Unit 3 Functions and Structures

Structure:

3.1 Introduction

In previous unit, you have studied execution flow of programs based on certain conditions/statements. You have also studied how to store collection of values in single variable using array. In this unit, we will discuss the reuse of code using function, how to declare and define a function. We will also discuss scope and visibility of variables, use of storage classes, declaration of strings and how to initialize values to strings. Finally we will discuss user defined data types, structures and unions.

Objectives:

After studying this unit you should be able to:

- explain functions and how they enable program organization
- discuss different types of variables based on scope
- define strings and explain its uses
- describe different types of string library functions
- define structure and union
- differentiate between structures and unions

3.2 Introduction to Functions

Till now you have studied the usage of conditional, un-conditional statements and loops. You must have experienced how execution control can be passed from one statement to another statement. Now it's time to discuss functions. To perform individual tasks, user uses functions. One of the benefits of function is reusability. Reusability means use of existing code i.e. we have defined our code somewhere and then we are using it.

A function is defined as a group of statements that performs a task together and the user can execute them whenever required. There are two types of functions: predefined functions and user defined functions. You can use a predefined function without worrying about the code to implement. In section 3.5.2, you will study examples of predefined function or standard library functions such as strcpy, strlen. In this section, we will discuss how to define and implement user defined functions.

Declaration of function

Every user defined function should be declared before it is used. The declaration of function contains mainly three parts: function_name, return_type and set of parameters (parameter list). If a user wants to call and execute the code defined in function, he/she has to refer it through function name. The general syntax of function declaration is:

```
return_type   function_name (parameter list);
```

Function declaration is also called as **function prototype**. Function name together with parameter list is known as **function signature** and it does not include return type of function.

If a user returns value to function, the data type of the return value should be specified. If the user does not return any value, it should be specified as void. Every program starts with a default function - void main (), which means the function returns no value. When you have several users defined functions in a program, the statements in the main () function is executed first irrespective of where it is located in the program.

The input to function can be passed through parameters. Every function can have zero or more number of parameters. During declaration of function along with parameter name, the data type of parameters should be specified.

If function has more than one parameter they should be separated with a comma.

A function that accepts data from other functions does so by accepting one or more parameters from the sending function. Information that is passed to a function is called arguments to the function or simple arguments.

Definition of function

A function definition provides the actual body of the function. A c++ function definition consists of a function header and function body. Function header is similar to prototype of function. Function header specifies return type of function, function name and parameter list. The only difference between the header and the prototype is the semicolon. If you specify semicolon at the end of function heading, it generates a syntax error. Function body is group of statements or syntactical statement. Function needs to be declared before it is used, but can be defined anywhere in the program or linked file. The idea of declaring a function before defining it is called forward declaration. If you define a function before calling it, you do not necessarily need a definition.

The general syntax of function definition is:

```
return-type function_name (parameter list) // function header
 {
   Statements;    //Function body
 }
```

The arguments should match during function call with respect to data type and order of the arguments. In case of default arguments, the value need not be specified during call.

Let us discuss an example of declaration of a function named square which inputs a number and returns the square of the number.

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

Please note that all the arguments are enclosed within brackets. If there are no arguments, void should be specified within the brackets. The following is the declaration of a function which does not have return value and does not have any arguments.

```
void xyz(void);
```

The functions can be invoked or called by referring to the function name followed by the variables or constants that have to be passed as arguments. If there are no arguments that have to be passed to the functions, then the function name should be followed by a pair of parenthesis. The following program would invoke the xyz function:

xyz();

To invoke a square function which inputs a number and returns a number, following statement can be used:

s=square (n);

Where s and n are integer variables. The function will compute square of n and the result will be stored in variable s. Please note that during function call, constants can also be passed.

To understand the relevance of functions and its use, let us take an example. Let us suppose, you would like to write a program that computes the factorial of a number. We can define a user defined function named fact which would input a number and compute the factorial of that number and return the result. The declaration for the function will be:

int fact(int);

We can compute the factorial of n just by specifying fact (n) which would call the function fact and execute the statements

The following program (fact.cpp) explains how to calculate factorial of a number

```
//fact.cpp
# include <iostream.h>
#include< iostream.h>
#include<conio.h>
int fact (int);   // function declaration
int main()
{
   int n, i, result;
  cout<<" enter the number";
   cin>>n;
  result=fact (n);
```

```
  cout<<" resultant factorial value is"<< result;
  getch ();
  return 0;
}
int fact(int n)      // function definition
{
   Int i, factorial=1;
 for (i=1; i<=n; i++)
 {
   factorial=factorial *i;
}
return factorial;
}
```

The program receives input from user. The user enters any integer number as input. To calculate factorial of the particular number, the fact () function is invoked by passing variable n. When a compiler encounters function definition i.e. first statement of the function, it copies variable n value. The variables i and factorial are defined to calculate factorial. The value of n is constant or same through the function. By using for loop the factorial of number is calculated. The return statement is used to return result to main program.

If the function does not have any return value, the return statement can be skipped. The control goes back to the main program once the last statement in the program is encountered. Alternatively, simply using the return statement without any variable would also imply that no value is returned to the main program.

Self Assessment Questions

1. Function declaration is also called _____.
2. Function name together with parameter list is known as _____.
3. _____ Statement is used to return value to the main program.
4. Function call should always contain function name followed by parenthesis even if there are no arguments. (True/False)

3.3 Passing Data to Functions

After a function is declared, you must be concerned with how interaction will happen to function from main () function or other function. Interaction with a function can happen by passing data to a function or by returning values from function. Data can be passed to functions in two ways: (i) Pass by value and (ii) Pass by reference.

3.3.1 Pass by value

One method of passing data to function is **passing by value**. The *fact* function discussed in previous section implements passing by value. In this way of passing variables, a copy of the variable (main program) is created during function call with the name specified in the function and initialized with the value in the original variable. All the operations in the function are then performed on the function variable. The values in the variables declared in the main program remain unchanged by the function operations.

3.3.2 Pass by reference

Another alternative to passing arguments is **passing by reference**. When we pass argument by reference, no copy of the variable is created. However, the variables in the main program are referred to by different name in the function. Since no copy is created, when the values in the function variables are modified, the values in the variables in the main program are also modified. Pass by reference provides an easy mechanism for modifying the variables by functions and also enables to return multiple variables.

To pass arguments by reference, all the variable names in the argument list should be prefixed with & (ampersand) or address of operator when function is declared and defined. The following example (passingbyreference.cpp) shows the implementation of passing argument by reference.

Function is invoked by calling the function with a pair of parentheses containing any number of parameters. Pass by reference function call is the same as normal function call. Only difference is in case of pass by reference a copy of address of the actual parameter is stored.

```
//passingbyreference.cpp
# include <iostream.h>
int swap(int& m, int& n);                    // function declaration
void main()
{
```

```
int a,b ;
cout<< “enter two numbers”;
cin>>a>>b;
swap(a,b);
cout<<”The value of a is”<<a<<endl;
cout<<”The value of b is”<<b<<endl;
}
void swap(int& m, int& n)
{ int temp;
temp=m;
m=n;
n=temp;
}
```

In the above program, the variables a and b are passed by reference which implies that they will be accessed directly. However, the variables will be referred as m and n in the function and are swapped. The result is that the function swaps the values in the original variables a and b.

You can also have more than one user defined functions that can have same name and perform different operations. This is a powerful feature of C++ and is known as **function overloading**. Every overloaded function should however have a different prototype. The following example (printoverload.cpp) program implements an overloaded function print line ()

```
# include <iostream.h>
void printline();
void printline(char ch);
void printline(char ch, int n);
void main()
{
printline();
printline(“*”);
printline(“*”, 20);
}
void printline();
{ for(int i=0;i<25;i++)
 cout<<”-“;
cout<<endl;
```

```
}
void printline(char ch);
{for (int i=0;i<25;i++)
cout<<ch;
cout<<endl;
}
void printline(char ch, int n);
{ for (int i=0;i<n;i++)
cout<<ch;
cout<<endl;
}
```

In the above program, the function printline has three different prototypes depending on the arguments passed to it. The relevant function is invoked depending on the type and number of arguments passed to it.

Functions can also contain default arguments. **Default arguments** are those whose values need not be explicitly passed during function call. However, the default arguments should be specified in the end of the argument list to avoid ambiguity arising during function overloading. The default values are specified while declaring the function along with the data type of the argument. The variable name may or may not be specified during declaration.

The prgram (printoverload.cpp) can also be implemented through default arguments as shown below example (default.cpp).

```
//defaultarg.cpp
# include <iostream.h>
void printline(char ch=”*”, int n=25);
void main()
{
printline();
printline(“-”);
printline(“-”, 20);
}
void printline(char ch=”*”, int n=25);
{ for(int i=0;i<n;i++)
 cout<<ch;
```

```
cout<<endl;
}
```

In the example 3.4, variable ch and n values are declared as value of variable n is 25 and value of ch is *. However, if the values are specified explicitly, then the default values will not be considered.

Arrays can be used as arguments to functions. It is similar to pass any other variables as shown in the example program (matrix.cpp) below:

```
//matrix.cpp
# include<iostream.h>
void display(int arr[3][4]);
void main()
{ int matrix1[3][4], matrix2[3][4], sum[3][4];
int i,j;
cout<<"Enter the elements of matrix one";
for(i=0; i<3;i++)
for(j=0:j<4;j++)
cin>>matrix1[i][j];
cout<<"Enter the elements of matrix two";
for(i=0; i<3;i++)
for(j=0:j<4;j++)
{
cin>>matrix2[i][j];
sum=matrix1[i][j]+ matrix2[i][j];
}
cout<<"sum is";
display(sum);
}
void display(int arr[3][4] )
{
for(int i=0;i<3;i++)
{for(int j=0;j<3;j++)
cout<<arr[i][j]<<" ";
cout<<endl;}
}
```

Self Assessment questions

5. The values in the original variable remain unchanged during _____ type of function call.
6. The advantage of passing arguments by reference is _____.
7. The feature of C++ which enables two or more functions to have same name but different prototypes are known as _____.
8. Default values for the arguments have to be specified during _____.

3.4 Scope and Visibility of variables in Functions

Functions help to save memory as all the calls to the function uses the same code for execution. However, with function, there is overhead of execution time as there must be instructions for jump to the function, saving data in registers, pushing arguments to stack and removing them, restoring data, instructions for returning to the calling program and return values. To save execution time of functions, one of the features in c++ is inline functions.

Inline functions:

The main use of inline function is that it reduces the size of the code and improves in the speed of program execution. You should define inline function before the main () function. Inline functions are same like normal functions except that the function declaration begins with the keyword inline. Whenever you call inline function the function code is placed within the calling program.

The syntax of inline function is:

```
inline return-type name_of_function (arguments)
{
   Statements of function; // function body
}
```

Inline functions are suitable for small functions which have to be repeatedly called. This enables the user to write programs using functions which would make programs look neat and less lengthy, at the same time the actual code is inserted in all the places where the function is called after compilation enabling faster execution of the program.

Example program to calculate addition of three numbers using inline function is given in program below (inlineexample.cpp)

```
//inlineexample.cpp
#include<iostream.h>
#include<conio.h>
inline int add(int x, int y, int z)
{
return (x+y+z);
}
int main()
{
int x=6;
int y=4;
int z=2;
cout<<add(x,y,z);
return 0;
}
```

3.4.1 Storage Classes

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C++ program i.e. storage class is used to determine when variable is allocated, how long it lives, how it is accessed, where it is stored.

C++ supports different types of storage classes. They are: automatic, static and external variables.

Automatic

Automatic variable also called local variables and these are default variables.

The variables which we create and use in programs are automatic variables. The keyword auto is used to declare automatic variables, but since the variables declared in functions are automatic by default, this keyword may be dropped during variable declarations. The scope of the automatic variable is within the function block where it is declared. Once the execution of function is over, the value of automatic variable is lost. The lifetime of automatic variable is limited and hence it saves memory. By default, the memory of variable is removed, if variable is not used by function. If no value is assigned or initialized to automatic variable it takes junk value

depending on the value stored in memory. Therefore, it is good practice to initialize values to avoid debugging problems.

External

External variables are variables defined outside/external to any function.

External variables are recognized globally i.e. once variable is declared, it can be used by multiple functions. It is easy to share the external variable values among various functions. An external variable can be declared before main function so declaration statement is not specific to any one function. External variables are also called global variables and are automatically initialized to zero. However, too many external variables can create problems in debugging as the programmer will have tough time figuring out which function modified which data. Object oriented programming provides an alternative to use external variables. This feature is just to support backward compatibility with C.

The program (external.cpp) shown below illustrates the difference between an external variable and an automatic variable.

```
//external.cpp
# include<iostream.h>
int x;
void f1();
void f2();
void main()
{
x=x+2;
f1();
f2();
cout<<x;
}
void f1()
{ int x;
x=x*2;
}
void f2()
{ x++;
}
```

In the above program, we have defined an external variable x. This variable will be automatically initialized to zero. The statement x=x+2; will store two in x. The function f1 will not have any impact on the value of the external variable as there is another local automatic variable x defined in the function which will be modified (Please note that when automatic and global variable have same name, the local variable will override the global variable).

The function f2 will increment global variable by one. Thus, the final output of the above program will be 3.

Static

The scope of static variables is within the function in which they are defined like automatic variables. Static variables are the variables which are initialized and storage space is allocated only once at the beginning of program execution. The static variable retains its value until the end of program and it is automatically initialized to zero like the external variables.

A variable is declared static by prefixing it its declaration by the keyword static. Both local and global variables can be declared as static. When a local variable is declared as static, it is known as static local variable. When a global variable is declared as static, it is known as static global variable. The lifetime and initialization of static global variable is the same as the static local variable.

The below program (static.cpp) shows the use of the static variables.

```
// static.cpp
# include <iostream.h>
void main()
{
int num;
char ch;
do
{
cout<<”Enter a number”;
cin>>num;
f1(num);
cout<<”Do u want to enter another number”;
cin>>ch;
}while (ch==’y’)
```

```
}
void f1(int x)
{ static int sum, n;
sum=sum+x;
n=n+1;
avg=sum/n;
cout<< “Average is”<<avg;
}
```

The above program allows the user to enter any number of integers and computes the average of the numbers. The static variables sum and n are initialized to zero. They are updated when user enters a new number and the function is called. The values of these variables are retained even after the function returns to the main program.

The summary of all three storage classes is shown below:

	Automatic	**Static**	**External**
Visibility	Function	Function	Program
Lifetime	Function	Program	Program
Initialized to	Junk value in memory	Zero	Zero
Purpose	Variables used by single function	Same as automatic but the value should be retained when function terminates	Variables used by several functions

Self Assessment Questions

9. Inline functions help in saving _____ of programs
10. Static variables are initialized to _____ automatically.
11. External variables are accessible throughout the _____.

3.5 Strings

C++ implements strings as character array. We know that C language does not support a built in string type. We have to use character arrays to store and manipulate strings. One problem with character array is that memory crashes due to insufficient declaration. To avoid that problem C++ provides a new class called string. The string object may be used like any other built-in data type. C++ provides a data type "string", by using string data type we can declare and initialize string values easily.

Strings are used to store character names like name, address, password etc.

Stings are similar to arrays. Like array, string sizes are also defined during declaration statement. Main difference between string and array is that every string in c++ must be terminated by null character (’/0’) to mark end of the string. Strings, unlike other arrays can be input without using a loop. When inputting strings using cin statement, the compiler stops taking input from the user once it encounters space or linefeed (pressing enter key).

3.5.1 Declaration and initialization of strings

The string variable declaration and initialization can be done by using single statement. The strings can also be initialized as arrays.

The following example declares an array of 6 elements of type char initialized with the characters that form the word "hello" plus a *null character* '\0' at the end. It is specified by enclosing the text (hello) between double quotes.

char str[6]= “hello”;

Strings can also be defined without specifying the size, but in that case they have to be initialized to a string constant as shown in the following example.

char str[]=”hello world”

However, there is no built in mechanism in C++ that disallows the user to enter characters than the string maximum size. The extra characters entered by the user will be truncated. To keep a check on the number of characters, setw () function can also be used. To use this function, iomanip.h header file should be included. Let us see one example of it in the following program (stringexample.cpp).

```
#include<iostream.h>
#include<iomanip.h>
const int size=10;
void main()
{
char str[size];
cout<<”enter a string”;
cin>>setw(size)>>str;
}
```

In the program shown above, the user can enter only nine characters because last one character stores null character (’/0’). While the strings are input, cin stops reading once it encounters space or linefeed character (when user presses enter). To read the text containing blank space and to read multiple lines of text cin.get function can be used. The following statement will allow the user to input a maximum of 39 characters which can even include blanks. It will stop reading once it encounters linefeed character.

cin.get(str, 40);

To read multiple line of text from the user, you have to specify a terminating character which will be used to recognize end of input. In the following example, the terminating character is $.

cin.get(str,40,$);

The above example will allow users to enter a maximum of 39 characters which can include embedded blanks and linefeed.

3.5.2 Standard C++ String functions

C++ standard library provides several string functions used to manipulate strings. All string library functions are defined in the header file string.h. Whenever you use string standard function, the header file “strig.h” has to be included. The table 3.1 lists some of commonly used string functions

Table 3.1: String library functions

String Library function	Meaning
strlen()	Used to find the length of string
strrev()	Used to arrange the characters in the string variable in reverse order except for the null character.
strcpy	Used to copy the contents of one string to another.
strcmp	Used to compare two strings.

Strlen Function

This function is used to find the length of the string. The general syntax is:

strlen(string variable)

strlen(n) function returns the number of characters or length of the string n.

The return value of the function is integer.

Example:

```
 int X;
char s[10]= " SMUDDE"
X= strlen(s);
```

The number of characters in string s is six characters. So the length of the string is stored in integer variable X.

Strrev function

This function is used to arrange the characters in string in reverse order except for the null character. The general syntax is:

strrev(string variable)

This function returns the reversed string.

Example:

```
char A[10] = "HAI"
cout<<strrev(A);
```

The output will be IAH

Strcpy function

This function is used to copy the content of one string to another. The general syntax is:

strcpy(destination string, source string);

This function copies the source string content to destination string content.

Example:

```
char  p[10] = "OOPS";
char q[10];
strcpy(q, p);
cout<<q;
```

The output will be OOPS.

The strcpy can also use string constants to be assigned to a string variable like in the following statement.

strcpy(q,"OOPS");

In the statement shown above, the string constant OOPS is assigned to the string variable name q. The assignment of a string variable cannot be done using an assignment operator. It should be done using strcpy function.

Strcmp function

Strcmp function is used to compare two strings. The syntax of strcmp is :

strcmp(string1,string2)

Every character of string1 will be compared with the corresponding character of string2. The ASCII values of the character will be used to decide the return value. The function returns an integer and the value returned will differ based on the conditions as shown below:

If string1 < string2, value returned will be <0
If string1==string2, value returned will be 0
If string1> string2, value returned will be greater than zero.

Thus, a return value 0 implies that strings are equal and a non-zero return value implies that the two strings are not equal. Please note that the above function is case sensitive. strcmpi () function can be used if the two strings have to be compared without case sensitiveness.

The following program (stcmpexample.cpp) implements strcmp function

```
// stcmpexample.cpp
#include <iostream.h>
#include <string.h>
void main ()
{
  char str1[30], str2[30];
   for(int i=0;i<10;i++)
   {
    cout<<"Enter first string: ";
    cin>>str1;
    }
  for(int i=0;i<10;i++)
  {
    cout<<"Enter first string: ";
   cin>>str2;
  }
```

```
 if(strcmp(str1,str2)==0)
    cout<<"Both strings are equal";
 else
    cout<<"Strings are unequal";
 return 0;
}
```

Self Assessment Questions

12. The string variable declaration and initialization can be done by using single statement. (True/False)
13. _____ Function is used to copy the contents of one string to another.
14. To read blanks between the strings or to read multiple lines of text, _____ function which is a member function of cin can be used

3.6 Structures and Unions

In unit 2 you have learnt arrays and the process of grouping together similar type of data. We can also group together different type of data which are logically related. We can do this through structures and unions. In this section, we will discuss the use of structures and unions.

Structures

Structures are used for grouping together elements with dissimilar types.

The general syntax of structure declaration is:

```
struct name   {
  datatype member1;
 datatype member2;
   .....
   .....
};
```

For example, if you want to store all the details of an employee such as employee id (integer), name (string), department code(integer), and salary as one entity, you can do this using structure as shown below:

```
struct employee
{ int empid;
char name[35];
```

```
int deptcode;
float salary;
};
```

Structure is a feature in C++ that enables you to define a user-defined datatype. Once you specify the definition of the structure, you can create variables of that structure. In the above definition, we have defined a structure using the keyword struct followed by the name of the structure (employee). All the data items that need to be defined are defined by specifying the datatype and name of the variable. Once the definition is done, variables of type employee can be created. For example, the following statement creates a variable e1 of type employee.

```
employee e1;
```

Structure variables can be initialized during declaration. The values for all the members should be specified separated by comma and all the values enclosed within flower brackets as shown below

```
employee e1= {1234, “Ajay”, 12, 6900.00};
```

Structure variables can be copied and assigned to another structure variable as shown below

```
employee e2;
e2=e1;
```

To access the members of the structure variables, dot operators can be used. For example to access empid of structure variable e1, you would say e1.empid (structure variable. member variable name).

Structures can be nested within each other. When accessing the members and sub members dot operators can be used. Let us suppose you have defined a structure named distance with two members’ length and width as declared below:

```
struct distance
{ int feet;
  int inches;
}
```

Let us now define a structure named room, which contains variables of type distance:

```
struct room
{ distance length;
distance width;
distance height;
} bedroom;
```

In the definition of room given above, we have declared a variable bedroom of type room along with the definition. To access the members of the bedroom you can use:

bedroom.length.feet
bedroom.length.inches
bedroom.width.feet
bedroom.width.inches and so on

To initialize nested structures you have to group together all the elements of the structure:

room dining = {{12, 3},{13,0},{11,2}}

In the declaration of room dining, the values 12 and 3 refer to dining.length.feet and dining.length.inches respectively. Similarly, the second and third set of values represent feet and inches of width and height respectively.

You can also create array of structures. For this, the index number should be specified immediately after the structure variable. For example, the statement

employee emp[50] ;

will create an array of structure employee.

To access the first array member, you can say emp[0].empid

The below program (structexample.cpp) implements a structure point with x and y co-ordinates.

```
//structexample.cpp#include<iostream.h>
#include<conio.h>
# include<math.h>
struct point
```

```
{ int x,y;
} p1,p2;
void main()
{
int s;
cout<< “Enter the co-ordinates for first co-ordinate” ;
cin>> p1.x>>p1.y;
cout<<”enter the co-ordinates for second co-ordinate”;
cin>>p2.x>>p2.y;
s=sqrt(((p2.y-p1.y)*(p2.y-p1.y))+((p2.x-p1.x)*(p2.x-p1.x))) ;
cout<<endl<<”the distance between the two points is”<<s;
getch();
}
```

In the program shown above, we have defined a structure point which stores x and y co-ordinates of the point. Along with the declaration, we have also created two point variables p1 and p2. This is a way to create variables along with the structure declaration. This can be a separate statement also as discussed earlier. The program accepts two points from the user and displays the distance between them.

Structures are good mechanism of creating user defined datatypes. C++ specifies another method known as enumerated data type to enable users create their own datatypes. Enumerated data types can store fixed set of values and you can perform arithmetic on them as well. They are defined using the keyword enum. The program shown below (enumerated.cpp) creates an enumerated datatype that stores different days of the week.

```
//enumerated.cpp
#include <iostream.h>
enum weekday {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
void main ( )
{
weekday day1,day2;
day1=Mon;
day2= Fri;
int diff=day2-day1;
cout<<”days between =”<<diff;
if (day1<day2)
```

```
cout<<"day1 comes before day2";
}
```

In the program given above, enumerated datatype - weekday is created. The first value (Sun) is assigned zero by default and so on. The variables day1 and day2 can be assigned only the values specified in the datatype declaration or their integer equivalent. They cannot be assigned any other value.

Union

Union is a special type of class that can hold only one of its non-static data members at a time. Union declaration is similar to structure declaration as it allows us to group together dissimilar type of elements inside a single unit.

The difference between structures and unions with regard to their implementation is that the size of structure type is equal to sum of the sizes of individual members. However, the size of union is equal to the size of large number of element.

In simple words, we can say that unions are memory-efficient alternatives of structures particularly in situations where it is not required to access the different member elements simultaneously. Example of union is:

union result

```
{
 int marks;
  char grade;
  float percentage;
};
```

The union occupies 4 bytes in memory as it is the largest size member element. However, if we define the same with the help of structure, it will occupy
7 bytes of memory (i.e. sum of size of individual member elements).

So, we can say that unions are memory-efficient alternatives of structures particularly in situations where it is not required to access the different member elements simultaneously.

Self Assessment Questions

15. Structures and unions are used to group data of similar type. (True/False)

16. _____ Keyword is used to create structures.
17. Member data of the structure are accessed by _____ operator.

3.7 Summary

Let us recapitulate the important points of this unit:

- Functions provide good mechanism to organize data. Functions are given a name which can be used to call the function and execute the statements in the functions.
- Functions can also take inputs which are known as arguments which passed during function call.
- Arguments can be passed either by value or by reference. They can also have return value which can return a computed value back to the main program.
- Inline functions enable programmers to save processing time by reducing the overheads involved in function call.
- Programs can make use of automatic, external and static variables which have different types of utility depending on the way the variable should be accessed and whether the value has to be retained.
- Strings are nothing but character arrays. C++ standard library provides several string functions used to manipulate strings.
- Structures and unions data types enable users to create user defined data types. Structure, is a group data item which has heterogeneous collection of data.
- Enumerated data types create data types that can store pre-defined and fixed set of values.
- Union is a special class type that can hold only one of its non-static data members at a time.
- Union declaration is similar to structure declaration as it allows us to group together dissimilar type elements inside a single unit.

3.8 Terminal Questions

1. Differentiate between the way arguments are passed in terms of pass by value and pass by reference.
2. Define inline function. Explain it using example.

3. What is a string? Explain with example how string can be declared and initialized.
4. Differentiate between structures and unions.
5. Define a structure named product with elements productcode, description, unitprice and qtyinhand. Write a C++ program that implements the structure and enables to store at least 100 product data.
6. Write a function that takes input as radius of the circle from keyboard and return the area of the circle.
7. Write a function that takes two integers, finds out which is smaller and then assigns zero to the smaller variable and returns the value.

3.9 Answers

Self Assessment Questions

1. function prototype
2. function signature
3. return
4. True
5. pass by value
6. returning multiple values and modifying original values
7. Function overloading
8. declaration
9. execution time
10. zero
11. Program file
12. True
13. Strcpy
14. get
15. False
16. struct
17. dot operator (structure variable name.member name)

Terminal Questions

1. One method of passing data to function is passing by value. In this way of passing variables, a copy of the variable (main program) is created during function call with the name specified in the function and initialized with the value in the original variable.

Another alternative to passing arguments is passing by reference. In passing by reference, no copy of the variable is created. However, the variables in the main program are referred to by different name in the function. Refer to section 3.3.1 and 3.3.2 for more details.

2. The main use of inline function is it reduces the size of the code and speed up the programs. You should define inline function before the main () function. Inline functions are same like normal functions except that the function declaration begins with the keyword inline. Refer to section 3.4 for more details.

3. The string variable declaration and initialization can be done by using single statement. The strings can be also initialized as arrays. Refer to section 3.5.1 for more details.

4. The difference between structures and unions as far as their implementation is concerned. The size of structure type is equal to sum of the sizes of individual members. However the size of union is equal to the size of large number of element. Refer to section 3.6 for more details.

5.
```
//arrayproduct.cpp
# include <iostream.h>
structure product
{ int productcode;
char description;
float unitprice;
int qtyinhand;
}p[100];
void main()
{ int i;
for(i=0;i<100;i++)
{
cout<<”enter product code”;
cin>>p[i].productcode;
cout<<”enter product description”;
cin>>p[i].description;
cout<<”enter unit price”;
cin>>p[i].unitprice;
cout<<”enter qty in hand”;
```

```
cout<<p[i].qtyinhand;
}
}
```

6. //areacircle.cpp

```
# include <iostream.h>
int carea(int radius);                    // function declaration
void main()
{
int radius ;
cout<< “enter radius”;
cin>>radius;
cout<<“The area of circle is”<<carea(radius)<<endl;
}
int carea(int radius)        //function defintion
{ int a;
a= 3.14*radius*radius;
return a;
}
```

7. //smallzero.cpp

```
# include <iostream.h>
void smallzero (int& m, int& n);              // function declaration
void main()
{
int a,b ;
cout<< “enter two numbers”;
cin>>a>>b;
smallzero(a,b);
cout<<“The value of a is”<<a<<endl;
cout<<“The value of b is”<<b<<endl;
}
void smallzero(int& m, int& n)     //function definition
{ if (m<n)
m=0;
else if (m>n)
n=0
else
```

```
{
m=0;
n=0;
}
}
```

References:

- An Introduction to Object-Oriented Programming in C++, 2nd edition, By Graham M. Seed, Springer Science & Business Media.
- Object Oriented Programming with C++ - Sixth Edition, by E Balagurusamy. Tata McGraw-Hill Education.
- Object-Oriented Systems in C++, by Dr. Durgesh Pant, Mahesh Kumar Sharma, K.S. Vaisla, Firewall Media.