CSC 201 CSC 150
C++ Programming

Dr. Mazen Zainedin

Stat & OR Dept.

College of sciences KSU

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

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;
```

The condition is true if a>b, and is false if a<=b

To write more than one condition we use Logical Operators. ||, &&, !

&&-AND

All the conditions connected by the AND operator have to be true for the statement to evaluate to true.

e.g.:

```
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

||-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

!-NOT

It is used to negate a particular condition

i.e it converts true to false, and false to true

```
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

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>)
   {   statement1;  ….  statement n;   }
…
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

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.

Syntax:

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

Note: A semicolon at the end and curly braces are always required.

Example:

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

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

Output:

1

2

3

For loop

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

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

Output:

1 2 3 4

i=1 initialization expression, It sets the value of the variable i to 1

i<=4 test expression and is evaluated before every iteration of the loop

i++ increase or decrease expression, and is executed at the end of each iteration of the loop

Note:
-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.
- A for loop can have more than one variable.
- A loop without statements: Note semicolon at the end of for statement

```
int i=1;
 for( ; i<=4; )
 {
   cout<<i++;
 }
```

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

```
int i;
for (i=1;i<=3;i++);
cout<<i;
Output: 4
```

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

Thus we can write the following code:

```
for(i=1;i<=4;i++)    //  outer loop
{
for(j=1;j<=i; j++)    //  inner loop
cout<<j;
cout<<endl;
}
```