Chapter 6

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

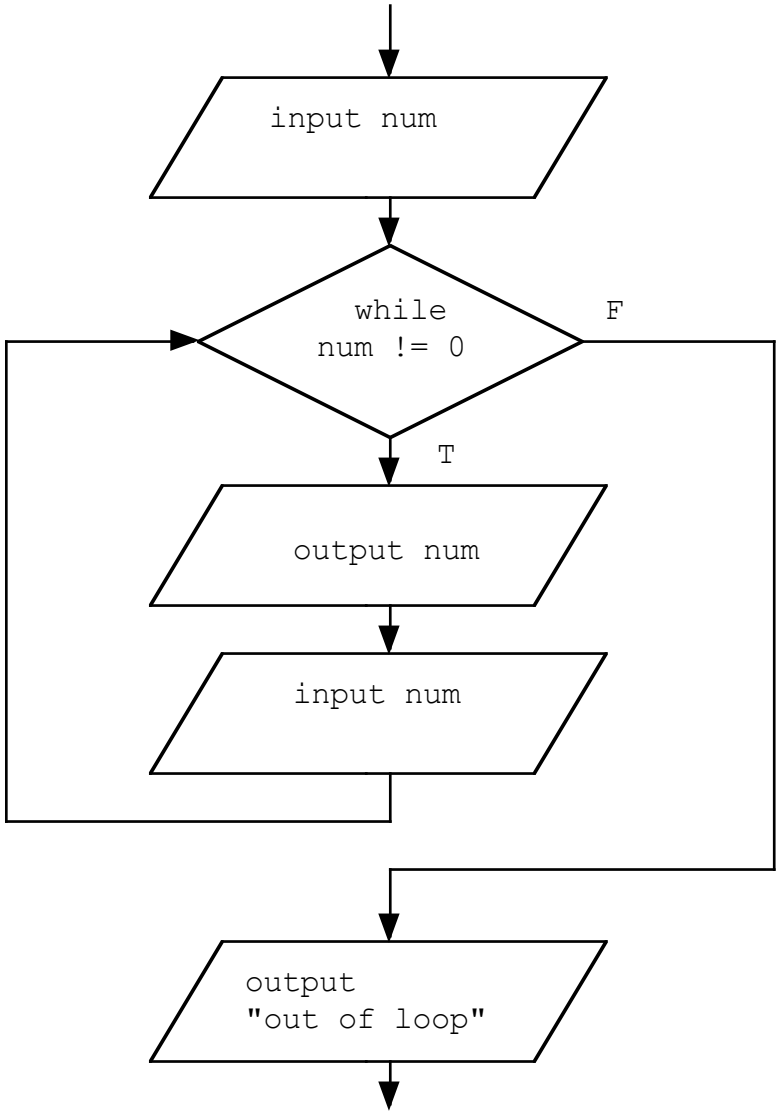
Sentinel value – a special value for the *loop control variable* that signifies that the loop should terminate

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

GENERAL FORM of a While statement

```
while (Boolean 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 page 3, 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 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.

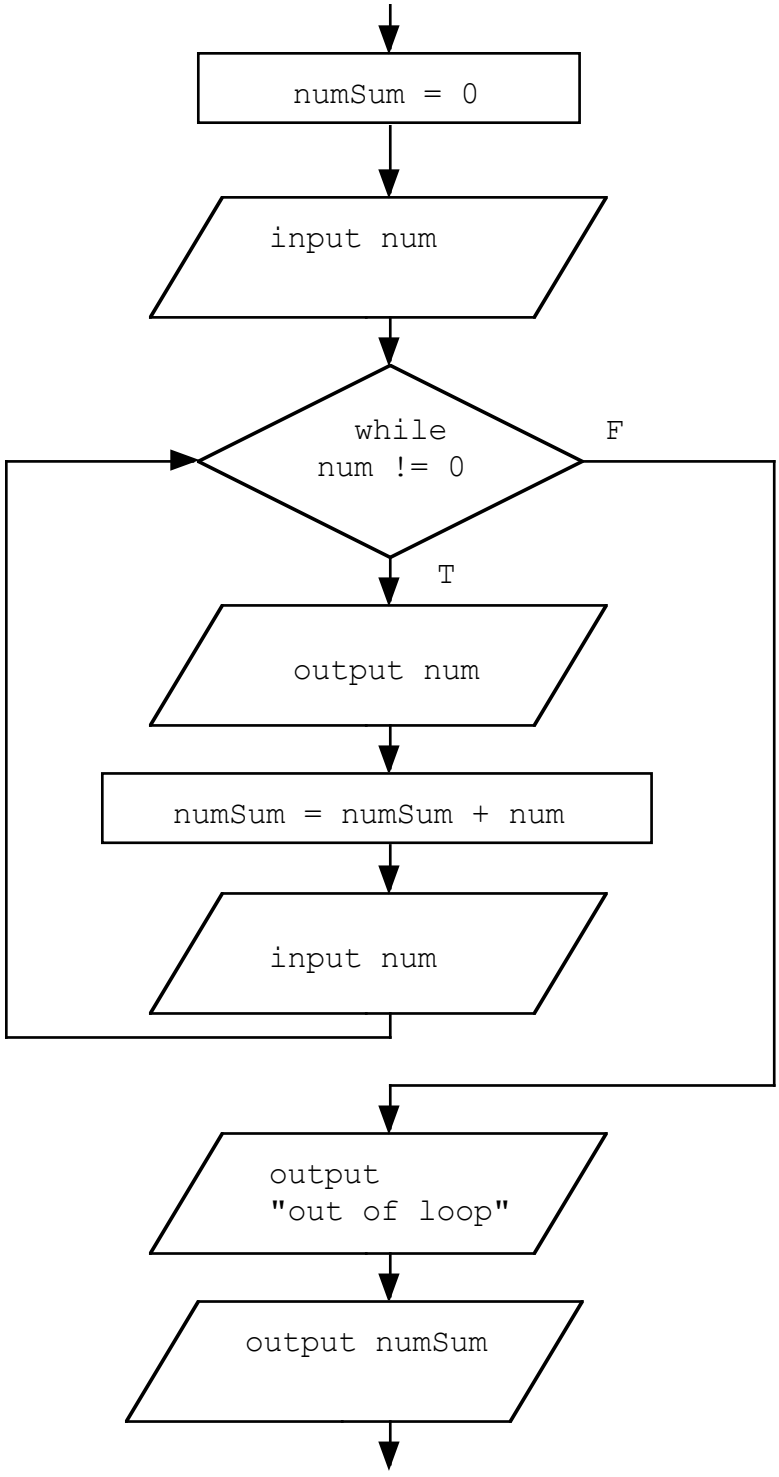
A common function performed by looping statements is *accumulation*. There are two types of accumulators, *counters* and *running totals*.

Counter – an accumulator that increments 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 would use a counter to count the number of people and a running total to add up all of their ages. The average would simply be the running total divided by the counter.

Let's modify the previous program to sum the integers 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).


numSum = 0
input num
while
num != 0
F
T
output num
numSum = numSum + num
input num
output
"out of loop"
output numSum

C++ code – complete program:

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

#include <iostream.h>

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;
}
```

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 ";
}
```

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?

When loops are to be executed a specific number of times the while loop is not always a good choice. The second loop we will examine is the *for* loop.

The C++ for Statement

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.

- Expression1 is simply the while condition
- Expression2 is typically used to increment of decrement the LCV
- InitStatement may be either the null statement (just a semicolon), a declaration statement, or an expression statement (this is the format we will use in 1A)

Examples:

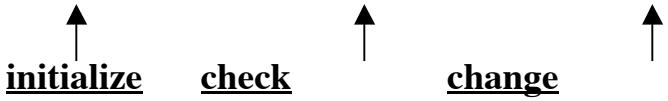
```
// This loop prints the integers 1 to 50
for (i = 1;  i <= 50;  i = i + 1)
{
  cout << i << endl;
}


// 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 program from page 10 modified to show the use of the For statement.

```
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;
}
```

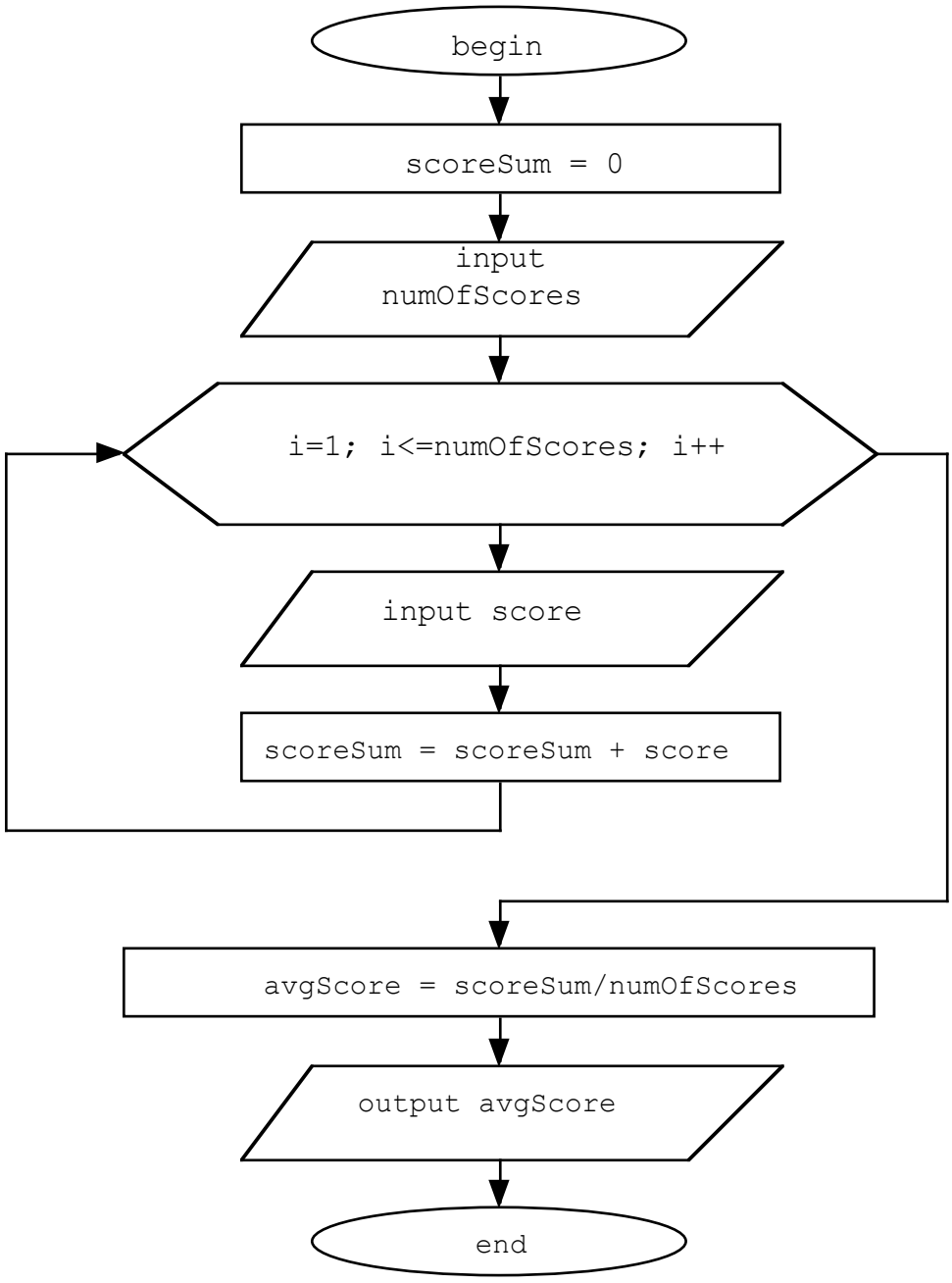
Notice that the essential steps for loop control are all built into the For statement itself.

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



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

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.



```
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. 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;
    }
}
```

2. 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;
    }
}
```

3. Show the output from the following code segment:

```
#include <iostream.h>
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;
    }
}
```

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

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

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);
```

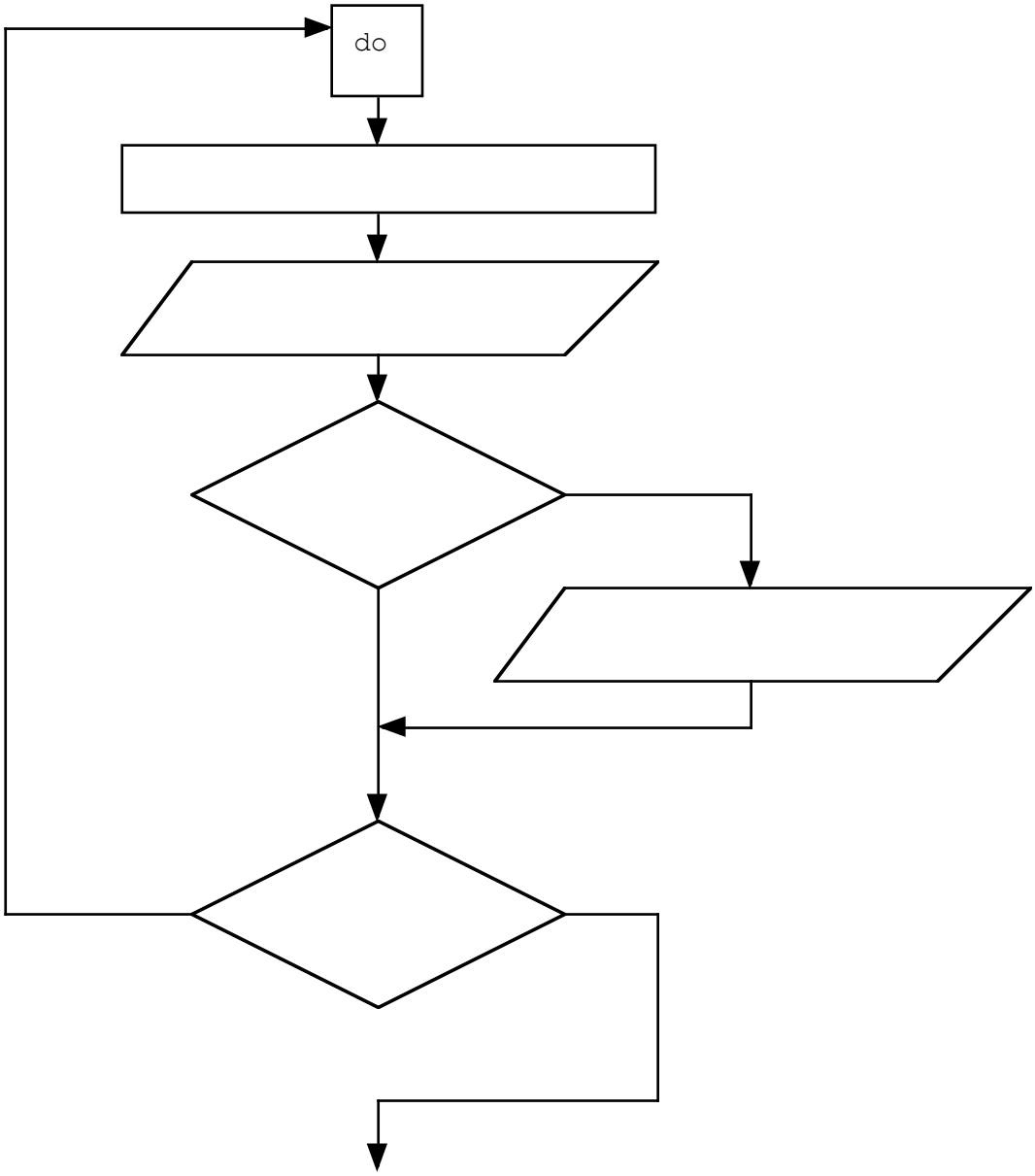
A common use for this loop is in checking user input.

This loop would continue to prompt the user for their weight until the weight entered was considered valid. For our purposes, the user must weigh at least 10 pounds.

```
void main(void)
{
  int weight;

  do
  {
    cout << "Enter your weight: ";
    cin >> weight;
    if (weight < 10)
    {
      cout << "You must weigh at least 10 pounds to use this program.\n";
    }
   }
   while (weight < 10);
   cout << "Out of loop with a valid weight of " << weight << " pounds.";
}
```

Flowcharting the Do While Statement

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.



```
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;
}
```

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.

Looping Exercises

1. Flowchart a loop to calculate N!. N! is defined to be 1 for N = 1 and N(N-1)! for N > 1. 5! = 5*4*3*2*1 = 120 The value of N is input from the keyboard. Choose the most appropriate loop.

2. Flowchart a loop that screens user input for numbers in the range 0 – 100 (inclusive) or -999. Any other value should cause an error message to be output and the loop to continue executing. Choose the most appropriate loop.

3. Flowchart a loop that sums a stream of integers while the value –999 has not been input. Output the sum when the loop terminates. Choose the most appropriate loop.

4. Combine the loops from questions 2 and 3. Use the loop in #2 to initialize and change the LCV in #3.

5. Design a loop that screens user input for odd numbers in the range 5 to 500 (inclusive) or the value –1. Any other value will cause an error message to be displayed and the loop to execute again.

6. A prime number is a number defined to be a number that is <u>evenly</u> divisible only by 1 and itself. I can check to see if an integer is prime in the following way. Let's look at 9.

    9 divided by 2 = 4.5 (still a candidate)
    9 divided by 3 = 3 (cannot be prime)

    What about 17?
    17 divided by 2 = 8.5 (keep checking)
    17 divided by 3 = 5.66 (keep checking)
    17 divided by 4 = 4.25 (keep checking)
    17 divided by 5 = 3.4 (looks prime)

    It is only necessary to divide by numbers that are less than or equal to the square root of the number being checked. Use the C++ sqrt( ) function. Design a program that reads in a number from the keyboard and checks to see whether it is prime or not. Use a while loop to check the number. The loop should terminate if you find that the number is not prime or you have divided by all of the numbers in the range 2 to the square root of the number being checked. Use the following:

```
bool isPrime;
int num;
int div;

// initialize the 2 loop control variables
isPrime = true;
div = 2;
while (                              &&                              )
{
   // ask a question and modify the appropriate loop control variable

}

if(isPrime)
   cout << "Number is prime";
else
   cout << "Number is not prime";
```