Обектно ориентирано програмиране

КЛАСОВЕ

ДИНАМИЧНИ ОБЕКТИ

Динамични обекти

Вече разгледахме в най-общ план разпределението на ОП по време на изпълнението на програма. Всяка програма има три “места” за памет:

- *област на статичните данни*
- *програмен стек (стек)*
- *област за динамичните данни (динамична памет, heap).*

Стекът е област за временно съхранение на информация. Той е кратковременна памет. C++ използва стека основно за реализиране на обръщения към функции. Всяко обръщение към функция предизвиква конструиране на нова стекова рамка, която се установява на върха на стека. По такъв начин когато функция A извика функция B, която от своя страна вика функция C, стекът нараства. Когато пък всяка от тези функции завършва, стековите рамки на тези функции автоматично се разрушава. Така стекът се свива.

Динамични обекти ...

Хийпът е по-постоянна област за съхранение на данни. Той е един вид дълготрайна памет. Особеност на тази памет е, че тя не се свързва с имена на променливи. С разположените в нея обекти се работи косвено – чрез указатели. Обикновено се използва при работа с т. нар. **динамични структури от данни**.

Динамичните данни са такива обекти (в широкия смисъл на думата), чийто брой не е известен в момента на проектирането на програмата. Те се създават и разрушават по време на изпълнението на програмата. След разрушаването им, заетата от тях памет се освобождава и може да се използва отново. Така паметта се използва по-ефективно.

Динамични обекти …

Използването на динамичната памет досега не се налагаше, тъй като структурите от данни, с които работехме, бяха статични. По нататък ще дефинираме и използваме динамичните структури от данни свързан списък, стек, опашка, дърво, граф и др., използването на тези средства е задължително.

Създаването и разрушаването на динамични обекти в C++ се осъществява чрез операторите **new** и **delete**.

Извикването на new заделя в хийпа необходимата памет и връща указател към нея. Този указател може да се съхрани в някаква променлива и да се пази докато е необходимо. За разлика от стека, заделянето на памет в хийпа е явно – чрез **new**.

Динамични обекти …

Освобождаването на паметта от хийпа също става явно, чрез delete. Всяко извикване на new трябва да бъде балансирано чрез извикване на delete. Последното се налага, тъй като за разлика от стека, хийпът не се изчиства автоматично. В C++ няма система за “събиране на боклуци”. Затова трябва явно да бъдат изтрити създадените в хийпа обекти.

Оператор new

Синтаксис

new <име_на_тип> [[size]] |

new <име_на_тип> (<инициализация>)

където

- *<име_на_тип>* е име на някой от стандартните типове int, double, char и др. или е име на клас;
- *size* е израз с произволна сложност, но трябва да може да се преобразува до цял. Показва броя на компонентите от тип <име_на_тип>, за които да се задели памет в хийпа и се нарича **размерност**;
- *<инициализация>* е израз от тип <име_на_тип> или инициализация на обект според синтаксиса на конструктора на класа, ако <име_на_тип> е име на клас.

Динамични обекти ...

Семантика

Заделя в хийпа (ако е възможно):

- sizeof(<име_на_тип>) байта, ако не са зададени size и <инициализация> или
- sizeof(<име_на_тип>)*size байта, ако явно е указан size или
- sizeof(<име_на_тип>) байта, ако е специфицирана <инициализация>, която памет се инициализира с <инициализация>

и връща указател към заделената памет.

Динамични обекти …

Забележка:

- Ако <име_на_тип> е име на клас и след него има кръгли скоби, в тях трябва да стоят фактически параметри (аргументи) на конструктора на класа.
- Ако скобите липсват, класът трябва да притежава конструктор по подразбиране или да няма явно дефиниран конструктор.
- Ако след името на класа е поставен заграден в квадратни скоби израз, new заделя място за масив от обекти на указания клас и извиква конструктора по подразбиране за инициализиране на отделената памет.

Примери:

а)
```
int *q = new int(2 + 5 * 5);
```

отделя (ако е възможно) 4B памет в хийпа, инициализира я с 27 - стойността на израза 2+5*5 и свързва q с адреса на тази памет

Динамични обекти …

б) `int *p = new int[10];`

отделя (ако е възможно) 40B в хийпа (за 10 елемента от тип int) и свързва p с адреса на тази памет

в) `Rational *r = new Rational(1, 5);`

отделя памет в хийпа за един обект от клас Rational, свързва r с адреса на тази памет и извиква конструктора Rational(1,5) за да я инициализира

г) `Rational *r = new Rational;`

отделя памет в хийпа за обект от тип Rational, записва адреса на тази памет в r и извиква конструктора по подразбиране на класа Rationalза инициализиране на тази памет

д) `Rational *r = new Rational[10];`

отделя памет в хийпа за 10 обекта от класа Rational, записва адреса на тази памет в r, извиква конструктора по подразбиране на класа Rational и инициализира отделената памет;

Динамични обекти ...

e) `Rational** parr = new Rational*[5];`

отделя 20B памет в хийпа за масив от 5 указателя към стойности от тип Rational и записва в parr адреса на тази памет

Заделянето на памет по време на компилация се нарича **статично** заделяне на памет, заделянето на памет по време на изпълнение на програмата - **динамично разпределение на паметта**. Паметта за променливите q, p, r и parr, от примерите по-горе, е заделена статично, а всяка една от тези променливи има за стойност адрес от хийпа. Казва се още, че q, p, r и parr адресират динамична памет.

Динамични обекти ...

Под период на **активност на една променлива** се разбира частта от времето за изпълнение на програмата, през което променливата е свързана с конкретно място в паметта.

Паметта за глобалните променливи се заделя в началото и остава свързана с тях до завършването на изпълнението на програмата.

Паметта за локалните променливи се заделя при влизане в локалната област и се освобождава при напускането ѝ.

Паметта на динамичните променливи се заделя от оператора new. Заделената по този начин памет остава свързана със съответната променлива докато не се освободи явно от програмиста. Явното освобождаване на динамична променлива се осъществява чрез оператора delete, приложен към указателя, който адресира съответната променлива.

Динамични обекти …

Оператор delete

Синтаксис

delete <указател_към_динамичен_обект>;

където <указател_към_динамичен_обект> е указател към динамичен обект (в широкия смисъл на думата), създаден чрез оператора new.

Семантика

Разрушава обекта, адресиран от указателя, като паметта, която заема този обект, се освобождава. Ако обектът, адресиран от указателя, е обект на клас, отначало се извиква деструкторът на класа и след това се освобождава паметта.

Динамични обекти ...

Ако в хийпа е заделена памет, след което тази памет не е освободена чрез delete, се получава загуба на памет. Парчето памет, което не е освободено, е като остров в хийпа, заемащо пространство, което иначе би могло да се използва за други цели.

За да се разруши масив, създаден чрез new по следния начин:

```
int* arr = new int[5];
```

трябва да се запише:

```
delete [] arr;
```

Забележка: Някои реализации на езика допускат разрушаването в горния случай да стане и чрез `delete arr`.

Ако обаче масивът съдържа в себе си указатели, първо трябва да бъде обходен и да бъде извикан операторът delete за всеки негов елемент.

Динамични обекти ...

Операторът delete трябва да се използва само за освобождаване на динамична памет, заделена с new. В противен случай действието му е непредсказуемо. Няма забрана за прилагане на delete към указател със стойност 0. Ако стойността на указателя е 0, той е свободен и не адресира нищо.

Пример:

```
void example() {
    ...
    int a = 7;
    char *str = "abv";
    int *pa = &a;
    Rational *ptr = 0;
    double *x = new double;
```

Динамични обекти …

```
  delete str; //некоректно обръщение,
              //str не е адресирано чрез new
    delete pa;  //некоректно обръщение,
                //pa не е адресирано чрез new
    delete ptr; //некоректно обръщение,
                //ptr не е адресирано чрез new
    delete x;   // коректно обръщение
    ...
}
```

Динамичната памет не е неограничена. Тя може да се изчерпи по време на изпълнение на програмата. Ако наличната в момента динамична памет е недостатъчна, new връща нулев указател. Затова се препоръчва след всяко извикване на new да се прави проверка за успешността ѝ.

Динамични обекти ...

Чрез оператора new могат да се създават т. нар. **динамични масиви** – масиви с променлива дължина. Динамичните масиви се създават в динамичната памет. Следващата програма илюстрира този процес.

Задача. Да се напише програма, която създава динамичен масив от цели числа. Да се изведе масивът.

```
#include <iostream>
using namespace std;
int main() {
   int size; // дължина на масива
   do
   {
      cout << "size of array: ";
      cin >> size;
    } while (size < 1);
```

Динамични обекти ...

```
    // създаване на динамичен масив arr от size
    // елемента от тип int
    int* arr = new int[size];
    int i;
    for (i = 0; i < size; i++)
        arr[i] = i;
    // извеждане на елементите на arr
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
    // освобождаване на заетата динамична памет
    delete[] arr;
    return 0;
}
```

Динамични обекти …

Методите на класовете също могат да използват динамична памет, която се заделя и освобождава по време на изпълнението им, чрез операторите new и delete.

```
class Product
{
private:
   char* name;
   double price;
   ...
public:
   void read();
   void print() const;
   ...
};
```

Динамични обекти ...

```
void Product::read() {
    static char s[40];
    cout << "name: ";
    cin >> s;
    name = new char[strlen(s) + 1];
    strcpy(name, s);
    cout << "price: ";
    cin >> price;
    ...
}
```

Динамични обекти ...

Ще отбележим също, че заделената от член-функциите динамична памет не се освобождава автоматично при разрушаване на обектите на класове. Освобождаването на тази памет трябва да стане явно чрез оператора delete, който трябва да се изпълни преди разрушаването на обекта. Този процес може да бъде автоматизиран чрез използване на специален вид методи, наречени **деструктори**.