C

Control Structures

Instructor: Maninder Kaur

Email: maninder.kaur2@sheridancollege.ca

Course: PROG20799

Conditional Statements

- While writing a program, there may be a situation when you need to adopt one path out of the given two paths.
- So you need to make use of conditional statements that allow your program to make correct decisions and perform right actions.
- C supports conditional statements which are used to perform different actions based on different conditions.

Conditional Statements

- C programming language supports following conditional statements:

Statement	Description
if statement	An if statement consists of a Boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
if...else if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.

if Statement

- The if statement is the fundamental control statement that allows C language to make decisions and execute statements conditionally.

Syntax:

```
if (condition)
{
   Statement(s) to be executed if expression is true
}
```

- Here condition is evaluated.
- If the resulting value is true, given statement(s) are executed.
- If expression is false then no statement would be executed.

if Statement - Example

```
#include <stdio.h>

main ()
{
   int a = 10;

   if( a < 20 )
   {
       printf("a is less than 20\n" );
   }

   printf("value of a is : %d\n", a);
}
```

if…else Statement

- The `if...else` statement is the next form of control statement that allows C to execute statements in more controlled way.

Syntax:

```
if (condition)
{
   Statement(s) to be executed if expression is true
}
else
{
   Statement(s) to be executed if expression is false
}
```

- Here `condition` is evaluated.
- If the `condition` is true, given statement(s) in the `if` block, are executed.
- If `condition` is false then given statement(s) in the `else` block, are executed.

if…else Statement - Example

```
#include <stdio.h>

main ()
{
   int a = 100;

   if( a < 20 )
   {
       printf("a is less than 20\n" );
   }
   else
   {
       printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);
}
```

if…else if… Statement

- An `if` statement can be followed by an optional `else if...else` statement, which is very useful to test various conditions using single `if...else if` statement.

Syntax:

```
if (condition 1){
   Statement(s) to be executed if condition 1 is true
} else if (condition 2){
   Statement(s) to be executed if condition 2 is true
} else if (condition 3){
   Statement(s) to be executed if condition 3 is true
} else{
   Statement(s) to be executed if no condition is true
}
```

- There is nothing special about this code, it is just a series of `if` statements, where each `if` is part of the `else` clause of the previous statement.
- Statement(s) are executed based on the true condition, if non of the condition is true then the last `else` block is executed.

if…else if… Statement - Example

```
#include <stdio.h>

main ()
{
   int a = 100;

   if( a == 10 )
   {
       printf("Value of a is 10\n" );
   }
   else if( a == 20 )
   {
       printf("Value of a is 20\n" );
   }
   else if( a == 30 )
   {
       printf("Value of a is 30\n" );
   }
   else
   {
       printf("None of the values is matching\n" );
   }

   printf("Exact value of a is: %d\n", a );
}
```

`switch` Statement

- The basic syntax of the `switch` statement is to give an expression to evaluate and several different statements to execute based on the value of the expression.
- The compiler checks each `case` against the value of the expression until a match is found. If nothing matches, a `default` condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

- The `break` statements indicate to the compiler the end of that particular case.
- If they were omitted, the compiler would continue executing each statement in each of the following cases.

switch Statement - Example

```
#include <stdio.h>

main ()
{
   char grade = 'B';

   switch(grade)
   {
   case 'A' :
      printf("Excellent!\n" );
      break;
   case 'B' :
      printf("Very good\n" );
      break;
   case 'C' :
      printf("Well done\n" );
      break;
   case 'D' :
      printf("You passed\n" );
      break;
   case 'F' :
      printf("Better try again\n" );
      break;
   default :
      printf("Invalid grade\n" );
   }
}
```

Loop Statements

- While writing a program, there may be a situation when you need to perform some action over and over again.
- In such situation, you would need to write loop statements to reduce the number of lines.
- C supports all the necessary loops to help you on all steps of programming.

`while` Loop

- The purpose of a `while` loop is to execute a statement or code block repeatedly as long as the *condition* is true.
- Once *condition* becomes false, the loop will exit.

Syntax:

```
while (condition)
{
   Statement(s) to be executed if expression is true
}
```

while Loop - Example

```
#include <stdio.h>

main ()
{
   int a = 10;

   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }
}
```

do…while Loop

- The `do...while` loop is similar to the `while` loop except that the condition check happens at the end of the loop.
- This means that the loop will always be executed at least once, even if the condition is false.

Syntax:

```
do
{
   Statement(s) to be executed;
} while (condition);
```

- Note the *semicolon* (`;`) used at the end of the `do...while` loop.

do…while Loop - Example

```
#include <stdio.h>

main ()
{
   int a = 10;

   do
   {
       printf("value of a: %d\n", a);
       a = a + 1;
   } while( a < 20 );
}
```

for Loop

- The `for` loop is the most compact form of looping and includes the following three important parts:
  - The ***loop initialization*** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
  - The ***test statement*** which will test if the given condition is true or not. If condition is true, then code given inside the loop will be executed otherwise the loop will exit.
  - The ***iteration statement*** where you can increase or decrease your counter.

Syntax:

```
for (initialization; test condition; iteration statement)
{
   Statement(s) to be executed if test condition is true
}
```

for Loop - Example

```
#include <stdio.h>

main ()
{
   int a;

      for(a = 10; a < 20; a++ )
      {
         printf("value of a: %d\n", a);
      }
}
```

C Loop Control

- C language provides you full control to handle your loops and switch statement.
- There may be a situation when you need to come out of a loop without reaching at its bottom.
- There may also be a situation when you want to skip a part of your code block and want to start next iteration of the loop.
- To handle all such situations, C provides `break` and `continue` statements.
- These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

`break` Statement

- The `break` statement in C programming language has following two usage:
- When the `break` statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the `switch` statement.
- If you are using nested loops (i.e. one loop inside another loop), the `break` statement will stop the execution of the innermost loop and start executing the next line of code after the block.

break Statement - Example

```
#include <stdio.h>

main()
{
   int a = 10;

   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;

   if( a > 15)
      {
          break;
      }
   }
}
```

`continue` Statement

- The `continue` statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

- When a `continue` statement is encountered:
  - program flow moves to the loop
  - checks condition immediately
  - and if condition remains true then it start next iteration
  - otherwise control comes out of the loop.

`continue` Statement - Example

- This example illustrates the use of a `continue` statement with a `do…while` loop. Notice how the `continue` statement is used to skip printing if value of a is 15:

```
#include <stdio.h>

main()
{
   int a = 10;

   while( a < 20 )
   {
      a++;

      if( a == 15)
      {
         continue;
      }
      printf("value of a: %d\n", a);
   }
}
```

Any questions please?