**Тема:** Указатели

**Цель:** Углубить знания и умения работы в среде программирования С

**Задачи:**

1. Изучить теоретический материал.
2. Выполнить упражнения.
3. Написать комментарии к примерам приведенным в лекции.

Указатель – это адрес поля памяти, занимаемого программным объектом. Пусть в программе определены три переменные разных типов: int a=5; char с='G'; float r=1.2E8; Эти величины разместились в памяти компьютера следующим образом:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Память | FFC0 | FFC1 | FFC2 | FFC3 | FFC4 | FFC5 | FFC6 |  |
| Переменные | a | | c | r | | | | |
| значения | 5 | | ‘G’ | 1,2\*108 | | | | |

**Операция & — адрес.** Применение этой операции к имени переменной дает в результате ее адрес в памяти. Для переменных из данного выше примера: &а равно FFC0, &с - FFC2, &r - FFC3. **Описание указателей.** Для хранения адресов используются переменные типа «указатель». Формат описания таких переменных следующий: тип \*имя\_переменной

Примеры описания указателей:

int \*pti; char \*ptc; float \*ptf;

После такого описания переменная pti может принимать значение указателя на величину целого типа; переменная ptc предназначена для хранения указателя на величину типа char; переменная ptf — на величину типа float.

Указателям могут присваиваться значения адресов объектов только того типа, с которым они описаны. В нашем примере допустимы операторы

pti=&a; ptc=&c; ptf=&r;

В результате указатели примут следующие значения:

pti - FFCO, ptc - FFC2, ptf - FFC3.

Как и для других типов данных, значения указателей могут инициализироваться при описании. Например:

int a=5; int \*pti=&a;

char c='G'; char \*ptc=&c;

float r=1.2E8; float \*ptf=&r; 223

В заголовочном файле stdio.h определена константа — нулевой указатель с именем NULL. Ее значение можно присваивать указателю. Например:

ptf=NULL;

Не надо думать, что после этого указатель ptf будет ссылаться на нулевой байт памяти. Нулевой указатель обозначает отсутствие конкретного адреса ссылки.

Использованный в описаниях указателей символ \* (звездочка) в данном контексте является знаком операции разадресации. С ее помощью можно сослаться через указатель на соответствующую переменную.

После приведенных выше описаний в записи выражений этой программы взаимозаменяемыми становятся а и \*pti, с и \*ptc, г и \*ptf. Например, два оператора

х=а+2; И x=\*pti+2; тождественны друг другу. В результате выполнения оператора cout<<\*pti<< a;

на экран выведется 55.

**Операции над указателями.** Записывая выражения и операторы, изменяющие значения указателей, необходимо помнить главное правило: единицей изменения значения указателя является размер соответствующего ему типа.

Продемонстрируем это правило на определенных выше указателях.

Выполнение операторов pti=pti+1; ИЛИ pti++; изменит значение указателя pti на 2, в результате чего он примет значение FFC2. В результате выполнения оператора pti --; значение указателя уменьшится на 2 и станет равным FFBE.

Аналогично для указателей других типов:

ptc++;увеличит значение указателя на 1;

ptf++; увеличит значение указателя на 4.

**Использование указателей для передачи параметров функции**. Рассматривая ранее правила использования функций, мы обращали внимание на то, что в языке Си возможна только односторонняя передача значений фактических параметров из вызывающей программы к формальным параметрам вызываемой функции. Возвращаемое значение несет сама функция, используемая в качестве операнда в выражении. Отсюда, казалось бы, следует неукоснительное правило: в процессе выполнения функции не могут изменяться значения переменных в вызывающей программе. Однако это правило можно обойти, если в качестве параметров функции использовать указатели.

В следующем примере функция swap () производит обмен значениями двух переменных величин, заданных своими указателями в аргументах.

void swap(int \*a,int \*b)

{ int с;

c=\*a; \*a=\*b; \*b=c;

}

Если в основной программе имеется следующий фрагмент:

int x=l,у=2;

swap(&х,&у);

printf("x=%d y=%d",x,y);

то на экран будет выведено: х=2 у=1

т. е. переменные хи у поменялись значениями.

передача параметров здесь тоже происходит по значению, только этими значениями являются указатели. После обращения к функции указатель а получил адрес переменной х, указатель b — адрес переменной у. После этого переменная х в основной программе и разадресованный указатель \*а в функции оказываются связанными с одной ячейкой памяти; так же — у и \*b.

Таким образом, можно сделать вывод о том, что использование указателей в параметрах функции позволяет моделировать работу процедур.

**Указатели и массивы.** Имя массива трактуется как указатель-константа на массив. Пусть, например, в программе объявлен массив:

int X[10];

В таком случае х является указателем на нулевой элемент массива в памяти компьютера. В связи с этим истинным является отношение

Х==&Х[0]

Отсюда следует, что для доступа к элементам массива кроме индексированных имен можно использовать разадресованные указатели по принципу:

имя [индекс] тождественно \* (имя + индекс)

Например, для описанного выше массива х взаимозаменяемы следующие обозначения элементов:

Х[5], ИЛИ \*(Х+5),ИЛИ \*(5+Х).

Напоминаем, что для указателей работают свои правила сложения. Поскольку х — указатель на величину целого типа, то Х+5 увеличивает значение адреса на 10.

В языке Си символ [ играет роль знака операции сложения адреса массива с индексом элемента массива.

Из сказанного должно быть понятно, почему индекс первого элемента массива всегда нуль. Его адрес должен совпадать с адресом массива:

Х[0]==\*(Х+0)

Поскольку имя массива является указателем-константой, то его нельзя изменять в программе, т. е. ему нельзя ничего присваивать. Например, если описаны два одинаковых по структуре массива

int X[10],Y[10];

то оператор присваивания X=Y будет ошибочным. Пересылать значения одного массива в другой можно только поэлементно.

Теперь рассмотрим двумерные массивы. Пусть в программе присутствует описание:

int P[5][10] ;

Это матрица из пяти строк и десяти чисел в каждой строке. Двумерный массив расположен в памяти в последовательности по строкам. По-прежнему Р является указателем-константой на массив, т. е. на элемент р [ 0 ] [ 0 ] . Индексированное имя р [ i ] обозначает i-ю строку. Ему тождественно следующее обозначение в форме разадресованного указателя: \*(P+i\*10) Обращение к элементу массива р[2] [4] можно заменить на \*(Р+2\*10+4) . В общем случае эквивалентны обозначения:

P[i][j] И \*(P+i\*10+j)

Здесь дважды работает операция «квадратная скобка». Последнее выражение можно записать иначе, без явного указания на длину строки матрицы р: \*(\*(P+i)+j) .

Очевидно, что по индукции для ссылки на элемент трехмерного массива A[i] [j] [k] справедливо выражение \* (\* (\* (A+i)+j)+k) и т.д.

**Массив как параметр функции.**

Пример 1. Составим программу решения следующей задачи.

Дана вещественная матрица А[м] [N]. Требуется вычислить и вывести евклидовы нормы строк этой матрицы.

Евклидовой нормой вектора называют корень квадратный из суммы квадратов его элементов:

Если строку матрицы рассматривать как вектор, то данную формулу надо применить к каждой строке. В результате получим M чисел.

Определение функции вычисления нормы произвольного вектора:

double Norma(int n, double X[])

{ int i;

double S=0;

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

S+=X[i]\*X[i];

return sqrt (S);

}

Заголовок этой функции можно было записать и в такой форме:

double Norma(int n, double \*X)

В обоих случаях в качестве второго параметра функции используется указатель на начало массива. Во втором варианте это более очевидно, однако оба варианта тождественны.

При вызове функции Norma () в качестве второго фактического параметра должен передаваться адрес начала массива (вектора).

Рассмотрим фрагмент основной программы, использующей данную функцию для обработки матрицы размером 5x10.

void main()

{ double A[5][10]; int i; //Ввод матрицы

//Вычисление и вывод нормы строк

for(i=0; i<5; i++)

cout«"HopMa"<<i<<"-и строки=" «Norma (10, A[i]) ;

}

В обращении к функции второй фактический параметр А [ i ] является указателем на начало i-й строки матрицы А.

Пример 2. Заполнить двумерную матрицу случайными целыми числами в диапазоне от 0 до 99. Отсортировать строки полученной Матрицы по возрастанию значений. Отсортированную матрицу вывести на экран.

#include <iostream.h>

#include <iomanip.h>

#include <conio.h>

#include <stdlib.h>

const n=5;//Глобальное объявление константы /

/Прототипы функций

void Matr(int M[][n]);

void Sort(int, int X[]);

//Основная программа

void main()

{

int i,j, A[n] [n] ;

clrscr () ;

cout«"\n"<<"MaTpицa до сортировки:"<<"\n";

Matr(A);

for(i=0; i<n; i++) Sort(n, A[i]);

cout<<"\n"<<"MaTpица после сортировки: "<<"\n";

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

{

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

cout<< setw(6) <<A[i] [j] ;

cout<< endl;

}

}

// Функция сортировки вектора

void Sort(int k, int X[])

{int i,j, Y;

for(i=0; i<k-1; i++)

for(j=0; j<k-i-1; j++)

if(X[j]>X[j+l])

{Y=X[j];

X[j]=X[j+1];

X[j+1]=Y;} }

//Функция заполнения матрицы и вывода на экран

void Matr(int M[][n])

{int i,j;

randomize{); //Установка датчика случайных чисел

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

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

{ M[i] [j]=rand()%100;

cout<<setw (6) <<M[i] [j ] ; }

cout<<endl; } }

Обратите внимание на прототип и заголовок функции Matr () . В них явно указывается вторая размерность параметра матрицы. Первую тоже можно указать, но это необязательно. Как уже говорилось выше, двумерный массив рассматривается как одномерный массив, элементами которого являются массивы (в данном случае — строки матрицы). Компилятору необходимо «знать» размер этих элементов. Для массивов большей размерности (3, 4 и т.д.) в заголовках функций необходимо указывать все размеры, начиная со второго. При обращении к функции

Matr () фактическим параметром является указатель на начало двумерного массива А, а при обращении к функции Sort () — указатели на начало строк.

Упражнения 1.

1.В оперативной памяти вектор int X[10] расположен, начиная с адреса B7F0. Какие значения примут выражения: а) х+1; б) х+5; в) х-4?

2. В программе объявлен массив:

int P[]={0,2,4,5,6,7,9,12};

Какие значения примут выражения:

а) Р[3]; б) \*р; в) \*(Р+4); Г) \*(Р+Р[2])?