struct {  
    double re, im;  
} KOMP;  
  
KOMP *p_x = malloc(sizeof(KOMP));  
KOMP *p_y = malloc(sizeof(KOMP));  
KOMP *p_z = malloc(sizeof(KOMP));
```

```
void scitaj(KOMP *p_a, KOMP *p_b, KOMP *p_c);  
p_c->re = p_a->re + p_b->re;  
scitaj(p_x, p_y, p_z); //bez &
```

je možné použiť aj prvý prístup, alebo ich kombináciu.

Operátor -> a operátor .

Operátor . sa používa na prístup k položkám štruktúry

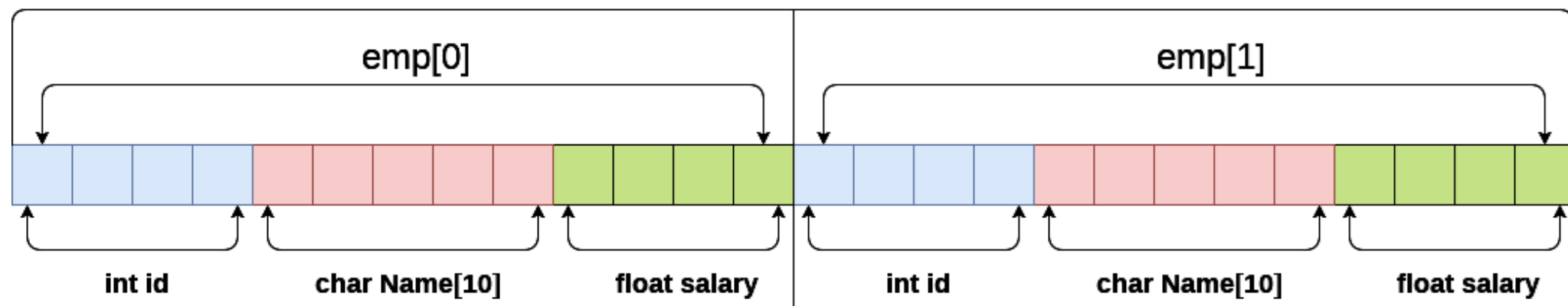
```
STUDENT student;  
student.rocnik = 1;
```

Ak je štruktúra alokovaná dynamicky, pracujeme s ukazovateľom na danú štruktúru

```
STUDENT *p_student = (STUDENT*)malloc(sizeof(STUDENT));
```

- operátor . sa nedá priamo použiť
- musíme najprv dereferencovať ukazovateľ `(*p_student).rocnik = 1;`
- Zjednodušenie pomocou operátora ->
 - `(*p_struct).atribut == p_struct->atribut`
 - `p_student->rocnik = 1;`

Pole štruktúr



```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

Podobne ako viacrozmerne pole, aj pole štruktúr je možné alokovať aspoň 4 rôznymi spôsobmi

<https://www.javatpoint.com/array-of-structures-in-c>

Vymenovaný typ - enum

slido

slido.com
4198161
PrPr – P8

Vymenovaný typ

- premenná s obmedzeným rozsahom hodnôt
- *enumerate type* - zoznam symbolických konštánt, ktoré sú zvyčajne vzájomne závislé
- zvyšuje čitateľnosť programu

```
typedef enum {  
    MODRA, CERVENA, ZELENA, ZLTA  
} FARBY;
```

```
FARBY c, d;  
c = MODRA;  
d = CERVENA;
```

tu nie je
bodkočiarka

ak nepriradíme konštantám
hodnoty, implicitne sú 0, 1, 2, ...

Príklad: boolovské hodnoty

```
typedef enum {  
    FALSE, TRUE  
} BOOLEAN;  
  
if (isdigit(c) == FALSE)
```

nie je nutné
ani definovať
premennú

```
BOOLEAN dobre;
```

premenná môže byť
definovaná

Explicitná a implicitná inicializácia

```
typedef enum {  
    MODRA = 0,  
    CERVENA = 4,  
    ZELENA = 2,  
    ZLTA  
} FARBY;
```

najhoršie možné: čiastočne explicitné, čiastočne implicitné (ZLTA bude mať hodnotu 3 -> o 1 viacej ako predchodca)

Explicitná inicializácia:

- zoradiť podľa veľkosti
- inicializujeme všetky prvky poľa

Výpis mena položky

```
typedef enum {  
    MODRA, CERVENA, ZELENA, ZLTA  
} FARBY;  
...  
FARBY c = MODRA;  
printf("Farba: %s \n", c);  
printf("Farba: %d \n", (int) c);  
switch (c) {  
    case MODRA:  
        printf("Modra farba.\n");  
        break;  
    ...  
}
```

chyba!

s pretypovaním:
výpis hodnôt

výpis mien položiek:
pomocou **switch**

Výpis mena položky: pomocou poľa

```
typedef enum {  
    MODRA, CERVENA, ZELENA, ZLTA  
} FARBY;  
  
FARBY farba = MODRA;  
char *nazvy[] = { "Modra", "Cervena", "Zelena", "Zlta" };  
printf("Farba: %s \n", nazvy[farba]);
```



len pre neinicializované vymenované typy

Union



slido.com
4198161
PrPr – P8

Union

Dátový typ:

- vyhradí sa pamäť pre najväčšiu položku
- všetky položky sa prekrývajú (v danom okamihu iba jedna položka)

```
typedef union {  
    char c;  
    int i;  
    float f;  
} ZIF;  
ZIF a, *p_a = &a;
```

vyhradí sa pamäť o veľkosti najväčšieho prvku

```
a.c = '#';  
p_a->i = 1;  
a.f = 2.3;
```

premazávajú sa hodnoty

Union neposkytuje informáciu o typu prvku, ktorý bol naposledy do neho uložený!

Union vložený do štruktúry

```
typedef enum {  
    ZNAK, CELE, REALNE  
} TYP;
```

vymenovaný typ: slúži na rozlíšenie typov

```
typedef union {  
    char c;  
    int i;  
    float f;  
} ZIF;
```

union: umožňuje uchovávať znak, celé a reálne číslo

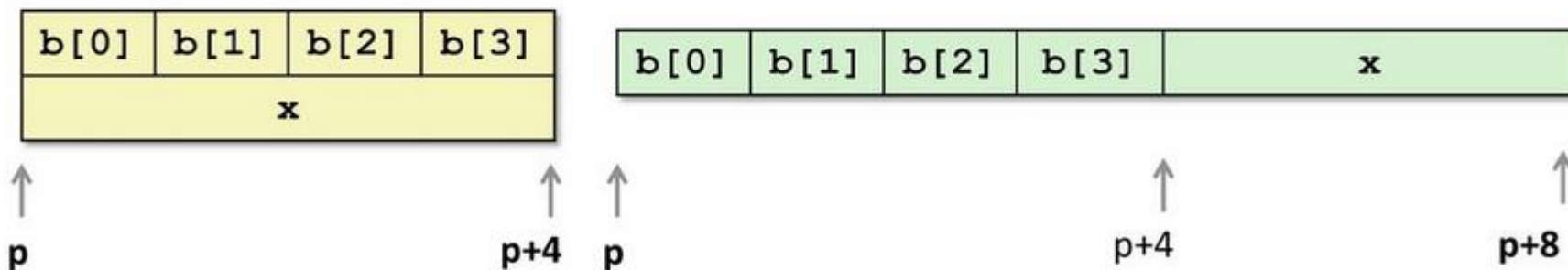
```
typedef struct {  
    TYP typ;  
    ZIF položka;  
} ZN_INT_FL;
```

štruktúra: obsahuje informáciu o type položky a samotnú položku

Uloženie unionu v pamäti

```
union my_union {  
    unsigned char b[4];  
    int x;  
}  
union my_union U;
```

```
struct my_struct {  
    unsigned char bytes[4];  
    int x;  
};  
struct my_struct S;
```



Opätovné využitie existujúcej položky

- reinterpretácia bitov
- veľké polia

Randal E. Bryant, David R. O'Hallaron
- Computer Systems_A
Programmer's Perspective (2015,
Pearson)

Ďakujem vám za pozornosť!

Spätná väzba: <https://forms.gle/6q5D2G6UwrtimXEx9>