A Repetitive Task

- Get six numbers from the user
- Add them all together
- Print the result to the screen

- Requires:
  - Six variables to hold input (e.g. num1, num2, num3, etc.)
  - Six input statements

- Repetitive and inefficient
  - Worse, what if it was 1000 numbers (perhaps from a file rather than from a user)?

Repetitive Execution

- A better solution:
  - Tell the computer to *iterate*, to do the same thing six times
    - Get a number from the user
    - Add it to a running total
  - Then print the result

- Requires:
  - Two variables (input and total)
  - One input statement for each *iteration*

- Pretty much any real program involves iteration

Conditional Execution

- `if…else` is used to control conditional execution

```
if( condition )
{
   // do some stuff only if condition is true

}
```

- Conditional execution happens 0 or 1 time
- Condition is a logical expression
  - Evaluates to `true` (`1`) or `false` (`0`)
  - Can be a literal, a variable, a function or an expression

Iterative Execution

- `while` used to control iterative execution (looping)

```
while( condition )
{
   // do some stuff repeatedly as long
   //  as condition is true
}
```

- Iterative execution happens 0 or more times
- Condition is a logical expression
  - Evaluates to `true` (`1`) or `false` (`0`)
  - Can be a literal, a variable, a function or an expression

`while` Looping (Repetition) Structure

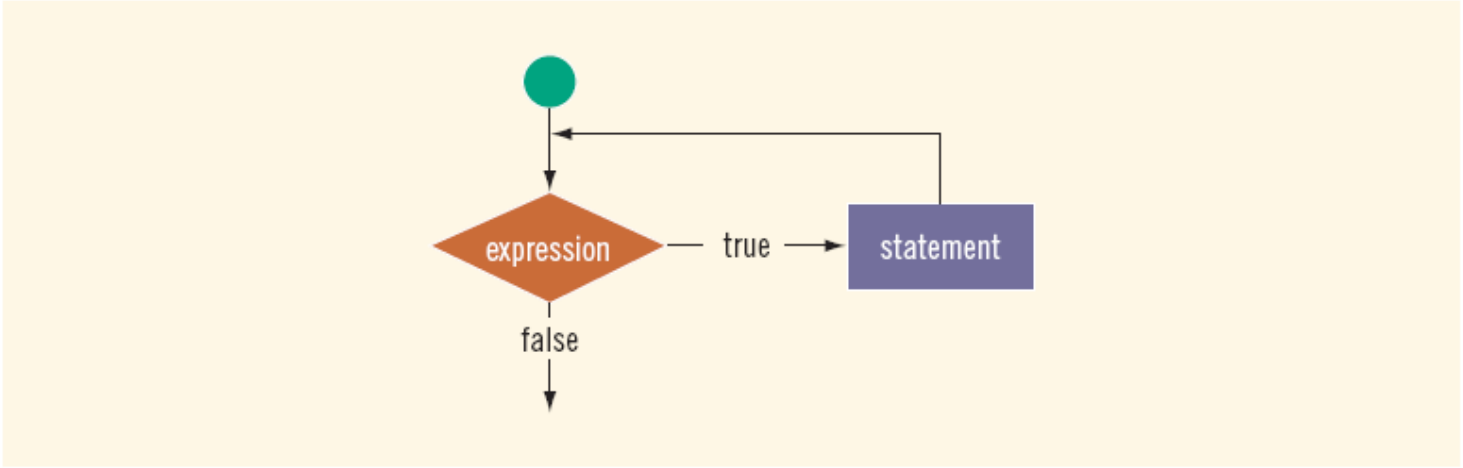

FIGURE 5-1 `while` loop

- Infinite loop: continues to execute endlessly
  - Avoided by including statements in loop body that assure exit condition is eventually `false`

Elements of an Iterative Statement

- There are three key parts to an iterative statement:
  - Initialization (before the loop)
    - What are the values of variables set to before the loop starts?
  - Condition (the while condition)
    - When does the loop quit?
  - Update (in the body of the loop)
    - How are those values changed in the loop?

Example Case: Counter Loop

- Use a `while` loop to do something a predetermined number of times
  1. Initialization (before the loop)
     - Declare a variable to use as a counter
     - Assign it the value to start counting at
  2. Condition (the while condition)
     - Check to see if the counter value has reached the target count
       - If it has, quit the loop
  3. Update (in the body of the loop)
     - Increment or decrement the counter value
     - Do the other repetitive tasks as well
  4. Steps 2 and 3 repeat

while Looping (Repetition) Structure (continued)

EXAMPLE 5-1

Consider the following C++ program segment:

```
i = 0;                          //Line 1

while (i <= 20)                 //Line 2
{
    cout << i << " ";           //Line 3
    i = i + 5;                  //Line 4
}

cout << endl;
```

Sample Run:

```
0 5 10 15 20
```

The Rest of the Loop

- The body of a counter loop must update the counter
  - But it also does whatever repetitive tasks you are trying to accomplish
    - Update other variables
    - Get input
    - Print output
    - Etc…

Exercise

```
int i = 0, j = 0;
while( i < 5 )
{
    j = j + 10;
    i++;
}
```

- Initialization:
  - Both `i` and `j` are set to 0 before the loop
- Update:
  - Both `i` and `j` are assigned new values in the body of the loop
- Condition:
  - The loop stops based on the value of `i`

Exercise

- What are the values of `i`, `j` at the beginning of each iteration of this loop?

```
int i = 0, j = 0;
while( i < 5 )
{
     j = j + 10;
     i++;
}
```

Iteration	i	j
first		
second		
third		
…		

Exercise

- Write the output of the following loops:

a:
```
int i = 0;
while ( i < 5 )
{
   cout << i << " ";
   cout << endl;
   i++;
}
```

b:
```
int i = 0;
while ( i < 5 )
{
   i++;
   cout << i << " ";
}
cout << endl;
```

Example Case: Input Condition

- Use a while loop to do something until input (user, file, etc) tells us to stop
  1. Initialization (before the loop)
     - Declare a variable to hold the user input
     - Assign it an initial value
  2. Condition (the while condition)
     - Check to see if the input variable matches the target value
       - If it does, quit the loop
  3. Update (in the body of the loop)
     - Get new input
  4. Steps 2 and 3 repeat

Exercise

- Write a while loop that:
  - Asks the user to enter a number
  - If the number is -99 it quits
  - Otherwise, adds that number to a running total
  - And repeat

  - Initialization
    - Variables to hold user input and the accumulated total
      - Initial values?
  - Condition
    - Is the latest input equal to -99?
  - Update
    - Add the last number to the total
    - Get the next user input