Learning Objectives

At the end of this chapter, student should be able to:

- ✓ Understand the requirement of a loop
- ✓ Understand the Loop Control Variable ()
- ✓ Use increment (++) and decrement (--) operators
- ✓ Program loop with while, for and do-while statements
- ✓ Understand Counter-controlled, Sentinel-controlled and Flag-controlled structures
- ✓ Learn the basic algorithms commonly use in loops
- ✓ Understand the flow of control when using nested loop
- ✓ Learn the use of break and continue statements

A. INTRODUCTION TO ITERATION

- Many programs may require a repetition capability in which the same calculation or sequence of instruction is repeated over and over while certain condition remains true using sets of data.
- Examples of repetition:
    - Continual checking of data entries until an acceptable entry
    - Counting and accumulating running totals
    - Recalculation of output values that only stop upon entry of sentinel value.
- Each repetition is referred to as ***iteration*** or ***loop*** because of its cyclic nature.
- Why repetition is needed?
    - Suppose we want to add five numbers to find their average. From what we have learned so far, we could proceed as follows (assume that all the variables are properly declared)

```
cin >> num1 >> num2 >> num3 >> num4 >> num5;
sum = num1 + num2 + num3 + num4 + num5;
average = sum / 5.
```

    - How about if we want to add 100 or even 1000 numbers? This would be so tedious and complex.
    - Therefore, *repetition* or *loop* structure offers a better way of performing those tasks. The example below while input sets of number until we entered number less than 1.

```
while (num > 0)
{
        cin >> num;
        sum = sum + num;
        count = count + 1;    //count++;
}
average = sum / count;
```

1. Types of Repetition Statement

There are three types of repetition statement. There are:

a. `while`

b. `for`

c. `do…while`

2. Types of Repetition Structure

There two types of repetition structure. There are

a. Counter-controlled loop

b. Sentinel-controlled loop

c. Flag-controlled loop

3. Requirements of a Repetition Structure

- An iteration structure requires:
  - ***a. Loop control variable (LCV)***

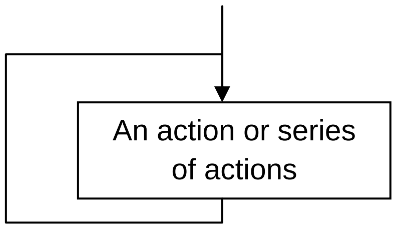
    A variable whose value determines whether the loop body will be executed or not.
  - ***b. Loop condition***

    If the condition is true, the loop body is executed; otherwise the loop exits.
  - ***c. Loop body***

    A block of statements to be repeated.

- Execution of the loop body is controlled by 3 operation:
  - a. Initialization of the loop control variable (LCV)
  - b. Evaluation of LCV in the loop condition
  - c. Update of the LCV by incrementing or decrementing

4. Increment and Decrement Operator



- Operator is placed after a variable to indicate the increment or decrement of one value from the variable.
- Example:
  - **a. x++**

    Use the current value of x in the expression in which x resides, and then increment by 1.

    x = x + 1 *is equivalent to* *x += 1* *or* *x++*

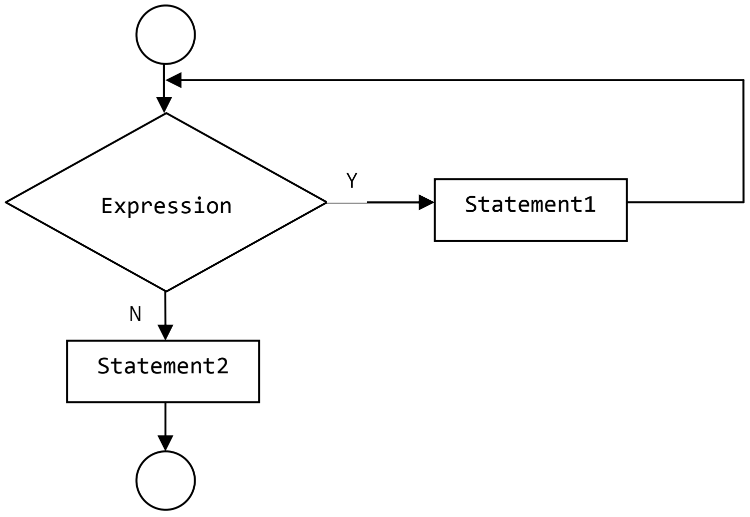
b. **x- -**
Use the current value of x in the expression in which x resides, then decrement by 1
x = x – 1 is equivalent to x -= or x--

B. `while` LOOP

- Expression is evaluated first and then executes the statement.
- If the expression is evaluate to true, it continue to be executed, and if the expression is evaluate to false, it exit the while loop.
- Syntax

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

- Flowchart

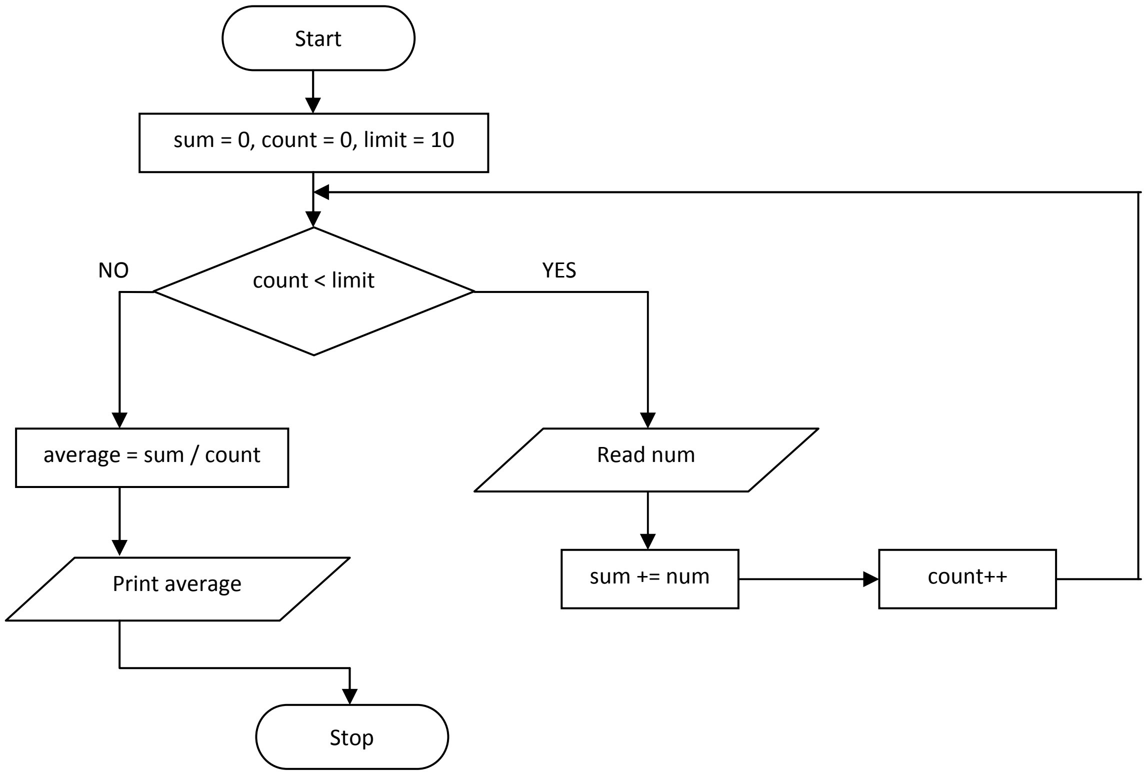


- `while` repetition structure uses three types of repetition structure. There are:
  a. Counter-controlled structure
  b. Sentinel-controlled structure
  c. Flag-controlled structure

1. Counter-Controlled Structure

1. Used when the number of time a segment code needs to be repeated is known in advance.
2. Example: Repeat a process for 10 times
3. Requirement:
   a. The name of a control variable or loop counter
   b. The initial value of the control variable
   c. The condition that test for the final value of the control variable
   d. The increment or decrement of the control variable each time loop is executed.

- It is also used by the other two iteration statements which are `do…while` and `for`.
- Example flowchart:



- Example C++:

```
#include <iostream>
void main()
{       int num, average, sum = 0, count = 0, limit = 10;

        while (count < limit)
        {       cout << "Enter any number ";
                cin >> num;
                sum += num;
                cout << "\ncount before increment = " << count;
                cout << "\nsum = " << sum;

                count++;
                cout << "\ncount after increment= " << count;
        }
        cout << "\nsum = " << sum;
        average = sum / count;
        cout << "Average for sum of ten integer numbers is " << average;
}
```

- Explanation of the program :

  The body of the while-loop

```
cout << “Enter any number ”;
cin >> num;
sum += num;
count++;
```

  will be repeated as long as the value of `count` is less or equal to the value of `limit`. If the value of `count` is greater than `limit`, the while-loop will exit and execute statement

```
average = sum / count;
cout << “Average for sum of ten integer numbers is ” << average;
```

- This program is called a counter-controlled structure because it will repeat the statement in the body of loop up to the specified number of iteration.

Using `while` loop to find the largest value

- In order to get the largest value of sequence of values entered, the first value entered is commonly treated as the largest. This value will be compared with the remaining values entered.
- **Example**: Write a complete C++ program to find the largest of sequence of integers. The number of integers is determined by user.

Problem definition

Input: number, totalNumber

Process: Accept numbers and compare to the largest

Output: largest

Program coding

```
#include <iostream.h>

void main()
{      int number, counter = 0, largest, totalNumber;

       cout << "How many numbers? "
       cin >> totalNumber;

       cout << "Enter any number";
       cin >> number;

       largest = number;

       while (counter < totalNumber-1)
       {      cout << "Enter any number ";
              cin >> number;
                     if (number > largest)
                            largest = number;
              counter++;
       }
       cout << "\n The largest numbers is " << largest;
}
```

Using the* `while` *loop to find the maximum and minimum values

- To find the maximum and minimum values, we must determine whether the data has a range of values or not.
- If there is a range of values, the minimum variable must be initialized to the largest value in the range and the maximum variable must be initialized to the smallest value in the range.
- If there is no range of values, both minimum and maximum variable are initialized to the first value entered by user.
- **Example of finding the minimum and maximum from numbers with unknown range of values**: Write a complete C++ program to find smallest and largest number from a sequence of five integers input from keyboard.

Problem definition
Input: Any 5 numbers
Process: Accept the number and determine smallest and largest
Output: maximum and minimum

`Program coding

```
#include <iostream.h>

void main()
{       int number, counter = 1, min, max;

        cout << "\n Enter any number ";
        cin >> number;

        min = number;
        max = number;

        while (counter < 5)
        {
                cout << "\n Enter any number ";
                cin >> number;

                if (number < min)
                        min = number;
                 if (number > max)
                        max = number;

                   counter++;

        }

        cout << "\n The largest numbers is " << max;
        cout << "\n The smallest numbers is " << min;
}
```

- **Example of finding the minimum and maximum vales from known range of values**: Write a complete C++ program to find the smallest and largest number from sequence of five numbers input from keyboard. The range of the numbers is from 0 to 100.

Problem definition

Input: Any 5 numbers

Process: Accept the numbers and determine smallest and largest

Output: maximum and minimum

Program coding

```
#include <iostream.h>

void main()
{      int number = 0, counter = 0, min = 100, max = 0;

       while (counter < 5)
       {
              cout << "\n Enter any number ";
              cin >> number;

              if (number < min)
                     min = number;
               if (number > max)
                     max = number;

                counter++;

       }

       cout << "\n The largest numbers is " << max;
       cout << "\n The smallest numbers is " << min;
}
```

2. Sentinel-Controlled Structure

- Used when the exact number of repetitions is not known in advance or the items are too numerous to count beforehand.
- Thus, some loops require the user to enter a special value to end the loop. This special value is called the sentinel value.
- The integer and character value can be used as a sentinel value.

Using integer as sentinel value in a while loop

- **Example:** Write a complete C++ program display ‘Incorrect pin number, please try again’ as long as the code entered is not 877.

```
#include <iostream.h.

int main()
{
       cout << “\nEnter your pin number”;
       cin >> number;

       while (number != 877)
       {
              cout << “\nIncorect pin number, please try again”;
              cout << “\nEnter your pin number”;
              cin >> number;
       }
       cout << “\nWelcome”
}
```

- **Example:** Write a complete C++ program to calculate the average score of a sequence of scores as long as the score entered is not -1.

Problem definition

Input: scores

Process: accepts scores, find the average

Output: average

Program coding

```
#include <iostream.h>
#include <conio.h>

int main()
{       int total = 0, scoreCounter = 0, score;
        float average;

        cout << "\n To stop entering scores, type -1 ";
        cout << "\n Enter score ";
        cin >> score;

        while (score != -1)
        {       total += score;       //total = total + score;
                scoreCounter++;       // scoreCounter = scoreCounter + 1;
                cout << "\n Enter score ";
                cin >> score;
        }

        if (scoreCounter > 1)
        {       average = total / scoreCounter;
                cout << "\n The average score is " << average << endl;
        }
        else
        {       cout << "\n No score has been entered ";
        }

        cout  << "\n End of program ";

getch();
return 0;
}
```

Using character input as sentinel value to count the number of occurrence in the while loop

- **Example:** Write a complete C++ program to count the number of scores entered by the user and display the total number of scores entered.

Problem definition

Input: scores

Process: count the number of scored entered

Output: numScore

Problem coding

```
#include <iostream.h>

void main()
{       char choice;
        int score, numScore = 0;

        cout << "Do you want to start? (Y/N): ";
        cin >> choice;

        while (choice != 'N' || choice != 'n')
        {       cout << "\n Enter a score: " ;
                cin >> score;

                numScore++;

                cout << "\n Continue? (Y/N): ";
                cin >> choice;
        }

        cout << "\n The number of scores entered: " << numScore << endl;
}
```

Using character input as sentinel value to calculate the sum in a while loop

- **Example:** Write a program that asks for quantity of items bought and the amount of each item for an unknown number of customers. The program will then print the total amount of money paid by all customers in a supermarket.

Problem definition

Input: quantity, priceItem

Process: calculate the total amount paid by all customer

Output: totalAll

Problem coding

```
#include <iostream.h>
#include <conio.h>

void main()
{      char nextCust = 'Y';
       double sum = 0.0, price = 0.0, amt = 0;
       int itemQty = 0;

       while (nextCust == 'Y'|| nextCust == 'y')
       {      cout << "\n Enter quantity and price " ;
              cin >> itemQty >> price;
              amt = itemQty * price;
              cout << "\n The price the customer must pay is RM" << amt;
              sum += amt;

              cout << "\n\n Next customer (Y/N):";
              cin >> nextCust;
       }
       cout << "\n Total amount paid by all customers is RM" << sum;
getch();
}
```

- **Example:** Write a complete C++ program to sum a list of positive values entered by user.

Problem definition

Input: positive number

Process: sum all the positive values entered

Output: sum

Program coding

```
#include <iostream.h>
#include <conio.h>

void main()
{      float number = 0, sum = 0.0;
       cout << "\n Enter a list of positive values (end with -1.0)";
       cin >> number;
       while (number > 0.0)
       {      sum +=number;
              cin >> number;
       }
       cout << "\n Sum is " << sum << endl;
getch();
}
```

3. Flag-Controlled Structure

- Uses a `bool` variable to control the loop.
- General form of flag-controlled structure:

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

- The variable `found` above, which is used to control the execution of the while-loop is called **flag** variable.

Using flag-controlled variable to check for password

- **Example:** Write a program using flag-controlled variable that will prompt user for a password, accept those password and display message `Password accepted` if the user has entered the correct password. Otherwise, the program will display “`Incorrect password`”. This program will loop until the user entered the correct password.

Problem definition

Input: sequence of character (password)

Process: verify the password

Output: Message

Program coding (type 1)

```
#include <iostream.h>
#include <string.h>
void main()
{       char password[7], myPassword[7] = {"secret"};
        bool pass = true;
        while (pass)
        {       cout << "\n Enter your password ";
                cin >> password;
                if (strcmp(password,myPassword) == 0)
                {       pass = false;
                }
                else
                {       cout << "\n\a Incorrect password";
                }
        }
        cout << "\n Password accepted";
}
```

Program coding (type 2)

```
#include <iostream.h>
#include <string.h>

void main()
{       char password[7], myPassword[7] = {"secret"};
        bool pass = false;

        while (!pass)
        {       cout << "\n Enter your password ";
                cin >> password;

                if (strcmp(password,myPassword) == 0)
                {       pass = true;
                }
                else
                {       cout << "\n\a Incorrect password";
                }
        }
        cout << "\n Password accepted";
}
```

Using flag-control variable to calculate the sum in a while loop

- **Example:** Write a program that asks for quantity of items bought and the amount of each item for an unknown number of customers. The program will then print the total amount of money paid by all customers in a supermarket.

Problem definition

Input: quantity, priceItem

Process: calculate the total amount paid by all customer

Output: totalAll

Program coding (Style 1)

```
#include <iostream.h>
#include <conio.h>
void main()
{      char nextCust = 'Y', inputNextCust;
       bool code = true;
       double sum = 0.0, price = 0.0, amt;
       int itemQty;

       while (code)
       {      cout << "\n Enter quantity and price " ;
              cin >> itemQty >> price;
              amt = itemQty * price;
              cout << "\n The price the customer must pay is RM" << amt;
              sum += amt;
              cout << "\n\n Next customer (Y/N):";
              cin >> inputNextCust;

              if (inputNextCust != nextCust)
                     code = false;
     }
     cout << "\n Total amount paid by all customers is RM" << sum;

getch();
}
```

Program coding (Style 2)

```
#include <iostream.h>
#include <conio.h>
void main()
{      char nextCust = 'Y', inputNextCust;
       bool code = false;
       double sum = 0.0, price = 0.0, amt;
       int itemQty;

       while (!code)
       {      cout << "\n Enter quantity and price " ;
              cin >> itemQty >> price;
              amt = itemQty * price;
              cout << "\n The price the customer must pay is RM" << amt;

              sum += amt;

              cout << "\n\n Next customer (Y/N):";
              cin >> inputNextCust;
                     if (inputNextCust != nextCust)
                     code = true;
     }
     cout << "\n Total amount paid by all customers is RM" << sum;
getch();}
```

`break` ***Statement***

- So far, the break keyword found is in the switch statement. The break statement can also be used in the repetition statements. The `break` causes an immediate exit from enclosing loop or switch statement.
- An example of `break` statement usage:

```
while (expression)
{
       statement-1;
       if (expression)
              break; ──────┐
       statement-2;        │
}                          │
statement-3; ◄─────────────┘
```

Upon evaluating the expression to true, the `break` *statement will exit from the* `while` *loop resulting the* `statement-2` *to be skipped. Next* `Statement-3` *will be executed.*

- Example:

```
int x;
for (x = 1; x <= 10; x++)
{
       if (x == 5)
       break;

       cout << x << " ";
}
cout << "\nThe loop terminated at x = " << x;
```

```
//Sample output
1 2 3 4
The loop terminated at x = 5
```

```
int x = 1;
while ( x <= 10)
{
       if (x==5)
       break;

       cout << x << " ";
       x++;
}
cout << "\nThe loop terminated at x = " << x;
```

```
/*What is the output of this
statement*/
```

```
int x = 1;
do
{      if (x==5)
       break;

       cout << x << " ";
       x++;
} while (x <= 10);
cout << "\nThe loop terminated at x = " << x;
```

```
/*What is the output of this
statement*/
```

`continue` ***Statement***

- The `continue` statement skip the remaining statements on the loop and proceeds with the next loop.
- It is used in a loop structure to alter the flow of control or to check for the incorrect data. Note that the `continue` statement may only used in `while`, `for` and `do-while` loops.
- An example of `continue` statement usage:

```
while (expression)
{
       statement-1;
       if (expression)
              continue;
       statement-2;
}
statement-3;
```

If the expression in the `while` *loop is evaluated to true,* `statement-1` *will be executed. Next if the expression in the* `if` *statement yields true, the* `continue` *statement cause the operation to re-executed the* `while` *loop.*

- Example:

```
int x;
for (x = 1; x <= 10; x++)
{
       if (x == 5)
              continue;
       cout << x << " ";
}
cout << "\nContinue statement causes x=5 not printed";
```

```
//Sample output
1 2 3 4 6 7 8 9 10
Continue statement causes
x=5 not printed
```

```
int x = 1;
while ( x <= 10)
{
       if (x == 5)
              continue;
       cout << x << " ";
       x++;
}
cout << "\nContinue statement causes x=5 not printed";
```

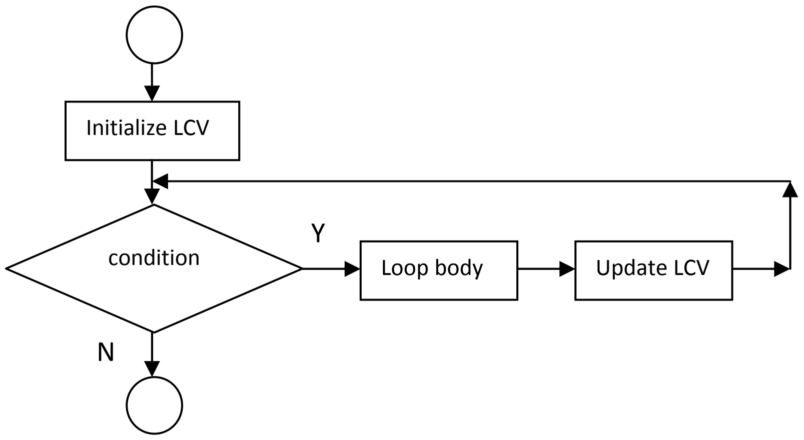
```
/*What is the output of
this program? This program
will execute infinitely
when x == 5. Modify this
program so that the output
will be the same as above
(for loop)*/
```

```
int x = 1;
do
{
       if (x == 5)
              continue;
       cout << x << " ";
       x++;
} while ( x <= 10);
cout << "\nContinue statement causes x=5 not printed";
```

```
/*What is the output of
this program? This program
will execute infinitely
when x == 5. Modify this
program so that the output
will be the same as above
(for loop)*/
```

C. `for` LOOP

- `for` statement is a counter-controlled loop in which a loop variable manages the repetition through counting. The advantages of using a for loop is described below:
  1. **Initialize loop control variable** is evaluated first before any iteration occurs.
  2. **Expression** is evaluated immediately after the initialization. It determines whether the loop should continue iterating. If it is true, the loop body is executed.
  3. **Update loop control variable**. The update expression is used to update the loop control variable, and it is evaluated after the loop body is executed.
- Flowchart:



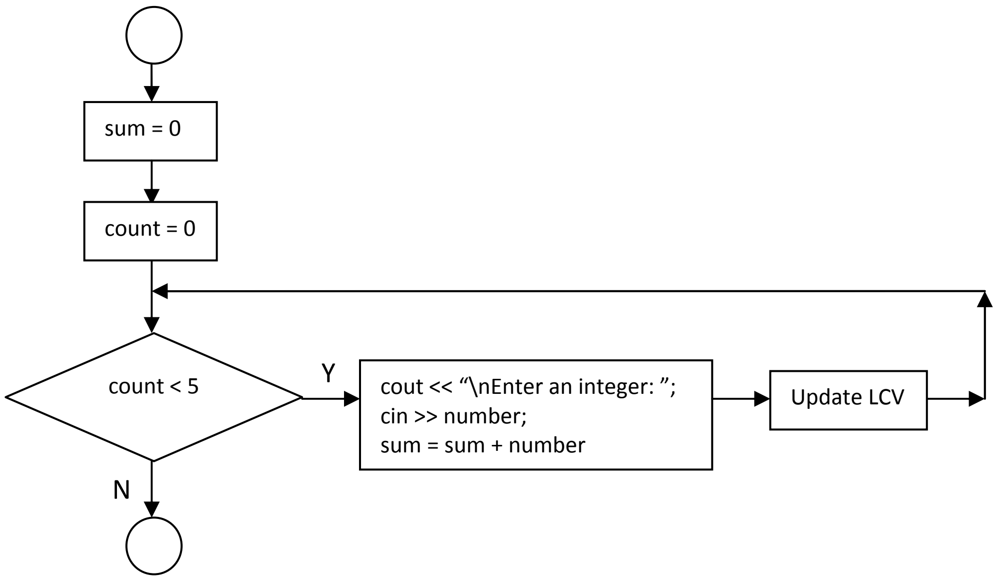
Using `for` loop to calculate sum

Example: The following example shows the use of `for` loop to calculate sum.

```
int sum = 0;
for (int count = 0; count < 5; count++)
{
       cout << "\nEnter an integer: ";
       cin >> number;
       sum = sum + number;
}
cout << "\nThe sum is " << sum;
```

```
//sample output
Enter an integer: 8
Enter an integer: 5
Enter an integer: 6
Enter an integer: 4
Enter an integer: 2
The sum is 25
```

It is equivalent to the flowchart given below:



Example: Write a complete C++ program to find the sum of all the even integers from 2 to 20 using `for` statement.

Problem Definition

Input: No input

Process: Sum up all even integers from 2 to 20

Output: Sum

Program Coding

```
#include <iostream.h>

void main()
{       int number, sum = 0;
        for (number = 2; number <= 20; number += 2)
               sum += number;
        cout << "The sum of all even numbers between 2 to 20 is " << sum;

}
```

Example: Write a program that accepts a series of genders from a class of 10 and counts the number of female in the class.

Problem definition

Input: Gender

Process: Count the number of female

Output: Number of female student

Program coding

```
#include <iostream.h>
#include <ctype.h>

void main()
{       char gender;
        int countFemale = 0;
        for (count = 0; count < 30; count++)
        {       cout << “\nEnter gender (F-female, M-male): ”;
                cin >> gender;
                if (toupper (gender) == ‘F’)
                countFemale++;
        }
        cout << “\n The number of female students are: ” << countFemale;
}
```

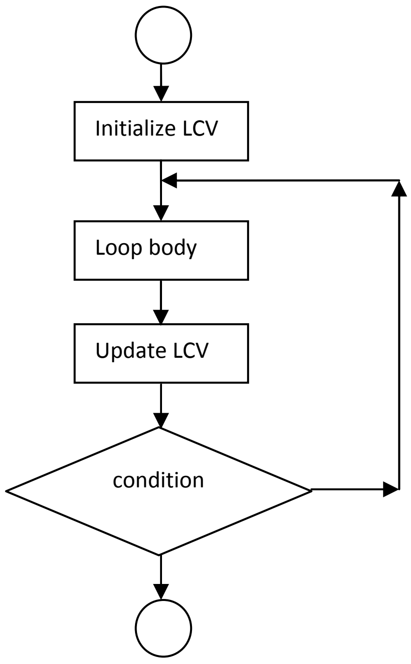
D. `do…while` **LOOP**

- `do…while` loop employs a posttest loop, testing the loop condition after each loop iteration. The `do…while` loop will execute at least once.
- Syntax form:

```
do
{
        Loop body;
} while (condition);
```

- The `do…while` statement is similar to the `while` statement except that its condition is evaluated at the end of the loop and not at the beginning. This means that a `do…while` loop will always iterate at least once, regardless of the value of its control condition.

- Flowchart



Using* `do…while` *loop to calculate total

The following example shows the use of the `do…while` loop to calculate total.

- **Example:** Write a complete C++ program to find the total salary of 5 employees entered by user.

Problem definition

Input: Salary

Process: Accept salary and accumulate total salary

Output: Total Salary

Program coding

```
#include <iostream.h>

void main()
{      int salary, totalSalary = 0, count = 0;

       do
       {      cout << "Enter salary: ";
              cin >> salary;
              totalSalary = totalSalary + salary;
              count++;
       } while (count < 5);
       cout << "The total salary is RM" << totalSalary;
}
```

- **Example:** Write a C++ program using `do…while` loop that accepts a valid input for salary within the range of 1000 and 3000 and display the total salary for the valid salary entered.

Problem definition

Input: Salary

Process: Accept and accumulate salary as long as the salary entered is within the range of 1000 and 3000

Output: Total salary

Program coding

```
#include <iostream.h>

void main()
{      int totalSalary = 0, salary = 0;

       do
       {      totalSalary = totalSalary + salary;
              cout << "Enter salary: ";
              cin >> salary;
       } while (salary > 1000 && salary < 3000);
       cout << "The total salary is RM" << totalSalary;
}
```

E. Summarized of `while`, do…`while` and `for`

PSEUDOCODE	***C++ STATEMENT***

`//DOWHHILE using counter-controlled loop`
 	
`sum (int) = 0`
`bil (int) = 1`
 	
`1. Start`
`2. DOWHILE bil <=5`
`      sum += bil`
`      bil++`
`   ENDDO`
`3. Stop`
	`//while statement using`
	`//counter-controlled loop`
	
	`int sum = 0;`
	`int bil = 1;`
	
	`while (bil <= 5)`
	`{      sum+= bil;`
	`       bil++;`
	`}`

`//REPEAT…UNTIL using counter-controlled loop`
 	
`sum (int) = 0`
`bil (int) = 1`
 	
`1. Start`
`2. REPEAT`
`      sum += bil`
`      bil++`
`   UNTIL bil > 5`
`3. Stop`
	`//do-while statement using`
	`//counter-controlled loop`
	
	`int sum = 0;`
	`int bil = 1;`
	
	`do`
	`{      sum += bil;`
	`       bil++;`
	`} while (bil <=5)`

`//DO loop_index = initial_value TO final_value`
`//This statement is also equivalent to while`
`//and do…while loop that using`
`//counter-controlled loop in C++ statement`
 	
`sum (int) = 0`
`bil (int) = 1`
 	
`1. Start`
`2. DO bil = 1 TO 5`
`      sum += bil`
`   ENDDO`
`3. Stop`
	`//for statement`
	
	`int sum = 0;`
	`for (int bil = 1; bil <=5; bil++)`
	`       sum +=bil;`

F. Infinite Loop

- A loop is called infinite when the loop repeats itself repeatedly with no end. The program does not work as expected.
- Example:

PSEUDOCODE	**C++**

`Infinite loop`
 	
`x (int) = 0`
 	
`1. Start`
`2. DOWHILE x == 0`
`      Print "No exit!"`
`      x = 0`
`   ENDO`
`3. Stop`
	`Infinite loop`
	
	`int x = 0;`
	
	`while (x == 0)`
	`{      cout << "No exit!";`
	`       x = 0;`
	`}`

G. NESTED LOOP – (Concentrate on nested loop using `for` only)

- A loop may be executed within another loop. An inner loop is considered as statements in the body of another outer loop.
- Syntax:

PSEUDOCODE	C++

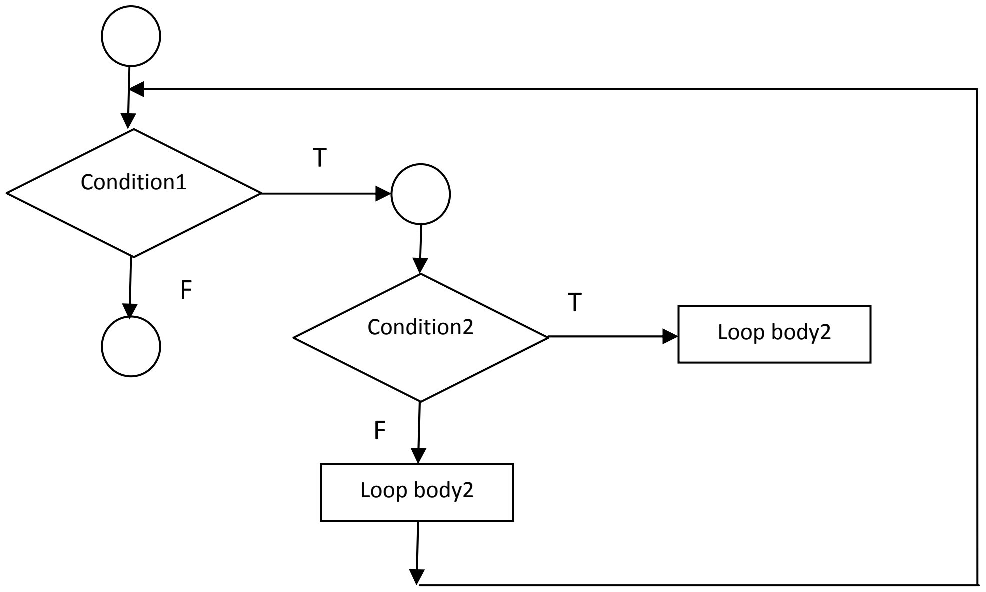
1. Start
2. DOWHILE condition
       DO index = initial TO final
               Loop body1
       ENDDO
       Loop body2
   ENDDO
3. Stop
	while (condition1)
	{ for (initial; condition2; update)
	       { loop body1;
	       }
	       Loop body2;
	}

1. Start
2. DOWHILE condition1
       DOWHILE condition2
               Loop body1
       ENDDO
       Loop body2
   ENDDO
3. Start
	while (condition1)
	{ while (condition2)
	       { loop body1;
	       }
	       Loop body2;
	}

1. Start
2. DO index1 = initial TO final
       DO index = initial2 TO final
               Loop body1
       ENDDO
       Loop body2
   ENDDO
3. Stop
	for (initial; condition1; update)
	{ for (initial; condition2; update)
	       { Loop body1;
	       }
	       Loop body2;
	}

- Explanation:
  - The outer loop will be tested and executed first. If the condition1 is TRUE, the inner loop will then be tested.
  - If the inner loop is TRUE the loop body 1 (statement) will be executed until the condition is no longer TRUE.
  - If the condition in inner loop no longer TRUE, it exit the loop and execute the loop body2 (statement) before passing back to outer loops.
  - The same process will be repeated until condition1 in the outer loop no longer TRUE, and then exit the outer loop and finish the looping process.

- Flowchart



- **Example:** Write a complete C++ program to calculate average mark of 4 subjects for 3 students.

Problem definition

Input: Marks

Process: Get 4 marks for each of 3 students and calculate the average mark

Output: Average

References:
Rosenah A. H., (2009), Introduction to Programming with C++, for Scientists and Engineers, UPENA UiTM Shah Alam
Norizan M., Mazidah, P., (2006), Problem Solving with C++, UPENA UiTM Shah Alam

Program coding

```
//using for - for
#include <iostream.h>

void main()
{      int sumStud, sumAll = 0, mark, studID;
       float avgMark;

       for (int stud = 1; stud <=3; stud++)
       {      sumStud = 0;
              cout << "Enter student ID: ";
              cin>> studID;
              for (int subject = 1; subject <= 4; subject++)
              {      cout << "Enter mark: ";
                     cin >> mark;
                     sumStud += mark;
              }
              sumAll += sumStud;
       }
       avgMark = sumAll / 3;
       cout << "\nThe total mark is :" << sumAll;
       cout << "\nThe average is :" << avgMark;
}
```

```
//using while - for
#include <iostream.h>

void main()
{      int sumStud, sumAll = 0, mark, studID;
       float avgMark;
       int stud = 1;
       while(stud <=3)
       {      sumStud = 0;
              cout << "Enter student ID: ";
              cin>> studID;
              for (int subject = 1; subject <= 4; subject++)
              {      cout << "Enter mark: ";
                     cin >> mark;
                     sumStud += mark;
              }
              sumAll += sumStud;
              stud++;
       }
       avgMark = sumAll / 3;
       cout << "\nThe total mark is :" << sumAll;
       cout << "\nThe average is :" << avgMark;
}
```

```
//using while - while
#include <iostream.h>

void main()
{      int sumStud, sumAll = 0, mark, studID;
       float avgMark;
       int stud = 1;
       while(stud <=3)
       {      sumStud = 0;
              cout << "Enter student ID: ";
              cin>> studID;
              int subject = 1;
              while (subject <=4)
              {      cout << "Enter mark: ";
                     cin >> mark;
                     sumStud += mark;
                     subject++;
              }
              sumAll += sumStud;
              stud++;
       }
       avgMark = sumAll / 3;
       cout << "\nThe total mark is :" << sumAll;
       cout << "\nThe average is :" << avgMark;
}
```

H. Selection within a loop

- A selection statement inside a loop is commonly found.
- A `for` or `while` loop is used to cycle through a set of numbers and select those numbers that meet the selection criteria.
- **Example:** Write an algorithm that input 10 integers and count the number of odd and even number.

Problem definition

Input: Numbers

Process: Get 10 numbers and count the odd and even number

Output: Count for odd, count for even

Program coding

```
//using while - while
#include <iostream.h>

void main()
{      int num, odd = 0, even = 0;
       for (int x = 0; x < 10; x++)
       {
              cout << “Enter number: ”;
              cin >> num;

              if (num % 2 == 0)
                     even++;
              else
                     odd++;
       }
       cout << “\nThe count for even number is ” << even;
       cout << “\nThe count for odd number is ” << even;
}
```

EXERCISES

1. What will be displayed by the following program fragment?

   a.
   ```
   sum = 0;
   count = 1;

   while (count < 10)
   {
       sum += 10;
           count++;
   }
   cout << “sum ” << “\t” << count << endl;
   ```

   b.
   ```
   sum = 10;
   count = 10;

   do
   {
       sum -= 2;
           count--;
   } while (count != 10);
   cout << “sum ” << “\t” << count << endl;
   ```

   c.
   ```
   sum = 0;
   count = 10;

   for (int x = 1; x < count; x++)
   {
           sum += x;
           count--;
   }
   cout << “sum ” << “\t” << count << endl;
   ```

2. Write a program fragment to display the following pattern using loop statements.

a.

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

b.

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

c.

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

d.

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