SHAMBHUNATH INSTITUTE OF ENGINEERING & TECHNOLOGY

Subject: Programming for Problem Solving **Subject Code: KCS-101**

B.Tech.: 1st Year **SEMESTER: 1st**

SOLUTION- 2nd Sessional Examination (2019-2020)

Branch: Computer Science & Engineering

Time – 1 hr. 30 min. **Max Marks – 30**

SECTION – A

Question 1(a): Array: Array is a finite collection of homogeneous elements in contiguous block of memory. To declare an array in C, following is the syntax −

data_type array_name [size];

where, data_type is type of elements, array_name is a user defined name given to array and size specifies the number of elements that can be stored in the array.

Question 1(b): Function: A function is a group of statements that is written to perform a certain task. Every C program has at least one function, i.e. main(). Apart from this, a C-program may have many more functions as well.

Function declaration/prototyping: A function declaration tells the compiler about:

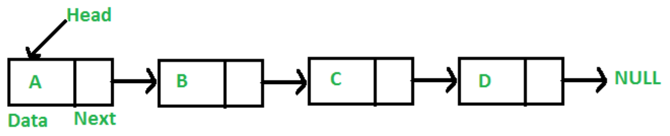
- ✓ **Function's name:** It's a user defined name (identifier) given to a function.
- ✓ **Return type:** It specifies the data type of the value returned by the function. If function does not return any value, then its return type would be given as 'void'
- ✓ **Parameters/Arguments:** It specifies the number of arguments along with their data type. Each parameter in the list is separated by a comma.

Syntax: *return_type function_name(parameter_list);*

Question 1(c): Two difference between linear and binary search:

Linear search technique	**Binary search technique**
Target value is compared one by one from beginning to end in the list to be searched.	In each step, target value is compared with the middle element in list to be searched.
Given list need not be sorted	Given list must be sorted

Question 1(d): Linked list: A linked list is a linear data structure where the elements in a linked list are linked using pointers as shown in the below image:



Structure of Node: Each node contains a data field and a reference(link) to the next node in the list. Structre of a node can be defined as follows:

```
struct Node
{
        int Data;
        struct Node *Next;
};
```

Question 1(e): Output of given program: 1

SECTION – B

Question 2(a): **Recursion:** In programming languages, if a function calls itself inside its own body, then it is called a recursive call of the function and this process is called recursion.

```
void myfunc()
{
        myfunc();     // Function myfunc() calling itself, therefore, it is a recursive call
}
```

Princliples of recursive programs:

- A recursive algorithm must have a base case.
  (**Base case:** is the condition where recursion finds concrete solution corresponding to the input and then terminates)
- A recursive algorithm must change its state and move toward the base case.
- A recursive algorithm must call itself, recursively.

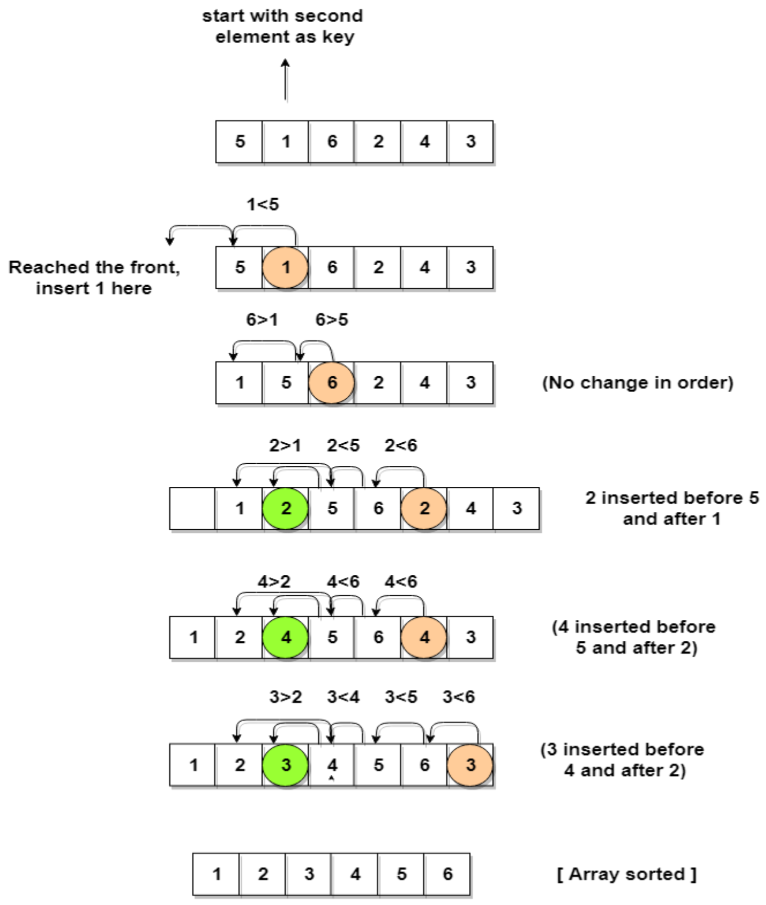
Program: Following is the recursive program to display factorial of values from 1 to 6.

```
#include<stdio.h>
#include<conio.h>
// Recursive definition of function: fact()
int fact(int n)
{
   if (n == 0 || n==1)        // Base case: on input 0 or 1, function returns concrete solution 1
      return 1;
   else
      return n*fact(n-1);
}

// Start of main function
void main()
{
   int f, i;
   for(i=1;i<=6;i++)
   {
        f = fact(i);          // calling the function: fact()
        printf("%d\n", f);
    }
```

```
        getch();
    }
```

Question 2(b): Insertion sort: Let's consider an array with values {5, 1, 6, 2, 4, 3}. Below, we have a pictorial representation of how insertion sort will sort the given array.



```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[6] = {5, 1, 6, 2, 4, 3};
    int i, j, min, index, t;

    for(i=1; i<=5; i++)
    {
        t = a[i];
        for(j=i-1; j>=0; j--)
        {
            if(t<a[j])
            {
                a[j+1] = a[j];
            }
            else
                break;
        }
        a[j+1] = t;
```

```
      }

   // Displaying sorted array a[]
   printf("\nsorted array is:");
   for(i=0; i<=9; i++)
          printf("%d ", a[i]);
   getch();
}
```

Question 2(c):
(i) <u>Call by Value:</u> In this parameter passing method, values of actual parameters are copied to function's formal parameters, and the parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

(ii) <u>Call by Reference:</u> Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument.

Example of 'Call by Value'	**Example of 'Call by Reference'**
`# include <stdio.h>`	`# include <stdio.h>`
`# include <conio.h>`	`# include <conio.h>`
`void swap(int p, int q)`	`void swap(int *p, int *q)`
`{`	`{`
`      int t;`	`      int t;`
`      //swapping logic`	`      //swapping logic using reference`
`      t=p;`	`      t=*p;`
`      p=q;`	`      *p=*q;`
`      q=t;`	`      *q=t;`
`}`	`}`
`void main()`	`void main()`
`{`	`{`
`   int a=10, b=5;`	`   int a=10, b=5;`
`   swap(a,b); //value of a and b is passed`	`   swap(&a, &b); //reference of a and b is passed`
`   printf("After swapping:\na=%d \n b=%d", a, b);`	`   printf("After swapping:\na=%d \n b=%d", a, b);`
`   getch();`	`   getch();`
`}`	`}`
Output:	**Output:**
After swapping:	After swapping:
a=10	a=5
b=5	b=10

Question 2(d): Register Storage Class: The register specifier declares a variable of register storage class. Variables belonging to register storage class are local to the block which they are defined in, and get destroyed on exit from the block. The features of a variable defined to be of register storage class are as given below:

- Storage - CPU registers.

- Default initial value - Garbage value.
- Scope - Local to the block in which the variable is defined.
- Life - Till the control remains within the block in which the variable is defined.

A good example of frequently used variables is loop counters. We can
name their storage class as register.

```
void main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "\n%d", i ) ;
}
```

Static Storage Class: The static specifier assigns the declared variable static storage class. Static variables can be used within function or file. Static variables are not visible outside their function or file, but they maintain their values between calls. The features of a variable defined to have a static storage class are as given below:

- Storage − Memory.
- Default initial value − Zero.
- Scope − Local to the block in which the variable is defined.
- Life − Value of the variable persists between different function calls.

```
#include <stdio.h>
static int gInt = 1;
static void staticDemo()
{
  static int i;
  printf("%d ", i);
  i++;
  printf("%d\n", gInt);
  gInt++;
}

int main()
{
  staticDemo();
  staticDemo();
}
```

OUTPUT:

```
0 1
1 2
```

SECTION – C

Question 3(a):

Structure: Structure is a user defined datatype. It is used to combine different types of data into a single type. It can have multiple members and structure variables. The keyword "struct" is used to define structures in C language. Structure members can be accessed by using dot(.) operator.

<u>Union</u>: Union is also a user defined datatype. All the members of union share the same memory location. Size of union is decided by the size of largest member of union. If you want to use same memory location for two or more members, union is the best for that.

<u>Difference between Structure and Union in C:</u>

BASIS OF COMPARISON	STRUCTURE	UNION
Basic	The separate memory location is allotted to each member of the 'structure'.	All members of the 'union' share the same memory location.

Declaration	struct struct_name{
	type element1;
	type element2;
	.
	.
	} variable1, variable2, ...;
		union union_name{
		type element1;
		type element2;
		.
		.
		} variable1, variable2, ...;

Size	Size of Structure= sum of size of all the data members.	Size of Union=size of the largest members.
At a Time	A 'structure' stores multiple values, of the different members, of the 'structure'.	A 'union' stores a single value at a time for all members.

Example definition and memory allocated for the variable.	struct emp{
	  int id;
	  float sal;
	  char name[20];
	}e1;
	
	Total memory for e1:
	
	2 + 4 + 20 = 26 bytes
		union emp{
		  int id;
		  float sal;
		  char name[20];
		}e2;
		
		Total memory for e2:
		
		Max(2,4,20) = 20 bytes

Question 3(b):

/* Program to transpose input matrix */

```
# include <stdio.h>
# include <conio.h>

void main()
{
    int a[10][10], b[10][10],row,col;
    printf("Enter dimension of input matrix a[][]:")
    scanf("%d%d",&row, &col);

    //Reading matrix a[][]
    for(i=0;i<=row-1;i++)
```

```
         for(j=0;j<=col-1;j++)
                scanf("%d",&a[i][j]);

  //Logic to transpose matrix a[][] and stroing in b[][]
  for(i=0;i<=row-1;i++)
  {
         for(j=0;j<=col-1;j++)
         {
                 b[j][i] = a[i][j];
         }
   }

  //Displaying transposed array b[][]
  for(i=0;i<=col-1;i++)
  {
         for(j=0;j<=row-1;j++)
         {
                  prinf("%d",b[i][j]);
         }
   }
   getch();
}
```

Question 4(a):

/* Program to create student database in a File */

```
#include<stdio.h>
#include<conio.h>
void main()
{
     FILE *fp;
     int roll, i;
     char n[20],add[30];
     float phy, chem, maths;

     fp=fopen("database.txt", "w");
     if(fp==NULL)
              printf("Error");
     else
     {
        for(i=1;i<=10;i++)
        {
            scanf("%d", &roll);
            gets(n);
            gets(add);
            scanf("%f%f%f", &phy,&chem,&maths);

           // Writing the record to file
          fprintf(fp,"%d  %s  %s  %f  %f  %f\n",roll, n, phy, chem, maths);
        }
      }
```

```
    fclose(fp);
    getch();
}
```

Question 4(b):
Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming. To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used. These functions are defined in the <stdlib.h> header file.

malloc(): The name "malloc" stands for memory allocation**.** The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be casted into pointers of any form.

Syntax of malloc(): ptr = (cast_type *) malloc(size);

Example: ptr = (int*) malloc(100 * sizeof(int));

The above statement allocates 200 bytes of memory. It's because the size of int is 2 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.

calloc(): The name "calloc" stands for contiguous allocation. The malloc() function allocates memory and leaves the memory uninitialized. Whereas, the calloc() function allocates memory and initializes all bits to zero.

Syntax of calloc(): ptr = (castType*)calloc(n, size);

Here, 'n' and 'size' specifies number of elements and size of each element respectively.

Example: ptr = (int *) calloc(25, sizeof(int));

The above statement allocates contiguous space in memory for 25 elements of type int.

Question 5(a): C PREPROCESSOR DIRECTIVES: Before a C program is compiled by a compiler, source code is processed by a program called preprocessor. This process based on the preprocessing commands and is called preprocessing. Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol.

Macros: Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro.

Syntax: #define Macro_name Value

/* Program to explain Macros example */

```
# include <stdio.h>
# include <conio.h>
#define SIZE 10             // Macro named SIZE with value 10
```

/* Before compilation value of Macro SIZE is replaced everywhere in the program */
```
void main()
{
    int a[SIZE], i;

    //Reading array a[]
    for(i=0;i<=SIZE-1;i++)
           scanf("%d",&a[i]);

   //Displaying array a[]
   for(i=0;i<=SIZE-1;i++)
         prinf("%d",a[i]);

    getch();
}
```

Question 5(b):

/* Program to check whether input string is palindrome or not */
```
# include <stdio.h>
# include <conio.h>
#include <string.h>

void main()
{
     char a[20];
     int j,j,len;

    //Reading string
   gets(a);
   len=strlen(a);

   j=len-1;

   //palindrome checking logic
   for(i=0;i<=(len/2)-1;i++)
   {
         if(a[i] != a[j])
             break;
        j--;
   }
   if(i==len/2)
             printf("String is Plaindrome");
    else
             printf("String is NOT a Plaindrome");
    getch();
}
```