Looping and while() loops

Comp Sci 1570 Introduction to C++

Outline

1. Control flow introduction
   - Loops
     - Loop control variable
     - Debugging
2. while loop
   - exiting loops
     - break
     - continue
     - exit

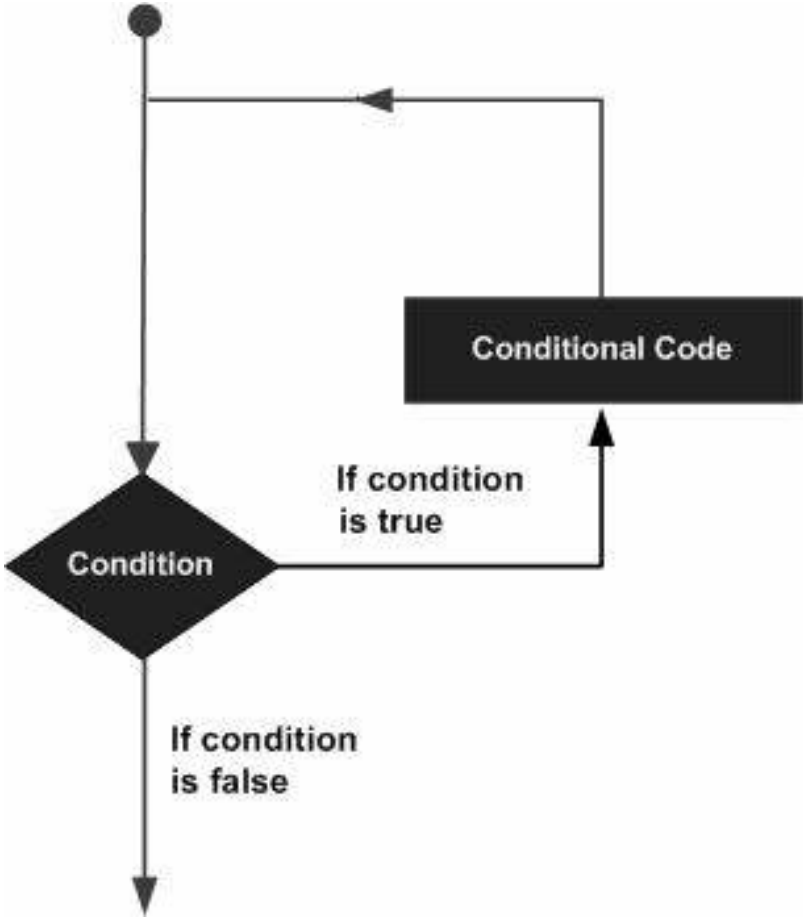
Control flow

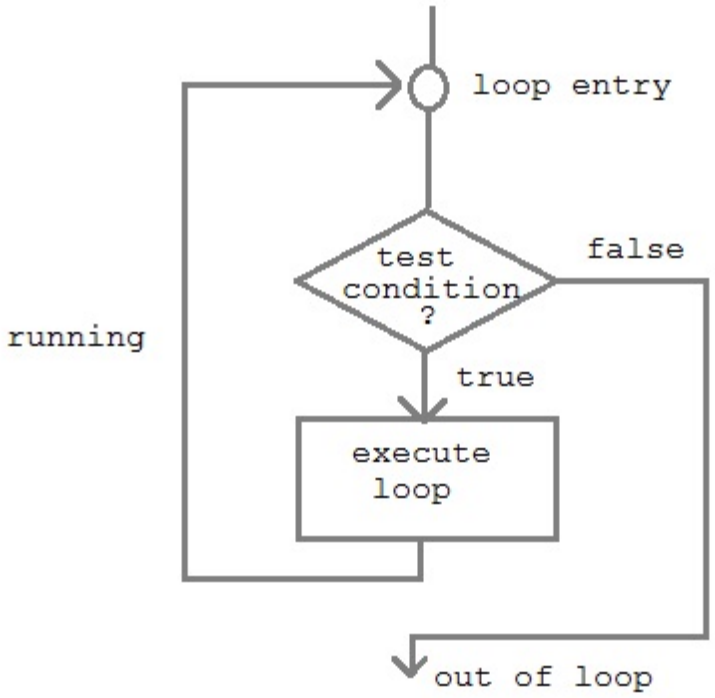
- if else
- switch case
- ternary
- Sentinel loops
  - while
  - do while
- counting loops
  - for
- more?

Outline

1. Control flow introduction
   - Loops
     - Loop control variable
     - Debugging
2. while loop
   - exiting loops
     - break
     - continue
     - exit

Loops





Loops and computation

- Being able to repeat an operation of any kind is the last capability that we need to complete a fully functional toolbox for computing.
- In every programming language there are looping structures.
- But in general, there are two kinds of loops: sentinel loops and counting loops.
- A sentinel loop has a sentinel value that triggers the termination of the looping.
- You use a counting loop when you know a priori how many times you wish the loop body to repeat.
- In C++, there are two sentinel loops: the do-while statement and the while statement.
- There is one counting loop, it is the for statement.

Variable re-assignment

It is legal to make more than one assignment to the same variable.

```
x=5;        // 5
x=7;        // 7
x=x+1;      // 8
x++         // 9
x+=1;       // 10
```

A new assignment makes an existing variable refer to a new value (and stop referring to the old value).

Loop control variable

In every loop, sentinel or counting, there should be a Loop Control Variable (LCV), and there should also be statement(s) to do the following three actions:

- initialize the LCV
- evaluate/check value of LCV
- update the LCV

If any of these three components are left out of your code for the loop, you run the risk of having a loop that either does nothing or does something, an "infinite loop"!

Infinite loops

- If you find yourself with an infinite loop, key in ctrl-C (push the 'ctrl' and 'c' buttons simultaneously). This will kill the process.
- The only way to exit an infinite loop is through a **return** statement, a **break** statement, an **exit** statement, a **goto** statement, an **exception** being thrown, or the user killing the program.
- Programs that run until the user decides to stop them sometimes intentionally use an infinite loop along with a return, break, or exit statement to terminate the loop.
- It is common to see this kind of infinite loop in web server applications that run continuously and service web requests.

Debugging

- One way to debug an infinite loop (or any bad code) is to place simple output messages in your code at appropriate places.
- First, try to reason how much of the code is being executed before the problem occurs.
- Then insert statements like:

```
cout<< "made it to this point" <<endl;
```

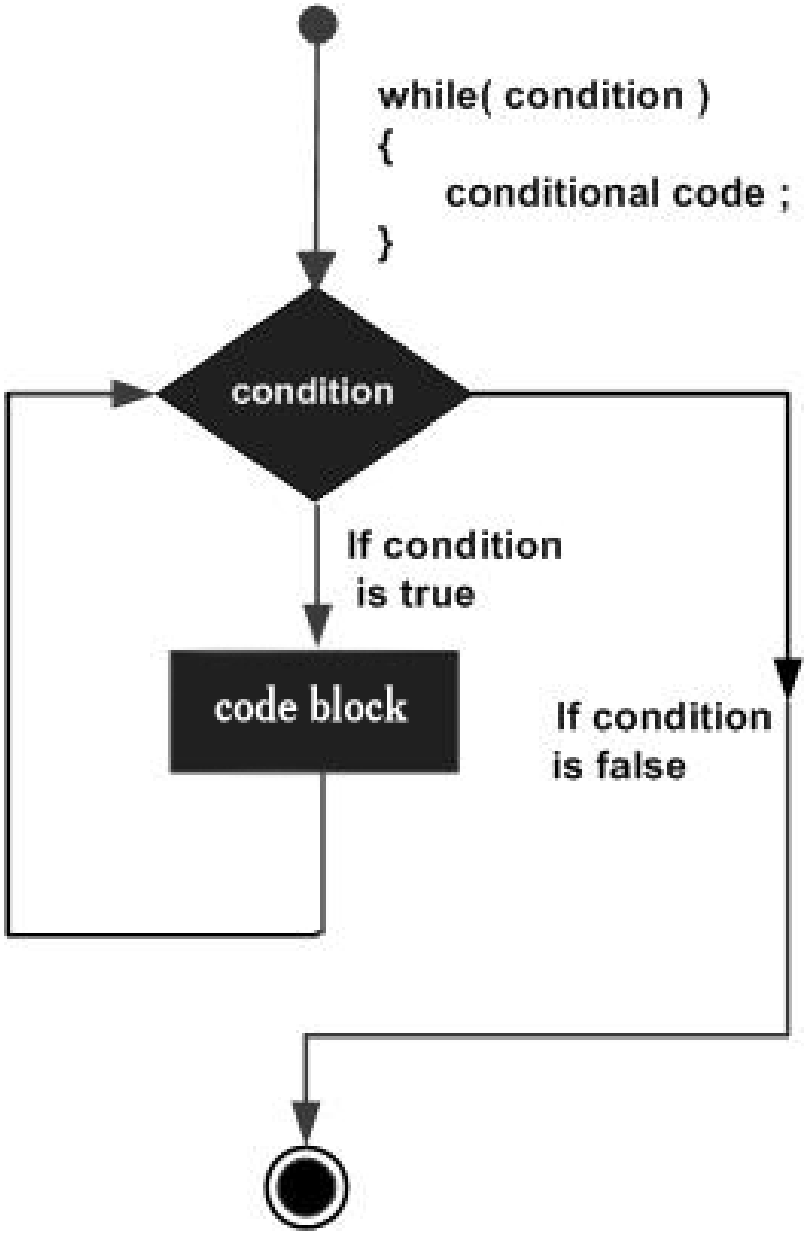
- If that message is displayed to the screen, you know the problem is after that point in the code.
- Keep doing this until you have narrowed it down.
- Warning: be sure to put the endl at the end of the cout statement, which will flush the output buffer so that you are sure the message is going to the screen before any other process is executed.

Outline

1 Control flow introduction
Loops
Loop control variable
Debugging

2 while loop
exiting loops
break
continue
exit

while loop



The syntax for the while is:


```
while(expression)
    statement
```

- The while-loop simply repeats statement while expression is true.
- This sequence of statement to be executed is kept inside the curly braces known as loop body.
- After every execution of loop body, condition is checked, and if it is found to be true the loop body is executed again.
- When condition check comes out to be false, the loop body will not be executed.

while requirements

- **while** is a reserved word.
- Expression must be fully contained within the parentheses.
- Expression is a valid C++ expression that evaluates to true or false or a numerical value.
- Statement is a simple or compound C++ statement (with all semi-colons included).
- This is a pre-test loop, which means that the condition in expression is checked before the body of the while loop (statement) might possibly be executed. This implies that the body of the loop may never be executed.
- Evaluation: expression is evaluated. If it is true, the body (statement) is executed and control passes back up to expression to be evaluated again. If it is false, control passes out of the loop statement.

Loop control variables

- To execute a certain number of times, use a loop variable, often called a counter.
- A loop variable is an integer variable that is declared for the sole purpose of counting how many times a loop has executed.
- Loop variables are often given simple names, such as i, j, or k.
- If you want to know where in your program a loop variable is used, and you use the search function on i, j, or k, the search function will return half your program! Many words have an i, j, or k in them.
- Consequently, a better idea is to use iii, jjj, or "countRevolutions" as your loop variable names.
- Because these names are more unique, this makes searching for loop variables much easier, and helps them stand out as loop variables.

Scope

- Each time a loop executes, it is called an iteration.
- Because the loop body is typically a block, and because that block is entered and exited with each iteration, any variables declared inside the loop body are created and then destroyed with each iteration.

Outline

1 Control flow introduction
Loops
Loop control variable
Debugging

2 while loop
exiting loops
break
continue
exit

break

- C ++ provides the break statement to implement middle-exiting control logic.
- The break statement causes the immediate exit from the body of the loop.

continue

- When a program's execution encounters a break statement inside a loop, it skips the rest of the body of the loop and exits the loop.
- The continue statement is similar to the break statement, except the continue statement does not necessarily exit the loop.
- The continue statement skips the rest of the body of the loop and immediately checks the loop's condition.
- If the loop's condition remains true, the loop's execution resumes at the top of the loop.

exit(0)

- If you want to terminate the whole program, use exit(), as we discussed last class