CHAPTER 3

BASIC STATEMENTS

Basic Statements in C++ are constructed using tokens. The different statements are

- Input/output
- Declaration
- Assignment
- Control structures
- Function call
- Object message
- Return

3.1 Input/output statements

Input /Output statements such as reading data, processing data and displaying information are the essential functions of any computer program. There are two methods for assigning data to the variables. One method is by assignment statement which we have already seen in the earlier section, and the other method is to read data during the runtime of a program. Data is read from the keyboard during runtime by using the object **cin (**pronounced as C in**).** **cin** is a predefined object that corresponds to a standard input stream. Input stream represents the flow of data from the standard input device – the keyboard. cin can read data from other sources also which will be dealt later. The declarations for the object cin are available in a **header file** called as **istream.h** The basic input/output operations are managed by a set of declarations available in the istream.h and ostream.h header files. Iostream.h file comprises the combined properties of istream and ostream.

- A header file comprises of all standard declarations and definitions for predefined functions.
- One can include the header file in the program by using a preprocessor directive
- A preprocessor directive starts with # , which instructs the compiler to do the required job.
- # include <iostream.h> is a typical preprocessor directive, that instructs the compiler to include the header file iostream.h In order to use cin / cout objects one has to include iostream.h in the program.
- The other header files are iomanip.h, stdio.h, ctype.h, math.h, fstream.h etc.

The >> is the extraction or get from operator. It takes the value from the stream object to its left and places it in the variable to its right. For example consider the following snippet :

```
float temperature;
cin >> temperature;
```

The extraction operator (>>) extracts data from the input stream object (cin) and places the value in the variable(temperature) to its right. Multiple values can be read from the input stream and placed in the corresponding variables, by cascading the extraction operator. For example, to read the values for temperature and humidity one can perform it as follows :

cin >> temperature >> humidity;

cout pronounced as (C out) is a predefined object of standard output stream. The standard output stream normally flows to the screen display – although it can be redirected to several other output devices. The operator **<<** is called the insertion operator or put to operator. It directs

the contents of the variable to its right to the object to its left. For example consider the following statements;

```
int marks = 85;
cout << marks;
cout << "\n Marks   obtained is : " << marks;
```

The value stored in marks is directed to the object cout, thus displaying the marks on the screen.

The second statement **cout << "\n Marks obtained is : " << marks;** directs both the message and the value stored in the variable to the screen. Cascading of insertion operator facilitates sending of multiple output via a single statement.

Examples :

```
cout << "\n The sum of the variables a,b .." << a+b;
cout << a+b << '\t' << a-b << '\t' << a/b;
cout << "\n The difference of numbers …." << a-b
     << "\n The sum of two numbers …. " << a+b;
```

3.2 My first C++ program - Structure of a C++ Program

```
// My first program – Program 3.1
# include <iostream.h>//preprocessor directive
# include <conio.h>
   float fact = 1; // declaration of variables
   int term;
int main()  // function header
{
   clrscr(); // predefined function
   cout << "\n This program computes factorial of a
number";
   cout << '\n' << "Enter a number ...";
   cin >> term;
   for(int x = 2; x <= term;fact *= x,x++);// looping
statement
   cout << "\nThe factorail of the given number .."
         << term << "  is .." << fact;
   return 0;
}
```

A C++ program has primarily three sections viz.,

- Include files
- Declaration of variables , data type , user defined functions.
- main() function

On successful compilation, when the program is executed the main() function will be automatically executed. It is from this block, that one needs to give call statements to the various modules that needs to be executed and the other executable statements.

3.3 Declaration Statements

Variables used in the declaration statements need to be declared and defined before they are used in a program.

```
int *iptr;   //  declares a pointer variable to int
iptr = new int;//fetches memory to store data – hence pointer variable gets defined
*iptr = 5; // stores data 5 only after fetching memory
```

Declaration of a variable introduces a variable's name and its associated data type. For example consider the declaration int *iptr; This statement may be read as iptr is a pointer variable to integer. All pointer variables are defined only when memory is fetched to store data .

Declaration statements are used to declare user defined data type identifiers, function headers, pointer variables and the like. Recall the declaration of user defined data types dealt in **2.3**

However, if a declaration also sets aside memory for the variable it is called as definition. For example consider the declaration statement - **int num;** This statement is called as definition statement because

memory is set aside to store data. Now consider the following snippet :

```
int num;   //  declares and defines an integer variable
num = 5; // The data 5 is stored .
```

Have you noticed , there is no explicit request for memory. That is because memory is set aside at the time of declaring the variable.

- ✓ **Declaration statement introduces a variable name and associates it with a specific data type**
- ✓ **A variable gets defined when memory is set aside .**
- ✓ **Some variables also get defined when they are declared**
- ✓ **Pointer variables get defined only when memory is fetched. For example by using new memory operator**

3.4 Assignment Statements

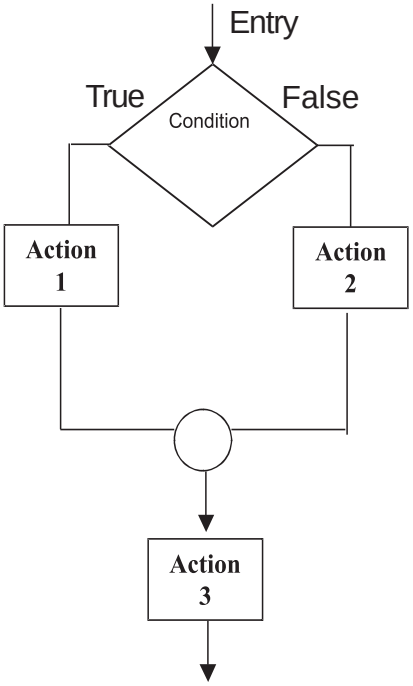
An assignment statement , assigns value on the right hand side of an expression to the variable on the left hand side of the assignment operator. '=' is the assignment operator . For example the different style of assigning values to the variables are as follow :

```
num = 5;
total = english+maths;
sum += class_marks;
```

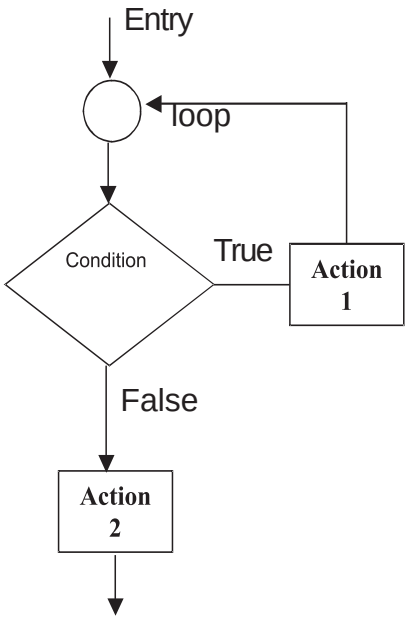
During assignment operation , C++ compiler converts the data type on the right hand side of the expression to the data type of the variable on the left hand side of the expression. Refer to implicit conversions and Type cast of 2.4.2.

3.5 Control Structures

Statements in a program need not necessarily be executed in a sequential order. Some segments in a program are executed based on a condition. In such situations the flow of control jumps from one part of the program to another. Program statements that cause such jumps are called as control statements or control structures. Now look at the following flow charts (Flow chart I & II).



Selection



loop

1. Trace out the steps to accept an integer and if it is odd add 1 to it. If it is even do nothing. Print the integer as depicted in Flow Chart I. The steps are executed in a sequential manner.

2. Trace out the steps to accept a integer, and print the message "EVEN" /"ODD" based on the divisibility of 2. Here the control

branches to statement " M = ODD" if there is remainder other wise branches to the statement "M = EVEN". This is depicted in Flow Chart 2.

> ✓ **Program statements that cause a jump of control from one part of a program to another are called Control Structures**

The two major categories of control structures are Decision making statements and Looping statements. The control structures are implemented in C++ using control statements as indicated in the following figure Fig. 4.1

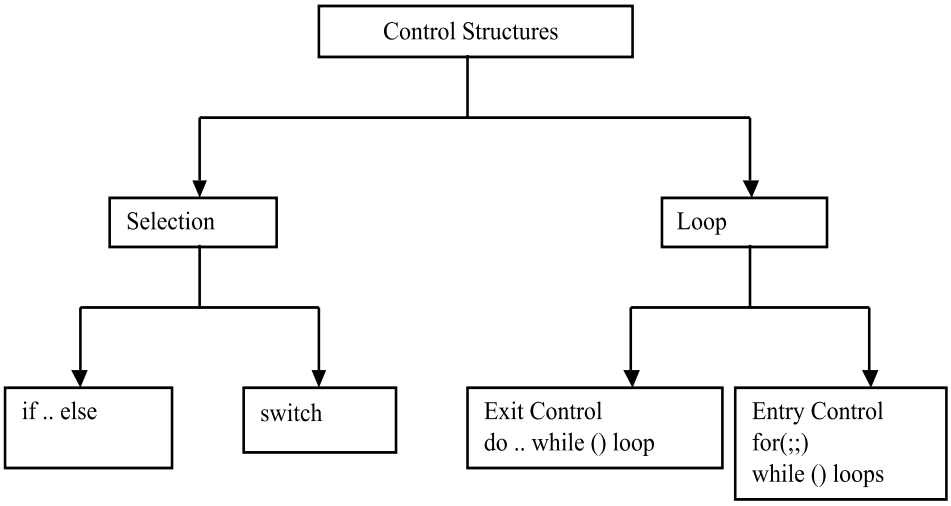


Fig. 4.1 Control Structures in C++

3.5.1 Selection Statements

In a program a decision causes a one time jump to a different part of a program. Decisions in C++ are made in several ways, most importantly with **if .. else …** statement which chooses between two alternatives. Another decision statement , **switch** creates branches

for multiple alternatives sections of code, depending on the value of a single variable.

if statement : is the simplest of all the decision statements. It is implemented in two forms

- Simple if statement
- if .. else statement

```
if ( condition / expression)
{
    action block
}
```

```
If (condition / expression )
{
      action block 1
}
else
{
      action block 2
}
```

The following Program - 3.2 demonstrates **if statement** :

Syntax :

```
// Program - 3.2
# include <iostream.h>
# include <conio.h>
// Demonstrates the use and syntax of if statement
void main()
{
  int a;
  clrscr();
  cout << "\nEnter a number ";
  cin >> a;
  if ( a%2 == 0)
      cout << "\nThe given number " << a << "is even";
  getch();
}
```

In the above program the message “The given….” gets printed if the condition is evaluated to true, otherwise the control jumps to getch(); statement directly by passing the statement cout << “\nThe given ….

The following Program -3.3 demonstrates **if .. else ..** statement :

```
// Program – 3.3
// Demonstrates the use and syntax of if else
statement

# include <iostream.h>
# include <conio.h>
void main()
{
int a;
clrscr();
cout << "\nEnter a number ";
cin >> a;
if ( a%2 == 0)
     cout << "\nThe given number " << a << "is
even";
else
      cout << "\nThe given number " << a << "is
odd";
getch();
}
```

In the above program **“The given number 10 is even”** is printed if the expression is evaluated to true, otherwise statement following else option will be executed.

Examples of if constructs where conditions/expressions are given in different styles :

Condition is expressed using the variable *branch*

```
int branch = 10 > 20;
if (branch )
{
   action block 1;
}
else
{
    action block 2;
}
```

Condition is expressed as 1, as any positive integer indicates TRUE state

```
if (1)
{
   action block 1;
}
else
{
    action block 2;
}
```

Expression is used for condition. If the value of the expression is evaluated to > 0 then action block 1 is executed other wise action 2 is executed.

```
if ( a % 2 )
{
   action block 1;
}
else
{
 action block 2;
}
```

Can you predict as to what will be printed when the following program is executed ?

```
// Program - 3.4
# include <iostream.h>
# include <conio.h>
void main()
{
   int count = 1;
   if (count > 0)
   {
     cout << "\nNegating count ....";
     count *= -1;
   }
   else
   {
     cout << "\nResetting count ...";
     count *= 1;
   }
   getch();
}
```

Output displayed will be **Negating count …** Why do you think block associated with else option is not executed since count was multiplied by –1 ?

Answer to this is that , once the true block is executed in an if .. else statement, then the else block will not be executed.

Else block is executed only if True block is not executed.

- ✓ **if .. else …** statement which chooses between two alternatives , executes the chosen block based on the condition.

The following if constructs are invalid because :

Sno	Invalid construct	Why invalid ?
1.	if a> b cout << “True”;	Condition should always be enclosed in a pair of brackets . The correct form is **if (a>b)** cout << “Condition should True”;

2.	if (a> b)
	a—; cout<<“\nVariable is decremented”;
	else
	a++;
	cout <<
	“Variable is incremented ..”
		Error thrown by the compiler is “**Misplaced else**” . If the action block is compound statements, then it should be enclosed in curly braces .

		The correct form is : if (a> b) { a--; cout<<“\nVariable is decremented”; }else { a++; cout << “Variable is incremented ..” }
3.	if (a > b); cout << “Greater.. ”;else cout << “Lesser ..”;	The semicolon placed after condition nullifies the effect of if statement , the compiler throws an error “**Misplaced else**” .The correct form :if (a > b) cout << “Greater..”;else cout << “Lesser ..”;

Table 3.1 if construct

Write appropriate if constructs for the tasks mentioned in table **3.2**

Sno	Task	If construct
1.	Set Grade to 'A' if marks are above 90.	
2.	Set Grade to 'A' if marks are above 90, otherwise set grade to 'B'	
3.	Print the message • "Accelerate – traffic to flow " if speed is less than 30 kmph, • "Moderate – accelerate by 10kmph" if speed is between 31– 40 kmph, other wise • "Good – be careful .."	

Table 3.2 Using if Constructs

Nested if statement : The statement sequence of if or else may contain another if statement ie., the if .. else statements can be nested within one another as shown below :

```
if  (expression 1)
   if (expression 2)
   {
       action 1;
   }
   else
    {
       action 2;
    }
else
    {
        action 3;
    };
```

In an nested if .. else statement, "**Each else matches with the nearest unmatched preceding if**"

For example

```
if (grade = = 'A')
   if (basic > 5500)
        incentive = basic * 10/100;
   else
         incentive = basic * 5/100;
else
    cout << "Try to attain Grade A";
```

Working of the above example :

- Grade = 'A' and basic == 5501, then incentive gets the value 550.
- Grade = 'A' and basic = = 5000, then incentive gets the value 250.

  - Grade <> 'A' – the inner if will not be executed , the outer else will be executed and thus prints "Try to attain Grade A.

Do you think this if construct is equivalent to the above construct ? Write your answers in the Reason it out box.

```
if (grade = = 'A'  && basic > 5500)
        incentive = basic * 10/100;
else if (grade = = 'A' && basic <5501)
         incentive = basic * 5/100;
else
    cout << "Try to attain Grade A";
```

Reason it out ……

switch Statement : This is a multiple branching statement where, based on a condition, the control is transferred to one of the many possible points.

This is implemented as follows :

```
switch (expression)
{
   case 1 :  action block 1;
             break;
   case 2 :  action block 2;
             break;
    case 3 : action block 3;
              break;
    default :
              action block 4;
}
```

```
switch (remainder)
{
   case  1 : cout << “remanider 1”;
             break;
   case  2 : cout << “remanider 2”;
             break;

    default :
              cout << “Divisible by 3”;
}
```

The following program demonstrates the use of switch statement.

```
// Program - 3.5
// to demonstrate the use of switch statement

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

void main()
{
   int a, remainder;
   cout << "\nEnter a number ...";
   cin >> a;
   remainder = a % 3;
   switch (remainder)
   {
      case 1 : cout << "\nRemainder is one";
               break;
      case 2 : cout << "\nRemainder is two";
               break;
      default: cout << "\nThe given number is divisible by 3";
               break;
   }
   getch();
}
```

The above program displays

- Remainder is two if a = 5 or so
- The given number is divisble by 3, if a = 9 or so

Or in other words the above program checks for divisibility by 3 and prints messages accordingly.

What do you think will be the output of the following program ?

```
// Program - 3.6
// to demonstrate the use of switch statement

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

void main()
{
   int rank = 1;
   char name[] = "Shiv";
   switch (rank)
   {
      case 1 : cout << '\n' << name << " secured 1st
rank";
      case 2 : cout << '\n' << name << " secured 2nd
rank";
   }
   getch();
}
```

Output displayed will be :

Shiv secured 1st rank
Shiv secured 2nd rank

Why do you think both the action blocks of case 1 and case 2 are executed ? Compare the action blocks of Program -3 .5 & Program-3.6. What do you think is missing in Program-3.6 ? Yes it is the **break;** statement.

What do we infer ?

Every action block should be terminated with a break statement. Otherwise all action blocks are executed sequentially from the point where the control has been transferred based on the condition.

In the above example(Program- 3. 6), control was transferred to case 1, as Rank is 1, hence action blocks of case 1 and case 2 are executed sequentially.

> ✓ Include **break;** in action block,
> in order to exit from switch statement.

The following switch constructs are invalid because :

1.
```
char name[] = "Shiv";
switch (name)
{
case "Shiv" : cout << '\n'
<< name << "
secured 1st rank";
case "Rama" : cout << '\n'
<< name << "
secured 2nd rank";
}
```
Compiler throws an error. "Switch selection expression must be of integral type "which means that switch expression should be evaluated to an integer constant only (char, enum,int)

2.
```
float value;
switch (value)
{
case 1.5 : cout << '\n'
<< value – 0.5;
case 2.9 : cout << '\n'
<< value + 0.1;
}
```
Value is of float type , hence not a valid switch expression.

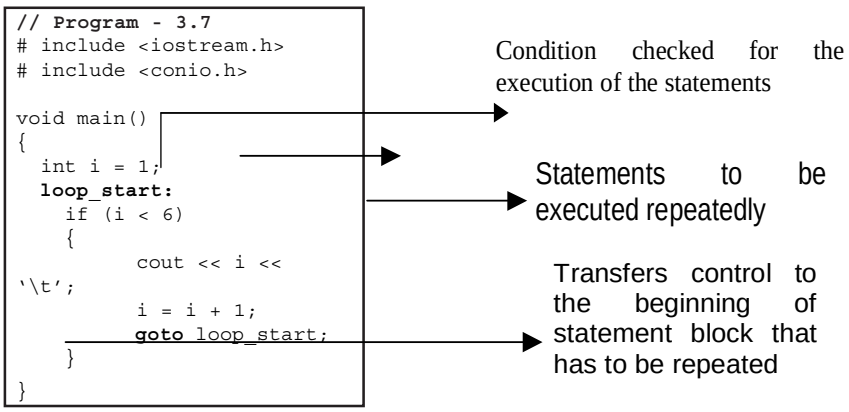
3.
```
switch (rank)
{
case 1 to 2  : cout << '\n'
<< "Best rank";
break;
case 3 to 4 : cout << '\n'
<< "Good rank";
}
```

Case 1 to 2 is an invalid case statement, as case label should have only one integral value. In order to use more than one value for a particular action block one may rewrite the code as :

```
switch (rank)
{case 1 :
 case 2 : cout << "Best
          rank";
          break;
case 3 :
case 4 : cout << "Good
          rank";
          break;
          }
```

3.5.2. Loops

Loops execute a set of instructions repeatedly for a certain number of times. For example consider the following Program – 3.7.



The above program on execution will print numbers between 1 and 5, as the action block of **if statement** is executed 5 times.

The Program - 3. 7 works as follows :

1. Declares and initializes the variable i
2. Checks the relational expression i<6
3. If True then executes the action block (cout << i; i = i + 1) and transfers the control back to the loop_start (**goto** loop_start). This enables the program to execute a set of instructions repeatedly, based on the condition of the relational expression. The variable **i** is referred to as the control variable, as the iterations of the block is totally controlled by this variable.

A looping block therefore consists of two segments viz., the body of the loop and the control statement. The control statement checks the condition, based on which directs the control back to the body of the loop to execute the segment repeatedly. Now look at the following snippets.

```
//Program - 3.7 A
void main()
{
  int i = 6;
  loop_start:
    if (i < 6)
    {
        cout << i << '\t';
        i = i + 1;
        goto loop_start;
    }
   cout << i;
}
```

```
//Program - 3.7 B
void main()
{
  int i = 6;
  loop_start:
     {
        cout << i << '\t';
        i = i + 1;
       if (i < 6)
          goto loop_start;
    }
 cout << i;
}
```

What do you think will be the output generated by the above snippets ?

The Program -3.7 A will display 6, where as Program -3.7 B will display 7. Why do you think the loop is executed in Program-3.7 B? In this program the condition is placed after the statements (cout << i; i = i + 1;),hence these statements are executed once, after which the condition is checked. Since the variable i takes the value as 7, the control is not transferred to loop_start. So what do we infer ??

- ✓ **Loops are unconditionally executed at least once, if the condition is placed at the end of the body ofthe loop**
- ✓ **Based on the position of the condition, the loops are classified as Entry-Check loop (as in Program-3.7 A) and Exit Check Loop (as in Program-3.7 B)**

In general, a looping process would work in the following manner :

1. Initializes the condition variable
2. Executes the segment of the body
3. Increments the value of the condition variable as required
4. Tests the condition variable in the form of a relational expression. Based on the value of the relational expression the control is either transferred to the beginning of the block, or it quits the loop.

There are three kinds of loops in C++, the **for** loop, the **while** loop and the **do .. while** loop.

do .. while Loop : The construct of a do .. while loop is :

```
do
{
action block
} while <(condition)>
```

Look at the following program

```
// Program - 3.8
# include <iostream.h>
# include <conio.h>

// to print the square of numbers
// between 2 to 5

void main()
{
   clrscr();
   int num = 2;
   do
   {
       cout << num * num << '\t';
       num += 1;
   }
   while (num < 6);
   getch();
}
```

Answer the following questions based on the Program - 3.8

A. Identify the
   1. control variable used .
   2. Identify the statements that form the body of the loop
   3. The test expression

B. How many times will the loop be executed ?

C. What is the output of the program?

D. What type of loop is this ?

A.
1. The control variable is num
2. Statements forming the body of the loop are :

```
cout << num * num << '\t';
    num += 1;
```

3. num < 6 is the test expression
B. 4 times
C. 4 9 16 25
D. Exit – check loop

1. Enters the loop
2. Prints the square of num
3. Increments the control variable by 2
4. Evaluates the condition , based on which the control is transferred to step 2
5. End

do … while <(condition)> is called as exit- check loop, as the condition(test expression) marks the last statement of the body of the loop. The following snippets show the various styles of constructing conditions.

```
Int ctr = 1, sum = 0, check =
1;
do
{
  cout << ctr;
   sum = sum + ctr;
   ctr = ctr + 2;
   check = (ctr < 11);
}while(check);
```

```
Int ctr = 5, sum = 0;
do
{

  cout << ctr;
   sum = sum + ctr;
   ctr = ctr -  2;

}while(ctr);
```

```
int ctr = 5,sum = 0,c=1;
do
{

  cout << ctr;
   sum = sum + ctr;
   ctr = ctr -  2;

}while(ctr >= 1);
```

What is the output displayed by the following snippets A and B ?

```
// snippet A
# include <iostream.h>
# include <conio.h>

void main()
{
   int i = 10;
   do
   {
        cout << i;
        i--;
   } while (i <= 10);
   getch();
}
```

```
//snippet B
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 10; choice = 1;
  do
  {
    cout << i;
    i++;
  }while (choice);
  getch();
}
```

Snippet A – the loop will be executed till the variable i gets the value as –32768, and the snippet B will result in infinite loop, as the value stored in the variable **choice** is 1 thus rendering the test expression to be TRUE all the time in both the snippets . It is very important to construct appropriate conditions that would evaluate to false at some point of time, and also incrementing/updating the control variable that is linked to the test expression in the while loop.

while <(condition)>{ … } loop : is called as the **entry-check** loop. The basic syntax is :

```
while <(condition)>
{
action block
}
```

The body of the while loop will be executed only if the test expression results true placed in the while statement. The control exits the loop once the test expression is evaluated to **false**. Let us rewrite all the programs that were discussed under do..while loop (Program - 3.9)

```
// Program - 3.9
# include <iostream.h>
# include <conio.h>

// to print the square of numbers
// between 2 to 5

void main()
{
   clrscr();
   int num = 2;
   while (num < 6)
   {
         cout << num * num << '\t';
         num += 1;
   }
   getch();
}
```

Condition (test expression) is placed at the entry of the body of the loop

The working of the above loop as follows :

1. Initialises the control variable num to 2
2. The test expression num < 2 is evaluated, control is transferred to step 3, only if the test expression is TRUE
3. Prints the square of the value stored in num
4. Increments num by 1
5. Control is transferred to step 2
6. End

Answer the following questions based on the Program - 3.10

```
//Program-3.10
# include <iostream.h>
# include <conio.h>

void main()
{
  int x = 3, y = 4, ctr = 2,res = x;
  while(ctr <= y)
  {
      res *= x;
      ctr += 1;
  }
  cout << "x to the power of y is : "
       << res;
  getch();

}
```

Answer the following questions based on the Program - 3.10

A. Identify the
   1. Control variable used .
   2. Statements that form the body of the loop
   3. The test expression

B. How many times will the loop be executed ?

C. What is the output of the program?

D. What type of loop is this ?

Answers :

1. Control variable used is ctr
2. res *= x; ctr += 1;
3. ctr <= y

B. 3 times

C. 81

Entry- check or entry – controlled loop

What will be the output of the following Program - 3.11 if the values read for choice is y,y,y,y,n?

```
// Program - 3.11
# include <iostream.h>
# include <conio.h>
void main()
{
  clrscr();
  int counter = 0;
  char choice = 'y';
  while (choice == 'y')
  {
       cout << "Continue <y/n> ...";
       cin >> choice;
       counter = counter + 1;
   }
   cout << "\The loop is executed .."
         << counter << "  times";
   getch();
}
```

The following snippets are invalid. Why are they invalid ? Correct the code for proper execution.

```
//Program - 12 A
# include <iostream.h>
# include <conio.h>
//to print numbers between
5&10
void main()
{
    int start = 5,end = 10;
    while (start >= end)
          cout << start++;
    getch();
}
```

```
//Program - 12 B
# include <iostream.h>
# include <conio.h>
//to print numbers between 5&10
void main()
{
     int start = 5,end = 10;
    while (start <= end)
          cout << start++;
    getch();
}
```

```
//Program - 13 A
# include <iostream.h>
# include <conio.h>
// to print numbers between
10&5
void main()
{
    clrscr();
    int start = 5,end = 10;
    while (start <= end)
          cout << start--;
    getch();
}
```

```
//Program – 13 B
# include <iostream.h>
# include <conio.h>
// to print numbers between 10&5
void main()
{
    clrscr();
    int start = 5,end = 10;
    while (start <= end)
          cout << end--;
    getch();
}
```

```
//Program – 14 A
# include <iostream.h>
# include <conio.h>
// to print numbers
       between 1 & 5
void main()
{
    clrscr();
    int start = 1;
    while (Start <=5)
          cout << start++;
    getch();
}
```

```
//Program – 14 B
# include <iostream.h>
# include <conio.h>
// to print numbers
       between 1 & 5
void main()
{
    clrscr();
    int start = 1;
    while (1)
          cout << start++;
    getch();
}
```

for (; ;) .. loop : is an entry controlled loop and is used when an action is to be repeated for a predetermined number of times. The syntax is

for(intial value ; test-condition ; increment)

{

action block;

}

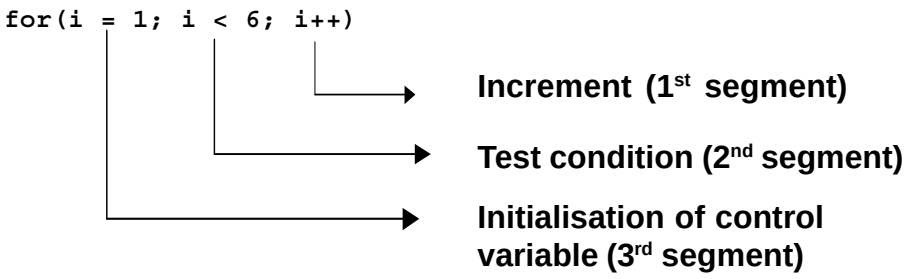
The general working of for(;;)loop is :

1. The control variable is initialized the first time when the control enters the loop for the first time
2. Test condition is evaluated. The body of the loop is executed only if the condition is TRUE. Hence for(;;) loop is called as entry controlled loop.

3. On repetition of the loop, the control variable is incremented and the test condition will be evaluated before the body of the loop is executed.
4. The loop is terminated when the test condition evaluates to false.

The following program illustrates for(;;) loop :

```
//Program - 3.15
# include <iostream.h>
# include <conio.h>

void main()
{
  int i,fact = 1;
  for(i = 1; i < 6; i++)
      fact *= i;
  cout << "\nThe factorial of the number is .." << fact;

}
```



- ✓ **Initialisation is executed only once, ie., when the loop is executed for the first time**
- ✓ **Test condition is evaluated before the commencement of every iteration**
- ✓ **Increment segment is executed before the commencement of new iteration.**

Now look at the following programs and write out as to what will be displayed?

```
//  Program – 3.16
# include <iostream.h>
# include <conio.h>

void main()
{
    int ctr = 10;
    for(; ctr >= 6; ctr—)
       cout << ctr << '\n';
}
```

Output

10

9

8

7

6

```
//Program – 3.17
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  for(int i=2,fact =
1;i<6;fact*=i,i++);
  cout << "\nThe factorial  .." <<
fact;
  getch();
}
```

Output displayed..

The factorial.. 120

Have you noticed the for statement, comprising of more than one statement in segments incrementation and initialisation ? Syntatically and logically the above statement is valid. Each segment in the for loop can comprise a set of instructions, each instruction should be separated by a comma operator. Can you analyse as to what will be the output of the following segment ?

```
void main()
{

  for (int i = 1, j = 0 ; i < 8,j<3;i++,j++)
       cout << i << '\t';
 for (int i = 1,,j = 0 ;j < 3,i < 8;i++,j++)
       cout << i << '\t';
}
```

Output produced will be :
1 2 3 // loop is executed till j < 3
1 2 3 4 5 6 7 // loop is executed till i < 8 Recall the working of comma operator.

Now look at the following for..loop constructs.

```
// Program – 3.18
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   int sum =0, ctr = 1;
   for(;ctr <= 5;)
   {
     sum += ctr;
     ctr = ctr + 1;
   }
   cout << "\nSum :" << sum;
   getch();
}
```

Output displayed will be
Sum : 15

Have you noticed, initialization and incrementation segments are not included in the for(..) construct.

```
//  Program  –  3.19
#  include  <iostream.h>
#  include  <conio.h>
void  main()
{
   clrscr();
   int sum =0, ctr = 1;
   char ch ='y';
   for(;ch  ==  'y';)
   {
     sum += ctr;
     ctr++;
      cout  <<  "\nContinue  <y/n>
?  ..";
     cin >> ch;
   }
   cout  <<  "\nSum  :"  <<  sum;
   cout  <<  "\nChoice  :  "  <<  ch;
   getch();
}
```

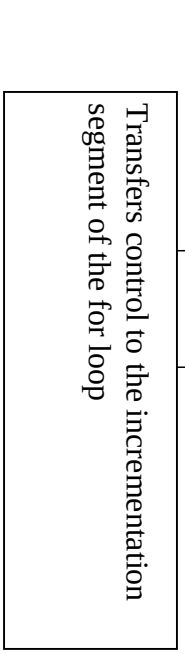
```
Continue  <y/n>  ?  ..y
Continue  <y/n>  ?  ..y
Continue  <y/n>  ?  ..y
Continue  <y/n>  ?  ..n
sum:10
Choice  :  n
 Have you noticed that a for loop
is used like a dynamic loop, where
the  iterations  are  determined
during  run  time.
```

What is wrong with the following snippets ?

What is the impact of the following statements ?
int sum = 0;
for(ctr = 1; ctr < 5; ctr++);
  sum += ctr;
cout << sum;
The output will be 5. Can you reason it out ?
The reason is a semicolon placed after for loop, hence the statement sum+=ctr is not treated as part of the for loop body.

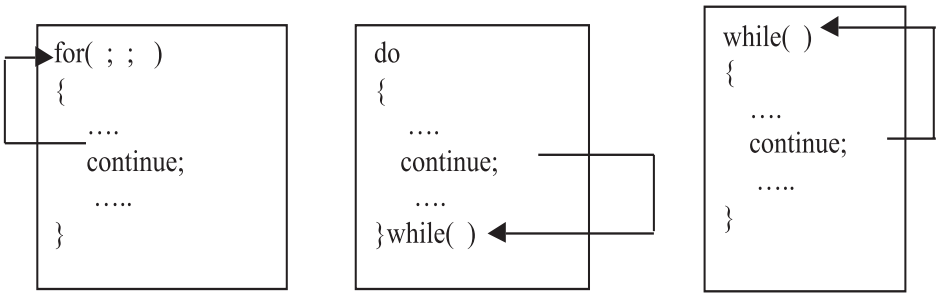
3.5.3 continue

The continue statement forces the next iteration of the loop to take place, skipping any code following the continue statement in the loop body.




```
//Program– 3. 20
# include <iostream.h>
# include <conio.h>

void main()
{
  int i = 1,sum = 0;
  for(;i<10;i++)
  {
      if( i % 2 == 0)
       {
         sum += i;
         continue;
       }
       cout <<i;
   }
   cout << "\nSum of even
nos.."<<sum;
   getch();
 }
```

Working of continue statement in various loops is as follows :



What will be the output of the following segments ?

```
int ctr = 1;
for( ; ctr < 10; ctr++ )
 {
    cout << ctr;
    ctr = 1;
}
```

Since ctr is intialised in the body of the for loop, the loop will re sult in an infinite loop, and the output displayed will be 1

```
int ctr = 1;
for( ctr = 1; ; ctr++ )
     cout << ctr;
```

Since test expression is missing, the loop wil be executed until ctr reaches 32767, the integer data maximum value.

3.5.4 break

A loop's execution is terminated when the test condition evaluates to false. Under certain situations one desires to terminate the loop , irrespective of the test expression. For example consider the Program - 3. 21

```
//Program - 3.21
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   int a[] = {1,2,3,4,5,6,7,8,9};
   int search_item = 7;
   for(int x=0; x<9;x++)
  {
       if (a[x] == search_item)
       {
           cout << "\nItem found at position .." << x;
          break;
        }
   }
    cout << '\n' << "value of index position is .." << x;
    getch();
}
```

Output displayed will be :

Item found at position .. 6
value of index position is .. 6

- ✓ The control is transferred to cout statement written outside the loop, because of break statement. The loop is terminated when x takes the value as 6, because of break statement.

- ✓ **break statement would exit the current loop only.**
- ✓ **break statement accomplishes jump from the current loop**

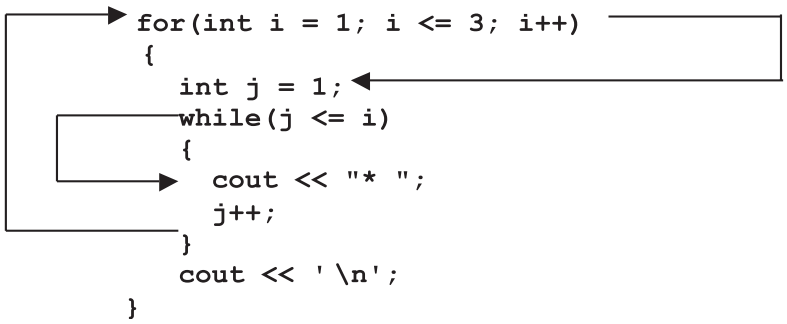
Nested loops : It is possible to nest loop construct inside the body of another. Look at the following Program - 3.22

```
// nesting of loops – Program- 3.22
# include <iostream.h>
# include <conio.h>

void main()
{
   clrscr();
   for(int i = 1; i <= 3; i++)
   {
       int j = 1;
       while(j <= i)
       {
         cout << "* ";
         j++;
       }
       cout << '\n';
  }
  getch();

}
```

Output displayed :

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

Working of the loops is as follows :



The iterations of the nested loops are as follows :

for ..loop	while loop
i = 1	is executed once (j<= I)
i = 2	Is executed twice (j = 1 .. 2)
i = 3	Is executed thrice (j = 1.. 3)

Table 3.4 Nested Loops Example

Can you write out as to what will be the output of the following program ?

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

void main()
{
   clrscr();
  int i = 1, j = 1;
  while(i <= 3)
  {
       cout << '\n';
       for(i=1;i<=j;i++)
        cout << '*';
       i++;
       j++;
  }
   getch();
}
```

Output ??

The rules for the formation of nested loops are :

1. An outer loop and inner loop cannot have the same control variable, as it will lead to logical errors
2. The inner loop must be completely nested inside the body of the outer loop.

3.6 Program Development

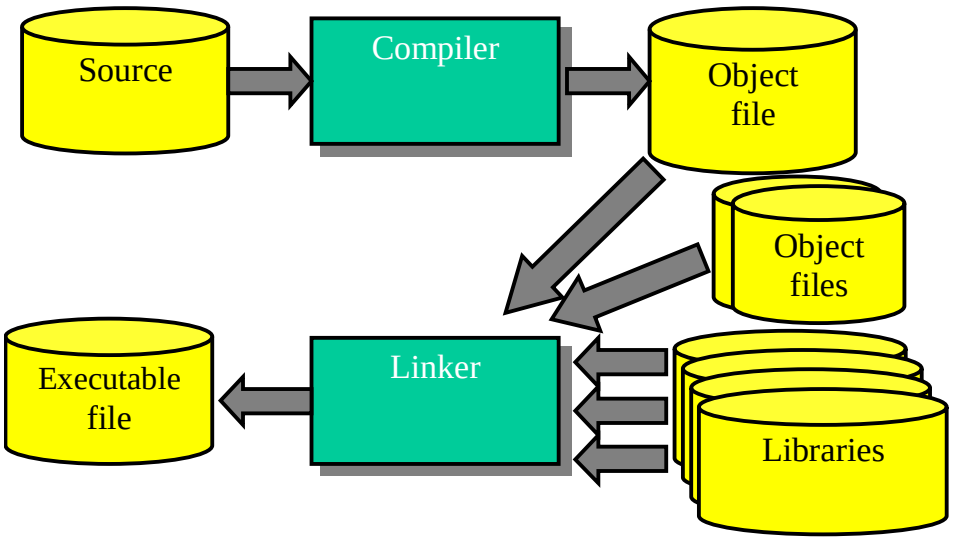


Fig. 3.1 Program Execution

Programs are written in high level language using the grammar of a computer language. A Program written in high level language is called as the Source Code. The source code has to be converted to machine-readable form. The machine-readable form of a program is called as Object file. Compilers create object files from source code. Compilers are translator programs that create a machine-readable program from the source code. Compiler checks for the grammar of language (syntax). An object file is created from an error free source code. The object file is linked with the essential libraries to generate an executable file. This sequence of actions is shown in Fig. 3.1.

Exercises

1. Categorise the following declarations as valid/invalid. If invalid, specify the reasons.

Declarations	Valid/Invalid	Reason
int A;a;		
char name(10);		
float f,int;		
double d, float f;		
int 1choice, _2choice		

2. Debug the following program. Rewrite the corrected program.

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

void main()
{
 int N1,n1;
 cin << ' \nEnter two number s ..';
 result := N1 * n1;
 cout << ' \n' << Result;
}
```

3. Write appropriate declaration statements for the following :
   a. To store the result of the expression 8/3 .
   b. To initialise Emp_Name with the value "Kalam"
   c. To accept choice from user indicating Y-yes and N – no

4. Point out errors in the following snippets :
   a. int a = 10, b = 5;
      if a > b
      cout << a;

b. if (a<b) && (a<0)
cout << “a is negative and …”

c. char option = ‘Y’;
do while option == ‘y’
{
cout << ‘*’;
……
}

d. for(int I = 1; I < 10; I++)
cout << I * 2;

e. do
{
cout << ‘*’;
}while(cout << “\nContinue <y/n>…”;cin>>ch;ch == ‘y’);

5. What will be the output of the following snippets / programs?

```
// 5 a.
# include iostream.h>
# include <conio.h>

void main()
{
   int feet;
   const int inch_conversion = 12;
   clrscr();
   cout << “\nEnter feet …”;
   cin >> feet;
   cout << “\nConverted to inches …”
           << feet * inch_conversion;
}
input-7 for feet
```

```
// 5 b.
# include <iostream.h>
# include <conio.h>

void main()
{
   int I = 1, sum = 0;
   clrscr();
   while(I++ <= 5)
   {
      cout << ‘\n’ << I;
      s += I;
   }
 cout << “\nValue of the variable I after
executing the while loop ..” << I << “\nSum
:…” << s;
```

// 5 c

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

void main()
{
  int i = 1, sum = 0;
  clrscr();
  while(++i <= 5)
  {
         cout << '\n' << i;
         sum += i;
  }
  cout << '\n' << i << '\t' << sum;
  getch();
}
```

// 5 d

```
# include <iostream.h>
# include <conio.h>
void main()
{
  int i = 1, sum = 0;
  clrscr();
  for(i = 1; i <= 5; i++)
  {       cout << '\n' << i;
          sum += i;
  }
  cout << '\n' << i << '\t' << sum;
  for(i = 1; i <= 5; ++i)
  {
          cout << '\n' << i;
          sum += i;
  }
  cout << '\n' << i << '\t' << sum;
 }
```

//5e

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

void main()
{
  int i = 1, j = 1;
  clrscr();
  do
  {
        while (j<=i)
        {
            cout << '#';
            j++;
        }
        cout << '\n';
        i++;
        j = 1;
  } while(i<= 5);
  getch();
}
```

// 5 f

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

void main()
{
  int num = 1784, s= 0, d = 0, x;
  x = num;
  clrscr();
  for(;num > 0;)
  {     d = num % 10;
        s += d;
        num = num / 10;
  }
  cout << "\nThe sum of digits of "
       << x << "is : "
       << s;
  getch();
}
```

//5 g

```
# include <iostream.h>
# include <conio.h>
void main()
{
  clrscr();
  for(int i = 1,s = 0; ; i++)
  {
        if (i%2 == 0)
           continue;
        s += i;
        if ( i > 9)
           break;
  }
  cout << "\nThe sum is ...." <<
s;
  getch();
```

// 5 h

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

void main()
{
  clrscr();
  for(int i = 1,x = 0;i <= 5; i++)
    x = x + i%2==0 ? i*1 : i * -1;
  cout << x;
  getch();
}
```

```
//5 j
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  do
  {
    cout << "\ndo loop ...";
  } while (0);
  getch();
}
```

```
//5 k
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int i = 0;
  for(i = -5; i >= 5; i—)
        cout << "Bjarne Stroustrup";
  cout << "\nReturning to Edit Window..";
  getch();
}
```

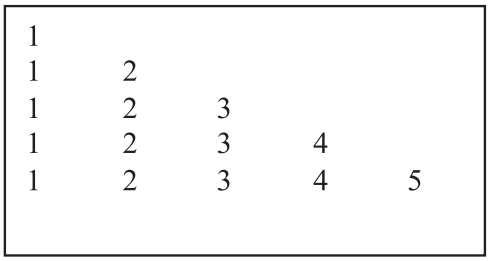
```
//5 l
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
 int month = 5;
 if (month++ == 6)
    cout << "\nMay ...";
 else if (month == 6)
   cout << "\nJune ...";
 else if (—month == 5)
   cout << "\nMay again ..";

}
```

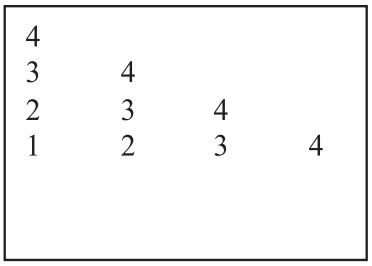
```
// 5 m
# include <iostream.h>
# include <conio.h>
void main()
{ int day = 3;
  switch (day)
  {
    case 0 : cout << "\nSunday ..";
    case 1 : cout << "\nMonday ..";
    case 2 : cout << "\nTuesday ..";
    case 3 : cout << "\nWednesday .. ";
    case 4 : cout << "\nThursday ..";
    case 5 : cout << "\nFriday ..";break;
    case 6 : cout << "\nSaturday ..";
  }
 }
```

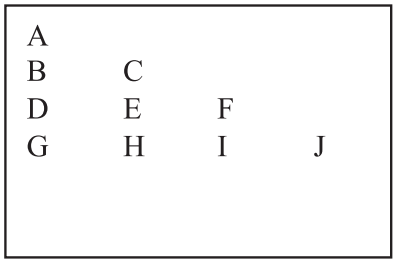
```
// 5 n
# include <iostream.h>
# include <conio.h>

void main()
{
  clrscr();
  int bool = 2,b =4;
  while(bool)
  {
    cout << bool << '\t' << ++b << '\n';
    bool—;
    b—;
  }
  getch();

}
```

6. Program Writing

a. Write a program to compute a^b where a and b are of real and integer types(use while .. loop)
b. Write a program to compute the factorial of a given number. (use for( ) loop)
c. Write a program to generate fibonacci series upto n^{th} term. Fibonacci series is : 0,1,1,2,3,5,8,12,20,32 …
d. Write a program to print the following patterns :







Using a switch, write a program to accept the day in a month, and print the messages as :

If day is 1, message is 1^{st} day in the month

If day is 2,22 , message is 2^{nd} / 22^{nd} day in the month

If day is 3,23, message is 3^{rd}/23^{rd} day in the month

If day is 4,14,15,16… message is 4^{th} /14^{th} .. day in the month