Тема: «Указатели»

Вопросы:

1. Указатели и операции над ними. Связь между указателями и массивами.
2. Организация работы с динамическими массивами. Примеры.

Назначение указателей

Указатели позволяют применять язык C++ в самом широком диапазоне задач – от драйверов устройств на уровне аппаратного обеспечения и управляющих систем реального времени до операционных систем и компиляторов, анимации и мультимедийных приложений. C++ – это идеальный инструмент для решения задач в поразительно широком многообразии прикладных областей.

Указатель – это особый тип переменной, содержащей в памяти адрес того элемента, на который он указывает. При этом имя элемента отсылает к его значению прямо, а указатель косвенно. Поэтому ссылка на значение посредством указателей называется косвенной адресацией. Смысл использования указателей состоит в том, что, отводя место только для запоминания адреса в памяти, Вы получаете идентификатор (переменную типа указатель), который может ссылаться на любой элемент в памяти указанного типа, причем в разный момент указатель может указывать на разные элементы, т.е. не надо перемещать в памяти с места на место кучи байтов, достаточно просто изменить указатель.

Указатель может указывать на любые объекты: переменные, массивы, классы, структуры, и даже на функции. При объявлении указателя необходимо указать тип элемента, на который он будет указывать. Сам указатель занимает ровно столько места, сколько требуется для хранения адреса элемента. Признаком указателя является символ *, который означает, что следующая за ним переменная является указателем. Рассмотрим примеры объявления указателей для различных типов переменных.

int *r; – указатель на целое число,

float*f; – указатель на действительное число, char*ch; – указатель на символьную переменную, long *g[10]; – массив из 10 указателей на длинное целое,

long (*t)[10]; – указатель на массив из десяти длинных целых, int(*fun)(double, int); – указатель на функцию с именем fun.

Для инициализации указателей используется операция присваивания "=". Присваивание начального значения означает, что при определении

указателя ему следует присвоить либо адрес объекта, либо адрес конкретного места в памяти, либо число 0. Примеры каждого типа инициализации выглядят так:

1. Присваивание указателю адреса существующего объекта:

С помощью операции получения адреса: int a=5; // целая переменная

int*p = &a; // указателю присваивается адрес переменной a int*p (&a); // то же самое другим способом

С помощью значения другого инициализированного указателя: int*r = p;

С помощью имени массива или функции, которые трактуются как адрес:

int b[10]; // массив

int *t = b; // присвоение адреса начала массива

2. Присваивание указателю адреса области памяти в явном виде:

char*z = (char*)0x00000417;

Здесь 0x00000417 – шестнадцатеричная константа, (char*) – операция приведения константы к типу указатель на char.

3. Присвоение пустого значения:

int*p = 0;

Поскольку гарантируется, что объектов с нулевым адресом нет, пустой указатель можно использовать для проверки, с помощью которой можно оценить: ссылается указатель на конкретный объект или нет.

4. Выделение участка динамической памяти и присвоение ее адреса указателю с помощью операции new:

int*n = new int; // 1

int*m = new int(10); // 2

int*q = new int[10]; // 3

В операторе 1 операция new выполняет выделение достаточного для размещения величины типа int участка динамической памяти и записывает адрес начала этого участка в переменную n. В операторе 2, кроме того, производится инициализация выделенной памяти значением 10. В операторе 3 операция new выделяет память под массив из 10 элементов и записывает адрес начала этого участка в переменную q, которая может трактоваться как имя массива.

Операции над указателями

С указателями связаны два специальных оператора: **&** и *****. Обе эти операции унарные, т.е. имеют один операнд, перед которым они ставятся. Операция & соответствует действию «взять адрес». Операция * соответствует словам «значение, расположенное по указанному адресу». Например:

int y = 5; int *py; py = &y; .

Здесь оператор py = &y; присваивает адрес переменной y указателю py. Говорят, что переменная py указывает на y. Оператор * обычно называют оператором косвенной адресации, или операцией разыменования, возвращающей значение объекта (элемента), на который указывает ее операнд (т.е. указатель). Например, оператор cout << *py <<”\n”; печатает значение переменной y, а именно 5. Использование * указанным способом позволяет обеспечить доступ к величине, адрес которой хранится в указателе.

Ниже приведен листинг программы, в которой рассматриваются примеры использования операций & и *.

```
#include<iostream.h> #include <conio.h> main()
{
int a; // a – целое число

int *pa ; // pa – указывает на адрес целого числа

clrscr();
a = 7;
pa = &a; // pa устанавливаем равным адресу переменной a

cout<<"\nАдрес a:"<<&a<<"\n"<<"Значение pa:"<<pa<<"\n";
cout<<"\nЗначение a:"<<a<<"\n"<<"Значение *pa:"<<*pa<<"\n\n";
cout<<"\nДоказывают, что & и * дополняют друг друга"

<<"\n"<<"&*pa="<<&*pa<<"\n"<<"*&ap="<<*&pa<<"\n";

cout<<"\n\n";

cout<<"\nНажмите любую клавишу ..."; getch();
return 0;
}
Значение pa:0xfff4
Значение a: 7 Значение *pa: 7
```

Доказательство того, что & и * дополняют друг друга

&*pa: 0xfff4 *&pa: 0xfff4

Программа демонстрирует операцию с указателями. Адреса ячеек памяти выводятся как шестнадцатеричные целые с помощью оператора вывода cout. В программе показано, что адрес переменной a и значение указателя pa идентичны, а операция разыменовывания указателя *pa выводит на печать значение переменной a. Операции & и * взаимно дополняют друг друга. При их поочередном применении к pa в любой последовательности будет напечатан один и тот же результат.

Выражения и арифметические действия с указателями

Указатели могут использоваться как операнды в арифметических выражениях, выражениях присваивания и выражениях сравнения [3]. Число арифметических операций с указателями ограничено. Указатели можно увеличивать (++), уменьшать (--), складывать с указателем целые числа (+ или +=), вычитать из него целые числа (- или -=) или вычитать один указатель из другого. Арифметические действия над указателями имеют свои особенности. Рассмотрим это на простейшем примере.

```
#include<iostream.h> #include <conio.h> main()
{
Int x; // x - целое число
int*p ,*p1; // указывают на целые числа

clrscr(); // очистка экрана
p = &x; // указателю присваивается адрес целого числа x p1=p+3;
cout<<"\nНачальное значение p:"<<p<<"\n"; cout<<"Конечное значение ++p:"<<++p<<"\n"; cout<<"Конечное значение --p:"<<--p<<"\n"; cout<<"Конечное значение p1:"<<p1<<"\n"; cout<<"\n\n";
cout<<"\nНажмите любую клавишу ..."; getch();
return 0;
}
```

Результаты работы программы:

Начальное значение
p: 0xfff4
Конечное значение ++p: 0xfff6
Конечное значение
--p: 0xfff4

Конечное значение
p1: 0xfffa

По результатам выполнения этой программы мы видим, что при операции ++p значение указателя p увеличивается не на 1, а на 2. И это правильно, так как новое значение указателя должно указывать не на следующий адрес, а на адрес следующего целого. А целое, как известно, занимает два байта памяти. Если бы базовый тип указателя был не int, a double, то были бы напечатаны адреса, отличающиеся на 8, именно столько байт памяти занимает переменная типа double, т.е. при каждой операции ++p значение указателя будет увеличиваться на количество байт, занимаемых переменной базового типа указателя, а при операции --p соответственно уменьшаться. К указателям можно прибавлять или вычитать некоторое целое. В данной программе указатель p1 представляет собой сумму значений указателя p и целого числа 3. Результат равен 0xfffa, т.е. увеличился на 6 по сравнению с исходным значением указателя p. Общая формула для вычисления значения указателя после выполнения операции p = p + + n; будет иметь вид < p > = < p > + n*< количество байт памяти базового типа указателя>. Можно так же вычитать один указатель из другого. Так, если p и p1 – указатели на элементы одного и того же массива, то операция p – p1 дает такой же результат, как и вычитание соответствующих индексов массива. Указатели можно сравнивать, при этом применимы все 6 операций: <, >, <=, >=, = , == , !=.

Сравнение p<g означает, что адрес, находящийся в p, меньше адреса, находящегося в g. Указателю можно присваивать значение другого указателя, а можно явно задать присваивание указателю адреса переменной того типа, на который он указывает. Для этого используют операцию взятия адреса &:

int r=10; int *t=&r; .

Здесь мы явно инициализировали указатель так, чтобы он указывал на переменную. Для того чтобы можно было использовать то, на что ссылается указатель через его имя, используют оператор разадресации *, т.е. при объявлении переменной символ * служит признаком указателя, а при реализации * служит знаком использования данных по адресу, если стоит перед указателем.

Пример использования операции разадресации.

```
#include <iostream.h> void main()
{
```

```
int a=1, b=2;
int *c=&b, *d=&a;
cout<< a <<' '<< b<<' '<<*c<<' '<< d<<' '<< c<<' '<< *d<<' '<< "\n";
}
```

Результаты работы программы:

a=1, b=2, *c=2, d=0xfff2, c=0xfff4, *d=1.

Массивы и указатели

Массивы и указатели в языке C++ тесно связаны и могут использоваться почти эквивалентно.

Так, имя массива является константным указателем на первый элемент массива, а указатели можно использовать для выполнения любой операции, включая индексирование массива.

Пусть сделано следующее объявление: int b[5] = {1,2,3,4,5}, *p;, которое означает, что объявлены массив целых чисел b[5] и указатель на целое p. Поскольку имя массива является указателем на первый элемент массива, можно задать указателю p адрес первого элемента массива с помощью оператора p = b;. Это эквивалентно присвоению адреса первого элемента массива другим способом: p = &b[0];. Теперь можно сослаться на элемент массива b[3] с помощью выражения *(p+3). В этой записи скобки необходимы, потому что приоритет * выше, чем +.

Ниже показан пример программы, которая с помощью указателей заносит данные в одномерный массив a[5]. Определяет сумму и количество положительных элементов. Выводит на экран полученный массив и адреса его элементов, а также результаты расчетов.

```
#include <iostream.h> #include <conio.h> void main()
{ clrscr();
int a[5], sum = 0, *p; int kol = 0, i;
p = &a[0]; // инициализация указателя адресом первого элемента cout <<
" Ввод данных в массив a[ ]\n";
for ( i = 0; i <5; i++)
{
cout << " a [ " << i << " ] = ";
cin >> *(p+i); // разыменовывание смещенного указателя
}
// расчет суммы и количества положительных элементов
for ( i = 0; i < 5; i ++)
if ( *(p+ i) > 0 )
```

```
{
sum += *( p+i ); kol ++;
}
// вывод исходных данных и результатов
cout << "\n\n\n Элемент массива Адрес элемента массива \n"; for ( i = 0; i
< 5; i++ )
{
cout << *( p+ i) << "\t\t " << (p+i) << "\n"; // вывод результатов
}
cout << "\nсумма = " << sum << "\nколичество = " << kol; cout<<"\n\n";
cout<<"\nНажмите любую клавишу ..."; getch();
}
```

Результаты работы программы:

```
Ввод данных в массив a[ ] : a[0]=1
a[1]=2
a[2]=5
a[3]=5
a[4]=4
```

Элемент массива	Адрес элемента массива
1	Oxffec
2	Oxffee
3	Oxfff0
4	Oxfff2
5	Oxfff4

сумма = 17 количество = 5

Ниже приведен пример программы, в которой с помощью указателей формируется двумерный массив a[2][2], а из минимальных элементов его столбцов формируется массив b[2]. Значения полученных массивов выводятся на дисплей.

```
#include <iostream.h>
#define I 2
#define J 2
#include <conio.h>
void main()
{
```

```
clrscr();
int a[I][J], b[J], min, *p ; int i,j;
p = &a[0][0]; // инициализация указателя адресом первой ячейки cout <<
"Введите данные в массив a[“<< I <<”][“<<J<<”]:\n"; for ( i = 0; i < I; i++ )
for ( j = 0; j < J; j++ )
{
cout << "a[" << i << "][" << j << "]="; cin >> *(p + i*I + j); // ввод массива
}
// расчет массива b[2]
for ( j = 0; j < J; j++ ) // цикл по столбцам
{
min = *(p + j); // присваивание min значения первого элемента столбца
for (i = 1; i < I; i++) // цикл по строкам, начиная со второго элемента
if ( ( *(p + i*I + j)) < min) min = *(p + i*I + j); *(b + j) = min;
}
cout << "\n\nВывод исходного массива a[,]:"; for (i = 0; i < I; i++)
{
cout << "\n";
for ( j = 0; j < J; j++)
{
cout <<"\t"<< *(p + i*I + j);
}
}
cout << "\n\nВывод полученного массива b[]:\n"; for ( j = 0; j < J; j++ )
{
cout <<"\t"<< *( b + j);
}
cout<<"\n\n";
cout<<"\nНажмите любую клавишу ..."; getch();
}
```

Результаты работы программы:

Введите данные в массив **a[2][2]: a[0][0]=1**
a[0][1]=4
a[1][0]=5
a[1][1]=3

Вывод исходного массива **a[2][2]:**

1 4
5 3

Вывод полученного массива **b[]:**

1 3