Unit 2 Control Statements, Arrays and Pointers

Structure:

2.1 Introduction

In the previous unit, you studied about the basic concepts of object- oriented programming, the important components of programing language and the process of compilation and execution of a program. Programs are usually executed sequentially. In this unit, we will discuss the execution of programs based on control and iterative statement. Control statements execute program statements sequentially, based on certain conditions. Iteration statements cause block of statements to execute repeatedly. In this unit, we will also discuss arrays, multi-dimensional arrays and pointers.

Objectives:

After studying this unit, you should be able to:

- describe the conditional and unconditional statements
- discuss iteration statements
- define an array and explain how to declare different types of arrays
- explain the use of multidimensional arrays
- create a pointer and initialize values to the pointer

2.2 Control Statements

There are basically two types of control statements in C++ which allow the programmer to modify the regular sequential execution of statements. They are- conditional and unconditional control statements.

2.2.1 Conditional control statements

The conditional control statements allow the user to choose a set of statements for execution, depending on a condition. *If* statement and *switch* statements are the examples of conditional controls statements which executes based on the given condition. One more conditional control statement is the conditional operator. Now let us discuss the three conditional control statements one by one.

If, nested If statements

'if statement' is the simple conditional control statement. It controls the sequential flow of executable statements based on certain conditions.

'if' is the keyword used to specify the condition. 'if' statement is followed by executable statements that are enclosed in the curly braces. Curly braces are optional in case of single executable statement

The syntax of if statement is given as:

Syntax:

```
if (expression or condition) {
        Statement1;
          Statement 2;
         }
```

The expression or if condition inside the brackets returns the Boolean value, i.e. true or false. If the condition or expression is true, then execution will continue through statement1 and statement 2. If the condition is false, then the program will skip both the statements (i.e. statement1 and statement2).

If –else statement;

Another type of if statement is if-else statement. it is useful to execute series of statements. If the condition is true, then it executes if block statements. If the condition is false then it executes else block statements.

Syntax:

```
if (expression or condition)
                {
                    Statement (s)
                }
                else
                {
                     Statement (s);
            }
```

The example program given below accepts a number from the user, and displays whether given number is even or odd number.

```
# include <iostream. h>
void main(){
  int x;
  cout <<"enter the value of x\n'\";
  cin>>x;
  if(x%2 == 0)
cout <<" print even number";
 else
   cout<< " print odd number";
  getch();
}
```

Nested-If statement

If-else statement is useful only if condition has to satisfy two possible alternatives. If there is a need to check multiple alternatives, then use a nested-if statement, i.e. an “if statement” in an “if statement” (one if statement having another if statement, and so on is known as ‘nesting’). The syntax of nested-if statement is specified as:

Syntax:

```
if (condition1)
{
     if (condition 2)
        {
            Statement1;
```

```
            Statement2;
        }
      else if (condition3)
        {
          Statement3;
        }
  else
    Statement 4 ;
}
Next statements after if statement;
```

The flowchart for the nested-if statement is shown in figure 2.1.

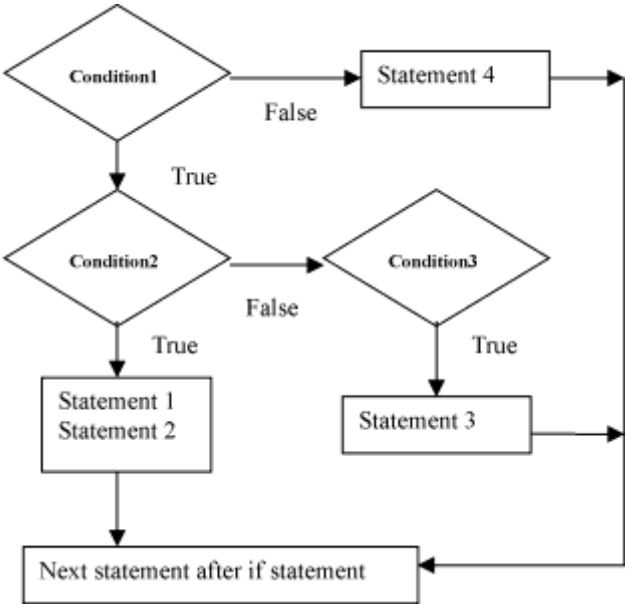

Figure 2.1: Nested- if statement

Note that && is the AND operator whereas || is the OR operator.

Multiple conditions can be checked using logical *&&* operator (AND) and || operator (OR). For example:

```
if ((condition1) && (condition2))
statement1;
else
statement2;
```

In the example given above, statement1 will be executed if both condition1 and condition2 are true. Otherwise, statement2 will be executed.

```
if ((condition1 || (condition2))
statement1;
else
statement2;
```

In the example given above, statement1 will be executed if either condition1 OR condition2 is true, and even if both are true. Statement2 will be executed if both the conditions are false.

Example: The program below demonstrates the use of **nested-if** statement.

```
#include <iostream.h>
void main ()
{
  int studentmarks = 80;
  if( studentmarks >= 70) {
    cout << "student passed with  A grade";
  }
  else {
    if( studentmarks >= 50) {
       cout << "student passed with  B grade";
    }
    else {
        if( marks >= 40) {
          cout << " student passed with  C grade!";
        }
        else {
          cout << " Better Luck next time";
     }
    }
  }
 }
```

The above program number reads student marks, and displays the grade obtained by the student. Here **nested-if** conditions are used to check the grade of a student. Once the condition is true, it displays corresponding message, i.e. the grade of the student.

Switch statement

If you are using more if- else statements i.e. nested-if statements in a program, then it looks a bit confusing. One alternative to nested-if is the switch statement. Switch statement helps you to check the different values of the same variable and execute the statements accordingly. The syntax of the switch statement is given below:

Syntax:

```
switch (variablename)
{
 case constant1: statement1;
                         break;
case constant2: statement2;
                         break;
case constant3:  statement3;
                         break;
default:       statement4;
}
```

If the variable in the switch statement is equal to constant1 then statement1 is executed, but if it is equal to constant2 then statement 2 is executed. If it is constant 3, then statement 3 is executed. If the variable value is not in any of the cases listed, then the default case statement i.e. statement 4 is executed. The default case specification is optional; however, keeping it is a good practice.

Every case should have a break statement as the last statement. Break statement takes the control out of the switch statement. Absence of the break statement can cause execution of statements in the next case. No break is necessary for the last case. In the above syntax, the default case does not contain a break statement.

The flowchart for the switch statement is shown in Figure 2.2.

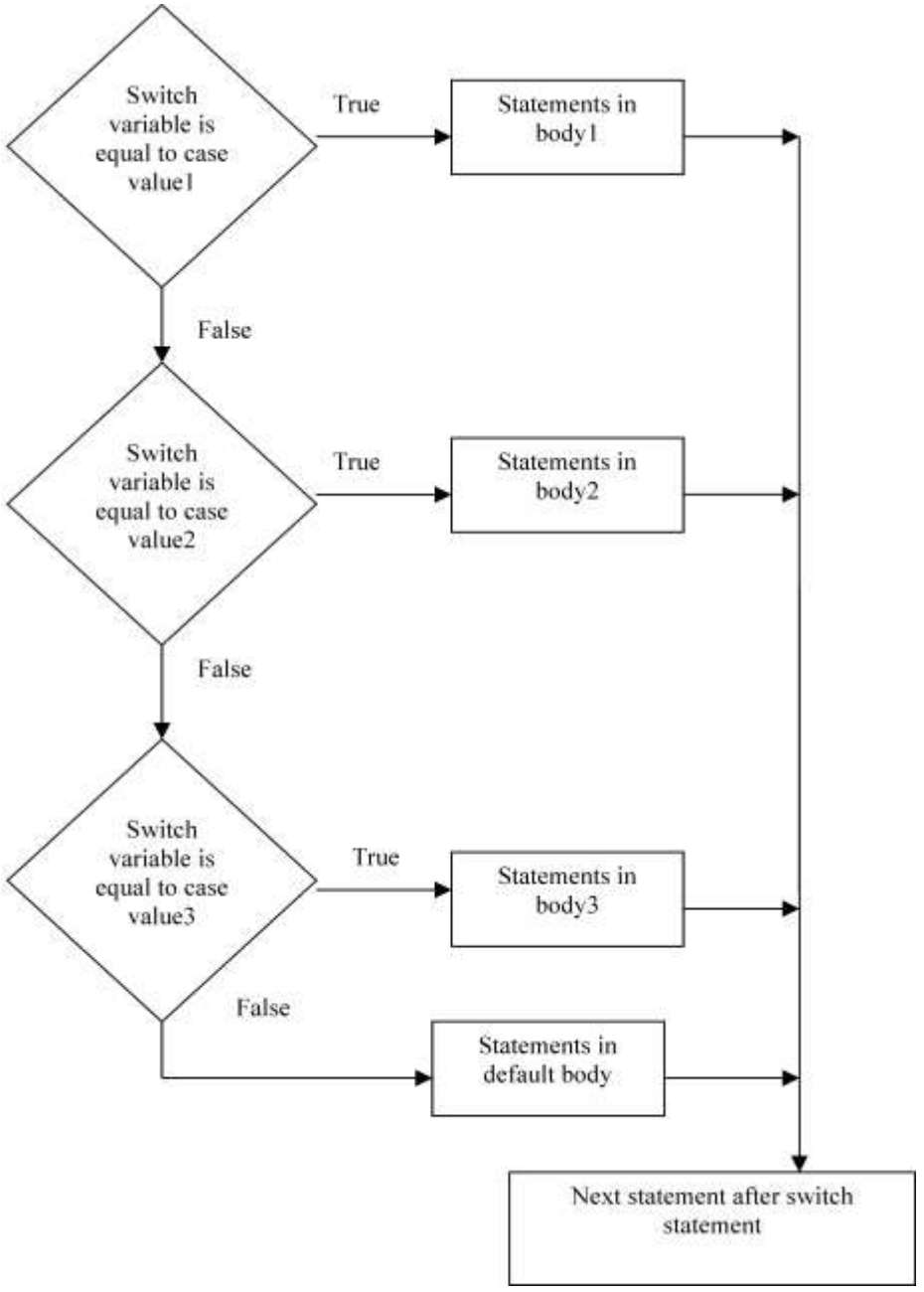

Figure 2.2: Switch Statement

Example: The following program implements the switch statement

```
# include<iostream.h>
void main(){
char x;
   float num1,num2;
   cout<<"Select an operator either + or - or * or / \n";
   cin>>x;
   cout<<"Enter two operands: ";
   cin>>num1>>num2;
      switch(x) {
      case '+':
         cout<<num1<<" + "<<num2<<" = "<<num1+num2;
         break;
      case '-':
         cout<<num1<<" - "<<num2<<" = "<<num1-num2;
         break;
      case '*':
         cout<<num1<<" * "<<num2<<" = "<<num1*num2;
         break;
      case '/':
         cout<<num1<<" / "<<num2<<" = "<<num1/num2;
         break;
      default:
                  cout<<" select any of the operator among + or - or * or / ";
            }
}
```

The example of switch statement given above is used to perform addition or subtraction or multiplication or division of two numbers. It reads operator values as character, and if the character value matches with any of the switch case values, then it executes a corresponding operation; otherwise it displays the default statement message.

Conditional Operator (?)

Conditional operator is also called ‘ternary operator’. The way a Conditional operator works is similar to that of if-else statements. The general syntax of conditional operator is:

(Condition)? True statement; False statement;

Here '?' is called 'ternary operator'; if the condition is true, it executes a true statement, otherwise it executes a false statement.

The statement given below compares two variables and assigns small value to c.

c= (a<b)? a; b;

This can also be rewritten as:

(a<b)? c =a; c=b;

Unconditional control statements

In order to stop the flow of execution unconditionally, C++ supports three types of unconditional control statements. They are: break, continue and exit statements.

Break, continue and exit statements

Break

When a break is encountered inside any conditional block or loop, a conditional block or loop is terminated, and control passes to the statement outside the conditional block. In the switch statement, a break statement is used in every case statement to terminate the switch statement, and this passes the control out of the switch block.

The following program demonstrates the use of break statement.

```
#include<iostream.h>
void main ()
{
int i=1,total=0;
if(i<10)
{
total= total+ i;
i++
if(i== 5)
break;
}
cout<<"total";
}
```

In the above program, the user reads two variables, i.e. i=1 and total=0. To find the sum of numbers, add i value to the total and increment the ‘i’ value by 1(represented with i++). Here, inside an if loop one more if loop is used. The first if loop checks whether the ‘i' is less than 10 or not. The second if loop verifies the condition whether i is equal to 5 or not. When i value becomes 5, then it executes the break statement that will terminate the loop, and then the control passes out of the if block.

Continue statement is used to take the control to the next iteration of the loop. It is used if the control has to be transferred to the next iteration of the loop based on a condition, skipping all the other statements in the loop. The following program shows the use of a continue statement.

Example

```
# include<iostream.h>
 void main()
{
  int i=1, n;
 cout< “enter the number” ;
  cin>>n;
   if(i<=n)
   {
     i++;
     if(i%2==0)
      cout<< i<<"\is even number \n";
     else
      continue;
    }
}
```

The above program enables you to use a continue statement. A Program reads the value of variable n and displays even number values up to n. There are two if conditions in a program: the first if condition checks whether i value is less than or equal to n, and the second if condition checks whether i value is an even number or not. If i value is even, it displays that value, or else the control of execution is passed to the continue statement. Here the continue statement, first of all, passes the control to if condition and helps to repeat the loop up to i value equal to n. Once i value becomes greater than

n, then the first if condition becomes false, and the control is passed out of loop.

Exit

Sometimes you need to end your program (or a sub section of a program) earlier than the normal termination. C++ supports exit statement to exit programs and returns an exit code. Exit is an inbuilt library function. To use exit program the header file *process.h* has to be included in the program.

The syntax of an exit program is specified as :

Exit (integer)

Here the integer value return to operating system (OS) while terminating program.

Exit (0) means - no error i.e. successful termination.

Exit (nonzero value) means– error while exiting the program.

```
#include<iostream>
#include<process.h>
int main()
{
 cout<" example program for exit';
 Exit(0);   // this statement terminates the program and returns 0 to OS
 cout <<' this statement will never executed";
}
```

The example shown above shows termination of the program before reaching the end of the program. Exit code 0 indicates a normal program termination.

Self Assessment Questions

1. Conditional operator is an alternative to _____ statement.
2. Each case in switch statement should end with _____ statement.
3. _____ statement is used to take the control to the next iteration of the loop.
4. _____ header file should be included to use exit function.
5. _____ Statement takes the control to the beginning of the loop.

2.3 Iteration statements

Loops or the iteration statements cause block of statements to execute repeatedly. There are three types of iterative statements: while, do while and for.

While loop

While statement starts with a condition and repeats block of statements as long as the condition is true. The syntax of while loop is:

Syntax:

```
while (condition)
        {
        //Body of the while loop
         Statement1;
        Statement 2;
        } Next statements after while loop;
```

In the syntax shown above, if the while condition is true, then it executes body of the while statement i.e. statement1 and statement2 are executed. After execution, the condition is checked again. If the condition is true, the statements inside the while loop are executed again. This continues until the loop condition becomes false.

The flowchart for the While statement is shown in Fig. 2.3.

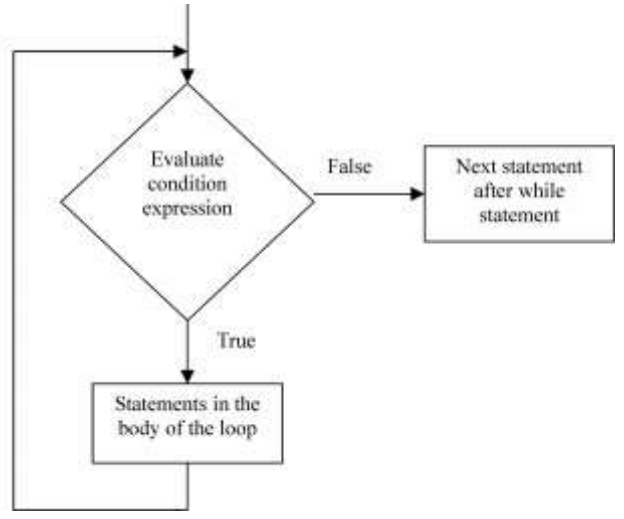

Figure 2.3: While Statement

The following program implements the while loop

```
#include<iostream.h>
void main () {
   int n, i = 1, factorial = 1;
   cout<< "Enter a positive integer: ";
   cin >> n;

   while ( i <= n) {
      factorial = factorial* i;
      i++;
   }
   cout<<"Factorial of "<<n<<" = "<<factorial;
   }
```

The program number accepts positive integer number n from the user and finds the factorial of that number. In the program shown above, the variables i and factorial are initialized to one. The variable i is incremented every time while loop becomes true. Calculate factorial value by multiplying i value with the factorial variable. Once i value becomes greater than n, while condition becomes false, and the control passes out of the loop and displays output i.e. factorial of a given number.

Do... while loop

Do-while is similar to while loop, except that a do--while loop body statement executes at least once before the condition is tested.

When the condition is true, the control statement jumps back to do statement, and the body of statements in loop executes again. This process repeats until condition becomes false.

In do-while loop, the condition is checked at the end of loop and so it is called exit-control statement, whereas while loop is called entry -control statement.

Syntax:

```
do
{
// do-while loop body of statements
```

Statement1;
Statement2
} while (condition expression);

Next statements after do-while condition;

In the syntax given above, statement1 and statement2 are executed, and the condition is checked. In other words, the do-while loop body statement executes before the condition. If the condition is true, then the statements are executed again, and if the condition is false, then the control is transferred to the next statement after the do-while statement.

The flowchart for the do... while statement is shown in Figure 2.4.

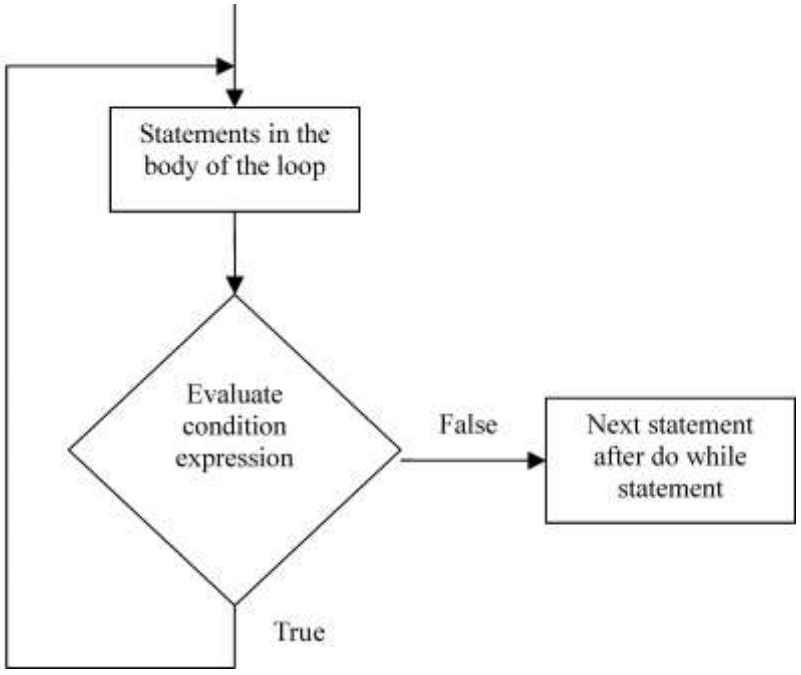

Figure 2.4: Do... while Statement

Please note that there is no semicolon after do, but there is a semicolon after the condition expression in while part.

The following program implements the do-while loop

```
#include<iostream.h>
int main() {
```

```
int  n, i = 1, factorial = 1;
cout<< "Enter a positive integer: ";
cin >> n;

do{
    factorial = factorial* i;
    i++;
}   while ( i <= n);
cout<<"Factorial of "<<n<<" = "<<factorial;
}
```

The decision on whether to use while or do.. while statement depends on whether the statements inside the loop have to be executed atleast once or not. If it has to be executed atleast once, then the do.. while statement should be used.

For loop

The for loops are the most useful type of loops. The syntax of the for loop is shown below. The loop condition contains three parts. They are: loop initialization, loop termination condition and statement for the next iteration, and these three parts are separated with semi-colons.

Syntax:

for(initialization statement; loop termination condition; statement to increment/decrement the loop variable)
{
//body of for loop;
Statement1;
Statement2;
}

In the example given above, initiation statement is executed first, and then the termination is checked. If the condition is true, body of the for loop is executed i.e. statement1 and statement2 are executed. Then the third statement in for loop is executed, which modifies the loop control variable.

The flowchart for the statement is shown in Figure 2.5.

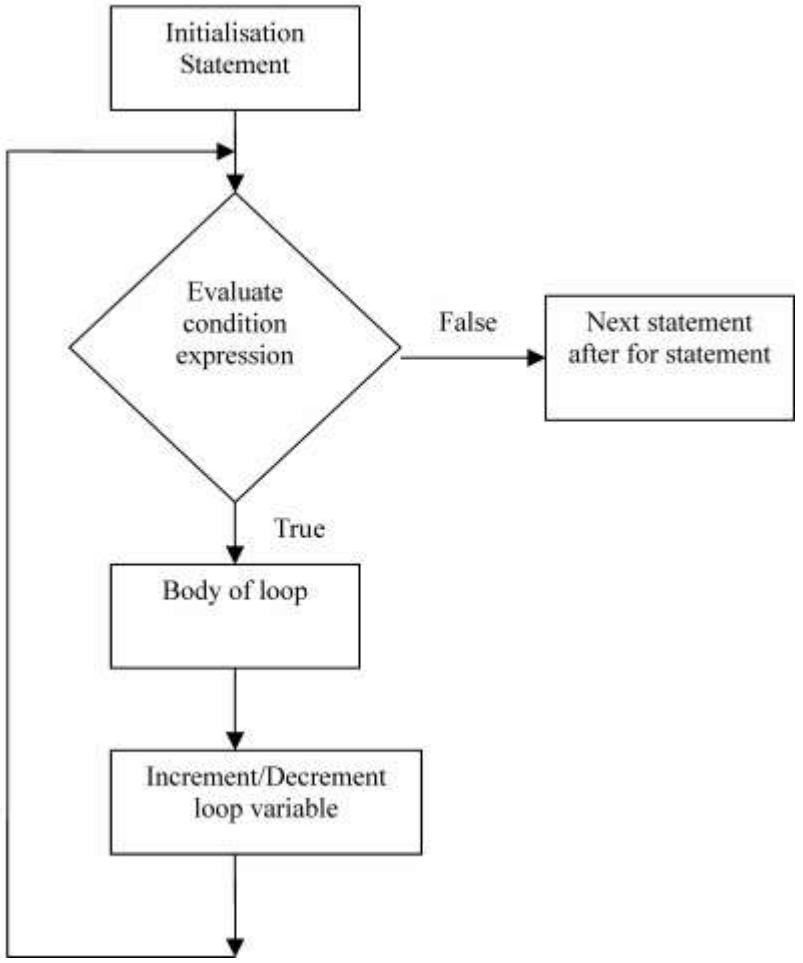

Figure 2.5: For Statement

The following program implements the factorial of a number using for loop

```
#include<iostream.h>
int main() {
    int  n, i , factorial = 1;
    cout<< "Enter a positive integer: ";
    cin >> n;

   for( i=1; i<=n; i++)
{
        factorial = factorial* i;
```

```
}
cout<<"Factorial of "<<n<<" = "<<factorial;
}
```

The for statement can also have multiple initialization and decrement and increment statements as shown in the following expression

```
for(i=0,j=0;i<=n;j++,i++)
{
Statements;
}
```

Please note that in the case of multiple statements, each statement should be separated by a comma.

Selection of the various types of loop to be used in the program depends on the programmer style. However, you should use for loop if you know in advance how many times the loop has to be executed. If the loop has to be executed at-least once, irrespective of whether the condition is true or false, then do-while loop should be used.

Self Assessment Questions

6. Which of the following is true for do-while loop?
   a. Entry control statement
   b. Exit control statement
   c. Temporary statement
   d. Conditional operator statement
7. Do-while loop will be executed at least once, even if the condition is _____.
8. The for statement can also have multiple initialization, and decrement and increment statements. (True/ False)

2.4 Introduction to Arrays

An Array is defined as a collection of elements of the same type. Till now you used variable to store single value In the case of an array, you can store multiple values of the same datatype in a single variable. The values of an array are called ‘array elements’. Supposing you stored four integer values in a single array, the array stores the values in a linear form. Each

element of the array can be accessed separately in any order. The size of the array matches with the number of values it contains.

2.4.1 Declaration and definition of arrays

Like variable declaration, arrays are also declared along with the data type. Array declaration reserves space for a number of elements. The declaration of array contains the name of the array along with square brackets. Inside the square brackets specify size of the array.

Int x[10];

An integer array x is declared with the size 10. The size is used to indicate the number of elements an array can store. The elements of an array can be accessed using the array name followed by index number within square brackets, i.e. x[0], x[1] to till x[9]. The index of an array starts with 0 and the last index number of an array is size-1.

2.4.2 Initialization of array

To specify values to an array, we should explicitly initialize specific values. You can specify declaration and initialization of the array in a single statement. Initialization of an array can be defined in the following form:

int a[5]= { 3, 6, 9,12,15}

The number of values between braces { } are not more than the size of the array. Declaration and initialization of the array can be done in a single statement. Then the above example can also be declared and initialized as follows:

Int a[]= { 3, 6, 9,12,15}

Supposing you declare the array and initialize values to it using the same statement, you can omit the size of the array.

The following program shows how you can add values of array to find the sum of elements of the array.

```
#include<iostream.h>
void main()

{
    int i, sum = 0;
    int  a[10] = {1, 2, 3,4,6,9,2,8,5,10};    //  array with size 10
```

```
    for (i=0; i<10; i++)

    {
         sum = sum + a[i];
    }

    // display the result...

    cout<<"\n Sum of values of an array is = "<<sum;

 }
```

Another interesting aspect of the array is that all the elements in the arrays are allotted consecutive spaces in the memory by the operating system.

The following program sorts an array. Here nested for loops are used to enable this.

```
# include<iostream.h>
void main()
{
int a[10],temp;
for (int i=0; i<10;i++)
{
cin>>a[i];
}
cout<<"Array after sorting is ";
for(int j=0; j<9;j++)
for(i=0;i<9;i++)
{ if (a[i]>a[i+1])
{temp=a[i];
a[i]=a[i+1];
a[i+1]=temp;
}
}
for (i=0; i<10;i++)
{
cout<<a[i]<<endl;
```

```
}
}
```

All the elements in the array should be initialized. Every data should be separated by a comma and all the elements enclosed within flower brackets are as shown below:

```
int data[5] = {3,5,7,4,9};
```

In the above initialization, data[0] is initialized to 3, and data[1] is initialized to 5 and so on..

3	data[0]
5	data[1]
7	data[2]
4	data[3]
9	data[4]

Self Assessment Questions

9. Arrays group the data of _____ type.
10. An array declared as int a[10] can store _____ integers.
11. In C++, the declaration a[5] represents the _____ integer in the array.
12. Array elements are accessed through _____.
13. When initializing values to an array, the number of values between braces { } are not more than the size of the array. (True/False)

2.5 Multidimensional Arrays

Multidimensional arrays can be described as “arrays of arrays”. The size of an array can be represented using a subscript. If an array has two subscripts we can call it a two dimensional array. If an array has more than one subscript, we can call it a multi-dimensional array. Multi-dimensional arrays contain as many indices as required, if it is not limited to two indices (i.e., two dimensions).

2.5.1 Two- Dimensional arrays

The simplest form of the multidimensional array is the two-dimensional array. For example, a two dimensional array can be created to store data of about

a matrix. A 3x4 matrix that has 12 data elements can be created using the following declaration

int A[3][4];

The array A[3][4] has three rows and four columns. The elements in the array can be accessed by two indices. To refer to an element in the first row and second column, A[0][1] has to be used since the index starts from zero.

The following program accepts two, 2x3 matrices from the user and adds the two matrices.

```
//matrix.cpp
# include <iostream.h>
void main()
{
int a[2][3], b[2][3], c[2][3];
int i,j;
cout<<"Enter the elements for the first 2x3 matrix";
for (i=0;i<2;i++)
for (j=0;j<3;j++)
{ cin>>a[i][j];}
cout<<"Enter the elements for the second 2x3 matrix";
for (i=0;i<2;i++)
for (j=0;j<3;j++)
{
cin>>b[i][j];
c[i][j]=a[i][j]+b[i][j];
}
cout<<"The resultant matrix is";
for ( i=0;i<2;i++)
{
for ( j=0;j<3;j++)
{
cout<<c[i][j]<<" ";
}
cout<< endl;
}
}
```

Multi-dimensional arrays can also be initialized by specifying row by row values. Every row elements are initialized by enclosing them in separate flower brackets.

int a[2][2] = {{ 3,4},{ 2, 7}}

In the example given above, elements 3 and 4 will be initialized to a[0][0] and a[0][1] respectively.

2.5.2 N- Dimensional arrays

Multi-dimensional arrays contain more than one index. If an array has 'n' indices, then it is called an 'n-dimensional array'. The general syntax of an n-dimensional array is:

Datatype name [Size1] [Size2]... [SizeN];

Example: if an array has 3 indices then it is called a 3-dimensional array.

Int P[2][3][4]

Self Assessment Questions

14. An array with the declaration int x[4][5] can store _____ numbers.
15. The array x[4][5] the number in the third row and fourth column can be accessed by _____.
16. int a[ ] [ ] is a valid declaration. (True/False)

2.6 Pointers

Pointers are a very powerful feature of the language with many uses in programming. Pointer is a variable that stores the address of another variable.

Some of the programing tasks are performed more easily with pointers such as dynamic memory allocation (i.e. obtaining memory at runtime from the system).

As you know, every variable created in a program is stored in the memory. The location of memory in a system can be accessed through the address. The address of a variable can be accessed using ampersand (&) operator.

If there is an integer variable named P in the program, and if you have statement cout<<&P, the program will display the address of the variable P, which will be a hexadecimal number.

2.6.1 Declaration and Initialization of pointers

Pointers are variables that store address values of other variables. They are defined by prefixing "*" (asterisk) to the variable name. The data type of the pointer variable can be any of the basic or user-defined data type, or even void. If a pointer is defined as a particular data type, then it can store addresses of variables of that data type alone. However, if the pointer is defined as void, then it can store addresses of variables of any data type.

The following statements create a pointer ptr and assign the address of the integer variable P. The declaration of pointer variable is the same as any other variable, except that you add an asterisk before the name of the pointer variable,

```
int *ptr;
int P=5;
ptr=&P;
```

The variable P can be accessed by two ways now: one, through the name P and the other, through the pointer.

The following statement displays the contents of P through the pointer ptr.

```
cout<<*ptr;
```

Here, * is known as the indirection operator or value of the variable pointed to by the pointer.

If you use cout<<ptr; statement in the program, it will simply display the address of the P or the contents stored in the ptr variable.

One of the important uses of pointers is getting memory during runtime. Pointer can be used with arrays. Both are related to each other. Arrays work similar to pointer to their first element. So, an array can always be implicitly converted to the pointer of the proper type.

Example: Let us declare two variables one is pointer type and another one is array type.

```
  int * A;
int B [10];
```

As we have already studied with regard to the array, brackets ([]) are used to specify the size of an array. We can assign pointer variable to an array. So, the following assignment statement is valid.

A = B;

Pointers and arrays support the same set of operations, with the same meaning for both. The main difference is that the pointers can be assigned new addresses, while the arrays cannot.

Following example illustrates the use of pointers and arrays

```
# include< iostream.h>
void main ()
{
  int A[5];
  int * p;
  p = A;   *p = 10;
  p++;  *p = 20;  p = &A[1];
  p++;  *p = 30;  p = &A[2];
  for (int n=0; n<3; n++)
    cout << A[n] << ", ";
  }
```

In the program given above, the statement p=A assigns values of an array to a pointer.

When Pointer p increments, address of pointer will move to the next address location. Here value of the array is assigned to the pointer. Using for loop, the program displays array values.

If we know the size of an array, we can easily assign values to the array using index without wastage of memory. When there is no idea about the size of an array in advance, array can be declared using pointer.

Pointer uses “New” operator to allocate memory to array during runtime and returns the memory location of the first element in the array. Using new operator, we can create (allocate) and delete (destroy) the objects dynamically.

The syntax of the new operator is:

```
Pointervariable = new data type [size];
```

where data type can be any basic or user-defined and the size is any integer value.

Delete operator is used to return the memory to the system. The syntax of delete operator is:

delete pointervariable;

where pointervariable is the pointer containing address returned by the system.

The following program creates a dynamic array, and finds the sum of array elements:

```
#include<iostream.h>
void main()
{
int n, sum=0;
int* ptr;
cout<<"enter the size of the array"<<endl;
cin>>n;
ptr=new int[n];
cout<<"enter"<<n<<"numbers";
for(int i=0;i<n;i++)
{cin>>ptr[i];
sum=sum+ ptr[i];}
}
cout<<"sum ="<<sum;
delete ptr;
getch();
}
```

In the program given above, the statement ptr=new int[n]; asks the system to allocate memory of size n. It store integer values and stores the address of the memory allotted in the pointer variable ptr.

Self Assessment questions

17. Pointers are variables that store _____.
18. _____ Operator can be used to request memory from the system during runtime.

2.7 Summary

Let us recapitulate the important points discussed in this unit.

- This unit discusses the various conditional control statements and iteration statements, arrays and pointers that are supported by C++.
- If statement and switch statement are the conditional control statements that are executed based on given condition.
- In the switch statement, the break statement is used in every case statement to terminate the switch statement and to pass the control out of the switch block.
- Continue statement is used to take the control to the next iteration of the loop.
- Iteration statements used to execute the block of statements for repeated number of times. There are three types of iterative statements: while, do while and for.
- Arrays are used to store large volumes of similar data together. The array elements can be accessed by specifying array name followed by the number which indicates the position of the element in the array.
- Arrays of basic datatypes such as integers, characters etc., and user-defined datatypes such as structures, objects can be defined. Arrays can be multidimensional, helping to store data that are related.
- Pointers are a very powerful feature of the language that has many uses in programming. Pointers are variables that store address values of other variables. They are defined by prefixing an * to the variable name.
- Pointer can be used with arrays. Both are related to each other. Pointer uses “New” operator to allocate memory to array during runtime, and returns the array of the memory location of the first element in the array.

2.8 Terminal Questions

1. Explain the following control statements:
   a. if-else b. conditional operator
2. What is the difference between the do-while and the while statements? Explain with syntaxes.
3. Define ‘array’. Discuss different types of array?
4. Write a program that stores 100 numbers in the array. The user should be able to find a particular number in the array.

5. Write a program that accepts a 3x3 matrix from the user and finds the transpose of it.
6. Explain the pointer operator “New” with an example program.

2.9 Answers

Self Assessment Questions

1. if –else statement
2. Break statement
3. Continue
4. process.h
5. break
6. B. Exit control statement
7. False
8. True
9. same
10. ten
11. sixth
12. array name followed by index number in square brackets
13. True
14. 20
15. a[2][3]
16. False
17. address values of other variables
18. New

Terminal Questions

1. **If-else:** Another type of if statement is if-else statement. If the condition is true, then it executes ‘if block’ statements. If the condition is false, then it executes ‘else block’ statements.

   Conditional operator: Conditional operator is also called ‘ternary operator’. Conditional operator works similar to if-else statements. For more details, refer section 2.2.1.
2. In do-while loop the condition is checked at the end of the loop. So it is called an ‘exit-control’ statement, whereas while loop is called ‘entry-control’ statement. For more details refer section 2.3

3. Array is defined as a collection of elements of the same type. If an array has two subscripts, we can call it a two-dimensional array. Multidimensional arrays contain as many indices as required. It is not limited to two indices (i.e., two dimensions). For more details refer section 2.4.

4.
```
//seqsearch.cpp
# include<iostream.h>
void main()
{ int a[100], num;
  int i, flag=0;
cout<<"please enter the numbers for the array";
for( i=0;i<100;i++)
{ cin>>a[i];
  }
cout<<"Enter the number to be searched";
cin>>num;
for(i=0;i<100;i++)
{
if (a[i]==num)
{ cout<<num<<" is found";
  flag=1;
  break;
}
}
if (flag==0)
cout<<num <<" not found";
}
```

5.
```
// program to read 3*3 matrix and transpose it.
# # include<iostream.h>
void main()
{
int a[3][3], at[3][3];
int i,j;
cout<<"Enter the elements for the 3x3 matrix";
for (i=0;i<3;i++)
for (j=0;j<3;j++)
```

```
{
cin>>a[i][j];
at[j][i]=a[i][j];
}
cout<<"The transpose is";
for ( i=0;i<3;i++)
{
for ( j=0;j<3;j++)
{
cout<<at[i][j]<<" ";
}
cout<< endl;
}
}
```

6. A Pointer uses the "New" operator to allocate memory to the array during runtime, and returns the array of the memory location of the first element in the array. For more details refer section 2.6.1.

References:

- *Learning C++ Programming Concepts* by Tickoo Sham, Pearson Education India.
- *C++ for Programmers* by Paul Deitel, Harvey M. Deitel, Pearson Education.
- *C++ Primer Plus* by Stephen Prata, Addison-Wesley Professional.
- *Object-oriented Programming with C++ - Sixth Edition* by E. Balagurusamy. Tata McGraw-Hill Education.