

Jazyk C

Paměť II

Mgr. Markéta Trnečková, Ph.D.



Palacký University, Olomouc

Zásobník

Halda

Globální
proměnné

Konstanty

Kód



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



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

Funkce malloc()



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



- Alokace paměti

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



- Změna velikosti alokované paměti

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



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


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

Example

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



Example

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



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[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						



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

Example

```
/* 1. krok */  
int **pole2d = malloc(m * sizeof(int *));  
  
/* 2. krok */  
for (i = 0; i < m; i++)  
    pole2d[i] = malloc(n * sizeof(int));
```



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

Example

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

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



- Struktura
- Union

- nový datový typ `struct jmeno`

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


- `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;



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



- Operátor .
- `promenna . polozka1`

Example

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

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

`usecka1.A.x`

A.x	A.y	A.z	B.x	B.y	B.z
-----	-----	-----	-----	-----	-----

Example

```
#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;
}
```

Example

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



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

```
JMENO s, *p_s;
```

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

```
p_s = &s;
```



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

Example

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

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

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

```
/* tento zapis je chybný, pretože operator . ma prednosť pred  
   operatorm dereferencie * */  
*p_s.polozka1;
```

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

Example

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


Máme sejf definovaný následujícím zdrojovým kódem.

Example

```
#include <stdio.h>

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

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

Example

```
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.)

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



- 1 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.
- 2 Definujte strukturu pro reprezentaci zlomku. Pro dva zadané zlomky vypište jejich součet, rozdíl a součin.
- 3 Napište program, který spojí dva textové řetězce (zadané přímo ve funkci `main()`). Ve funkci vytvořte nový řetězec a ten pak vypište.
- 4 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.