Solution with scheme-Model Answer

Prof.Rajeshwari R									
Internal Assessment Test 2 – May 2024									
Sub	**Introduction to C Programming**					Sub code	**BESCK204E**	Branch	**ECE**
Date	**22.05.2024**	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	**II Sem P-Cycle (I,J,K,L)**		**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Define function with syntax and example. Also explain the various types of parameter passing techniques with example.	[10]	CO3	L2
2. a.	What is a recursive function? Write the code for finding the factorial of a number using the same.	[6]	CO3	L2
b.	Differentiate actual and formal parameters.	[4]		
3.	Write a C program to implement bubble Sort technique (ascending order). Demonstrate the steps for the input: 9, 20, 6, 3, 2, 12, 15.	[10]	CO3	L2
4.	Write a C code to implement Matrix multiplication and validate the rules of multiplication.	[10]	CO3	L3
5. a.	List out the four different ways to initialize the strings each with example.	[4]	CO4	L3
b.	Define string. Write a program to get a string data from the user and display it using gets and puts function.	[6]	CO4	L2
6.	Write a C program to create user defined functions that implements string operations such as compare, concatenate, copy and length of the string using parameter passing techniques.	[10]	CO4	L3
7.	Explain declaration and initialization of 1-dimensional arrays with example.	[5]	CO3	L2
	Explain declaration and initialization of 2-dimensional arrays with example.	[5]		
8. a.	Write a C program to concatenate two strings without using the library function.	[5]	CO4	L3
b.	Write a C program to transpose a matrix of order 3x3.	[5]	CO4	L3

1.	**Define function with syntax and example. Also explain the various types of parameter passing techniques with example.**
	Definition and Syntax [4M]
	Function is a building block that contains set of programming statements to perform a particular task.
	Function Declaration Syntax:
	return-type function-name(datatype1 parameter1, parameter2,....., datatype n parameter n);
	Function Call Syntax:
	function-name (parameter1,parameter2, ...,parameter n);
	Function Definition Syntax:
	return-type function-name(datatype1 parameter1, parameter2,....., datatype n parameter n)
	{
	Statements;
	}
	The various types of parameter passing techniques are Call by Value and Call by Reference.
		[10]

Program with Output [3M]

Call By Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

Example:

```
#include<stdio.h>
void swapx(int x, int y);
int main()
{
int a = 10, b = 20;
swapx(a, b);
printf("a=%d b=%d\n", a, b);
return0;
}
void swapx(intx,inty)
{ int t;
t = x;
x = y;
y = t;
printf("x=%d y=%d\n", x, y);
}
```

Sample Output: x=20 y=10
a=10 b=20

Program with Output [3M]

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

Example:

```
#include <stdio.h>
voidswapx(int*,int*);
int main()
{
int a = 10, b = 20;
swapx(&a,&b);
printf("a=%d b=%d\n", a, b);
return0;
 }
void swapx(int*x,int*y)
{ intt;
t =*x;
*x =*y;
*y =t;
printf("x=%d y=%d\n", *x, *y);
}
```

2 a **What is a recursive function? Write the code for finding the factorial of a number using the same.** [6]

Definition and Syntax [2M]

Recursive function can be defined as a routine that calls itself the function directly or indirectly.

Syntax Model:

```
int recursion(x);
{
if(x==0)
return;
```

recursion(x-1);}

Program with Output [4M]

```
#include<stdio.h>
int find_factorial(int);
int main()
{
intnum,fact;
printf("\nEnter any integer number:");
scanf("%d",&num);
fact=find_factorial(num);
printf(" \nfactorial of %d is: %d",num, fact);
return0;
}
int find_factorial(int n) {
//Factorial of 0 is 1
if(n==0)
return(1);
return(n*find_factorial(n-1));
}
```

Output:
Enter any integer number:5
factorial of 5 is:120

2 b **Differentiate actual and formal parameters.** [4]

Differences [each 1M]

Actual Parameters	Formal Parameters
When a function is called, the values (expressions) that are passed in the function call are called the arguments or actual parameters.	The parameter used in function definition statements which contain data type on its time of declaration is called formal parameter.
These are the variables or expressions referenced in the parameter list of a subprogram call.	These are the variables or expressions referenced in the parameter list of a subprogram specification.
Actual Parameters are the parameters which are in calling subprogram.	Formal Parameters are the parameters which are in called subprogram.
There is no need to specify datatype in actual parameter.	The datatype of the receiving value must be defined.

3. **Write a C program to implement bubble Sort technique (ascending order). Demonstrate the steps for the input: 5, 1, 4, 2, 8.** [10]

Program [4M]

Program:

```
#include<stdio.h>
int main()
{
int a[20],n,i,j,temp;
printf("Enter the number of elements");
scanf("%",&n);
printf("Enter %d integers);
for(i=0;i<n;i++)
```

```
{
Scanf(“%d”,&a[i]);
}
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1-i;j++)
{
if(a[j] > a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}}}
printf(”The sorted array is ....\n;);
for(i=0;i<n;i++)
{
Printf(”%d”,a[i]);
}
Printf(“\n”);
return 0;
}
```

Output [3]

Output

Steps for the Input: 9, 20, 6, 3, 2, 12, 15
Initial Array: 9, 20, 6, 3, 2, 12, 15
F
irst Pass:

Steps[3]

Compare 9 and 20, no swap needed.
Compare 20 and 6, swap to get: 9, 6, 20, 3, 2, 12, 15
Compare 20 and 3, swap to get: 9, 6, 3, 20, 2, 12, 15
Compare 20 and 2, swap to get: 9, 6, 3, 2, 20, 12, 15
Compare 20 and 12, swap to get: 9, 6, 3, 2, 12, 20, 15
Compare 20 and 15, swap to get: 9, 6, 3, 2, 12, 15, 20
Second Pass:
Compare 9 and 6, swap to get: 6, 9, 3, 2, 12, 15, 20
Compare 9 and 3, swap to get: 6, 3, 9, 2, 12, 15, 20
Compare 9 and 2, swap to get: 6, 3, 2, 9, 12, 15, 20
Compare 9 and 12, no swap needed.
Compare 12 and 15, no swap needed.
Third Pass:
C
ompare 6 and 3, swap to get: 3, 6, 2, 9, 12, 15, 20
Compare 6 and 2, swap to get: 3, 2, 6, 9, 12, 15, 20
Compare 6 and 9, no swap needed.
Compare 9 and 12, no swap needed.
Fourth Pass:
Compare 3 and 2, swap to get: 2, 3, 6, 9, 12, 15, 20
Compare 3 and 6, no swap needed.
Compare 6 and 9, no swap needed.
Fifth Pass:
Compare 2 and 3, no swap needed.
Compare 3 and 6, no swap needed.
Sixth Pass:
No swaps needed as the array is already sorted.
The final sorted array is: 2, 3, 6, 9, 12, 15, 20

4	**Write a C code to implement Matrix multiplication and validate the rules of multiplication.** **Program [7]**
	
	Program:
	#include<stdio.h>
	int main()
	{
	int a[10][10],b[10][10],c[10][10];
	int m,n,p,q;
	int i,j,k;
	printf("Enter the order of matrix A :");
	scanf("%d%d",&m,&n);
	printf("Enter the order of matrix B:");
	scanf("%d%d",&p,&q);
	if(n!=p)
	{
	printf("Number of columns of Matrix A is not equal to number of rows of matrix B\n");
	printf("Multiplication of matrices not possible....\n");
	return (-1);
	}
	printf("\nEnter %d elements into matrix A : ", m*n);
	for(i=0;i<m;i++)
	for(j=0;j<n;j++)
	scanf("%d",&a[i][j]);
	printf("\nThe matrix A is ---\n");
	for(i=0;i<m;i++)
	{
	for(j=0;j<n;j++)
	{
	printf("%d\t",a[i][j]);
	}
	printf("\n");
	}
	printf("\nEnter %d elements into matrix B : ", p*q);
	for (i=0;i<p;i++)
	for (j=0;j<q;j++)
	scanf("%d",&b[i][j]);
	printf("\nThe matrix B is ---\n");
	for(i=0;i<p;i++)
	{
	for(j=0;j<q;j++)
	{
	printf("%d\t",b[i][j]);
	}
	printf("\n");
	}
	for(i=0;i<m;i++)
	{
	for(j=0;j<q;j++)
	{
	c[i][j] = 0;
	for(k=0;k<n;k++)
	{
	c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
	}
	}
		[10]

```
}
printf("\nThe product matrix is ---\n\n");
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
return 0;
}
```

Output[3]

Output:

```
Enter the order of matrix A :2
2
Enter the order of matrix B:2
2
Enter 4 elements into matrix A : 1
2
3
4
The  matrix A is ---
1       2
3       4
Enter 4 elements into matrix B : 1
3
2
4
The matrix B is ---
1       3
2       4
The product matrix is ---

5       11
11      25
```

5 a **List out the four different ways to initialize the strings each with example.** [4]

Types with example [4M]

1. Assigning a String Literal without Size
   Eg: char str[] = "CMRIT";
2. Assigning a String Literal with a Predefined Size
   Eg: char str[7] = "CMRIT";
3. Assigning Character by Character with Size
   Eg: char str[7] = { 'C','M','R','I','T','\0'};
4. Assigning Character by Character without Size
   Eg: char str[] = { 'C','M','R','I','T','\0'};

5 b **Define string. Write a program to get a string data from the user and display it using gets and puts function.**

Definition[2]

Strings are defined as an array of characters. The difference between a character array and a string is

	the string is terminated with a special character '\0'.
	Eg: char c[]="Hi Students"
	Program[4]
	
	Program:
	int main()
	{
	char str[100];
	printf("Enter a string: ");
	fgets(str, sizeof(str), stdin);
	str[strcspn(str, "\n")] = 0;
	printf("You entered: ");
	puts(str);
	return 0;
	}
	Output:
	Enter a string: Hello
	You entered: Hello
		[6]

6.	**Write a C program to create user defined functions that implements string operations such as compare, concatenate, copy and length of the string using parameter passing techniques.**
	Program[7M]
	Program:
	#include <stdio.h>
	int str_length(char []);
	int str_compare(char [], char []);
	void str_concat(char [], char []);
	int main()
	{
	char str[50];
	char str1[50], str2[50];
	char str_des[100], str_src[50];
	int length, comp_res;
	printf("\nEnter a string :");
	scanf("%s",str);
	length = str_length(str);
	printf("The length of %s is %d\n",str,length);
	printf("\nEnter two strings for string compare :");
	scanf("%s%s",str1,str2);
	comp_res=str_compare(str1,str2);
	if (comp_res < 0)
	{
	printf("String \"%s\" is less than string \"%s\"\n",str1,str2);
	}
	else if (comp_res == 0)
	{
	printf("String \"%s\" is same as string \"%s\"\n",str1,str2);
	}
	else
	{
	printf("String \"%s\" is greater than string \"%s\"\n", str1,str2);
	}
	printf("\nEnter two strings for string concatenation :");
	scanf("%s%s",str_des,str_src);
		[10]

```
 str_concat(str_des,str_src);
 printf("The string after concatenation is \"%s\"\n", str_des);
 return 1;
 }
 int str_length(char s[])
 {
 int i;
for(i=0;s[i]!='\0';i++);
return i;
 }
 int str_compare(char s1[], char s2[])
 {
int i,j;
for(i=0,j=0;(s1[i] != '\0' && s2[j] != '\0');i++,j++)
 {
 if (s1[i] != s2[j])
 {
 return (s1[i] - s2[j]);
 }
 }
 if (s1[i] == '\0' && s2[j] == '\0')
 {
 return 0;
}
 else if(s1[i] == '\0' || s2[i] == '\0')
 {
 return (s1[i] - s2[i]);
 }
 }
void str_concat(char s1[], char s2[])
 {
 int i,j;
for(i=0;s1[i] != '\0';i++);
for(j=0;s2[j] != '\0';i++,j++)
 {
 s1[i] = s2[j];
 }
 s1[i] = '\0';
 }
```

Output[3M]

Output
Enter a string :rainbow
The length of rainbow is 7
Enter two strings for string compare :rain
rainbow
String "rain" is less than string "rainbow"
Enter two strings for string concatenation :rain
bow
The string after concatenation is "rainbow"

7. **Explain declaration and initialization of 1-dimensional arrays with example.**
Def & Syntax with eg[5] [5]

Array in C is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name.
An array is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.

Declaration of 1-D array
Syntax of Array Declaration
data_type array_name [size];
or
data_type array_name [size1] [size2]...[sizeN];
where N is the number of dimensions.

Example of Array Declaration

```
#include <stdio.h>

int main()
{

    // declaring array of integers
    int arr_int[5];
    // declaring array of characters
    char arr_char[5];

    return 0;
}
```

Array Initialization
An initializer list is the list of values enclosed within braces { } separated b a comma.
data_type array_name [size] = {value1, value2, ... valueN};

Array Initialization with Declaration without Size
data_type array_name[] = {1,2,3,4,5};

Array Initialization after Declaration (Using Loops)
We initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

```
for (int i = 0; i < N; i++) {
   array_name[i] = valuei;
}
```

Example of Array Initialization in C

```
#include <stdio.h>

int main()
{

    // array initialization using initialier list
    int arr[5] = { 10, 20, 30, 40, 50 };

    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };

    // array initialization using for loop
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    return 0;
}
```

7b. Explain declaration and initialization of 2-dimensional arrays with example. Def & Syntax with eg[5]

A 2-dimensional array in C is essentially an array of arrays. It's useful for representing data in a tabular form (rows and columns).

Declaration
To declare a 2-dimensional array, you need to specify the data type, the array name, and the dimensions in square brackets. The general syntax is:

```
data_type array_name[row_size][column_size];
```

Here, data_type represents the type of data the array will hold (e.g., int, float), array_name is the identifier for the array, row_size is the number of rows, and column_size is the number of columns.

Example:

```
int matrix[3][4];
```

This declaration creates a 2-dimensional array named matrix with 3 rows and 4 columns.

Initialization
You can initialize a 2-dimensional array at the time of declaration. There are multiple ways to do this:

1. Full Initialization:
You can provide values for all elements in the array.

```
int matrix[3][4] = { {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
};
```

2.Partial Initialization:
If you do not provide enough values to initialize the entire array, the remaining elements are set to zero by default.

```
int matrix[3][4] = {
   {1, 2, 3},    // 4th element of the first row is 0
   {5, 6},       // 3rd and 4th elements of the second row are 0
   {9}           // 2nd, 3rd, and 4th elements of the third row are 0
};
```

3. Initialization Without Specifying All Dimensions:
If you do not specify the size of the first dimension, the compiler determines it based on the initializer.

```
int matrix[][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
};
```

Accessing Elements
To access or modify an element in a 2-dimensional array, you use the row and column indices:

```
 matrix[1][2] = 7; // Sets the element in the 2nd row and 3rd column to 7
int value = matrix[0][3]; // Gets the element in the 1st row and 4th column
```

Example

```
#include <stdio.h>

int main() {
   // Declaration and initialization of a 2-dimensional array
   int matrix[3][4] = {
      {1, 2, 3, 4},
      {5, 6, 7, 8},
      {9, 10, 11, 12}
   };
    // Printing the array elements
   printf("The 2-dimensional array is:\n");
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 4; j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
    return 0;
}
```

OUTPUT:
The 2-dimensional array is:
1 2 3 4
5 6 7 8
9 10 11 12

[5]

8 a **Write a C program to concatenate two strings without using the library function.** [5]

Program[4.5]

Program:

```
#include <stdio.h>
 int main()
 {
 char str1[100] = "Geeks", str2[100] = "World";
 char str3[100];
 int i = 0, j = 0;
  printf("\nFirst string: %s", str1);
  printf("\nSecond string: %s", str2);
 while (str1[i] != '\0') {
   str3[j] = str1[i];
  i++,j++;
 }
 i=0;
  while (str2[i] != '\0') {
  str3[j] = str2[i];
  i++, j++;
 }
 str3[j] = '\0';
  printf("\nConcatenated string: %s", str3);
  return 0;
 }
```

Output[0.5]

Output:

```
First string: Geeks
Second string: World
Concatenated string: GeeksWorld
```

8 b [5]

Write a C program to transpose a matrix of order 3x3.

Program[4.5]

Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int mat[3][3], i, j, matTrans[3][3];
    printf("Enter 3*3 Matrix Elements: ");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            scanf("%d", &mat[i][j]);
    }
    // Transposing the Matrix...
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            matTrans[j][i] = mat[i][j];
    }
    printf("\nTranspose of given Matrix is:\n");
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            printf("%d  ", matTrans[i][j]);
        printf("\n");
    }
    getch();
    return 0;
}
```

Output[0.5]

Output:

```
Enter 3x3 Matrix Elements:
1 2 3
4 5 6
7 8 9
```

```
Transpose of given Matrix is:

1  4  7
2  5  8
3  6  9
```