Программирование на языке C++

Зариковская Наталья Вячеславовна

Лекция 6

Операции и выражения в С и С++

- Выражения состоят из операций и операндов. В языке С в качестве операндов могут использоваться: константы, переменные, указатели функций и выражения. Использование выражений в качестве операндов является отличительной особенностью языка. Порядок выполнения операций в выражениях соответствует их приоритету. Выражения, не содержащие скобок, которые явно задают порядок вычислений, вычисляются слева направо согласно их приоритета за исключением операций: унарных, присваивания и условия [Таблица].
- В выражениях на языке С допускаются использование операндов различных типов (смешивание). В процессе вычисления выражения операнды различных типов приводятся к единому по следующему правилу: любое char , short или enum преобразуются в int или unsigned; если после первого шага противоречия в различие типов не разрешены, то происходит дополнительное преобразование типов операндов, согласно иерархии типов
- **int<unsigned<long< unsigned long<float<double<long double**
- Следует заметить, что само представление операндов в памяти не изменяется, а создаются промежуточные переменные, в которых во время вычисления выражения (временно) хранятся преобразованные значения.
- Кроме автоматического преобразования типов в языке С++ допустимо и явное преобразование. Оператор явного преобразования типа имеет вид:
- **(type) идентификатор** или **(type) выражение,**
- где (type) - идентификатор предопределённого типа. Допустимо использование этого оператора и в функциональном стиле:
- **type(идентификатор) или type(выражение).**

Операции и выражения в TC и C++

приоритет	операции выполняемые слева направо	операции, выполняемые справа налево
1	++ -- постфиксные	! ~ + - * & (type) ++ -- префиксные sizeof
2	( ) [ ] . --> ::	
3	* / % арифметические мультипликативные	
4	+ - арифметические аддитивные	
5	<< >>сдвиги (выдвигаемые биты теряются)	
6	< <= > >= отношения 1	
7	== != отношения 2	
8	& побитовая конъюнкция	
9	^ побитовая сумма по mod2	
10	| побитовая дизъюнкция	
11	&& логическая конъюнкция (и)	
12	|| логическая дизъюнкция (или)	
13		? : условие
14		= += -= *= /= %= &= ^= != <<= >>=
15	, запятая	

Операции и выражения в ТС и С++

- Оператор явного преобразования типа является одноместным оператором и как видно из таблицы имеет тот же приоритет и ассоциативность, что и другие одноместные операторы.
- В результате действия этого оператора в выражениях само значение идентификатора не изменяется, а лишь временно создаётся промежуточная переменная, значение которой соответствует значению идентификатора или выражения типа type.
- Примеры использования оператора преобразования типа могут быть такими:
- **y=j+(float)i;**
- **y=long(z=i+d); .**
- По числу операндов участвующих в операции, различают две группы операторов: унарные и бинарные.
- По типу выполняемой операции различают: арифметические, логические, поразрядные

  логические, присваивания и др.

Унарные операции: инкремента и декремента

- Эти операции могут иметь две формы записи: префиксная, когда операнд располагается после знака операции (++a, --a); постфиксная, когда операнд располагается слева от знака операции (a++ ,a--).
- В префиксной форме сначала выполняется увеличение (++a) или уменьшение (--a) операнда на 1 и увеличенное (уменьшенное) значение используется в выражении, например:

- **int a=0,b;**
- **b=++a; //эквивалентно a=a+1 (1) b=a(1)**
- **b=--a; //эквивалентно a=a-1 (-1) b=a(-1)**
-
- В постфиксной форме сначала берется значение операнда, которое используется в выражении и только после этого его значение увеличивается (a++) или уменьшается (a--) на 1, например:

- **b=a++; // эквивалентно b=a(0);a=a+1 (1)**

- **b=a--; //_____ b=a(0);a=a-1 (-1)**

Унарные операции: инкремента и декремента

- Сказанное выше может быть проиллюстрировано приведенной ниже **программой:**
- #include <iostream.h>
- void main()
- {int a=1, b=1, y, z;
- y=(a++)*5;
- cout<<" a=1 y=(a++)*5="<<y<<" a="<< a<<'\n';
- z=(++b)*5;
- cout<<" b=1 z=(++b)*5="<<z<<" b="<<b <<'\n';
- cin>>y;
- }
- В результате получим
- a=1 y=(a++)*5=5 a=2
- b=1 z=(++b)*5=10 b=2
- В случае, если операции инкремента и декремента используются как самостоятельные операторы, префиксная (++a;) и постфиксная (a++;) формы записи становятся эквивалентными.

Операция “ ~ “ - поразрядная инверсия (дополнение)

- Поразрядное дополнение (или поразрядный - не) оператор производит поразрядную инверсию операнда. Операнд должен быть целого типа. Работа этого оператора поясняется следующим примером:

```
#include <iostream.h>
void main()
{
cout<<"sizeof(unsigned int )="<<sizeof(unsigned int )<<'\n';
unsigned int  a1=0xfffffff0;
~a1;  //изменение в памяти не происходит
cout<<"a1="<<hex<<a1<<" ~a1="<<~a1<<'\n';
unsigned int  a2=0xfff1,a11;
cout<<"a2="<<a2<<" ~a2="<<~a2<<'\n';
unsigned int a3=0xfff2;
cout<<"a3="<<a3<<" ~a3="<<~a3<<'\n';
unsigned int  a4=0xfff3;
cout<<"a4="<<a4<<" ~a4="<<~a4<<'\n';
 a11=3+~a4;
cout<<"a11=3+~a4 равно "<<a11<<'\n';
cin>>a1;
}
```

Операция “ ~ “ - поразрядная инверсия (дополнение)

- В результате получим

- sizeof(unsigned int)=4
- a1=fffffff0 ~a1=f
- a2=fff1 ~a2= ffff000e
- a3=fff2 ~a3= ffff000d
- a4=fff3 ~a4= ffff000c

- a11=3+~a4 равно ffff000f

Операция sizeof

- Результат операции sizeof является размер в байтах типа или объявленной переменной. Применение операции sizeof к массивам возвращает число байтов, не обходимое для размещения всех элементов массива.
- Например:
- int r;
- r=sizeof (char); // r=1
- r=sizeof(long double); //r=10
- int array[8]={1,2};
- int col;
- col=sizeof(array)/sizeof(int);
- Применение операции sizeof к имени типа или к идентификатору, имеющему тип структуры или объединения приводит, к выдаче фактического размера, который может включать участки памяти, используемые для выравнивания элементов структуры или объединения.
-

Арифметические операторы

- К арифметическим операциям относятся операции: “ **+** “ плюс, ” **-** “ минус, “***”** умножение, “/” деление, “%” определение остатка (операнды только целого типа), “++” инкремента, “--“ декремента. Операторы “+”, “-“, “*”, “/” можно отнести к классическим и не имеющим отличий от их обычного применения за исключением описанного выше преобразования типа. Но использование операции деления ‘’/’’ имеет некоторые особенности. Если делимое и делитель целые, то при наличии остатка последний отбрасывается, например:
- 7/5 //результат деления 1 остаток 0.4 отброшен.
- 8/4 //----------2 остатка нет.
- 3/5 //----------0.
- 7.0 //1.4 промежуточная переменная вещественного типа.
-
- Операция ‘’%’’ – определения остатка от деления первого операнда на второй применяется только для целых операндов. Если второй операнд равен нулю, то выдается сообщение.
- int a=7,b=5,c=0,d;
- d=a%b; //d=2 - остаток от целого деления 7/5.
- d=b%a; //d=5 - остаток от целого деления 5/7.
- d=b%c; //сообщение об ошибке.
-

Арифметические операторы

- В некоторых случаях может быть полезным использование знака результата остатка от деления. Знак результата зависит от конкретной реализации. В данной реализации знак результата совпадает со знаком делимого и не зависит от знака делителя. Например, в результате выполнения простейшей программы с изменяемыми знаками операндов
- #include <iostream.h>
- void main()
- {int a=99, b = 10, apolbpol, apolbmin, aminbpol, aminbmin, z;
- apolbpol = a % b;
- apolbmin = a % (-b);
- aminbpol = (-a) % b;
- aminbmin = (-a) % (-b);
- cout<<"apolbpol="<<apolbpol<<"apolbmin="<<apolbmin <<'\n';
- cout<<"aminbpol="<<aminbpol<<" aminbmin="<<aminbmin<<'\n';
- cin>>z;
- }
- будут такими
- apolbpol==9 apolbmin =9
- aminbpol =-9 aminbmin =-9

Операции сдвига

- Поразрядные логические операции сдвига “>>” и “ <<”. Операции сдвига применяются к данным целого типа (char, short, int, long). Результат операции также будет целым и соответствует двоичному представлению числа, смещённому, влево или право, на заданное число бит. При этом выдвигаемые,за пределы типа, биты теряются.
- Эта операция может быть представлена в следующем виде
- A<<b или A>>b,
- где A – данное целого типа, b – константа или константное выражение целого типа соответствующее числу сдвигаемых бит. Примеры использования операции сдвига могут быть следующими:
- а) формирование двойного слова типа long unsigned по известному старшему и младшему словам:
- unsigned high_byte =0xff00 ,low_byte=0x00ff;
- long unsigned ent;
- ent = ((long)high_bite<<16)!(long)low_bite;
- video_word=((usigned)attribut<<8)!(unsigned)sym

Операции сдвига

- б) сдвиг влево (вправо) целого числа в двоичном представлении соответствует умножению (делению) соответствующего числа на два.
- p2_div=p2>>1;//деление на 2
- p2_div=p2>>2;//деление на 4
- .....
- p2_p2=p2<<1;//умножение на 2
- p2_p2=p2<<2;//умножение на 4
- в) char sym=’A’, attribut=’\0x07’;
- unsigned video_word;

- video_word=((usigned)attribut<<8)!(unsigned)sym

Логические операции сравнения в C++

- До версии C++ 4.5 язык не поддерживал специально определенного логического типа. Но язык C++ поддерживал операции логического типа.
- В этом случае скалярные типы, такие как целые, интерпретируются как булевские: ноль представляет значение false и любое ненулевое значение как true.
- Операторы сравнения возвращают false или true в форме численного значения 0 или 1. Результатом логических операций также являются значения 0 или 1.
- C++ поддерживает 2 группы операторов сравнения. Первая - меньше «<», меньше или равно «<=», больше или равно «>=», больше «>», которые имеют одинаковый приоритет и операции равенства - равно «==» и не равно « != », которые имеют более низкий приоритет.
- Операторы сравнения находят применения в различных операторах управления, например:

- **while ((sum += i)<max) {.....}**

Логические операции в С++

- В С++ имеются следующие логические операции: «&&» -И; «||» -ИЛИ и унарная операция «!»-не.

- **Логическая операция -&& (И)** имеет вид:
- **выражение && выражение,**
- где выражение - выражение произвольного типа, результат вычисления которого должен иметь один из основных типов или быть указателем. Эта операция возвращает 1(истина), если оба операнда не нулевые, и 0 (ложь) в противном случае. Операция && гарантирует вычисление слева направо, при этом, второй операнд не вычисляется, если первый операнд есть 0. Результат всегда имеет тип int.

- **Логическая операция - || (ИЛИ)** имеет вид:
- **выражение || выражение**
- Эта операция возвращает 1, если хотя бы один из ее операндов ненулевой, и 0 в противном случае. Операция || гарантирует вычисление слева направо, при этом, второй операнд не вычисляется, если первый операнд не есть 0.

Логические операции в C++

- **Логическая операция - ! (НЕ)** имеет вид:
- **!выражение**
- Операция логического отрицания «!» вырабатывает значение «истина» -0, если операнд имеет не нулевое значение, и значение 1, если операнд равен нулю (0). Результат имеет тип int. Операнд должен быть целого, вещественного типа или типа указатель.
- Следует заметить, что поскольку любое ненулевое значение в языке C++ трактуется как true, то двойное использование операции отрицания приводит к инвертированию не истинного значения аргумента, а его логического эквивалента.
- Пример:
- !!4 //дает 1.
- Логические операции применяются для записи предикатов и используются в управляющих операторах.

Поразрядные логические операции

- C++ поддерживает следующие поразрядные логические операции: “&”- поразрядное логическое И (AND); “ ^ ” - поразрядное сложение по mod 2 (XOR - исключающее или) ;
- “|» -поразрядное логическое или (OR- дизъюнкция).

- Примеры.
- **Поразрядное логическое И - «&»** используется для проверки наличия 1 в каком-либо разряде. Таблица истинности для операции «&» имеет вид
- 1&1->1
- 1&0->0
- 0&1->0
- 0&0->0
-
- A&1 -> Проверить наличие 1 в 0 разряде.
- Операция применима только к целым операндам.

Поразрядные логические операции

- **Поразрядное сложение по mod2** используется для селективного сброса 1 в каком - либо разряде. Таблица истинности для операции «^» имеет вид
- 1^1 -> 0
- 0^0 -> 0
- 1^0 -> 1
- 0^1 -> 1
-
- A^64 -> Убрать 1 в 6 разряде.
- Операция применима только к целым операндам.
- **Поразрядное логическое или «|»** используется для установки 1 в каком - либо разряде. Таблица истинности для операции «|» имеет вид
- 1|1 ->1
- 0|1 ->1
- 1|0 ->1
- 0|0 ->0
-
- A|64 - Установить 1 в 6 разряде.
- Операция применима только к целым операндам.

Поразрядные логические операции

- **Операция условия (?:) тернарный оператор**
- Операция условия (?:) в общем виде имеет следующее представление:
- **Лог_выражение ? оператор_TRUE:оператор_FALSE;,**
- где «**?**» и «**:**» - служебные символы оператора условия.
- При выполнении «оператора условия» сначала вычисляется логическое выражение-Лог_выражение. Если оно-истина(TRUE), результатом операции условия является результат выражения оператор_TRUE, если ложь(FALSE)- результат выражения оператор_FALSE. Эта операция нашла широкое применение при написании программ на языке С. Так в заголовочном файле stdlib.h в директивах препроцессора описаны макро функции выбора минимального и максимального числа из двух заданных
- **#define max(a,b) (((a)>(b))?(a):(b))**
- **#define min(a,b) (((a)<(b))?(a):(b)).**
- Оператор условия используется и при написании программ. Например
- **int a=4,b=3,c;**
- **c=a>b?a*a+b*b:0;//c=a*a+b*b при а больше b.**
- Из этого примера видно, что эквивалентная запись с использованием оператора условия ко роче.
-

Операторы присваивания

- В простейшем случае оператор присваивания задается символом “=” и имеет вид a=b;. Дей ствие этого оператора означает, что значение выражения справа от знака равенства посыла ется в ячейку памяти, выделенную компилятором под переменную a. Если тип значения вы ражения (b) не совпадает с типом переменной (a), то происходит автоматическое преобразо вание типа выражения b к типу переменной. При этом C++ допускает преобразование расши рения (так int может быть расширено до double), так и сужения (double может быть назначен int или char).
- Использование автоматического преобразования типов в операторах присваивания может повлиять на правильность программы, что требует особого внимания со стороны программи стов.
- В связи с тем, что в C++ оператор присваивания интерпретируется как обычный оператор, то допускается многократное присваивание в одном выражении. Например:
- a=b=c=d=0;//эквивалентно d=0; c=d; b=c; a=b;
- При этом операторы присваивания выполняются справа налево. Язык C++ допускает также комбинированное использование операторов присваивания. Например:
- a=b+(c=d+2); //эквивалентно c=d+2; a=b+c;

Операторы присваивания

- Заметим, что ввиду существующей на данный момент зависимости порядка вычислений от версий компиляторов (правоассоциативный или лево ассоциативный) следует избегать в выражениях сложных комбинаций операторов присваивания. Так, выражения
- n=(a=x++)*(b=x++);//при левоассоциативном выполнении
- //операций и x=a*b(1*2)(2); побочный эффект
- //при правоассоциативном выполнении операций
- //получим a=2 и b=1
- В Borland C++ принята левоассоциативная последовательность выполнения операций. Помимо простого оператора присваивания C++ поддерживает следующие виды оператора присваивания:
- a+=b; //эквивалентно a=a+b
- a-=b; // эквивалентно a=a-b
- a*=b; // эквивалентно a=a*b
- a/=b; // эквивалентно a=a/b
- a%=b; // эквивалентно a=a%b
- a&=b; // эквивалентно a=a&b побитовая конъюнкция
- a^=b; // эквивалентно a=a^b побитовая сумма по mod 2
- a|=b; // эквивалентно a=a|b побитовая дизъюнкция
- a<<=b; // эквивалентно a=a<<b сдвиг влево на b разрядов
- a>>=b; // эквивалентно a=a>>b сдвиг вправо на b разрядов

Операторы присваивания

- В выражениях на языке C++ может быть использовано значительное число функций. Однако, в отличие от большинства языков высокого уровня (Fortran, Pascal), использование функций в выражениях (не только математических) требует подключения заголовочных файлов, в которых описаны прототипы используемых функций. Например, использование стандартных математических функций требует подключения заголовочного файла <math.h>.
- Эти функции представлены **в таблице на следующем слайде**
- определён также и ряд констант pi,log2=1.44...,log10==0.43... и т.д.
- #difine cabs(z) Chypot((z).x,(z).y)
- описан
- typedef enum
- {domain=1,//ошибка диапазона задания аргумента—log(-1)
- sing, //точка сингулярности (точка в которой вычисляемая функция=бесконечности
- //или =-бесконечности pow(0,-2)
- overflow, //переполнение—exp(1000)
- underflow, //потеря значимости (отрицательное переполнение)--exp(-1000)
- tloos, //полная потеря значимости—sin(10e70)
- ploos // частичная потеря значимости - не используется
- }

Встроенные математические функции

прототип функции	имя	содержание
double acos(double _x);	acos(x)	арккосинус
double asin(double _x);	asin(x)	арксинус
double atan(double _x);	atan(x)	арктангенс
double atan2(double_y, double_x);	atan2(y,x)	арккотангенс от y/x
double ceil(double _x);	ceil(x)	округление в большую сторону
double cos(double _x);	cos(x)	косинус, x- в радианах
double cosh(double _x);	cosh(x)	косинус гиперболический
double exp(double _x);	exp(x)	e в степени x(e^x)
double fabs(double _x);	fabs(x)	абсолютное значение |x| типа double
double floor(double _x);	floor(x)	возвращает ближайшее целое, не большее x
double fmod(double _x, double_y);	fmod(x)	остаток от деления x на y
double frexp(double_x, int*_exponent);	frexp(x,&p)	возвращает нормализованную мантиссу и не сдвинутый порядок p
Double ldexp(double_x, double*_ipart);	ldexp(x,n)	возвращает число по заданным мантиссе и не сдвинутому порядку
double log(double _x);	log(x)	натуральный логарифм
double log10(double x);	log10(x)	десятичный логарифм
double modf(double _x, double);	modf(x,&p)	выделяет целую и дробную части числа
double pow(double _x, double _y);	pow(x,y)	x^y
double sin(double _x);	sin(x)	синус x, в радианах
double sinh(double _x);	sinh(x)	синус гиперболический
double sqrt(double _x);	sqrt(x)	корень из x, x>0
double tan(double _x);	tan(x)	тангенс x, x в радианах
double tanh(double _x);	tanh(x)	тангенс гиперболический
int abs(int _x);	abs(x)	модуль x типа int
double atof(const char*_s);	atof(s)	преобразует строку символов в число с плавающей запятой
double hypot(double _x, double_y);	hypot(x,y)	корень из(x^2+y^2)
long labs(long _x);	labs(x)	абсолютная величина типа long |x|
int matherr(struct exception*_e);		внутренняя функция, выводящая на stderr сообщение об ошибке в виде заглушки
Double poly(double_x, int _degree, double coeffs []);	poly(x,n,a)	вычисляет для заданного аргумента x, показателя степени degree на начало ко торого указывает coeffs [] poly=a[n-1]*x^n +a[n-2]x^n1+ ...+a[1]*x+a[o]
double pow10(int _p)	pow10(p)	возвращает 10^p

Операция " , "

- Операция " , "- операция последовательного вычисления для двух или более выражений там, где по синтаксису допустимо только одно выражение. Вычисления производятся слева направо. При выполнении операции последовательного вычисления преобразование типов не производится. Пример:
- for (i=0,j=0; i < max && j < max ; i++)
- {.....}
-
- Запятая может также использоваться как разделитель. Например:
- ИМЯ функции (пар1 , пар2 , пар3).
-

Краткие выводы из содержания лекции :

- *1) В С++ нет булевского типа; ложью считается ноль, а истиной не ноль.*
- *операции отношения и логические операции оперируют с любыми значениями*
- *приводимыми к целому и возвращают ноль или единицу.*
- *2) присваивание является операцией, а не оператором и может использовать-*
- *ся в выражениях.*
- *3) в С++ есть составное присваивание*
- *4) увеличение (уменьшение) на единицу короче записывается с помощью*
- *инкремента и декремента. Следует различать их префиксную и постфиксную*
- *форму записи.*
- *5) несколько операций можно перечислять через запятую.*
-

Операторы. Составной оператор

- Операторы программы в языке Си могут быть простыми и составными. Простой оператор - это оператор, не содержащий другие операторы. Разделителем операторов служит точка с запятой - «;».Специальным случаем простого оператора является пустой оператор, состоящий из единственного символа «;». Другими словами - это просто точка с запятой. Этот оператор используется, когда после метки или операторов цикла не должно следовать никакого оператора.
- Например:
- **label:; //метка с пустым оператором**
- **while (лог. выражение); //пока лог. выражение true дальнейшее выполнение программы**
- **//не производится**
- Другим специальным случаем простого оператора является оператор метки. Оператор метки - оператор выражения (возможно пустой), ограниченный точкой с запятой, перед которым стоит идентификатор, заканчивающейся символом «:».
-

Операторы. Составной оператор

- Например:
- **Point_out:return result; ,**
- **где: Point_out - метка.**
- Следует знать, что область действия идентификатора метки ограничена функцией, в которой он используется.
- Составной оператор представляет собой ноль или более операторов, заключенных в фигурные скобки. Использование этого оператора допустимо в любом месте программы, где допустим, простой оператор.
- Отличительной особенностью составного оператора языка C++ от других (например PASKAL) состоит в том, что он определяет новую область действия, т.е. переменные, определенные внутри составного оператора, являются локальными в нем и скрывают внешние переменные с теми же именами. Например:
- t=10.24;
- {int t=a; a=b; b=t;}//t-локальная переменная, отличная от предыдущей переменной t.
-

Операторы управления

- **Условные операторы (if, switch) и оператор breake**
- Операторы управления вычислительным процессом используются для организации: ветвления, циклического повторения операторов программы, а также передачи управления в необходимое место программы в зависимости от выполнения каких - либо условий.
- Оператор управления -if. Оператор ветвления if имеет следующий вид:
- **if(выражание) True_оператор;**
- **следующий оператор; ,**
- где «выражение»- выражение произвольного вида интерпретируемого языком С++ как логическое, т.е. любое отличное от нуля значение как True и False в противном случае. Работа этого оператора состоит в следующем: вычисляется выражение; если значение выражения отлично от нуля, то выполняется True-оператор ,а затем следующий за if оператор; если значение есть ноль ,то True-оператор пропускается и управление передается к следующему за if оператору.
- Например:
- int i=0, j=0;
- .....
- //i-счетчик отрицательных элементов массива m.
- if (m[j]<0)i++;

Операторы управления

- Рассмотренный выше оператор if можно рассматривать как частный случай оператора if...else, имеющего вид:
- **if(выражение)True-оператор;**
- **else**
- **False-оператор;**
- **следующий оператор;.**
- Работа этого оператора (if..else) состоит в следующем: если вычисленное выражение отлично от нуля то выполняется True_оператор, а затем следующий за if..else оператор; если вычисленное выражение равно нулю то выполняется False_оператор,а затем следующий за if..else оператор.
- Например:
- int i=0,j=0,k=0; //int i=j=k=0;
- .....
- if (m[j]<0)i++; //счётчик отрицательных элементов
- else
- k++; //счётчик положительных элементов
- .....

Операторы управления

- Оператор if..else могут быть вложенными. В этом случае операторы True_оператор или False_оператор могут быть представлены такой же конструкцией if..else.
- Например:
- int i=0,j=0,k=0,n=0;
- .....
- if(m[j]==0)n++ //счётчик нулевых элементов
- else
- if(m[j]<0)i++; // счётчик отрицательных элементов
- .....
- Для устранения неоднозначности компилятор С интерпретирует вложенные if таким образом, что ветвь else соотносит к ближайшему предыдущему if.

Операторы множественного ветвления switch

- Оператор switch имеет следующую форму:
- **switch(выражение) { // начало тела оператора switch**
- **case конст. целочисленное выражение 1:оператор 1;[break;]**
- **case ---------//----------//---------- 2: оператор 2;[break;]**
- **case ------//------------//---------- n: оператор n;[break;]**
- **[default: оператор n+1;] } // конец оператора switch**
- **следующий оператор; // в эту точку передается управление.**
- Оператор switch работает следующим образом:
- сначала вычисляется switch_выражение (switch_выражение - выражение целочисленного типа char, int, unsigned int, longint, long unsigned);
- вычисленное выражение сравнивается со значениями констант или константных выражений группы - константное целочисленное выражение 1...константное целочисленное выражение n;
- если какое-либо из константных целочисленных выражений совпало со значением switch_выражения, то выполняется соответствующий оператор и управление (при наличии оператора break) передаётся следующему за оператором switch оператору;
- если соответствующего совпадения не обнаружено, то выполняется оператор n+1 ветви default (если он есть) и управление передаётся следующему после switch оператору.

Операторы множественного ветвления switch

- Отметим, что оператор break относится к группе операторов безусловной передачи управле ния. При его исполнении управление передаётся в тело охватывающего блока (составного оператора, цикла) или для самого внешнего блока (цикла) на оператор, следующий после блока. Оператор break может быть использован и в других операторах цикла, рассматриваемых позднее.
- Следует знать, что при отсутствии операторов break и теле оператора switch и совпадении значения switch_выражения с каким-либо константным целочисленным выражением выполняется сквозной проход через всё оставшееся тело оператора switch, включая и оператор ветви default.
- Пример:
- switch(getchar()) {
- case ‘a’: puts(‘’введена буква а ‘’); break;
- case ‘б’: puts(‘’ введена буква б ‘’); break;
- case ‘c’: puts(‘’ введена буква с ‘’); break;
- default: puts(‘’ ни одна из трёх букв а, б, с не введена’’); break;
- }

Операторы множественного ветвления switch

- При вводе с клавиатуры одного из символов a, б или c на экран дисплея будет выведено соответствующее извещение. Например, при вводе символа ‘a’ на экране дисплея будет отображено следующее сообщение ***введена буква a.***
- После чего управление будет передано следующему за switch оператору. Если будет введена буква отличная от a,b или c на экран будет выдано сообщение: ***ни одна из трёх букв a, б, c не введена.***
- Затем управление также передается следующему за switch оператору.
- Приведенный выше пример без использования операторов break даст следующие результаты: при вводе символа ‘a’ на экране дисплея будет отображено:
- введена буква a
- введена буква b
- введена буква c
- ни одна из трех букв a,b или c не введена;
- При вводе символа ‘c’ на экране будет отображено: введена буква c ни одна из трех букв a,b или c не введена.
- Как видно из рассмотренного примера, отсутствие оператора break может привести к не корректным результатам, если это не предусмотрено специально.

Операторы цикла. Операторы while, do_while, for

- Язык C++ поддерживает три оператора цикла **while, do...while и for**.
- Оператор while в общем виде записывается:
- **while (выражение)**
- **оператор;**
- **следующий за циклом while оператор;**
- Оператор while обеспечивает реализацию цикла с предусловием. Это означает, что оператор в теле цикла вообще не вычисляется, если вычисленное выражение имело нулевое значение (ложь), а управление передается следующему за циклом while оператору. Если выражение отлично от нуля (истинно), тогда вычисляется оператор, и управление передается обратно к началу цикла. В результате тело цикла оператора while-оператор, выполняется до тех пор, пока выражение примет значение ноль (ложь), а управление будет передано следующему за циклом оператору. Пример:
- int i=1,p=1;
- while(i<=10){//Вычисление 10!
- p*=i;
- i++;
- }

Операторы цикла. Операторы while, do_while, for

```
//программа обеспечивает вывод первого отрицательного числа вектора
#include<iostream.h>
const int n=5;
void main()
{ int a[n];
for(int i=0; i<n; i++)
{cout<<" \n введите "<<i<<"элемент";
cin>>a[i];}
int i=0;
while (a[i]>0 && i<n)
i++;
if(i==n) cout<<"в последовательности нет отрицательных чисел";
else
cout<<"первое отрицательное число вектора"<<a[i]<<'\n';
}
```

Операторы цикла. Операторы while, do_while, for

- Оператор **do..while** в общем виде записывается:
- **do**
- **оператор;**
- **while(выражение);**
- Следующий за оператором **do** оператор;
- В отличие от предыдущего, оператор do реализует цикл с постусловием, что гарантирует выполнение тела, цикла хотя бы один раз, после чего производится вычисление выражения. Если значение выражения отлично от нуля (истинно), тогда управление передается обратно к началу оператора do и процесс вычисления повторяется. В том случае, когда значение «выражения» ноль (ложь) управление передается следующему за оператором do оператору. Пример:
- int n=1,p1,p2;
- do{
- p1=1./n; n++;
- p2=p1+1./n;
- }
- while(1./n<0.001);

Операторы цикла. Операторы while, do_while, for

- **Пример:**
- Программа обеспечивает взаимозамену первого встретившегося отрицательного (положительного) числа с первым встретивимся положительным (отрицательным) элементом вектора.
- #include<iostream.h>
- const int n=5;
- void main()
- {
- int a[n],prom;
- for(int i=0; i<n; i++) {cout<<" \n введите "<<i<<"элемент";cin>>a[i];} int i=0; if(a[i]>0)
- {
- while (a[i]>0 && i<n) i++; prom=a[0]; a[0]=a[i]; a[i]=prom;
- if(i= =n)cout<<"в поседовательности нет отрицательного числа" ; else
- for(int i=0; i<n; i++) {cout<<" \n "<<i<<"элемент "<<a[i];}} else
- { while (a[i]<0 && i<n) i++;
- prom=a[0]; a[0]=a[i]; a[i]=prom;
- if(i==n)cout<<"в последовательности нет положительного числа" ; else for(int i=0; i<n; i++)
- {
- cout<<" \n ""<<i<<"элемент "<<a[i];}} cin>>prom;
- }

Операторы цикла. Операторы while, do_while, for

- Оператор цикла **for** в общем случае записывается:
- **for(оператор1;выражение1;выражение2)**
- **оператор2;**
- **следующий за оператором for оператор;.**
-
- Оператор for работает следующим образом. Вначале выполняется оператор1 (обычно это оператор инициализации), который в частном случае может быть пустым. Затем вычисляется выражение1.Если значение выражения1 имеет значение ноль (ложь), то управление передается следующему за оператором for оператору, и оператору2,когда значение выражения1 имеет не нулевое значение. После этого выполняется выражение2 и управление передается на выражение1.Таким образом оператор2-тело цикла повторяется до тех пор пока выражение1 не примет значение 0.
- Примеры:
- //фрагмент программы определения минимального элемента вектора
- int min=a[0]
- for(int i=0,i<100,i++)
- if (min>a[i]) min=a[i];.

Операторы цикла. Операторы while, do_while, for

```
//поменять местами элементы главной и побочной диагонали заданной матрицы
#include<iostream.h>
const int n=3;
void main()
{int a[n][n],prom;
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
{cout<<"введите i"<<i<<" j элемент"<<j<<'\n';
cin>>a[i][j];}
//такая конструкция цикла позволяет произвести замену
// значений элементов главной и побочной диагонали
for(int i=0,j=n-1;i<n;i++,j--)
{    prom=a[i][i];    a[i][i]=a[i][j];   a[i][j]=prom;}
  for(int i=0;i<n;i++)
{for(int j=0;j<n;j++)
cout<<a[i][j]<<"  ";   cout<<'\n';}
   cin>>prom;}
```

Операторы цикла. Операторы while, do_while, for

- Следует отметить, что каждое или оба выражения могут быть опущены. И могут иметь следующий вид:
- **for(оператор1; ;выражение2)**
- **оператор2;**
- или
- **for(оператор1;выражение1;)**
- **оператор2;**
-
- Отсутствие выражения 1 делает предложение эквивалентным while(1), и приводит к необходимости обеспечивать выполнение условия выхода посредством действия оператора2.
- Отсутствие выражения 2 приводит к необходимости организации изменения операндов выражения 1 в теле цикла.
- Заметьте, что если оператор1 является описанием, то область видимости описанного имени распространяется до конца блока, охватывающего оператор for.
- Возможны и другие модификации параметров оператора for. Однако в этих случаях использование оператора for имеет ограниченный характер. Так, например, использование оператора for вида **for(;;);**
- приведёт к зацикливанию. А его использование потребует организации принудительго выхода из программы.

Оператор безусловный передачи управления- goto

- **Оператор безусловный передачи управления- goto** обеспечивает передачу управления оператору, помеченному меткой. Использование этого оператора при решении реальных задач не рекомендуется, так как он затрудняет понимание программ и возможность их модификаций.
- Оператор **goto** в общем виде записывается:
- goto имя-метки;
- ...
- имя-метки: оператор;
- Действие этого оператора состоит в том, что управление передаётся на оператор, помеченный меткой имя-метки.
- При использовании оператора goto необходимо знать следующие правила: метка должна находиться внутри текущей функции; оператор объявления с инициализацией нельзя помечать меткой; нельзя передавать управление в составной оператор, расположенный после оператора объявления с инициализацией; разрешается передавать управление за пределы составного оператора.

Оператор continue

- **Оператор continue** может использоваться только внутри тела цикла (for, do..while, while). В результате выполнения этого оператора управление передается: для оператора for на вычисление выражения 2 (оператор условия); для операторов while..do и while на вычисление выражения.
- Пример:
- for (int i=0; i<100; i++)
- { if (i%3) continue;
- ... /* решение какой-либо задачи, где i не делится на 3 */
- }

Краткие выводы из содержания лекции:

- *1) операторные скобки в C++ - фигурные скобки { }. После } точка с запятой не ставится, а перед ней ставится обязательно.*
- *2) если после if идет один оператор, то перед else будет точка с запятой. Скобки в условии if( ) обязательны.*
- *3) после каждой ветви case в операторе switch нужно ставить break, чтобы выполнялась только эта ветвь, иначе выполнятся и все последующие.*
- *4) цикл do...while имеет в конце условие, обратное к паскалевскому repeat...until*
- *5) в заголовке цикла for могут быть разнообразные выражения. Можно перечислять несколько выражений через запятую. Можно в теле цикла изменять переменную цикла. Можно делать цикл по нескольким переменным можно брать не целые переменные и т.д.*
- *6) массивы в C++ индексируются от 0. a[0] даст нам первый, а a[2] - третий элемент массива a.*
- *7) нельзя присвоить массив массиву.*