Loops and Relational Expressions:

Computers do more than store data. They analyze, consolidate, rearrange, modify, extrapolate, synthesize, and otherwise manipulate data. Sometimes they even distort and trash data.

To perform their manipulative miracles, programs need tools for performing repetitive actions and for making decisions.

Of course, C++ provides such tools. Indeed, it uses the same for loops, while loops, do while loops, if statements, and switch statements, that regular C employs.

General Form:

for (initialization; test-expression; update-expression)

body

See the program – ForLoops1.cpp

Each part is an expression, and semicolons separate the expressions from each other. The statement following the control section is called the body of the loop, and it is executed as long as the test expression remains true.

Actually, C++ doesn't limit test-expression to true/false comparisons. You can use any expression, and C++ will type cast it to type bool. Thus, an expression with a value of 0 is converted to the bool value false, and the loop terminates. If the expression evaluates to nonzero, it is type cast to the bool value true, and the loop continues.

See the program – ForLoops2.cpp

A C++ expression is a value or a combination of values and operators, and every C++ expression has a value.

See The Program – ForLoops3.cpp

C++ new for loop syntax:

for (for-init-statement condition ; expression)

statement

See The Program – ForLoopsFact.cpp

You Can also use your own value to increment or decrement the variable.

See The Program – ForLoopsUserTable.cpp

You can also reverse the string using for loops:

See the program – ReverseString.cpp

The **Increment**(++) and **Decrement**(--) Operator:

Each operator comes in two varieties. The prefix version comes before the operand, as in ++i. The postfix version comes after the operand, as in i++.

The two versions have the same effect on the operand, but they differ in terms of when they take place.

The notation ***i++*** means "use the current value of ***i*** in evaluating an expression, and then increment the value of ***i***."

Similarly, the notation ***++i*** means "first increment the value of ***i*** and then use the new value in evaluating the expression.

See The Program – IncrementDecrement.cpp

Side Effects and Sequence Points:

A side effect is an effect that occurs when evaluating an expression modifies something, such as a value stored in a variable.

A sequence point is a point in program execution at which all side effects are guaranteed to be evaluated before going on to the next step.

In C++ the semicolon in a statement marks a sequence point. That means all changes made by assignment operators, increment operators, and decrement operators in a statement must take place before a program proceeds to the next statement.

In short, for built-in types, it most likely makes no difference which form you use. For user-defined types having user-defined increment and decrement operators, the prefix form is more efficient.

The Increment/Decrement Operators and Pointers:

You can use increment operators with pointers as well as with basic variables. Recall that adding an increment operator to a pointer increases its value by the number of bytes in the type it points to. The same rule holds for incrementing and decrementing pointers.

double arr[5] = {45.8, 457.85, 85.65, 98.545, 59.25};

double *ptr = arr; // ptr points to, i.e. arr[0] = 45.8 and so on

++ptr; // ptr points to, i.e. arr[0] = 457.85 and so on

You can also use these operators to change the quantity a pointer points to by using them in conjunction with the * operator. Applying both * and ++ to a pointer raises the questions of what gets dereferenced and what gets incremented. Those actions are determined by the placement and precedence of the operators. The prefix increment, prefix decrement, and dereferencing operators all have the same precedence and associate from right to left. The postfix increment and decrement operators both have the same precedence, which is higher than the prefix precedence. These two operators associate from left to right.

E.g.

*++ptr; // increment pointer, 85.65 or arr[2]

while,

++*ptr; // increment the pointed to value, change 85.65 to 86.65

Here, ptr remains pointing to arr[2].

*ptr++; // dereference original location, then increment pointer.

Thus, the value of *ptr++ is arr[2], or 85.65, but the value of ptr after the statement completes is the address of arr[3].

Combination Assignment Operators:

x = x + y; is same as x+=y.

This implies that the left operand must be something to which you can assign a value, such as a variable, an array element, a structure member, or data you identify by dereferencing a pointer.

int k = 5;

k += 3; // K set to 8

int *pa = new int[10]; // pa points to pa[0]

pa[4] = 12;

pa[4] += 6; // pa[4] set to 18

*(pa + 4) += 7; // pa[4] set to 25

pa += 2; // pa points to the former pa[2]

34 += 10; // quite wrong

Other Operator:

-=, *=, /=, %=.

More Syntax Tricks—The Comma Operator:

The comma is not always a comma operator.

int i, j; // comma is a separator here, not an operator

Check the program – ForLoops4.cpp

Comma Operator Tidbits:

C++ does provide the operator with two additional properties. First, it guarantees that the first expression is evaluated before the second expression. (In other words, the comma operator is a sequence point.)

i = 15, j = 5*i; // i set to 15, then j set to 75

Second, C++ states that the value of a comma expression is the value of the second part of the expression.

The comma operator has the lowest precedence of any operator. E.g.

data = 45, 55;

get read as (data = 45), 55; // data set to 45, 55 does nothing

But,

data = (45, 55); // cat set to 55, Because the value of the expression on the right

// of the comma

Relational Expressions:

for (cin >> x; x == 0; cin >> x) // continue while x = 0

The relational operators have a lower precedence than the arithmetic operators.

e.g.

x + 4 > y -5; // expression 1

corresponds to this

(x+4) > (y-5); // expression 2

and not this:

x + (4>y) – 5; // expression 3

Expression 3 can also be solved as value is promoted to int and the value will be either 0 or 1.

Caution:

Don't use = to compare for equality; use ==.

If you use this in for loop the loop may run unlimited number of times or it can freeze all other programs and you might need to reboot your system.

Comparing C-Style Strings:

Because C++ handles C-style strings as addresses, you get little satisfaction if you try to use the relational operators to compare strings. Instead, you can go to the C-style string library and use the strcmp() function to compare strings. This function takes two string addresses as arguments. That means the arguments can be pointers, string constants, or character array names. If the two strings are identical, the function returns the value 0.If the first string precedes the second alphabetically, strcmp() returns a negative value, and if the first string follows the second alphabetically, strcmp() returns a positive value. Actually, "in the system collating sequence" is more accurate than "alphabetically." This means that characters are compared according to the system code for characters .

See the program – CmprStr.cpp

You can do the same program using String.

See the program – CmprStr2.cpp

The *while* Loop:

The while loop is a for loop stripped of the initialization and update parts; it has just a test condition and a body:

```
while (test-condition)
        body
```

Check the program – While1.cpp

for Versus *while*:

In C++ the for and while loops are essentially equivalent.

For Loop:

```
for(init-expression; test-expression; update-expression){
        statement(s);
}
```

Could be re-written this way:

```
init-expression;
while(test-expression){
        statement(s);
        update-expression;
}
```

Similarly while loop

```
while (test-expression){
        body
}
```

Could be re-written this way:

```
for ( ; test-expression;){
        body
}
```

This for loop requires three expressions (or, more technically, one statement followed by two expressions), but they can be empty expressions (or statements).Only the two semicolons are mandatory.

```
for ( ; ; ){
        body
}
```

Incidentally, a missing test expression in a for loop is construed as true, so this loop runs forever.

There are three differences:

1. As just mentioned, is that an omitted test condition in a for loop is interpreted as true.
2. You can use the initializing statement in a for loop to declare a variable that is local to the loop; you can't do that with a while loop.
3. Finally, There is a slight difference if the body includes a continue statement.

Typically, programmers use for loops for counting loops because the for loop format enables you to place all the relevant information—initial value, terminating value, and method of updating the counter—in one place. Programmers most often use while loops when they don't know in advance precisely how many times a loop will execute.

Building a Time-Delay Loop:

We'll add some delays in our program.

You can use the system clock for do the timing for you.

Because if you define your custom time, the problem you have to face is the counting limit when you change the computer processor speed.

The ANSI C and the C++ libraries have a function to help you do this. The function is called clock(), and it returns the system time elapsed since a program started execution. There are a couple complications, though. First, clock() doesn't necessarily return the time in seconds. Second, the function's return type might be long on some systems, unsigned long on others, and perhaps some other type on others.

But the ctime header file (time.h on less current implementations) provides solutions to these problems.First, it defines a symbolic constant, CLOCKS_PER_SEC, that equals the number of system time units per second. So dividing the system time by this value yields seconds. Or you can multiply seconds by CLOCKS_PER_SEC to get time in the system units. Second, ctime establishes clock_t as an alias for the clock() return type. This means you can declare a variable as type clock_t, and the compiler converts it to long or unsigned int or whatever is the proper type for your system.

See the program – Waiting.cpp

Type Aliases:

C++ has two ways to establish a new name as an alias for a type. One is to use the preprocessor.

#define BYTE char // preprocessor replaces BYTE with char

The preprocessor then replaces all occurrences of BYTE with char when you compile a program, thus making BYTE an alias for char.

The second method is to use the C++ (and C) keyword typedef to create an alias. For example, to make byte an alias for char, you use this:

typedef char byte; // makes byte an alias for char

General Form:

typedef typeName aliasName ;

To make byte_pointer an alias for pointer-to-char

typedef char * byte_pointer; // pointer to char type

typedef ability to handle more complex type aliases makes using typedef a better choice than #define—and sometimes it is the only choice.

Notice that typedef doesn't create a new type. It just creates a new name for an old type.

The *do while* loop:

It's different from the other two(for and while) because it's an exit-condition loop.

That means this devilmay-care loop first executes the body of the loop and only then evaluates the test expression to see whether it should continue looping. If the condition evaluates to false, the loop terminates; otherwise, a new cycle of execution and testing begins. Such a loop always executes at least once because its program flow must pass through the body of the loop before reaching the test.

```
do
        body
while (test-expression);
```

See the program – DoWhile.cpp

The Range-based for loop(C++11):

The C++11 adds a new form of loop called the range-based for loop. It simplifies one common loop task—that of doing something with each element of an array , or, more generally, of one of the container classes, such as vector or array.

```
double prices [5] = {45.9, 84.5, 95.6, 44.6, 25.8};
for (double x : prices)
        cout << x << endl;
```

This loop displays every value included in the range of the array.

To modify array values, you need a different syntax for the loop variable:

To modify array values, you need a different syntax for the loop variable:

```
for (double &x : prices)
        x = x * 0.75; //25%  off sale
```

The & symbol identifies x as a reference variable.

The range-based for loop also can be used with initialization lists:

```
for (int x : {3, 5, 2, 8, 6})
        cout << x << " ";
```

Using Unadorned cin for Input:

If a program is going to use a loop to read text input from the keyboard, it has to have some way of knowing when to stop. How can it know when to stop? One way is to choose some special character, sometimes called a sentinel character, to act as a stop sign.

See the program – WhenToStop.cpp

Apparently, sam runs so fast that he obliterates space itself—or at least the space characters in the input.

cin.get(char) to the Rescue:

Usually, programs that read input character-by-character need to examine every character, including spaces, tabs, and newlines. The istream class (defined in iostream),to which cin belongs, includes member functions that meet this need. In particular, the member function cin.get(ch) reads the next character, even if it is a space, from the input and assigns it to the variable ch. By replacing cin>>ch with this function call, you can fix previous program.

See the program – WhenToStop2.cpp

Some systems do not support simulated EOF from the keyboard. Other systems support it imperfectly. If you have been using cin.get() to freeze the screen until you can read it, that won't work here because detecting the EOF turns off further attempts to read input. However, you can use a timing loop like we did earlier to keep the screen visible for a while. Or you can use cin.clear(), to reset the input stream.

You can read char to the end of the file.

See the program – WhenToStop3.cpp

You should realize that EOF does not represent a character in the input. Instead, it's a signal that there are no more characters.

See the program – WhenToStop4.cpp

Two Dimensional Array:

```
int maxtemps [m][n] = {
        {00, 01, 02, ….., 0n},
        {10, 11, 12, ….., 1n},
        .
        .
        .
        {m0, m1, m2, ….., mn},
}
```

You can store and multiple cities and their temperature in years.

See the program – CityTemp2D.cpp

Thank You