ЛЕКЦИя 2. Решения и циклы

Сравнение значений

Существует шесть фундаментальных операций для сравнения двух доступных значений:

< меньше чем

> больше чем

= = равно

<= меньше или равно

>= больше или равно

!= не равно

Каждая из этих операций сравнивает значения двух своих операндов и возвраща­ет одно из двух возможных значений типа bool:

*true* — если сравнение истинно,

*false* — если нет.

Можно увидеть, как это работает, рассмотрев несколько простых примеров сравнений.

Примеры.

Предположим, что объявлены целочисленные переменные i и j со значениями 10 и -5 соответственно.

Все представленные ниже выраже­ния возвращают значение *true*:

*i > j i ! = j j > -8 i<= j + 15*

Предположим, что мы определили следующие переменные :

*char first = 'A', last = 'Z';*

*first == 65 first < last ‘E’ <= first first != last*

Все выражения сравнивают значения кодов ASCII.

Первое выражение возвращает *true*, т.к. *first* инициализирована символом '*A*', что эквивалентно десятичному числу 65.

Второе выражение проверяет, меньше ли значение first, которое равно '*А*', чем значение *last*, которое равно '*Z*'. Если заглянуть в таблицу кодов символов ASCII , то мы увидим, что заглавные буквы представлены последовательными числовыми величинами - от 65 до 90, причем 65 представляет '*A*', а 90-'*Z*', поэтому второе сравнение также вернет *true*. Третье же выражение вернет *false*, потому что’*E*’ больше, чем значение *first*.

Последнее выражение вернет *true*, поскольку '*А*' определенно не равно ' *Z*'.

Теперь рассмотрим несколько более сложные сравнения чисел. Имея переменные, определенные следующим образом:

*int i = -10, j = 20;*

*double x = 1.5, у = -0.25E-10;*

взгляните на такие выражения:

*-1 < у*

*j < (10 - i)*

*2. 0 \* х >= (3 + у)*

Как видите, в качестве операндов сравнения можно использовать выражения, возвращающие числовые значения. Если вы заглянете в таблицу приоритетов, при­веденную в главе 2, то увидите, что скобки не являются совершенно необходимыми, однако они помогают сделать выражения яснее.

Первое сравнение истинно, поэтому возвращает *bool*-значение *true*. Переменная *у* содержит очень малое отрицательное

число, — 0,000000000025, а потому оно больше, чем -1.

Второе сравнение возвращает *false*. Выражение *10- i* равно 20, то есть тому же, что и *j*.

Третье выражение воз­вращает *true*, потому что *3 + у* чуть меньше, чем 3.

Вы можете использовать операции отношений для сравнения значений любого фундаментального типа, поэтому все, что вам нужно — какой-то практический способ использования результатов сравнения для модификации поведения программы.

Оператор if

Базовый оператор i f позволяет программировать выполнение единственного оператора или блока операторов, заключенных в фигурные скобки, если данное условное выражение оценено как истинное, или же пропустить оператор или блок операторов, если условие оценено как ложное. Это показано на рисунке.



Примеры

if (letter == ‘A’)

cout << “first letter”;

Проверяемое условие помещается в скобки, следующие за ключевым словом *if*, после чего следует оператор, который должен быть выполнен, если условие возвращает *true*. Точка с запятой идет после следующего за *if* со скобками

оператора. Оператор сдвинут, чтобы отметить, что он должен быть выполнен только в том случае, когда условие вернет *true*.

Можно расширить пример.

if (letter == ‘A’)

{

cout << “first letter”;

letter = ‘a’;

}

Операторы блока выполняются, когда условие *true*. В блоке может находиться необходимое количество операторов. Без скобок только первое выражение было бы субъектом *if*.

Вложенные операторы *if*

Оператор, который должен быть выполнен в случае истинности условия if, также может быть еще одним оператором if. Такая организация называется вложенным if. Вложенный if, в свою очередь , может содержать еще один вложенный if.

*if (a > 3)*

*if (a <10)*

*cout <<” a belongs to the interval 3 -10”;*

*cout <<” a does not belongs to the interval”;*

Расширенный оператор if

Эта версия if позволяет выполнить один оператор, когда if возвращает true, и другой, когда оно возвращает false.После этого выполнение программы продолжается со следующего оператора по порядку.

// Листинг 6

// определить знак числа

// использование расширенного оператора if

#include <iostream.h>

void main ()

{

int a;

cout << " input number\n";

cin >> a;

if (a>=0)

cout <<" number positive" <<a<<"\n";

else cout << " namber negative" <<a<<"\n";

}

// Листинг 7

//определить четность числа

// использование расширенного оператора if

#include <math.h>

#include <iostream.h>

int main()

{

long number = 0L;

cout << endl

<< “ input integer number less than two milliards”

<<endl;

cin >> number;

if (number % 2L)

cout << endl

<< “number odd “

<<endl;

else

cout << endl

<< “number even “

<<endl;

return 0;

}

После чтения входного значения в number, оно проверяется на четность путем взятия остатка при делении на 2 и проверки его в условии оператора if. В данном случае условие возвращает целое, а не булевское значение. Оператор if интерпретирует ненулевое значение, возвращенное условием, как true, а нулевое, как false. Эквивалентно такому (number % 2L != 0).

Условие оператора if может быть выражением , возвращающим значение любого из фундаментальных типов данных. Когда условное выражение возвращает числовое значение вместо bool, то компилятор вставляет автоматическое приведение такого результата к типу bool. Приведение к bool ненулевого значения дает true, а нулевого false.

// Листинг 8

//Реализация примитивного калькулятора

//Вложенный оператор if

#include <math.h>

#include <iostream.h>

void main ()

{

double x,y;

char oper;

cout <<"input number1 operator number2\n";

cin >>x>>oper>>y;

if (oper=='+') cout<<"="<<x+y<<"\n";

else if (oper=='-') cout<<"="<<x-y<<"\n";

else if (oper=='\*') cout<<"="<<x\*y<<"\n";

else if (oper=='/') cout<<"="<<x/y<<"\n";

else cout <<"unknoun operator\n";

}

Вложенные операторы *if – else*

Можно вкладывать *if* – else внутрь операторов *if*, а оператор *if* - внутрь *if – else.*

Пример

if (coffee == ‘y’)

if (donuts == ‘y’)

cout << “we have coffee and donuts”;

else

cout << “we have coffee and have not donuts”;

При неправильном структурировании *else* можно допустить ошибку.

*else* всегда относится к ближайшему предшествующему *if*, у которого нет другого *else*.

В случае сложного кода применяются фигурные скобки.

Можно записать следующим образом.

if (coffee == ‘y’)

{

if (donuts == ‘y’)

cout << “we have coffee and donuts”;

else

cout << “we have coffee and have not donuts”;

}

Теперь рассмотрим вложение if внутрь if – else.

if (coffee == ‘y’)

{

if (donuts == ‘y’)

cout << “we have coffee and donuts”;

}

else

{

if (donuts == ‘y’)

cout << “we have not coffee and have donuts”;

}

А теперь рассмотрим случай вложения *if – else* в другие операторы *if – else.*

if (coffee == ‘y’)

{

if (donuts == ‘y’)

cout << “we have coffee and donuts”;

else

cout << “we have coffee and have not donuts”;

}

else

{

if (tea == ‘y’)

cout << “we have not coffee, but have tea and may be donuts…”;

else

cout << “we have not coffee, have not tea but may be some donuts…”;

}

Необходимости в фигурных скобках нет, но код выглядит яснее.

Логические операции и выражения.

Использование if выглядит неуклюже, когда нужно проверить два или более взаимосвязанных условия. В предыдущем примере выяснялась непростая ситуация с кофе и пончиками, но на практике необходимо проверять более сложные условия.

Простое решение обеспечивают логические операции.

&& Логическое И

|| Логическое ИЛИ

! Логическое НЕ

**Логическое И**

Операция логического И (&&) применяется тогда, когда есть два условия и оба должны вернуть результат true, чтобы общий результат был true.

Таблица истинности для операции логического И

|  |  |  |  |
| --- | --- | --- | --- |
|  | условие 2 | | |
| условие 1 | && | false | true |
| false | false | false |
| true | false | true |

Пример

Проверяем, готов ли завтрак

if( (coffee == ‘y’) && (donuts == ‘y’))

cout << “we have coffee and donuts”;

Вывод сообщения произойдет только в том случае , если оба условия, объединенные операцией && окажутся истинными.

Можно проверить находится ли *number* в интервале *0< number <1*

if( (number >0) && (number <1))

cout << “number is situated in an interval”;

else

cout << “number is not situated in an interval”;

**Логическое ИЛИ**

Операция логического ИЛИ(||) применяется тогда, когда имеются два условия и нужно получить результат true, если одно из них или оба возвращают true.

Таблица истинности для операции логического ИЛИ

|  |  |  |  |
| --- | --- | --- | --- |
|  | условие 2 | | |
| условие 1 | || | false | true |
| false | false | true |
| true | true | true |

Пример

Можно проверить находится ли *number* вне интервала 0-1, т.е. number должно быть больше 1 или же меньше 0.

if( (number >0) || (number <1))

cout << “number is situated outside an interval”;

else

cout << “number is situated in an interval”;

**Логическое НЕ**

Третья логическая операция –НЕ(!) – принимает только один операнд *bool* и инвертирует его значение.

Если х имеет значение 10, то выражение *!(х > 5)* будет *false*, т.к. *х > 5* соответствует *true*.

И, наконец, можно применять операцию ! к другим базовым типам данных. Предположим, что имеется переменная *rate* типа *float*, содержащая значение 3.2.

По некоторой причине может возникнуть желание убедиться. что значение *rate* отлично от нуля. Можно использовать следующее выражение *!( rate).*

Значение 3.2 отлично от нуля, и поэтому преобразуется в значение true типа bool, и результат всего выражения будет false.

**Комбинирование логических операций.**

Можно комбинировать условные выражения и логические операции, как будет удобно. например, можно сконструировать тест , определяющий, относится ли символ к буквам, применив единственный оператор if.

// Листинг 9

//Проверка буквы с использованием логических операций

#include <iostream.h>

int main()

{

char letter = 0; //здесь сохранить ввод

cout <<endl

<<"input symbol:";

cin >> letter;

if(((letter >= 'A') && (letter <= 'Z')) ||

((letter >= 'a') && (letter <= 'z'))) //Проверка на //вхождение в алфавит

cout<< endl

<< " you symbol is letter "

<< endl;

else

cout<< endl

<< " you symbol is not letter "

<< endl;

return 0;

}

Первое логическое выражение возвращает *true,* если введена буква верхнего регистра, а второе – если это буква нижнего регистра.

Если в выражении есть операторы && и ||, то часть операторов в нем может не выполниться. Точнее, их выполнится ровно столько, сколько нужно, чтобы узнать результат всего выражения.

Давайте посмотрим вот на такие логические выражения:

*( (i!=0) && (j!=1) )*

*( (i!=0) || (j!=1) )*

В первом выражении, если i равно нулю, то есть первое условие не выполняется, то совершенно не важно, равно ли j единице или нет. В любом случае результат всего выражения будет равен нулю. Точно так же, если во втором выражении i не равно нулю, нет смысла считать дальше - в результате все равно получим единицу.

Факт, на мой взгляд, очень важно знать по двум причинам.

Во-первых, поглядите на вот такое выражение:

*( (i!=0) && (--j!=1) )*

В зависимости от того, выполняется ли первое условие, второе условие может быть проигнорировано при работе задачи, а значит, не выполнится стоящий в нем оператор декремента.

Повторение блока операторов. Цикл.

Возможность повторно выполнять группу операторов – фундаментальна для большинства приложений. Приходится выполнять одни и те же действия многократно. Для реализации этой потребности в языках программирования предусмотрены соответствующие операторы – они называются операторами цикла.

Но как задать желаемую длительность выполнения цикла? Вот тут вам очень пригодятся условные выражения. А сильные не очень.

Цикл выполняет последовательность операторов до тех пор, пока истинно или ложно определенное условие.

Использование цикла for

// Листинг 10

//Суммирование целых чисел циклом for

#include <iostream.h>

int main()

{

int i = 0, sum = 0;

const int max = 10;

**for(i = 1; i <= max; i++)**

**sum += i;**

cout << endl

<< "sum = "<< sum

<<endl

<< "i = "<< i

<<endl;

return 0;

}

Результат

sum = 55

i = 11

Условия, определяющие операцию цикла, появляются в скобках после ключевого слова *for*. В скобках содержатся три выражения, разделенных точкой с запятой.

Первое выражение **i = 1**  выполняется один раз, в начале. и устанавливает начальное условие цикла. В данном случае переменной *i* присваивается значение 1.

Второе выражение **i <= max**  – логическое - определяет, до каких пор должен выполняться оператор цикла ( или блок операторов). Если второе выражение истинно, цикл продолжает выполняться; когда оно ложно, он завершается и выполнение программы продолжается с оператора, расположенного за телом цикла.

В данном случае оператор цикла в строке, следующей за for, выполняется до тех пор , пока значение i меньше или равно max.

Третье выражение **i++** выполняется после оператора цикла (или блока операторов), и в данном случае оно увеличивает *i* на 1 на каждой итерации.

Обобщенная форма цикла for выглядит следующим образом

***for (выражение инициализации; выражение проверки; выражение инкремента)***

***оператор внутри цикла;***

*оператор внутри цикла –* может быть отдельным оператором или блоком операторов в фигурных скобках.

Логику цикла *for* можно рассмотреть на схеме, приведенной ниже.

Прямоугольники обозначают действия, а ромбы – условие.



Вариации цикла *for*

Выражение инициализации может включать объявление переменной цикла

*for (int i = 1; i<= max; i++)*

*sum += i;*

Но, в этом случае , переменная цикла исчезает после завершения цикла.

Можно вообще исключить инициализирующее выражение из цикла.

*int i = 1;*

*for (; i<= max; i++)*

*sum += i;*

Но по-прежнему важна точка с запятой, отделяющая выражение инициализации от проверочного условия цикла. Фактически обе точки с запятой должны присутствовать, независимо от того, пропущено ли какое-то одно из управляющих выражений. иначе компилятор не может понять, какой именно из трех управляющих операторов отсутствует.

Цикл for может быть пустым. Можно поместить оператор цикла в выражение инкремента.

*for (i = 1; i<= max; sum += i++);*

Точка с запятой необходима, чтобы указать, что оператор цикла пуст.

В каждый из разделов for можно вставить столько выражений, сколько понадобиться.

*for (i = 0,power = 1; i<= max; i++, power += power);*

Цикл *while*

Второй тип циклов в С++ цикл *while.* В то время как цикл *for* предназначен главным образом для повторения оператора или блока операторов определенное количество раз, цикл *while* служит для выполненияоператора или блока до тех пор, пока указанное условие остается истинным. Общая форма цикла *while* :

*while (условие)*

*оператор внутри цикла;*

Здесь *оператор внутри цикла* выполняется повторно до тех пор, пока выражение *условие* имеет значение t*rue*. После того, как условие становится равным *false*, программа выходит из цикла и переходит к оператору, следующему за ним. Единственный *оператор внутри цикла* может быть заменен блоком операторов в фигурных скобках.



// Листинг 11

//использование цикла while для вычисления среднего арифметического

#include <iostream.h>

int main()

{

double value = 0.0; //Введенное значение

double sum = 0.0; //сумма значений

int i= 0; //счетчик значений

char indicator = 'y'; //значение индикатора продолжения

while(indicator == 'y')

{

cout <<endl

<<"input number:";

cin >> value; //читать значение

++i; //увеличить счетчик

sum = sum + value; //добавить текущее значение к сумме

cout <<endl

<<"input number (for finish input n)";

cin >> indicator; //читать индикатор

}

cout << endl

<< " AV from " << i

<< endl

<< sum/i

<< endl;

return 0;

}

Результат

input number:5

input number (for finish input n)y

input number:6

input number (for finish input n)y

input number:7

input number (for finish input n)y

input number:8

input number (for finish input n)n

AV from 4

6.5

Цикл do-while

Цикл *do-while* подобен циклу *while* в том, что он выполняется до тех пор, пока указанное условие остается истинным. Главное отличие состоит в том, что здесь условие проверяется в конце цикла – что отличает его от циклов *while* и *for*, где условие проверяется в начале.

Общая форма цикла *do-while*:

*do*

{

*оператор внутри цикла;*

*}while (условие);*

логика этой формы цикла представлена на рисунке



// Листинг 12

//использование цикла do-while для вычисления среднего арифметического

#include <iostream.h>

int main()

{

double value = 0.0; //Введенное значение

double sum = 0.0; //сумма значений

int i= 0; //счетчик значений

char indicator = 'y'; //значение индикатора продолжения

do

{

cout <<endl

<<"input number:";

cin >> value; //читать значение

++i; //увеличить счетчик

sum = sum + value; //добавить текущее значение к сумме

cout <<endl

<<"input number (for finish input n)";

cin >> indicator; //читать индикатор

}

while(indicator == 'y');

cout << endl

<< " AV from " << i

<< endl

<< sum/i

<< endl;

return 0;

}

input number:5

input number (for finish input n)y

input number:6

input number (for finish input n)y

input number:7

input number (for finish input n)y

input number:8

input number (for finish input n)y

input number:9

input number (for finish input n)n

AV from 5

7

Между двумя последними версиями цикла нет особой разницы, за исключением того, что правильная работа этой версии не зависит от начального значения переменной индикатора.