Increment and the While

Class 15

Increment and Decrement Operators

Increment and Decrement

Increase or decrease a value by one, respectively.

- the most common operation in all of programming is to increment or decrement an unsigned integer:
  `x += 1;` and `x -= 1;`
- so common that there are operators just for this:
  the increment operator ++ and the decrement operator --
- they are unary operators
- assuming x is a variable with a value, x++ causes x to have the next larger value
- ```
  unsigned x = 5;
  x++; // now x is 6
  x--; // now x is 5 again
  ```
  ```

Prefix and Postfix

- each operator comes in two forms, a prefix form and a postfix form
- ```
  unsigned x = 5;
  x++; // now x is 6
  ++x; // now x is 7
  ```
- in the example above, both prefix and postfix forms do the same thing
- but there is a huge difference in how they actually operate
  ```

Prefix and Postfix

```
unsigned number = 5;

cout << number++ << endl;
cout << ++number << endl;
```

- the difference is in when they operate
- in each case, number is being used in an expression
- and also being incremented

Prefix and Postfix

```
unsigned number = 5;

cout << number++ << endl;
cout << ++number << endl;
```

- the difference is in when they operate
- in each case, number is being used in an expression
- and also being incremented

- in the prefix form, the increment happens first
- in the postfix form, the increment happens last
- thus the output from the above lines of code is:

Prefix and Postfix

```
unsigned number = 5;

cout << number++ << endl;
cout << ++number << endl;
```

- the difference is in when they operate
- in each case, number is being used in an expression
- and also being incremented

- in the prefix form, the increment happens first
- in the postfix form, the increment happens last
- thus the output from the above lines of code is:
  5
  7

Prefix and Postfix

- another example

```
unsigned foo = 6;
unsigned bar;

bar = foo++;
unsigned quux = --foo;
```

- after this code runs
- bar has the value
- quux has the value

Prefix and Postfix

- another example

```
unsigned foo = 6;
unsigned bar;

bar = foo++;
unsigned quux = --foo;
```

- after this code runs
- bar has the value 6
- quux has the value

Prefix and Postfix

- another example

```
unsigned foo = 6;
unsigned bar;

bar = foo++;
unsigned quux = --foo;
```

- after this code runs
- bar has the value 6
- quux has the value 6

Increment and Decrement with Floating Point

- unary increment and decrement also work with floating point value
- ```
  double x = 2.25;
  x++; // now x is 3.25
  ```
- but they are almost never used this way (why?)
  ```

Using Increment and Decrement

- the following code snippets are completely legal:

```
x *= y / ++z;
if (a++ > 10)
```

- but they are very dangerous
- they are confusing and hard to figure out
- never do this!
- a best practice of programming is that each statement or expression should do only one thing
- these are doing two things at once
- they should be split into separate statements

Non-Linear Control Flow II

- the focus of chapter 4 was non-linear program flow
- using the concept of the if statement
- technically called branching

- the focus of chapter 5 is a different form of non-linear program flow: looping

Non-Linear Control Flow II

- the focus of chapter 4 was non-linear program flow
- using the concept of the if statement
- technically called branching

- the focus of chapter 5 is a different form of non-linear program flow: looping

- C++ has three looping constructs
  1. the while loop
  2. the do-while loop
  3. two versions of the for loop

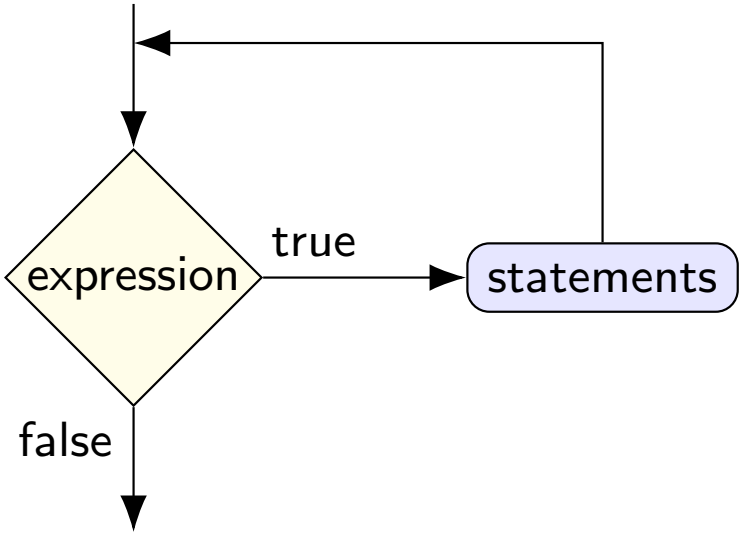
The while Loop

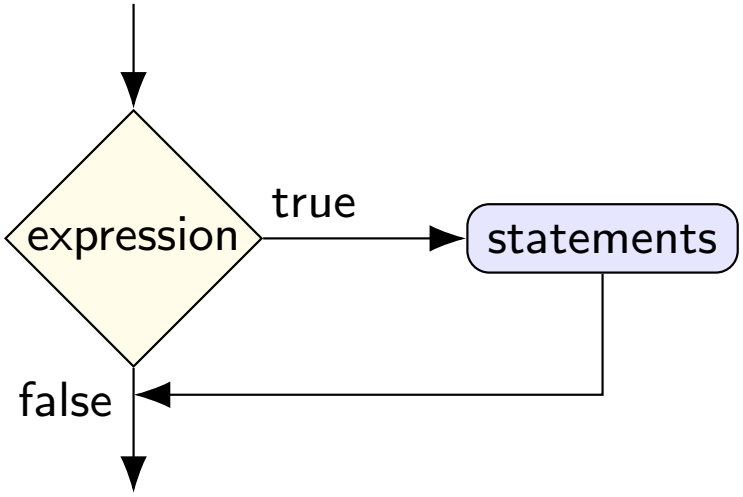
- the while loop is extremely similar in both structure and action to the simple if statement
- the only difference in structure is one word

```
while (expression)
{
  statement;
  statement;
  ...
}
```

- all the rules of style and the common mistakes are identical to those for the if statement

Flowchart





- the flowchart of while is almost identical to simple if
- but there is one crucial difference
- simple if sends the program forward around a detour
- while sends the program around a detour and then backwards
- this allows for the program to repeat a block of statements more than once

- looping aka iteration aka repetition

```
1  // A simple while loop from
2  // Gaddis Program 5-3 page 238
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8    unsigned number = 0;
9
10   while (number < 5)
11   {
12     cout << "Hello" << endl;
13     number++;
14   }
15   cout << "Done" << endl;
16   return 0;
17 }
```

- line 8: the loop control variable is initialized
- line 10: the loop control variable is tested
- line 13: the loop control variable is incremented
- lines 12 and 13 repeat while the Boolean expression is true
- when the Boolean expression becomes false, control jumps to line 15
- Hello is printed exactly 5 times

Pretest

- the while loop is a pretest loop
- the Boolean expression is tested each time before the loop body is executed
- the loop body may be executed zero times
- the loop body may be executed exactly one time
- the loop body may be executed many times

Pretest

- the while loop is a pretest loop
- the Boolean expression is tested each time before the loop body is executed
- the loop body may be executed zero times
- the loop body may be executed exactly one time
- the loop body may be executed many times

- in fact, the loop body may be executed infinitely many times
- this is a bad thing — an infinite loop — usually caused by
  - incorrect logic
  - failure to correctly initialize the loop control variable before the loop
  - failure to modify the loop control variable

Counters and Accumulators

- look at program count_accumulate.cpp
- the variable `count_of_scores` is a counter
- it keeps track of how many times the loop body executes
- it is incremented each time the body executes

Counters and Accumulators

- look at program count_accumulate.cpp
- the variable `count_of_scores` is a counter
- it keeps track of how many times the loop body executes
- it is incremented each time the body executes

- the variable `sum_of_scores` is an accumulator
- it keeps a running sum of the non-zero scores that were entered
- it is added to each time the body executes

Counters and Accumulators

- look at program count_accumulate.cpp
- the variable `count_of_scores` is a counter
- it keeps track of how many times the loop body executes
- it is incremented each time the body executes

- the variable `sum_of_scores` is an accumulator
- it keeps a running sum of the non-zero scores that were entered
- it is added to each time the body executes

- both variables must be initialized before the loop begins
- the counter variable may or may not be the loop control variable (here it is not)

The do-while Loop

- the second loop construct of C++ is the do-while loop
- its form is below — note the semicolon!

```
do
{
   statement;
   statement;
   ...
} while (expression);
```

- this is a posttest loop
- its Boolean expression is tested after the loop body executes
- it is guaranteed that the loop body will execute at least once
- look at program count_accumulate_do_while.cpp, which is the previous program converted to use a do-while loop

Controlling a Loop With a Flag

```
1   bool done = false;
2
3   while (!done)
4   {
5     cout << "Enter a plan: ";
6     char plan;
7     cin >> plan;
8
9     if (plan == 'A')
10       // stuff for plan A ...
11    else if (plan == 'B')
12       // stuff for plan B ...
13    else if (plan == 'C')
14       // stuff for plan C ...
15    else
16    {
17      done = true;
18    }
19  }
```

- it is extremely common to control a while loop with a Boolean flag
- the flag is initialized to false
- when the loop exit condition is recognized, the flag is set to true

Input Validation

- a common use for a while or a do-while loop is input validation

```
unsigned act_score;
bool valid_score;

do
{
  cout << "Please enter an ACT score: ";
  cin >> act_score;
  valid_score = act_score > 0 && act_score <= 36;
  if (!valid_score)
  {
    cout << "Invalid score. Try again" << endl;
  }
} while (!valid_score);

... now act_score is valid, so use it
```