PDEU PANDIT DEENDAYAL ENERGY UNIVERSITY

Formerly Pandit Deendayal Petroleum University

Computer Programming – I (24CP101T)

Dr. Rajeev Kumar Gupta
Assistant Professor
Pandit Deendayal Energy University
Gandhinagar, Gujarat

What is a Function

- A function is a self-contained block of statements that perform a coherent task of some kind.
- Every C program is a collection of these functions

Why Function ???

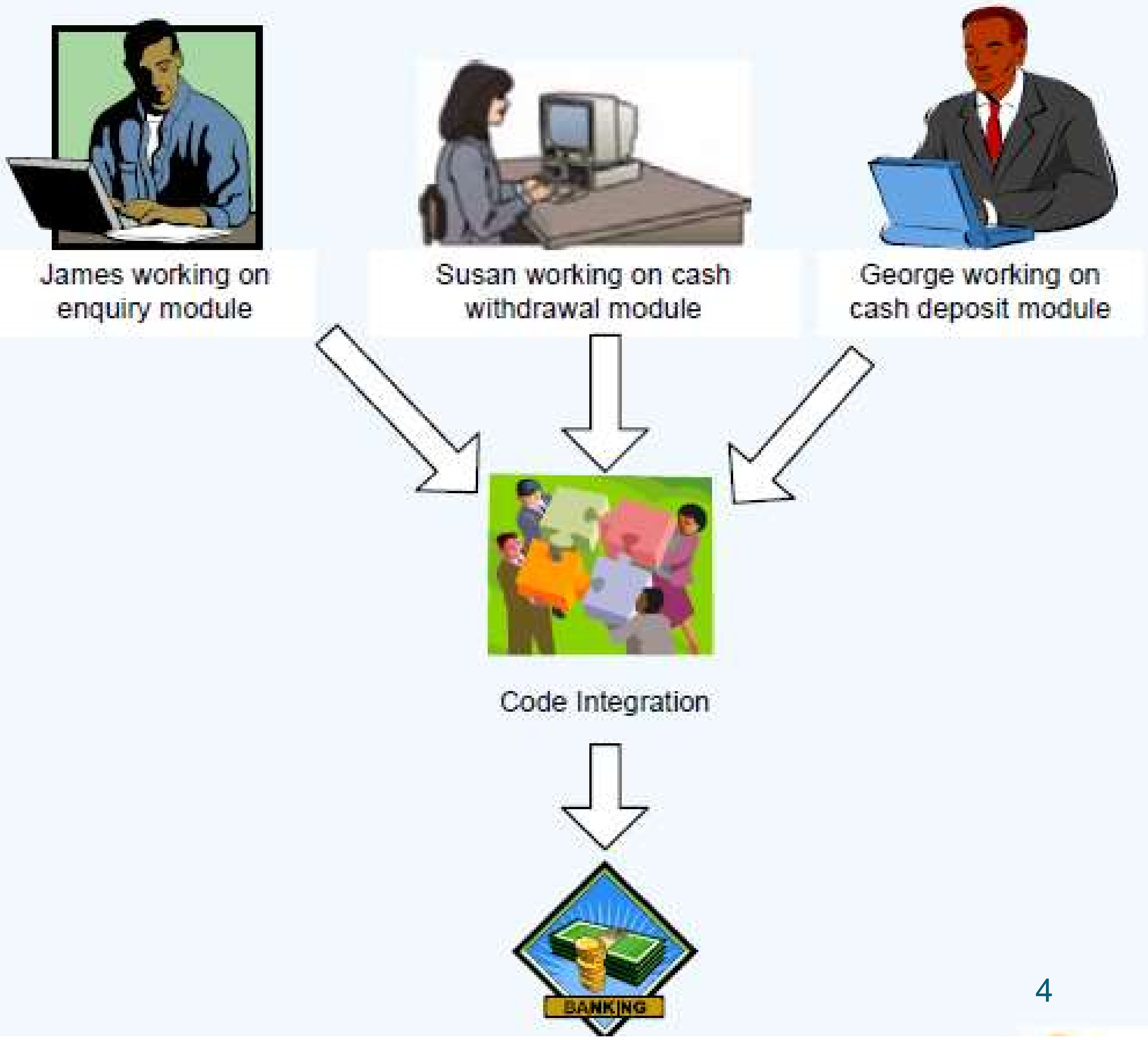
- Suppose you have a task that is always performed exactly in the same way

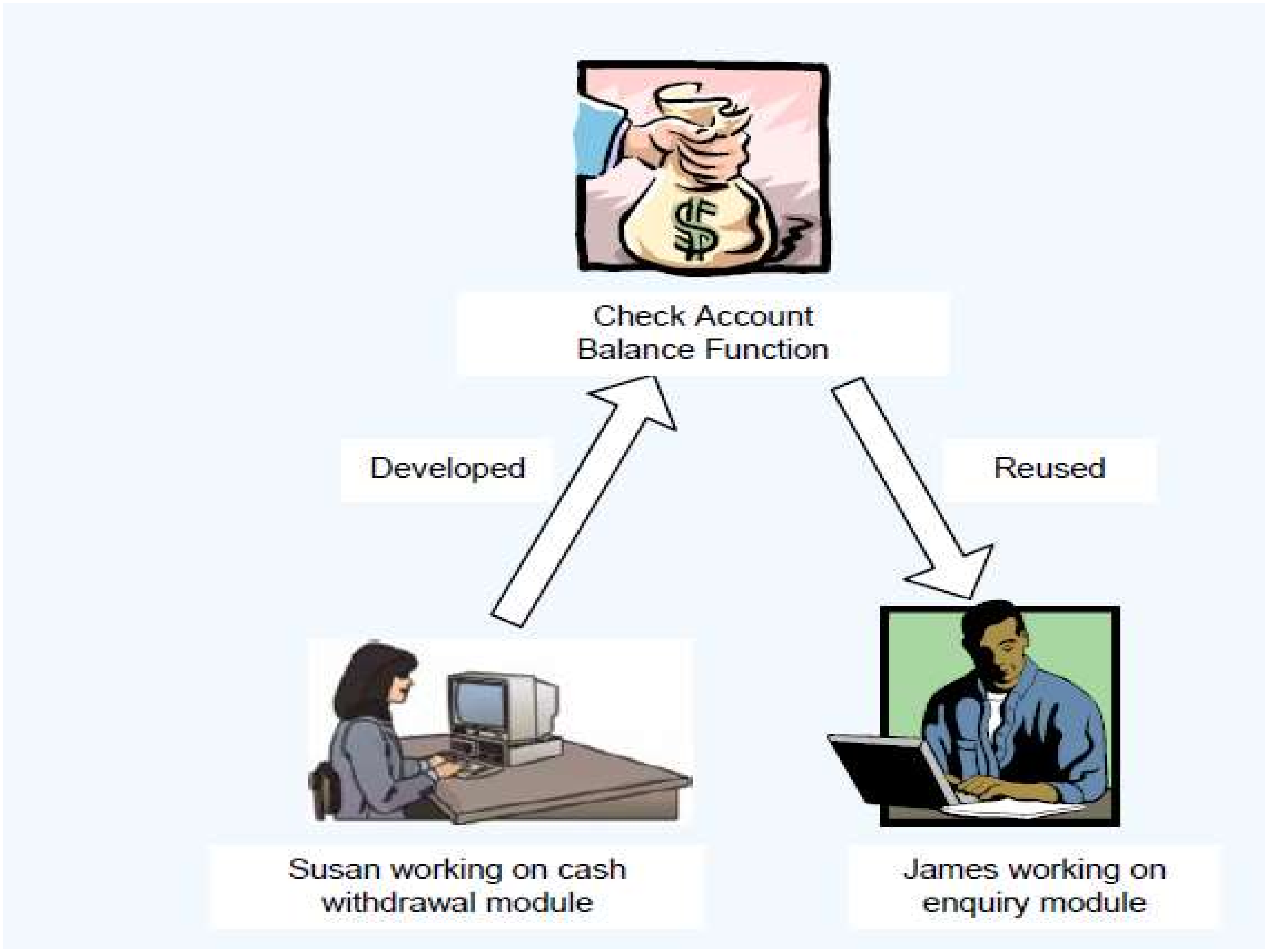
Advantages of Functions

1) The functions can be developed by different people and can be combined together as one application.
2) Easy to code and debug.
3) Functions support ***reusability.***

Advantages conti....

Work Allotment


James working on enquiry module
Susan working on cash withdrawal module
George working on cash deposit module
Code Integration
BANKING


Check Account Balance Function
Developed
Reused
Susan working on cash withdrawal module
James working on enquiry module

Classification of Functions

- **There are two type function in c**

1. Library functions
2. User Defined functions

Library functions

- Defined in the language
- Provided along with the compiler

Ex. **printf(), scanf() etc.**

User Defined functions

User Defined functions

- written by the user

Ex. **main**() or any other user-defined function

Note:A user defined function can become a part of the library function

Elements of a Function

- Function Declaration or Function Prototype
  - The function should be declared prior to its usage
- Function Definition
  - Implementing the function or writing the task of the function
  - Consists of
    - Function Header
    - Function Body
- Function Invocation or Function call
  - To utilize a function’s service, the function have to be invoked (called)

Declaring Function Prototypes

- A function prototype is the information to the compiler regarding the user defined **function name**, the **data type** and the **number of values** to be passed to the function and the **return data** type from the function

Syntax:

```
Return_data_type FunctionName(data_type arg1,data_type arg2,...,data_type argn);
```

Example:

```
int factorial(int n);
```

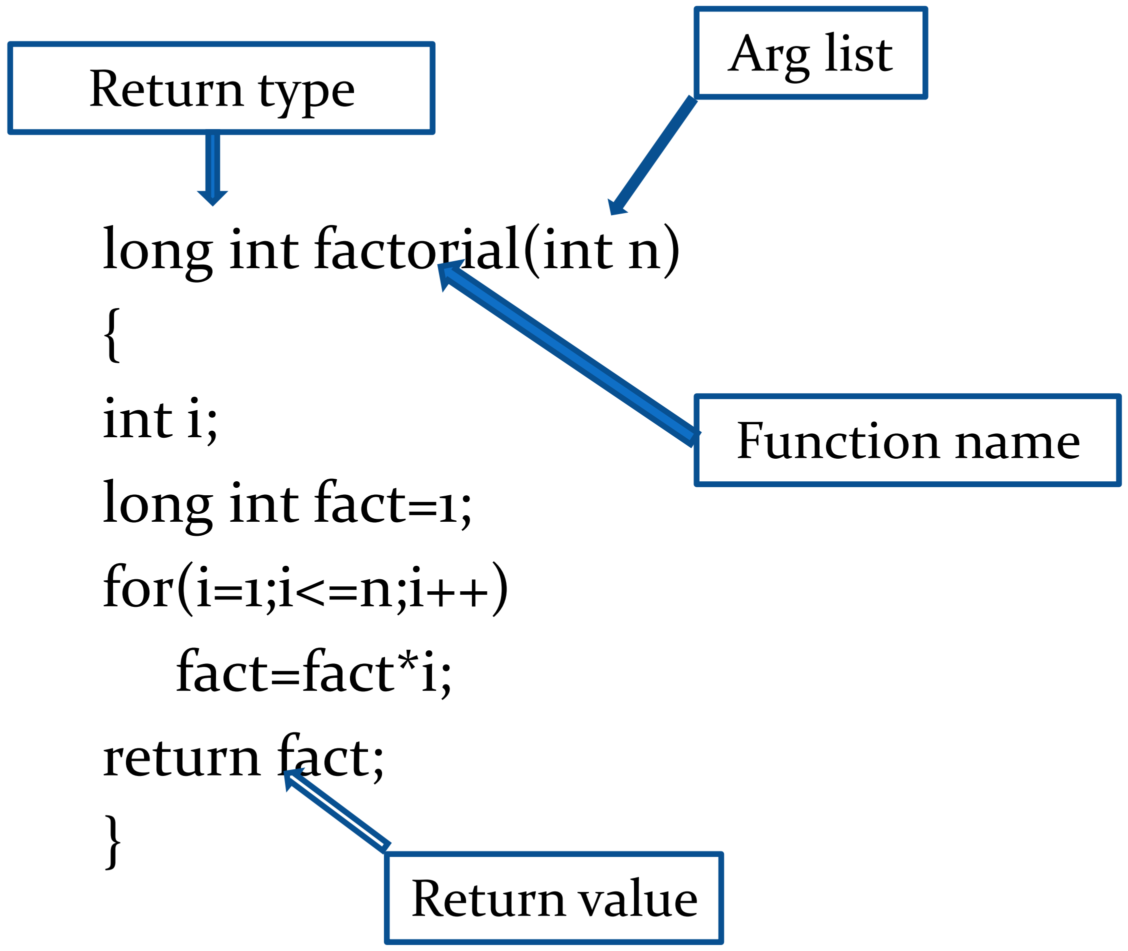
Function Definition

- A function header and body looks like this:

```
Return-data-type function-name(data-type argument-1,
data-type argument-2,….)
{
Local variable declarations;
                                    /* Write the body of the function here */
Statement(s);
return (expression);
}
```

- ➢The return data type can be any valid data type in C
- ➢If a function does not return anything then the void is the return type
- ➢A function header does not end with a semicolon
- ➢The return statement is optional. It is required only when a value has to be returned
- ➢By default return type is int in c.

Writing User-Defined Functions



```
long int factorial(int n)
{
int i;
long int fact=1;
for(i=1;i<=n;i++)
    fact=fact*i;
return fact;
}
```

Returning values

- The result of the function can be given back to the calling functions.
- Return statement is used to return a value to the calling function.

Syntax:

return (expression);

Example:

```
return (iNumber*iNumber);
return 0;
return (3);
return;
return (10 * i);
```

Calling Function

- A function is called by giving its name and passing the required arguments

Ex. fact=factorial(n);

1) The variables can also be sent as arguments to functions.
2) The constant can also be sent as arguments to functions.
3) Calling a function which does not return any value.
4) Calling a function that do not take any arguments and do not return anything.

Calling User-Defined Functions

```
Void main()
{
int n
long int fact
clrscr();
Printf("enter number\n");
Scanf("%d",&n);
fact=factorial(n);
Printf("factorial of %d =%ld",n,fact);
Printf("factorial=%ld",factorial(5));
getch();
}
```

Actual arg

Function call

formal arg

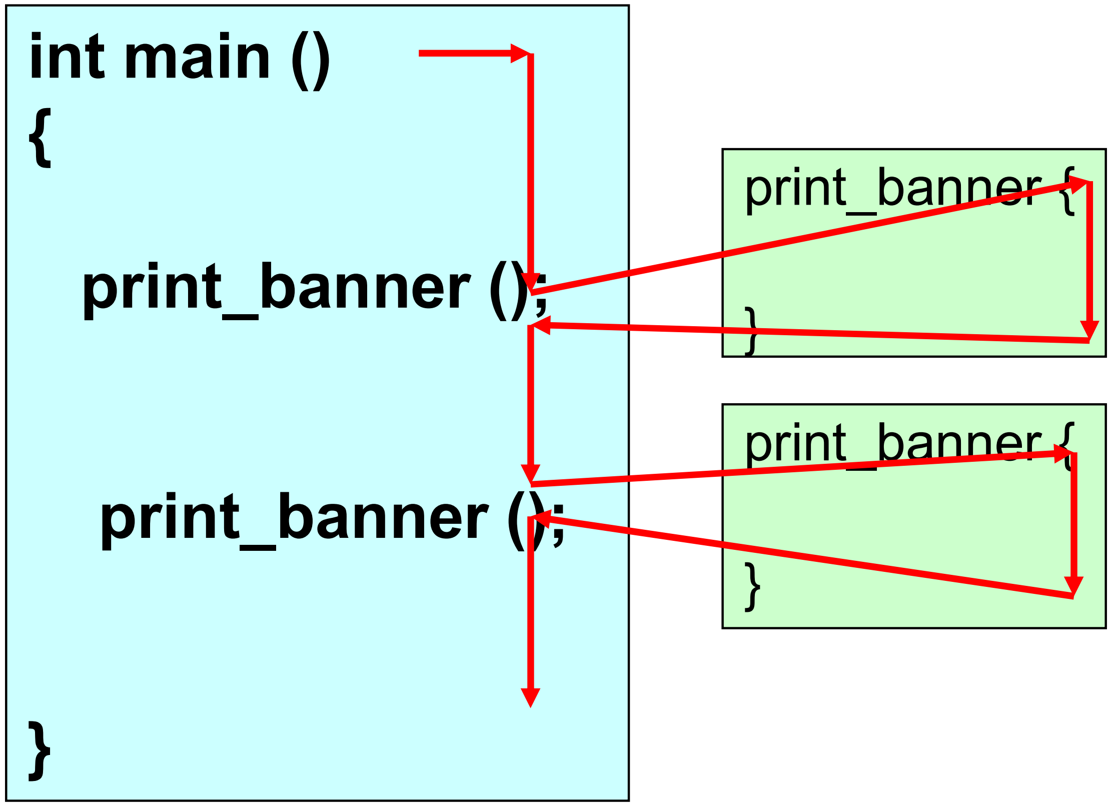
```
long int factorial(int n)
{
int i;
long int fact=1;
for(i=1;i<=n;i++)
    fact=fact*i;
return fact;
}
```

Function Control Flow

```
void print_banner ()
{
   printf("************\n"
   );
}
```

```
int main ()
{
     . . .
     print_banner ();
     . . .
     print_banner ();

}
```



Formal and Actual Parameters

- The variables declared in the function header are called as **formal parameters**
- The variables or constants that are passed in the function call are called as **actual parameters**
- The **formal parameter** names and **actual parameters** names can be the same or different.
- The types of the arguments passed and the parameters declared must match. If they don’t match, automatic type casting is applied. If these still do not match, there is an error.

Simple C function

```c
#include<stdio.h>
void message();  //fun prototype
Void main( )
{
  message( ) ;
  printf ("\nCry, and you stop the
monotony!" ) ;
}

void message( )
{
  printf ("\nSmile, and the world
smiles with you..." ) ;
}
```

```c
#include<stdio.h>
void message( )
{
  printf ("\nSmile, and the world
smiles with you..." ) ;
}


Void main( )
{
  message( ) ;
  printf ("\nCry, and you stop the
monotony!" ) ;
}
```

```
Smile, and the world smiles with you...
Cry, and you stop the monotony!
```

```c
italy( )
{
  printf ( "\nI am in italy" ) ;
}

brazil( )
{
  printf ( "\nI am in brazil" ) ;
}

argentina( )
{
  printf ( "\nI am in argentina" ) ;
}

main( )
{
printf ( "\nI am in main" ) ;
italy( ) ;
brazil( ) ;
argentina( ) ;
}
```

```
I am in main
I am in italy
I am in brazil
I am in argentina
```

Example-1

Finding the sum of two numbers using functions (No parameter passing and no return)

```
#include<stdio.h>
#include<conio.h>
void main()
{
void fnSum();
clrscr();
fnSum();
getch();
}
```

```
void fnSum()
 {
int n1,n2,Sum;
printf("\nEnter the two numbers:");
scanf("%d%d",&n1,&n2);
Sum = n1+n2;
printf("\nThe sum is %d\n",Sum);
}
```

```
Enter the two numbers:10
15
The sum is 25
```

```
main( )
{
 printf ( "\n am in main" ) ;
 italy( ) ;
 printf ( "\n am finally back in main" ) ;
}

italy( )
{
  printf ( "\n am in italy" ) ;
  brazil( ) ;
  printf ( "\n am back in italy" ) ;
}

brazil( )
{
  printf ( "\n am in brazil" ) ;
  argentina( ) ;
}
argentina( )
{
  printf ( "\n am in argentina" ) ;
}
```

```
I am in main
I am in italy
I am in brazil
I am in argentina
I am back in italy
I am finally back in main
```

Calling function (Caller)

Called function (Callee)

parameter

```
void main()
{ float cent, fahr;
  scanf(“%f”,&cent);
  fahr = cent2fahr(cent);
  printf(“%fC = %fF\n”,
   cent, fahr);
}
```

```
float cent2fahr(float data)
{
  float result;
  result = data*9/5 + 32;
  return result;
}
```

Parameter passed

Returning value

Calling/Invoking the cent2fahr function

Variable Scope

```
#include <stdio.h>
int A = 1;
void main()
{
myProc();
printf ( "A = %d\n", A);
}

void myProc()
{   int A = 2;
 if ( A==2 )
 {
  A = 3;
  printf ( "A = %d\n", A);
 }
}
```

Global variable

Hides the global A

```
Output:

A = 3


A = 1
```

Example-2

Finding the sum of two numbers using functions (parameter passing and no return)

```
#include<stdio.h>
#include<conio.h>
void fnSum(int , int);

void main()
{
int n1,n2;
clrscr();
printf("\nEnter the two numbers:");
scanf("%d%d",&n1,&n2);
fnSum(n1,n2);
getch();
}
```

```
void fnSum(int n1,int n2)
 {
int Sum;
Sum = n1+n2;
printf("\nThe sum is %d\n",Sum);
}
```

```
Enter the two numbers:10
15
The sum is 25
```

Example-3

Finding the sum of two numbers using functions (parameter passing and returning value)

```
#include<stdio.h>
#include<conio.h>
void main()
{
int fnSum(int , int);
int n1,n2,Sum;
clrscr();
printf("\nEnter the two numbers:");
scanf("%d%d",&n1,&n2);
Sum=fnSum(n1,n2);
printf("\nThe sum is %d\n",Sum);
getch();
}
```

```
int fnSum(int n1,int n2)
 {
int Sum;
Sum = n1+n2;
return Sum;
}
```

```
Enter the two numbers:10
15
The sum is 25
```

Conclusions of functions

1) Any C program contains at least one function.
2) If a program contains only one function, it must be main( ).
3) If a C program contains more than one function, then one (and only one) of these functions must be main( ), because program execution always begins with main( ).
4) There is no limit on the number of functions that might be present in a C program.
5) Each function in a program is called in the sequence specified by the function calls in main( ).
6) After each function has done its thing, control returns to main( ). When main( ) runs out of function calls, the program ends

Exercise

1) Write a program in C to find the square of any number using the function.
2) Write a Program to Reverse a string Using function.
3) Write a program in C to convert a decimal number to a binary number using the function.
4) Write a program in C to check whether a number is a prime number or not using the function.