# Программирование на языке C++

Зариковская Наталья Вячеславовна

Лекция 7

# Указатели

- Указатель - особый тип данных, предназначенный для хранения адреса в памяти. В языке С определены два вида указателей: указатели на объект без определенного типа- void*; указатели на объекты определенного типа.
- Указатели на объекты определенного типа бывают также двух видов: указатели константы и указатели переменные. Указатели переменные определяются в операторах объявления с по мощью символа «*», помещенного перед идентификатором указателя.

- **Например:**
- int*p;
- long a,*c;
-
- где int* - тип указатель на некоторую переменную типа int; p - идентификатор указателя на пе ременную int*;
- a- идентификатор переменной типа long; c - идентификатор указателя на некоторую переменную типа (long*).

# Указатели

- С точки зрения программной модели указатель будет содержать адрес первого байта некоторой переменной.
- Информация о количестве байт этой переменной поставляется компилятору оператором объявления. Количество байт, занимаемых указателем зависит от разрядности платформы, под которую компилируется программа.
- Инициализация указателя выполняется с помощью оператора взятия адреса –“&”. Так если в теле программы встретился оператор объявления «long L; », тогда использование оператора-&L даст в результате адрес размещенного в памяти значения переменной L.
- Инициализация указателей может быть, выполнена в разделе оператора объявления
- Long L,*a=&L,G,*p; // a указывает на L .
- p=&G; // p указывает на G

# Указатели

- Отметим, что использование указателя в теле программы производится без использования символа «*». Поскольку допустимый диапазон значений любого указателя включает специальный адрес 0 и набор положительных целых чисел, которые интерпретируются как машинные адреса, то допускается инициализация указателей с использованием символических констант.

- **Например:**
- Long L,*a=&L,b,*p,*pt=(Long*)300;
- где указатель pt получает начальное значение 300. Использование символических констант допустимо и в теле программы.

- **Например:**
- Long L,*a=&L,b,*p,*pt=(long*)300;
- p=(long*)400; //указатель p имеет значение 400.

# Указатели

- Особое место занимает указатель, который имеет значение равное 0 - null. Этот указатель используется в качестве специального флага условия. Например: при работе с динамическими переменными исчерпана свободная память; конец динамической структуры данных (списка, стека, дерева). Следует знать, что поскольку указатель-переменная тоже имеет адрес, то возможна организация косвенной адресации (в указателе хранится адрес другого указателя на какую-либо переменную).

- **Например:**
- Long L,*p,*a=&L;
- p=(long*)&a; //p хранит адрес указателя a на переменную L.

# Указатели

- При работе с указателями, часто используется операция взятия адреса- «&». Поэтому необходимо знать ограничения, накладываемые на эту операцию.
- Эти ограничения следующие:
  - нельзя определить адрес символической константы;
  - нельзя определить адрес арифметического выражения;
  - нельзя определить адрес переменной, описанной как register.

**Например**, перечень недопустимых выражений может быть следующим

- unsigned register var; int a,*p;
- .....
- p=&0xf700;//нельзя определить адрес символьной константы
- .....
- //нельзя определить адрес арифметического выражения
- p=&(a*5);
- .....
- //переменная var описана как register, что недопустимо.
- p=&var;

# Указатели

- Указатель типа void*, как указатель на объект неопределенного типа, также имеет отличи тельные свойства. А именно, ключевое слово void извещает компилятор об отсутствии информации о размере объекта в памяти. В связи с этим при описании переменных - указателей типа void* необходимо выполнить операцию явного приведения типа.

- **Например:**
- unsigned long a=0xffccdd77L;
- void*p=&a;

# Указатели

- Для указателей разрешены некоторые операции: присваивание; инкремента и декремента; сложение и вычитание; сравнение. Особенность использование этих операций состоит в том, что результат арифметических операций зависит от типа указателя или вернее от типа объекта на который ссылается указатель.
- Так для представленного арифметического выражения следующего вида
- указатель_0_типа=указатель_0_типа+N,
- (где **указатель_0_типа**- указатель на объект определенного типа-**_0_типа**; n- константа целого типа), указатель_0_типа изменяется на величину N*sizeof(_0_типа). Например:
- {
- int*p=(int*)200,*p1=(int*)220;
- long*p2=(long*)200,*p3=(long*)300;
- long double p4=(long double)200, p5=(long double)300;
- printf("p1-p=%d,p3-p2=%d,p5-p4=%d,p4-p5=%d",p1-p,p3-p2,p5-p4,p4-p5);
- },
- в результате, приведенных выше арифметических операций, будут получены следующие значения
- p1-p=10; p3-p2=25; p5-p4=10; p4-p5=-10.

# Указатели

- Указатели, как было отмечено выше, могут сравниваться между собой. Не вдаваясь в под робности способа формирования физического адреса, отметим, что результаты сравнения не всегда могут быть корректными. Имеет смысл лишь сравнения на равенство и неравенство. Однако, если p1 и p2 указатели на элементы одного и того же массива, то к ним можно применять все допустимые операции с указателями (==,!=,<,>,>=,-,+).

- **Например,** пример решения задачи определения количества символов в заданной строке:
- int strlen(char *s) // функция возвращает длину строки
- { char *p=s; // отметка начала
- while( *p!=’\0’’)
- p++; // на выходе отметка конца
- return p-s;} // конец - начало = длина

# Разыменование

- Операция разыменования - одноместная операция («*») с тем же приоритетом и ассоциатив ностью справа налево, что и другие одноместные операции. Смысл этой операции состоит в переходе от указателя к значению объекта на который он указывает. Таким образом, если в программе имеется объявление
- int a,*p;  // p - указатель на int
- a=1;  .....  p=&a; то выражение ++*p; //*p=*p+1;
- означает - увеличить значение переменной целого типа a на единицу (a=2)
- Как видно, из вышеописанного примера, унарная операция разыменования -‘*’, используемая в теле программы, идентифицируется компилятором по наличию соответствующего объяв ления указателя.
- При программировании с использованием переменных типа указатель необходимо обращать особое внимание на порядок выполнения действий.
- **Например:**
- *p++;  // соответствует *(p+1) или *(p++ )
- ++*p;  // *p=*p+1

# Оператор указателя на структуру

- C++ в дополнение к оператору « . » , используемого для доступа к членам структуры , поддерживает оператор указателя на структуру « ->». Оператор указателя на структуру печатается на клавиатуре как сочетание знаков «минус» и «больше».
- Если определён указатель на структуру, то доступ к элементам структуры можно обеспечить двумя способами:
- классическим - используя, оператор разыменования
- 
- **(*указатель_на_структуру).имя_члена ;**
- 
- используя, оператор указателя на структуру
- 
- **указатель_на_структуру-> имя_члена ;**
- 

- Например, при решении задачи, приведенной ниже, поиска фамилии студента родившегося в искомом месяце использованы два способа обращения к полям структуры.

```
# include <iostream.h>
# include <stdio.h>
# include <string.h>
# define  n 2
void main()
{struct stud
{char fam[20];
char gr[7];} std[n],*p=&std[0]; //после объявления структур, объявляем массив студентов размером n (n=2), и указатель на студента, которому присваиваем указатель на первого студента
int i,k;
char m[7];
for(i=0;i<n;i++,p++)
{cout<<"\n"<<"введите фамилию \n";
//обращение к полю fam используя, оператор -> указателя на структуру
cin>>p->fam;
 cout<<"\n"<<"введите месяц\n";
cin>>(*p).gr;
// обращение к полю gr используя, оператор разыменования
cout<<'\n'<<((*p).fam)<<" "<<((*p).gr);}
p-=n;
cout<<"\n"<<"введите искомый месяц\n";
cin>>m;
k=0;
for(i=0;i<n;i++,p++)
{if(strcmp(((*p).gr),m)==0)
cout<<(*p).fam<<'\n';
else
{k++;
if (k==n)
cout<<"нет студента родившегося в данном месяце";}}}
```

# Ссылка

- Ссылка - способ определения альтернативного имени переменной. Ссылка не является са мостоятельным типом. Она определяется в операторах объявления при помощи символа «&» и существует только после обязательной инициализации.

- **Например:**
- int a=29;
- //**alta**- ссылка, определяющая альтернативное имя переменной **a**, т.е. для **alta** и **a** выделена одна и та же ячейка памяти, в которой хранится значение 29
- int &alta=a;

# Ссылка

- Ссылка внутри программной единицы может быть, инициализирована только один раз. Время существования ссылки определяется временем жизни этой программной единицы, т.е. от момента вызова функции (активизации) до момента передачи управления в вызывающую среду. Последнее, предоставляет функциям удобный механизм изменения значения передаваемых им аргументов и повышает эффективность при передаче составных типов данных, поскольку не требует их копирования в стек. Кроме этого вызов функции приобретает вид общепринятого, для большинства языков программирования.

- **Например:**
- // пример функции с использованием ссылок
- void swap ( int &a,int &b)
- { int t=a;
- a=b;
- b=t;
- }

# Ссылка

- Вызов функции с использованием ссылок не требует передачи адресов и имеет вид swap(x,y);
- // пример функции с использованием указателей
- // вызов функции имеет вид- swap(&x,&y);
- void swap (int *a, int *b)
- { int t = *a;
- *a= *b;
- *b=t;
- }
- Ссылка может быть, определена и для указателей.
- Например:
- int a,*b=&a; // b - указатель на переменную a
- int *&altb = b; // alt - ссылка на указатель b
- Вышеприведенный пример можно интерпретировать следующим образом: altb есть альтернативное имя указателя «b» где хранится адрес переменной «a».
- При использовании указателей необходимо знать следующие ограничения:
- не допускаются операции над ссылками (допустимы операции над объектами ссылок);
- недопустимы ссылки на ссылки и битовые поля структур.

# Массивы

- Тесную связь с указателями имеет предопределенный в языке C++ тип данных - массив.
- Массивы - это структурированный тип данных, представляющие собой непрерывные блоки памяти, содержащие множество элементов одного и того же типа. Признаком вектора при описании является наличие парных скобок –“[ ]”.
- Массив описывается путем указания типа элементов, имени и квадратных скобок. Положи тельная константа или константного выражение, внутри квадратных скобок, задает число элементов массива. Элементы массива нумеруются с 0.
- **Например:**
- long arr1[32];
- char arr2[79];,
-
- где объявлены массивы arr1, содержащий 32 элемента типа long и arr2, содержащий 79 элементов типа char.
- При объявлении массивов arr1[32] и arr2[79], определяется не только выделение памяти для 32 и 79 элементов массивов, но и для указателей с именами arr1 и arr2, значение которых равно адресам первых по счету (нулевых) элементов массива. Сами массивы остается безымянными, а доступ к элементам массива осуществляется через указатели с именем arr1 и arr2. С точки зрения синтаксиса языка указатели являются константами, значения которых можно использовать в выражениях, но изменить эти значения нельзя.

# Массивы

- В C++ разрешены два способа доступа к элементам массива: классический - с помощью индекса; с использованием механизма указателей.
- Доступ к элементам массива путем указания имени и индексного выражения в квадратных скобках не требует особого пояснения, поскольку широко используется практически во всех языках программирования. При таком способе доступа записываются два выражения, причем второе выражение заключается в квадратные скобки. Одно из этих выражений должно быть указателем, а второе - выражением целого типа. Синтаксис языка C++ допускает изменения последовательности записи этих выражений, но в квадратных скобках записывается выражение следующее вторым. Например,  запись arr1[2] или 2[arr1] будут эквивалентными и обозначают элемент массива с номером два.
- Применение механизма указателей основано на использовании факта, что имя массива является указателем - константой, равной адресу начала массива - первого байта первого элемента массива (arr1==&arr1[0]). В результате этого, используя операцию разыменования «*» можно обеспечить доступ к любому элементу массива. Так, эквивалентны, будут обращения к i-му элементу массива с использованием индекса - arr1[i] и ссылки *(arr1+i), поскольку (arr1+i)==&arr1[i].

# Векторы

- Для подтверждения сказанного выше сделайте анализ результатов приведенной ниже программы.
- const int n=5;
- #include<iostream.h>
- main(){
- int i,j,A[5],c,*v=A;
- for (i=0; i<n; i++)
- { *(A+i)=i;}
- cout<<"&A="<<&A<<'\n';
- cout<<"&A[0]="<<&A[0]<<'\n';
- cout<<"A="<<A<<'\n';
- for (j=0; j<n; j++)
- cout<<*v++;
- cout<<'\n';
- for (j=0; j<n; j++)
- cout<<*(v+j);
- cin>>c;
- return 0;
- }
- На экране дисплея мы увидим:
- &A=0x0064fdd0
- &A[0]=0x0064fdd0
- A=0x0064fdd0
- 12345
- 12345

# Массивы

- Следует обратить внимание, что выражение *(arr1+1) не эквивалентно ошибочному выражению *(arr1++), поскольку arr1 есть указатель константа. При использовании операции инкремента (декремента) необходимо определить промежуточную переменную - указатель.
- **Например,** следующее определение переменных
- int arr[20];
- int*ukar=&arr[0];
- позволяет, в операторах цикла, для доступа к элементам использовать выражение *ukar++.
- В частности после выполнения присваивания ukar=arr доступ ко второму элементу массива можно получить с помощью указателя ukar в форме ukar[2] или 2[ukar].

# Массивы

- При описании массива может быть, выполнена инициализация его элементов.  В C++ опреде лены два правила инициализации: по умолчанию; явная инициализация.
- Правило инициализации «по умолчанию» определено для статических и внешних перемен ных : векторов, многомерных массивов, строк, указателей, структур и объединений. По умолчанию все элементы этих переменных инициализируются нулями.
- Явная инициализация массива производится путем задания набора начальных  значений элементов, разделенных запятыми и заключенных в фигурные скобки, по следующей форме:
- int  b[2] ={1,2}; // b[0]=1, b[1]=2

# Массивы

- В языке C++ допускается несовпадение значения индекса массива в операторе объявления и числа начальных значений элементов. Когда число начальных значений меньше значения индекса то инициализируются только первые элементы массива, а остальные либо обнуляются, либо не определены.
- **Например:**
- int b[2]={1}; //b[0]=1, b[1]=0
- Если число начальных значений больше заданного значения элементов, то будет сгенерировано сообщение об ошибке.
- int b[2]={1,2,3,4}; // ошибка
- При инициализации допустимо, также, не указывать значение индекса. В этом случае, компилятор определяет число элементов массива исходя из количества начальных значений.
- int b[ ]={1,2}; // эквивалентно заданию b[2]
- Описание массива без указания значения индекса и списка инициализации допустимо только в том случае, когда компилятор не производит резервирование памяти. А именно, когда массив описывается с атрибутом extern, указывающим, что информация о числе элементов находится в месте первоначального объявления, например:
- extern int b[ ]; ,
- или если описание производится в поле объявления формальных параметров функции, например:
- int streg (char S1[ ],charS2[ ])
- {тело функции}

# Массивы

- При работе с массивами следует учитывать, что в языке С не выполняется контроль допус тимости значения  индекса. В связи с чем, при работе с массивами, рекомендуется использо вать операцию  sizeof  которая, применительно к ним возвращает число байт, выделяемых компилятором для массива.
- Интерактивный ввод данных массива выполняется с помощью операторов цикла. Например, для ввода значений массива может быть предложена следующая последовательность операторов:
- for (int i=0; i<n;i++){
- cout<<"введите a[ "<<i<<" ] эл-т"<<'\n';
- cin>>a[i]; }
- Примером задачи с использованием массивов может служить задача слияния двух массивов упорядоченных по возрастанию. Алгоритм решения этой задачи состоит в следующем: вводится рабочий массив размерность которого равна сумме сливаемых; производится поэлементное сравнение в результате, которого в рабочий массив записывается элемент, удовлетворяющий заданному  условию, после чего он изымается из рассмотрения; программа заканчивает свою работу после перезаписи всех элементов в рабочий вектор.
- Внимание! При написании программы необходимо предусмотреть анализ значений индексов, а также заблокировать после слияния последний элемент одного из массивов.

# Массивы

```
//программа слияния двух массивов упорядоченных по возрастанию
#include<iostream.h>
const int n=3;
const int m=3;
const maxint=32767;
void main(){
int c[n+m],a[n],b[m];
int j=0;
for (int i=0; i<n;i++){
cout<<”введите  “<<i<<” эл-т”<<’\n’;
cin>>a[i]; }
for (int k=0; k<m;k++){
cout<<”введите  “<<k<<” эл-т”<<’\n’;
cin>>b[k];}
for (int j=0,i=0,k=0; j<n+m;j++){
if (a[i]<b[k])
{
c[j]=a[i];if(i<n-1)i++;//не позволяет выйти за пределы массива else
a[i]=maxint;//обеспечивает корректную работу if (a[i]<b[k]) //при отсутствии этого оператора для последнего i- a[i] в //операторе сравнения не изменится
}
else
{c[j]=b[k];
if(k<m-1)k++;//не позволяет выйти за пределы массива else
b[k]=maxint;//обеспечивает корректную работу if (a[i]<b[k]) //при отсутствии этого оператора для послееднего k- b[k] в // операторе сравнения не изменится
   };
 }
for (int j=0; j<n+m;j++)
cout<<*(c+j)<< "   "
cin>>j;
 }
```

# Массивы

- В отличии от других языков высокого уровня в C++  векторы имеют только одну размерность . Многомерные массивы в C++  представляются в виде массивов указателей на многомерные массивы. Многомерные массивы описываются следующим образом
- int V[2][2]; // 2 массивы из 2 int каждый
- float BV [4][4][4]; // 4 массивов указателей на 4 векторов из 4 float
-
- Синтаксис языка C++  не накладывает ограничений на размерность массивов. Однако реально используются размерности не выше трех.
- Обращение к элементам многомерных массивами может производиться как классическим способом - путем указания значений индексов, например,
- **V[1][2]=3;,**
- так и с использованием механизма указателей
- *(V[1]+2)=3;  //  V[1][2]=3.
- Использование механизма указателей для доступа к некоторому i,j,k-ому элементу трехмерного массива удобно рассмотреть на примере следующих эквивалентных обращений
- BV[i][j][k]=*(BV[i][j]+k)=*(*(BV[i]+j)+k)=*(*(*(BV+i)+j)+k)

# Массивы

- В этих обращениях использован тот факт, что, имя многомерного массива является указателем-константой на массив указателя - констант массива строки, первый элемент которого есть тоже указатель - константа строки. Элементы многомерных массивов хранятся в памяти в порядке возрастания самого правого индекса - по строкам.
- Аналогично одномерным массивам многомерные массивы могут быть так же инициализированы при их описании, например:
- float  BV[3][3][3]={1.1,1.2,1.3,
- 2.1,2.2,2.3,
- 3.1,3.2,3.3}
-
- В этом  случае, набор начальных значений, задаваемый при описании массива, соответствует порядку  размещения элементов в памяти.
- При инициализации массивов одна из размерностей, а именно левая может не указываться. В этом случае число элементов массива определяется по числу членов в наборе инициализации.
- int V[ ][3]={1,2,3,
- 4,5,6,
- 7,8,9}; // V[3][3]

# Массивы

- Эта же инициализация может быть выполнена следующим образом :
- 
- static int b[3][3] = { {1,2,3 }, { 4,5,6 },{7,8,9} };
- 
- C++ допускает инициализацию не всех, а только первых элементов строк многомерного массива. Оставшиеся элементы инициализируются 0. В этом случае элементы строк заключаются в фигурные скобки, например:
- int  V[ ][3]={{1,2},
- {4,5,6},{7}};
- 
- элементы первой строки получат значения 1,2 и 0,  второй  4,5,6, а третей 7,0,0.

# Массивы

- Пример программы, вычисляющей сумму элементов главной диагонали матрицы
- #include<stdio.h>
- void main( )
- {int V[ ][3]={1,2,3,
- 4,5,6,
- 7,8,9}
- int i, s, s2;s1=0;s2=0;
- for (i=0;i<3;i++)
- s1+=V[i][i];
- for(i=0;i<3;i++)
- s2+=*(*(V+i)+i);
- printf("s1=%d  s2=%d " ,s1,s2);
- }

# Массивы

- Примеры решения задач с использованием матриц.
- //Вычислить суммы элементов каждой строки матрицы //arr1[n,n]. Найти минимальную из этих сумм и номер соответствующей строки

```
#include<iostream.h>
#include<math.h>
#define n 2
void main(void){
int arr1[n][n],arr2[n];
int sum=0;
cout<<"введите данные  arr[" <<n<<","<<n<<"]\n";
for(int i=0;i<=n;i++){
for (int j=0;j<=n;j++){
cout<<"введите arr["<<i<<","<<j<<"] элемент";
cin>>arr1[i][j]; }}
for (int i=0;i<=n;i++)
{sum=0;
for (int j=0;j<=n;j++)
{
sum+=arr1[i][j];}
arr2[i]=sum;}
int min=arr2[0],ind=0;
for (int j=0;j<=n;j++){
if(min>arr2[j]){min=arr2[j],ind=j;}}
cout<<"min сумма ="<<sum<<" соответсвуящая строка "<<ind ;
cin>>ind;
  }
```

# Массивы

- **Краткие выводы из содержания лекции :**
- 
- 1) для доступа к элементу многомерного массива m в C++ нужно писать m[i][j], а не m[i,j].
- 2) если указатель p указывает на элемент массива, то p+1 - указывает на следующий элемент и т.д вне зависимости от типа элементов массива.
- 3) имя массива означает то же, что и адрес первого элемента массива a == &a[0]
- 4) доступ к массиву через индекс и через указатель со смещением аналогичны : a[i] == *(a+i)
- 5) указатель тоже можно использовать в выражениях с индексом. Если pa указывает на элемент массива, то
- *(pa+i) == pa[i]
- 6) pa++ продвигает указатель на следующий элемент массива. С именем массива так делать нельзя