Модуль 6. Указатели. Работа с динамическими массивами

Рассматриваются понятия указателя, ссылки, способы формирования динамических массивов и их обработки

Указатели

- на объект
- на функцию
- на void

a=5;

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа:

тип *имя;

```
int *a,*b; // указатели на целые переменные
float *pi; // указатель на вещ.переменную pi
const int * pci; // ук.на целую константу
int **t; // адрес ячейки, где хранится адрес
```

Инициализация указателей

1.Присваивание указателю адреса существующего объекта:

- с помощью операции получения адреса:

```
int a = 5;
int *p = &a; // & - операция взятия адреса
// в указатель записывается адрес a
```

с помощью значения другого инициализированного указателя:

```
int *r = p;
```

- с помощью имени массива или функции:

```
int b[10];
int *t = b;
// указатель на массив – адрес его начала
void f(int a ){ /* ... */ }
void (*pf)(int);  // указатель на функцию
pf = f;
```

2. Присваивание указателю адреса области памяти в явном виде:

```
char *vp = (char *)0xB8000000;
```

3.Присваивание пустого значения:

```
int *sn = NULL;
int *rulez = 0;
```

4. Выделение участка динамической памяти и присваивание ее адреса указателю:

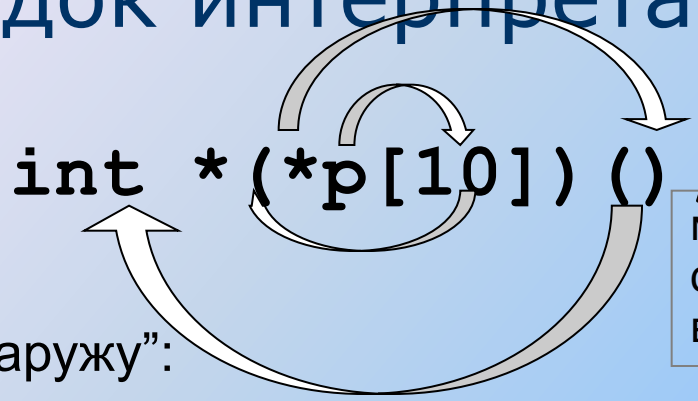
- с помощью операции new:

```
int *n = new int;
int *m = new int (10);
int *q = new int [10];
```

- с помощью функции malloc:

```
int *u = (int *)malloc(sizeof(int));
```

Порядок интерпретации описаний



массив из 10 указателей на функции без параметров, возвращающих указатели на int

“изнутри наружу”:

.если справа от имени имеются квадратные скобки, это массив, если скобки круглые — это функция;

.если слева есть звездочка, это указатель на проинтерпретированную ранее конструкцию;

.если справа встречается закрывающая круглая скобка, необходимо применить приведенные выше правила внутри скобок, а затем переходить наружу;

.в последнюю очередь интерпретируется спецификатор типа.

Освобождение памяти:

delete n; delete [] q; free (u);

Операции с указателями

- разадресация
- присваивание
- сложение с константой
- вычитание
- инкремент (++), декремент (– –)
- сравнение
- приведение типов

При работе с указателями часто используется операция получения адреса (&).

Операция разадресации

```
char a;  char * p = new char;
*p = 'Ю'; a = *p;
```

```
#include <stdio.h>
int main(){
   unsigned long int A = 0Xcc77ffaa;
   unsigned int* pint = (unsigned int *) &A;
   unsigned char* pchar = (unsigned char *) &A;
   printf(" | %x | %x |", *pint, *pchar);
}
```

| ffaa | aa |

число A | cc77 | ffaa |

Преобразование указателей

```
void *pointer = static_cast<void *>( &a );
pointer = reinterpret_cast<void *>( pa );
```

Присваивание указателей

Присваивание без явного приведения типов допускается в двух случаях:

- указателям типа void*;
- если тип указателей справа и слева от операции присваивания один и тот же.

Приведение типов:

(тип) выражение

Арифметические операции с указателями

- инкремент и декремент

```
short * p = new short [5];
p++;
long * q = new long [5];
 q++;
```

```
char *pc;
int *pi;
float *pf;
. . . . .
pc++;//значение увеличится на 1
pi++;//значение увеличится на 4
pf++;//значение увеличится на 4
```

```
*p++ = 10;   //*p = 10; p++;

(*p)++; // инкремент значения в
//ячейке, на которую показывает указатель
```

- сложение с константой

- разность

Операция получения адреса &

Унарная операция получения адреса & применима к величинам, имеющим имя и размещенным в оперативной памяти. Таким образом, нельзя получить адрес скалярного выражения, неименованной константы или регистровой переменной.

```
int a = 5;
int* p = &a;
```

Ссылки

тип & имя;

```
int kol;
int& pal = kol; //ссылка pal - альтернативное имя для kol
const char& CR = '\n'; //ссылка на константу
```

- Переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она:
  - является параметром функции
  - описана как extern
  - ссылается на поле данных класса
- После инициализации ссылке не может быть присвоена другая переменная.
- Тип ссылки должен совпадать с типом величины, на которую она ссылается.
- Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.

Динамические массивы

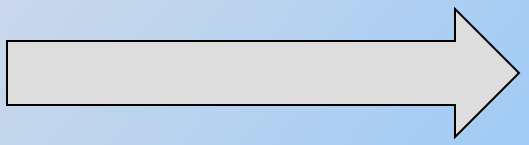
int a[100];
int k=sizeof(a);
// результатом будет 4*100=400 (байтов).
int n=sizeof(a)/sizeof(a[0]);
//количество элементов массива
Результатом операции & является адрес нулевого элемента массива:
имя_массива=&имя_массива=&имя_массива[0]

Обращение к элементу массива:
*(имя_массива+индекс)

```
for(int i=0;i<n;i++)//печать массива
cout<<*(a+i)<<" ";
// к имени адресу массива добавляется константа i и по полученному
// адресу выбирается число (элемент массива) из памяти
```

Многомерные массивы

int a [4][3];



```
a[i][j]
*(a[i]+j)
*(*(a+i)+j)
```

Адрес начала матрицы:

```
int **a;
```

Работа с указателем для адресации одномерного массива:

```
int *ptr, a[10];
ptr= & a[5]; /* адрес элемента a[5] */
ptr ++; /* адрес элемента a[6] */
ptr - -; /* снова адрес элемента a[5] */
```

Выделение памяти под одномерный динамический массив:

new тип_массива

Пример: double *b=new double[10];

```
// выделение динамической памяти размером
// 10*sizeof (double) байтов
```

Выделение памяти под двумерный динамический массив

1. long(*la)[4];//указатель на массив из 4 элементов типа long
   la=new[2][4];//выделение динамической памяти размером
   //2*4*sizeof(long) байтов
2. int **matr=(int**)new int[4][6];

//еще один способ выделения памяти под двумерный массив

3. int **matr;
   matr=new int*[4];

//выделяем память под массив указателей int* из 4 элементов
   for(int I=0;I<4;I++)matr[I]=new int[6];
//выделяем память под строки массива

```
int nstr, nstb;
cout << " Введите количество строк и столбцов :";
cin >> nstr >> nstb;
int **a = new int *[nstr];     // 1
for(int i = 0; i<nstr; i++)       // 2
    a[i] = new int [nstb];     // 3
```

Освобождение памяти от динамического массива

Одномерный массив:

```
delete[] a;
//освобождает память, выделенную под массив,
 // если a адресует его начало

delete[] la;
```

Двумерный массив:

```
for(I=0;I<4;I++)delete [] matr[I];
//удаляем строки

delete [] matr;
//удаляем массив указателей
```

Пример работы с динамическими матрицами

```
#include <iostream.h>
#include <stdlib.h>
void main()
{
int n,m;//размерность матрицы
int i,j;
cout<<"\nEnter n";
cin>>n;//строки
cout<<"\nEnter m";
cin>>m;//столбцы
//выделение памяти
int **matr=new int* [n];// массив указателей на строки
for(i=0;i<n;i++)
matr[i]=new int [m];//память под элементы матрицы
```

```
//заполнение матрицы
for(i=0;i<n;i++)
    for(j=0;j<m;j++)
       matr[i][j]=rand()%10;//заполнение матрицы
//удаление строки с номером к
int k;
cout<<"\nEnter k";
cin>>k;
int**temp=new int*[n-1];//формирование новой матрицы
for(i=0;i<n;i++)
  temp[i]=new int[m];
//заполнение новой матрицы
int t;
for(i=0,t=0;i<n;i++)
   if(i!=k)
{
    for(j=0;j<m;j++)
    temp[t][j]=matr[i][j];
    t++;
}
//удаление старой матрицы
for(i=0;i<n;i++)
delete matr[i];
delete[]matr;
}
```