**IX-UNIT**

**UNDERSTAND USER DEFINED FUNCTIONS**

**9.1: Define function:**

 A function is **a self-contained block of code that performs a particular task**. Once a function has been designed and packed it can be treated as a black-box that **takes some data from the main program and returns a value.**

**Types of function:** there are 2-types of functions known as

1. **Predefined functions (or) library functions**
2. **User-define functions.**

|  |  |
| --- | --- |
| **Predefined functions (or) library functions** | **User-define functions.** |
| * They are **not required to implement**
* To use library functions we must **include** corresponding **header file** into our program.
* E.g.: printf(),scanf(),sqrt(),strcat()……….
 | * Where as a user-defined function **has to be developed** by the user at the time of writing a program.
* Here we need not include any header file to use it. just call to use it.
* E.g.: main ()…..
 |

* 1. **State the need for user defined functions:**
* If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit. These independently coded programs are called subprograms or functions that **are easier to understand, debug and test** instead of putting everything in main program that lead to program become too large and complex.
* In some times certain operations or calculations are repeated at many points throughout a program. In such situations, we may repeat the program statements whenever they are needed.

So, if we use the functions that can be called and used whenever required. This **saves both time and space.**

* 1. **Discuss the advantages of functions:**
* It **provides modularity to the program**.
* Easy to **re-use the code** by simply calling that functions.
* In case of large program with thousands of code lines, **debugging and editing and testing becomes easy if we use functions.**
	1. **Discuss the elements of function:** In order to use a user-defined function, we need to establish 3-elements that are related to functions.

Elements of functions are:

1 **function definition**

**2 function call**

**3 function declaration.**

1. Function definition (or) function implementation: Is an **independent program module** that is specially **written to implement the requirements of the function**. The general format of a function definition is:

|  |
| --- |
| **return\_type function\_name (arguments list)****{****Local-variable declaration;** **execution statement1;****execution statement1;****execution statement1;****…****…****…****return statement;****}** |

Function definition has following 6-elements:

1. Function name.
2. Function or return type.
3. List of parameters or arguments.
4. Local variable declarations.
5. Function statements.
6. A return statement.

All the six elements are grouped into 2-parts. Namely,

* **Function header(first three elements)**
* **Function body(second three elements)**
1. **Function header:** the header consists of 3-parts: the function type, The function name,The formal parameter list.

**🡪Return type and Function name**: return type **specifies the type of value**(like int,float…) **that the function is expected to return** to the calling function. If function is not returning any values, we need to specify the return type as **void.**

The function name is nay valid C identifier and follows same rules as for variable names in C.

🡪**The formal parameter list**: the parameter list **declares the variables that will receive the data sent by the calling program**. They serve as input data to the function to carry out the specified task.

 Parameter list contains declaration of variables separated by commas and surrounded by parentheses.

e.g.: float mul(int a, int b,int c){ ……… }

int sum(int a, int b){ ……… }

double power(double x,int n){……….}

Note: a function need not always receives values from the calling program. In such cases, we use keyword **void.**

 int sum(void)

 {

 ……..

 }

1. **Function Body**: contains declarations and **statements necessary for performing the required task**. The body enclosed in braces, contains
* Local variable declarations that specify the variables needed by the function
* Function statements that perform the required task.
* A return statement that returns the value evaluated by the function.

|  |  |
| --- | --- |
| **float mul(float x,float y)****{** **float r; /\*local variable\*/** **r=x\*y; /\*computes the product\*/** **return ( r); /\* returns the result\*/****}** | **void sum(int a,int b)****{****printf(“sum=%d”,a+b) /\*no local variables computes sum\*/** **return; /\* nothing returning\*/****}** |

* 1. **Discuss about return values and their types:**a function may or may not send back any value to the calling function, if it does, it is done through the **return** statement. The general form is:

 **return;**

 **or**

 **return(value);**

 **or**

 **return value;**

Note 1: a function may have **more than one return statements**.

E.g.: if(x>0)

 return 0;

 else

 return(1);

Note 2: if a function that do computations using doubles, yet return int. then return value is truncated to an integer.

 int product(void)

 {

 return(2.5\*3.0);

 }

Will return only 7.

* 1. **Define a function call:** A function can be called by simply using the function name followed by a list of actual parameters , enclosed in parentheses.

e.g.: main()

 {

 int y;

 y=mul(10,5); /\*function call\*/

 printf(“%d”, y);

 }

When the compiler encounters a function call, the control is transferred to the function mul(). The function call sends two integer values 10,5 to the function

 int mul(int x,int y)

* There are many different ways to call a function. Such as,

mul(10,5);

mul(m,5);

mul(m,n);

mul(m+5,5);

mul(10,n-2);

mul(10,mul(12,5));

Note 1: the **function used in expression**, **must return a value**.for e.g.: y=mul(p,q); in this the function definition of mul() must have a return statement.

Note 2: a function **can’t be used on the left side of assignment statements**.

mul(a,b)=15; illegal;

* 1. **Define function prototype or function decalration:**

Like variables, all functions in a C program can be declared, before they are used.

A function declaration or prototype consists of 4-parts.

* Function type
* Function name
* Parameter list
* Terminating semicolon.

General Format is:

 **Function-type function-name(parameter list);**

e.g.: int mul(int m,int n);

Note:

* The Parameter list must be separated by commas.
* The Parameters names do not need to be same in the prototype declaration and function definition.
* The parameters must match in type and in number and order
* Use of parameter names in declaration is optional.

The equivalent acceptable forms of declarations are:

 **int mul(int,int);**

 **int mul(int a, int b);**

 **void display(void);**

**A**prototype or function declaration may be placed in 2-places in a program:

1. **Above all the functions(global declaration)**
2. **Inside a function definition.**

**Note: prototype or function declaration is optional**

* 1. **Illustrate function declaration in programs:**

int sum(int,int); /\*function prototype or declaration\*/

int main()

{

 int a,b,result;

 printf("enter any 2 values\n");

 scanf("%d%d",&a,&b);

 result=sum(a,b); /\*calling function sum\*/

 printf("sum of two given numbers is%d",result);

 return 0;

}

int sum(int x,int y) /\*function definition\*/

{

 int result; /\*local variable declaration\*/

 result=x+y;

 return result; /\*return statement\*/

}

* 1. Discuss and illustrate **functionswith no arguments and no return values** with sample programs**:**
* When a function has no arguments, it does not receive any data from the calling function.
* When it does not return a value, the calling function does not receive any data from the called function.
* i.e., no data transfer between the calling function and the called function.

Function2()

{

………………

………………

……………….

………………..

………………..

}

Function1()

{

…………….

……………..

Function2()

……………..

……………..

}

 No input

 No output

**No data communication between functions**

|  |
| --- |
| /\* functios without arguments and without return values\*/void sum(); void main(){sum();}void sum(void){Int a,b;printf("entera, b values");scanf("%d%d",&a,&b);printf(“addition of two numbers=%d”,a+b);} |

9.10Discuss and illustrate **functions with arguments with no return values** with sample programs:

* The nature of data communication between the calling function and the called function with arguments but no return value is :
* The actual and formal arguments should match in number, type and order.
* The values of actual arguments are assigned to the formal arguments on a one to one basis , starting with the first argument as shown in below:

Function2(f)

{

…………………………

………………………

………….

}

Function1()

{

……………….

Function2(a)

……………….

……………….

}

 Values of arguments

 No return value

**One-way data communication**

|  |
| --- |
| /\* functios with arguments and without return values\*/void sum(int,int); void main(){inta,b;printf("entera,b values\n");scanf("%d%d",&a,&b);sum(a,b);}void sum(int x, int y){printf(“addition of two numbers=%d”,a+b);} |

9.10Discuss and illustrate **functions with arguments with return values** with sample programs:

* This type of functions will have two-way data communication as shown in fig:

Function2 (f)

{

……………..

……………..

…………….

…………..

…………..

return(e)

}

Function1()

{

………………..

Function2(a)

…………

……………….

}

 Values of arguments

 Function result

 **Two- way data communication between functions**

|  |
| --- |
| /\* functios with arguments and with return values\*/Int sum(int,int); Void main(){Int a,b,result;printf("entera,b values\n"); scanf("%d%d",&a,&b); result=sum(a,b); printf("addition of two numbers=%d",result);}int sum(int x, int y){int result;result=a+b;return(result); }  |

* 1. Discuss and illustrate **functions with no arguments with return values** with sampleprograms**:**
* In some occasions we may need to design functions that may not take any arguments but returns a value.

e.g: getchar() function , it cannot take any parameter from the calling program but returns an integer type data that represents a character.

|  |
| --- |
| /\* functios with out arguments and with return values\*/int sum();void main(){int result;; result=sum(); printf("addition of two numbers=%d",result);}int sum(void){inta,b,result; printf("entera,b values\n"); scanf("%d%d",&a,&b); result=a+b;return(result);} |

**9.13: illustrate functions that return multiple values with sample programs:**

1.We know a return statement can return only one value. **Suppose, we want to get more information from a function, we can’t use return statement**.

2. But, we can achieve this in C **using the arguments not only to receive information but also to send back information to the calling function.**

3. The arguments that are used to “send out” information are called “**output parameter**”.

4. The mechanism of sending back information through arguments is achieved using what are known as the **address operator (&)** and **indirection operator** (**\*)**.

For e.g.:

|  |
| --- |
| void mathoperation (int x,int y,int \*s,int \*d);void main(){ int x=20,y=10,s,d; mathoperation(x,y,&s,&d); printf("s=%d\n d=%d",s,d);}void mathoperation(int a, int b , int \*sum, int \*diff){ \*sum =a+b; \*diff=a-b;} |

When the function is called the following assignments occurs:

 Value of x to a

 Value of y to b

 Address of s to sum

 Address of d to diff

The sum & diff in the header store the address, not actual values of variables. Now the variables sum & diff point to the memory locations of s &d respectively.

 The statement \*sum=a+b adds values of a& b and the result is stored in the memory location pointed by sum, here the location pointed by sum is same as s.

 Similarly, the value of a-b is stored in the location pointed by diff, which is same as d.

Note: here, the s and d are passed by address for communicating with called function; this type of communication is called as “**pass –by-reference**” or “**pass-by-address**”.