Dynamická alokace paměti

IZP-cv07

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

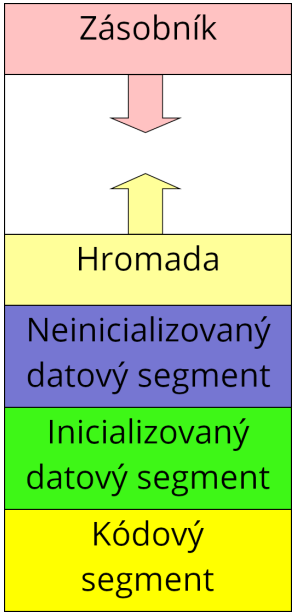
9. listopadu 2023

Dynamická alokace paměti

Zásobník (stack) obsahuje automaticky alokované proměnné které na konci funkce přestávají existovat:

- Přístup k neexistující proměnné způsobí havárii programu (neplatný přístup do paměti).

```
1  int* foo()        //definice funkce
2  {
3      int a = 10; //automaticky alokovane cislo "a"
4      return &a;  //vracime adresu cisla "a"
5  }
6
7  int main()        //zacatek programu
8  {
9      int* x;       //ukazatel "x"
10     x = foo();    //do "x" ukladame adresu cisla "a"
11     printf("%i\n", *x); //CHYBA -- vypis promenne
12                         //ktera jiz neexistuje
13     return 0;     //konec programu
14 }
```



Dynamická alokace paměti

Hromada (heap) obsahuje dynamicky alokované proměnné které alokujeme voláním funkce malloc z knihovny stdlib.h:

- Vstupem je velikost alokované paměti, výstupem je její adresa.
- Pokud alokace paměti selhala funkce vrátí konstantu NULL.

```
1  int* alokuj()              //definice funkce
2  {
3      int* b;                //ukazatel "b"
4      b = malloc(sizeof(int)); //alokujeme misto pro jedno cele cislo
5      *b = 10;               //jeho hodnotu nastavujeme na 10
6      return b;              //vracime ukazatel "b"
7  }
8
9  int main()                 //zacatek programu
10 {
11     int* y;                //ukazatel "y"
12     y = alokuj();          //do "y" ukladame adresu cisla "b"
13     printf("%i\n", *y);    //vypis cisla (probehne bez problemu)
14     return 0;              //konec programu
15 }
```

Uvolňování paměti

Dynamicky alokované proměnné budou v paměti existovat tak dlouho dokud je neuvolníme voláním funkce free z knihovny stdlib.h:

- Vstupem funkce je adresa nějaké dynamicky alokované paměti.
- Veškerou alokovanou paměť **MUSÍME** před koncem programu také uvolnit.

```
1  int* x;                          //ukazatel na cele cislo
2  x = malloc(sizeof(int));         //alokujeme misto pro jedno cele cislo
3
4  if (x != NULL)                   //pokud alkoace uspela
5  {
6      printf("Alokace pameti uspela\n"); //vypis
7      free (x);                    //uvolnujeme alokovanou pamet
8      return 0;                    //program konci bez chyby
9  }
10 else                             //jinak
11 {
12     fprintf(stderr, "Alokace pameti selhala\n"); //chybovy vypis
13     return 1;                    //program konci s chybou
14 }
```

Vyzkoušejte si:

- Napište program který opakovanou alokací stále rostoucího množství paměti zjistí kolik místa má program maximálně k dispozici.
- Po každém pokusu o alokaci vypište jestli alokace uspěla nebo selhala.
- Množství alokované paměti zvyšujte po MebiBajtech (MiB).
- Nezapomeňte že po každé úspěšné alokaci musíte paměť také uvolnit.

Například:

- Alokace 1 MiB uspela
  Alokace 2 MiB uspela
  ...
  Alokace 1878 MiB uspela
  Alokace 1879 MiB selhala

Alokace polí

Hodnota pole (bez hranatých závorek) je adresou jeho prvního prvku:

- K ukazateli se tedy můžeme chovat jako by to bylo pole.
- Pro pole alokujeme počet krát velikost prvku bytů.

```
1  int main()                                        //zacatek programu
2  {
3      int delka = 5;                                //pocet prvku pole
4      int* pole = malloc(delka * sizeof(int)); //alokujeme pole cisel
5
6      if (pole != NULL)                             //pokud alokace uspela
7      {
8          for (int i = 0; i < delka; i++)           //pro vsechny prvky pole
9              scanf("%i", &pole[i]);                //nacitame vstup
10
11         for (int i = 0; i < delka; i++)           //pro vsechny prvky pole
12             printf("pole[%i] = %i\n", i, pole[i]); //vypis
13
14         free(pole);                               //uvolnujeme alokovanou pamet
15     }
16     return 0;                                     //konec programu
17 }
```

Vyzkoušejte si:

- Napište funkci soucet která alokuje nové pole celých čísel, a naplní ho součtem dvou vstupních polí stejné délky.
- Napište funkci konkatenace která alokuje nové pole znaků, a bez použití funkce strcat ho naplní konkatenací dvou řetězců.

```
1 int* soucet(int delka, int* poleA, int* poleB);
2 char* konkatenace(char* strA, char* strB);
```

- Ve funkci main vytvořte dvě pole čísel, předejte je funkci soucet a vypište výsledek; vytvořte dva řetězce, předejte je funkci konkatenace a vypište výsledek.
- Obě alokovaná pole nezapomeňte na konci programu také uvolnit!

Například:

- (10 20 30, 40 50 60) => 50 70 90
- (”Hello”, ”Ahoj”) => ”HelloAhoj”

Velikost dynamického pole

Velkost dynamicky alokované paměti nelze zjistit příkazem – `sizeof` –:

- Příkaz vrátí velikost ukazatele (adresy), která je vždy stejná bez ohledu na velikost alokované paměti na kterou ukazuje.

```
1  int poleA[10];                                  //automaticke pole cisel
2  int* poleB = malloc(10 * sizeof(int));          //dynamicke   pole cisel
3  printf("PoleA ma %i B\n", sizeof(poleA)); //vypis velikost poleA (40)
4  printf("PoleB ma %i B\n", sizeof(poleB)); //vypis velikost poleB (4)
5  free(poleB);                                    //uvolni dynamicke pole
```

Pole a jeho velikost (počet položek) můžeme spojit do struktury:

```
6  typedef struct Smnozina //deklarujeme datovy typ pro strukturu
7  {                       //jmenem "Smnozina" se dvema polozkami
8      int delka;          //pocet polozek
9      int* pole;          //dynamicky alokovane pole
10 } mnozina;              //jmeno tohoto typu je "mnozina"
```

Pokud je pole prázdné jeho adresu nastavujeme na NULL:

```
11 mnozina A;              //promenna typu "mnozina" jmenem "A"
12 A.delka = 0;            //pole obshauje 0 prvku
13 A.pole = NULL;          //adresa pole je NULL (neni alokovano)
```

Vyzkoušejte si:

- Implementujte následující funkce pro práci s množinami:

```
1 mnozina* alokujMnozinu(int delka);
2 void nactiMnozinu(mnozina* A);
3 void vypisMnozinu(mnozina* A);
4 bool jeMnozina(mnozina* A);
```

- Funkce alokujMnozinu která alokuje množinu, alokuje její pole, a nastaví délku.
- Funkce nactiMnozinu ze vstupu načte hodnoty a uloží je do pole množiny.
- Funkce vypisMnozinu hodnoty z pole množiny vypíše na výstup.
- Funkce jeMnozina ověří že se hodnoty v poli množiny neopakují.
- Kód funkce main si zkopírujte z následujícího slajdu.

Například:

- (10 20 30 40, 10 10 10) => 10 20 30 40
  => 10 10 10 => 3
- (10 20 30 40, 20 40 60) => 10 20 30 40
  => 20 40 60 => 0

Na procvičení

```
1  typedef struct Smnozina              //deklarujeme datovy typ pro strukturu
2  {                                    //jmenem "Smnozina" se dvema polozkami
3      int delka;                       //pocet polozek
4      int* pole;                       //dynamicky alokovane pole
5  } mnozina;                           //jmeno tohoto typu je "mnozina"
6
7  int main()                           //zacatek programu
8  {
9      mnozina* A = alokujMnozinu(4); //alokujeme mnozinu A (a jeji pole)
10     if(A==NULL || A->pole==NULL) return 1; //osetreni alokace
11     nactiMnozinu(A);                 //do mnoziny A nacitame vstup
12     vypisMnozinu(A);                 //vypisujeme mnozinu A
13
14     mnozina* B = alokujMnozinu(3); //alokujeme mnozinu B (a jeji pole)
15     if(B==NULL || B->pole==NULL) return 2; //osetreni alokace
16     nactiMnozinu(B);                 //do mnoziny B nacitame vstup
17     vypisMnozinu(B);                 //vypisujeme mnozinu B
18     if(!jeMnozina(A) || !jeMnozina(B)) return 3; //osetreni vstupu
19
20     return 0;                        //program konci bez chyby
21 }
```

Na procvičení

Vyzkoušejte si:

- K předcházející úloze přidejte implementaci těchto funkcí:

```
1 mnozina* konkatenaceMnozin(mnozina* A, mnozina* B);
2 void uvolniMnozinu(mnozina* A);
3 mnozina* prunikMnozin(mnozina* A, mnozina* B);
```

- Funkce konkatenaceMnozin vytvoří novou množinu která bude konkatenací jejích vstupů (vaším úkolem není udělat sjednocení).
- Funkce uvolniMnozinu uvolní pole množiny a množinu.
- Funkce prunikMnozin vytvoří novu množinu která bude průnikem jejích vstupů.
- Kód funkce main doplňte o řádky z následujícího slajdu.

Například:

- (10 20 30 40, 10 10 10) => 10 20 30 40
  => 10 10 10 => 3
- (10 20 30 40, 20 40 60) => 10 20 30 40
  => 20 40 60
  => 10 20 30 40 20 40 60
  => 20 40 => 0

```
1  int main()                          //zacatek programu
2  {
3      /*
4      tady bude kod z predchazejici ulohy
5      */
6
7      mnozina* C = konkatenaceMnozin(A, B);  //C je konkatenaci A a B
8      if(C==NULL || C->pole==NULL) return 4; //osetreni alokace
9      vypisMnozinu(C);                //vypisujeme mnozinu C
10
11     uvolniMnozinu(C);               //uvolnujeme mnozinu C
12
13     C = prunikMnozin(A, B);         //C je prunikem A a B
14     if(C==NULL || C->pole==NULL) return 5; //osetreni alokace
15     vypisMnozinu(C);                //vypisujeme mnozinu C
16
17     uvolniMnozinu(C);               //uvolnujeme mnozinu C
18     uvolniMnozinu(B);               //uvolnujeme mnozinu B
19     uvolniMnozinu(A);               //uvolnujeme mnozinu A
20     return 0;                       //program konci bez chyby
21 }
```