Указатели

Работа с динамической памятью

Указатель

- – переменная, значением которой является адрес ячейки памяти.
- То есть **указатель** ссылается на блок данных из области памяти, причём на самое его начало.
- **Указатель** может ссылаться на переменную или функцию

Работа с указателями

- Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в C++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.
- Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Ведь программист заранее не может знать, какого размера нужен массив тому или иному пользователю, в таком случае используется динамическое выделение памяти под массив. Любой указатель необходимо объявить перед использованием, как и любую переменную.

Инициализация указателей

- 1. Присваивание указателю адреса существующего объекта:
- С помощью операции получения адреса:
  - int a=5;//целая переменная
  - int *p=&a;//в указатель записывается адрес a
- С помощью значения другого инициализированного указателя:
  - int *r=p;
- С помощью имени массива, которые трактуются как адрес:
  - int b[10]; //массив
  - int *t=b;//присваивание адреса начала массива

Инициализация указателей

- 2. Присваивание указателю адреса области в явном виде:
  - char *cp=(char*) 0xb7000000;
  - где 0xb7000000 – шестнадцатеричная константа,
  - (char*)- операция привидения типа: константа преобразуется к типу «указатель на char»

Инициализация указателей

- 3. Присваивание указателю пустого значения:
  - int *sum=NULL;
  - int *rez=0;
  - Где NULL в некоторых заголовочных файлах определена как указатель, равный нулю

Выделение и освобождение динамической памяти

- Выделение и присваивание ее адреса указателю:
  - Int* n=new int;//выделение достаточного для размещения величины типа int памяти и записывает адрес начала этого участка в переменную n
  - Int*m=new int(10);//+инициализация выделенной динамической памяти значением 10
  - Int*q=new int[10];//выделение памяти под 10 величин типа int и записывает адрес начала этого участка в q, которая трактуется как имя массива
- Освобождение памяти
  - Delete n;
  - Delete m;
  - Delete [ ]q;//размерность массива при этом не указывается

Выделяется память с помощью оператора new, а освобождается — с помощью оператора delete.

- В момент, когда динамическая память выделена, она должна быть связана с некоторым указателем, подходящего типа (при выделении указывается тип и количество необходимых ячеек данного типа).

```
int* p;
p = new int;
*p = 10;
cout << *p; // 10
delete p; // память освобождена
```

- Сразу после создания динамический массив автоматически заполняется нулями (в отличии от обычного массива в статической или стековой памяти).

- Если в указатель, уже хранящий адрес какого-то фрагмента динамической памяти, записать новый адрес, то фрагмент динамической памяти будет потерян, т. е. он не будет освобождён, но к нему никак нельзя будет обратиться (например, чтобы освободить этот фрагмент).

```
int* p;
p = new int(13);
int a = 666;
p = &a; // теперь до 13 никак не добраться
```

Если не освобождать динамическую память, то она будет занята до завершения программы, что неприемлемо.

- При выделении одной динамической переменной (одной ячейки памяти), можно сразу инициализировать её значение:

```
int* p;
p = new int(10);
cout << *p; // 10
delete p; // память освобождена
```

- Можно выделять сразу несколько ячеек динамической памяти, получая динамический массив. Для этого его размер указывается в квадратных скобках после типа. Чтобы удалить динамический массив и освободить память используется оператор delete[].

```
int* p;
p = new int[13];
for (int i=0; i<13; i++) {
    *(p+i) = i + 1;
    cout << *(p+i) << ' '; // 1 2 3 ... 13
}
delete[] p; // память освобождена, из неё удалены все элементы
```

```
#include <iostream>
using namespace std;

void main()
    {
    setlocale(LC_ALL, "Russian");
    // Вводим размерность массива
    int N;
    cout << "Введите размерность массива: ";
    cin >> N;
    // Создаем массив динамически
    int *a = new int[N];
    // Вводим элементы массива с клавиатуры
    for (int i = 0; i < N; i++)
        {
        cout << "Введите " << i << "-й элемент массива: ";
        cin >> a[i];
        }
    // Объявляем переменную для хранения суммы элементов
    int s = 0;
    // Просуммируем элементы массива
    for (int i = 0; i < N; i++)
        s += *(a + i);
    // Выводим на экран элементы массива и их сумму
    cout << "a:";
    for (int i = 0; i < N; i++) cout << " " << i[a];
    cout << endl << "Сумма элементов: " << s << endl;
    // Очищаем выделенную память
    delete[] a;
    }
```

Динамические многомерные массивы

Память выделяется в 2 этапа:

- Сначала под столбец указателей на строки матрицы, а затем в цикле под каждую строку
- Освобождение памяти осуществляется в обратном порядке

```
#include <iostream>
using namespace std;

void main()
   {
   setlocale(LC_ALL, "Russian");
   int N=4, M=3, Sum=0;
   // Ссоздаем одномерный массив указателей размером N элементов
   // (то что это массив указывают [], а то что указателей - *)
   int **a = new int* [N];

   for(int i=0; i < N; i++)
      {
      // динамически создаем одномерный массив размерностью M
      // и присваиваем его адрес элементу массива указателей
      a[i] = new int[M];
       // Заполняем массив случайными числами и выведем на экран
       for(int j=0; j < M; j++)
         {
         //путем получения остатка (%) зададим числа
         //  в диапазоне от 0 до 9
         a[i][j] = rand()%10;
         // здесь по индексу i выбирается указатель на одномерный
         // массив, а индексу j - элемент в этом массиве
         cout << a[i][j] << "   ";
         }
      cout << endl;
      }
   // Находим сумму элементов массива и выводим ее на экран
   for (int i=0; i < N; i++)
       for (int j=0; j < M; j++)
           Sum += a[i][j];
   cout << "Сумма элементов: " << Sum << endl;

   // Освобождаем выделенную память в обратном порядке
   for(int i=0; i < N; i++)
       //сначала удаляем одномерные массивы
       delete [] (a[i]);
   //а затем и сам массив указателей
   delete [] a;
   }
```

- Проблема становится особенно острой, когда в памяти теряются целые массивы (они занимают больше места, чем отдельные переменные).

```
int* p;
for (int i=1; i<=10; i++) {
    p = new int[100];
}
delete[] p;
```

- На каждом шаге цикла создаётся динамический массив из 100 элементов. Всего таких массивов будет создано 10, но только от последнего из них память будет освобождена после выхода из цикла. 9 массивов продолжат занимать место в памяти до конца программы. 9 массивов * 100 элементов * 4 байта = 3600 байт потерянной памяти, которую никак нельзя использовать (ни в этой программе, не в других запущенных).