Ch5: Loops and Files

- Increment and Decrement Operators
- Loops
  - The while Loop
  - The do-while Loop
  - The for Loop
- Nested Loops
- Ending Loop Iterations Early
- Loop Examples
- Data Files for Storage

Increment and Decrement Operators

- unary operators similar to compound assignment operators
- ++ increment
  - add 1 to operand
- -- decrement
  - subtract 1 from operand

```
int i = 10;
i = i + 1;
i += 1;
++i;
```

i = 11

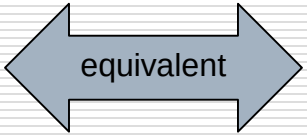
```
int i = 10;
i = i - 1;
i -= 1;
--i;
```

i = 9

Increment and Decrement Operators

Operator	Mode	Description
++var	prefix	The expression (++var) increments var by 1 and then evaluates the expression with the new value.
var++	postfix	The expression (var++) evaluates the expression and then increments var by 1.
--var	prefix	The expression (--var) decrements var by 1 and then evaluates the expression with the new value.
var--	postfix	The expression (var--) evaluates the expression and then decrements var by 1.

```
int i = 10;
int newNum = 10 * ++i;
```

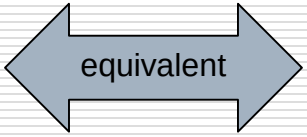



```
int i = 10;
i = i + 1;
int newNum = 10 * i;
```

Increment and Decrement Operators

Operator	Mode	Description
++var	prefix	The expression (++var) increments var by 1 and then evaluates the expression with the new value.
var++	postfix	The expression (var++) evaluates the expression and then increments var by 1.
--var	prefix	The expression (--var) decrements var by 1 and then evaluates the expression with the new value.
var--	postfix	The expression (var--) evaluates the expression and then decrements var by 1.

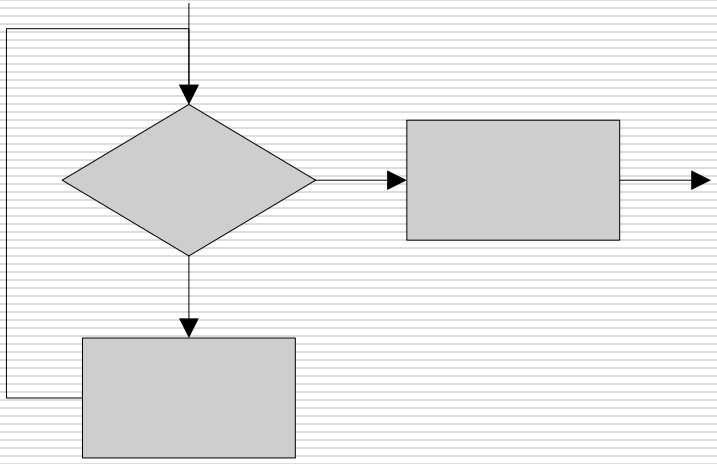
```
int i = 10;
int newNum = 10 * i++;
```




```
int i = 10;
int newNum = 10 * i;
i = i + 1;
```

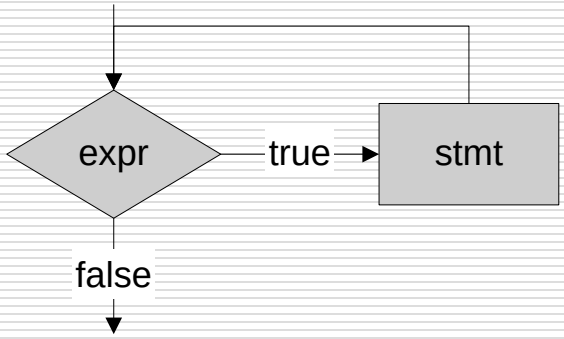
Loops

- repetition flow of control
  - statement, or group of statements, that repeat
- types
  - while
  - do-while
  - for



The while Loop

- **expr** must have arithmetic or pointer type
- **stmt** executed repeatedly as long as expr evaluates to nonzero (TRUE)
- **stmt** can also be a group of statements enclosed within braces { }




```
while (expr)
  stmt;
```

```
while (expr)
{
  stmt1;
  stmt2;
}
```

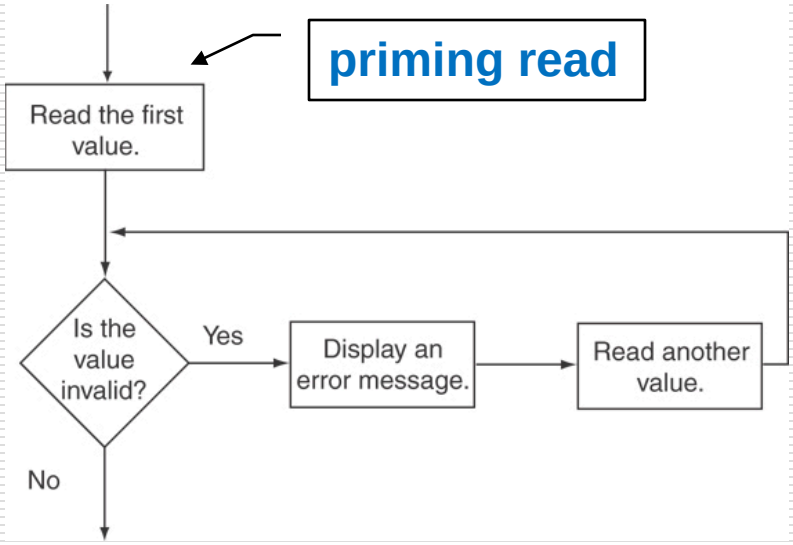
```
while (expr);
  stmt;
```

```
while (expr)
  stmt1;
  stmt2;
```

The while Loop

- test of **expr** happens *before* **stmt** execution
  - pre-test loop
  - possible to have no repetition
- while loop *should* contains a way to terminate as part of **expr**
  - infinite loop has no way of stopping

Example: Input Validation Program 5-5



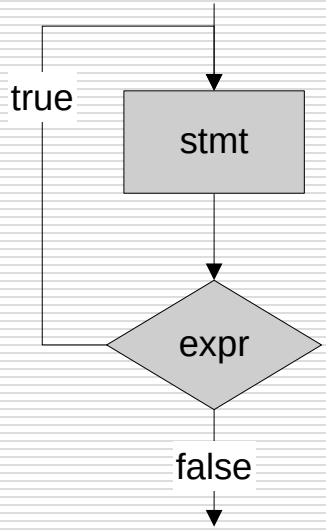

```
36      // Get the number of players available.
37      cout << "How many players are available? ";
38      cin >> players;
39
40      // Validate the input.
41      while (players <= 0)
42      {
43          // Get the input again.
44          cout << "Please enter 0 or greater: ";
45          cin >> players;
46      }
```

The do-while Loop

- **expr** must have arithmetic or pointer type
- **stmt** executed repeatedly as long as expression evaluates to nonzero (TRUE)
- **stmt** can also be a group of statements enclosed within braces
- Semicolon needed at end

```
do
{
  stmt1;
  stmt2;
} while (expr);
```

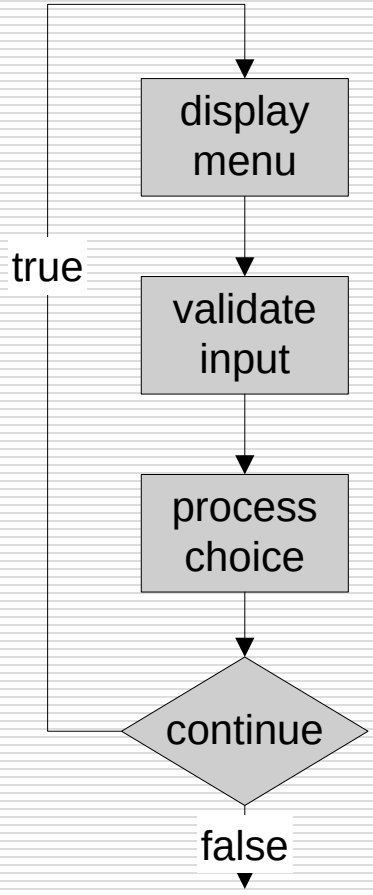
```
do
  stmt;
while (expr);
```



The do-while Loop

- test of **expr** happens *after* **stmt** execution
  - post-test loop
  - must have at least one **stmt** execution
- do-while loop *should* contains a way to terminate as part of **expr**

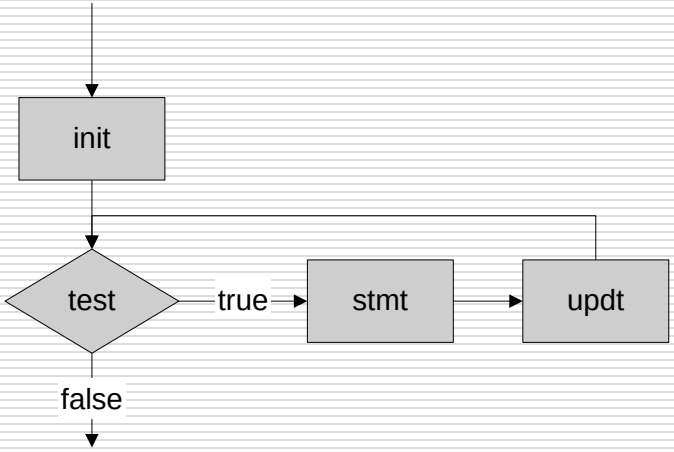
Example: Menu Program 5-8




```
do
{
  // display menu
  cout << "...";
  // validate choice
  while (...)
  // process choice
  switch(choice)
  ...
} while (choice != QUIT_CHOICE);
```

The for Loop

- **test** must have arithmetic or pointer type
- **init** is evaluated once, at the beginning
  - can have multiple initializations separated by commas
- **stmt** is executed repeatedly so long as **test** evaluates to nonzero (TRUE)




```
for (init; test; updt)
    stmt;
```

The for Loop

- **updt** is evaluated every time after **stmt** execution and before test_expr
  - can have multiple updates separated by commas
- **stmt** can also be a group of statements enclosed within braces
- test of **test** happens before **stmt** execution
  - pre-test loop
  - possible to have no iteration

```
for (init; test; updt)
{
    stmt1;
    stmt2;
}
```

Example: For Loop (Figures 5-6 and 5-7)

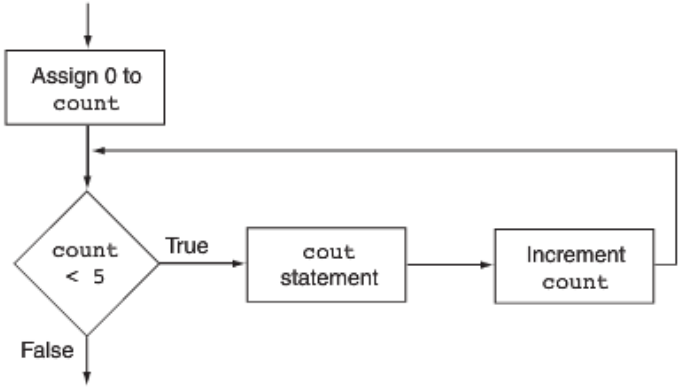
Step 1: Perform the initialization expression.

Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

```
for (count = 1; count <= 5; count++)
   cout << "Hello" << endl;
```

Step 3: Execute the body of the loop.

Step 4: Perform the update expression, then go back to Step 2.




```
Hello
Hello
Hello
Hello
Hello
```

Loop Summary

	while	do-while	for
pre-test *(may not iterate at all)*	X	O	X
post-test *(must iterate at least once)*	O	X	O
conditionally controlled	X	X	O
counter controlled	O	O	X

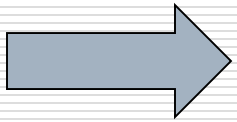
Nested Loops

- Loops can be nested
  - no requirement on the types of loops that can be nested
  - inner loop goes through all iterations for each iteration of an outer loop
    - example:
      - inner loop = 3
      - outer loop = 2
      - total loops = 3 * 2
  - use indentation to help with viewing nested loops
    - indentation will **not** enforce nesting

Nested Loops Example

- Loop through a day of 24 hours
  - Loop through an hour of 60 minutes
    - Loop through a minute of 60 seconds

```
for (int hours = 0; hours < 24; hours++)
{
     for (int minutes = 0; minutes < 60; minutes++)
     {
          for (int seconds = 0; seconds < 60; seconds++)
          {
               cout << setw(2) << hours << ":";
               cout << setw(2) << minutes << ":";
               cout << setw(2) << seconds << endl;
          }
     }
}
```

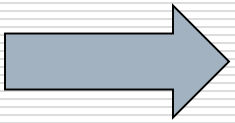


```
00:00:00
00:00:01
00:00:02
   .
   .
   .
23:59:59
```

Breaking Out of a Loop

- Break statement
  - Causes loop to terminate early
  - May appear only within a looping statement (While, Do-While, For) or a Switch statement
  - When used in nested loops, only affects loop in which it is placed

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++){
  sum = sum + i;
  if (i == 2)
      break;
}
cout << "sum is " << sum;
```



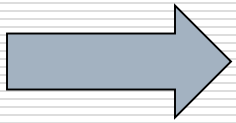
sum of numbers 1 to 2

sum is 3

Continue With Next Loop Iteration

- Continue statement
  - Stops current loop iteration and begins next
  - May appear only within a looping statement (While, Do-While, For)

```
int sum, i;
for (sum = 0, i = 1; i < 5; i++){
  if (i % 2 == 1)
    continue;
  sum = sum + i;
}
cout << “sum is “ << sum;
```

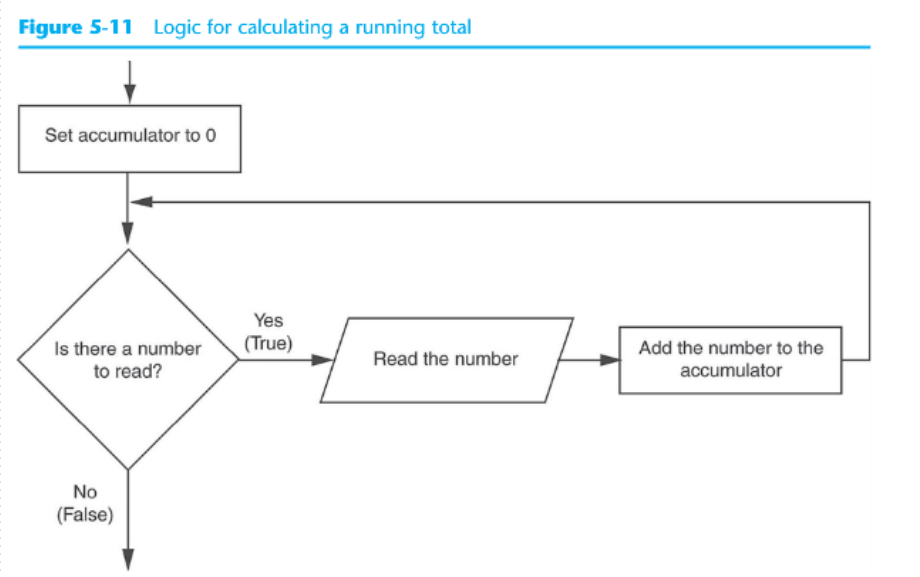


sum of even numbers 1 to 4

sum is 6

Using a Loop: Keeping a Running Total

- **running total** → accumulated sum of numbers from each loop repetition
- **accumulator** → variable that holds running total



Using a Loop: Keeping a Running Total

- Program 5-12
  - accumulate sales for input number of days

```
// Get the sales for each day and accumulate a total.
for (int count = 1; count <= days; count++)
{
   double sales;
   cout << "Enter the sales for day " << count << ": ";
   cin >> sales;
   total += sales;  // Accumulate the running total.
}
```

Using a Loop: Sentinel

- **sentinel** → value in a list of values that indicates end of data
  - Used to terminate input when user *may not know* how many values will be entered
  - Special value that cannot be confused with a valid value, *e.g.*, -999 for a test score

Using a Loop: Sentinel

- Example: Program 5-13
  - accumulate soccer points for series of games
  - -1 for end of input

Program 5-13

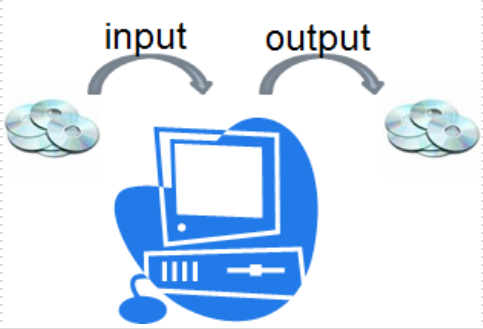
```
1  // This program calculates the total number of points a
2  // soccer team has earned over a series of games. The user
3  // enters a series of point values, then -1 when finished.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9       int game = 1,    // Game counter
10          points,      // To hold a number of points
11          total = 0;   // Accumulator
12
13      cout << "Enter the number of points your team has earned\n";
14      cout << "so far in the season, then enter -1 when finished.\n\n";
15      cout << "Enter the points for game " << game << ": ";
16      cin >> points;
17
18      while (points != -1)
19      {
20           total += points;
21           game++;
22           cout << "Enter the points for game " << game << ": ";
23           cin >> points;
24      }
25      cout << "\nThe total points are " << total << endl;
26      return 0;
27 }
```

Data Files for Storage

- Types of files
  - Binary → encoded for computer use
  - Text → encoded in ASCII or Unicode
- Files on storage are identified by filename
  - Filename extension indicates file type
- Files used by programs are identified by file stream objects

Data Files for Storage

- Open the file:
  - Output → means creating and preparing it for writing data
  - Input → means opening a file and preparing it for reading data
- Process the file:
  - Output → Writes data to file
  - Input → Reads data from file
- Close the file:
  - Flush buffer contents (*for write*); disconnects from application



Data Files for Storage

- Using file streams
  - `#include <fstream>`

Table 5-1

File Stream Data Type	Description
`ofstream`	Output file stream. You create an object of this data type when you want to create a file and write data to it.
`ifstream`	Input file stream. You create an object of this data type when you want to open an existing file and read data from it.
`fstream`	File stream. Objects of this data type can be used to open files for reading, writing, or both.

Data Files for Storage

- Using file streams for data input
  - Creating and opening file for input

```
ifstream inputFile;
inputFile.open("Customers.txt");
inputFile.open("C:\\temp\\Customers.txt");
```

  - Reading from file
    - Read pointer advances as each item is read

```
inputFile >> name >> gpa;
```

  - Closing a file

```
inputFile.close();
```

Data Files for Storage

- Using file streams for output
  - Creating and opening file for output

```
ofstream outputFile;
outputFile.open("C:\\logfile.txt");
```

  - Writing to file

```
outputFile << "Hello " << name << endl;
```

  - Closing a file

```
outputFile.close();
```

Data Files for Storage

- Testing for open errors before read/write
  - Using stream status

```
if (!inputFile) {
   cout << “Error!” << endl;
    exit(-1);
}
```

  - Using stream function

```
if (inputFile.fail()) {
   cout << “Error!” << endl;
    exit(-1);
}
```

Data Files for Storage

- Looping to read from file
  - Using stream status

```
while (inputFile >> name) {
    cout << name << endl;
}
```

  - Using stream function

```
while (!inputFile.eof()) {
    inputFile >> name;
     cout << name << endl;
}
```