Programming Fundamentals I

Chapter 5:

Control Structures II (Repetition)

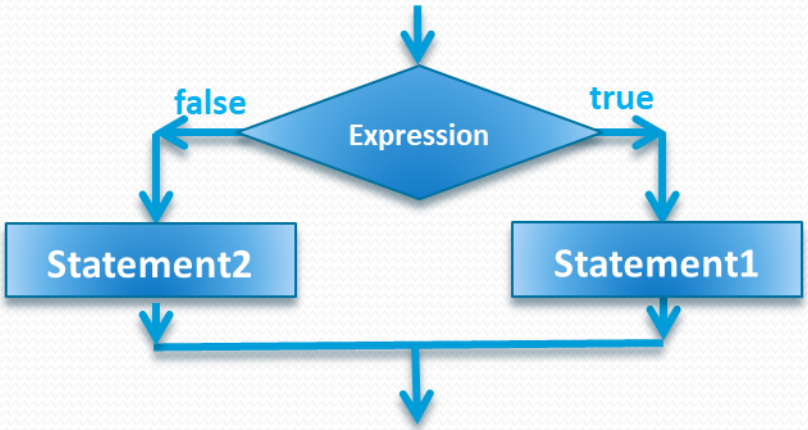
Dr. Adriana Badulescu

Objectives

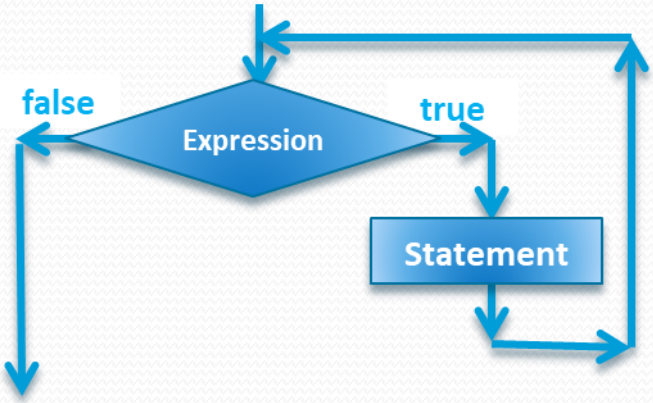
- Learn about repetition (looping) control structures
- Explore how to construct and use different types of repetition structures
- Learn about the while, do-while and for loop repetition statements
- Examine break and continue statements
- Discover how to form and use nested loops control structures
- Learn how to debug loops

Control Structures

Sequence



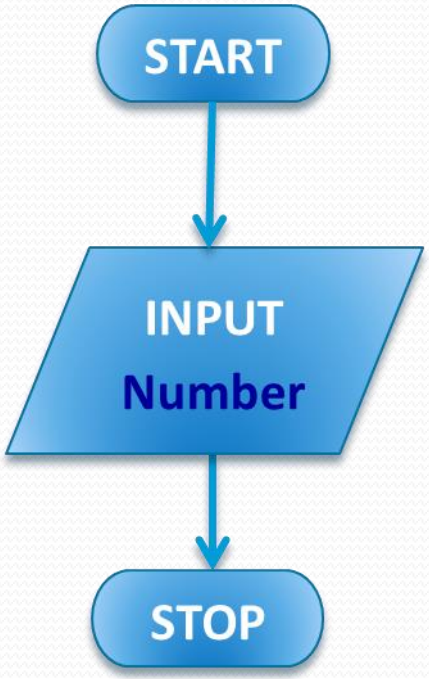
Selection



Repetition

Why is Repetition Needed?

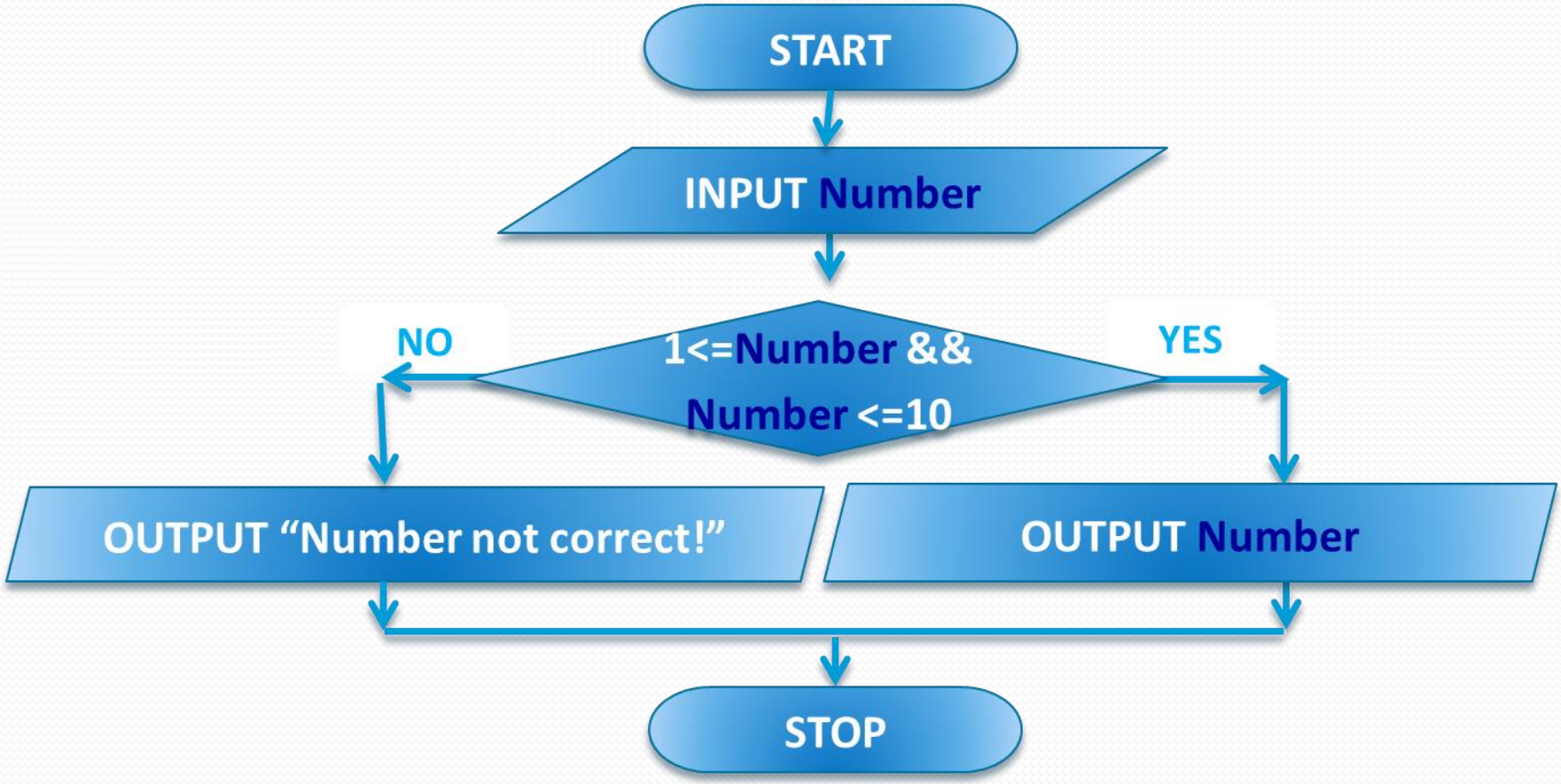
- Read number between 1 and 10




```
//define Number
int Number;
//prompt for Number
cout << "Enter a number between 1 and 10: ";
//read Number
cin >> Number;
```

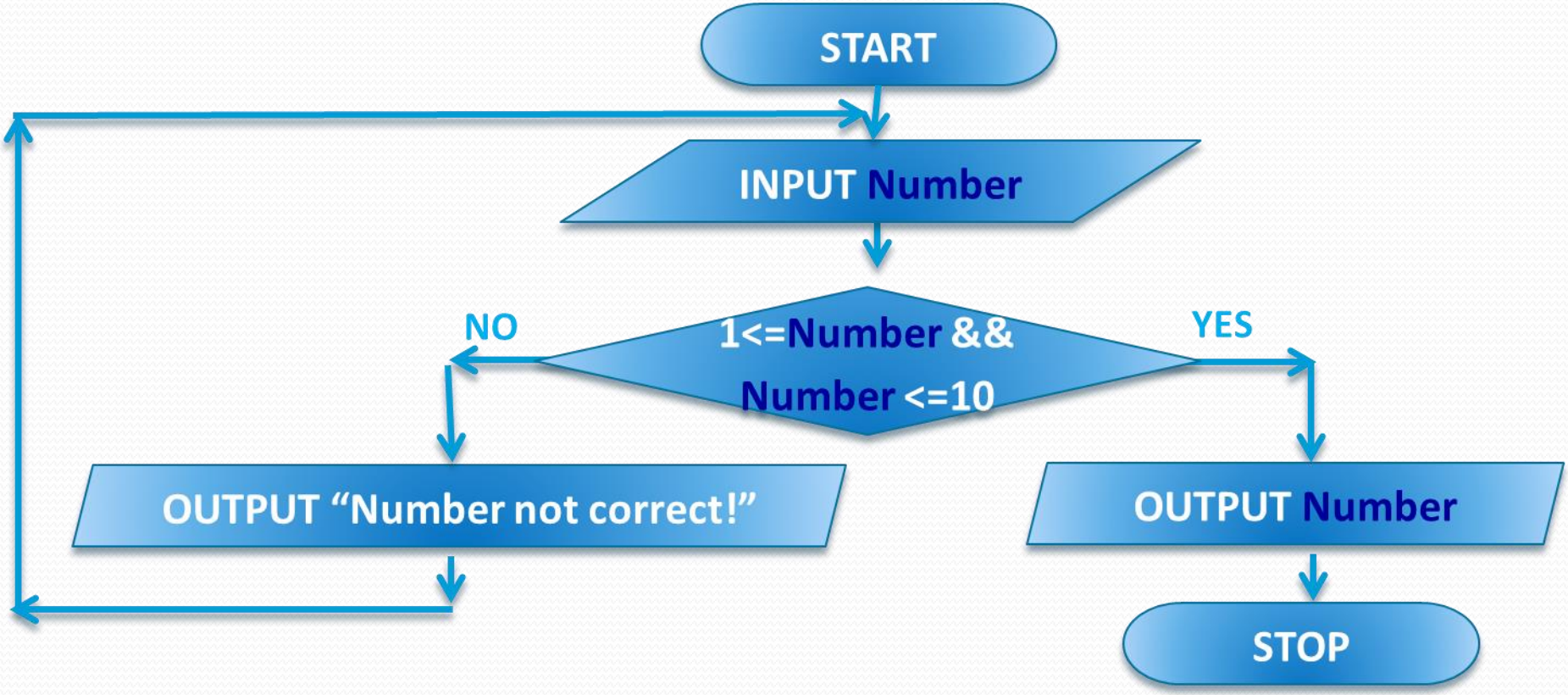
Why is Repetition Needed?

- Read number between 1 and 10



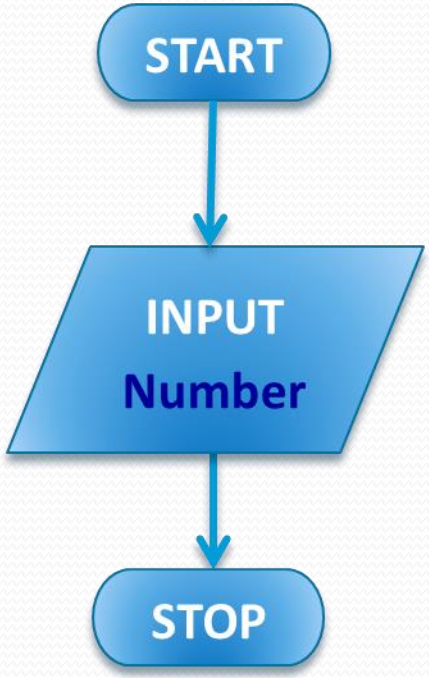
Why is Repetition Needed?

- Read number between 1 and 10



Why is Repetition Needed?

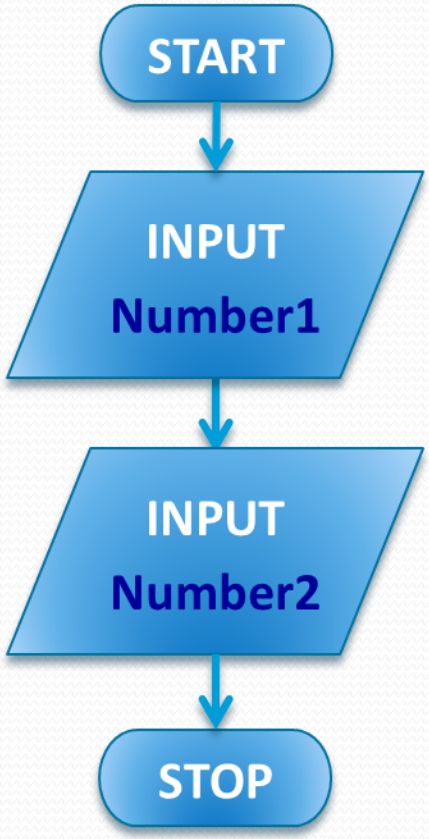
- Read one number




```
//define Number
int Number;
//prompt for Number
cout << "Enter a number: ";
//read Number
cin >> Number;
```

Why is Repetition Needed?

- Read 2 numbers




```
//define Number1
int Number1;
//prompt for Number1
cout << "Enter a number: ";
//read Number1
cin >> Number1;
```

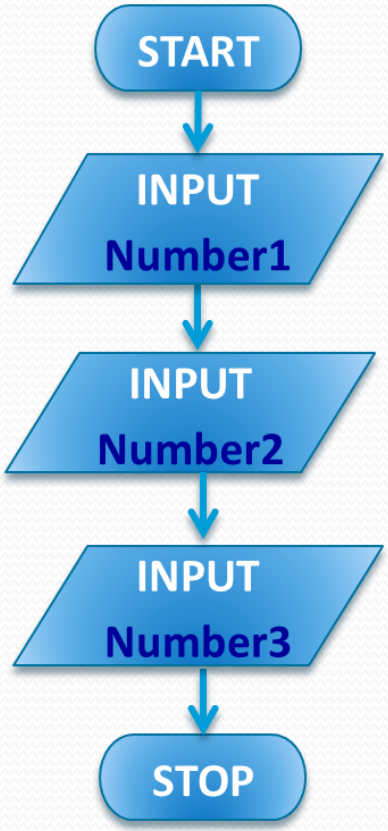
Number1

```
//define Number2
int Number2;
//prompt for Number2
cout << "Enter a number: ";
//read Number2
cin >> Number2;
```

Number2

Why is Repetition Needed?

- **Read 3 numbers**




```
//define Number1
int Number1;
//prompt for Number1
cout << "Enter a number: ";
//read Number1
cin >> Number1;
```

Number1

```
//define Number2
int Number2;
//prompt for Number2
cout << "Enter a number: ";
//read Number2
cin >> Number2;
```

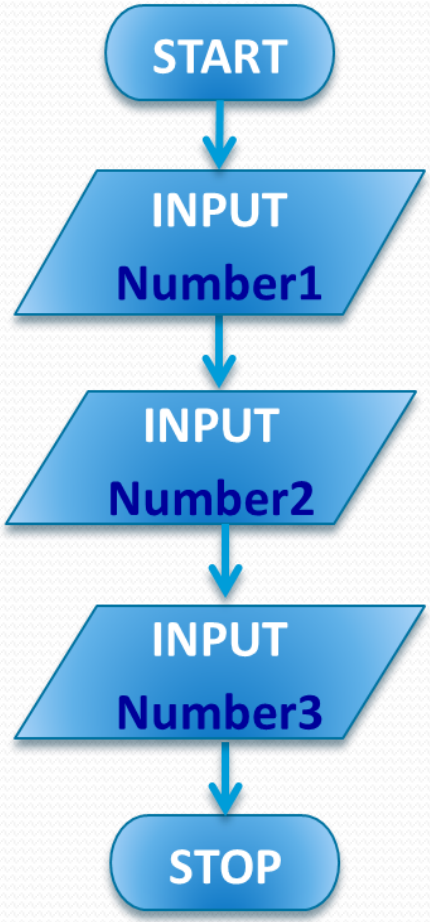
Number2

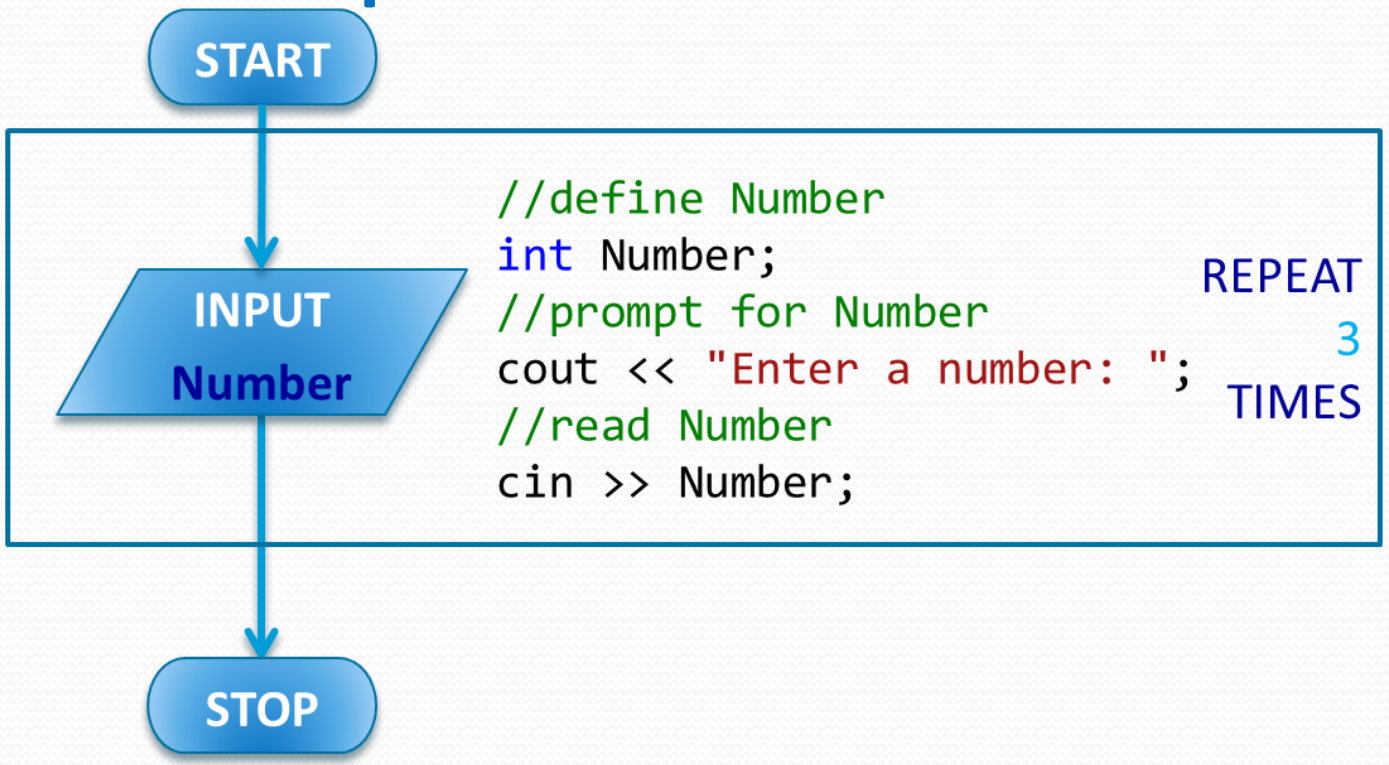
```
//define Number3
int Number3;
//prompt for Number3
cout << "Enter a number: ";
//read Number3
cin >> Number3;
```

Number3

Why is Repetition Needed?

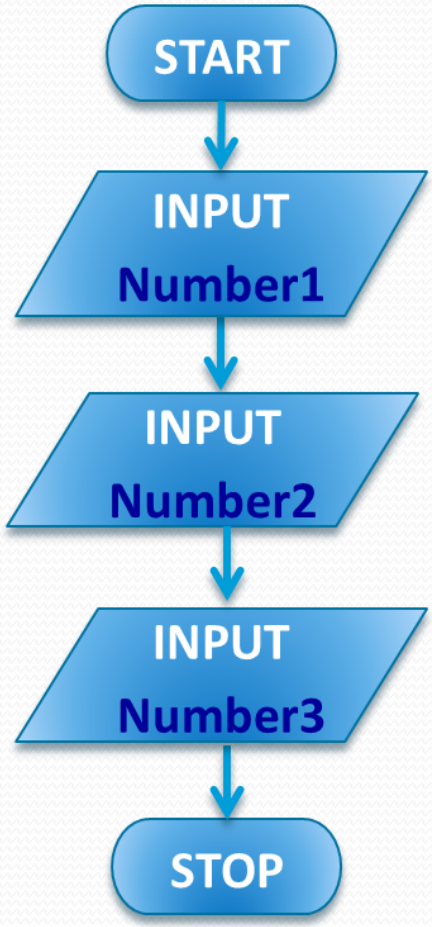
- Read 3 numbers = Repeat read 1 number 3 times

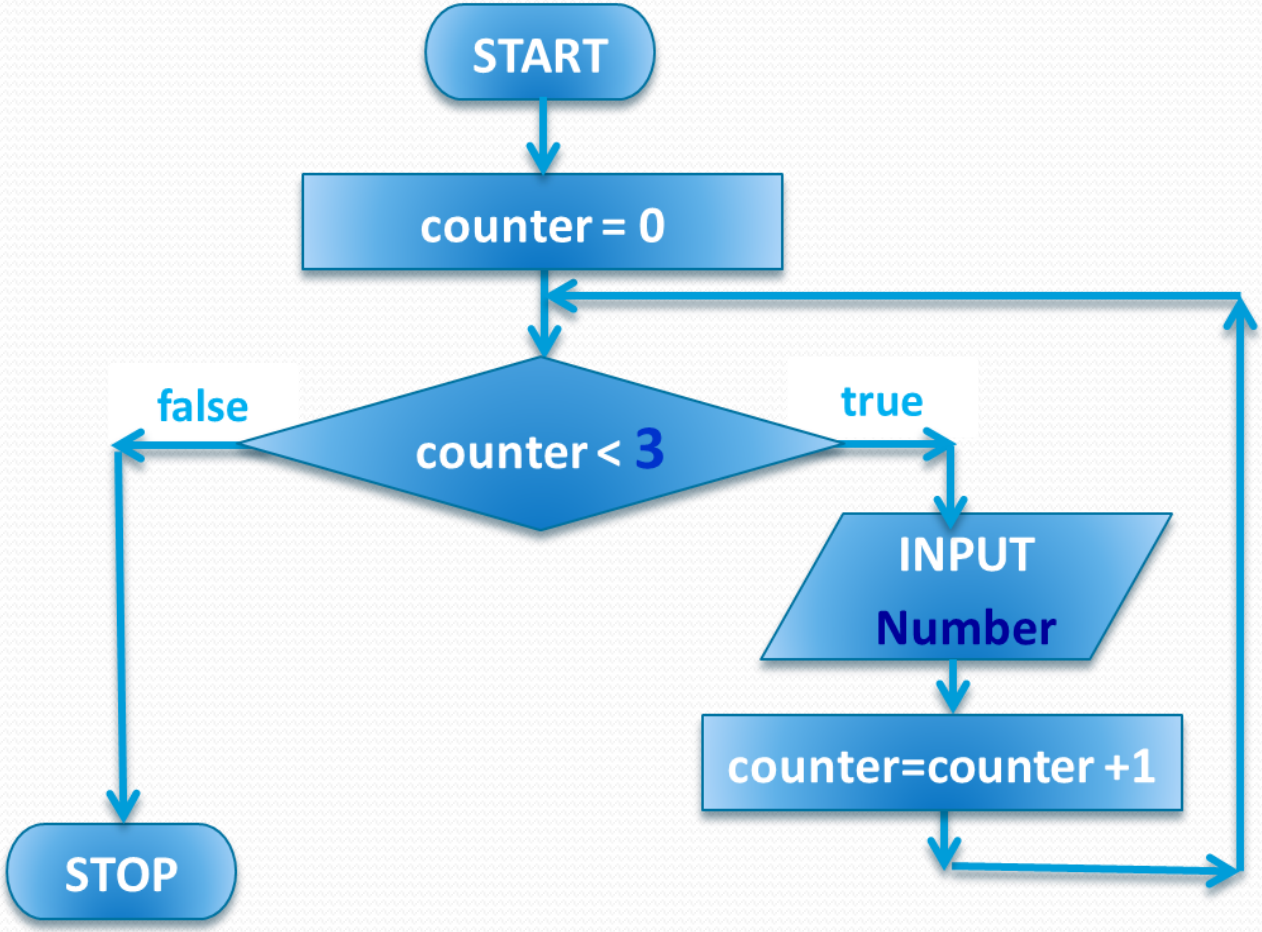




Why is Repetition Needed?

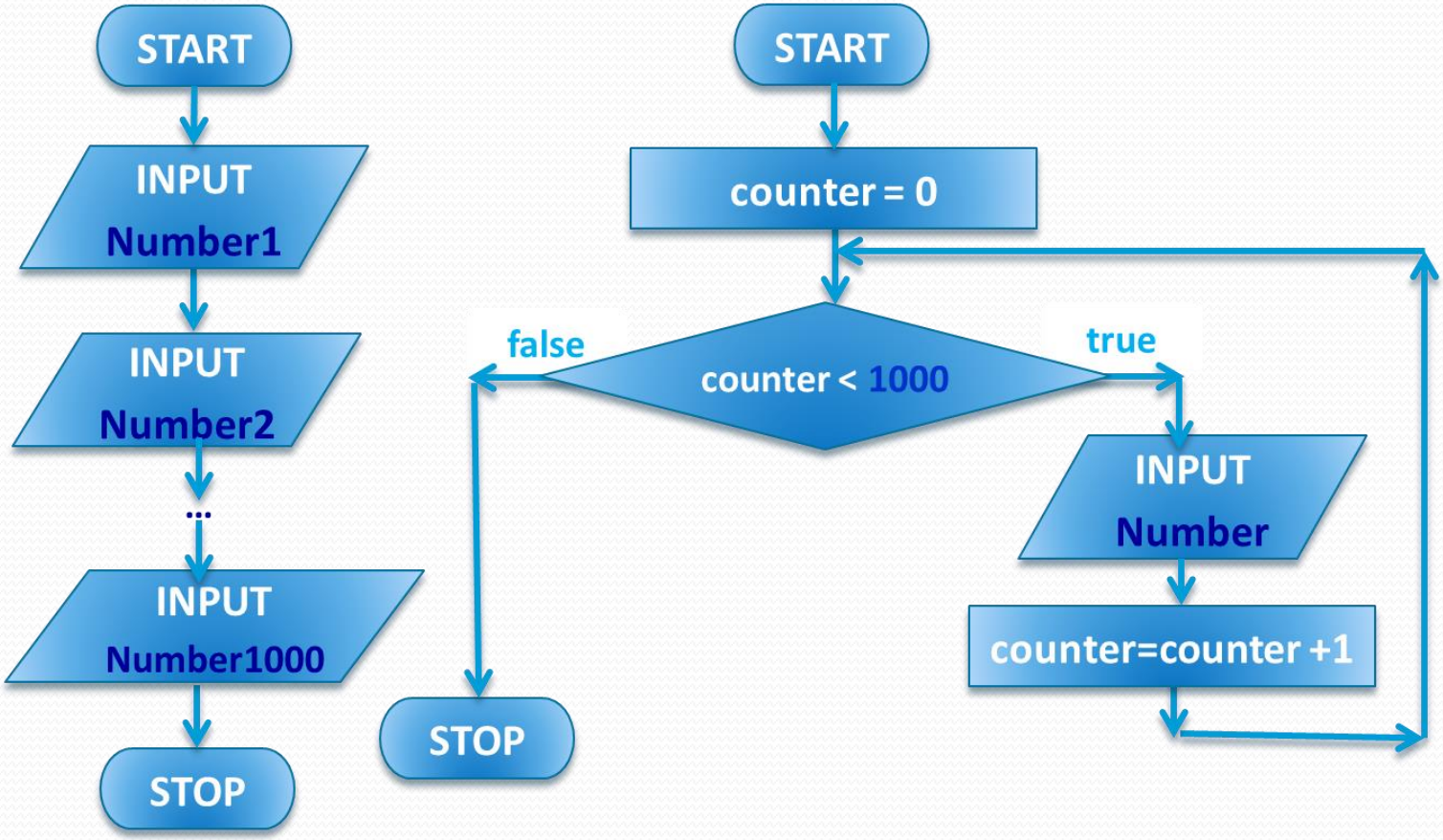
- Read 3 numbers





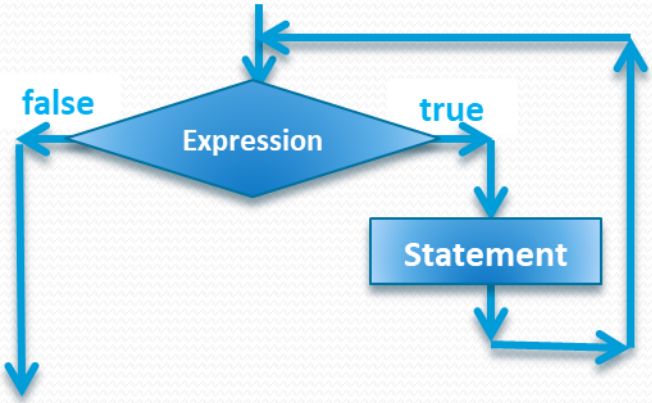
Why is Repetition Needed?

- Read 1000 numbers



Why Is Repetition Needed?

- Repetition allows you to efficiently use variables and memory
- Validate values
- Program faster
- Generate patterns

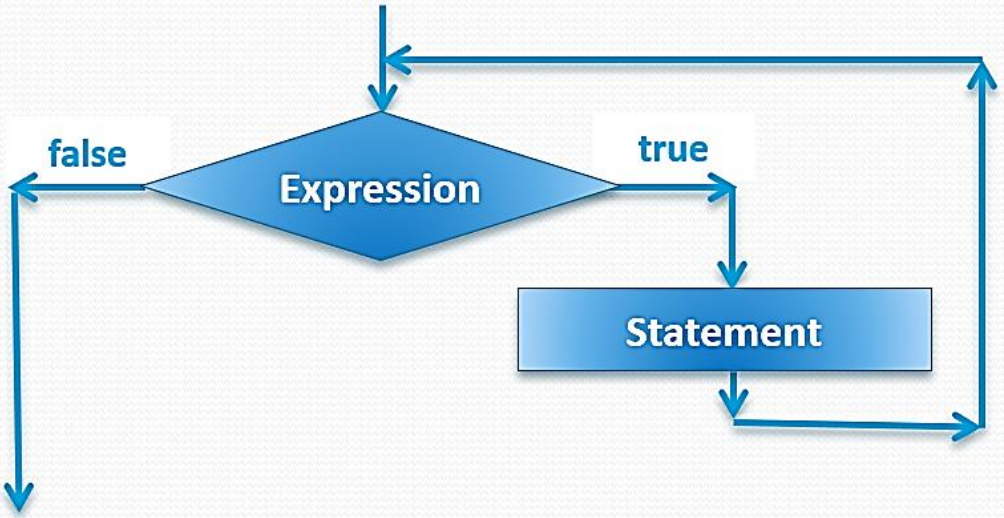


Repetition

`while` Looping (Repetition) Structure

- The general form of the **`while`** statement is:

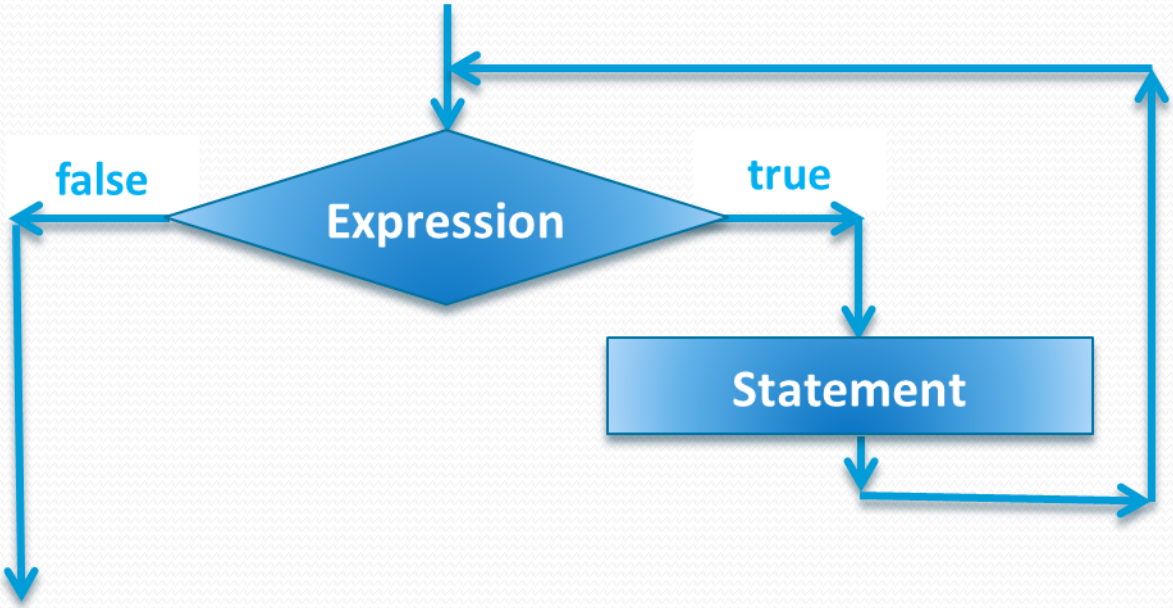
```
while ( Expression )
      Statement;
```



- Expression acts as a decision maker and is usually a logical expression
- Statement is called the body of the loop and can be simple or compound

`while` Looping (Repetition) Structure

```
while ( Expression )
      Statement;
```



- **Infinite loop**: continues to execute endlessly
  - Not solving the problem, not an algorithm
  - Avoided by including statements in loop body that assure exit condition is eventually `false`

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Expression

Statement

Sample run:

```
0 5 10 15 20
```

5 times

while Looping (Repetition) Structure

```
i=0;                                    START
while (i<=20)                           STOP        Expression
{
   cout << i << " ";                                Statement
   i=i+5;                               UPDATE/STEP
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=25;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

0 times

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
while (i<20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

depending on i

Random number

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=0;
while (i>=-20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20 25 …
```

INFINITE times

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=0;
while (i<=20);
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
_
```

INFINITE times

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=0;
while (1)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20 25 …
```

INFINITE times

while Looping (Repetition) Structure

```
i=0;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=0;
while (i<20)
{
   cout << i << " ";
}
cout << "\n";
```

Sample run:

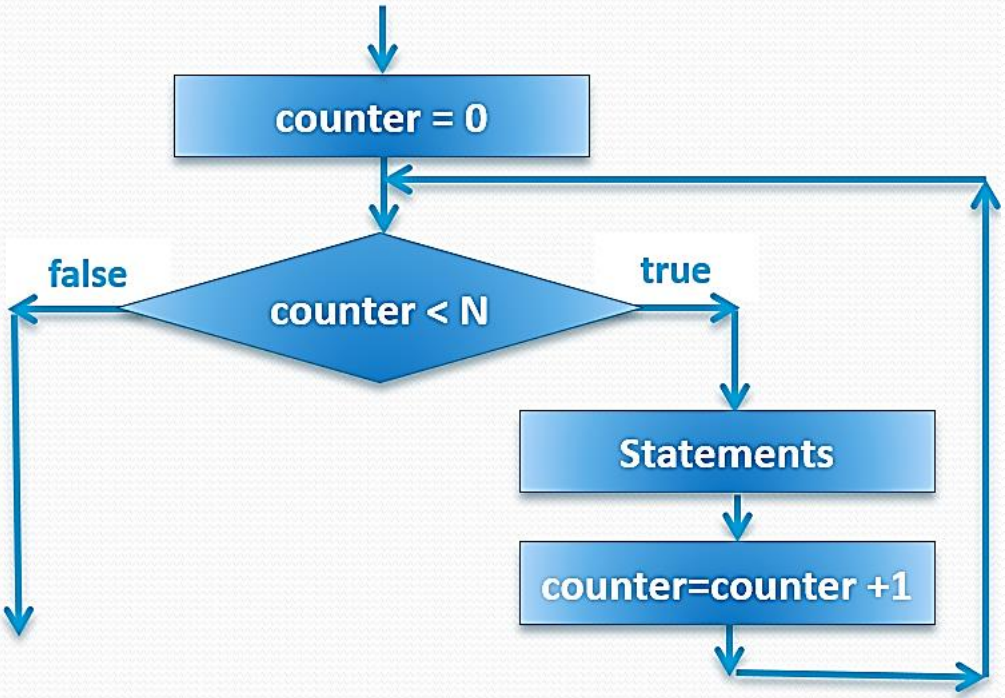
```
0 0 0 0 0 0 0 0 0 0 ....
```

INFINITE times

Case 1: Counter-Controlled Loops

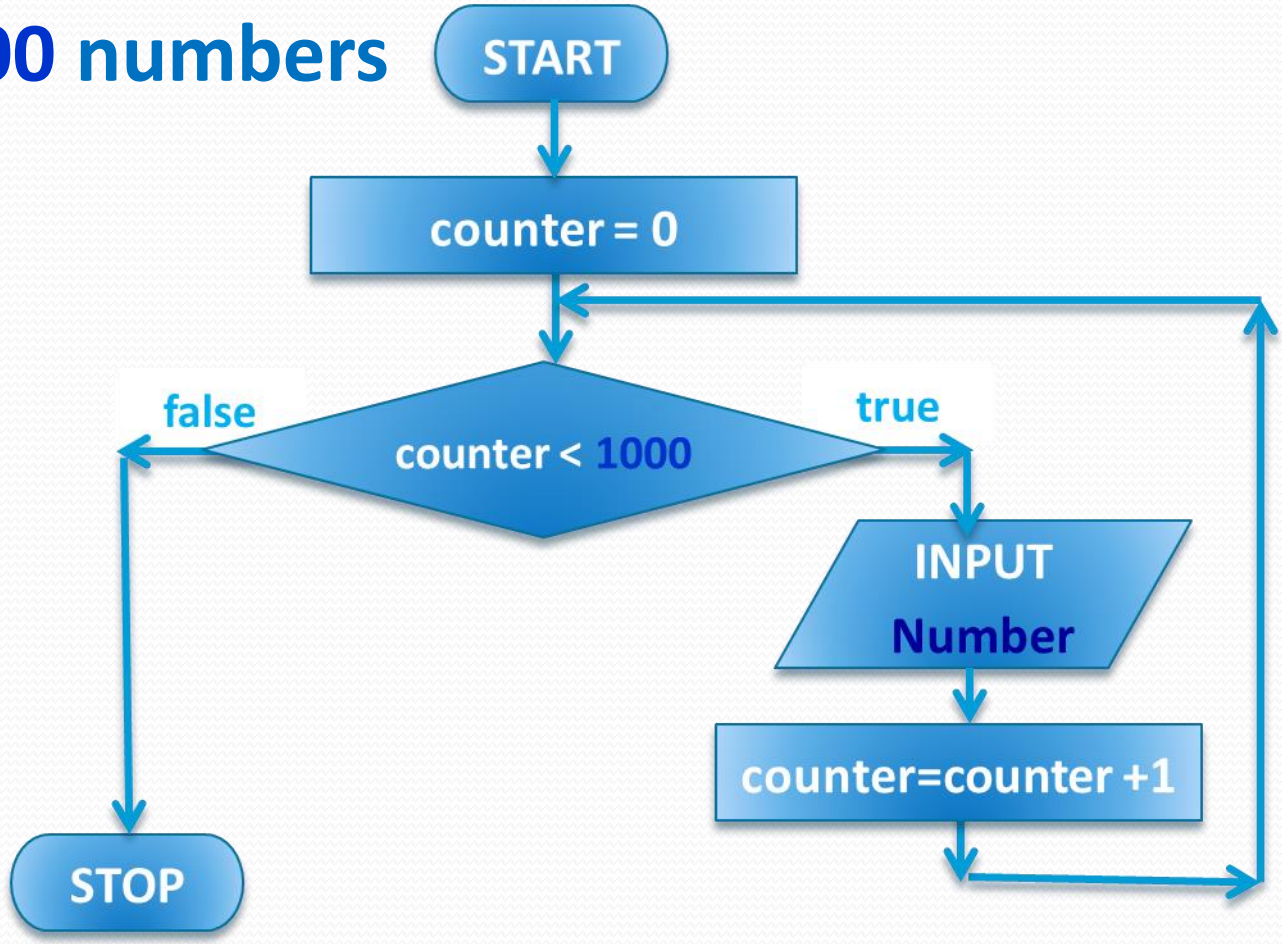
- If you know exactly how many times you need to repeat

```
//initialize the counter
counter = 0;
//test the counter
while (counter < N)
{
    //loop statements
    Statements;
    //update the counter
    counter++;
}
```



Case 1: Counter-Controlled Loops

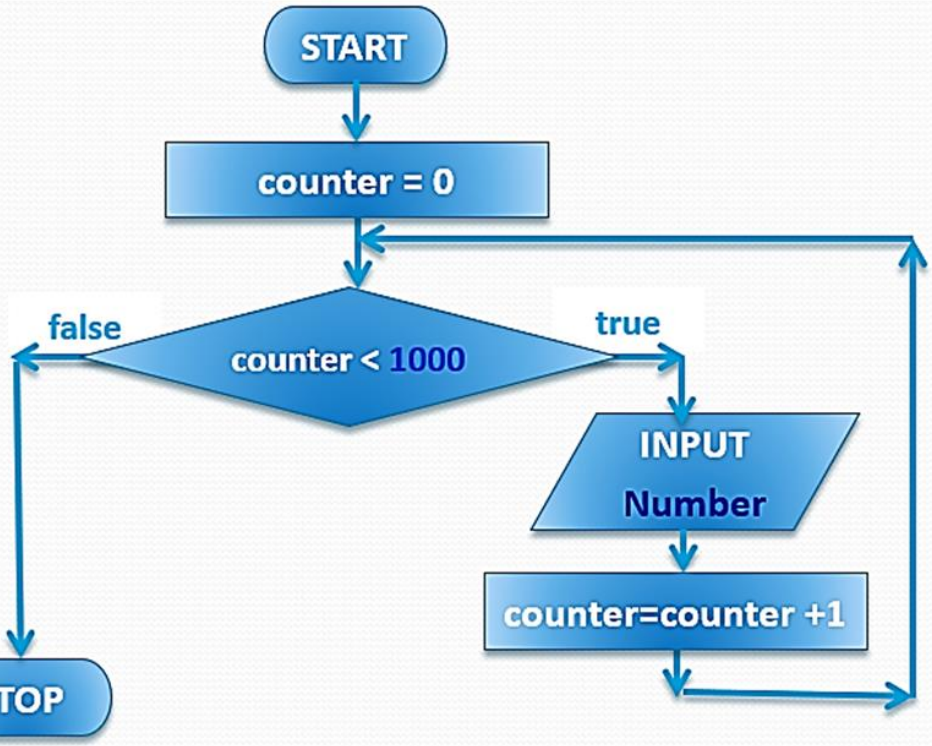
- Read 1000 numbers



Case 1: Counter-Controlled Loops

- **Read 1000 numbers**

```
//initialize the counter
int counter = 0;
//loop condition
while (counter < 1000)
{
    //loop statements
    //define Number                    INPUT
    int Number;                        Number
    //prompt for Number
    cout << "Enter a number: "
    //read Number
    cin >> Number;

    //update the counter
    counter++;
}
```



Example: Sum of Numbers

```
int limit;
int number;
int sum;
int counter;

cout << "Enter the number of " << "integers in the list: ";
cin >> limit;
cout << endl;

sum = 0;
counter = 0;
cout << "Enter " << limit << " integers." << endl;
```

Example: Sum of Numbers

```
while (counter < limit)
{
   cin >> number;
   sum = sum + number;
   counter++;
}

cout << "The sum of the " << limit << " numbers is " << sum
<< endl;

if (counter != 0)
   cout << "The average = " << sum / counter << endl;
else
    cout << "No input." << endl;
```

Exercise: Counter-Controlled Loop

- Read 5 integral numbers and compute the sum, average, and maximum number

```
//declare and initialize variable for Sum
int Sum = 0;

//declare and initialize variable for Max
int Max = 0;

//declare and initialize variable for Min
int Min = 1000000;

//initialize counter
int counter = 0;
```

Exercise: Counter-Controlled Loop

```
//loop for condition
while (counter < 5)
{
        //read number
        //declare variable
        int N;
        //prompt user for number
        cout << "\n\nEnter an integral number N: ";
        //input the value
        cin >> N;

        //update the sum
        Sum = Sum + N;

        //update the maximum
        if (Max < N)
                Max = N;

        //update the min
        if (Min>N)
                Min = N;

        //update counter
        counter++;
}
```

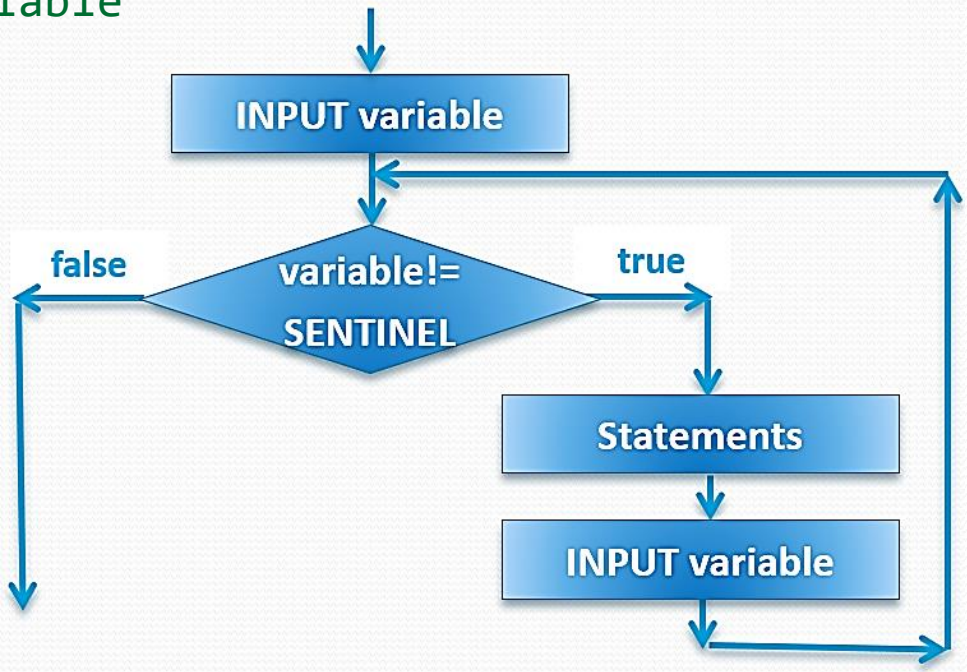
Exercise: Counter-Controlled Loop

```
//ouput the sum
cout << "\n\nThe sum is " << Sum;

//compute the average
double Average = static_cast<double>(Sum) / counter;

//output the average
cout << "\n\nThe average of the numbers is "<< Average;

//output the maximum
cout << "\n\nThe maximum of the numbers is " << Max;
```

Case 2: Sentinel-Controlled Loops

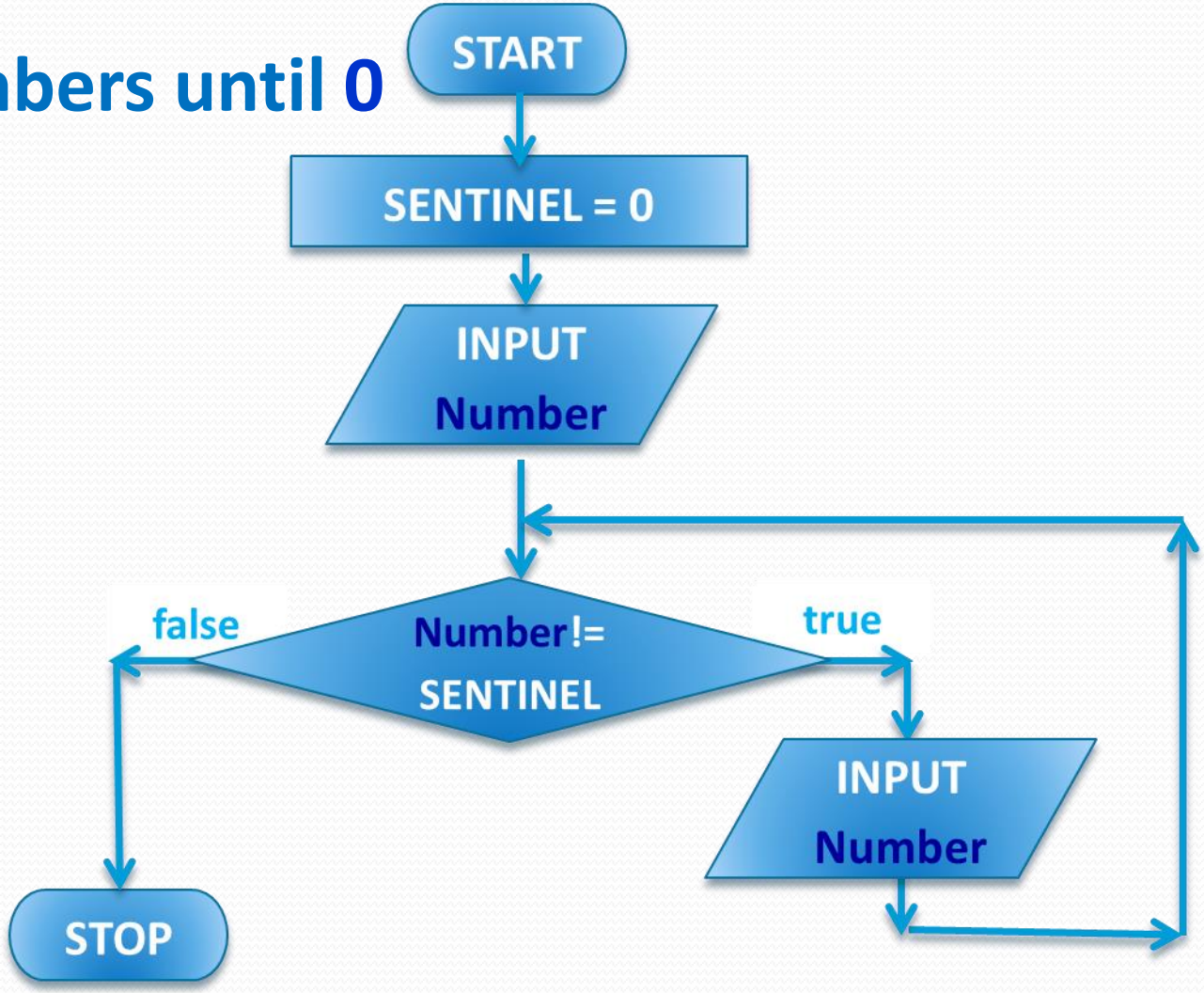
- Loop ends when sentinel variable is encountered

```
//initialize the loop control variable
cin >> variable;
//test the loop variable
while (variable != SENTINEL)
{
        //loop statements
        Statements;
        //update the loop variable
        cin >> variable;
}
```



Case 2: Sentinel-Controlled Loops

- Read numbers until 0



Case 2: Sentinel-Controlled Loops

- **Read numbers until 0**

```
//define the sentinel
const int SENTINEL = 0;
//initialize the loop control variable - read Number
int Number;                                                    INPUT
cout << "\n\nEnter a number (use "<<SENTINEL<<" to end): ";    Number
cin >> Number;
//loop condition
while (Number != SENTINEL)
{
    //loop statements

    //update - read another Number
    //prompt for Number                                        INPUT
    cout << "Enter a number (use "<<SENTINEL<<" to end): ";    Number
    //read Number
    cin >> Number;
}
```

Exercise: Sentinel-Controlled Loop

- Read integral numbers until a sentinel value and compute the sum, average, and maximum number

```
//constant for the sentinel
const int SENTINEL = 0;

//initialize the sum
Sum = 0;

//initialize the NumberOfNumbers
int NumberOfNumbers = 0;

//declare Number
int Number;
//initialize Number
cout << "\nPlease enter an integral number (use "<<SENTINEL<<" to
end): ";
cin >> Number;
```

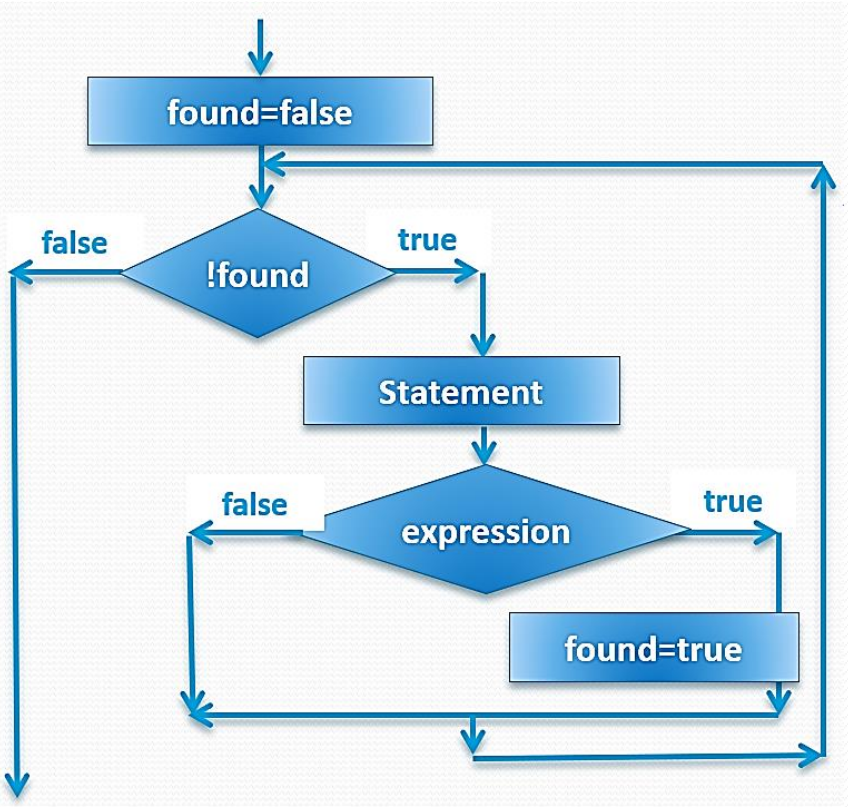
Exercise: Sentinel-Controlled Loop

```
//loop condition
while (Number != SENTINEL)
{
    //loop statements
    //update the sum
    Sum = Sum + Number;
    //update NumberOfNumbers
    NumberOfNumbers++;
    //update Number
    cout << "\nPlease enter an integral number (use " << SENTINEL << " to
    end): ";
    cin >> Number;
}

//output the sum
cout << "\nThe sum of the numbers is " << Sum;
//compute the average
Average = static_cast<float>(Sum) / NumberOfNumbers;
//output the average
cout << "\n\nThe average of the numbers is " << Average;
```

Case 3: Flag-Controlled Loops

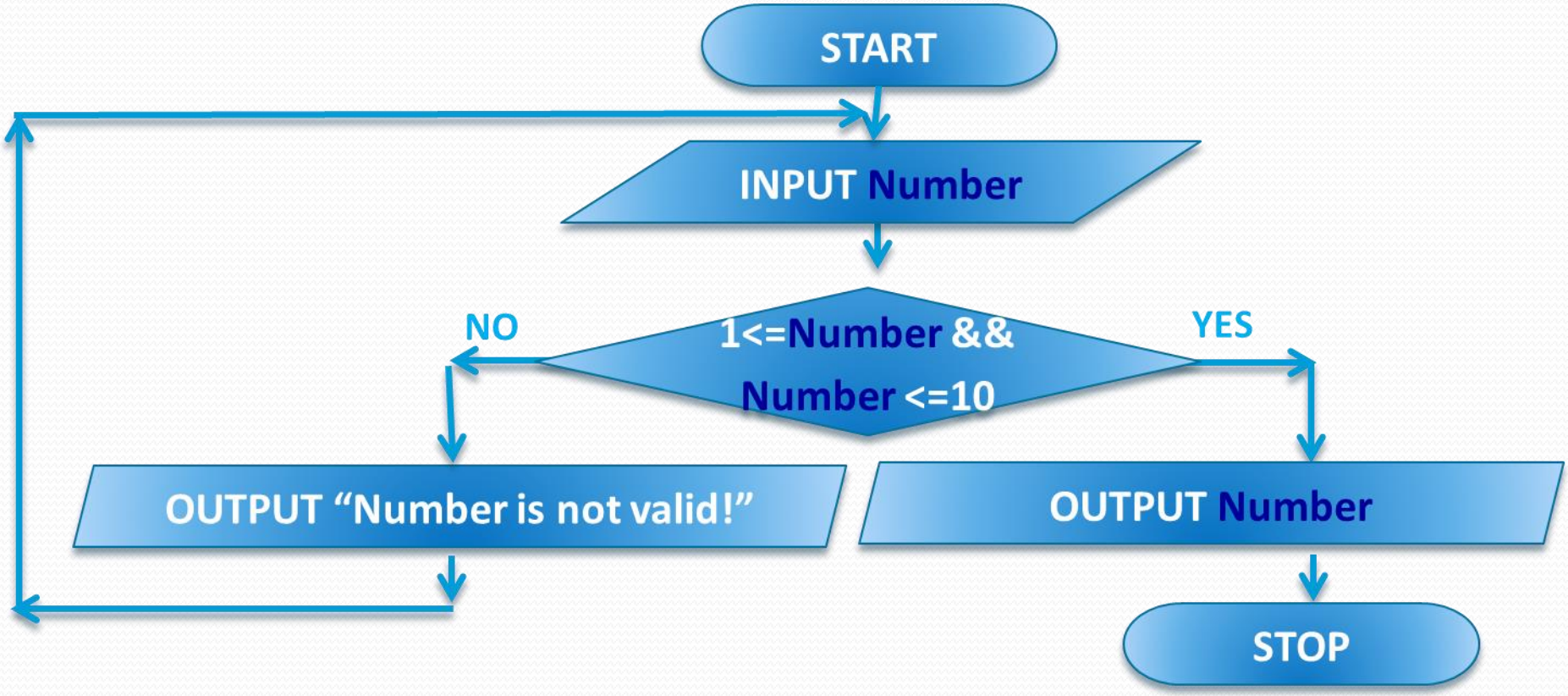
- A **flag-controlled loop** uses a `bool` variable to control the loop

```
//initialize the loop variable
found = false;
//test the loop variable
while (!found)
{
        //loop statements
        Statements;
        //update loop variable
        if (expression)
                found = true;
}
```



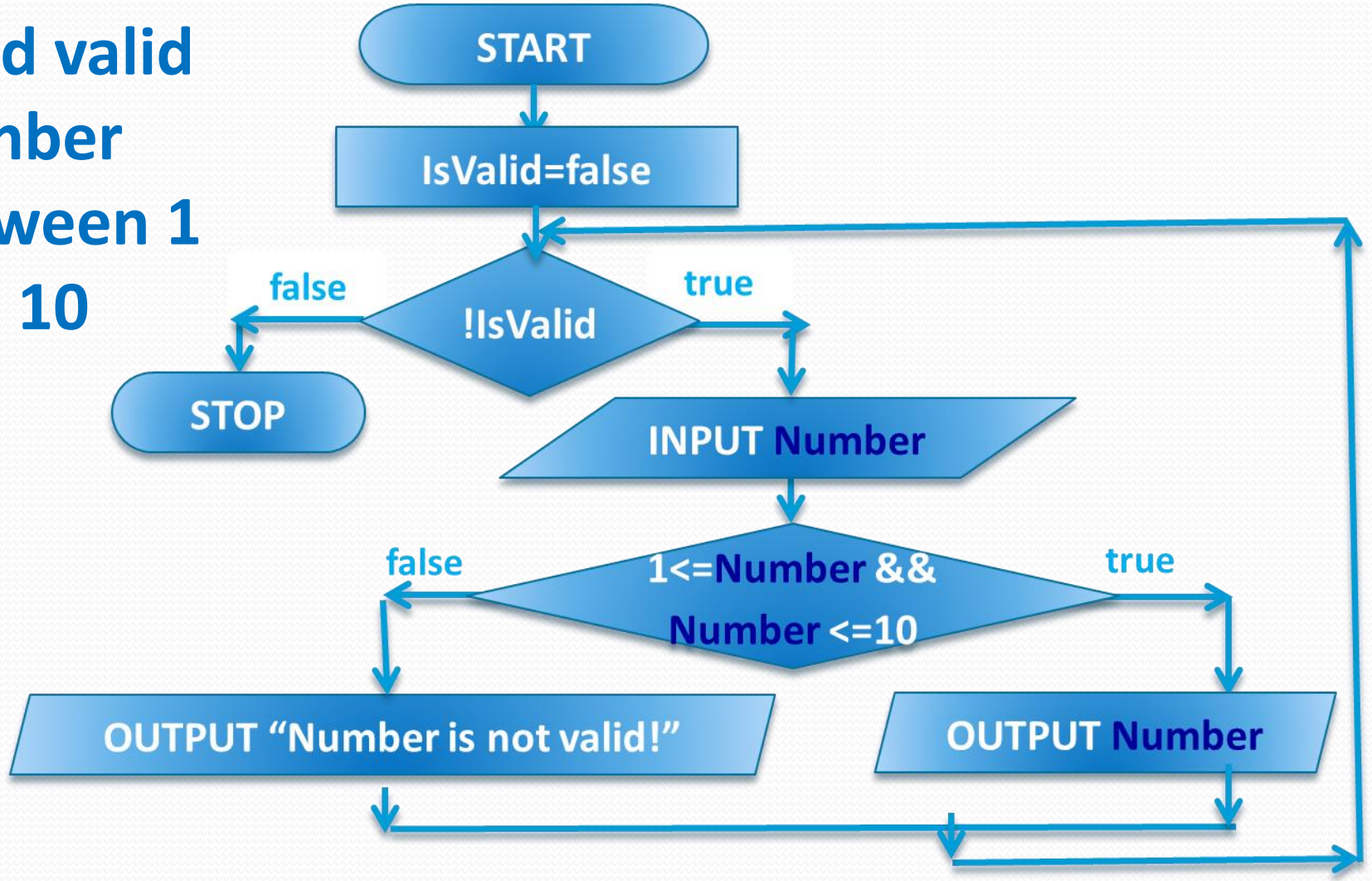
Case 3: Flag-Controlled Loops

- Read valid number between 1 and 10



Case 3: Flag-Controlled Loops

- Read valid number between 1 and 10



Case 3: Flag-Controlled Loops

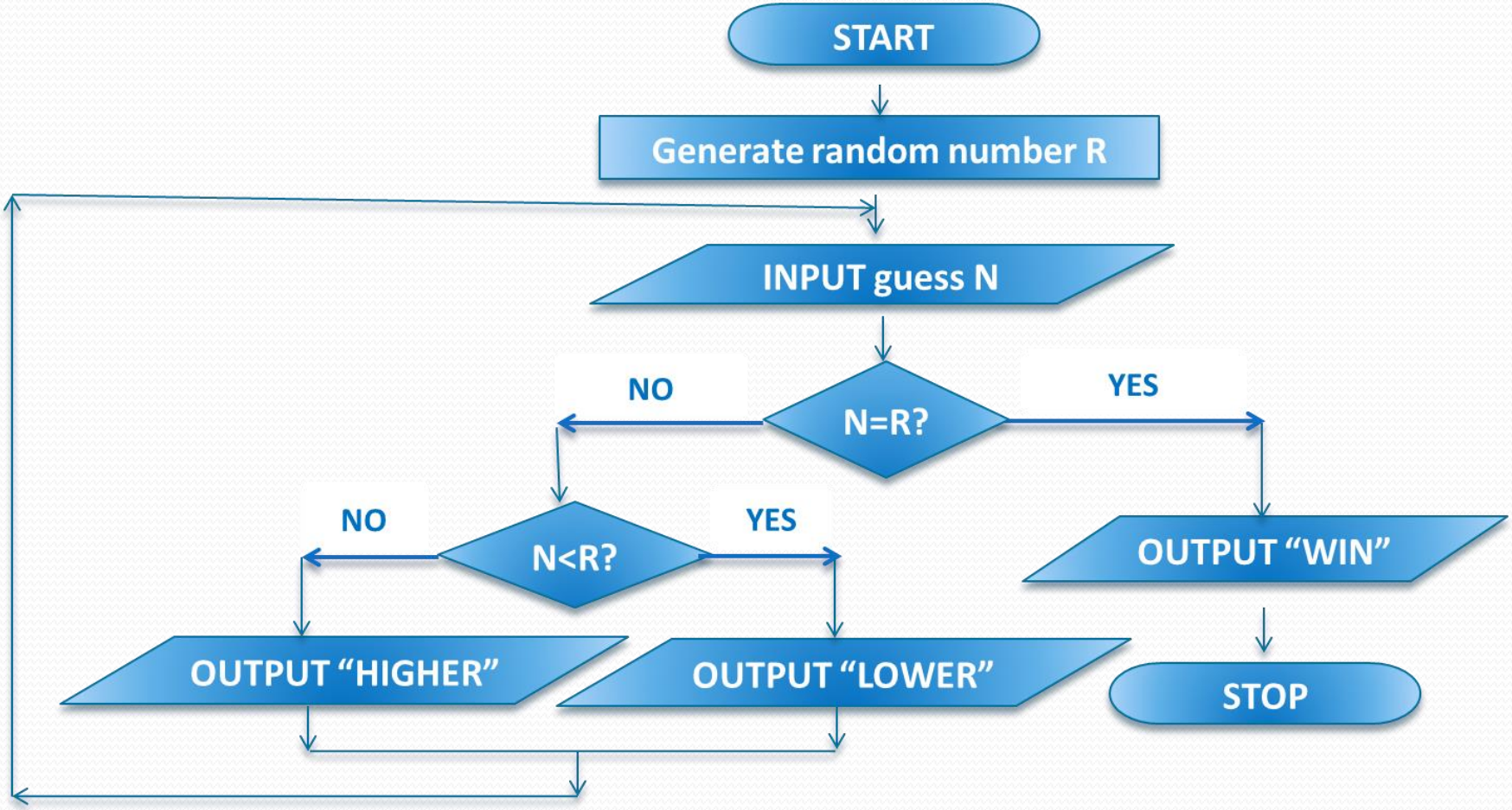
- Read valid number between 1 and 10

```
//initialize flag
bool IsValid = false;
//loop condition
while (!IsValid)
{
    //LOOP STATEMENTS
    //INPUT Number
    cout << "\n\nEnter an integral number: ";
    cin >> Number;
    //Validate the Number and update flag
    if ((1 <= Number) && (Number <= 10))
    {
        cout << "\nThe valid number is " << Number;
        IsValid = true;
    }
    else
        cout << "\n\nThe number is not valid! Try again!";
}
```

Example: Number Guessing Game

- Design an algorithm to play a number-guessing game
- Generate a random integer between 1 and 100. Then, prompt the player to guess the number.
- If the guessed number is the random number, the players win.
- If the guessed number is less that the random number output “Your guess is lower than the number”
- If the guessed number is higher that the random number output “Your guess is higher than the number”

Example: Number Guessing Game



Example: Number Guessing Game

- **rand** function generate a random int value between 0 and 32767

```
#include <cstdlib>
```

- **srand** function initializes the pseudo-random number generator is initialized using the argument passed as seed and allowing the pseudo-random number generator to generate a different succession of results in the subsequent calls to rand.
- To convert it to an integer between 1 and 100:

```
rand() % 100 + 1
```

Example: Number Guessing Game

```
int num;
int guess;
bool isGuessed;

srand(time(0));
num = rand() % 100+1;

isGuessed = false;

while (!isGuessed)
{
    cout << "Enter an integer between 1 and 100: ";
    cin >> guess;
    cout << endl;
```

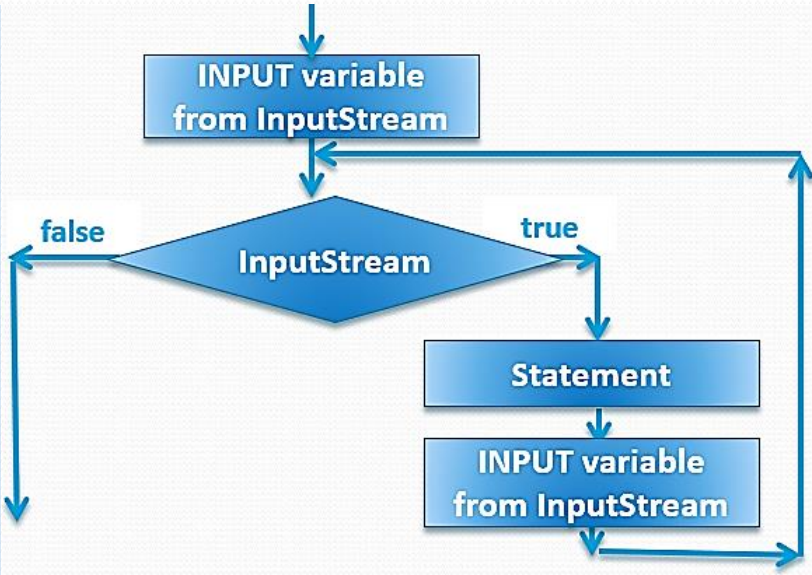
Example: Number Guessing Game

```
if (guess == num)
{
  cout << "You guessed the correct " << "number." <<
  endl;
  isGuessed = true;
}
else
if (guess < num)
    cout << "Your guess is lower than the number.\n Guess
again!"<< endl;
else
    cout << "Your guess is higher than the number.\n
Guess again!"<< endl;
}
```

Case 4: EOF-Controlled Loops

- Use an **EOF (End Of File)-controlled loop** to read from an input stream (file) as long as there is something in that stream

```
//initialize the loop control variable
InputStream >> variable;
//test the loop control variable
while (InputStream)
{
    //loop statements
    Statements;
    //update the loop control variable
     InputStream >> variable;
}
```



eof Function

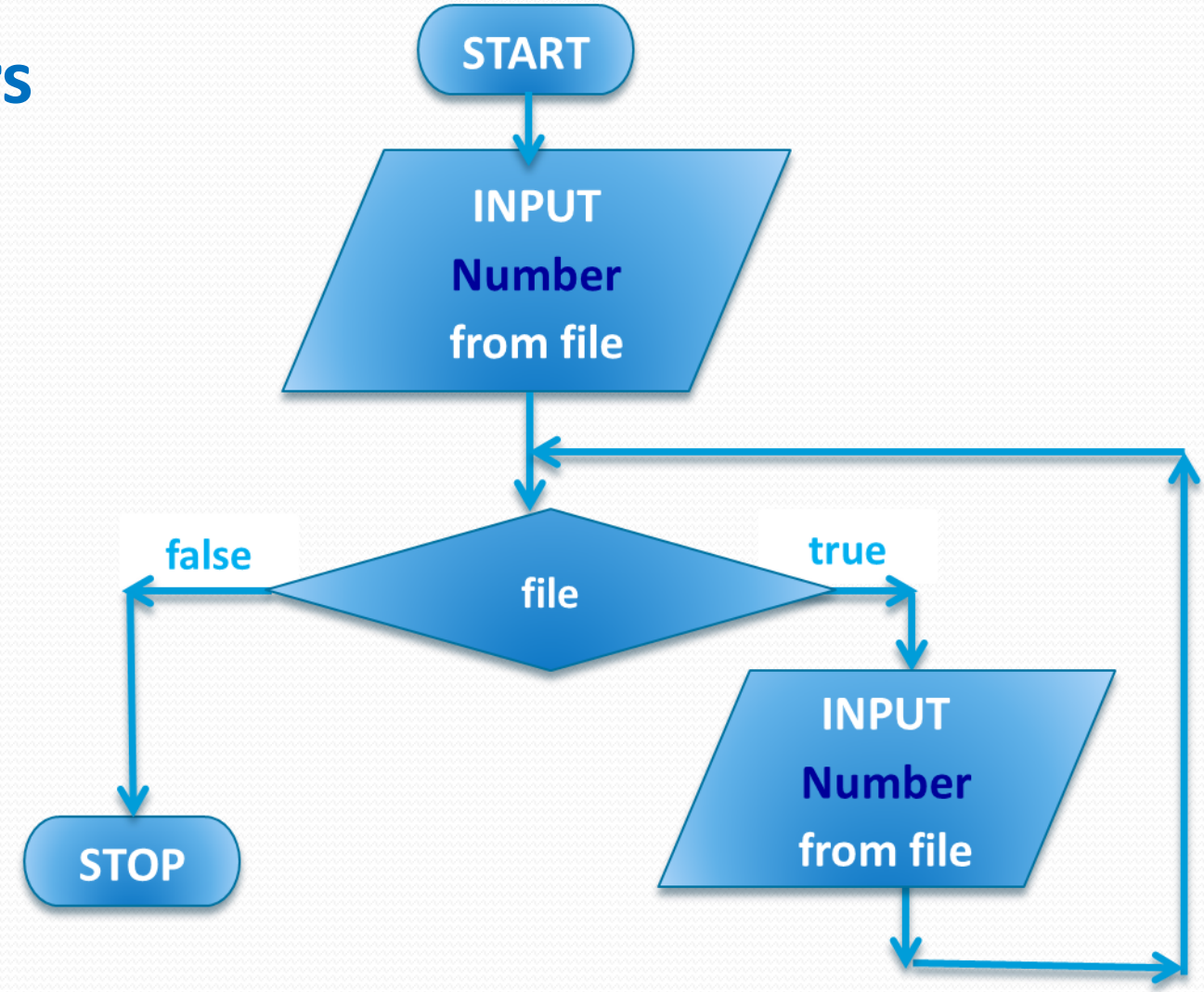
- The function `eof` can determine the end of file status
- Is `true` when you read beyond the end of the file
- `eof` is a member of data type `istream`
- The syntax for the function **`eof`** is:

```
istreamVar.eof()
```

where `istreamVar` is an input stream variable, such as `cin` or a input file variable

Case 4: EOF-Controlled Loops

- Read numbers from a file



Case 4: EOF-Controlled Loops

- **Read numbers from a file**

```
//open the file
ifstream in("InputFile.txt");
//read Number
int Number;
in >> Number;
//read from the file
while (in)
{
        //loop statement
        //output the Number
        cout << Number << "\n";
        //read another Number
        in >> Number;
}
//close the file
in.close();
```

Example: Grades

- Design an algorithm to calculate the letter grade and the class average for each student from a class of students that has taken one test worth 100 points
- Data consists of students' names and their test score

Example: Grades

	First Name	Last Name	TestScore	Letter Grade
$Student_1$				
$Student_2$				
$Student_3$				
$Student_4$				
$Student_5$				
...				
$Student_N$				
AVERAGE				

LetterGrade=
A if 90<=*TestScore*
B if 80<=*TestScore*<90
C if 70<=*TestScore*<80
D if 60<=*TestScore*<70
F if *TestScore*<60

For each student

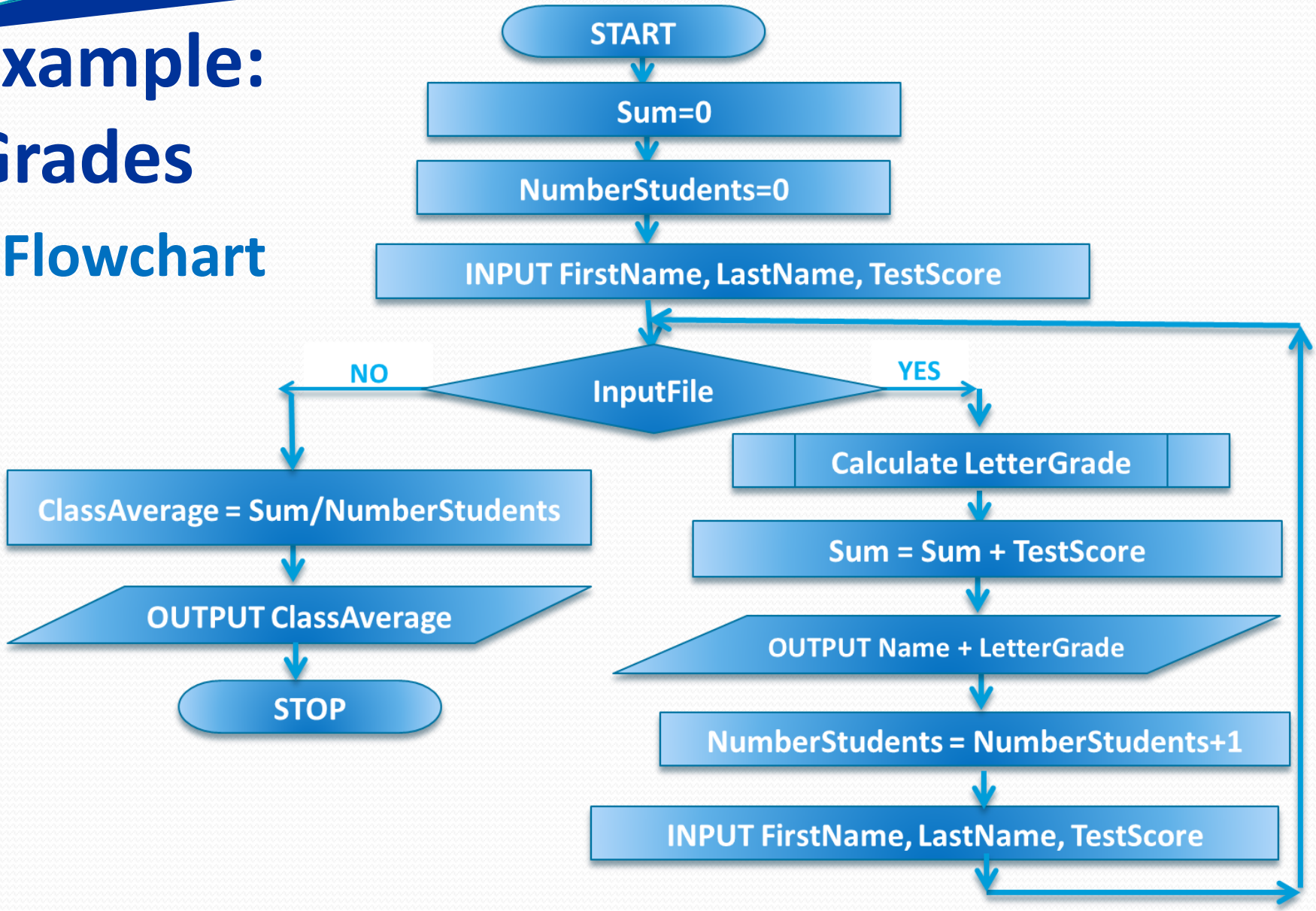
ClassAverage= $\frac{TestScore_{Student1} + \cdots ++ TestScore_{StudentN}}{N}$

$= \frac{\sum_{k=1}^{N} TestScore_{Studentk}}{N}$

For the class

Example: Grades

- Flowchart



Example: Grades

```
//Open input file
inFile.open("Ch5_stData.txt");
if (!inFile)
{
        cout << "Cannot open input file. Program terminates!\n";
return 1;
}
//Open output file
outFile.open("Ch5_stData.out");

outFile << fixed << showpoint << setprecision(2);

inFile >> firstName >> lastName;  //read the name
inFile >> testScore;        //read the test score
```

Example: Grades

```
while (inFile)
{
        sum = sum + testScore; //update sum
        count++;               //increment count

        //determine the grade
        switch (static_cast<int> (testScore) / 10)
        {
        case 0:  case 1: case 2: case 3: case 4: case 5:
                grade = 'F';  break;
        case 6: grade = 'D'; break;
        case 7: grade = 'C'; break;
        case 8: grade = 'B'; break;
        case 9: case 10: grade = 'A'; break;
        default: cout << "Invalid score." << endl;
} //end switch
```

Example: Grades

```
    outFile << left << setw(12) << firstName
    << setw(12) << lastName
    << right << setw(4) << testScore
    << setw(2) << grade << endl;

    inFile >> firstName >> lastName; //read the name
    inFile >> testScore;        //read the test score
}//end while
outFile << endl;

if (count != 0)
    outFile << "Class Average: " << sum / count << endl;
else
    outFile << "No data." << endl;
inFile.close();
outFile.close();
```

Exercise: EOF-Controlled Loop

- Read name and numbers from a file , compute the sum, average, maximum and minimum of the numbers and output the name and the number statistics in a formated table

```
//declare and open the file
ifstream in("InputFile.txt");
if (!in)
{
    cout << "\n\n\nCannot open the file!\n";
}
else
{//we did open the file!

    cout << "\n\n\nReading from the file!\n\n\n";

    //output table header
    cout << setw(10) << left << "NAME"
```

Exercise: EOF-Controlled Loop

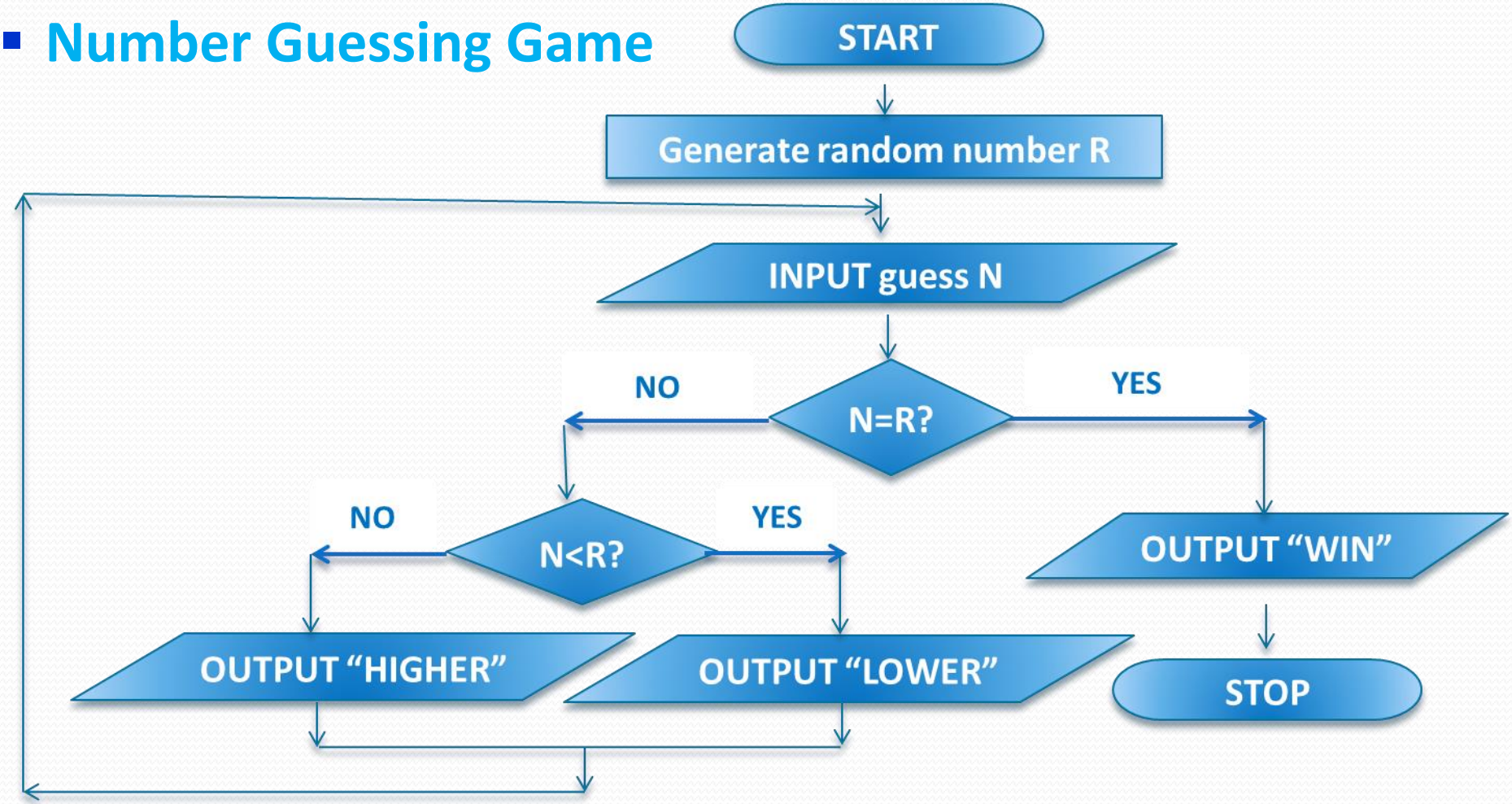
```
//output table header
cout<<setw(10)<<left<<"NAME“ <<setw(6)<<right<<"NUMBER"<<"\n";
//read from the file
string Name;
int Number;
int Sum = 0;
int NumberOfNumbers=0;
int Max = INT_MIN;
int Min = INT_MAX;
do
{
    //read line from file
    //read Name
    in >> Name;
    //check if we pass the eof
    if (in.eof())
    break;
    //read Number
    in >> Number;
```

Exercise: EOF-Controlled Loop

```
    //output table row
    cout <<setw(10)<<left<<Name <<setw(6)<<right<<Number <<"\n";
    //update statistics
    Sum = Sum + Number;
    NumberOfNumbers++;
    if (Max < Number)
        Max = Number;
    if (Min > Number)
        Min = Number;
}
while (in);
double Average = static_cast<float>(Sum) / NumberOfNumbers;
//output statistics
cout<<setw(10)<<left<<"SUM"<<setw(6)<<right<<Sum <<"\n";
cout<<setw(10)<<left<<"NUMBERS"<<setw(6)<<right<<NumberOfNumbers<<"\n";
cout<<setw(10)<<left<<"AVERAGE"<<setw(6)<<right<<Average<<"\n";
cout<<setw(10)<<left<<"MIN"<<setw(6)<<right<<Min<<"\n";
cout<<setw(10)<<left<<"MAX"<<setw(6)<<right<<Max<<"\n";
//close the file
in.close();
```

More on Expressions in `while` Statements

- Number Guessing Game



More on Expressions in `while` Statements

- The expression in a `while` statement can be complex

```
noOfGuesses=0;
while ((noOfGuesses < 5) && (!isGuessed))
{
    cout << "Enter an integer greater than or equal to 0
     and less than 100: ";
    cin >> guess;
    cout << endl;
    noOfGuesses++;

  …
}
```

- Combine different cases/types of loops

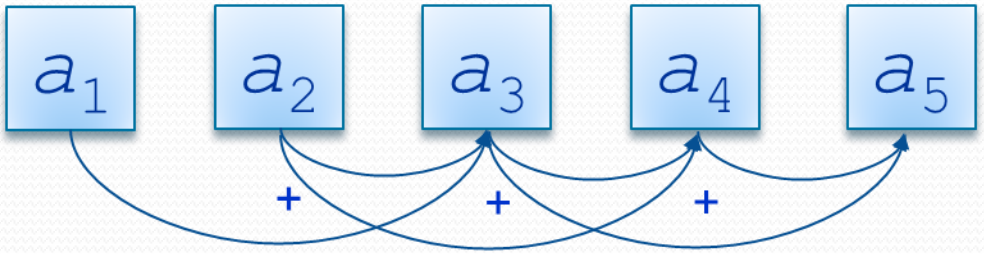
Types of Loops

Counter-Controlled Loop	Sentinel-Controlled Loop	Flag-Controlled Loop	EOF-Controlled Loop

`//initialize`
`counter = 0;`
`//test`
`while`
`(counter<N)`
`{`
`  //loop`
`  statements`
`  Statements;`
`  //update`
`  counter++;`
 	
`}`
	`//initialize`
	`cin >> var;`
	`//test`
	`while`
	`(var!=SENTINEL)`
	`{`
	`  //loop`
	`  statements`
	`  Statements;`
	`  //update`
	`  cin >> var;`
	
	`}`
		`//initialize`
		`found = false;`
		`//test`
		`while  (!found)`
		`{`
		`  //loop`
		`  statements`
		`  Statements;`
		`  //update`
		`  if`
		`  (expression)`
		`  found=true;`
		`}`
			`//initialize`
			`InStream>>var;`
			`//test`
			`while (InStream)`
			`{`
			`  //loop`
			`  statements`
			`  Statements;`
			`  //update`
			`  Instream>>var;`
			
			`}`

Programming Example: Fibonacci Number

- Consider the following sequence of numbers:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, ….
- Given the first two numbers of the sequence (say, a_1 and a_2 which are usually 1)
  - *n*th number a_n, $n >= 3$, of this sequence is given by: $a_n = a_{n-1} + a_{n-2}$



Programming Example: Fibonacci Number

- Input:
  - first two Fibonacci numbers
  - the desired Fibonacci number `n`
- Output: *n*th Fibonacci number

1. Get the first two Fibonacci numbers
2. Get the desired Fibonacci number n
3. Calculate the next Fibonacci number by adding the previous two elements of the Fibonacci sequence
4. Repeat Step 3 until the *n*th Fibonacci number is found
5. Output the *n*th Fibonacci number

Programming Example: Fibonacci Number

- Main Algorithm:
  1. Prompt the user for the first two numbers—that is, `previous1` and `previous2`
  2. Read (input) the first two numbers into `previous1` and `previous2`
  3. Prompt the user for the position of the desired Fibonacci number
  4. Read the position of the desired Fibonacci number into `nthFibonacci`

Programming Example: Fibonacci Number

6. a. `if (nthFibonacci == 1)`

   The desired Fibonacci number is the first Fibonacci number.
   Copy the value of `previous1` into `current`

   b. `else if (nthFibonacci == 2)`

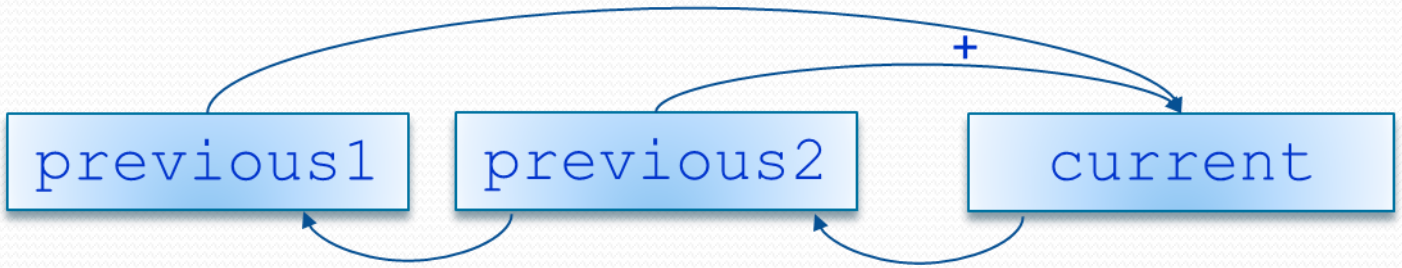
   The desired Fibonacci number is the second Fibonacci number.
   Copy the value of `previous2` into `current`.

   c. `else` calculate the desired Fibonacci number as follows:

   - Initialize `counter` to `3` to keep track of the calculated Fibonacci numbers.

Programming Example: Fibonacci Number

- Calculate the next Fibonacci number, as follows:
  `current = previous2 + previous1;`
- Assign the value of `previous2` to `previous1`
- Assign the value of `current` to `previous2`
- Increment `counter`
- Repeat until Fibonacci number is calculated



7. Output the nth Fibonacci number, which is current

Programming Example: Fibonacci Number

- Variables

```
int previous1;   //variable to store the first Fibonacci number
int previous2;   //variable to store the second Fibonacci number
```

```
int current;     //variable to store the current
                 //Fibonacci number
int counter;     //loop control variable
int nthFibonacci; //variable to store the desired
                  //Fibonacci number
```

Programming Example: Fibonacci Number

```
cout << "Enter the first two Fibonacci "
     << "numbers: ";                                  //Step 1
cin >> previous1 >> previous2;                        //Step 2
cout << endl;

cout << "The first two Fibonacci numbers are "
     << previous1 << " and " << previous2
     << endl;                                         //Step 3

cout << "Enter the position of the desired "
     << "Fibonacci number: ";                         //Step 4
cin >> nthFibonacci;                                  //Step 5
cout << endl;

if (nthFibonacci == 1)                                //Step 6.a
    current = previous1;
else if (nthFibonacci == 2)                           //Step 6.b
    current = previous2;
```

Programming Example: Fibonacci Number

```
else                                                    //Step 6.c
{
    counter = 3;                                        //Step 6.c.1

          //Steps 6.c.2 - 6.c.5
    while (counter <= nthFibonacci)
    {
        current = previous2 + previous1;                //Step 6.c.2
        previous1 = previous2;                          //Step 6.c.3
        previous2 = current;                            //Step 6.c.4
        counter++;                                      //Step 6.c.5
    }//end while
}//end else

cout << "The Fibonacci number at position "
     << nthFibonacci << " is " << current
     << endl;                                           //Step 7
```

for Looping (Repetition) Structure

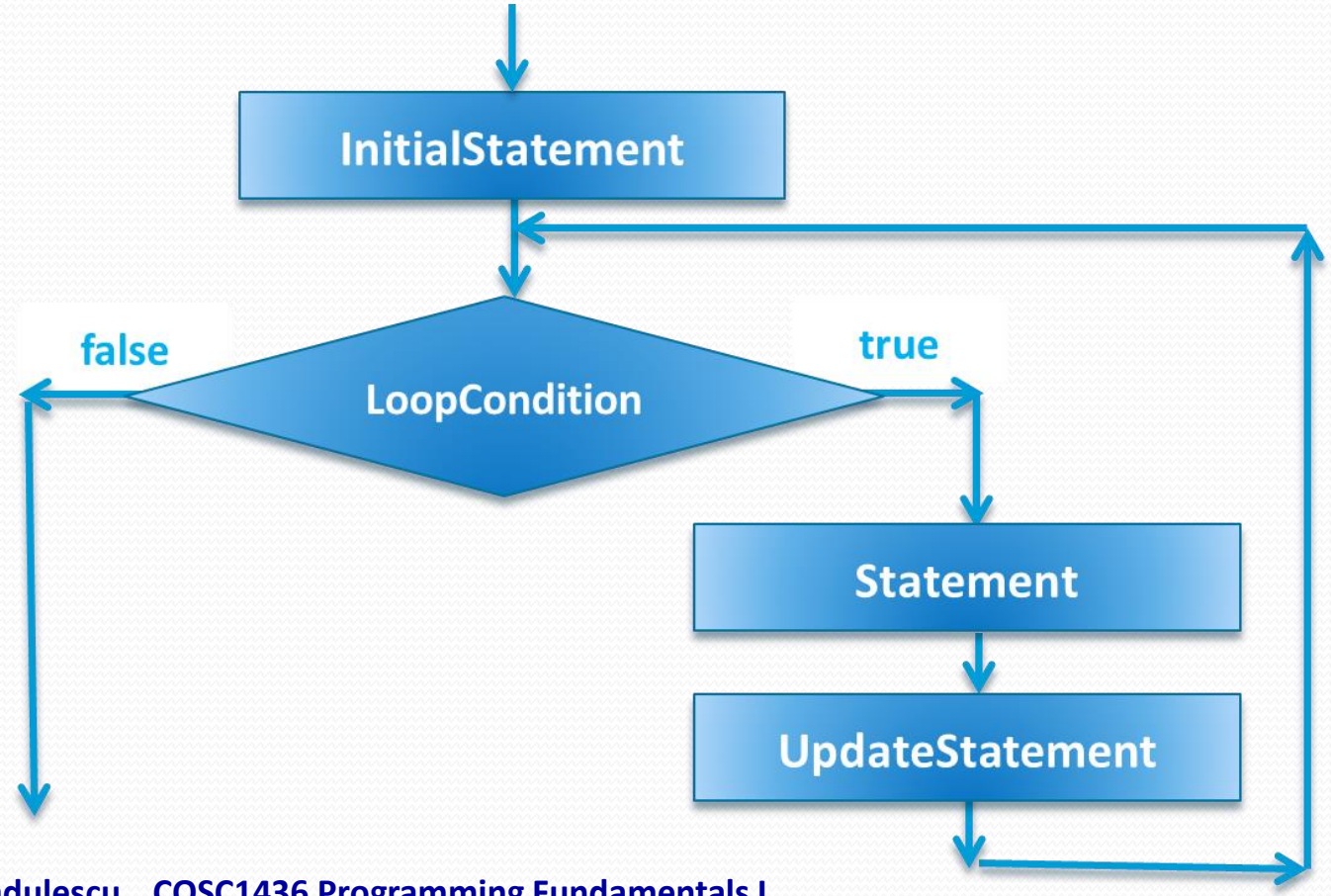
- The general form of the **for** statement is:

```
for( InitialStatement; LoopCondition; UpdateStatement )
   Statement;
```

- The InitialStatement, LoopCondition, and UpdateStatement are called for loop control statements
  - InitialStatement usually initializes a variable (called the **for loop control**, or **for indexed, variable**)
- In C++, for is a reserved word

for Looping (Repetition) Structure

```
for( InitialStatement; LoopCondition; UpdateStatement )
   Statement;
```



for Looping (Repetition) Structure

```
for (i=0; i<10; i++)
   cout << i << " ";
cout << endl;
```

Sample output:

```
0 1 2 3 4 5 6 7 8 9
```

for Looping (Repetition) Structure

for loop

```
for (i=0; i<10; i++)
  cout << i << " ";
cout << endl;
```

while loop

```
i=0;
while (i<10)
{
  cout << i << " ";
  i++;
}
cout << endl;
```

for Looping (Repetition) Structure

```
for(i=0; i<10; i++)
  cout << "Hello";
  cout << "*";
```

simple statement

```
for(i=0; i<10; i++)
{
   cout << "Hello";
   cout << "*";
}
```

compound statement

for Looping (Repetition) Structure

```
for(i=0; i<10; i++)
{
   cout << "Hello";
   cout << " * ";
}
```

10 times

```
for(i=0; i<10; i++);
{
   cout << "Hello";
   cout << " * ";
}
```

1 time

`for` Looping (Repetition) Structure

- The following is a legal `for` loop:

```
for (;;)
   cout << "Hello\n";
```

- The following is a legal `while` loop:

```
while (1)
   cout << "Hello\n";
```

- Both loops are legal, but not semantically correct.
  - They are **infinite loops** (never end).

`for` Looping (Repetition) Structure

- C++ allows you to use fractional values for loop control variables of the `double` type
  - Results may differ. Avoid using them.
- An semicolon before the body of the loop will terminate it.

```
for (i=0;i<5;i++);
      cout << "*" << endl;
```

- If the `initial statement`, `loop condition`, or `update statement` are omitted (empty) they are assumed to be true and the loop is an infinite loop.

```
for (;;)
      cout << "*" << endl;
```

for Looping (Repetition) Structure

- Not necessarily in increasing order – you can decrement the loop control variable

```
for (i=10; i>0; i--)
   cout << i << " ";
cout << endl;
```

```
for (i=10; i>=1; i--)
   cout << i << " ";
cout << endl;
```

Sample output:

```
10 9 8 7 6 5 4 3 2 1
```

`for` Looping (Repetition) Structure

- Not necessarily consecutive values for the loop control variable

```
for (i=1; i<=10; i=i+2)
   cout << i << " ";
cout << endl;
```

Sample output:

```
1 3 5 7 9
```

```
for (i=2; i<=10; i=i+2)
   cout << i << " ";
cout << endl;
```

Sample output:

```
2 4 6 8 10
```

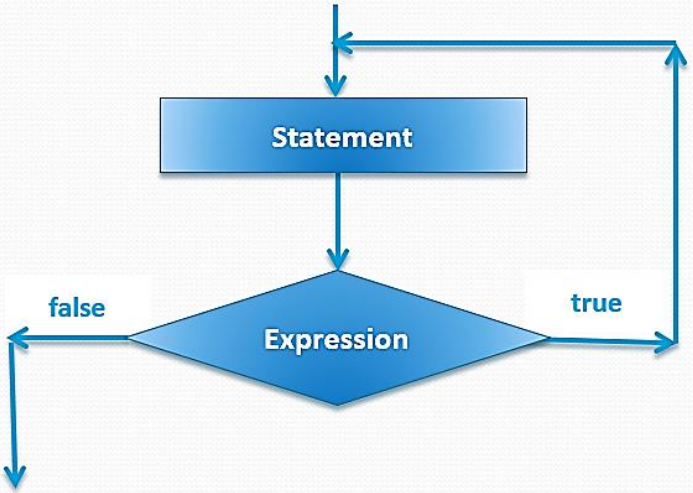
Example: Fibonacci Number

```
int previous1;
int previous2;
int current;
int nthFibonacci;

cout << "Enter the first two Fibonacci numbers: ";
cin >> previous1 >> previous2;
cout << endl;

cout << "The first two Fibonacci numbers are " << previous1 << " and " << previous2 << endl;

cout << "Enter the position of the desired Fibonacci number: ";
cin >> nthFibonacci;
cout << endl;

if (nthFibonacci == 1)
    current = previous1;
else if (nthFibonacci == 2)
    current = previous2;
else
{
    for (int counter = 3; counter <= nthFibonacci; counter++)
    {
        current = previous2 + previous1;
        previous1 = previous2;
        previous2 = current;
    }//end for
}//end else
cout << "The Fibonacci number at position " << nthFibonacci << " is " << current << endl;
```

do…while Looping (Repetition) Structure

- General form of a **do…while**:

```
do
        Statement;
while ( Expression );
```



- The `statement` executes first,

and then the `expression` is evaluated

- To avoid an infinite loop, body must contain a statement that makes the expression `false`
- The `statement` can be simple or compound
- Loop always iterates at least once

do…while Looping (Repetition) Structure

```
i=0;
do
{
   cout << i << " ";
   i=i+5;
}
while (i<=20);
cout << "\n";
```

Sample run:

```
0 5 10 15 20
```

5 times

```
i=25;
do
{
   cout << i << " ";
   i=i+5;
}
while (i<=20);
cout << "\n";
```

Sample run:

```
25
```

1 times

do…while Looping (Repetition) Structure

```
i=25;
do
{
   cout << i << " ";
   i=i+5;
}
while (i<=20);
cout << "\n";
```

Sample run:

```
25
```

1 times

```
i=25;
while (i<=20)
{
   cout << i << " ";
   i=i+5;
}
cout << "\n";
```

Sample run:

0 times

Looping (Repetition) Structure

while

```
while (LoopCondition)
      LoopStatement;
```

As a for

```
for ( ; LoopCondition; )
     LoopStatement;
```

As a do while

```
do
{
   if (LoopCondition)
      LoopStatement;
}
while (LoopCondition);
```

do while

```
do
      LoopStatement;
while (LoopCondition);
```

As a while

```
LoopStatement;
while (LoopCondition)
   LoopStatement;
```

As a for

```
LoopStatement;
for ( ; LoopCondition; )
     LoopStatement;
```

for

```
for ( InitialStatement;
LoopCondition;
UpdateStatement )
     LoopStatement;
```

As a while

```
InitialStatement;
while (LoopCondition)
{
   LoopStatement;
   UpdateStatement;
}
```

As a do while

```
InitialStatement;
do
     if (LoopCondition)
     {
          LoopStatement;
          UpdateStatement;
     }
while (LoopCondition);
```

Choosing the Right Looping Structure

- If you know or can determine in advance the number of repetitions needed, the **`for` loop** is the correct choice
- If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a **`while` loop**
- If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a **`do...while` loop**

for	while	do while

`for ( InitialStatement;`
`LoopCondition;`
`UpdateStatement )`
`LoopStatement;`
	`while (LoopCondition)`
	`LoopStatement;`
		`do`
		`LoopStatement;`
		`while (LoopCondition);`

Choosing the Right Looping Structure

Counter-Controlled Loop	Sentinel-Controlled Loop	Flag-Controlled Loop	EOF-Controlled Loop

`counter = 0;`
`while (counter<N)`
`{`
`  LoopStatements;`
`  counter++;`
`}`
	`cin >> var;`
	`while (var!=SENTINEL)`
	`{`
	`  LoopStatements;`
	`  cin >> var;`
	`}`
		`found = false;`
		`while (!found)`
		`{`
		`  LoopStatements;`
		`  if (expression)`
		`  found=true;`
		`}`
			`InStream>>var;`
			`while (InStream)`
			`{`
			`  LoopStatements;`
			`  Instream>>var;`
			`}`

for	do while	while

`for ( InitialStatement;`
`LoopCondition;`
`UpdateStatement )`
`  LoopStatement;`
	`do`
	`  LoopStatement;`
	`while (LoopCondition);`
		`while (LoopCondition)`
		`  LoopStatement;`

`break` and `continue` Statements

- `break` and `continue` alter the flow of control
- **`break`** statement is used for two purposes:
  - To exit early from a loop
    - Can eliminate the use of certain (flag) variables
  - To skip the remainder of the `switch` structure
- After the `break` statement executes, the program continues with the first statement after the structure

`break` and `continue` Statements

- **`continue`** is used in `while`, `for`, and `do…while` structures
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop

`break` and `continue` Statements

With break

```
for (i=0; i<10; i++)
{
  if (i==5)
     break;
  cout << " " << i;
}
```

```
 0 1 2 3 4
```

With continue

```
for (i=0; i<10; i++)
{
   if (i==5)
      continue;
   cout << " " << i;
}
```

```
0 1 2 3 4 6 7 8 9
```

`break` and `continue` Statements

Without break

```
done=false;
for (i=0; (i<10) &&
  (!done); i++)
{
  if (i==5)
      done=false;
  cout << " " << i;
}
```

```
 0 1 2 3 4
```

Without continue

```
for (i=0; i<10; i++)
{
   if (i==5)
      {}
else
   cout << " " << i;
}
```

```
0 1 2 3 4 6 7 8 9
```

Nested Control Structures

- Print N stars

```
for (j = 1; j <= N; j++)
        cout << "*";
```

- Print the numbers from 1 to 5, one per line

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

Nested Control Structures

- Create and print the following pattern:

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

- Code:

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

Avoiding Bugs by Avoiding Patches

- **Software patch**
  - Piece of code written on top of an existing piece of code
  - Intended to fix a bug in the original code
- Some programmers address the symptom of the problem by adding a software patch
- Should instead resolve underlying issue

Debugging Loops

- Loops are harder to debug than sequence and selection structures
- Use loop invariant
  - Set of statements that remains true each time the loop body is executed
- Most common error associated with loops is off-by-one

Summary

- C++ has three looping (repetition) structures: while, for, and do…while
- while and for loops are called pretest loops
- do...while loop is called a posttest loop
- while and for may not execute at all, but do...while always executes at least once

Summary

- while: expression is the decision maker, and the statement is the body of the loop
- A while loop can be:
  - Counter-controlled
  - Sentinel-controlled
  - Flag-controlled
  - EOF-controlled
- for loop: simplifies the writing of a counter-controlled while loop
- Executing a break statement in the body of a loop immediately terminates the loop
- Executing a continue statement in the body of a loop skips to the next iteration