Chapter 2

Functions

LEARNING OBJECTIVES

- *Functions*
- *Library functions*
- *User defined functions*
- *Defining user defined functions*
- *Recursion*
- *Parameter passing*
- *Pass by value*
- *Pass by address*
- *Scope*
- *Life time*
- *Binding*

Functions

A function is a block of code that performs a specific task. It has a name and is reusable, i.e., it can be executed from as many different parts in a program as required.

Functions make possible top down modular programming. In this style of programming, the high-level logic of overall problem is solved first, whereas the detail of each lower-level function is addressed later. This approach reduces the complexity in writing program.

1. Every C program can be thought of collection of functions.
2. main( ) is also a function.

Types of Functions

Library functions

These are the in-built functions of '*C*' library. These are already defined in header files.

Example 1: printf( ); is a function which is used to print at output. It is defined in '`stdio.h`' file.

User-defined functions

Programmers can create their own function in '*C*' to perform specific tasks.

Example 2:
```
# include <stdio.h>
main( )
{
message( );
}
message( )
{
printf("Hello");
}
```

- A function receives zero (or) more parameters, performs a specific task, and returns zero or one value.
- A function is invoked by its name and parameters.
- No two functions have the same name in a single C program.
- The communication between the function and invoker is through the parameter and the return value.
- A function is independent.
- It is "completely" self-contained.
- It can be called at any place of your code and can be ported to another program.
- Functions make programs reusable and readable.

Example 3: Return the largest of two integers.
```
int maximum (int a, int b)
{
if (a > b)
return a;
else
return b;
}
```

Note: Function calls execute with the help of execution stack. Execution of 'C program' starts with main( ) function. Main( ) is a user-defined function.

Defining User-defined Functions

In order to work with user-defined functions, it requires the following concepts about functions:

- Declaration of a function
- Definition of a function
- Function call

Declaration specifies what

- is the name of the function
- are the parameters to pass (type, order and number of parameters).
- it returns on completion of execution

Example 4: int maximum (int, int); int maximum (int *a*, int *b*);

Syntax:

```
Return_type Function_Name(Parameter_list);
```

- Names of parameters are optional in declaration.
- Default return type for 'C' functions is 'int'.
- A function, whose return type is void returns nothing.
- Empty parenthesis after a function name in declaration says, function does not accept any parameters.

Definition specifies *how*

- to perform the specified task
- to accept passed parameters
- to process parameters or execute instruction to producer equired results (return value).

Function definition is a self-contained block of instructions, will be executed on call:

Syntax:

```
Return _type Function -Name(paralist)
{
Local declaration(s);
Executable statements(s);
}
int maximum (int a, int b)
{
if (a > b)
  return a;
else
  return b;
}
```

Function call specifies

1. where to execute the function
2. when to execute the function

Note: If the function definition provided before use (call), the declaration is optional.

The following example describes control flow during function call:

```
void hello(); // Declaration
void main()
{
printf("\n function");
hello();
printf("\n Main after call to hello")
void hello()//Definition
 {
printf ("\n function Hello");
return;
 }
```

A *return* statement has two important uses:

1. first, it causes an immediate exit from the function.
2. second, it may be used to return a value.

If a function does not return any value, then the return statement is optional.

Recursion

In general, programmers use two approaches to write repetitive algorithms. One approach using *loops*, the other is *recursion*.

Recursive functions typically implement recurrence relations, which are mathematical formula in which the desired expression (function) involving a positive integer, *n*, is described in terms of the function applied to corresponding values for integers less than '*n*'.

1. The function written in terms of itself is a recursive case.
2. The recursive case must call the function with a decreasing '*n*'.
3. Recursion in computer programming is exemplified when a function defined in terms of itself.
4. Recursion is a repetitive process in which a function calls itself.

Note: Recursive function must have an *if* condition to force the function to return without recursive call being executed. If there is no such condition, then the function execution falls into infinite loop.

> Rules for designing recursive function
> 1. Determine base case
> 2. Determine general case
> 3. Combine, base and general case into a function

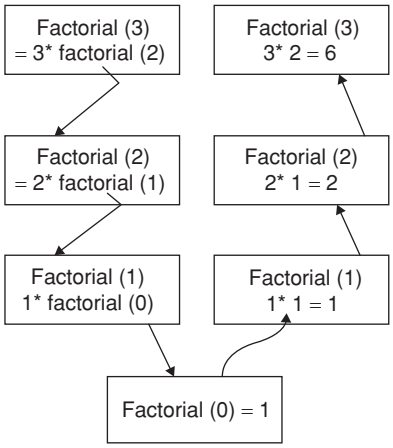
Example 5: Recursive factorial function

```
1. int factorial (int n)
2. {
3. if (n = = 0)
4. return 1;
5. else
6. return (n * factorial (n - 1));
7. }
```

The statement 3 is a base condition, which stops the recursive call of function.

The statement 6 reduces the size of problem by recursively calling the factorial with $(n - 1)$.

Execution sequences for factorial (3):



Disadvantages:

1. Recursive programs increase the execution time of program.
2. Recursive programs typically use a large amount of computer memory and the greater the recursion, the more memory is used.
3. Recursive programs can be confusing to develop and extremely complicated to debug.

Parameter Passing

There are two ways of passing parameters to functions in 'C' language.

1. Pass-by-value: When parameters are passed by value, create copies in called function. This mechanism is used when we do not want to change the value of actual parameters.
2. Pass-by-address: In this case, only the addresses of parameters are passed to the called function. Therefore, manipulation of formal parameters affects actual parameters.

Examples 6:

```
void swap1(int, int); /* function – to swap
two numbers by passing values */
void swap2 (int *, int *); /* function to
swap two numbers by passing Address * /
   void main ()
{
int a = 10, b = 15, c = 5, d = 25;
printf("value of a and b before swapping
:%d, %d" a , b );
swap1(a, b);
printf("values of a and b after swapping :
%d, %d", a, b);
printf ("values of c and d before swapping
:%d%d", c,d );
Swap2(&c, &d);
printf("values of c and d after swapping
%d, %d", c, d);
}
void swap1(int x, int y )
{
int temp;
temp = x;
x = y;
y = temp;
}
void swap2 (int *x, int *y)
{
int temp;
temp = *x;
*x = *y:
*y = temp;
}
```

Output:

Value of *a* and *b* before swapping: 10, 15
Value of *a* and *b* after swapping: 10, 15
Value of *c* and *d* before swapping: 5, 25
Value of *c* and *d* after swapping: 25, 5

Solved Examples

Example 1: Consider the program below:

```
#include<stdio.h>
int fun (int n, int *fp)
```

Table 1 *Comparison of pass-by-value and pass-by-address*

	Pass-by-value		Pass-by-address
1.	Also known as call-by-value	1.	Also known as call-by-address or call by-reference
2.	Pass the values of actual parameters	2.	Pass the address of actual parameters
3.	Formal parameters act as duplicates or as a copy to actual parameters	3.	Formal parameters acts as references to the actual parameters
4.	Operations on formal parameter does not affect actual parameters	4.	Operations on formal parameters affect actual parameters
5.	Passing of parameters is time consuming as the data size increases	5.	The size of parameters does not affect the time for transferring references.
6.	Actual parameters are secured	6.	Helps to return multiple parameters

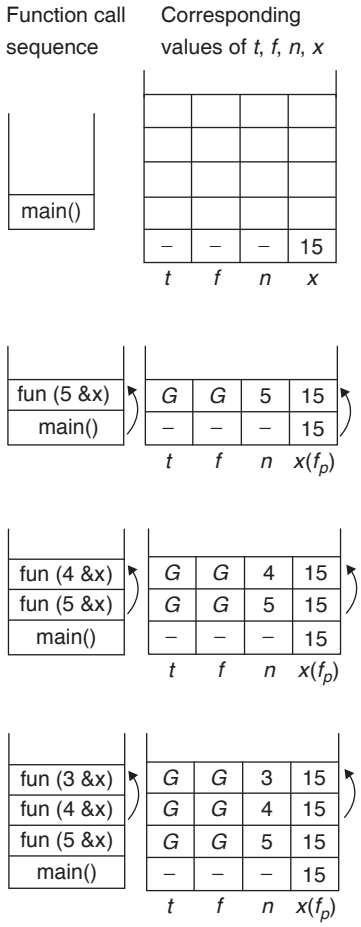
```
{
int t,f;
if (n < =1)
{
*fp=1;
return 1;
 }
t = fun(n-1, fp);
f = t+ *fp;
*fp = t;
return f;
}
int main ()
{
int x = 15;
printf ("%d\n", fun(5,&x));
return 0;
}
```

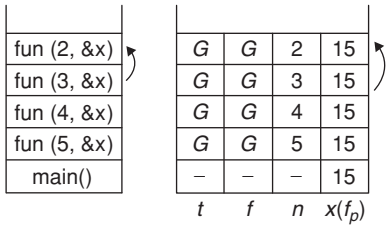
What is the output?

(A) 2 (B) 4
(C) 8 (D) 16

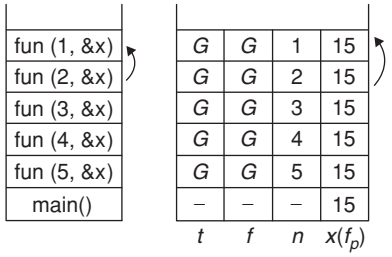
Solution: (C)

Execution stack

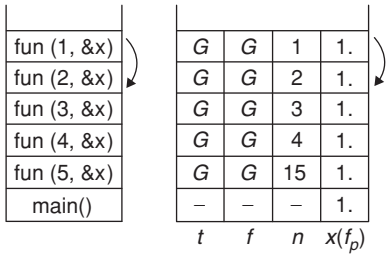


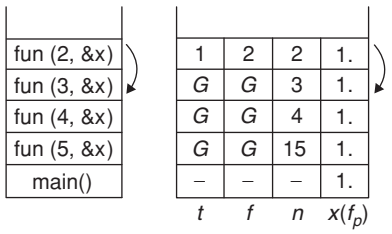


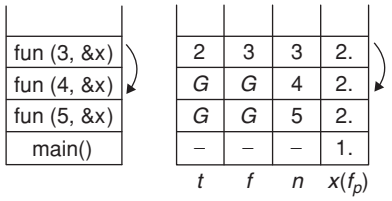
Note: '–' indicates *no memory* allocated to variable. '*G*' indicates *garbage value*.

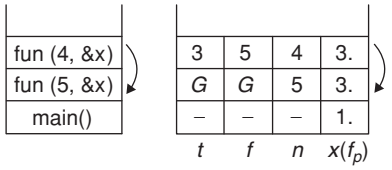


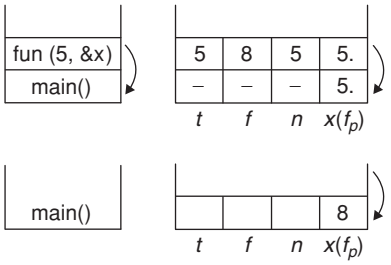
For the function call fun(1, &x) condition (*n*<=1) is true. So Assigns '1' to fp and returns '1'.











Finally, *x* contains '8', so printf prints '8'.

Example 2: What does the following program prints?

```
#include < stdio.h>
void f (int *p, int *q)
{
p=q;
*p=12;
}
int i = 0, j=1;
int main()
{
f(&i, &j);
printf(" %d%d ", i, j);
return 0 ;
}
```

(A) 2 12 (B) 12 1
(C) 0 1 (D) 0 12

Solution: (D)

```
main( )
f (&i, &j)
address of 'i' is stored in to p.
and address of 'j' is stored into 'q'.
i.e., *p and*q refers i and j.
The statement:
p = q; updates pointer 'p', so that both
pointers refer to parameter 'j'.
*p = 12
Changes value of 'j' to '12' But 'i' does
not effected. So, prints 0 12.
```

Example 3: What is the value printed by the following program?

```
# include <stdio.h>
int f(int *a, int n)
{
if (n<=0) return 0 ;
else if(*a%2 = = 0)
return *a + f(a+1, n-1);
else
return *a - f(a+1, n-1);
}
int main ( )
{
int G [ ] = { 12, 7, 13, 4, 11, 6};
printf("%d", f (a,b));
return 0;
}
```

(a) –9 (b) 12
(c) 15 (d) 20

Solution: (C)

	0	1	2	3	4	5
a	12	7	13	4	11	6

$f(a, 6)$ is the first call to function $f( )$.

The array _ name refers to base address of array, i.e., address of first element.

Thus,

$F(a, 6)$

12 % 2 = 0. So,

$$12 + f\underset{\downarrow 7}{(\overline{a+1}),\left(\frac{n-1}{5}\right)}\left[*a \text{ is even}\right]$$

$$12+\left(7-\left(\underset{\downarrow 13}{f(a+1),\left(\frac{n-1}{4}\right)}\right)\right)[*a \text{is odd}]$$

$$12+\left(7-\left(13-f\left(\underset{\downarrow 4}{(a+1)},\left(\frac{n-1}{3}\right)\right)\right)\right)[*a \text{ is odd}]$$

$$12+\left(7-\left(13-\left(4+f\left(\underset{\downarrow 11}{(a+1),\frac{(n-1)}{2}},\right)\right)\right)\right)[*a \text{ is even}]$$

$$12+\left(7-\left(13-\left(4+\left(11-f\left(\underset{\downarrow 6}{(a+1),\left(\frac{n-1}{1}\right)}\right)\right)\right)\right)\right)[*a \text{ is odd}]$$

$$12+\left(7-\left(13-\left(4+\left(11-\left(6+\underset{\downarrow 0}{f(a+1),\frac{(n-1)}{0}}\right)\right)\right)\right)\right)[*a \text{ is even}]$$

$$12 + (7 \quad (13 - (4 + (11 - (6 + 0))))) = 15$$

Scope, Lifetime and Binding

Storage classes specify the scope, lifetime and binding of variables. To fully define a variable, one needs to mention not only its 'type' but also its 'storage class'.

A variable name identifies some physical location within computer memory where a collection of bits are allocated for storing value of variable.

Storage class tells us:

1. Where the variable would be stored (either in memory or CPU registers)?
2. What will be the initial value of a variable, if no value is specifically initialized?
3. What is the scope of a variable (where it can be accessed)?
4. What is the life of a variable?

Scope

The scope defines the visibility of an object. It defines where an object can be referenced/accessed; generally, the scope of variable is local or global.

1. The variables defined within a block have *local scope*. They are *visible only to the block* in which they are defined.
2. The variables defined in global area are visible from their definition until the end of program. It is *visible everywhere* in program.

Lifetime

The lifetime of a variable defines the duration for which the computer allocates memory for it (the duration between allocation and deallocation of memory).

In *C*, variable can have automatic, static or dynamic lifetime.

1. Automatic: Variables with automatic lifetime are created each time their declaration are encountered and are destroyed each time their blocks are exited.
2. Static: A variable is created when the declaration is executed for the first time and destroyed when the execution stops/terminates.
3. Dynamic: The variable's memory is allocated and deallocated through memory management functions.

Binding

Binding finds the corresponding binding occurrence (declaration/definition) for an applied occurrence (usage) of an identifier. For Binding.

1. Scope of variables should be known. What is the block structure? In which block the identifier is variable?
2. What will happen if we use same identifier name again? '*C* forbids use of same identifier name in the same scope'. Same name can be used in different scopes.

Examples:

1.
```
double f,y;
int f(  ) // error
{
.
.
.
}
double y; // error
```

2.
```
double y;
int f( )
{
double f;// legal
int y; //legal
}
```

There are four storage classes in *C*.

Storage class	Storage Area	Default Initial Value	Lifetime	Scope	Keyword
Automatic	Memory	Till the control remains in block	Till the control remains in block	Local	auto
Register	CPU register	An unpredictable value (or) garbage value	Till the control remains in block	Local	register
Static	Memory	Zero	Value of variable persist between function calls	Local	static
External	Memory	Unpredictable or garbage value	Throughout program execution	Global	extern

Note: Default storage class is auto.

Example 4: What will be the output for the program?

```
int i = 33;
main( )
{
   extern int i;
     {
         int i = 22;
          {
               const volatile unsigned i
               = 11;
                    printf (" %d ", i);
              }
                    printf (" %d ", i);
              }
printf ("%d ", i) ;
}
```

(A) error
(B) 11 22 33
(C) 11 22 garbage
(D) 11 11 11

Solution: (B)

'{' introduces new block and thus new scope. In the innermost block, *i* is declared as const volatile unsigned which is a valid declaration. *i* is assumed of type int. So printf prints 11. In the next block, *i* has value 22 and so printf prints 22. In the outermost block, *i* is declared as extern, so no storage space is allocated for it. After compilation is over, the linker resolves it to global variable, *i* since it is the only variable visible there. So it prints its value as 33.

Example 5: Consider the following C program:

```
int f(int n)
{
static int r;
if (n<=0) return 1;
if (n> 3)
{
r=n;
return (f(n-2)+2));
}
return f(n-1) + r;
}
```

What is the value of *f*(5)?

(a) 15 (b) 17
(c) 18 (d) 19

Solution: (C)

Call Sequence	*r*	Return Sequence
f(5) ↓	5	18
f(3)+2	5	16+2 ↓
f(2)+*r*	5	11+5 ↓
f(1)+*r*	5	6+5 ↓
f(0)+*r*	5	1+5 ↓

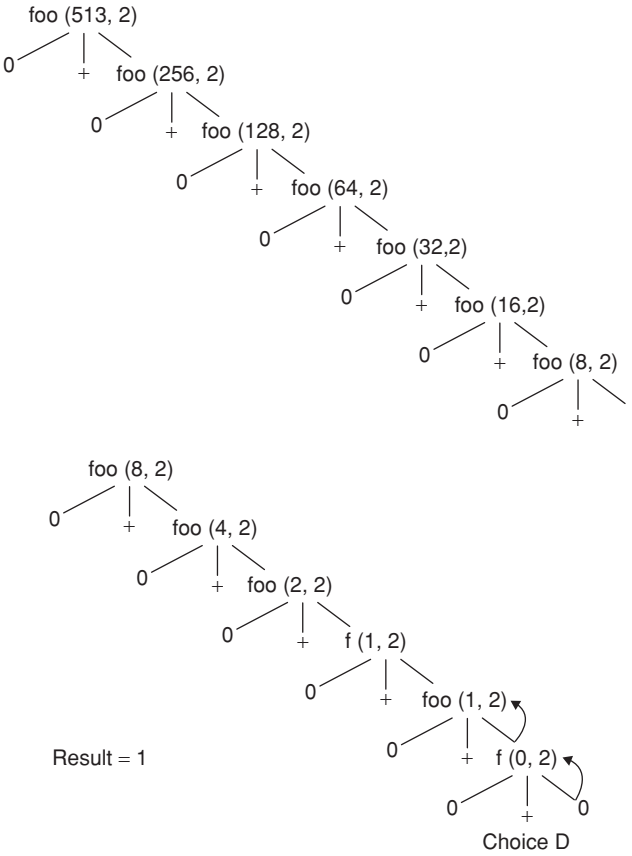
Common data for questions 6 and 7: Consider the following recursive 'C' function that takes two arguments.
unsigned int foo (unsigned int *n*, unsigned int *r*)

```
{
if (n>0)
return ((n%r)+ foo(n/r,r));
else
return 0;
}
```

Example 6: What is the return value of the function foo when it is called as foo (512,2)?

(A) 9 (B) 8
(C) 2 (D) 1

Solution: (D)

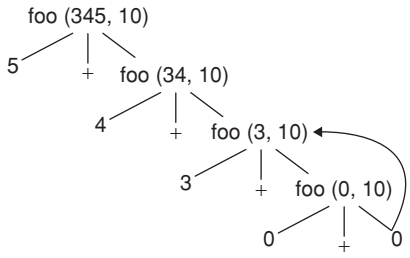


Result = 1

Example 7: What is return value for the function call foo (345, 10)?

(A) 345 (B) 12
(C) 5 (D) 3

Solution: (B)



result 5 + 4 + 3 = 12

EXERCISES

Practice Problems I

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. What will be the output of the following program?

```
main( )
{
main( );
}
```

(A) overflow error (B) syntax error
(C) returns 0 (D) returns 1

2. Output of the following program is

```
 main( )
  {
static int var = 6;
printf("%d\t", var--);
if(var)
                    main( );
  }
```

(A) 5 4 3 2 1 0 (B) 6 6 6 6 6 6
(C) 6 5 4 3 2 1 (D) Error

3. Which of the following will be the output of the program?

```
main( )
{
 char str[ ] = "Hello";
 display( str );
}
void display (char *str)
    {
        printf ( "%s", str) ;
    }
```

(A) compilation error (B) hello
(C) print null string (D) no output

4. Consider the following C function

```
int fun (int n)
{
  static int x = 0;
  if (n<=0) return 1;
  if (n>3)
     {
          x = n;
       return fun(n-2)+3;
     }
 return fun(n-1)+ x;
 }
```

What is the value of fun(5)?

(A) 4 (B) 15
(C) 18 (D) 19

5. For the following C function

```
void swap (int a, int b)
{
   int t;
          t = a;
          a = b;
     b = t;
}
```

In order to exchange the values of two variables *w* and *z*,

(A) call swap (*w*, *z*)
(B) call swap (and *w*, and *z*)
(C) swap (*w*, *z*) cannot be used as it does not return any value
(D) swap (*w*, *z*) cannot be used as the parameters are passed by value

6. Choose the correct option to fill? *x* and? *y* so that the program below prints an input string in reverse order. Assume that the input string is terminated by a new line character:

```
void Rev(void) {
  int a;
  if (?x) Rev( );
  ?y
      }
main( ) {
printf("Enter the text");
printf(" \n");
Rev( );
printf("\n");
}
```

(A) `? x is (getchar( )! = '\n')`
`? y is getchar (A);`
(B) `? x is((A = getchar( )) ! = '\n')`
`? y is getchar(A) ;`
(C) `? x is (A! = '\n')`
`? y is putchar (A);`
(D) `? x is (A = getchar ( )) ! = '\n')`
`? y is putchar(A) ;`

7.

```
main ( )
  {
     extern int a;
      a = 30;
       printf ("%d", a);
  }
```

What will be the output of the above program?

(A) 30 (B) Compiler error
(C) Runtime error (D) Linker error

8. Which of the following will be the output of the program?

```
void main ( )
 {
    int n = ret(sizeof(float));
    printf("\n value is %d ", ++n);
}
int ret(int ret)
{
```

```
    ret += 2.5;
   return (ret);
}
```

(A) Value is 6 (B) Value is 6.5
(C) Value is 7 (D) Value is 7.5

9. The following program

```
main( )
{
    pt( ); pt( );pt( );
}
pt( )
{
    static int a;
    printf("%d", ++a) ;
}
```

prints
(A) 0 1 2
(B) 1 2 3
(C) 3 consecutive, but unpredictable numbers
(D) 1 1 1

10. What is the output of the following program?

```
main( ) {
    int i = 0;
    while (i < 4) {
                        sum(i);
                        i++;
                    }
          }
void sum(int i) {
    static int k;
printf ("%d", k + i);
k++;
                    }
```

(A) 0 2 4 6 (B) 0 1 2 3
(C) 0 2 0 0 (D) 1 3 5 7

11. What will be the output of following code?

```
# include <stdio.h>
aaa() {
printf("hi");
}
bbb() {
printf("hello");
}
ccc()
{
printf("bye");
}
main ()
{
int *ptr[3]( );
ptr[0] = aaa;
ptr[1] = bbb;
ptr[2] = ccc;
ptr[2]();
}
```

(A) hi (B) hello
(C) bye (D) Garbage value

12. What is the output?

```
void main()
{
static int i = 5;
if(--i)
{
main();
printf("%d", i);
}
}
```

(A) 5 (B) 5 5 5 5
(C) 0 0 0 0 (D) 1 1 1 1

13. If the following function gets compiled, what error would be raised?

```
double fun(int x, double y)
{
int x;
x = 100;
return y;
}
```

(A) Function should be defined as int fun(int *x*, double *y*)
(B) Missing parenthesis in return
(C) Redeclaration of *x*
(D) All of these

14. Consider the following function:

```
fun(int x)
{
if ((x/2)! = 0)
return (fun (x/2)  10 + x%2);
else return 1;
}
```

What will happen if the function 'fun' called with value 16 i.e., as fun(16).
(A) Infinite loop
(B) Random value will be returned
(C) 11111
(D) 10000

15. What is the output of the following program?

```
void main( )
{
static int x = 5;
printf("%d", x - - );
if (x ! = 0)
main( );
}
```

(A) error:main( ) cannot be called from main( )
(B) Infinite loop
(C) 5 4 3 2 1
(D) 0

Practice Problems 2

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. An external variable
(A) is globally accessible by all functions
(B) has a declaration "extern" associated with it when declared within a function
(C) will be initialized to 0, if not initialized
(D) all of the above

2. The order in which actual arguments are evaluated in a function call
(A) is from the left (B) is from the right
(C) is unpredictable (D) none of the above

3. In *C* language, it is necessary to declare the type of a function in the calling program if the function
(A) returns an integer (B) Returns a float
(C) both (A) and (B) (D) none of the above

4. What is the output?

```
void main()
{
int k = ret(sizeof(int));
printf("%d", ++k);
}
int ret (int ret)
{
ret + = 2.5;
return (ret);
}
```

(A) 3.5 (B) 5
(C) 4 (D) logical error

5. When a recursive function is called, all its automatic variables are
(A) maintained in stack
(B) retained from last execution
(C) initialized during each call of function
(D) none of these

6. Consider the following program segment:

```
int fun(int x, int y)
{
if(x > 0)
return ((x % y) + fun(x/y, y));
else
return 0;
}
```

What will be the output of the program segment if the function is called as fun(525, 25)?
(A) 25 (B) 12
(C) 21 (D) 42

7. Consider the following C program segment:

```
int fun (int x)
{
static int i = 0;
if (x < = 0)
return 1;
else if (x > 5)
{
i = x;
return fun (x - 3) +2;
}
return fun (x - 2) + i;
}
```

What is the value of fun(7)?
(A) 17 (B) 10
(C) 11 (D) 9

8. Consider the following C program:

```
void rearrange( )
{
char ch;
if (X)
rearrange( );
Y;
}
void main ( )
{
printf("\n enter text to print reverse
order :");
rearrange( ) ;
}
```

Choose the correct option to fill *X* and *Y*, so that the program prints the entered text in reverse order. Assume that input string terminates with new line.

(A)
```
X:   (getchar(ch) = = '\n')
Y:   putchar(ch);
```
(B)
```
X:   (getchar(ch)! = '\n')
Y:   ch = putchar( );
```
(C)
```
X:   ((ch = getchar( ) )! = '\n')
Y:   putchar(ch);
```
(D)
```
X:   ((ch = getchar( )) = = '\n')
Y:   putchar (ch);
```

9. Consider the following C function:

```
int f(int n)
{
static int i = 1;
if (n > = 5) return n;
n = n + i;
i++ ;
return f(n);
}
```

The value returned by *f*(1) is
(A) 5 (B) 6
(C) 7 (D) 8

10. Consider the following C function:

```
int incr (int i)
{
static int count = 0;
count = count + i;
return (count);
```

```
}
main ( )
{
int i, j;
for (i = 0; i < =4; i++)
j = incr (i);
}
```

The *j* value will be
(A) 10
(B) 4
(C) 6
(D) 7

11. The following function

```
int Trial (int a, int b, int c)
{
if ((a > = b) && (c < b))
return b;
else if(a > = b)
return Trail(a, c, b);
else return Trail (b, a, c);
}
```

(A) finds the maximum of *a*, *b*, *c*
(B) finds the middle value of *a*, *b*, *c* after sorting
(C) finds the minimum of *a*, *b*, *c*
(D) none of the above

12. Consider the following pseudo code

```
f(a, b)
{
while(b! = 0)
{
t = b;
b = a % b;
a = t;
}
return a;
}
```

(A) The above code computes HCF of two numbers *a* and *b*
(B) The above code computes LCM of *a* and *b*
(C) The above code computes GCD of *a* and *b*
(D) None of the above

13.
```
1. main ( )
2. {int a = 10, *j;
3. void *k;
4. j = k = &a;
5. j++;
6. k++;
7. printf("\n %u, %u", j, k);
8. }
```

Which of the following is true in reference to the above code?
(A) The above code will compile successfully
(B) Error on line number 6
(C) Error on line number 3
(D) Error on line number 4

14. Aliasing in the context of programming language refers to
(A) multiple variables having the same memory location
(B) multiple variables having the same value
(C) multiple variables having the same identifier
(D) multiple uses of the same variable

15. Match the following:

X:	*m* = malloc (5); *m* = NULL;	1:	Using dangling pointers
Y:	free (*n*); *n* ↓ value = 5;	2:	Using un initialized pointers
Z:	char **p*; **p* = '*a*';	3:	Lost memory

(A) X – 1 Y – 3 Z – 2
(B) X – 3 Y – 1 Z – 2
(C) X – 3 Y – 2 Z – 1
(D) X – 2 Y – 1 Z – 3

Previous Years' Questions

1. In the following C function, let $n \geq m$.

```
int gcd(n,m)
{
if (n%m ==0) return m;
   n = n%m;
return gcd(m,n);
}
```

How many recursive calls are made by this function? **[2007]**

(A) $\Theta(\log_2 n)$ (B) $\Omega(n)$

(C) $\Theta(\log_2 \log_2 n)$ (D) $\Theta(\sqrt{n})$

2. What is the time *complexity* of the following recursive function?

```
int DoSomething (int n) {
if (n <= 2)
     return 1;
 else
return(DoSomething(floor(sqrt(n)))+ n);}
```

[2007]

(A) $\Theta(n^2)$ (B) $\Theta(n \log_2 n)$

(C) $\Theta(\log_2 n)$ (D) $\Theta(\log_2 \log_2 n)$

3. Choose the correct option to fill ? 1 and ? 2 so that the program below prints an input string in reverse order. Assume that the input string is terminated by a newline character.

```
void reverse (void) {
int c;
if (?1) reverse( );
?2
}
main ( ) {
printf ("Enter Text") ; printf ("\ n");
reverse ( ); printf ("\ n") ;
}
```

[2008]

(A)
```
?1 is (getchar( )! = '\n')
   ?2 is getchar(c);
```

(B)
```
?1 is (c = getchar( ))! = '\n')
   ?2 is getchar(c);
```

(C)
```
?1 is (c! = '\n')
   ?2 is putchar(c);
```

(D)
```
?1 is ((c = getchar( ))! = '\n')
   ?2 is putchar(c);
```

4. Consider the program below:

```
# include < stdio.h >
   int fun(int n, int * f_p) {
      int t, f;
      if (n <=1) {
         *f_p =1;
      return 1;
      }
      t = fun (n-1, f_p);
      f = t+*f_p;
      *f_p = t;
      return f;
      }
      int main( ) {
      int x = 15;
      printf ("%d\n", fun(5,&x));
      return 0;
      }
```

The value printed is **[2009]**

(A) 6 (B) 8

(C) 14 (D) 15

5. What is the value printed by the following C program?

```
#include <stdio.h>
int f(int *a, int n)
{
if (n <= 0)return 0;
else if(*a % 2 = = 0) return * a + f(a+1,
n-1);
else return *a-f(a+1, n-1);
}
int main( )
{
int a[ ] = {12, 7, 13, 4, 11, 6};
printf("%d", f(a,6));
return 0;
}
```

[2010]

(A) –9 (B) 5

(C) 15 (D) 19

Common data for questions 6 and 7: Consider the following recursive C function that takes two arguments.

```
unsigned int foo (unsigned int n, unsigned int r)
{
if ( n > 0 ) return ((n % r) + foo (n /r,
r));
else return 0;
}
```

6. What is the return value of the function foo when it is called as foo (513, 2)? **[2011]**

(A) 9 (B) 8

(C) 5 (D) 2

7. What is the return value of the function foo when it is called as foo (345, 10)? **[2011]**

(A) 345 (B) 12
(C) 5 (D) 3

Common data for questions 8 and 9: Consider the following C code segment

```
int a, b, c = 0;
void prtFun (void);
main ( )
{ static int a = 1;
prtFun( );
a+ = 1;
prtFun( );
printf("\n %d %d", a, b);
}
void prtFun(void)
{static int a = 2;
int b = 1;
a+ = ++b;
printf("\n %d %d", a, b);
}
```

8. What output will be generated by the given code segment if: **[2012]**
Line 1 is replaced by **auto int** a **= 1;**
Line 2 is replaced by **register int** a **= 2;**

(A)	(B)	(C)	(D)
3 1	4 2	4 2	4 2
4 1	6 1	6 2	4 2
4 2	6 1	2 0	2 0

9. What output will be generated by the given code segment? **[2012]**

(A)	(B)	(C)	(D)
3 1	4 2	4 2	3 1
4 1	6 1	6 2	5 2
4 2	6 1	2 0	5 2

10. What is the return value of $f(p, p)$, if the value of p is initialized to 5 before the call? Note that the first parameter is passed by reference, whereas the second parameter is passed by value.

```
int f(int &x, int c) {
   c = c - 1;
   if (c == 0) return 1;
   x = x + 1;
   return f(x, c) * x;
}
```
[2013]

(A) 3024 (B) 6561
(C) 55440 (D) 161051

11. Consider the following pseudo code. What is the total number of multiplications to be performed? **[2014]**

```
D = 2
for i = 1 to n do
for j = i to n do
for k = j +1 to n do
D = D * 3
```

(A) Half of the product of the three consecutive integers.
(B) One-third of the product of the three consecutive integers.
(C) One-sixth of the product of the three consecutive integers.
(D) None of the above.

12. Consider the function func shown below:

```
int func (int num) {
int count = 0;
while (num) {
count ++;
num>>=1;
}
return (count);
}
```

The value returned by func(435) is _____ **[2014]**

13. Consider the following function

```
double f (double X)
if (abs(X*X - 3) < 0.01)return X;
else return f(X/2 + 1.5/X);
}
```

Give a value q(to two decimals) such that $f(q)$ will return q:_____ **[2014]**

14. Consider the following pseudo code, where x and y are positive integers **[2015]**

```
begin
        q := 0
        r := x
        while r ≥ y do
              begin
                    r := r - y
                    q := q + 1
              end
        end
```

The post condition that needs to be satisfied after the program terminates is

(A) $\{r = qx + y \wedge r < y\}$
(B) $\{x = qy + r \wedge r < y\}$
(C) $\{y = qx + r \wedge 0 < r < y\}$
(D) $\{q + 1 < r - y \wedge y > 0\}$

15. Consider the following C function **[2015]**

```
int fun(int n) {
    int x = 1, k;
    if (n == 1) return x;
    for (k = 1; k < n; ++k)
        x = x + fun(k) * fun(n - k);
    return x;
}
```

The return value of fun(5) is _____

16. Consider the following recursive C function

```
void get (int n)
        {
            if (n < 1) return;
            get (n - 1);
            get (n - 3);
            printf("%d", n);
        }
```

If get (6) function is being called in main ( ) then how many times will the get ( ) function be invoked before returning to the main ( )?

(A) 15 (B) 25
(C) 35 (D) 45

17. Consider the following C program **[2015]**

```
#include<stdio.h>
int f1(void);
int f2(void);
int f3(void);
int x = 10;
int main ( )
 {
 int x = 1;
 x += f1( ) + f2 ( ) + f3 ( ) + f2 ( );
    printf("%d", x);
    return 0;
 }
int f1 () { int x = 25; x++; return x;}
int f2 () { static int x = 50; x++; return x;}
int f3 () { x *= 10; return x};
```

The output of the program is _____

18. Suppose $c = < c[0],\ldots,c[k-1]>$ is an array of length k, where all the entries are from the set $\{0, 1\}$. For any positive integers a and n, consider the following pseudo code. **[2015]**

DOSOMETHING (c, a, n)
$z \leftarrow 1$
for $i \leftarrow 0$ to $k - 1$
do $z \leftarrow z^2 \bmod n$
if $c[i] = 1$
then $z \leftarrow (z \times a) \bmod n$
return z

If $k = 4$, $c = <1, 0, 1, 1>$, $a = 2$ and $n = 8$, then the output of DOSOMETHING(c, a, n) is _____

19. What will be the output of the following C program? **[2016]**

```
void count (int n) {
static int d = 1;
printf("%d ",n);
printf("%d ",d);
d ++;
if (n > 1) count (n -1);
printf("%d ", d);
}
void main ( ) {
count (3);
}
```

(A) 3 1 2 2 1 3 4 4 4
(B) 3 1 2 1 1 1 2 2 2
(C) 3 1 2 2 1 3 4
(D) 3 1 2 1 1 1 2

20. The following function computes X^Y for positive integers X and Y. **[2016]**

```
int exp (int X, int Y)
{
int res = 1, a = X, b = Y;
while (b! = 0)
{
if (b%2 = = 0) {a = a*a; b = b/2;}
else {res = res *a; b = b -1;}
}
return res;
}
```

Which one of the following conditions is **TRUE** before every iteration of the loop?

(A) $X^Y = a^b$
(B) $(\text{res} * a)^Y = (\text{res} * X)^b$
(C) $X^Y = \text{res} * a^b$
(D) $X^Y = (\text{res} * a)^b$

21. Consider the following two functions.

```
void fun1 (int n) {
    if (n == 0) return;
    printf ("%d", n);
    fun2 (n - 2) ;
    printf ("%d", n);
}
```

```
void fun2 (int n) {
if (n == 0) return;
printf ("%d", n);
fun1(++n)
printf ("%d", n);
}
```

The output printed when fun1 (5) is called is **[2017]**

(A) 53423122233445 (B) 53423120112233
(C) 53423122132435 (D) 53423120213243

22. Consider the C functions foo and bar given below:

```
int foo (int val) {
   int x = 0;
   while (val > 0) {
      x =x + foo (val--);
   }
   return val;
}
int bar (int val) {
   int x = 0;
   while (val > 0) {
      x =x +bar (val - 1);
      }
   return val;
}
```

Invocations of foo (3) and bar (3) will result in: **[2017]**

(A) Return of 6 and 6 respectively.
(B) Infinite loop and abnormal termination respectively.
(C) Abnormal termination and infinite loop respectively.
(D) Both terminating abnormally.

23. The output of executing the following C program is_____.

```
# include <stdio.h>
int total (int v)  {
   static int count = 0;
   while (v)  {
      count + = v&1;
      v >> = 1;
   }
    return count;
}
void main ( )  {
   static int x = 0;
   int i = 5;
   for (; i > 0,i--)  {
      x = x + total (i);
   }
   printf ("%d\n", x);
}
```

[2017]

24. Consider the following C program:

```
#include <stdio.h>
int counter = 0;
int calc (int a, int b) {
   int c;
   counter++;
   if (b==3) return (a*a*a);
   else {
      c = calc (a, b/3);
      return (c*c*c);
   }
}
int main () {
   calc (4, 81);
   printf ("%d", counter);
}
```

The output of this program is _____. **[2018]**

25. Consider the following program written in pseudo-code. Assume that *x* and *y* are integers.

```
Count (x,y) {
if (y ! = 1) {
   if (x ! = 1) {
      print ("*");
      Count (x/2, y);
   }
   else {
      y = y-1;
      Count (1024, y);
      }
   }
}
```

The number of times that the print statement is executed by the call count (1024, 1024) is _____. **[2018]**

Answer Keys

Exercises

Practice Problems 1

1. A	**2.** C	**3.** A	**4.** D	**5.** D	**6.** D	**7.** D	**8.** C	**9.** B	**10.** A
11. C	**12.** C	**13.** C	**14.** D	**15.** C					

Practice Problems 2

1. D	**2.** C	**3.** B	**4.** B	**5.** C	**6.** C	**7.** A	**8.** C	**9.** C	**10.** A
11. B	**12.** C	**13.** B	**14.** A	**15.** B					

Previous Years' Questions

1. C	**2.** -	**3.** –D	**4.** B	**5.** C	**6.** D	**7.** B	**8.** D	**9.** C	**10.** B
11. C	**12.** 9	**13.** 1.72 to 1.74		**14.** B	**15.** 51	**16.** B	**17.** 230	**18.** 0	**19.** A
20. C	**21.** A	**22.** C	**23.** 23	**24.** 4	**25.** 10230				