Chapter 9
Repetition

Repetition in C++

The final control/logic structure is repetition:

Repetition – repeating a block of code until a condition is met

There are three repetition statements available in C++. The *while*, the *do while* and the *for*. The `while` statement is controlled by a condition. The condition is tested at the top of the loop and if the condition evaluates to `true`, the loop is entered and if the condition evaluates to `false`, the loop is bypassed. A `while` loop is an *event-controlled* loop. Because the condition is tested at the top, a `while` loop is said to be a *pre-test* loop.

Event-controlled loop – a loop that terminates based on a condition and a *sentinel* value – this loop executes an unspecified number of times

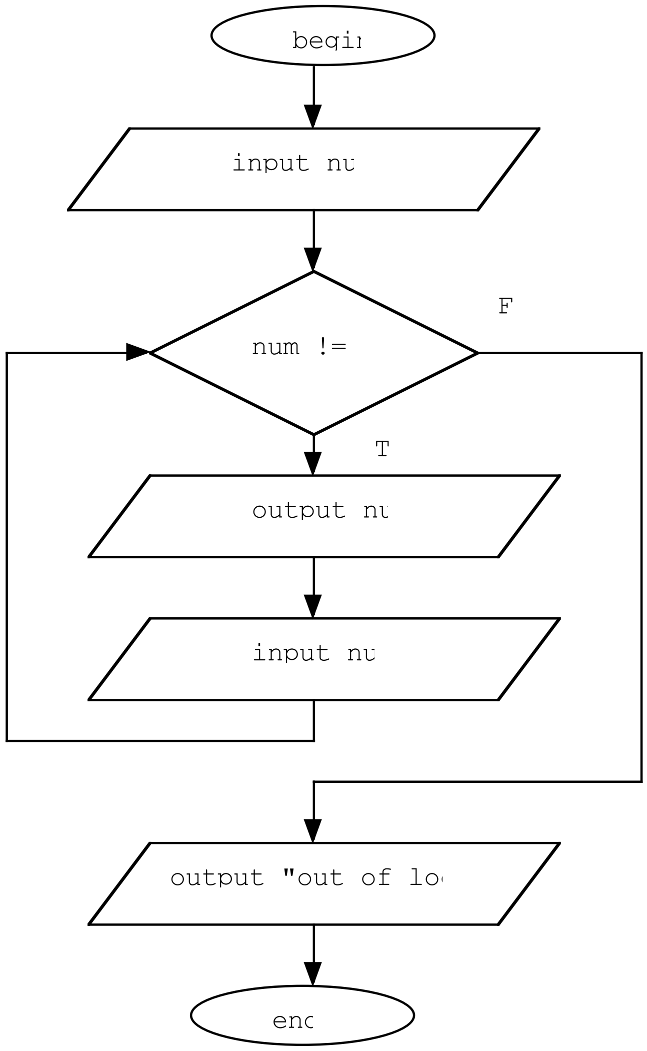
Sentinel value – a special value for the *loop control variable* that signifies that the loop should terminate

Loop Control Variable (LCV) – the variable whose value determines whether the loop should be terminated

GENERAL FORM of a *while* statement

```
while (Expression)
{
  statement;
}
```

Example: This program will obtain integers from the user. It will continue to read and "echo back" integers while the integer entered is not a zero.



C++ code for loop only:

```
cout << "Enter an integer – 0 to exit: ";
cin >> num;
while (num != 0)
{
  cout << "Value entered was " << num << endl;
  cout << "Enter another integer – 0 to exit: ";
  cin >> num;
}
cout << "Out of loop" << endl;
```

In the example on the previous page, `num` is the loop control variable. Notice that a value is read for `num` before the loop, the value of `num` is checked in the condition and finally, a new value is read for `num` before the end of the loop (before the program returns to the condition). These three steps are essential for looping structures.

1.) **INITIALIZE** the loop control variable

2.) **CHECK** the loop control variable (compare with the sentinel value)

3.) **CHANGE** the loop control variable

The code inside the French braces (`{ }`) is called the body of the loop. This code is executed as long as the condition is `true` and is skipped when the condition is `false`.

A common function performed by looping statements is *accumulation*. There are two types, of accumulators, *counters* and *running totals*.

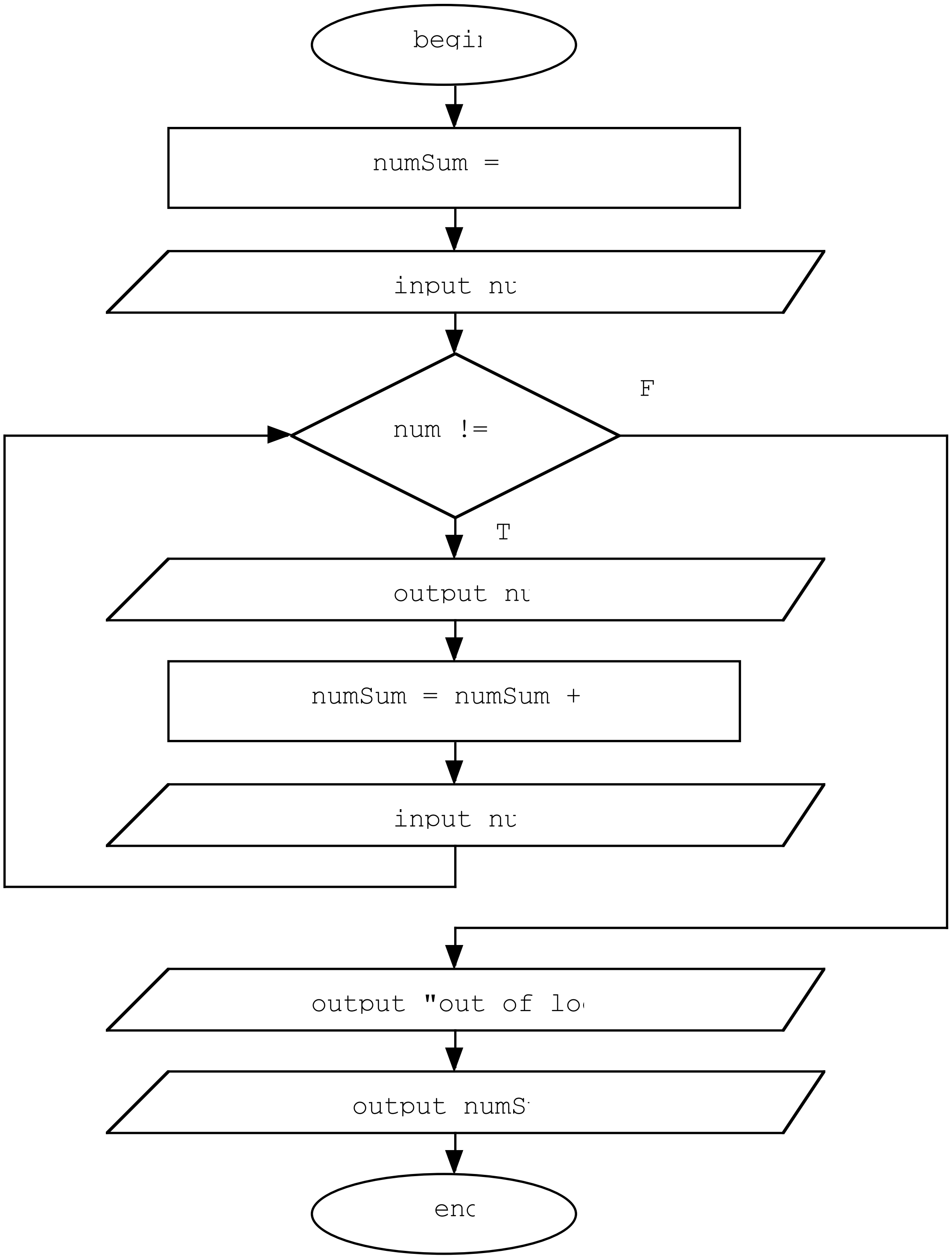
Counter – an accumulator that increments or decrements by one

Running Total – an accumulator that increments by a value being totaled

Example:

To calculate the average age for a group of people we need two pieces of information, the total number of people and the sum of all of their ages. We will use a counter to count the number of people and a running total to add up all of their ages. The average is simply the running total divided by the counter.

Let's modify the previous program to sum the integers being read and output the sum at the end of the program. Before an accumulator can be used it must be ***initialized*** (given an appropriate starting value).


begi
numSum =
input nu
num !=
F
T
output nu
numSum = numSum +
input nu
output "out of lo
output numS
enc

C++ code – complete program:

```
// This program sums a stream of integers
// and demonstrates the use of the while statement

#include <iostream>
using namespace std;

void main(void)
{
  // declaration section with data table
  int num;     // integer value - INPUT
  int numSum;  // running total of integers - CALC & OUTPUT

  // initialize the running total
  numSum = 0;

  // initialize the LCV by reading first integer
  cout << "Enter an integer - 0 to exit: ";
  cin >> num;

  // check the LCV against the sentinel value 0
  while (num != 0)
  {
     cout << "Value entered was " << num << endl;

     // add integer to running total before reading a new value
     numSum = numSum + num;

     // change the LCV by reading a new value
     cout << "Enter another integer - 0 to exit: ";
     cin >> num;
   }

   // output message and running total
   cout << "\n\nOut of loop\n\n";
   cout << "The sum of the integers is " << numSum << endl;
}
```

Note the spacing and indentation. Specifically, the body of the loop is indented two spaces so that it is easy for the reader to distinguish the loop from the rest of the program.

A further modification to this program would be to include the average of the numbers as well as the sum. To calculate the average we need not only the sum of the numbers but also the number of numbers entered. The program requires the addition of two variables, a counter we shall call numCount and a place to store the average we shall call numAvg. Draw the modified flowchart below to include these new variables.

```
// This program sums a stream of integers
// and demonstrates use of the while statement

#include <iostream>
using namespace std;

void main(void)
{
  // declaration section with data table
  int num;     // integer valid - INPUT
  int numSum;  // running total of integers - CALC & OUTPUT
  int numCount;  // counts # of integers entered - CALC & OUTPUT
  float numAvg;  // average of numbers entered - CALC & OUTPUT

  // initialize the running total and counter
  numSum = 0;
  numCount = 0;

  // initialize the LCV by reading first integer
  cout << "Enter an integer - 0 to exit: ";
  cin >> num;

  // check the LCV against the sentinel value 0
  while (num != 0)
  {
    cout << "Value entered was " << num << endl;

    // add integer to running total and increment counter before
    // reading a new value
    numSum = numSum + num;
    numCount = numCount + 1;

    // change the LCV by reading a new value
    cout << "Enter another integer - 0 to exit: ";
    cin >> num;
   }

   // calculate the average
   numAvg = numSum / numCount;

   // output message and calculated values
   cout << "\n\nOut of loop\n\n";
   cout << "The sum of the integers is " << numSum << endl;
   cout << "The number of integers entered is " << numCount << endl;
   cout << "The average of the integers is " << numAvg << endl;
}
```

While Loop Exercises

1. Draw the flowchart for a program that will read an <u>unknown</u> number of ages from the keyboard and output the average of the ages. Variables include age (holds the current age entered), ageSum (the sum of all the ages), ageCount (the number of ages) and ageAvg (the average of the ages). Which one should be used to control the loop?

2. Modify the flowchart in question 1 to also output the largest age. This requires a combination of repetition and selection.

3. Draw the flowchart to match the instructions below:
    - initialize 3 counters called pCount, nCount, zCount to zero
    - read a value from the keyboard into an integer variable called theNum
    - while theNum does not equal the value -999 do the following:
        - if the value of theNum is greater than zero increment the variable called pCount, otherwise if the value of theNum is equal to zero increment the variable called zCount, otherwise increment the variable called nCount.
        - read another value for theNum
    - output the values for the three counters

    What task do the counters pCount, zCount and nCount perform in this algorithm? What term is used to describe the value -999?

4. Show the EXACT output from the partial code sample below.

```
void main(void)
{
  int num1;
  int num2;
  int num3;

  num1 = 2;
  num2 = num1 * 2;
  num3 = num1 + num2;
  while (num1 <= 15)
  {
    cout << num1 << " " << num2 << " " << num3 << endl;
    num1 = num1 + num2;
    num2 = num2 + num3;
    cout << num3 << " " << num2 << " " << num1 << endl;
  }
  cout << "Out of loop ";
}
```

5. Show the EXACT output from the partial code sample below.

```
void main(void)
{
  int num1;
  int num2;
  int num3;

  num1 = 2;
  num2 = num1 * 2;
  num3 = num1 + num2;
  while (num1 <= 15)
  {
    cout << num1 << " " << num2 << " " << num3 << endl;
    if (num1 < 6)
    {
      num1 = num1 + num2;
    }
    else
    {
      num1 = num1 + num3;
    }
    num2 = num2 + num3;
    cout << num3 << " " << num2 << " " << num1 << endl;
  }
  cout << "Out of loop ";
}
```

```
void main(void)
{
  int loopCount;   // the loop control variable
  inr num;    // integer to be summed - INPUT
  int numSum; // sum of the numbers - CALC & OUTPUT

  // initialize the LCV and the running total
  numSum = 0;
  loopCount = 1;
  while (loopCount <= 10)
  {
    // obtain number from user
    cout << "Enter an integer: ";
    cin >> num;

    // increment the accumulator
    numSum = numSum + num;

    // change the LCV
    loopCount = loopCount + 1;
  }

  // output the sum
  cout << "The sum of the " << loopCount -1 << " numbers is " <<
    numSum << endl;
}
```

- How does the loop above differ from the while loops in the previous examples and exercises?

- What task is performed by variable loopCount?
-
- Is loopCount an accumulator?

- Is this an interactive program?

- Does the user have any control over variable loopCount?

- Why was the value   loopCount - 1   output at the end of the program rather than the value loopCount? How could the code be modified so the value of loopCount would be accurate when the loop ends?

The C++ *for* Statement

When loops are to be executed a specific number of times the while loop is not always a good choice. The writing of counter-controlled loops is simplified by the use of the *for* statement.

GENERAL FORM of a *for* statement

```
for (InitStatement; Expression1; Expression2)
{
   statement;
}
```

The actual writing of the `for` statement will not match the general form exactly. A little confusing, but just a matter of punctuation. For the purposes of CS !A

- InitStatement is an expression statement used to initialize the LCV
- Expression1 is a boolean expression
- Expression2 is typically used to increment or decrement the LCV

Examples:

```
// This loop prints the integers 1 to 50
for (i = 1;  i <= 50;  i = i + 1)
{
  cout << i << endl;
}


// This loop prints the integers 10 down to 1
for (i = 10; i >= 1; i = i - 1)
{
  cout << i << endl;
}
cout << "Blast Off";
```

Following is the integer summing program modified to show the use of the *for* statement.

```
#include <iostream>
using namespace std;

void main(void)
{
  int loopCount; // the loop control variable
  int num;       // integer to be summed - INPUT
  int numSum;    // sum of the numbers - CALC & OUTPUT

  // initialize the running total
  numSum = 0;

  for (loopCount = 1; loopCount <= 10; loopCount = loopCount + 1)
  {
    // obtain number from user
    cout << "Enter an integer: ";
    cin >> num;

    // increment the accumulator
    numSum = numSum + num;
  }

  // output the sum
  cout << "The sum of the " << loopCount -1 << " numbers is "
       << numSum << endl;
}
```

Notice that the essential steps for loop control are all built into the *for* statement itself.

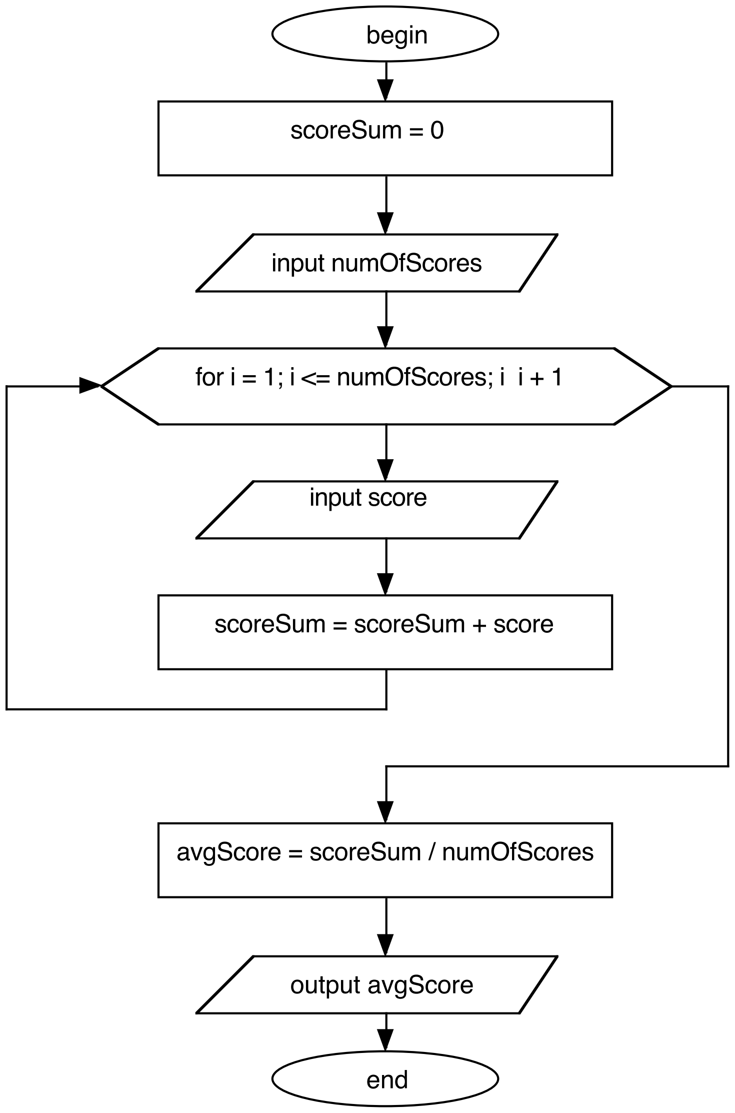
for (loopCount = 1; loopCount <= 10; loopCount = loopCount + 1)

↑ **initialize** ↑ **check** ↑ **change**

Remember that this loop should only be used when the exact number of iterations (repetitions) is known in advance.

Flowcharting the for Loop

Following is the flowchart for a program that will ask the user to enter the number of students that took an exam. The program will read that many exam scores and calculate the average exam grade.


begin
scoreSum = 0
input numOfScores
for i = 1; i <= numOfScores; i i + 1
input score
scoreSum = scoreSum + score
avgScore = scoreSum / numOfScores
output avgScore
end

```
#include <iostream>
using namespace std;

void main(void)
{
  int i;                // the LCV for the FOR loop
  int numOfScores;      // number of scores to average - INPUT
  int score;            // individual exam scores - INPUT
  float scoreSum;       // sum of all scores entered - CALC
  float avgScore;       // average exam score - CALC & OUTPUT


  // initialize the running total
  scoreSum = 0;

  // read the number of exam scores to sum
  cout << "Enter the number of exams: ";
  cin >> numOfScores;

  for (i = 1; i <= numOfScores; i = i + 1)
  {
    // obtain score from user
    cout << "\nEnter exam score: ";
    cin >> score;

    // increment the accumulator
    scoreSum = scoreSum + score;
  }

  // calculate the average score
  avgScore = scoreSum / numOfScores;

  // output the average
  cout << "\n\nThe average of the exam scores is " << avgScore << endl;
}
```

- What task does numOfScores perform?

- Why is the statement i = i + 1 not found in the body of the loop?

- If we sold this program to all the instructors at Saddleback, what aspect of the program would they find the most frustrating from the perspective of a user?

- If this program used a variable called highScore to keep track of the highest exam score, would highScore be an accumulator?

For Loop Exercises

1. Draw the flowchart for a program that displays the square and cube for all integers from 1 to 100.

Number	Square	Cube
1	1	1
2	4	8
3	9	27
etc.		

2. Modify the program on the previous page to output not only the average but also the highest and lowest exam score. Exam scores range from 0 to 100. Give careful thought to initial values for the variables that will keep track of the highest and lowest score.

3. Modify the payroll exercise (if-else) from chapter 8 to calculate a payroll for 7 employees. At the end of the program output the total overtime paid for all employees and the average paycheck for the week. Test your program making certain that at least two employees worked less than 40 hours, at least two more than 40 hours, and at least one that worked exactly 40 hours.

4. You are all familiar with the story about the young man who persuades a mathematically challenged king to pay him what appears to be a small reward for rescuing the king's daughter. He requests that one penny be put on the first square of a chess board, two on the next, four on the next and so on. By the time the chess board is full, the king must come to terms with the fact that the reward is larger than the total wealth of all the civilizations in the history of the world. Design an algorithm to calculate an allowance where the child receives one penny on the first day of the month, two on the next, etc. Assume there are 30 days in the month. Produce a chart that shows the day, the allowance for that day and the total received to date.

Day	Allowance	Total Allowance
1	0.01	0.01
2	0.02	0.03
3	0.04	0.07
4	0.08	0.15
5	0.16	0.31
	etc.	

5. Show the EXACT output from the following code segment:

```
void main(void)
{
    int i;
    int j;
    int k;

    j = 5;
    k = 2*j;
    for (i = j; i < 75; i = i + k)
    {
        cout << i << " " << j + k << endl;
        k = k + 5;
     }
}
```

6. Show the output from the following "nested" *for* loops:

```
void main(void)
{
    int i;
    int j;
    for (i = 1; i <= 5; i = i + 1)
    {
        for (j = 1; j <= 5; j = j + 1)
        {
            cout << i * j << "     ";
        }
        cout << endl;
    }
}
```

7. Show the output from the following code segment:

```
void main(void)
{
    int i;
    int j;

    for (i = 1; i <= 5; i = i + 1)
    {
        for (j = 1; j <= i; j = j + 1)
        {
            cout << "*";
        }
        cout << endl;
    }
}
```

8. Modify the above code so the output produced is as follows:

```
*****
****
***
**
*
```

Name ____

Due Date _____________________

Loan Amortization Table

Using the monthly payment and remaining balance calculations, write a program to produce a loan amortization table. A sample run follows. Be sure your output is formatted **EXACTLY** as shown.

```
Original Loan Amount: 18500
Annual Interest Rate: 7.5
Term of Loan (Years): 4
Monthly Payment: 447.31

Payment #   Interest Paid To Date   Remaining Balance

  1            115.62                 18168.32
  2            229.18                 17834.56
  3            340.64                 17498.71
  4            450.01                 17160.77
  5            557.26                 16820.72
  6            662.39                 16478.54
  7            765.38                 16134.22
  8            866.22                 15787.75
  9            964.89                 15439.11
          .
          .
          .
 44            2943.19                 1761.63
 45            2954.20                 1325.33
 46            2962.48                  886.30
 47            2968.02                  444.53
 48            2970.80                    0.00
```

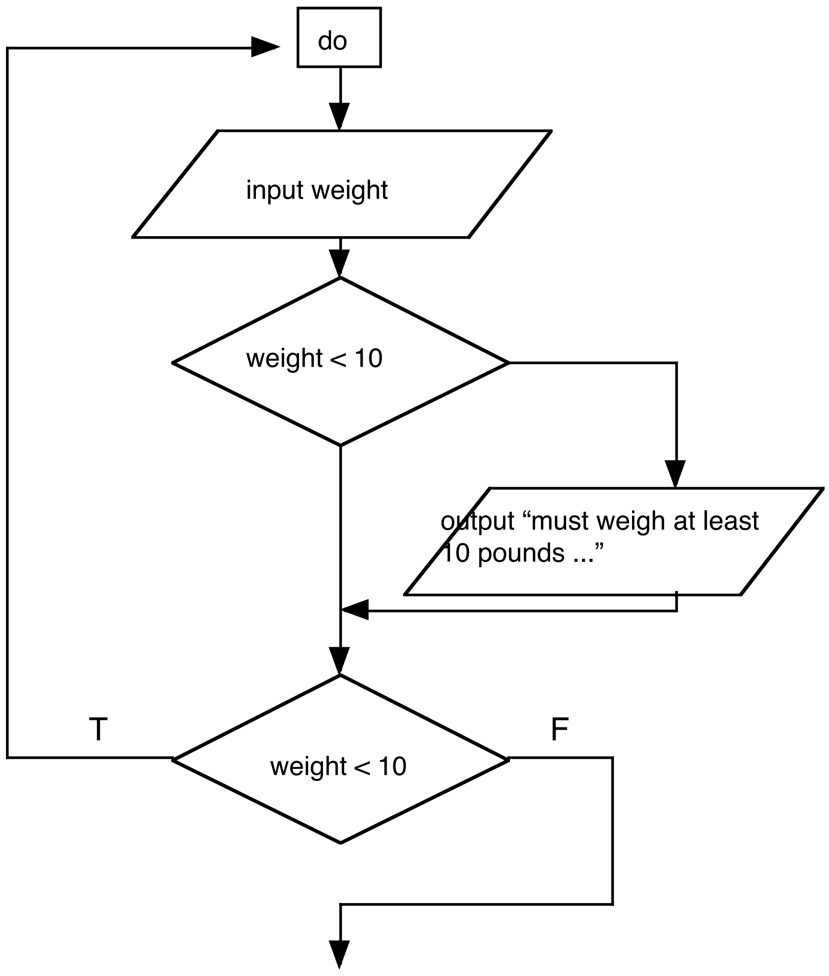
The C++ *do while* Statement

The *do while* statement like the *while* statement is an event-controlled loop. In the *while* loop, the condition is checked at the top of the loop (a pre-test loop) and it is possible that the loop may be skipped completely. In the *do while* loop, the condition is at the bottom. This is referred to as a *post-test* loop and in this type of loop, the body is always executed at least one time. This loop will be chosen only when we know the body must execute at least one time.

GENERAL FORM for the *do while* statement

```
do
{
   statement;
}
while (Expression);
```

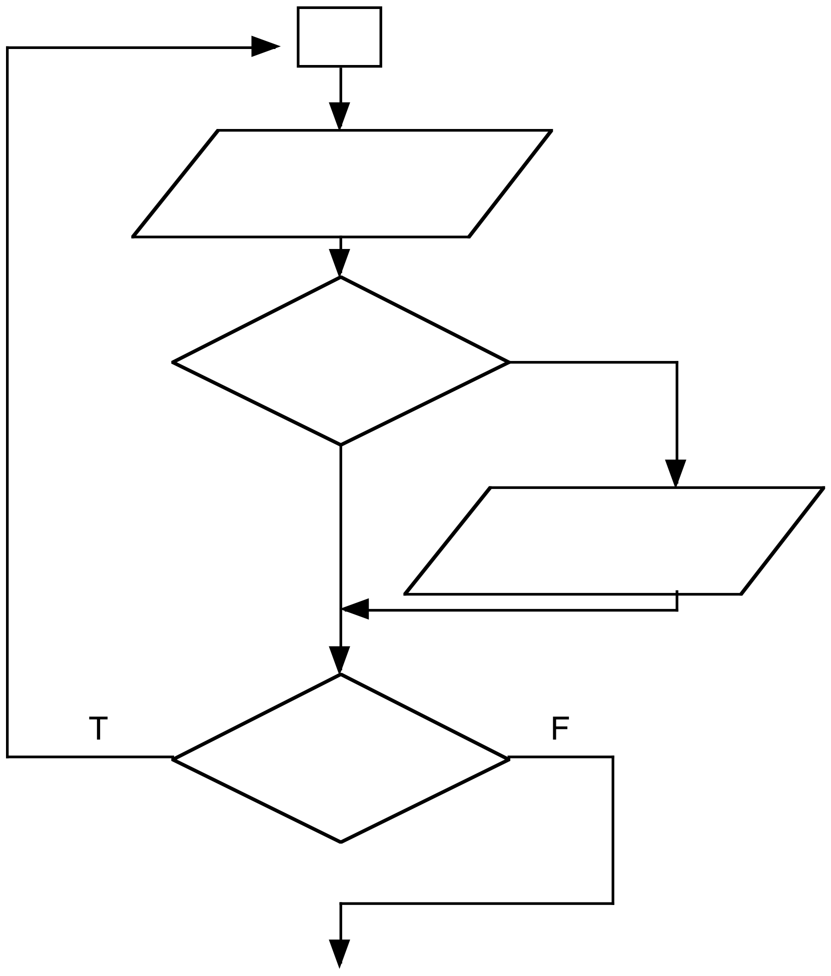
A common use for this loop is in checking user input. On the following page is a loop that will continue to prompt the user for their weight until the weight entered is considered valid. For our purposes, the user must weigh at least 10 pounds.



Note the expression weight < 10 appears twice in the flowchart. The expression is used in an *if* statement to determine whether an error message should be printed and used again in the do loop to determine whether the loop should execute or exit. C++ code follows:

```
do
{
    cout << "Enter your weight: ";
    cin >> weight;
    if(weight < 10)
    {
        cout << "You must weigh at least 10 pounts\n";
    }
}
while(weight < 10);
```

Following is a flowchart to check user input for an integer in the range of 1 to 10. Fill in the appropriate text for each flowchart symbol. The input value will be stored in a variable location called num. Note: The loop should continue to execute when any value less than 0 or greater than 10 is entered. In a previous example (Mission Viejo Electric Company) we used the boolean AND operator (&&) to check a range of numbers looking for a valid month. There is another operator, the OR operator (||) that works as follows. If any part of the expression is true, the entire expression becomes true. Look at the examples on the next page before attempting the flowchart.



Examples:

```
if (x == 1 || x = = 5)
{
    cout << "value of x was 1 or 5"
}
```

If the value entered for x is 3 the condition evaluates as follows:
x ==1 is FALSE
x == 5 is FALSE
FALSE OR (||) FALSE is FALSE (message would not be printed)

If the value entered for x is 5 the condition evaluates as follows:
x ==1 is FALSE
x == 5 is TRUE
FALSE OR (||) TRUE is TRUE (message would be printed)

```
if (ans == 'Y' || ans == 'y')
{
    cout << "the answer is yes"
}
```

Evaluate the above expression using various characters as input.

Write an expression that will be true when a number is NOT in the range of 1 to 10. Use this expression in the flowchart on the previous page..

C++ code

```
#include <iostream>
using namespace std;

void main(void)
{
  int num;

  do
  {
    cout << "Enter a number from 1 to 10 ";
    cin >> num;
    if (num < 1 || num > 10)
    {
      cout << "Number is out of requested range.\n";
    }
   }
   while (                                              );
   cout << "Out of loop with a valid integer value of "
        << num << endl;
}
```

Show the output from each of the loops shown below.

```
num = 4;
while(num <= 3)
{
  cout << num << endl;
  num = num + 1;
}
cout << "\nFinal value of num is " << num;
```

```
num = 4;
do
{
  cout << num << endl;
  num = num + 1;
}
while(num <= 3);
```

```
n = 1;
for(i = 1; i <= 3; i = i + 1)
{
   do
   {
      n = 2 * n;
   }
   while(n <= 3);
}
cout << n << endl;
```

```
n = 1;
for(i = 1; i <= 3; i = i + 1)
{
   while(n <= 3)
   {
      n = 2 * n;
   }
}
cout << n << endl;
```

A Few Hints for Looping Algorithms

1. When the loop is to execute a specific number of times, the counter-controlled *for* loop is the best choice.

2. If the loop is controlled by an event rather than a counter and the body may or may not be executed, the *while* loop is the best choice.

3. If the loop is controlled by an event rather than a counter and the body must be executed at least one time, the *do while* loop is the best choice.

4. When in doubt, use the *while* loop.

Review Questions

1. Name the three steps necessary for the proper execution of all loops.
_______________, _________________ and __________________.

2. A special value that signals that the loop should be ended is called a _____________.

3. All loops are controlled with a ______________________________.

4. A loop that has the condition at the top is called a ______________ loop.

5. A loop that has the condition at the bottom is called a _______________ loop.

6. The while and for loops are examples of _____________________ ___________________________. The do..while loop is an example of ____..

CS 1A Programming Toolkit

Basic Statements:	Decision Statements:
assignment	if
cin	if .. else
cout	

Accumulators:	Repetition Statements:
counter	while
running total	for
	do..while

If an accumulator is used in a program it must be ______________________________.

A program to process grades for 100 students should be written using a ____________ loop.

A program that prompts the user for ages and quits processing when the value –999 is entered for the age should be written using a ___________________ loop. It is possible that –999 could be the first value entered.

A variable used to add up all the ages entered is an example of a _________________ __________________________.

A variable that keeps track of the total number of people who took an exam is an example of a ____.

List all tools and variables necessary for a program that reads in weights from the keyboard until the value -1 is entered. At the end of the program, the average weight, the number of people weighed, the highest weight, and the lowest weight are output. Assume that at least one weight will be entered.

How would this program differ if there was a possibility that no weights would be entered?

What about exactly 50 weights?

Find the Errors and Output Exercises

Following is a program that will count the number of positive integers entered, the number of negative integers entered and the number of integers equal to zero that were entered. Assume all variables have been previously declared.

```
zeroCt = 0;
posCt = 0;
negCt = 0;
cout << "Enter an integer (-999) to stop: ";
cin >> num;
while(num != -999)
{
    if(num = 0)
    {
        zeroCt = zeroCt + 1;
    }
    else
    {
        if(num < 0)
        {
            negCt = negCt + 1;
        }
        else
        {
            posCt = posCt + 1;
        }
    }
    cout << "Enter an integer (-999) to stop: ";
    cin >> num;
}
cout << posCt << endl << negCt << endl << zeroCt;
```

Show the output given the following input values:

5 12 -3 0 -1 6 0 -999

Following is a program to calculate the exam average for each of 20 students. Each student took 3 exams. Comment on the code and indicate any necessary corrections.

```
scoreSum = 0;
for(i = 1; i <= 20; i = i + 1)
{
    for(j = 1; j <= 3; j = j + 1)
    {
        cout << "Enter score #" << j << " : ";
        cin >> score;
        scoreSum = scoreSum + score;
    }
    avgScore = scoreSum / 3.0;
    cout << "Exam average for student " << i << " : " << avgScore;
}
```

What is the output from the following loop?

```
for(i = 1; i <= 100; i = i - 1)
{
    cout << i << endl;
}
```

Correct any syntax errors in this program

```
#include (iostream)
#include (iomanip)
using namespace std;

void main(void)
{
    int const VAL = 5;
    float num1;
    float num2;
    char 'Y';

    cout >> "Enter first value: ";
    cin << num1;
    cout >> "Enter second value: ";
    cin << num2;
    average = (num1 + num2 + VAL) / 3.0;
    cout << "The average is " << average;
    VAL = VAL + 1;
}
```

The following program should calculate N! (N factorial). Comment on this code.

```
cout << "Enter an integer for a factorial calculation: ";
cin >> num;
factorial = 0;
for(i = num; i >= 1; i = i - 1)
{
    factorial = factorial * i;
}
cout << num << " factorial is " << factorial;
```

C++ Final Exam Review

In C++ ____________________________ are used to name things. They must begin with a _________________ or _____________________ and they may contain ________________ ____.

A ____ is a location in memory that contains a data value that does not change.

A _________________________________ is a location in memory that contains a data value that may be changed.

The implicit change from one data type to another performed by the compiler is called ____. Give an example of a statement where this would occur.

The explicit change from one data type to another performed by the programmer is called ____. Give an example of a statement where this would occur.

The _______________________ and _______________________ loops are examples of pre-test loops. Describe the action of a pre-test loop.

The ________________________ is a post-test loop. Describe the action of a post-test loop.

All loops must have a ____________________ and it must be ______________________ before the loop, __________________________ in a boolean expression, and ____________________ before the end of the loop.

Given the following

```
do
{
    cout << "Enter an integer: ";
    cin >> num;
    if(num <1 || num >100)
    {
        cout <<"Value out of range\n";
    }
}
while(num < 1 || num > 100);
```

What is the purpose of the above loop?

What is output from the following code fragment?

```
val = 10;
while(val >= 1)
    cout << val << ' ';
    val = val - 1;
cout << "Done";
```