Лабораторная работа №7
Указатели и динамические массивы

ЦЕЛЬ РАБОТЫ: приобретение практических навыков программирования с использованием указателей и в составлении программ по обработке динамических массивов.

Теоретический материал

Основные понятия

Понятие указателя Указатели являются специальными объектами в программах на C/C++. Указатели предназначены для хранения адресов памяти. Когда компилятор обрабатывает оператор определения переменной, например,

int i=10;

то в памяти выделяется участок памяти в соответствии с типом переменной (для int размер участка памяти составит 2 или 4 байта) и записывает в этот участок указанное значение. Все обращения к этой переменной компилятор заменит адресом области памяти, в которой хранится эта переменная.

Программист имеет возможность работать непосредственно с адресами, для чего определен соответствующий тип данных – указатель. Указатель имеет следующий формат:

```
тип *имя указателя
```

Например:

```
int *a;
double *b, *d;
char *c;
```

Знак «звездочка» относится к имени указателя. Значение указателя соответствует первому байту участка памяти, на который он ссылается. На один и тот же участок памяти может ссылаться любое число указателей. В языке Си существует три вида указателей:

1. Указатель на объект известного типа. Содержит адрес объекта определенного типа.

2. Указатель типа void. Применяется, если тип объекта заранее не определен.

3. Указатель на функцию.

Указатель — это переменная, содержащая адрес переменной.

Размер указателя зависит от модели памяти. Можно определить указатель на указатель:

```
int** a;
```

Указатель может быть константой или переменной, а также указывать на константу или переменную.

```
 int i; //целая переменная
const int ci=1;   //целая константа
int* pi; //указатель на целую переменную
const int* pci; //указатель на целую константу
int* const pci; //константный указатель на целую переменную
```

Указатель можно сразу проинициализировать

```
int* pi=&i;   //указатель на целую переменную
```

Для инициализации указателя существуют следующие способы: Присваивание адреса существующего объекта:

1) с помощью операции получения адреса

```
int a=5;
int *p=&a; //указателю присвоен адрес a
int p(&a); //указателю присвоен адрес a
```

2) с помощью проинициализированного указателя

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

3) адрес присваивается в явном виде

```
char*cp=(char*)0x B800 0000;
```

где 0x B800 0000 – шестнадцатеричная константа, (char*) – операция приведения типа.

4) присваивание пустого значения:

```
int*N=NULL;
int *R=0;
```

С указателями можно выполнять следующие операции:

- разыменование (*);
- присваивание;
- сравнение;
- сложение, вычитание, арифметические инкремент ++, декремент --
- приведение типов.

Операция разыменования предназначена для получения значения переменной или константы, адрес которой хранится в указателе. Если указатель указывает на переменную, то это значение можно изменять, также используя операцию разыменования.

```
int a; //переменная типа int
int* pa=new int; //указатель и выделение памяти под
//динамическую переменную
*pa=10;//присвоили значение динамической
//переменной, на которую указывает указатель
a=*pa;//присвоили значение переменной a
```

Арифметические операции применимы только к указателям одного типа:

1) Инкремент увеличивает значение указателя на величину **sizeof(тип).**

```
char* pc;
int* pi;
double* pd;
. . . . .
pc++; //значение увеличится на 1
pi++; //значение увеличится на 4
pd++; //значение увеличится на 8
```

2) Декремент уменьшает значение указателя на величину sizeof(тип)
3) Разность двух указателей – это разность их значений, деленная на размер типа в байтах.
4) Суммирование двух указателей не допускается.
5) Можно суммировать указатель и константу.

Работа с динамической памятью

Все переменные, объявленные в программе размещаются в одной непрерывной области памяти, которую называют сегментом данных. Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточно для размещения больших массивов информации. Выходом из этой ситуации является использование динамической памяти. Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы. Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программ, либо до тех пор, пока не будут уничтожены с помощью специальных функций или операций.

В языке Си определены библиотечные функции для работы с динамической памятью, они находятся в библиотеке :

1) void*malloc(unsigned s) – возвращает указатель на начало области динамической памяти длиной s байт, при неудачном завершении возвращает NULL;

2) void*calloc(unsigned n, unsigned m) – возвращает указатель на начало области динамической для размещения n элементов длиной m байт каждый, при неудачном завершении возвращает NULL;

3) void*realloc(void *p,unsigned s) – изменяет размер блока ранее выделенной динамической до размера s байт, p – адрес начала изменяемого блока, при неудачном завершении возвращает NULL;

4) void *free(void *p) – освобождает ранее выделенный участок динамической памяти, p- адрес начала участка.

Пример:

```
int *u=(int*)malloc(sizeof(int));
free(u); //освобождение выделенной памяти
```

в функцию передается количество требуемой памяти в байтах, т. к. функция возвращает значение типа void*, то его необходимо преобразовать к типу указателя (int*).

Для создания динамических переменных используют операцию new, определенную в СИ++:

```
указатель = new имя_типа(инициализатор);
```

где инициализатор – выражение в круглых скобках. Операция new позволяет выделить и сделать доступным участок динамической памяти, который соответствует заданному типу данных. Если задан инициализатор, то в этот участок будет занесено значение, указанное в инициализаторе.

Для удаления динамических переменных используется операция delete, определенная в СИ++:

```
delete указатель;
```

где указатель содержит адрес участка памяти, ранее выделенный с помощью операции new.

```
int*x=new int(5);
delete x;
```

Массивы и указатели

При определении массива ему выделяется память. После этого имя массива воспринимается как константный указатель того типа, к которому относятся элементы массива. Исключением является использование операции sizeof(имя_массива) и операции &имя_массива.

```
int a[100];
/*определение количества занимаемой массивом памяти,
в нашем случае это 4*100=400 байт*/
int k=sizeof(a);
/*вычисление количества элементов массива*/
int n=sizeof(a)/sizeof(a[0]);
```

Результатом операции & является адрес нулевого элемента массива:

```
имя_массива==&имя_массива=&имя_массива[0]
```

Имя массива является указателем-константой, значением которой служит адрес первого элемента массива, следовательно, к нему применимы все правила адресной арифметики, связанной с указателями. Запись **имя_массива[индекс]** это выражение с двумя операндами: имя массива и индекс. Имя_массива – это указатель-константа, а индекс определяет смещение от начала массива. Используя указатели, обращение по индексу можно записать следующим образом: ***(имя_массива+индекс).**

```
for(int i=0;i<n;i++)
```

```
    cout<<*(a+i)<<" ";
/*к имени (адресу) массива добавляется константа i и полученное значение разыменовывается*/
```

Так как имя массива является константным указателем, то его невозможно изменить, следовательно, запись *(a++) будет ошибочной, а *(a+1) – нет. Указатели можно использовать и при определении массивов:

```
int a[100]={1,2,3,4,5,6,7,8,9,10};
int* na=a; // указатель на уже определенный массив
```

Многомерный массив – это массив, элементами которого служат массивы. Например, массив int a[4][5] – это массив из указателей int*, которые содержат имена одноименных массивов из 5 целых элементов:

```
Например:
a[0] == &a[0][0] == a+0*n*sizeof(int);
a[1] == &a[1][0] == a+1*n*sizeof(int);
a[i] == &a[i][0] == a+i*n*sizeof(int);
```

Динамические массивы

Операция new при использовании с массивами возвращает указатель, значением которого служит адрес первого элемента массива. При выделении динамической памяти размеры массива должны быть полностью определены:

```
//выделение динамической памяти 100*sizeof(int) байт
int* a = (int*)malloc(100*sizeof(int));
//или
int* a = new int[100];
```

Примеры:

```
1. int *a=new int[100];//выделение динамической памяти размером 100*sizeof(int) байтов
   double *b=new double[10];// выделение динамической памяти размером 10*sizeof(double) байтов

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

3. int**matr=(int**)new int[5][10];//еще один способ выделения памяти под двумерный //массив

4. int **matr;
```

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

Указатель на динамический массив затем используется при освобождении памяти с помощью операции delete. Примеры:

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

Для двумерного массива

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

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программ, либо до тех пор, пока не будут уничтожены с помощью специальных функций или операций. При формировании матрицы сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под n одномерных массивов.

```
/*выделение динамической памяти под двумерный динамический массив*/
int **matr=new int*[n];//выделение памяти по массив указателей
for(int i=0 ;i<n;i++) //выделение памяти 100*sizeof(int) байт для массива значений
 matr[i]=new int [m]; return matr;//возвращаем указатель на массив указателей

// работа с массивом

//освобождение памяти
for(int i=0;i<n;i++)
    delete [] matr[i];
delete[]matr;
```

Задание

Задача А. Обработка одномерных массивов. Переделать задание из лабораторной работы 6 с применением динамических массивов.

Для полученного массива реализовать удаление всех элементов с заданным значением (значение запрашивается с клавиатуры).

Задача Б. Обработка многомерных массивов. Переделать задание из лабораторной работы 6 с применением динамических массивов.

Для полученного массива реализовать добавление строки (для желающих посложнее столбца) элементов в заданной позиции (значение позиции и элементов строки вводятся с клавиатуры).

!!!Размерность массива запрашивать с клавиатуры. Выполнять проверку размерности массива в соответствии с условиями задачи.