Functions

Week 4

Functions

- Functions are the building blocks of C in which all programs activity occurs.
- Function in programming is a segment that groups a number of program statements to perform specific task.
- A C program has at least one function main( ).

Types of C Functions

- Library Function
- User defined Function

Library Function

- Library functions are the in-built function in C programming system.
- For example: main(), printf(), scanf()

User-Defined Function

- C permits programmer to define their own function according to their requirement known as user defined functions.
- A programmer wants to find factorial of a number and check whether it is prime or not in same program. Then, two separate user-defined functions in that program: one for finding factorial and other for checking whether it is prime or not.

Syntax of a Function Definition

Type-specifier function_name (parameter list)

{

Body of the function

}

Type-specifier

- Specifies the type of value that the function returns using the return statement.
- If no type is specified, the function is assumed to return an integer result.

Syntax of a Function Definition

Parameter List

- Is a comma-separated list of variables that receive the values of the arguments when the function is called.
- A function may be without parameters, in which case the parameter lists contains only the keyword **void**.

Return Statement

The return Statement

- It causes an immediate exit from the function it is in. That is, it causes program execution to return to the calling code.
- It can be used to return a value.

Syntax of return Statement

return (expression);

OR

return;

For example:

return;

return a;

return (a+b);

User- Defined Function Types

Value-Returning

- Always returns a single value to its caller and is called from within an expression.

Void

- Never returns a value to its caller, and is called as a separate statement.

Syntax of Function Call

function_name(argument(1),....argument(n));

- Control of the program cannot be transferred to user-defined function unless it is called (invoked).

Example hello.c

Function definition

Function call

```
#include <stdio.h>

/*
 * Print a simple greeting.
 */

void sayHello ( void )
{
  printf("Hello World!\n");
}

/*
 * Call a function which prints a
 * simple greeting.
 */

int main()
{
  sayHello();

  return 0;
}
```

Example: hello.c

Function name

Function body

```
#include <stdio.h>

/*
 * Print a simple greeting.
 */

void sayHello ( void )
{
  printf("Hello World!\n");
}

/*
 * Call a function which prints a
 * simple greeting.
 */

int main()
{

  sayHello();

  return 0;
}
```

Example: hello.c

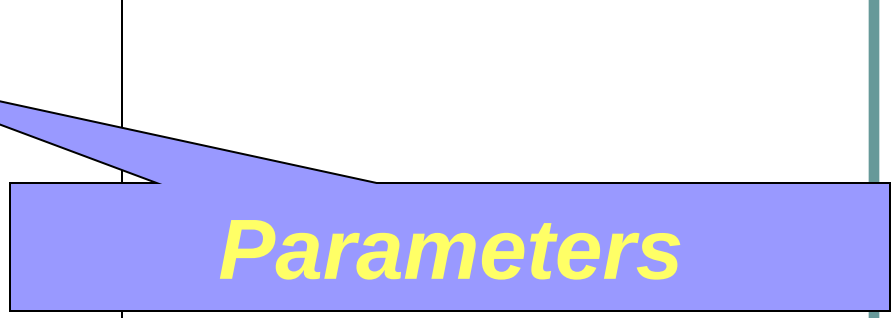
Return type

Formal Parameter List

```
#include <stdio.h>

/*
 * Print a simple greeting.
 */

void sayHello ( void )
{
  printf("Hello World!\n");
}

/*
 * Call a function which prints a
 * simple greeting.
 */

int main()
{

  sayHello();

  return 0;
}
```

Example: sort.c

```
/* Print two numbers in order. */

void Sort ( int a, int b )
{
  int temp;

  if ( a > b )
  {
    printf("%d %d\n", b, a);
  }
  else
  {
    printf("%d %d\n", a, b);
  }
}
```



Example: sort.c

Formal parameters

```
/* Print two numbers in order. */

void Sort ( int a, int b )
{
  int temp;

  if ( a > b )
  {
    printf("%d %d\n", b, a);
  }
  else
  {
    printf("%d %d\n", a, b);
  }
}
```

Actual parameters

```
int main()
{
  int x = 3, y = 5;

  Sort ( 10, 9 );
  Sort ( y, x + 4 );
  return 0;
}
```

Function Prototype (declaration)

- Every function in C programming should be declared before they are used. These type of declaration are also called function prototype.
- Function prototype gives compiler information about function name, type of arguments to be passed and return type.

Syntax of Function Prototype

return_type function_name(type(1) argument(1),....,type(n) argument(n));

int add (int a, int b);

1. name of the function is add()
2. return type of the function is int.
3. two arguments of type int are passed to function.

- Function prototype are not needed if user-definition function is written before main() function.

Working of Function

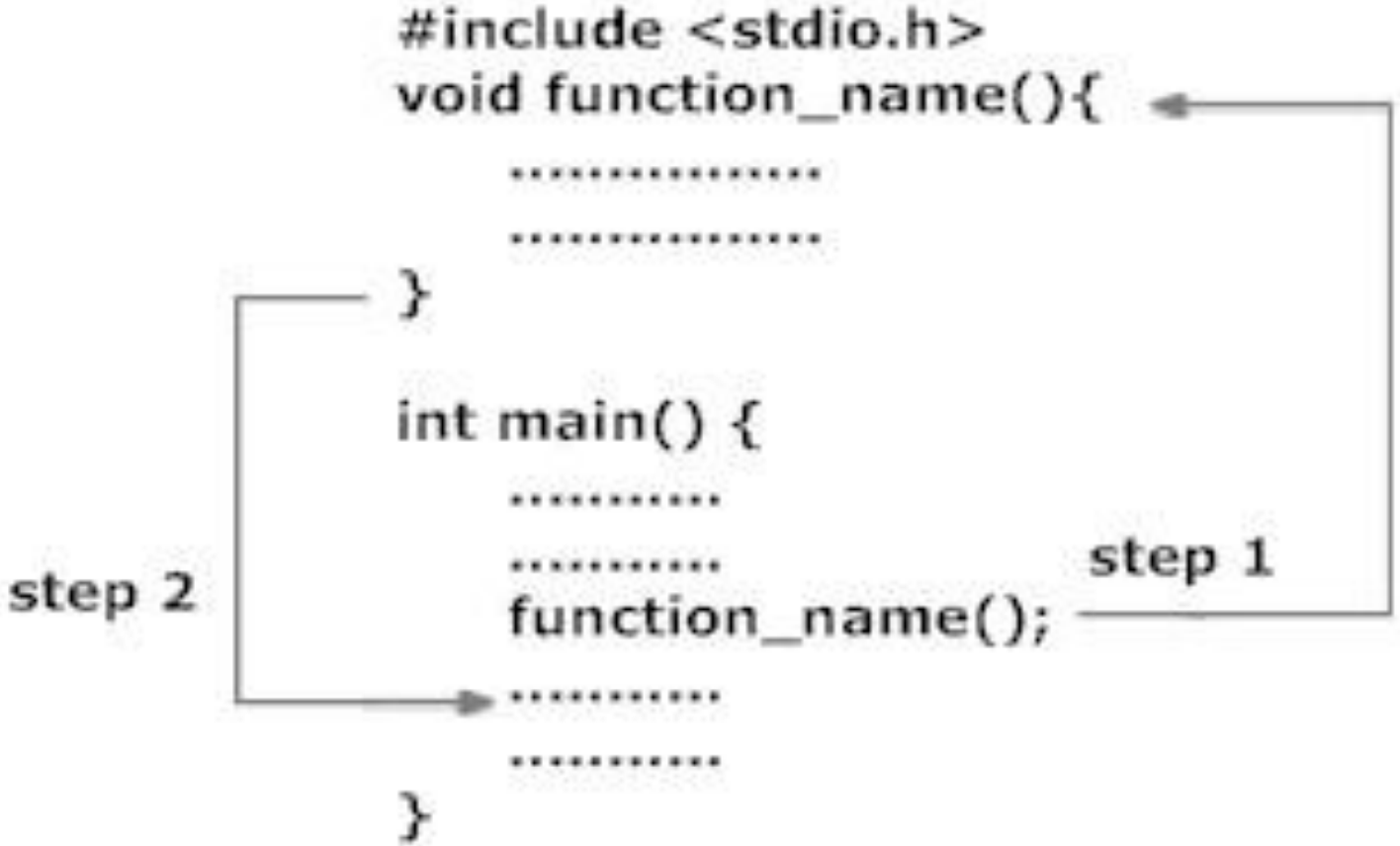
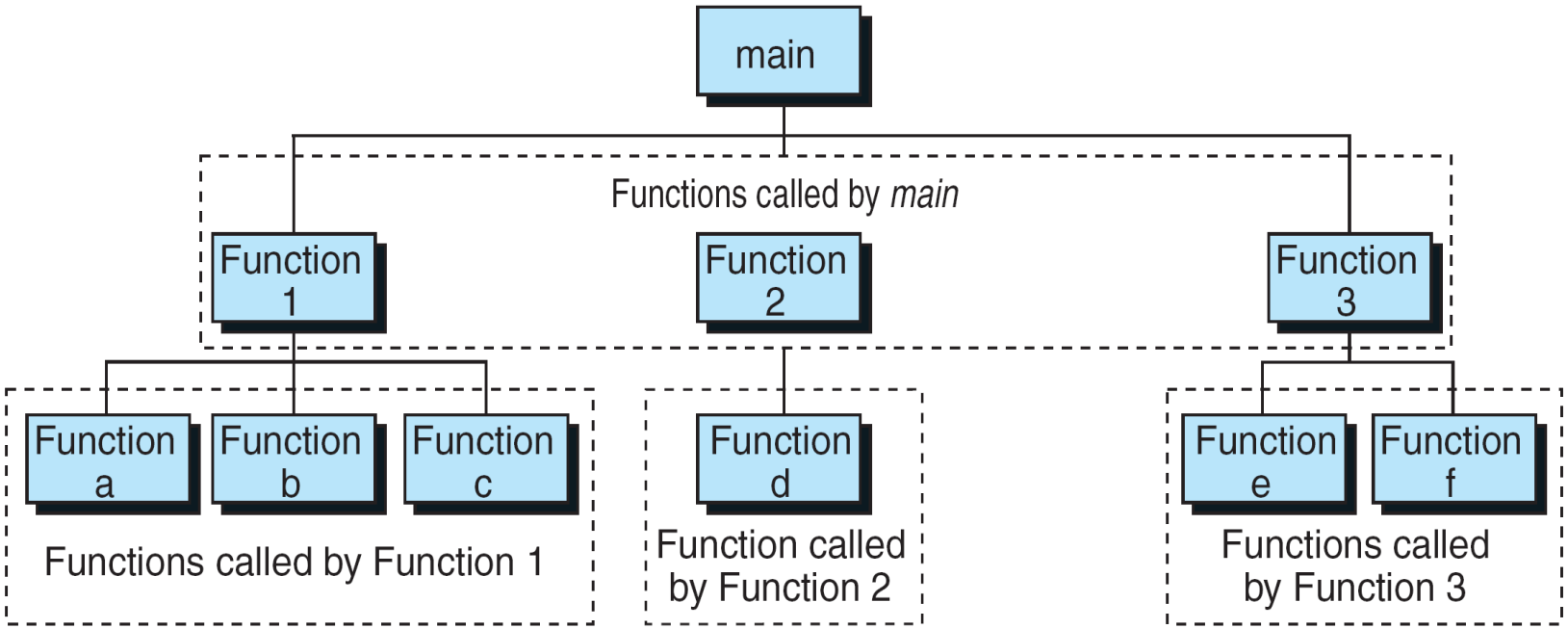


Fig: Working of Functions

Working of Function

- Every C program begins from main() and program starts executing the codes inside main() function.
- When the control of program reaches to function_name() inside main() function. The control of program jumps to void function_name() and executes the codes inside it.
- When, all the codes inside that user-defined function are executed, control of the program jumps to the statement just after function_name() from where it is called.

Structure Chart for a C Program



Call by Value & Call by Reference

Call by Value

- This method copies the value of an argument into the formal parameters.
- Changes made to the parameters of the subroutine have no effect on the variables used to call it.

Call by Value & Call by Reference

Call by Reference

- In this method, the address of an argument is copied into the formal parameters.
- Inside the subroutine, the address is used to access the actual argument used in the call.
- Changes made to the parameters of the subroutine affect the variables used to call it.

Example: bad_swap.c

```
/* Swap the values of two variables.
   */

void badSwap ( int a, int b )
{
  int temp;

  temp = a;
  a = b;
  b = temp;

  printf("%d %d\n", a, b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  badSwap ( a, b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
```

Example: bad_swap.c

```
/* Swap the values of two variables.
   */

void badSwap ( int a, int b )
{
  int temp;

  temp = a;
  a = b;
  b = temp;

  printf("%d %d\n", a, b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  badSwap ( a, b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
```

Example: bad_swap.c

```
/* Swap the values of two variables.
   */

void badSwap ( int a, int b )
{
  int temp;

  temp = a;
  a = b;
  b = temp;

  printf("%d %d\n", a, b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  badSwap ( a, b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
3  5
```

Example: bad_swap.c

```
/* Swap the values of two variables. */

void badSwap ( int a, int b )
{
  int temp;

  temp = a;
  a = b;
  b = temp;

  printf("%d %d\n", a, b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  badSwap ( a, b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
3  5
```

Example: swap.c

```
/* Swap the values of two variables. */

void Swap ( int *a, int *b )
{
  int temp;

  temp = *a;
  *a = *b;
  *b = temp;

  printf("%d %d\n", *a, *b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  Swap ( &a, &b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
```

Example: swap.c

```
/* Swap the values of two variables.
   */

void Swap ( int *a, int *b )
{
  int temp;

  temp = *a;
  *a = *b;
  *b = temp;

  printf("%d %d\n", *a, *b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  Swap ( &a, &b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
```

Example: swap.c

```
/* Swap the values of two variables.
   */

void Swap ( int *a, int *b )
{
  int temp;

  temp = *a;
  *a = *b;
  *b = temp;

  printf("%d %d\n", *a, *b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  Swap ( &a, &b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
5  3
```

Example: swap.c

```
/* Swap the values of two variables.
   */

void Swap ( int *a, int *b )
{
  int temp;

  temp = *a;
  *a = *b;
  *b = temp;

  printf("%d %d\n", *a, *b);
}
```

```
int main()
{
  int a = 3, b = 5;

  printf("%d %d\n", a, b);
  Swap ( &a, &b );
  printf("%d %d\n", a, b);

  return 0;
}
```

Output:

```
3  5
5  3
5  3
```

Calling Functions with Arrays

- In C programming, a single array element or an entire array can be passed to a function.
- Also, both one-dimensional and multi-dimensional array can be passed to function as argument.

Passing a single element of an array to function

```
#include <stdio.h>
void display(int a)
{ printf("%d",a);}
int main()
{int c[]={2,3,4};
display(c[2]);//Passing array element c[2] only.
return 0;}
```

Output

4

Passing Entire One-Dimensional Array to a Function

- While passing arrays to the argument, the name of the array is passed as an argument (i.e., starting address of memory area is passed as argument).

Passing Entire One-Dimensional Array to a Function

- Write a C program to pass an array containing age of 6 persons to a function. This function should find average age and display the average age in main function.

Passing Entire One-Dimensional Array to a Function

```
#include <stdio.h>
float average(float a[]);
int main()
{
float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
avg=average(c);/* Only name of array is passed as argument. */
printf("Average age=%.2f",avg);
return 0;}
float average(float a[])
{int i;float avg, sum=0.0;
for(i=0;i<6;++i)
{ sum+=a[i];}
avg =(sum/6);
return avg;}
```

Passing Multi-dimensional Arrays to Function

- To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array.

Passing Multi-dimensional Arrays to Function

```
#include <stdio.h>
void Function(int c[2][2]);
int main()
{int c[2][2],i,j;
 printf("Enter 4 numbers:\n");
  for(i=0;i<2;++i)for(j=0;j<2;++j)
{ scanf("%d",&c[i][j]);}
Function(c);/*passing multi-dimensional array to function */
return 0;}
void Function(int c[2][2])
{/* Instead to above line, void Function(int c[][2]){ is also valid */
int i,j;
printf("Displaying:\n");
for(i=0;i<2;++i)for(j=0;j<2;++j)
printf("%d\n",c[i][j]);}
```

Example:

```
/* shows how function can return
multiple results */
#include <stdio.h>
void test1(int m, int n);
void test2(int *m, int *n);
void test3(int a, int *b);
int main(void) {
    int a=10, b=16;
    printf("a=%d,  b=%d\n",a,b);
    test1(a,b);
    printf("a=%d, b=%d\n",a,b);
    test2(&a,&b);
    printf("a=%d, b=%d\n",a,b);
    test3(a,&b);
    printf("a=%d, b=%d\n",a,b);
    system("pause");
    return 0;
}
```

```
void test1(int m, int n) {
   m=5;
   n=24;
}

void test2(int *m, int *n) {
   *m=5;
   *n=24;
}

void test3(int a, int *b) {
   a=38;
   *b=57;
}
```

```
a=10,  b=16
a=10, b=16
a=5, b=24
a=5, b=57
Press any key to continue . . .
```

```
/* computes the area and circumference of a circle, given its radius */
#include <stdio.h>

void area_circum (double radius, double *area, double *circum);

int main (void) {
    double radius, area, circum ;

    printf ("Enter the radius of the circle > ") ;
    scanf ("%lf", &radius) ;

    area_circum (radius, &area, &circum) ;
    printf ("The area is %f and circumference is %f\n", area, circum) ;
    system("pause");
    return 0;
}

void area_circum (double radius, double *area, double *circum) {
    *area = 3.14 * radius * radius ;
    *circum = 2 * 3.14 * radius ;
}
```

```
/* Takes three integers and returns their sum, product and average */
#include<stdio.h>

void myfunction(int a,int b,int c,int *sum,int *prod, double *average);

int main (void) {
        int n1, n2, n3, sum, product;
        double av_g;
        printf("Enter three integer numbers > ");
        scanf("%d %d %d",&n1, &n2,&n3);
        myfunction(n1, n2, n3, &sum, &product, &av_g);
        printf("\nThe sum = %d\nThe product = %d\nthe avg =
%f\n",sum,product,av_g);
        system("pause");
        return 0;
}

void myfunction(int a,int b,int c,int *sum,int *prod, double *average) {
    *sum=a+b+c;
    *prod=a*b*c;
    *average=(a+b+c)/3.0;
}
```

```
/* takes the coefficients of quadratic
equation  a, b and c and returns its roots
*/
#include<stdio.h>
#include<math.h>

void quadratic(double a,double b, double
c, double *root1, double *root2);

int main(void) {
    double a,b,c,r1,r2;
    printf("Please enter coefficients of the
equation: [a b c] > ");
    scanf("%lf%lf%lf",&a,&b,&c);

    quadratic(a,b,c,&r1,&r2);

    printf("\nThe first root is : %f\n",r1);
    printf("The second root is : %f\n",
r2);
    system("pause");
    return 0;
}
```

```
void quadratic(double a,double b,
double c, double *root1, double *root2)
{
    double desc;

    desc =b*b-4*a*c;
    if(desc < 0) {
      printf("No real roots\n");
      system("pause");
      exit(0);
    }
    else {
       *root1=(-b+sqrt(desc))/(2*a);
       *root2=(-b-sqrt(desc))/(2*a);
    }
}
```