6.096 Introduction to C++
January (IAP) 2009

6.096
Lecture 2
Flow of Control, Conditional Constructs, Loops

Conditional Constructs

if-else construct: executes a set of statements if the specified condition is true

Syntax:

```
if (<condition>)        {
  statement1;
  statement 2;
  ….
  statement n;
}
else
{
  statement1;
  statement 2;
  ….
  statement n;
}
```

In this: if the specified condition is true, then the statements in the first curly braces will be executed, else the statements in the next curly braces will be executed.

A few points to note:

- If there is only one statement following the *if*, that is only one statement to be executed if the specified condition is true, then the curly braces are optional. Similarly for the statements following the *else*.
- The *else* is optional in the construct.
- An *else*, without an *if* has no meaning.

The *if-else* constructs executes statements based on conditions, but how do we write conditions?

To write conditions we need **Relational Operators**
Relational or Comparison Operators are used to compare values. Return Boolean value of True or False (1 or 0) to the statements they are in.
<, <=,>,=>, !=, ==

eg:

```
if (a>b)
  cout<<a;
```

```
else
  cout<<b;
```

In the above construct, if the condition is true i.e if a>b, then the relational operator >, returns a value True (or 1) to the statement it is in, i.e in the *if* statement and thus the value of the variable a is displayed. If the condition is false, (i.e if a<=b), then the relational operator returns a value False (or 0) to the if statement and control goes to the statement after the *else*, and the value of variable b is displayed.

Note**:** Be careful that to compare two values == has to be used, else it is an assignment statement!

To write more than one condition we use Logical Operators. These are ||, &&, !
&&- AND
All the conditions connected by the AND operator have to be true for the statement to evaluate to true.

Eg:

```
if ((a>b)&&(a>c))
  cout<<a;
```

Here if both conditions are true i.e if a>b and a>c, then the statement will evaluate to true and the value of variable a will be displayed

||- OR
Any one of the conditions connected by the AND operator have to be true for the statement to evaluate to true.

```
if ((a>b)||(a>c))
  cout<<a;
```

Here if any of the conditions is true i.e if a>b or a>c, then the statement will evaluate to true and the value of variable a will be displayed

!- NOT
It is used to negate a particular condition i.e it converts true to false, and false to true

Eg:

```
if !(a>b)
  cout<<a;
```

In the above example, if a>b, then the statement should return a value true, but the not operator negates this, and returns a value false to the statement instead, and nothing is displayed.

On the other hand, if the condition is false, (i.e if a<=b), then the statement should return a value false, but the not operator negates this, and returns a value true to the statement instead, and the value of the variable a is displayed.

Nested if-else ladders- used when the *if-else* construct has to be used over and over again.

```
if (<condition1>)
{
  statement1;
  ….
  statement n;
}
else if (<condition2>)// if the previous condition is false, then a new construct begins,
{                          //   based on another condition
  statement1;
  ….
  statement n;
}
else if (<condition 3>)
{
  statement1;
  ….
  statement n;
}
…….// This can continue any number of times
else if (<condition n>)
{
  statement1;
  ….
  statement n;
}
else
{
  statement1;
  ….
  statement n;
}
```

Note: The last else is optional

Loops: Used to repeatedly execute a set of statements till a given statement is true.

while loop- executes a set of statements as long as the condition specified at the beginning is true. The condition is evaluated at the beginning of the loop, so it has to be true in the beginning for the loop to execute even once.

Syntax:

```
while(<condition>)
{
  statement1;
  …
  statement n;
}
```

Note: Similar to the if statement, the curly braces are optional if there is only one statement in the loop

e.g

```
int a=1;
while(a<=3)
{
  cout<<a++<<endl;
}
```

Output:

```
1
2
3
```

In the above example, as long as the condition continues to be true (i.e as long as a<=3), the statement within the loop will continue to be executed over and over again.

do-while loop- it is similar to while loop except the condition is at the end of the loop. As a result, the loop always executed at least once, regardless of whether the condition is true or not.
Note: A semicolon at the end and curly braces are always required.

Syntax:

```
do
{
  statement 1;
  ..
  statement n;
}while(<condition>);
```

e.g
```
int a=1;
do
{
  cout<<a++<<endl;
}while(a<=3);
```

The output is the same as of the while loop above. The difference is when the condition is false in the beginning itself. E.g if we have assigned ‘a’ a value 6 in the beginning instead of 1, the while loop will not execute even once as the condition is evaluated in the beginning, but the do-while loop will execute at least once (6 will be displayed) as the condition is evaluated only at the end of the first iteration.

For loop- It is used to execute a set of statements as long as the condition specified is true.

E.g

```
for(i=1;i<=4;i++)
{
  cout<<i;
}
```
Output: 1234

- i=1 is called the initialization expression, and is evaluated only once when the loop is first encountered. It sets the value of the variable i to 1
- i<=4 is called the condition, or the test expression and is evaluated before every iteration of the loop. As long as this condition remains true, the loop continues to be true
- i++ is called the increase or decrease expression, and is executed at the end of each iteration of the loop. If there is no increase or decrease expression, then the loop results into an infinite loop.
- The curly braces are optional if the loop has only 1 statement.

The initialization and increase expression can be left blank, and the tasks they do can be done outside and inside the loop respectively.

e.g
```
int i=1;
for( ; i<=4; )
{
  cout<<i++;
)
```
A for loop can have more than one variable.

e.g

```
for(i=1, j=2; i<=4; i++, j+=2)
{
  cout<<i<<':'<<j<<endl;
}
```

- i=1, j=2 is the combined initialization expression.
- i<=4 is the condition. Note that if you write more than one condition in the for loop, then some compilers might return an error. For more than one condition, it is better to use while or do-while loops.
- i++, j+=2 is the combined increase expression.

A loop without statements: Note the semicolon at the end of the for statement

```
for(i=1;i<=3;i++);
cout<<i;
```

Output: 4

This loop keeps repeating the for loop till the specified condition is true (i.e till i<=3). After i becomes 4, the loop is exited, and then the next statement (cout) is executed.

Often, one construct is used inside another. E.g using if-else constructs within loops, etc.

Nested loops- using one looping construct inside another. Used when for one repetition of a process, many repetitions of another process are needed. Some applications are- to print patterns, find sum of a repeating series etc.

Eg.
Pattern:
1
12
123
1234

- The outer loop runs the number of times the pattern has to be repeated (in the above case 4 times. Thus it can run from 1 to 4, from 4 to 1, from 2 to 6 etc.
- The inner loop depends on the values being displayed, and often has to be related to the outer loop

Thus we can write the following code:

```
for(i=1;i<=4;i++)
{
  for(j=1;j<=i; j++)
```

```
    cout<<j;
  cout<<endl;
}
```