INTRODUCTION TO

C++

Part-3

```
int main()
{
int a,b;
cout<<"enter the value of a & b\t";
cin>>a>>b;
if(a>b)
        cout<<"\n A is greater than B";
else
        cout<<"\n B is greater than A";
getch();
return 0;
}
```

- Output

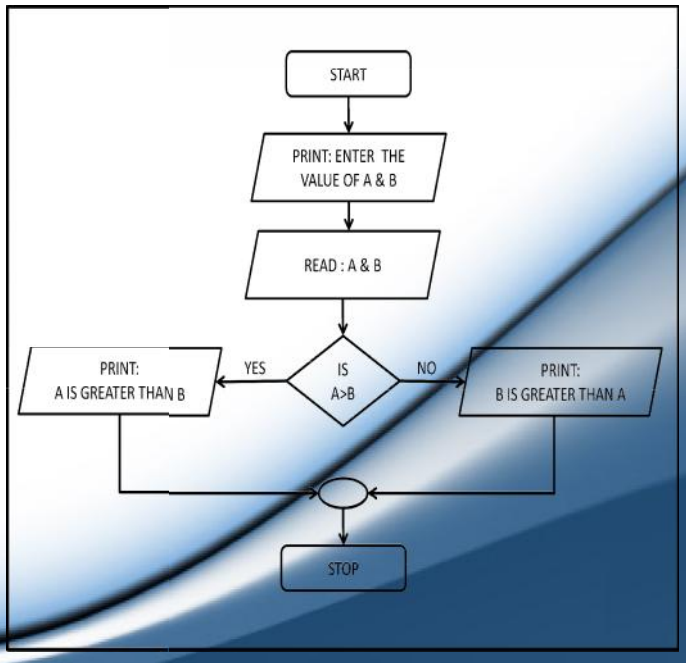
First Run:

```
Enter the value of a & b        2 4
B is greater than A
```

Second Run:

```
Enter the value of a & b        8 4
A is greater than B
```

- The above program could be better understood with the following flowchart:



Nested if-else statements:

- C++ allows nesting of if-else statements ie. We can have another if or if-else statement inside either the if part or else part as follows:

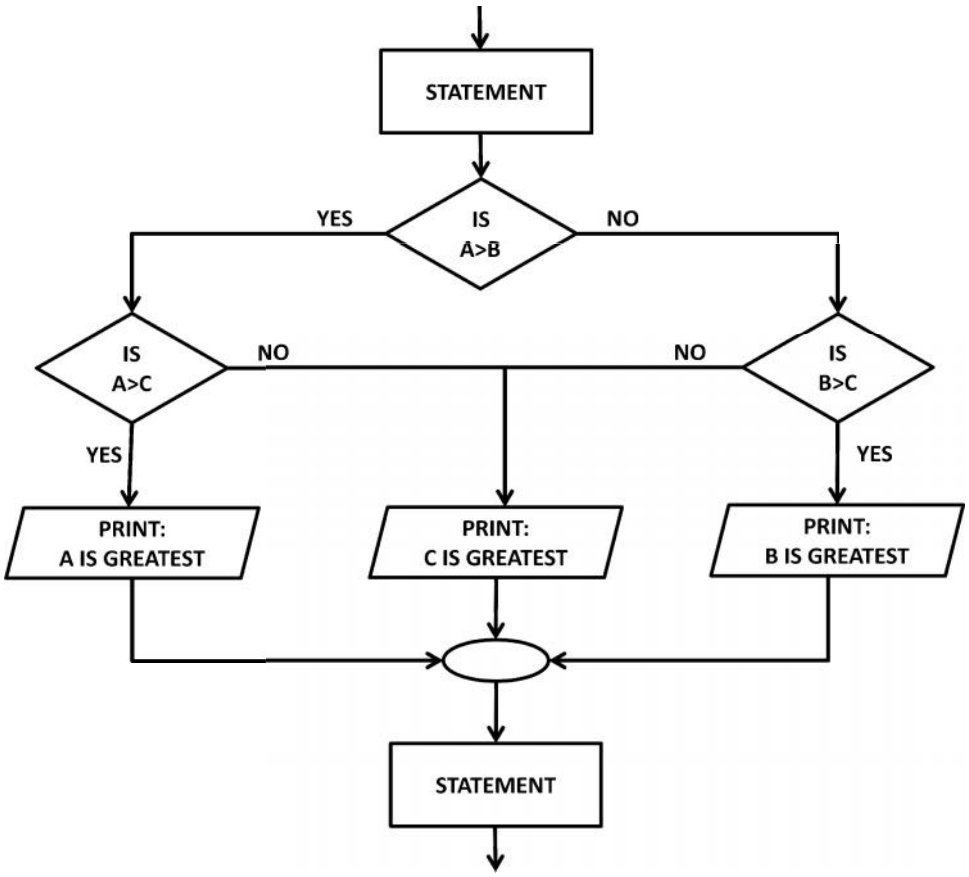
```
if(condition)
 {
        statements;
        if(condition)
         {

        statements;
         }
        else
         {

        statements;
         }
         statements;
 }
else
 {
        statements;
 }
```

```
if(condition)
 {
        statements;
 }
else
 {
        statements;
        if(condition)
         {

        statements;
         }
        else
         {

        statements;
         }
        statements;
 }
```

```
/*************************************************************
****
Program 6.3
Author : Nikhil Pawanikar
Description : Program to take three inputs from user and print the
greater
                Of the three using nested if-else statements
*************************************************************
****/
#include<iostream.h>
#include<conio.h>

int main()
{

 int a, b, c, greatest;
 cout<<"Enter the valules for a, b & c\n";
 cin>>a>>b>>c;
```

```
if(a > b)
{
    if(a>c)
    {
            cout<<"\n A is the greatest of three";
    }
    else
    {
            cout<<"\n C is the greatest of three";
    }

}
else
{
    if (b>c)
    {
            cout<<"\n B is the greatest of three";
    }
    else
    {
            cout<<"\n C is the greatest of three";
    }

}
 getch();
 return 0;
}
```

The flowchart for nested if else for the above program is as follows:



6.2.3 Conditional operator:

- It is also known as ternary operator since it takes three parameters.
- It is represented as **? :**
- In one sense it is short form of if-else statement.
- Its syntax is as follows:

```
Expression 1 ? Expression 2 :  Expression3;
```

- It is read as follows: If Expression 1 is true. i.e non zero, expression 2 is returned else expression 3 is returned.

- Example :

```
int a = 3;
int b = (a > 5 ? 1 ; 0);
```

In the above example is the first line reads an assignment statement where variable a is assigned the value 3.

The second line uses conditional operator. Her if value of a is greater than 5 then b is assigned a value of 1 else a value of zero.

- With the if-else statement the above example could be written as:

```
int a = 3;
int b;
if(a > 5)
    b= 1;
else
    b= 0;
```

- Conditional operators need not be limited to arithmetic expressions only. It can also be used as follows:

```
char gender;
cout<<"enter gender\t";
cin>>gender;
(gender=='m' ? cout<<"Gender is Male" : cout<<"Gender is Female" );
```

If the user enters 'm' then the line printed will be "Gender is Male" else the line printed will be "Gender is Female".

- Nesting of conditional operators is also possible. Ex. Greater of three numbers

```
int a =4, b=5, c=2;
int greater;
greater = ( a > b ? (a > c ? a : c) : ( b > c ? b : c ) );
```

- The only limitation of Conditional operator is that it allows only one statement to executed after ? or :
- Example:

```
/*************************************************
Program 6.3
Author : Nikhil Pawanikar
Description : Program to take three inputs from user and print the
greater
                Of the three using conditional operator
*************************************************/

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

```
int main()
{

 int a, b, c, greatest;
 cout<<"Enter the valules for a, b & c \n";
 cin>>a>>b>>c;

 (a > b ? (a > c ? greatest = a  : greatest = c )  : (b > c ? greatest = b  :
greatest = c ));

 cout<<"\nGreatest of three is "<< greatest;
 getch();
 return 0;
}
```

Note: Semicolon is used only once with conditional opertors at the end of statement.

4. switch statement:

- We use if statements to choose one among the available alternatives.
- When the number of alternatives goes on increasing the complexity to implement also goes on increasing.
- C++ has a multiway decision statement called **switch.**
- The possible values of expression/condition is represented by **case.** A switch statement can have multiple cases representing multiple decisions to be taken.
- The keyword **break** is used inside every case of switch statement to exit the statement once the case is matched and executed.
- The syntax of a switch statement is as follows:

```
switch(integer expression)
{
        case constant1:
                do this;
                break;
        case constant1:
                do this;
                break;
        case constant1:
                do this;
                break;
```

```
                default:
                        do this;
                        break;
        }
```

- The **integer expression** is any expression that will evaulate to give an integer value.
- The keyword **case** is followed by an integer or character constant which may represent the value of integer expression. Each constant character or integer must be unique.
- Every case contains a set of valid c++ statements. The keyword **break** is the last statement inside every case and forces the execution to come out of the switch statement. Without break the execution is said to **fall through** the cases i.e if there is no break the execution proceeds to the next case even if it is not supposed to execute.
- When a program containing a switch statement is executed the integer expression following the switch keyword is evaluated first. This value is then matched one by one with the constant values that follow the case keyword. If a match is found the statements following the case are executed. If no match is found then the statements under the default case are executed.
- Example: Consider the following program that displays a specific message as per the input given by the user.

```
/*************************************************
Program 6.4
Author : Nikhil Pawanikar
Description : This program prompts the user to enter any
              number between 1-5 and displays corresponding
              papers nomenclature.
*************************************************/
#include<iostream.h>
#include<conio.h>

int main()
{
 int number;
 cout<<"Semester 1 has five papers."<<endl;
 cout<<"Enter any number in 1 - 5 to know the nomenclature of
the paper \n";
 cin>>number;
cout<<endl;
 switch(number)
```

```
{
  case 1:
          cout<<"Paper 1 is Professional Communication Skills\n";
          break;
  case 2:
          cout<<"Paper 2 is Applied Mathematics-1 \n";
          break;
  case 3:
          cout<<"Paper 3 is Fundamentals of digital computing \n";
          break;
  case 4:
          cout<<"Paper 4 is Electronics & Communication
Technology \n";
          break;
  case 5:
           cout<<"Paper 5 is Introduction to C++ Programming -1
\n";
          break;
   default:
           cout<<"Not a valid input";
           break;
 }
 getch();
 return 0;
}
```

Output:

First Run:

```
Semester 1 has five papers.
Enter any number in 1 - 5 to know the nomenclature of the paper
5
Paper 5 is Introduction to C++ Programming -1
```

Second Run:

```
Semester 1 has five papers.
Enter any number in 1 - 5 to know the nomenclature of the paper
8
Not a valid input
```

- In the program above the user is supposed to enter any number between 1 – 5 which stands for the papers at FYBScIT Sem I. The value entered by the user is stored in a variable number which

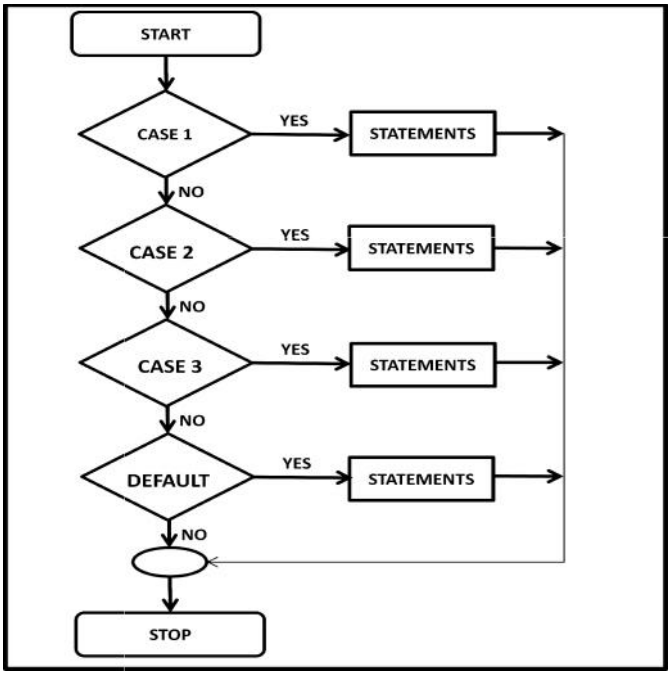
also happens to be the integer expression following the keyword switch.

- Since we have 5 subjects at sem I of FYBScIT we have written 5 cases. Here integer constants are used since the integer expression is an integer. If the expression was character then the case constants would also have to be characters.
- In the first run when the user enters 5, 5 is stored in the variable number. Then the value of variable number is matched with every case constant inside switch statement in the order in which they appear. Since a match is found that equals to 5, the statements following that case are executed.
- When the user enters 8 as an input there is no match with the case constants so the default case is executed.
- The same program with character input and character constant would look like this:

```
/*************************************************************
Program 6.5
Author : Nikhil Pawanikar
Description : This program prompts the user to enter any
              character between A-E and displays corresponding
              papers nomenclature.
*************************************************************/

#include<iostream.h>
#include<conio.h>
int main()
{
 char choice;
  cout<<"Semester 1 has five papers."<<endl;
 cout<<"Enter any character from A - E to know the nomenclature of the
paper \n";
 cin>>choice;

 switch(choice)
 {
    case 'A':
          cout<<"Paper A is Professional Communication Skills\n";
          break;
```

```
  case 'B':
        cout<<"Paper B is Applied Mathematics-1 \n";
        break;

case 'C':
        cout<<"Paper C is Fundamentals of digital computing \n";
        break;

case 'D':
        cout<<"Paper D is Electronics & Communication Technology \n";
        break;

case 'E':
        cout<<"Paper E is Introduction to C++ Programming -1 \n";
        break;

default:
        cout<<"Not a valid input";
        break;
}
getch();
return 0;
}
```

The flowchart for switch statement is as follows:



3. COMPOUND STATEMENTS

- The statements that are written inside a pair of parenthesis { }, are called compound statements.
- We have already seen the use of compound statements in if, if-else and switch statements.
- Ex.

```
int a,b,c;
a=2,b=3,c=0;
if(a!=0)
{
c=a+b;
cout<<"c="<<c;
}
```

Here the statements written inside parenthesis { } are compound statements, once the control passes into this block all the statements inside it are executed unless forced to quit or jump out before reaching the last statement.

Compound Conditions:

- Just as we have compound statements we have compound conditions too.

- Compound conditions are used to combine two or more conditions using logical operators && (and), || (or) and ! (not).
- They are defined as:

A && B	-	Evaluates to true if both A and B are true
A || B	-	Evaluates to true if either A or B is true or in other words Evaluates to false if and only if both A & B are both false
! A	-	Evaluates to true if and only if A evaluates to false

- Using truth tables it could be defined as follows:

A	B	A&&B	A||B	!A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Example : Program to find the greater of 3 numbers

```
/*******************************************************
Program 6.3
Author : Nikhil Pawanikar
Description : Program to take three inputs from user and print the
greater
                Of the three using nested if-else statements
*******************************************************/
#include<iostream.h>
#include<conio.h>

int main()
{
      int a, b, c, greatest;
      cout<<"Enter the values for a, b & c \n";
      cin>>a>>b>>c;

     if(a > b && a >c)
     {
              cout<<"\n A is the greatest";
     }
     if (b>a  && b > c)
```

```
    {
            cout<<"\n B is the greatest of three";
    }
    if (c>a && c > b)
    {
            cout<<"\n C is the greatest of three";
    }
     getch();
     return 0;
}
```

6.4 INCREMENT (++) & DECREMENT(--) OPERATORS

- These are unary operators i.e. they take only one operand.
- Increment & Decrement operators are used to increase or decrease the value of integer variables or integer constants by 1.
- There are two forms of Increment operator: Pre-increment & Post-increment.
- Example: an integer variable m can be incremented in two ways:
  m++; ++m;
  & decremented in two ways m--; --m;

```
/*************************************************************
Program 6.
Author : Nikhil Pawanikar
Description : Program to display the use of Increment & Decrement
operators
*************************************************************/
#include<iostream.h>
#include<conio.h>
int main()
{
 int a, b;
 a=0;
 b=2;
 clrscr();
 a = ++b;
 cout<<" value of a is "<<a;          //value of a is 3
 cout<<" value of b is "<<b;          //value of b is 3
 b=4;
 a=b++;
 cout<<" value of a is "<<a;          //value of a is 4
```

```
cout<<" value of b is "<<b;        //value of a is 5
a = --b;
cout<<" value of a is "<<a;        //value of a is 3
cout<<" value of b is "<<b;        //value of b is 3
b=4;
a=b--;
cout<<" value of a is "<<a;        //value of a is 4
cout<<" value of b is "<<b;        //value of a is 3
getch();
return 0;
}
```

6.5 Review Questions

1. Explain the different forms of if statement.
2. Explain the difference between if statement & switch statement.
3. What happens if we do not use break in switch statement?
4. Write a program in C++ for the following:
    1. Take an integer number from the user and find if it is even.
    2. Take an integer number from the user and find if it is odd or even.
    3. Take the value of cost price and selling price of a particular item from user and print if it profit or loss and how much.
    4. Take the value of 5 subjects from user. Calculate sum, percentage and display grade(use compound conditions)
    5. Using switch case, write a program to perform arithmetic operations (+,-,* ,/). Show a menu to the user and allow selecting an option.

6.6 Reference & Further Reading

1. Let us C – Yashwant Kanetkar, Chapters 2 & 4.
2. Object Oriented Programming with C++ - E. Balaguruswamy, Chapter 3
3. Programming in C++, Schaums Outlines – John R. Hubbard, Chapters 2 &3.

Chapter 7

Looping Flow of Control

Unit Structure

1. Introduction
2. Loop Control Instructions
   - 7.2.1 while
   - 7.2.2 do while
   - 7.2.3 for
3. Review Questions
4. References & Further Reading

7.1 INTRODUCTION

In the previous chapter we had seen the concept of Branching Flow of control which involves making decision which involves having a condition. If the condition is satisfied then the decision is to execute a particular set of instructions, if not then another set of instructions.

This chapter is dedicated to Looping or iteration. It is used when a set of instructions is to be executed multiple times or for a fixed number of times until a particular condition is satisfied.

This repetitive action is done through Loop Control Instruction. C++ offers the following Loop Control Instructions:

1. for
2. while
3. do while

2. LOOP CONTROL INSTRUCTIONS

1. while loop:

- The while loop has the following syntax:

while (condition is true)
statement;
	Initialize counter variable;
	while (condition is true)
	{
	statement;
	statement;
	increment counter;

	}

- Here condition is an integer expression just like the one we have seen in the previous chapter.
- When the system reads the keyword while it evaluates the expression that follows.

  If the expression is evaluated as true the statement following the condition is executed. It may be a simple statement or a compound statement enclosed in parenthesis { }.
- Example : consider the following example to find the cube of the number entered by user.

```
/*******************************************************
Program 7.1
Author : Nikhil Pawanikar
Description : Program to display sum of numbers upto n
*******************************************************/
#include<iostream.h>
#include<conio.h>
int main()
{
 int num,sum=0;;
 clrscr();

 cout<<" \n Enter any positive integer number \n ";
 cin>>num;
int i=0;

 while(i<=num)
        sum=sum + i++;
cout<<"\n Sum of numbers till  "<<num<<" = "<<sum;

 getch();
 return 0;
}
```

Output

First Run

```
Enter any positive integer number
2
Sum of numbers till  2 = 3
```

Second Run

```
Enter any positive integer number
5
Sum of numbers till  5 = 15
```

- In the above example the while loop will execute a single statement until the condition i<=num evaluates to true.

- The following example executes a set of statements once the condition becomes true.

```
/*************************************************************
**************
Program 7.2
Author : Nikhil Pawanikar
Description : Program to display the cube of a number until the user
types zero
*************************************************************
*************/
#include<iostream.h>
#include<conio.h>

int main()
{

 int num,cube=0;;
 clrscr();


 int i=0;
 while(i<=num)
 {
        cout<<" \n Enter any positive integer number to find its cube,
type 0 to exit \n ";
        cin>>num;
        cube = num * num * num;
        cout<<"\n Cube of num "<<num<<" ="<<cube;
        i=i+1;
 }

 getch();
 return 0;
}
```

Output

First Run

```
Enter any positive integer number to find its cube, type
0 to exit
2
Cube of 2 = 8
```

Second Run

```
Enter any positive integer number to find its cube, type
0 to exit
```

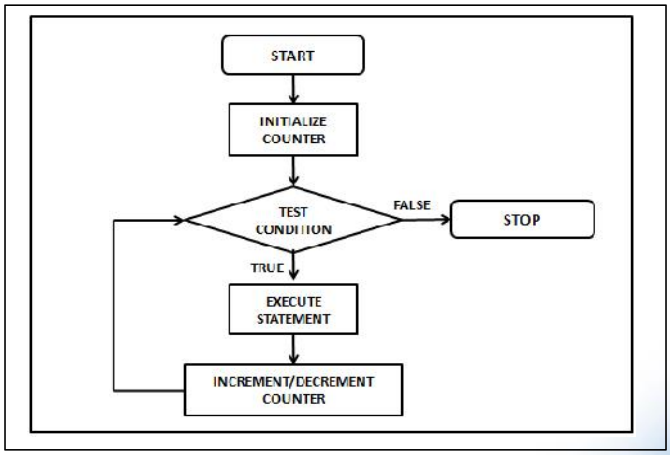
```
0
Cube of 0 = 0
```

Note: The while loop may execute infinite number of times if the condition in the loop happens to be true for every value.

Ex.

```
while (1)
{
cout<<"\nhello world";
}
```

- Since the condition following while returns a non zero value it happens to be true every time a condition is checked and runs infinitely.
- Instead of incrementing the counter value we can also decrement it.
- The flowchart for a while loop is as follows:



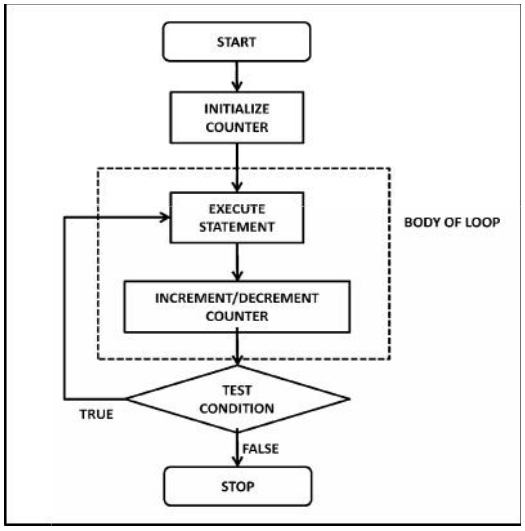
2. do while

- The syntax for the do-while loop is as follows:

```
do
   statement;
while (condition is true);
```

```
Initialize counter variable;
do{
    statement;
    statement;
    increment counter;
} while (condition is true);
```


- The do while loop executes in the same way as the while loop. The only difference is that in do-while, the statements are executed at least once even if the condition is evaluated to be false.
- The do while statement is always terminated with a semicolon (;) after the loop ends.
- The following flow chart will help understand the do-while loop:



- As shown in the flowchart above, the body of the loop is executed and then the condition is tested. So when the program executes the first iteration will always execute the loop and then test for the condition. If the condition evaluates to true it will again execute the block of statements else it will exit the loop and execute the next statement following the do-while loop if any.
- **Example:** Consider the program discussed for while lo p. Cube of number. We shall see how it could be done using do-while loop.

```
/*************************************************************
Program 7.3
Author : Nikhil Pawanikar
Description : Program to display the cube of a number until the user
wants to using
Do-while loop
*************************************************************/
#include<iostream.h>
#include<conio.h>

int main()
```

```
{
 int num,cube=0;;
 char ans;  clrscr();
  int i=0;
 do
 {
   cout<<" \n Enter any positive integer number to find its cube \n ";
   cin>>num;
    cube = num * num * num;
    cout<<"\n Cube of num "<<num<<" = "<<cube;
    i=i+1;
    cout<<"\nPress Y to continue";
    cin>>ans;
}while(ans=='y' || ans == 'Y');

 return 0;
}
```

Output

First Run

```
Enter any positive integer number to find its cube
2
Cube of  2 = 8
Press Y to continue
Y
Enter any positive integer number to find its cube
4
Cube of  4 = 64
Press Y to continue
a
```

Second Run

```
Enter any positive integer number to find its cube
3
Cube of  3 = 27
Press Y to continue
r
```

- In the above example the condition is on whether the user wants to continue the program or exit.
- In first run, the program executes the loop for one time and then checks the condition where the user enters ‘y’ and executes again. Next time when the user was prompted to continue, a key other than y was pressed and hence the condition was false and the control came out of the loop.
- In the second run, the program executes the loop for one time and then checks the condition where the user enters ‘y’ and executes again. . Next time when the user was prompted to

continue, a key other than y was pressed and hence the condition was false and the control came out of the loop.

3. for

- The syntax of the for loop is as follows:

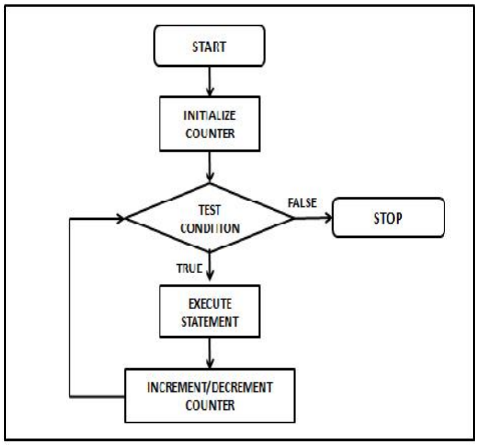
```
for (initialization ; test condition ; increment/decrement loop counter)
        statement;
```

or

```
for (initialization ;  condition ; increment/decrement loop counter)
 {
    statement;
    statement;
 }
```

- The for loop allows to specify three things in a single line:
  1. Create and initialize loop counter variable
  2. Test condition
  3. Increase or decrease the value of loop counter after every iteration

- The flowchart of for loop is same as while loop.



- Example:The program for calculating sum of numbers till n using for loop is as follows:

```
/*************************************************
Program 7.4
Author : Nikhil Pawanikar
Description : Program to display sum of numbers upto n
*************************************************/
#include<iostream.h>
#include<conio.h>
int main()
{
    int num,sum=0;;
   clrscr();

   cout<<" \n Enter any positive integer number \n ";
   cin>>num;
   for(int i =0; i<=num ; i ++)
{
   sum=sum + i;
}

cout<<"\n Sum of numbers till  "<<num<<" = "<<sum;
getch();
return 0;
}
```

Output:

First Run

```
Enter any positive integer number
2
Sum of numbers till  2 = 3
```

Second Run

```
Enter any positive integer number
```

```
5
Sum of numbers till  5 = 15
```

- In the program above, for statement first initializes the variable i to zero, then it checks for the condition. If the condition is true the statements inside the block following for statement are executed. Once the loop is over the value of i is incremented by one and the condition is tested again.
- As long as the value of i happens to be less than or equal to num the loop will execute. When the value of i becomes greater than num the control is passed to the next statement after the loop.

Nesting of Loops:

Nesting of loop control structures is possible in the same way as in decision control.

7.3 REVIEW QUESTIONS

1. List and explain the statements under Loop Control Structure.
2. Explain the difference between the while and do-while loop.
3. Explain the difference between the while and for loop.
4. Write a program in C++ to do the following:
5. Take a number from user and print Fibonacci series on the screen containing that many numbers(i.e. 0 ,1, 1, 2, 3, 5, 8, 13, 21)
6. Print the odd and even number from 1 to 100
7. Print the following pattern

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

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

7.4 REFERENCES & FURTHER READING

1. Let us C – Yashwant Kanetkar, Chapter 3.

2. Object Oriented Programming with C++ - E. Balaguruswamy, Chapter 3
3. Programming in C++, Schaums Outlines – John R. Hubbard, Chapter 4.
4. Theory & Problems of Programming with c ++ – John R. Hubbard, Chapter 3.

Chapter 8

Break and continue

Unit Structure

1. Introduction
2. Break and continue.
3. Manipulators: endl , setw,sizeof.
4. Type Cast Operators
5. Scope resolution operators

8.1. INTRODUCTION

In the previous chapter we studied Loop control structure. In this chapter we study two mechanisms to bypass the looping construct.

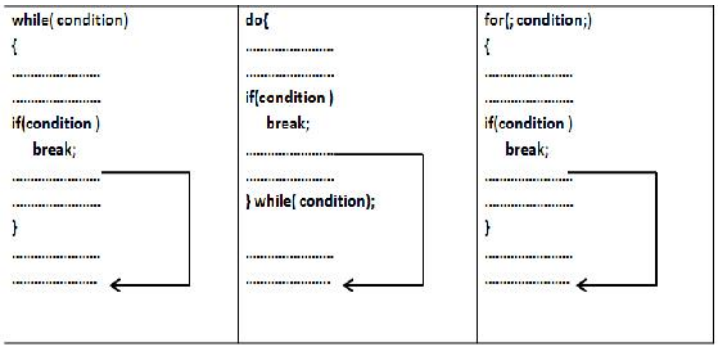
These are break and continue.
Then we study the use of following manipulators:
endl , setw,sizeof & finally operators like
Type Cast Operators & Scope resolution operators

8.2 BREAK AND CONTINUE

break

- Normally a **while** loop, a **do..while** loop, or a **for** loop will terminate only at the beginning or at the end of the complete sequence of statements in the loop's block.
- But sometimes there may be situations when you require the control to exit the loop. This can be fulfilled by using the **break** keyword.
- **A break statement terminates a loop**. The control is transferred to the first statement immediately following the loop construct.
- We have seen the use of break statement in switch statement. When a match is found the statements under the case are executed and then break statement causes the switch loop to terminate.
- A break statement is usually associated with a condition, so there will be an if statement.

The general form of break is as follows:



- Example: Consider program 7.1. to print sum of integers upto n to be done using break

```
/*************************************************
*****
Program 8.1
Author : Nikhil Pawanikar
Description : Program to display sum of numbers upto n using
break
*************************************************
****/
#include<iostream.h>
#include<conio.h>
int main()
{
 int num,sum=0;;
 clrscr();
cout<<" \n Enter any positive integer number \n ";
cin>>num;
int i=0;
while(1)
{
if(i>num)
            break;
sum=sum + i++;
}
cout<<"\n Sum of numbers till  "<<num<<" = "<<sum;
 getch();
 return 0;
}
```

Output

First Run

```
Enter any positive integer number
```

```
2
Sum of numbers till  2 = 3
```

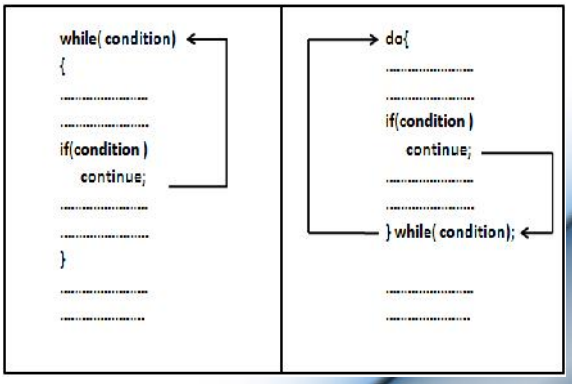
Second Run

```
Enter any positive integer number
5
Sum of numbers till  5 = 15
```

- The above program is same as program 7.1. Here the body of while loop runs infinitely. The only condition implemented is (i>num) if this condition evaluates to true the control will exit the while loop and proceed to execute the statements after the loop.

Continue:

- Sometimes there may be situations when you require the control to exit the loop or skip the particular iteration and go to the next. This can be done using the keyword **continue.**
- A **continue** statement causes the rest of the body of the loop to be omitted, for the current iteration. The control is transferred to the code that evaluates the normal test condition for loop termination.
- The continue statement is similar to the break statement but instead of terminating the loop, it transfers execution to the next iteration of the loop. It continues the loop after skipping the remaining statements in its current iteration.
- The general form of continue is as follows:



Example: Program to print the following pattern

1

1 2

1 2 3 4

1 2 3 4 5

Here the third line in the sequence is missing: 1 2 3

This pattern can be printed on the screen by using continue

```
/*****************************************
Program 8.2
Author : Nikhil Pawanikar
Description : Program to display the following pattern
using continue
                1
                1 2
                1 2 3 4
                1 2 3 4 5
*****************************************/
#include<iostream.h>
#include<conio.h>
int main()
{
 //int num,sum=0;;
 clrscr();

for(int i=0; i<=5 ; i++)
{
  if (i==3)continue;
  for(int j=1;j<=i;j++)
  {
            cout<<j;
  }
  cout<<endl;
}

 getch();
 return 0;
}
```

8.3 MANIPULATORS: ENDL , SETW ,SIZEOF.

The operators that are used to format the output that is supposed to be displayed on the screen are called Manipulators.

1. endl manipulator:

It is used to skip the current line and go to the next line. It is same as \n

2. setw() manipulator:

It is used to right align the output.

To use setw() manipulator, the header file <iomanip> has to be included in the program.

Example: Consider the program to input the contact information of a person and display the output formatted to the right.

```
/*************************************************************
Program 8.3
Author: Nikhil Pawanikar
Description: Program to display the use of manipulators setw() and
endl
*************************************************************/
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<string.h>
int main()
{
 char name[10],gender[6],location[10],contact_no[10];
 clrscr();
 cout<<"enter name:\t";
 cin>>name;
 cout<<"\nenter gender (m/f):\t";
 cin>>gender;
 cout<<"\nEnter Location: \t";
 cin>>location;
 cout<<"\nContact Number:\t";
 cin>>contact_no;
 cout<<"\NAME                    :"<<setw(15)<<name<<endl;
 cout<<"\nGENDER                  :"<<setw(15)<<gender<<endl;
 cout<<"\nLOCATION                :"<<setw(15)<<location<<endl;
 cout<<"\NCONTACT NUMBER          :"<<setw(15)<<contact_no;
 getch();
 return 0;
}
```

Output

```
enter name:       NIKHIL P
nenter gender (m/f):      MALE
Enter Location:    SANTACRUZ
Contact Number:       9988776655
```

NAME							N	I	K	H	I	L		P
GENDER											M	A	L	E
LOCATION						S	A	N	T	A	C	R	U	Z
CONTACT NUMBER					9	9	8	8	7	7	6	6	5	5

sizeof() operator

The **sizeof** operator yields the size of its operand with respect to the size of type **char**

Example:

```
/*************************************************************
Program 8.3
Author: Nikhil Pawanikar
Description: Program to display the use of manipulators setw() and
endl
*************************************************************/
#include <iostream.h>
#include<conio.h>
int main()
{
  char var_name[] = "INSTITUE OF DISTANCE & OPEN LEARNING!";

  cout  <<"The size of a char is: "
        <<"\nThe length of " << var_name <<" is: "
        <<sizeof( var_name);
  getch();
 return 0;
}
```

Output

```
The length of INSTITUE OF DISTANCE & OPEN LEARNING! is
38
```

8.4 TYPE CAST OPERATORS

- C++ provides Type cast **operators** to explicitly convert the type of variables or expression from one type to another.
- The general syntax is

```
type-name(expression)
```

- Here type-name is the target datatype we want to convert the variable expression into.
- Example: float p = **float(a);**
- **A typename behaves as if it is a function for converting a variable type into desired target type.**
- C++ has introduced the following new type cast operators:
  1. const_cast
  2. static_cast

3. dynamic_cast
4. reinterpret_cast

8.5 REVIEW QUESTIONS

1. Explain the use of break and continue with examples.
2. Explain the difference between break & continue.
3. Explain the use of Manipulators endl & setw.
4. Write short note on sizeof operator.

8.6 REFERENCES & FURTHER READING

1. Let us C – Yashwant Kanetkar, Chapter 3.
2. Object Oriented Programming with C++ - E. Balaguruswamy, Chapter 3
3. Programming in C++, Schaums Outlines – John R. Hubbard, Chapter 2.

9

Chapter 9
Introduction to Functions

Unit Structure

1. Introduction
    1. What is a function?
    2. Why use a function?
    3. How does it work in a program?
2. Types of Function
    1. Built-in functions
        1. Math Library Functions.
    2. User defined functions
        1. Local Variables in Functions
        2. Function Prototypes
3. Function overloading
4. Review Questions
5. References & Further Reading

1. INTRODUCTION

Usually programs are much larger than the programs that we have seen so far. To make large programs manageable and less complicated, they are broken down into subprograms. These subprograms are called functions.

The basic principle of Functions is ***DIVIDE AND CONQUER***. Using functions we can divide a larger task into smaller subtasks that are manageable.

1. What is a function?

1. A function is a self contained block of statements.
   A function is self contained in the sense it may have its own variables and constants

2. It is designed to do a well defined task.

Since a function is a part of a larger program (i.e. a subprogram) it has a particular job to perform.

3. It has a name.
   A function can be used (invoked) by the name given to it.

4. It may have return type
   A function invoked by a calling program may or may not return a value to it. In case it returns a value the functions return type is the same as the variables data type.

5. A program that has functions should have the following three things in it:
   a. Function Declaration or Prototype
   b. Function Call
   c. Function definition, which are discussed in a later part of this chapter.

2. Why use it?

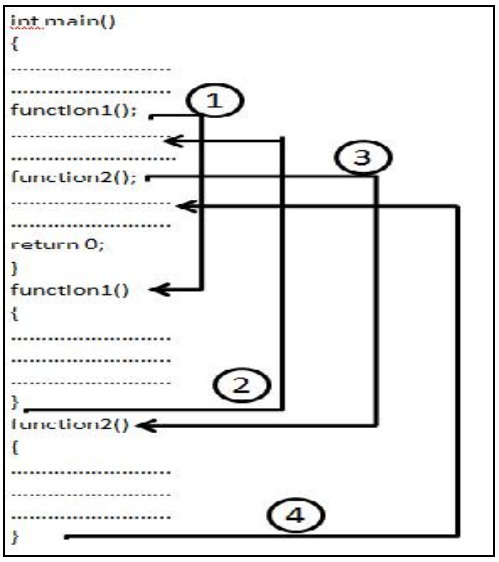
1. Functions are a structured way to programming. Larger programs get divided into smaller manageable units.

2. If a specific block of statements has to be executed multiple times (for example. taking contact details from 100 users), it can be written as a function and that function can be repeatedly executed. This implies that redundancy in writing the same piece of code multiple times is removed.

3. Dividing a large program into smaller subprograms using functions help to easily code them and reduces the debugging effort.

3. How does it work in an program?

Consider the following:



The program to the left contains three functions. First one is the main() function, second is function1() and third is function2().

The execution of any program begins with the execution of main function. Unless there is a decision or looping construct the execution of the program proceeds in a serial manner.

In the diagram to the left, the program execution begins with main(), all the statements get executed.

A function gets called when the function name is followed by a semicolon.

A function is defined when function name is followed by a pair of braces in which one or more statements may be present.

When the system encounters a call to function1() the program control jumps outside the main() function to execute the block of statements named function1() shown by arrow number1.

Once the last statement in function1() is executed the program control is again transferred to main() and the immediate statement after main is executed, shown by arrow number2.

When the system encounters a call to function2() the program control jumps outside the main() function again to execute the block of statements named function2() shown by arrow number 3.

Once the last statement in function2() is executed the program control is again transferred to main() and the immediate statement after main is executed, shown by arrow number 4.

2. TYPES OF FUNCTIONS

Functions are of two types

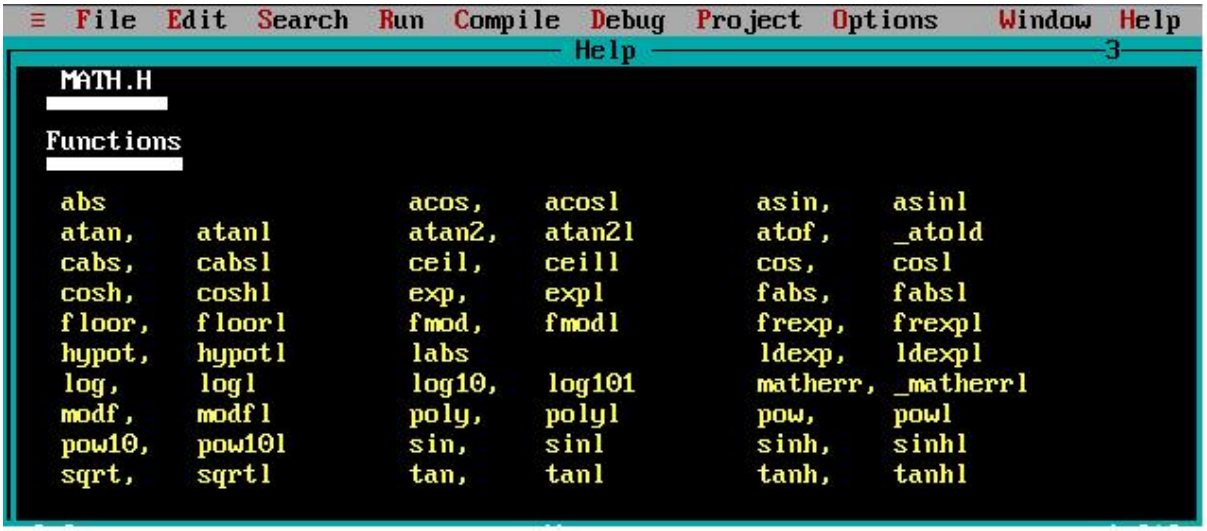
1. Built-in functions
2. User defined functions

1. Built-in Functions:

- These are also called Standard Library Functions. As the name suggests it is a Library of functions. These are the functions that are already present .i.e. predefined in the system.
- They have been written, compiled and placed in libraries under header files.
- We can use these functions in our programs by just specifying the name of the **header files** that contains the function of our interest.
- Example: we have already used a built-in function in chapter 8, the setw() manipulator. To use this function we have to add the header file <iomanip.h> to our program. The function definition for setw() is compiled and placed in the header file <iomanip.h>. If we do not include this file in our program and still use setw(), the compiler will return a **FUNCTION PROTOTYPE MISSING** error.
  For code refer to example on Page **>>pls enter page no of program 8.3 chapter 8<<**

1. Math Library Functions

- C++ provides a library of math related functions called **Math Library Functions.**
- These functions are placed in header file <math.h> and it contains 59 functions.
- The following is a snapshot of the help menu of Turbo C++ displaying the list of available built-in functions under <math.h>.



Example : Print Square Root of Numbers from 1 to 10

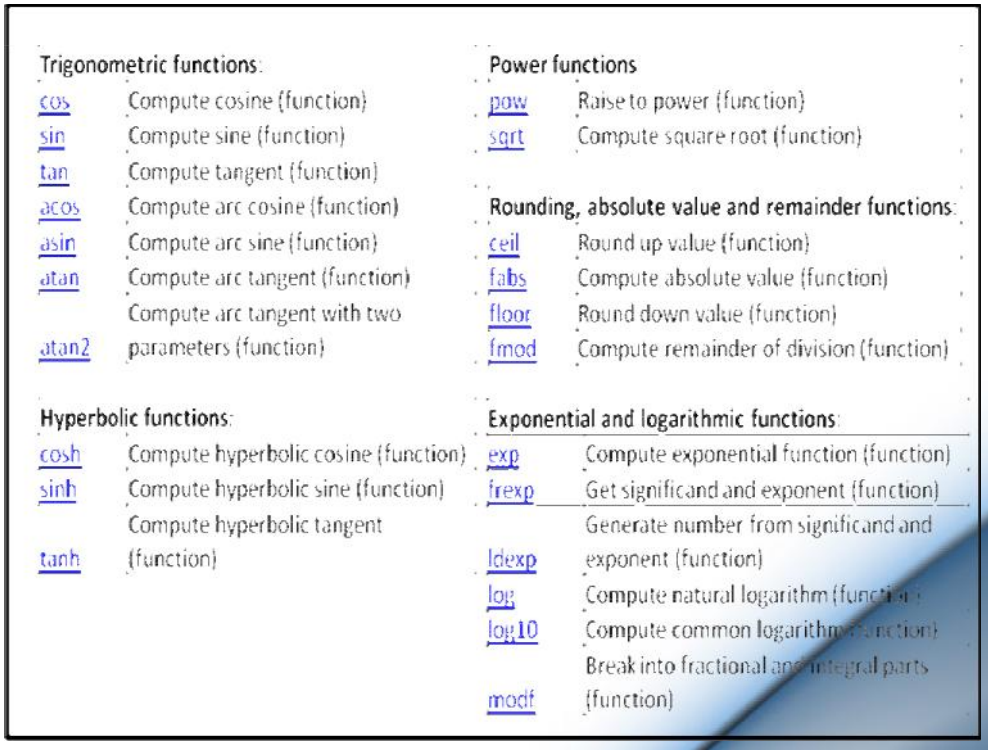
```
/*************************************************************
**************
Program 9.1
Author: Nikhil Pawanikar
Description: Program to display the use of math library functions
*************************************************************
**************/
#include <iostream.h>
#include<conio.h>
#include<math.h>c

int main()
{
  cout<<"Number"<<"\tSquare Root\n";
  for(int i=1;i<=10;i++)
        {
                cout<<i<<"\t"<<sqrt(i)<<endl;
        }
  getch();
 return 0;
}
```

Output

Number	Square Root
1	1
2	1.414214
3	1.7320513
4	2
5	2.236068

6	2.44949
7	2.645751
8	2.282427
9	3
10	3.162278

- This program prints the square roots of the numbers 1 through 10. The value of variable i from the loop counter is passed to sqrt(i).
- i is called the parameter passed to function sqrt.
- Each time the expression sqrt(x)is evaluated in the for loop, the sqrt() function is executed for the value of i passed to it.
- Its actual code is hidden away within the Standard C++ Library.
- Following are some of the functions available under the header file <math.h> and their uses:



Trigonometric functions:

cos	Compute cosine (function)
sin	Compute sine (function)
tan	Compute tangent (function)
acos	Compute arc cosine (function)
asin	Compute arc sine (function)
atan	Compute arc tangent (function)
atan2	Compute arc tangent with two parameters (function)

Hyperbolic functions:

cosh	Compute hyperbolic cosine (function)
sinh	Compute hyperbolic sine (function)
tanh	Compute hyperbolic tangent (function)

Power functions

pow	Raise to power (function)
sqrt	Compute square root (function)

Rounding, absolute value and remainder functions:

ceil	Round up value (function)
fabs	Compute absolute value (function)
floor	Round down value (function)
fmod	Compute remainder of division (function)

Exponential and logarithmic functions:

exp	Compute exponential function (function)
frexp	Get significand and exponent (function)
ldexp	Generate number from significand and exponent (function)
log	Compute natural logarithm (function)
log10	Compute common logarithm (function)
modf	Break into fractional and integral parts (function)

9.2.2. User defined functions:

These are the functions other than the Standard Library Functions. These are created by the users and the user has the flexibility to choose the function name, the statements that will be executed, the parameters that will be passed to the user & the return type of the function.

Any program using functions will have the following three necessary things:

1. **Function prototype or function declaration**
   It is the name of the function along with its return-type and parameter list.

2. **Function call**
   Any function name inside the main() followed by semicolon (;) is a Function Call.

3. **Function Definition:**
   A function name followed by a pair of parenthesis {} including one or more statements.

- In case of built-in functions, function prototype and function definition are not necessary, they have been already declared and defined in the libraries.
- Consider the following example:

```
return-type function1();

int main()
{
……..
……..
function1();
……..
……..
return 0;
}

function1()
{
……..
……..
return();
}
```

- The first line of the above example **return-type function1();** is called the **function prototype or function declaration**. It is used to declare the function to the compiler.
  This statement is always written outside(before) the main().

- The statement **function1()** along with the statements in the parenthesis shown below is called the **function definition.** The function definition contains the instructions to be executed when the function is called.

```
function1()
{
……..
…….
}
```

Function definition is always done outside the main().

- The statement **function1();** inside main() is a function call.
  A function gets called when the function name is followed by a semicolon.
  When this statement function1(); is executed the program control gets transferred to the the function definition of function1() which is outside the main(). All the statements inside function1() are executed and then the control gets transferred to the next statement after the function call.

Note:
From this point onwards Function prototype & Function definition means prototype & definition for a user-defined function, Since only user-defined functions have function prototype/ declaration and function definition.

Function Definition:

- A function is defined when function name is followed by a pair of braces in which one or more statements may be present.

- A function definition has 2 parts
  1. Function head
  2. Function Body

- Example :

```
int square(int x)
{                       //      returns
return x*x;
square of x:
}
```

- The function returns the square of the integer passed to it. Thus the call square(3) would return 9.

1. Function Head

- The syntax for the **head** of a function is

```
return-type
name(parameter-list)
```

- The above statement tells the compiler three things about the function:
  - **i.** **Name** of the function
  - ii. Its **return-type** i.e type of value to be returned by the function
  - iii. Its **parameter list**.

In the example shown above the head of the function is:

```
int square(int x)
```

- **i.** **square** is the name of the function,
- **ii.** **int** is the type of value that the function is returning to main()
- iii. and the parameter list **(int x)** contains a single parameter that is passed to the function square by the main()

2. Function Body

- The **body** of a function is the block of code that follows its head.
- It contains the code that performs the function's action.
- It includes the **return** statement that specifies the value that the function sends back to the place where it was called usually main().
- The body of the square function is

```
{
return x*x;          //          returns
square of x:
}
```

1. Local Variables in Functions

A variable can be declared inside a function definition, but it would be only local to the function. It cannot be used anywhere outside the function.

They exist only when the function is executing. The variables in the parameter list of function definition are called formal arguments and they are also local variables and exist only for the duration of the function execution.

2. Function Prototypes:

- The general syntax of a function prototype is

```
return-type     function-name
(parameter list);
```

- The above statement tells the compiler three things about the function:
  1. **Return-type** i.e type of value to be returned by the function
  2. **Name** of the function
  3. **Parameter list**. (the number of parameters the function will receive and their data-types).

- A Function Prototype is terminated by a semicolon

- Example: The complete program for finding square of a program is written as follows:

```
/*************************************************************
**************
Program 9.2
Author: Nikhil Pawanikar
Description: Program to display the square of a number entered by user.
             (Demonstrate the concept of function prototype)
*************************************************************
**************/
#include <iostream.h>
#include<conio.h>
int square(int m);                                  // Function Prototype
int main()
{
   int num, sqr=0;
```

```
  cout<<"\nEnter number to find its Square"<<"\t ";
  cin>>num;
  sqr=square(num);                                  // Function call
  cout<<"\nSquare of "<<num<<" = " <<sqr;
  getch();
  return 0;
 }

int square(int x)                                   // Function definition
{
return x*x;              // returns square of x:
}
```

Output:

First Run

```
Enter number to find its Square        5
Square of 5 = 25
```

Second Run

```
Enter number to find its Square
Square of 8 = 64
```

- The statement below is called **function declaration or function prototype.**

```
int
square(int m);
```

- The function prototype in the program above also contains the same:
  1. The function would return an **integer** value, hence, its return type is **int**
  2. The name of the function is **square**
  3. The function receives one parameter of type integer from the place where it is called from i.e. main().

- Following are some examples of function declaration:

```
float area(float length, float
breadth);
```

```
float perimeter(float side1,
float side2);
```

- Inside the function declaration each variable must be declared independently, the following declaration is invalid

```
float sum_of_angle(int angle1, int
angle2, angle3);
```

- The parameter names are optional in a function declaration, they are simply **dummy** variables.

```
float sum_of_angle(int, int, int);
```

- The above is valid since a Function prototype expects only the number of parameters and its data-types from the parameter list.

  For every function to be used there should be a function prototype. During program execution when the compiler encounters a function call, it first matches the prototype having the same number and type of arguments and then calls the appropriate function for execution.
- A function prototype is different from a function call, it(function call) does not indicate the return-type of the function.
- **Actual & Formal arguments:** In the program above, the statement sqr= square(num);

  The variable num being passed to the function square is called actual parameter & and the variable x in the function head of function square is called formal parameter.

3. FUNCTION OVERLOADING

- Overloading means using one thing for different purposes. C++ supports function overloading.
- Function overloading is also called **FUNCTION POLYMORPHISM**. Under this, the same function name can be used to create multiple function definitions to perform different tasks.
- It means that we can have a set of functions that have the same name but different signatures. A function signature includes its return type and parameter list.

- Example: an overloaded function multiply() is shown below with possible function prototypes and associated function calls and function definitions.

```
//function declarations
        int multiply(int m, int n);                          //prototype 1
        int multiply(int m, int n, int p); double            //prototype 2
        multiply(double m , double n); double                //prototype 3
        multiply(double m , int n); double                   //prototype 4
        multiply(int m, double n);                           //prototype 5
```

```
//function calls
        int mul = multiply(10, 20);
        int mul = multiply(10, 20,3);
        double mul = multiply(2.5, 3.5);
        double mul = multiply(1.2, 3);
        double mul = multiply(2, 3.5);
```

```
//function definitions

int multiply(int m, int n)
{
        return (m*n);
}

int multiply(int m, int n, int p)
{
        return (m*n*p);
}

double multiply(double m , double n)
{
        return (m*n);
}

double multiply(double m , int n)
{
        return (m*n);
}
double multiply(int m, double n)
{
        return (m*n);
}
```

When a function call is encountered the compiler tries to select the best function to be executed. To do this the compiler matches the prototype having the same number and type of arguments as in the function call and then invokes the appropriate function for execution.

9.4 REVIEW QUESTIONS

1. What is a function?
2. Explain types of functions?
3. Explain function prototyping.
4. Explain user-defined functions
5. Explain built-in functions with examples

9.5 REFERENCES & FURTHER READING

1. Let us C – Yashwant Kanetkar, Chapter 5.
2. Object Oriented Programming with C++ - E. Balaguruswamy, Chapter 4
3. Programming in C++, Schaums Outlines – John R. Hubbard, Chapter 5.

Solved Example:

Program to do arithmetic operations using function and switch case.

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

int getinput();
void arith(int);
void getdata();
void add();
void sub();
void mul();
void div();

double a, b;
int main()
{
        clrscr();
        int choice;
```

```
        while(1)
        {
        clrscr();
        choice= getinput();
        arith(choice);
        getch();
        }
        return  0;
}
int getinput()
{
        int input;
        cout<<"\n Select the operation you want to perform\n"
           <<"1. Addition\n"
           <<"2. Subtraction\n"
           <<"3. Multiplication\n"
           <<"4. Division\n"
           <<"5. Exit\n";
        cin>>input;
        return input;
}

void arith(int choice)
{
                switch(choice)
                {
                        case 1:
                                add();
                                break;
                        case 2:
                                sub();
                                break;
                        case 3:
                                mul();
                                break;
                        case 4:
                                div();
                                break;
                        case 5:
                                exit(1);
                        default:
                                cout<<"\n Please select a proper
input";
                                break;
                }
```

```
}
void getdata(double &a, double &b)
{
        cout<<"\nEnter the values of a and b\t";
        cin>>a>>b;
}

void add()
{
        getdata(a,b);
        cout<<"\n Result of Addition is  : "<<a+b;

}
void sub()
{
        getdata(a,b);
        cout<<"\n Result of Subtraction is  : "<<a-b;
}
void mul()
{
        getdata(a,b);
        cout<<"\n Result of Multiplication is  : "<<a*b;
}
void div()
{
        getdata(a,b);
        cout<<"\n Result of division is  : "<<a/b;
}
```

10

Chapter 10

Unit Structure

10.0 OBJECTIVES

1. INTRODUCTION

Now that we know what is a function? And how to use it? We can proceed to some advanced topics related to functions. There are two ways in which a function can be called, they are:

1.Call by value

2.Call by reference.

2. CALL BY VALUE

- The examples that we have seen above are all examples of call by value. In this method of calling a function we pass the value of variables to the function as parameters.
- Such function calls are called ‘call by value’. In call by value the changes made to the formal parameters do not change the actual parameters.
- The called function creates a new set of variables and copies the values of actual arguments into formal arguments.
- The function does not have access to the variables declared in the calling program and can only work on the copies of values i.e. the formal arguments.
- Example: Consider the following program for swapping of two numbers.

```
/*************************************************************
**************
Program
Author: Nikhil Pawanikar
Description: Program to swap the values of two numbers.
*************************************************************
**************/
#include <iostream.h>
#include <conio.h>
void swap(int,int);                    //prototype
int main(void)
{
        int a,b;
        cout << "Please enter 2 positive integers:\t ";
        cin >> a>>b;

        cout<<"\n Values before swapping are (in main ()):\n a = "
          <<a<<"\t b = "<<b<<endl;

        swap(a,b);                     //call by value, actual arguments

        cout<<"\n Values after swapping are (in main ()):\n a = "
          <<a<<"\t b = "<<b<<endl;
        getch();
        return 0;
}
void swap(int m,int n)                 //definition, formal arguments
{
        int temp;

        cout<<"\n Values before swapping are (in swap ()):\n m = "
          <<m<<"\t n = "<<n<<endl;
        temp = m;
        m = n;
        n = temp;

        cout<<"\n Values after swapping are(in swap ()):\n m = "
          <<m<<"\t n = "<<n<<endl;
}
```

Output:

First Run

```
Please enter 2 positive integers:        1 4
Values before swapping are (in main ()):
a = 1            b = 4
Values before swapping are (in swap ()):
m = 1            n = 4
Values after swapping are (in swap ()):
m = 4            n = 1
```

Thank You