KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Jazyk C
Paměť II

Mgr. Markéta Trnečková, Ph.D.

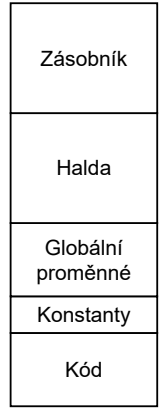
Jazyk C

„Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.“

(Linus Torvalds – Linux)

Paměť
Správa paměti

- **Statická alokace**
- **Dynamická alokace**
  - Na zásobník
  - Na haldu



Funkce pro práci s pamětí

- **Knihovna**: `stdlib`
- **Přidělení paměti**: `malloc()`
- **Uvolnění paměti**: `free()`
- **Další funkce**:
  - `calloc()`
  - `realloc()`

Funkce pro práci s pamětí

Funkce `malloc()`

- Alokace paměti
- `void *malloc(size_t size)`
- `int *ptr_i = malloc(sizeof(int));`
- `ptr_i = (int *) malloc(sizeof(int));`

Funkce pro práci s pamětí

Funkce `calloc()`

- Alokace paměti + inicializace

```
ptr = calloc(n, sizeof(int));
ptr = (int *) malloc(n * sizeof(int));
```

Funkce pro práci s pamětí

Funkce `realloc()`

- Změna velikosti alokované paměti

```
void *realloc(void *adresa, size_t velikost);
```

Funkce pro práci s pamětí

- Pracujeme s ukazateli (ale i `[ ]`)
- Je však rozdíl mezi statickými a dynamickými poli
- **Příklad**: `alokace.c`

Funkce pro práci s pamětí

Další

- **Knihovna**: `string`
- `memcpy()`
- `memset()`
- `memmove()`

Vícerozměrná pole

```
typ jmeno[v1]...[vn]
```

Příklad

```
/* definice dvourozmerneho pole */
int pole[2][3];

/* definice ctyrrozmerneho pole */
int pole2[2][3][4][5];
```

Vícerozměrná pole – inicializace

Inicializace

Příklad (vp_init.c)

```
/* jednorozmerne pole */
int pole1[4] = {1, 2, 3, 4};

/* dvourozmerne */
int pole2[2][3] = {{1, 2, 3}, {1, 2, 3}};

/* trojrozmerne */
int pole3[2][3][4] = {{{1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}},
                      {{5, 6, 7, 8}, {5, 6, 7, 8}, {5, 6, 7, 8}}};
```

Vícerozměrná pole

Uložení v paměti

`x[2][3]`

adresa	100	102	104	106	108	110	112
prvek	`x[0][0]`	`x[0][1]`	`x[0][2]`	`x[1][0]`	`x[1][1]`	`x[1][2]`	volno

Vícerozměrná pole

Uložení v paměti

`x[2][3]`

adresa	100	102	104	106	108	110	112
prvek	`x[0][0]`	`x[0][1]`	`x[0][2]`	`x[1][0]`	`x[1][1]`	`x[1][2]`	volno
	`x[0]`			`x[1]`			
	`x`						

Příklad

Jaký je výsledek:

```
x + 1
x[0] + 1
```

Vícerozměrná pole

Alokace

1. Alokujeme pole pointerů o `m` prvcích.
2. Alokujeme `m` jednorozměrných polí o `n` prvcích, přičemž pointery uložíme do pole alokovaného v prvním kroku.

Příklad

```
/* 1. krok */
int **pole2d = malloc(m * sizeof(int *));

/* 2. krok */
for (i = 0; i < m; i++)
    pole2d[i] = malloc(n * sizeof(int));
```

Vícerozměrná pole
Dealokace

1. Dealokujeme `m` jednorozměrných polí.
2. Dealokujeme pole pointerů.

Příklad

```
/* 1. krok */
for (i = 0; i < m; i++){
    free(pole2d[i]);
    pole2d[i] = NULL;
}

/* 2. krok */
free(pole2d);
pole2d = NULL;
```

Struktury

- nový datový typ `struct jmeno`

```
struct jmeno{
 polozka_1;
 polozka_2;
 ...
 polozka_n;
};
```

Struktury

- `struct jmeno{ ... } promenna_1, promenna_2, ..., promenna_n;`
- `struct { ... } promenna_1, promenna_2, ..., promenna_n;`
- `struct jmeno{ ... };`

  `struct jmeno promenna_1, promenna_2, ..., promenna_n;`
- ```
  typedef struct{
   polozka_1;
   polozka_2;
   ...
   polozka_n;
  } jmeno;
  ```
  ```

Struktury
Inicializace

- `struct jmeno promenna = {h1, ..., hn};`
- `struct jmeno promenna = {.polozka_i = hi,`
  `.polozka_j = hj, ...};`

Přístup k prvkům

- Operátor `.`
- `promenna.polozka1`

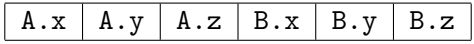
Příklad (structusecka.c)

```
struct bod{
 float x;
 float y;
 float z;
};

struct usecka{
 struct bod A;
 struct bod B;
};
```

`usecka1.A.x`

Struktury



Příklad (structsizeof.c)

```c
#include <stdio.h>
struct struktura{
    float a;
    char b;
    int c;
};

int main(){
   printf("Velikost struktury: %i\n", sizeof(struct struktura));
   printf("Velikost polozek: %i\n", sizeof(float) + sizeof(int)
                                  + sizeof(char));
   return 0;
}
```

Struktury
Velikost

Příklad

```
struct data{
    struct data d;  /* chyba */
    ...
}
```

Struktury
Ukazatel

```
typedef struct{
    polozka1
    ...
    polozkan
} JMENO;

JMENO s, *p_s;


p_s = (JMENO *) malloc(sizeof(JMENO));


p_s = &s;
```

Struktury

Ukazatel

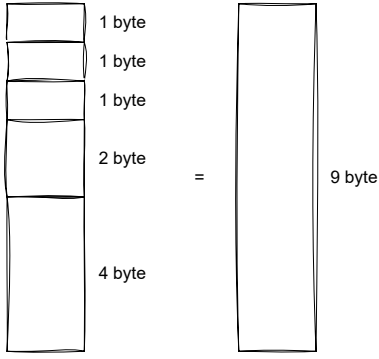
```
JMENO s, *p_s = &s;
```

Příklad

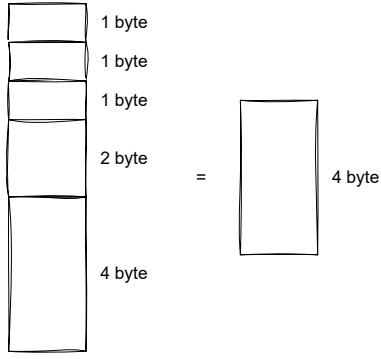
```
/* pomoci jmena struktury */
s.polozka1;

/* pomoci pointeru p_s komplikovane */
(*p_s).polozka1;

/* pomoci pointeru p_s jednoduseji */
p_s->polozka1;

/* tento zapis je chybny, protoze operator . ma prednost pred
   operatorem dereference *  */
*p_s.polozka1;
```

Struktury, uniony

Struktura



Union



Uniony

```
typedef union{
    polozka_1;
    polozka_2;
    ...
    polozka_n;
} nazev_union;
```

Příklad

```
typedef struct{
    int info;
    nazev_union x;  /* union */
} nazev_struktury;
```

Příklad: `union.c`

Cvičení

Máme sejf definovaný následujícím zdrojovým kódem.

Příklad

```c
#include <stdio.h>

typedef struct{
    char* popis;
    float hodnota;
} lup;

typedef struct{
    lup *lup;
    char *sekvence;
} kombinace;
```

Příklad

```c
typedef struct{
    kombinace cislo;
    char *vyrobce;
} sejf;

int main(){
    lup zlato = {"ZLATO!", 1000000.0};
    kombinace cislo = {&zlato, "1234"};
    sejf s  = {cislo,"SEJF2021"};

    return 0;
}
```

Jakou kombinaci následujících částí je potřeba zadat, abyste získali slovo `"ZLATO!"`?(Z každého sloupce vyberte jednu z možností.)

	.	s	+	lup	.	hodnota
s	->	cislo	.	popis	-	lup
	:	lup	->	hodnota	->	popis
	-	zlato	-	sekvence	+	zlato

Cvičení

1. Vytvořte union a strukturu se stejnými položkami a pomocí `sizeof()` zjistěte jejich velikost.
2. Napište program, který pro celočíselné argumenty m a n (zadané od uživatele) alokuje dvourozměrné pole o velikosti $m \times n$. Prvky pole budou reprezentovat násobky. Tedy prvek pole na indexu i, j bude roven $i \cdot j$. Toto pole vypište.
3. Definujte strukturu pro reprezentaci zlomku. Pro dva zadané zlomky vypište jejich součet, rozdíl a součin.
4. Napište program, který spojí dva textové řetězce (zadané přímo ve funkci `main()`). Ve funkci vytvořte nový řetězec a ten pak vypište.
5. Napište program, který porovná dva textové řetězce a vypíše, který z nich je menší (menší ve smyslu lexikografického uspořádání.), případně, že jsou shodné. Při práci s textovými řetězci používejte výhradně ukazatele, operátor dereference a pointerovou aritmetiku.