Dynamická alokace paměti

IZP-cv07

Ing. Jakub Husa Ph.D.

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

5. listopadu 2024

Dynamická alokace paměti

Různé datové typy zabírají v paměti počítače různé množství místa:

- Velikost zjistíme příkazem – `sizeof` – který vrací velikost v bajtech (byte).
- Velikost datových typů **NENÍ** pevně definována a závisí na operačním systému.

```
1     printf("int    = %i\n", sizeof(int));    //cele cislo            //4
2     printf("float  = %i\n", sizeof(float));  //desetinne cislo       //4
3     printf("double = %i\n", sizeof(double)); //dvojnasobna presnost  //8
4     printf("char   = %i\n", sizeof(char));   //znak                  //1
```

- Velikost pole je násobkem počtu a velikosti jeho položek, protože položky pole jsou v paměti ukládány jako jeden souvislý blok dat.
- Velikost struktury může být větší než je součet velikostí jejích položek, protože položky struktury jsou v paměti automaticky zarovnávány na velikost slova:

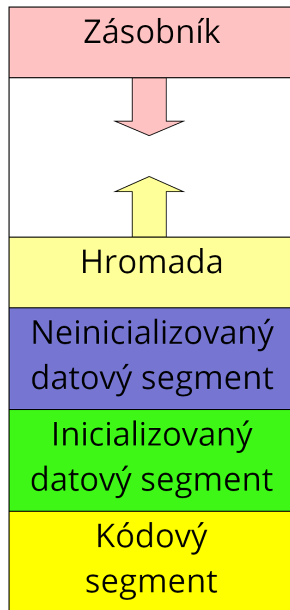
```
5  typedef struct Styp {char x; int i;} typ;    //struktura obsahujici
6  int main()                                   //znak (1) a cele cislo (4)
7  {
8      int arr[10];                             //pole deseti celych cisel
9      printf("arr    = %i\n", sizeof(arr));    //4 * 10 = 40
10     printf("typ    = %i\n", sizeof(typ));    //1 + 4 + zarovnani = 8
11     return 0;
12 }
```

Paměťový prostor programu se skládá z pěti segmentů:

- Zásobník (stack) obsahuje automaticky alokované proměnné vytvářené uvnitř funkcí a parametry předávané při jejich volání, roste směrem shora dolů.
- Hromada (halda, heap) obsahuje dynamicky alokované proměnné a roste směrem zdola nahoru.
- Neinicializovaný datový segment obsahuje globální proměnné bez počáteční hodnoty.
- Inicializovaný datový segment obsahuje globální proměnné s počáteční hodnotou.
- Kódový segment obsahuje a zdrojový kód programu, a za běhu programu do něj nelze zapisovat.

Operační paměť počítače má omezenou velikost:

- Když se zásobník s hromadou potkají program havaruje!
- O ostatních segmentech se dozvíte více až v předmětu ISU.



Automatická a dynamická alokace paměti

Automaticky alokované proměnné na konci funkce přestávají existovat:

- Přístup k neexistující proměnné způsobí havárii na neplatný přístup do paměti.

Problém vyřešíme dynamickou alokací voláním funkce malloc z knihovny stdlib.h:

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

- Automatická alokace na zásobníku.

```c
int *foo()      //"a" je automaticky
{               //alokovane cislo
    int a;
    a = 10;
    return &a;//vracime adresu
}               //promenne "a"

int main()      //vypisujeme hodnotu
{               //smazane promenne
    int *x = foo();
    printf("%i\n", *x); //HAVARIE
    return 0;
}
```

- Dynamická alokace na hromadě.

```c
int *bar()      //"b" je ukazatel na
{               //alokovanou pamet
    int* b = malloc(sizeof(int));
    *b = 10;
    return b; //vracime adresu
}               //alokovane pameti

int main()      //vypisujeme hodnotu
{               //z alokovane pameti
    int *x = bar();
    printf("%i\n", *x); //OK
    return 0;
}
```

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.
- Pokud paměť neuvolníme, nastává únik paměti.

```
1   int* x;                       //ukazatel na cele cislo
2   x = malloc(sizeof(int));      //alokujeme misto pro jedno cele cislo
3
4   if (x != NULL)                //pokud alokace uspela (osetreni)
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  }
```

Alokace polí

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

- S ukazatelem tedy můžeme pracovat stejně jako s polem.
- 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.

```
int* soucet(int delka, int* arr1, int* arr2);
char* konkatenace(char* str1, char* str2);
```

Například:

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

```c
int main()                                  //zacatek programu
{
    int arr1[3] = {10, 20, 30};             //prvni pole
    int arr2[3] = {40, 50, 60};             //druhe pole
    int* arr3 = soucet(3, arr1, arr2);      //volame funkci soucet
    if (arr3 != NULL)                       //pokud alokace uspela
    {
        for (int i = 0; i < 3; i++)         //pro vsechny prvky pole
            printf("arr3[%i] = %i\n", i, arr3[i]); //vypis
        free(arr3);                         //uvolnujeme pamet
    }

    char str1[6] = "Hello";                 //prvni retezec
    char str2[5] = "Ahoj";                  //druhy retezec
    char* str3 = konkatenace(str1, str2);   //volame funkci konkatenace
    if (str3 != NULL)                       //pokud alokace uspela
    {
        printf("%s\n", str3);               //vypisujeme retezec
        free(str3);                         //uvolnujeme pamet
    }
    return 0;                               //konec programu
}
```

Velikost dynamicky alokovaného pole

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

- Příkaz vrátí velikost ukazatele (adresy), která je vždy stejná, bez ohledu na množství alokované paměti na kterou ukazuje.
- Velikost ukazatelů (délka adres) je dána počtem bitů operačního systému.

```
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    //8
5
6  printf("char*     = %i\n", sizeof(char*)); //ukazatel na znak        //8
7  printf("int*      = %i\n", sizeof(int*));  //ukazatel na cele cislo  //8
```

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

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

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);
5 mnozina* konkatenaceMnozin(mnozina* A, mnozina* B);
6 void uvolniMnozinu(mnozina* A);
7 mnozina* prunikMnozin(mnozina* A, mnozina* B);
```

- 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.
- Funkce konkatenaceMnozin vytvoří novou množinu která bude konkatenací (ne sjednocením) dvou množin.
- Funkce uvolniMnozinu uvolní pole množiny a množinu.
- Funkce prunikMnozin vytvoří novu množinu která bude průnikem dvou množin.

Na procvičení (2/2)

Například:

- (10 20 30 40, 10 10 10)
  - => Mnozina A:
  - => 10 20 30 40
  - => Mnozina B:
  - => 10 10 10
  - => Zadana pole nejsou mnozinami
- (10 20 30 40, 20 40 60)
  - => Mnozina A:
  - => 10 20 30 40
  - => Mnozina B:
  - => 20 40 60
  - => Zadana pole jsou mnozinami
  - => Konkatenace:
  - => 10 20 30 40 20 40 60
  - => Prunik:
  - => 20 40

```c
int main() {    // ve VSCode formatovani spravime zkratkou "Shift + Alt + F"
    mnozina *A =alokujMnozinu(4);          // alokujeme mnozinu A (a jeji pole)
    mnozina *B =alokujMnozinu(3);          // alokujeme mnozinu B (a jeji pole)
    if (A ==NULL ||A->pole ==NULL ||B ==NULL ||B->pole ==NULL) {// osetreni alokace
        printf("Alokace selhala\n"); return 1;}

    nactiMnozinu(A);                       // do mnoziny A nacitame vstup
    nactiMnozinu(B);                       // do mnoziny B nacitame vstup
    printf("Mnozina A:\n"); vypisMnozinu(A); // vypisujeme mnozinu A
    printf("Mnozina B:\n"); vypisMnozinu(B); // vypisujeme mnozinu A

    if (!jeMnozina(A) ||!jeMnozina(B)) {   // osetreni vstupu
        printf("Zadana pole nejsou mnozinami\n"); return 2;}
    printf("Zadana pole jsou mnozinami\n");

    mnozina *C =konkatenaceMnozin(A, B);   // C je konkatenaci A a B
    if (C ==NULL ||C->pole ==NULL) {       // osetreni alokace
        printf("Alokace konkatenace selhala\n"); return 3;}
    printf("Konkatenace:\n"); vypisMnozinu(C); // vypisujeme mnozinu C
    uvolniMnozinu(C);                      // uvolnujeme mnozinu C

    C = prunikMnozin(A, B);                // C je prunikem A a B
    if (C ==NULL ||C->pole ==NULL) {       // osetreni alokace
        printf("Alokace pruniku selhala\n"); return 4;}
    printf("Prunik:\n"); vypisMnozinu(C); // vypisujeme mnozinu C

    uvolniMnozinu(C); uvolniMnozinu(B); uvolniMnozinu(A); // uvolnujeme mnoziny
    return 0;
}
```