Робота з даними в динамічній пам'яті

На відміну від звичайного розподілу пам'яті, де вона розподіляється під час компіляції (наприклад ***int a[10]***, де виділяється 10 елементів), мова С дозволяє під час самого виконання програми виділяти необхідну пам’ять, наприклад задати в діалоговому режимі з клавіатури.

Динамічна дані зберігаються в області оперативної пам'яті (ОП), що підлягає динамічному розподілу.

Цю область пам'яті називають ***heap*** (що в перекладі означає *купа*). ***Купа*** – це вільна частина ОП, не зайнята ОС та активними програмами.

Переваги використання динамічної пам'яті:

- Виділення пам'яті лише на час опрацювання цих даних, потім звільнення для інших потреб;
- Обсяг пам'яті задається програмно і можна встановлювати відповідним до реального розміру даних (важливо для масивів і рядків);
- В процесі роботи можна змінити обсяг ділянки динамічної пам'яті;
- Здебільшого вільна динамічна пам’ять достатньо велика (її обяг перевищує обсяги сегментів статичних даних і стека), тому можна розташовувати великі масиви та інші об’єкти ;
- Можливість створення ефективних динамічних структур даних.

Функції динамічного виділення пам'яті

Прототипи функцій оголошені в стандартному заголовному файлі ***<stdlib.h>***.

Основною функцією виділення пам’яті є

void* malloc (size_t msize);

Параметр ***msize*** задає обсяг у байтах неперервної ділянки, яка повинна бути виділена в динамічній пам'яті. Тип ***size_t*** задекларовано і відповідає одному з цілих беззнакових типів (для Borland C рівнозначний типу *unsigned int*). Функція повертає адресу першого байта ділянки заданого обсягу, якщо вільної пам'яті недостатньо, повертає NULL.

*Приклад виділення ділянки заданого обсягу 600 байтів. Можна розмістити dsize/sizeof(*pm) елементів (наприклад масив з 300 даних типу int):*

```
#include <stdlib.h>
. . .
 int *pm, dsize=600;
 pm=malloc(dsize);
if (pm==NULL) puts (“Відсутня вільна пам’ять”);
. . .
```

Іншою функцією для виділення динамічної пам’яті є

void* calloc (size_t num, size_t size);

Функція виділяє неперервну ділянку для розташування масиву. ***num*** – кількість елементів, ***size*** – розмір кожного з елементів. Результат виділення збігається з результатом виклику ***malloc(num*dsize)***. Особливість у тому, що всі біти виділеної пам'яті заповнюються нулями.

Приклад: ***double *arr;***

arr =(double *) calloc (k, sizeof(double));

Функція для зміни обсягу ділянки виділеної пам'яті

void* realloc (void *ptr, size_t newsize);

ptr – вказівник на ділянку динамічної пам'яті, обсяг якої треба змінити; ***newsize*** – новий обсяг ділянки в байтах.

У разі успішного виконання функція повертає вказівник на ділянку нового обсягу, а в разі невдачі – **NULL**.

Для звільнення ділянки динамічної пам'яті, виділеної раніше однією із функцій, застосовують

void free (void* ptr);

Функція повертає в область пам'яті, що підлягає динамічному розподілу, ділянку, на початок якої вказує ***ptr***.

Необхідно стежити, щоб параметр функції дійсно був адресою ділянки для звільнення!

Приклад:

free(pm);

Приклад використання динамічної пам'яті для створення одновимірного масиву довжини, заданої в процесі користування програмою

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int *a,n,i;
 scanf("%d",&n);                    //Введення кількості
                                    // елементів масиву
 a=(int*)calloc(n, sizeof(int));        // Виділення динамічної
                                        // пам'яті
      if (!a)                       //Перевірка на можливість
                                     // виділення пам'яті
```

```
        {
            puts("ERROR");
            return 0;
          }

 for (i=0; i<n; i++)
 scanf("%d", a+i);      //Введення даних масиву за
                                  // допомогою вказівників
   for (i=0; i<n; i++)
   printf("%d",*(a+i)); //Виведення елементів масиву
   free(a);                //Звільнення динамічної пам'яті
  return 0;
}
```

Проблеми, пов'язані з вказівниками

Некоректним використанням покажчиків може бути:

- спроба працювати з неініціалізованим покажчиком, тобто з покажчиком, що не містить адреси ОП, яка виділена змінній;
- втрата вказівника, тобто значення покажчика через присвоювання йому нового значення до звільнення ОП, яку він адресує;
- незвільнення ОП, що виділена за допомогою функції ***malloc*();**
- спроба повернути як результат роботи функції адресу локальної змінної класу ***auto***.

При оголошенні покажчика на скалярне значення будь-якого типу , пам’ять для значення (що адресується) не резервується. Виділяється тільки ОП для покажчика, але покажчик при цьому не має значення.

Якщо покажчик має специфікатор ***static***, то ініціюється початкове значення покажчика, рівне нулю

```
static int *pi, *pj;        /* pi = NULL; pj= NULL; */
```

Спроба працювати з непроініціалізованим покажчиком:

```
int *x;   /* змінній-покажчику 'x' виділена ОП, але 'x' не
                  містить значення адреси ОП для змінної */
*x = 123;     /* - груба помилка */
```

Таке присвоювання помилкове, тому що змінна-покажчик ***x*** не має значення адреси, за яким має бути розташоване значення змінної.

Компілятор видасть попередження:

Warning: Possible use of 'x' before definition

При цьому випадкове значення покажчика (сміття) може бути неприпустимим адресним значенням! Наприклад, воно може збігатися з адресами розміщення програми або даних. Компілятор не виявляє цю помилку, це повинен робити програміст!

Помилки не буде у випадку використання функції ***malloc()***

```
int *x;                  /* x - ім'я покажчика, він одержав ОП */
x = (int *) malloc ( sizeof(int));     /* Виділена ОП цілому
                                          значенню, на яке вказує 'x' */
*x = 123;                /* змінна, на яку вказує 'x', одержала
                            значення 123*/
```

Щоб уникнути помилок при роботі з функціями не слід повертати як результат їхнього виконання адреси локальних змінних функції. Оскільки при виході з функції пам'ять для таких змінних звільняється, повернута адреса може бути використаною системою й інформація за цією адресою може бути невірною. Можна повернути адресу ОП, що виділена з купи.

Одна з можливих помилок - подвійна вказівка на дані, розташовані у купі, і зменшення об'єму доступної ОП через незвільнення отриманої ОП.

Приклад фрагмента програми з подвійною вказівкою і зменшенням об'єму доступної ОП через незвільнення ОП

```
#include<alloc.h>
 void main ()
 {
     /* Виділення ОП динамічним змінним x, y и z: */
     int *x = (int *) malloc ( sizeof(int)),
     *y = (int *) malloc ( sizeof(int)),
     *z = (int *) malloc ( sizeof(int));
     /* Ініціалізація значення покажчиків x, y, z;*/
     *x = 14; *y = 15; *z = 17;
/*Динамічні змінні одержали конкретні цілі значення*/
     y=x;     /* груба помилка - втрата покажчика на
                 динамічну  змінну без попереднього
                  звільнення її ОП*/
 }
```

У наведеному вище прикладі немає оголошення імен змінних, є тільки покажчики на ці змінні. Після виконання оператора ***y = x;*** ***x*** та ***y*** є двома покажчиками на ту саму ОП змінної ****x***. Тобто ****x = 14;*** і ****y = 14***. Крім того, 2 байти, виділені змінній, яку адресував ***y*** для розміщення цілого значення, стають недоступними, тому що значення ***y***, його адреса, замінені значенням ***x***.
А в купі ці 2 байти для ****y*** вважаються зайнятими. Щоб уникнути такої помилки треба попередньо звільнити ОП, виділену змінній ****y***, а потім виконати присвоювання значення змінній ***y***.

```
free (y);          /* звільнення ОП, виділеної змінної  '*y'  */
y = x;        /* присвоювання нового значення змінній  'y' */
```

Масиви вказівників

За допомогою масивів покажчиків можна формувати великі масиви. Розмір одного масиву даних повинний бути не більше 64 Кб. Але в реальних задачах можуть використовуватися масиви, що вимагають ОП, більшої ніж 64 Кб. Наприклад, масив даних типу ***float*** з 300 рядків і 200 стовпців потребує для розміщення 300 * 200 * 4 = 240000 байтів. Для вирішення задачі можна використовувати масив покажчиків і динамічне виділення ОП для кожного рядка матриці. Рядок матриці не повинен перевищувати 64 Кб.

Для всіх рядків масиву треба оголосити масив покажчиків, по одному для кожного рядка. Потім кожному рядку масиву виділити ОП, привласнивши кожному елементу масиву покажчиків адресу початку розміщення рядка в ОП, і заповнити цей масив.

У прикладі представлена програма для роботи з великим масивом цілих значень з 200 рядків і 300 стовпців. Для розміщення він вимагає: 200 * 300 * 2 = 120000 байтів. При формуванні великого масиву використовується ***p*** – статичний масив покажчиків. При виконанні програми перебираються ***i*** -номери рядків масиву.

Для кожного рядка за допомогою функції ***malloc()*** виконується запит ОП з купи і формується ***p[i]*** - значення покажчика на дані ***i***-рядки.

Потім перебираються ***i***-номери рядків від 1 до 200. Для кожного рядка перебираються ***j***-номери стовпчиків від 1 до 300. Для кожного ***i*** та ***j*** за допомогою генератора випадкових чисел формуються і виводяться ****(p[i] + j)*** - значення елементів масиву. Після обробки масиву за допомогою функції ***free(p[i])*** звільняється ОП виділена ***i***-рядку масиву.

Використовуються звертання до *Ai,j* - елементів масиву у вигляді:

****(p[i]+j),*** де ***p[i] + j*** - адреса *Ai,j*-елемента масиву.

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
void main()
{
    int *p[200], i, j;        /* p - масив покажчиків */
    clrscr();
    randomize();
    for (i=0;i<200;i++)

/* Запит ОП для рядків великого масиву: */
p[i] = (int*) malloc (300 * sizeof (int));
    for (i = 0; i < 200; i++)
        for (j = 0; j < 300; j++ )
```

```
{
   *(p[i] + j ) = random(100);
          printf("%3d", *(p[i] + j ));
    if ( (j + 1) % 20 == 0 )
          printf ("\n" ) ;
      }
      /* Звільнення ОП рядків великого масиву: */

   for ( i=0; i < 200; i++ )
          free( p[i] );
}
```