Správa paměti

SSPŠ
PVA4
2025 / 2026
Ing. Matěj Cajthaml

Správa paměti

- RAM
- Přiřazování bloků paměti
- Spravuje OS, různé "úrovně"

- Někdy moc práce, v rámci high-level jazyků existuje Garbage Collector
- Pokud existuje v paměti něco, co se nevyužívá, smaže se to

- Některé jazyky nemají nic takového - např. C
- Přímo voláme OS, které nám vytvoří blok souvislé paměti
- Velká zodpovědnost

Proč manuálně?

- Absolutní kontrola
- Žádné nečekané záseky
- Minimální paměťová režie navíce

- Ale vysoké riziko kritických chyb

Data

ZÁSOBNÍK

Rychlá, spravuje se automaticky, řeší aktuální volání funkce. Platné proměnné v scope. Omezená velikost a riziko stack overflow.

HALDA

Pro velké data a objekty, které žijí i mimo konec funkce. Velikost omezena operační pamětí.
Nutná manuální správa.

Funkce v C

- malloc(size): alokuje daný počet bytů paměti
- calloc(count, size): alokuje paměť a vyčistí ji
- realloc(ptr, size): změní velikost již existujícího bloku paměti
- free(ptr): uvolní paměť

Ukázka / 1

```c
int *array = (int*) malloc(10 * sizeof(int));

if (array == NULL) {
    printf("Nedostatek paměti!\n");
    return 1;
}

array[0] = 42;

free(array);
array = NULL;
```

Časté problémy

- Memory leak:
- zapomeneme uvolnit alokovanou paměť

- Dangling pointer:
- použití ukazatele na paměť, která již byla uvolněna

- Double free
- Buffer overflow

Pravidla pro psaní bezpečného kódu

- Pravidlo 1:1
- Vynulování ukazatelů
- Kontrola úspěšnosti
- Jasné vlastnictví

Ukázka / 2

```c
#include <stdio.h>

struct Player {
    int hp;
    int score;
};

int main() {
    struct Player *player = (struct Player*) malloc(sizeof(struct Player));

    if (player != NULL) {
        player->hp = 100;
        player->score = 0;
        printf("Hp: %d", player->hp);
    }
```

OTÁZKA

Proč prostě nepředáváme struct jen jako hodnotu?

Kde chyba?

```
1  void process(int size) {
2      int *array = (int*) malloc(size * sizeof(int));
3
4      if (size <= 0) {
5          printf("Chybná velikost!\n");
6          return;
7      }
8
9      array[0] = 10;
10     free(array);
11     array[0] = 20;
12 }
```

Řetězce

- V C je opravdu string jen pole znaků
- A končí znakem \0
- Vždy tedy musíme počítat size+1

Ukázka / 3

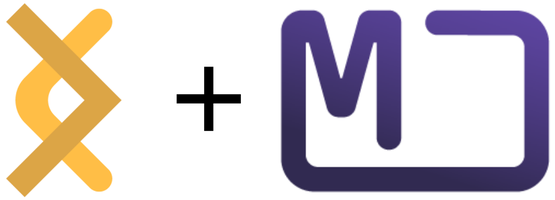
```c
#include <string.h>

char *text = (char*) malloc(5 * sizeof(char));

strcpy(text, "Ahoj");

free(text);
```

PRÁCE

Naprogramujte v C nafukovací pole. Ze vstupu budete číst data a postupné je přidávat. Pole začne na velikosti např. 4, a potom se bude zvětšovat dle potřeby.

Děkuji za pozornost

matej.cajthaml@ssps.cz
ssps.cajthaml.eu



Vytvořeno pomocí aplikace Materialist