**Практическое занятие №7**

**«Указатели. Динамические массивы и матрицы»**

**Теория**

**ТИП ДАННЫХ УКАЗАТЕЛЬ**

**Объявления и присваивания**

Указатель – это переменная, хранящая адрес некоторой области памяти. Указатели играют в языке C большую роль. Это связано с тем, что некоторые операции можно выполнить только с помощью указателей, а также тем, что использование указателей в ряде случаев позволяет записать код более компактно и эффективно по сравнению с другими способами.

В языке различают три вида указателей: на объект, на функцию и на void. Различные виды указателей различаются свойствами и набором допустимых операций. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим типом данных.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного). Простейшее объявление указателя на объект имеет вид:

Тип \*имя;

Тип может быть любым, кроме ссылки и битового поля [], причем тип к этому моменту может быть только объявлен, но еще не определен. Поэтому в структуре, например, может присутствовать указатель на структуру того же типа.

Звездочка относится непосредственно к имени, поэтому при объявлении нескольких указателей требуется ставить ее перед каждым именем, например:

char \*c1, \*ch, \*c2;

int \*n,\*m;

float \*a, \*b;

Здесь в первом случае объявлены три указателя на символьный тип с именами c1, ch, c2, во втором случае – два указателя на целое с именами n, m, в третьем случае – два указателя на тип float с именами a, b.

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, в одной и той же области памяти в разные моменты времени предполагается хранить адреса объектов разных типов).

Указателю на void можно присвоить значение указателя любого типа, а также сравнить его с любым указателем, но перед выполнением каких-либо действий с областью памяти, адрес которой он хранит, требуется преобразовать его к конкретному типу явным образом.

Указатель на функцию содержит адрес в сегменте кода, по которому располагается исполняемый код функции, т.е. адрес, по которому передается управление при вызове функции. Указатели на функцию используются для косвенного вызова функции: не через ее имя, а через обращение к переменной, хранящей ее адрес, а также для передачи имени функции в другую функцию в качестве параметра. Указатель на функцию имеет тип “указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа”:

Тип (\*имя) (список типов аргументов);

Например, float (\*fun1) (float, float); задает указатель с именем fun1 на функцию, возвращающую значение типа float и имеющую два аргумента типа float.

Указатель может быть константой или переменной, а также указывать на константу или переменную. Рассмотрим следующие примеры:

float a; //переменная типа float

const float b=2.1; //константа типа float

float \*p; //указатель на переменную типа float

const float \*pa; //указатель на константу типа float

float \* const cc=&a;//указатель-константа на переменную типа float

const float \*const pb=&b; //указатель-константа на константу типа float

Модификатор const, находящийся между именем указателя и звездочкой, относится к самому указателю и запрещает его изменение, а const слева от звездочки задает постоянное значение, на которое он указывает. Для инициализации указателей используется операция получения адреса &. Значения указателей можно вводить и выводить. Для этого используется спецификация p.

Например,

float \*p10,\*p11;

scanf("%p %p",&p10,&p11);

printf("\n p10=%p p11=%p",p10,p11);

При этом надо иметь в виду, что адрес представляет собой восьмиразрядное шестнадцатеричное число, например, вводились значения 0022FFAB и 0022FFBB. При этом следует заметить, что ввод адресов большого смысла не имеет, т.к. пользователю не известно, какая информация хранится в памяти по заданному адресу.

Начальное значение адресу можно присвоить при его объявлении, т.е. выполнить инициализацию. Это можно сделать разными способами.

Во-первых, с помощью операции получения адреса:

float a=6.6; float \*p=&a; или float \*p(&a);

Во-вторых, млжно присвоить значение другого инициализированного указателя:

float a=6.6; float \*p=&a; float \*pp=p;

В-третьих, можно присвоить указателю адрес некоторой области памяти явно:

char \*ch=(char \*)0x0044FFBB;

Шестнадцатеричная константа 0x0044FFBB с помощью операции приведения типов (char \*) преобразуеться к типу “указатель на char ”.

В-четвертых, если значение указателю будет присваиваться в ходе выполнения программы, то целесообразно в начале программы присвоить ему значение пустого указателя NULL (0). Это в определенной смысле аналог нулевого значения для числовых типов. Поскольку объектов с нулевым значением адреса не существует, то пустой указатель используется для проверки, ссылается ли указатель на конкретный объект или нет. Присваиивание нулевого значения указателю часто помогает избежать ошибок в программах, связанных с использованием неинициализированных указателей.

**Операции с указателями**

С указателями можно выполнять следующие операции: разадресация (косвенное, т.е. по адресу, обращение к объекту), присваивание, сложение с константой, вычитание, инкремент, декремент, сравнение, приведение типов.

Операция разадресации, или разыменования, предназначена для доступа к содержимому памяти, адрес которой хранится в указателе.. Эту операцию можно использовать как для получения хранимого в памяти значения, так и для изменения этого значения (если эта величина не объявлена как константа). Операция разадресации записывается как \*<имя переменной типа указатель>. Например:

float \*d, b,x;

b=4.89;

x=2.2;

d=&b;

\*d=\*d/(\*d+x);

В данном примере указателю d будет присвоено значение адреса переменной b, а затем содержимое этой области памяти (фактически значение переменной b) будет изменено в соответствии с записанным оператором присваивания, т.е. в область памяти, адрес которой хранит переменная a, занесется содержимое этой области (значение b), деленное на сумму слагаемых, одно из которых есть то же самое, что и числитель, а второе – значение переменной x.

Указателю можно присваивать значение адреса (показано ранее), а также значение другого указателя такого же типа (связанного с тем же типом). Присваивание указателя другого типа требует явного приведения типов. Это иллюстрируется следующими примерами:

float \*d,\*b, c=7.8;

int m=9;

b=&c;

d=b;

printf("res=%6.2f",\*d);

n=&m;

d=(float \*)&m;

printf("res=%6.2f",\*d);

Однако надо иметь в виду, что в первом случае результат будет отпечатан верный, т.е. на экране пользователь увидит значение 7.8. Во втором случае содержимое памяти, где хранится целое значение 9, будет рассматриваться как действительная величина и результат будет получен, отличный от 9.

Присваивание значения без явного преобразования типов допускается также в случае использования указателей void \*, стоящих в левой части оператора присваивания:

void \*p1;

float \*a;

p1=a;

Обратное присваивание a=p1; является неверным, поэтому требуется преобразование типов в явном виде: a=(float \*)p1;

Таким образом, неявное преобразоапние указателей производится только к типу void. Присваивание указателей на объекты указателям на функции (и наоборот) недопустимо. Запрещено присваивать значения указателям-константам .

Например, последовательность операторов

float aa;

float \* const cc; // cc является указателем-константой

cc=&aa;

является неправильной, т.к. делается попытка присваивания указателю-константе некоторго значения (адреса переменной).

Указателям на константу и переменную допускается присваивать значения.

Приводимая ниже последовательность операторов является допустимой

float aa;

const float \*pa; // pa является указателем на константу

pa=&aa;

Здесь указателю на константу присваивается значение адреса переменной.

Арифметические операции с указателями автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателячм одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например, с массивами.

Сложение указателя с константой:

float \*a;

const int n=4;

a=a+n;

Аналогично можно вычитать константу, т.е. складывать с отрицательным значением:

a=a-n;

В приведенном примере при сложении с константой указатель фактически получает значение a+n\*sizeof(float), т. к. в данном случае указатель связан с типом float.

Разность двух указателей – это разность их значений, деленная на размер в байтах элемента типа данных, с которым связан указатель. Например, можно записать следующую последовательность операторов:

float aa,bb;

float \*pr1,\*pr2;

int nr;

pr1=&bb;

pr2=&aa;

nr=pr1-pr2;

printf("\npr1=%p pr2=%p nr=%4d\n",pr1,pr2,nr);

Результатом будет строка вывода:

pr1=0043FAD4 pr2=0043FAE0 nr= -3

Из этой записи видно, что разность двух указателей является величиной целочисленной, показывающей фактически, каким по счету является элемент, на который указывает первый указатель, относительно элемента, на который указывает второй указатель. Применять вычитание просто к указателям, смысла нет, т.к. между перменными, на которые указывают рассматриваемые указатели, в памяти могут размещаться данные разных типов, занимающие в памяти, соответственно, и разное количество байт. Поэтому разность указателей используют при работе с массивами, когда элементы массива размещаются в соседних ячейках памяти.

Инкремент (++) и декремент (--) являются по сути частными случаями сложения указателя с константой. Эти операции прибавляют к указателю или вычитают из указателя величину sizeof(тип), где тип – тип данных, с которым связан указатель.

С указателями можно выполнять операции отношения. Например, если pr1, pr2 – однотипные указатели, то можно записать следующие операторы:

printf("\npr1=pr2 =%d pr1!=pr2 =%d pr1>pr2=%d pr1<pr2=%d\n",

pr1==pr2,pr1!=pr2,pr1>pr2,pr1<pr2);

printf("\npr1>=pr2 =%d pr1<=pr2=%d\n",pr1>=pr2,pr1<=pr2);

Результатом будут строки выода:

pr1=pr2 =0 pr1!=pr2 =1 pr1>pr2=0 pr1<pr2=1

pr1>=pr2 =0 pr1<=pr2=1

Указатели нельзя складывать, умножать, делить.

**Приоритеты операций при работе с указателями.**

Операции разадресации, инкремента, декремента и взятия адреса имеют одинаковый приоритет и выполняются справа налево. Рассмотрим примеры выполнения этих операций.

Пример 1. \*++p – здесь сначала выполнится операция инкремент указателя (сложение значения с единицей, т.е. прибавление к его значению количества байт, отводимых для хранения данных того типа, с которым связан указатель), а затем будет получено содержимое памяти по новому адресу.

Пример 2. \*p++ - в данном выражении сначала будет использовано значение переменной, хранящейся по адресу p, а затем указатель увеличится на величину, равную количеству байт, занимаемых значением типа, с которым связан указатель, т.к. инкремент является здесь постфиксным.

Пример 3. (\*p)++ - в этом выражении значение переменной, хранящейся по адресу p, увеличится на единицу.

Пример 4. ++\*p – в приведенном выражении производится сначала разадресация и полученное значение переменной, хранящейся по адресу p, увеличивается на единицу.

Пример 5. ++\*++p – в данном выражении сначала увеличивается значение указателя на единицу, затем производится разадресация и полученное значение переменной, хранящейся по новому адресу, увеличивается на единицу.

Пример программы 1. Следующая программа иллюстрирует выполнение рассмотренных примеров.

int main() {

float a[5];

float \*p;

a[0]=7.7; a[1]=8.8;

a[2]=9.9; a[3]=10.8; a[4]=17.7;

p=a;

printf("a[0]=%6.2f \*p=%6.2f p=%p",a[0],\*p,p);

printf("\na[1]=%6.2f \*++p=%6.2f ++p=%p adres [1]=%p", a[1],\*++p,p,&a[1]);

printf("\na[1]=%6.2f ",a[1]);

(\*p)++ ;

printf("\n (\*p)++=%6.2f",\*p);

++\*p;

printf("\n ++\*p=%6.2f",\*p);

++\*++p ;

printf("\n ++\*++p=%6.2f",\*p);

return 0;

}

**Взаимосвязь массивов и указателей**

В языке C существует очень тесная взаимосвязь между указателями и массивами. Любую операцию, выполняемую с помощью индексации массива, можно выполнить с применением указателей, причем в этом случае программа обычно работает быстрее [5]. В приведенной выше программе был объявлен массив float a[5]; и указатель на действительную величину float \*p;. Поскольку имя массива в языке C есть не что иное, как адрес первого элемента массива, то присваивание p=a; означает, что указатель теперь хранит адрес первого элемента массива a. То же самое можно было выполнить и с помощью оператора присваивания p=&a[0]; Если выполнить оператор присваивания p+=i;, то указатель будет хранить адрес i-го элемнта, расположенного после p, в данном случае просто i-ого элемента массива a. Тогда выражение \*(p+i) означает не что иное, как содержимое i-го элемента массива, т.е. a[i]. При вычислении выражения a[i] компилятор сам преобразует его в форму \*(a+i), т.к. эти две формы являются тождественными. Если применить к этим двум выражениям адресную операцию & то можно получить также две тождественные записи: &a[i] и a+i. Указатель можно также использовать в выражениях с индексом: p[i] будет соответствовать записи \*(p+i). Таким образом, выражение в виде обращения к элементу массива по индексу эквивалентно ссылке по указателю со смещением.

Однако есть и различие между именем массива и указателем. Укеазатель является переменной, поэтому операторы p=a; p++; допустимы, а операторы с именем массива, подобные приведенным, недопустимы: a=p; a++;

**Динамические переменные и массивы**

**Динамические переменные**

Динамические переменные – это переменные, память для хранения которых выделяется в ходе выполнения программы. Такие переменные хранятся в динамической памяти переменного размера (куче). Обращение к таким переменным выполняется с помощью указателей, хранящих их адреса. Фактически у динамических переменных. нет имен. Создание динамической переменной означает выделение для нее памяти в соответствии с ее типом (занимаемым объемом) в динамической области памяти, занесение в выделенный участок памяти значения и использование этого значения в ходе выполнения программы. Для выделения памяти под динамическую переменную используется операция new (введена в языке C++):

new <базовый тип указателя>.

В результате выполнения этой операции находится свободный участок памяти в динамической области, достаточный для хранения переменной указанного типа, выделяется требуемая память и адрес начала выделенной области помещается в указатель, при этом выделенный участок памяти определяется как занятый. Оператор присваивания переменной - указателю полученного адреса выделенной памяти может быть записан следующим образом:

<переменная - указатель> = new <базовый тип указателя>;

Занятый участок памяти можно освобождать, т.е. возвращать в кучу. Для этого используется операция delete (введена в языке C++):

delete <указатель>;

Для доступа к выделенной области памяти используется разыменование указателя (разадресация). Конструкция \*<.имя переменной - указателя> фактичеки является динамической переменной, к которой можно применять все операции, допустимые для переменных базового типа.

Пример программы 2. Следующая программа демонстрирует работу с динамическими переменными на примере ввода и выполнения четырех арифметических операций с данными целого и действительного типов.

#include "stdio.h"

int main()

{

setlocale(0,”Rus”);

int \*m,\*n,\*sk,\*rk,\*pk,\*dk;

m=new int; n=new int; sk=new int; //выделенеие динамической памяти

rk=new int; pk=new int; dk=new int;

printf("Введите два целых числа: ");

scanf("%d %d",m,n);

\*sk=\*m+\*n; \*rk=\*m-\*n; \*pk=\*m\*\*n; \*dk=\*m/(\*n);

printf("\n s=%3d rasn=%3d pr=%3d ch=%3d",\*sk,\*rk,\*pk,\*dk);

float \*a,\*b,\*sr,\*rr,\*pr,\*dr;

a=new float; b=new float; sr=new float; rr=new float; pr=new float;

dr=new float;

printf("\nВведите два вещественных числа: ");

scanf("%f %f",a,b);

\*sr=\*a+\*b; \*rr=\*a-\*b; \*pr=\*a\*\*b; \*dr=\*a/(\*b);

printf("\n s=%6.2f rasn=%6.2f pr=%6.2f ch=%6.2f \n",\*sr,\*rr,\*pr,\*dr);

delete m,n,sk,rk,pk,dk,a,b,sr,rr,pr,dr; //освобождение памяти от динамических //переменных

return 0;

}

**Динамические массивы**

Память для динамических массивов выделяется в ходе выполнения программы в соответствии с фактичеким количеством элементов. Это создает определенные удобства по сравнению с использованием статических массивов, т.к. каждый раз выделяется минимально необходимый объем памяти для хранения элементов массива. При использовании статических массивов приходится выделять память с запасом, т.е. в расчете на максимально возможное (прогнозируемое) количество элементов. Объявление указателя на динамический массив записывается так же, как и на динамическую переменную:

<тип компонентов массива> \*<идентификатор указтеля>;

Для выделения памяти под массив используется операция new, при этом необходимо указать тип элементов массива и их количество:   
<указатель-идентификатор массива>=new <тип элементов массива> [<количество>];

В результате выполнения операции выделится непрерывная область памяти для размещения заданного количества элементов масива. Объем памяти определяется как произведение количества элементов на объем памяти, занимаемый одним значением указанного типа. Адрес начала выделенной области памяти заносится в указатель.

Для освобождения памяти используется оператор delete, размерность при этом не указывается:

delete[]<идентификатор указателя>;

Пример программы 3. Приводимая далее программа поясняет работу с динамическим одномерным массивом. В программе вводится количество элементов массива, затем выделяется память, выполняется ввод элементов динамического массива, а затем находится максимальный элемент массива и вычисляется произведение всех его элементов. В заключение выводятся элементы массива, полученный результат, после чего выполняется освобождение выделенной памяти.

int main()

{

setlocale(0,”Rus”);

int n,i;

float \*p,max,pp;

printf("Введите количество элементов: ");

scanf("%d",&n);

p=new float[n];

printf ("Введите элементы массива ");

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

scanf("%f",&p[i]);

printf ("Вывод массива ");

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

printf("%6.2f",p[i]);

max=p[0];

pp=1;

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

{

if (p[i]>max)

max=p[i];

pp\*=p[i];

}

printf("\n max=%6.2f pr=%6.2f\n",max,pp);

delete []p;

return 0;

}

**Двумерные динамические массивы**

Двумерный динамический массив (матрицу) удобно рассматривать как одномерный массив, состоящий из одномерных массивов. Поэтому каждой строке матрицы (одномерному массиву) можно сопоставить указатель, хранящий адрес первого элемента очередной строки. Совокупность этих указателей образует одномерный массив указателей, этому массиву можно сопоставить указатель на массив указателей, т.е. указатель на указатель. В связи с этим при объявлении динамической матрицы следует объявить указатель на указатель:

<тип элементов массива> \*\* <идентификатор указателя>;

Идентификатор указателя является в данном случае указателем на массив указателей. После объявления указателя следует выделить память под матрицу, причем это делается в два этапа. Сначала надо выделить память для массива указателей. Подобный массив может быть объявлен следующим образом:

<тип элементов> \*<идентификатор>[<количество строк матрицы>];

В этом объявлении используется правило: “суффикс крепче префикса”, т.е. данная конструкция воспринимается как массив указателей (а не указатель на массив). Память под массив указателей выделяется, как и ранее, операцией new:

<идентификатор> = new <тип элементов> \*[ количество строк];

Например:

a=new float \*[20];

n=new char \*[30];

В первом случае выделяется память под массив указателей на тип float, состоящий из 20-ти элементов, адрес заносится в переменную-указатель a. Во втором случае выделяется память под массив указателей на тип char, состоящий из 30-ти элементов, адрес массива заносится в переменную-указатель n.

На втором этапе необходимо выделить память для хранения всех элементов каждой строки матрицы. Обычно это выполняют в цикле с помощью операции new, при этом очередному указателю (элементу массива) присваивается адрес первого элемента очередной строки матрицы. Здесь следует заметить, что количество элементов строк динамической матрицы может быть различным, поэтому можно работать с динамическими матрицами различной конфигурации. Доступ к элементам массива указателей при выделении памяти подстроки динамической матрицы можно выполнять несколькими способами: с помощью смещения от начала массива, с помощью индекса, с помощью текущего указателя. Приведем примеры использования каждого способа для первой матрицы:

1) использование смещения

for (int i=0; i<20;i++)

\*(a+i)=new float[i+1];

2) использование индекса

for (int i=0; i<20;i++)

a[i]=new float[i+1];

3) использование вспомогательного текущего указателя:

float \*\*p;

for (p=a; p<a+20;p++)

\*p=new float[i+1];

Освобождение памяти, выделенной под динамическую матрицу, выполняется также в два этапа: сначала освобождается память, выделенная под каждую строку матрицы, а затем освобождается память, выделенная под массив указателей на каждую строку матрицы. В приведенном примере это может выглядеть следующим образом:

for (int i=0;i<20;i++)

delete [] a[i];

delete []a;

**Практика**

**Указатели**

**Пример 1.** Распечатываем значение указателя, содержимое переменной и квадрат переменной **(lab7\_1.cpp).**

**Пример** 2. Создание указателей в динамической памяти **(lab7\_2.cpp).**

**Пример** 3. Рассмотрим пример **(lab7\_3.cpp).** Из примера следует, как можно работать с массивами в терминах указателей. Также очевидно, что если к указателю прибавить целое число, то это эквивалентно использованию элемента массива с соответствующим индексом. Из примера также следует, что при обработке выражений, в которых используются указатели и целые числа, транслятор учитывает тип переменных, для которых построен указатель.

**Динамические массивы**

**Пример 4.** Выделение памяти с помощью функции calloc **(lab7\_4.cpp)**

**Пример 5.** Выделение памяти с помощью функции malloc **(lab7\_5.cpp)**

**Пример 6.** Выделение памяти с помощью функции realloc **(lab7\_6.cpp)**

**Пример 7.** Выделение памяти с помощью функции new **(lab7\_7.cpp).** Найти минимальный и максимальный элементы, и умножить каждый элемент массива на их отношение. Создать отдельные функции для ввода, вывода и преобразования массива. (Подсказка: если динамический массив создается не в функции main, то в функцию надо передавать адрес этого указателя, чтобы возвратить из функции результат, а внутри функции работа будет выполняться с указателем на указатель). И в примере **(lab7\_7 new в main.cpp)** показан тот же пример, только память выделяется не внутри функции input(), а в main().

**Для всех заданий использовать динамическое выделение памяти!**

**Задание на "Массивы":**

Задание 1. Из массива Х, упорядоченного по невозрастанию значений элементов, переписать в массив Y числа, исключив их повторы и обеспечив упорядоченность по возрастанию.

**Задание на "Матрицы":**

Задание 2. Выполнить работу с динамической матрицей: выделить память под динамическую матрицу и одномерный динамический массив, осуществить ввод элементов матрицы, а затем вычислить сумму элементов каждой строки матрицы и занести в одномерный массив.

Задание 3. Составить программу, которая находит в каждой строке матрицы Q(k,l) сумму положительных элементов, расположенных между первым и последним отрицательными элементами этой же строки. Если сумму вычислить нельзя (нет положительных элементов между первым и последним отрицательными элементами), то считать ее равной нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых - найденную сумму, первый и последний отрицательные элементы.

Задание 4. Выполнить операцию вычитания двух треугольных матриц (Использование динамических массивов позволяет работать с матрицами произвольной формы, т.е. каждая строка матрицы может состоять из любого количества элементов, изменяющегося от строки к строке). Создать отдельные функции для ввода, вывода и вычитания матриц.