QP CODE 210393

# PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

**END SEMESTER EXAMINATION**
**SECOND HALF 2021(Supplementary)**
**BRANCH: FE (COMP / IT / ECS)**

**Subject: Programming in C** **Time: 02.00 Hours**
**Max. Marks: 60** **Date: 02-06 -2022**
**N.B 1. Q.1 is compulsory**
**2. Attempt any two from the remaining three questions**

| Q.1. | Attempt all | M | BT | CO |
|---|---|---|---|---|
| a) | **Define Algorithms. Write an algorithm to find the area and perimeter of a circle.**<br>An algorithm is a sequence of finite logical steps required to perform a specific task.<br><br>Algorithm to find the area and perimeter of a circle:<br>STEP 1 : START<br>STEP 2 : INPUT radius<br>STEP 3 : Calculate Area = 3.14 * radius * radius<br>STEP 4 : Calculate Perimeter = 2 * 3.14 * radius<br>STEP 5 : DISPLAY Area and Perimeter<br>STEP 6 : STOP | 5 | 1 | 1 |
| b) | **Explain switch statement with syntax and example.**<br><br>The switch statement allows us to execute one code block among many alternatives. The expression is evaluated once and compared with the values of each case label. If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to constant2, statements after case constant2: are executed until break is encountered. If there is no match, the default statements are executed.<br>If we do not use the break statement, all statements after the matching label are also executed. The default clause inside the switch statement is optional.<br>Syntax :<br>`switch (expression)`<br>`{`<br>`    case constant1:`<br>`      // statements`<br>`      break;`<br><br>`    case constant2:`<br>`      // statements`<br>`      break;`<br>`    .`<br>`    .`<br>`    .`<br>`    default:`<br>`      // default statements`<br>`}`<br>Any valid example | 5 | 4 | 3 |

**PILLAI COLLEGE OF ENGINEERING, NEW PANVEL**
**(Autonomous) (Accredited 'A+' by NAAC)**

**END SEMESTER EXAMINATION**
**SECOND HALF 2021(Supplementary)**
**BRANCH: FE (COMP / IT / ECS)**

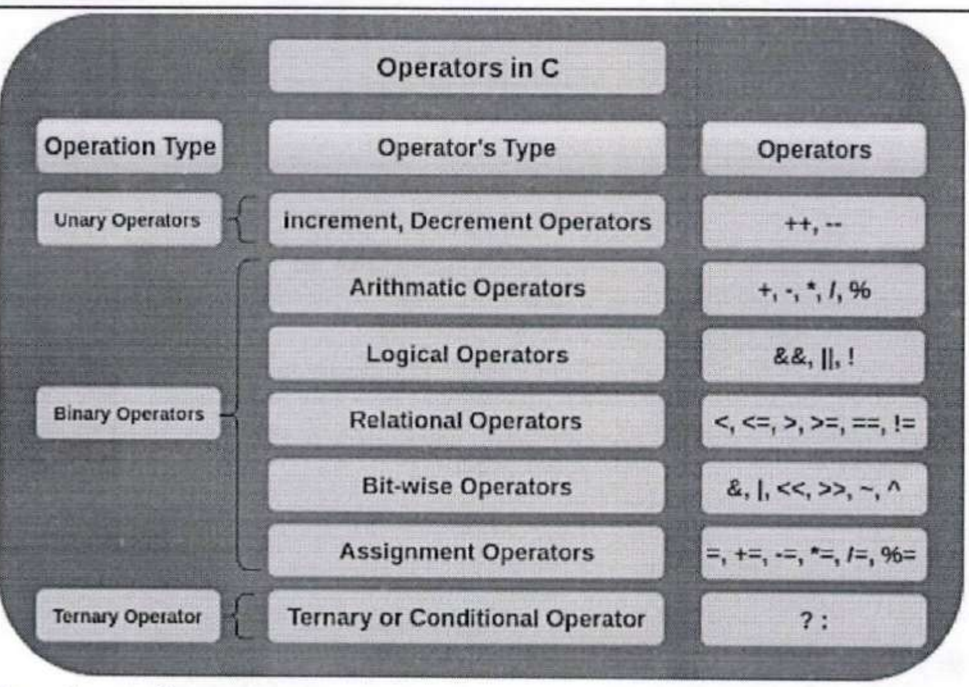
c) **Write a C-program using a function to check whether the given number is prime or not.** | 5 | 3 | 4

```
#include<stdio.h>

int checkPrime(int x);

void main()
{
      int n;
      clrscr();

      printf("Enter n = ");
      scanf("%d",&n);

      if(checkPrime(n))
            printf("It is prime no.");
      else
            printf("It is not prime no.");

      getch();
}

int checkPrime(int x)
{
      int i;

      for(i=2; i<=x/2; i++)
      {
            if(x%i == 0)
                  return 0;
      }

      return 1;
}

/*
OUTPUT :

Enter n = 13
It is prime no.

*/
```

| | | | | |
|---|---|---|---|---|
| d) | **List and explain operators in C . Also write operators precedence.**<br>An operator is simply a symbol that is used to perform operations. Explain all operators: | 5 | 1 | 2 |

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)



Operators in C

| Operation Type | Operator's Type | Operators |
|---|---|---|
| Unary Operators | increment, Decrement Operators | ++, -- |
| Binary Operators | Arithmatic Operators | +, -, *, /, % |
| | Logical Operators | &&, \|\|, ! |
| | Relational Operators | <, <=, >, >=, ==, != |
| | Bit-wise Operators | &, \|, <<, >>, ~, ^ |
| | Assignment Operators | =, +=, -=, *=, /=, %= |
| Ternary Operator | Ternary or Conditional Operator | ? : |

Precedence of operators:

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

**PILLAI COLLEGE OF ENGINEERING, NEW PANVEL**
**(Autonomous) (Accredited 'A+' by NAAC)**

**END SEMESTER EXAMINATION**
**SECOND HALF 2021(Supplementary)**
**BRANCH: FE (COMP / IT / ECS)**

**Q.2.** **Attempt all**

**a)** **Differentiate between call by value and call by reference.** — 4 | 4 | 4

| Call by value | Call by reference |
|---|---|
| A copy of the value is passed into the function | An address of value is passed into the function |
| Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters. | Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters. |
| Actual and formal arguments are created at the different memory location | Actual and formal arguments are created at the same memory location |

with example

**b)** **Implement a C program to find the reverse of an integer number and check whether it is palindrome or not.** — 4 | 6 | 3

```
#include<stdio.h>
void main()
{
      int n, rem, reverse=0, temp;
      clrscr();

      printf("Enter n = ");
      scanf("%d",&n);

      temp=n;

      while(n!=0)
      {     rem = n%10;
            n = n/10;
            reverse = reverse*10 + rem;
      }

      printf("Reverse of no. = %d\n",reverse);

      if(temp == reverse)
      {     printf("Palindrome no.");
      }
      else
      {     printf("Not a palindrome no.");
      }

      getch();
}
```

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

c) **Write a C program to read N integers into an array A and to find the (i)sum of odd numbers,(ii) sum of even numbers,(iii) average of all numbers. Output the results computed with appropriate headings.** 6 6 4

```
#include<stdio.h>

void main()
{
      int n, a[10], sum_odd=0, sum_even=0,i;
      clrscr();

      printf("Enter n = ");
      scanf("%d",&n);

      printf("Enter elements : ");
      for(i=0; i<n; i++)
      {
            scanf("%d",&a[i]);
      }

      //calculate sum
      for(i=0; i<n; i++)
      {
            if(a[i] % 2 == 0)
                  sum_even = sum_even + a[i];
            else
                  sum_odd = sum_odd + a[i];
      }

      printf("\nSum of even terms = %d",sum_even);
      printf("\nSum of odd terms = %d", sum_odd);
      printf("\nAverage of all terms = %f", (float)(sum_even
+ sum_odd)/n);
      getch();
}

/*
OUTPUT :

Enter n = 5
Enter elements : 1
2
3
4
5

Sum of even terms = 6
Sum of odd terms = 9
Average of all terms = 3.000000

*/
```

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| d) | **Explain dynamic allocation using calloc, malloc, realloc, free.**<br><br>**Dynamic Memory Allocation**<br>An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it. 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. Library functions defined in the <stdlib.h> header file used are :<br><br>**malloc()**<br>Stands for memory allocation<br>It reserves a block of memory of the specified number of bytes.<br>It returns a pointer of void which can be casted into pointers of any form.<br>Syntax : ptr = (castType*) malloc(size);<br>Example : ptr = (float*) malloc(100 * sizeof(float));<br><br>**calloc()**<br>Stands for contiguous allocation.<br>The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.<br>Syntax:<br>ptr = (castType*)calloc(n, size);<br>Example:<br>ptr = (float*) calloc(25, sizeof(float));<br><br>**realloc()**<br>To change the size of previously allocated memory.<br>Syntax:<br>ptr = realloc(ptr, newSize);<br><br>**free()**<br>Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.<br>Syntax:<br>free(ptr); | 6 | 4 | 5 |
| Q.3. | **Attempt all** | | | |
| a) | **What is a pointer? Explain how the pointer variable is declared and initialized?**<br>Pointer is a variable which stores the address of another variable. The size of the pointer is 2 bytes irrespective of its datatype.<br>**Declaring a pointer :** int *a;<br>Example:<br>int number=50;<br>int *p;<br>p=&number; **//initializing pointer** | 4 | 1 | 5 |

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

b) **Explain the array of pointers with an example? or Explain how pointers and arrays are related with examples.** 4 3 5

An array of pointers is an array that consists of variables of pointer type, which means that the variable is a pointer addressing to some other element.

```
int *ptr[5];      // array of 5 integer pointers.
```

In the above declaration, we declare an array of pointers named as ptr, and it allocates 5 integer pointers in memory.

```
int a; // variable declaration.
ptr[2] = &a;
```

In the above code, we are assigning the address of 'a' variable to the third element of an array 'ptr'.

We can also retrieve the value of 'a' by dereferencing the pointer.

```
*ptr[2];
```

Ex:

```
int main()
{
 char *names[5] = {"john",  "Peter", "Marco",  "Devin",  "Ronan"};
 for(int i=0;i<5;i++)
   {
     printf("%s\n", names[i]);
   }
   return 0;
}
```

OUTPUT:

```
john
Peter
Marco
Devin
Ronan
```

In most contexts, array names decay to pointers. In simple words, array names are converted to pointers. That's the reason why you can use pointers to access elements of arrays. However, you should remember that pointers and arrays are not the same.

An array is represented by a variable that is associated with the address of its first storage location. A pointer is also the address of a storage location with a defined type, so D permits the use of the array [ ] index notation with both pointer variables and array variables. For example, the following two D fragments are equivalent in meaning: p = &a[0];

```
printf(a[2]);
printf(p[2]);
```

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| c) | **Define a string. Explain any 4 string library functions with syntax and example.**<br>A string can be defined as a one-dimensional array of characters terminated by a null charater('\0').<br>**String functions - string.h (any 4)**<br>strlen(s1) - returns the length of string s1.<br>strcpy(s1,s2) - copy string s2 to s1.<br>strncpy(s1,s2,n) - copy n characters from s2 to s1.<br>strcat(s1,s2) - joins string s2 at the end of s1.<br>strncat(s1,s2,n) - joins n characters from s2 at the end of s1.<br>strcmp(s1,s2) - compares string s1 & s2, returns<br>= 0 if s1 = s2<br>> 0 if the first non-matching character in s1 is greater (in ASCII) than that of s2.<br>< 0 if the first non-matching character in s1 is lower (in ASCII) than that of s2.<br>strcmpi(s1,s2) - ignores uppercase & lowercase for comparison (same as strcmp(s1,s2)).<br>strrev(s1) - returns reverse string.<br>strlwr(s1) - returns string characters in lowercase.<br>strupr(s1) - returns string characters in uppercase.<br>strchr(s1,ch) - to find the first occurrence of a specified character in string s1.<br>strrchr(s1,ch) - to find the last occurrence of a specified character in string s1.<br>strstr(s1,s2) - to find the first occurrence of substring s2 in string s1.<br>strrstr(s1,s2) - to find the last occurrence of substring s2 in string s1. | 6 | 1 | 4 |
| d) | **What is a file? Explain with an example the file handling functions.**<br>A file is a collection of bytes which is stored on a secondary storage device like a hard disk to store data permanently.<br>**File handling:**<br>1. Declare FILE pointer variable<br>Syntax : FILE *fptr;<br>where FILE is a predefined structure which is defined in the header file <stdio.h>.<br>2. Open file using **fopen()**<br>Syntax : fptr = fopen("filename.txt", "mode");<br>where "mode" specifies in which mode (read/write) file is to be opened.<br>3. Process file using suitable read/write functions.<br>**Reading from file**<br>fgetc(fptr)<br>- returns a single character from the file. It returns EOF at the end of file.<br>fscanf(fptr, "format specifiers", address of variables)<br>- is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.<br>fgets(str1,n,fptr)<br>- reads a line of characters from file. It gets string from a stream.<br>fread(pointer, size_of_each, no. of elements, fptr)<br>- reads the entire record at a time.<br>**Writing to file**<br>fputc(ch, fptr) | 6 | 1 | 6 |

- it is used to write a single character into file.

fprintf(fptr, "format string", list of variables)

- it is used to write set of characters into file. It sends formatted output to a stream.

fputs("string",fptr)

- writes a line of characters into file. It outputs string to a stream.

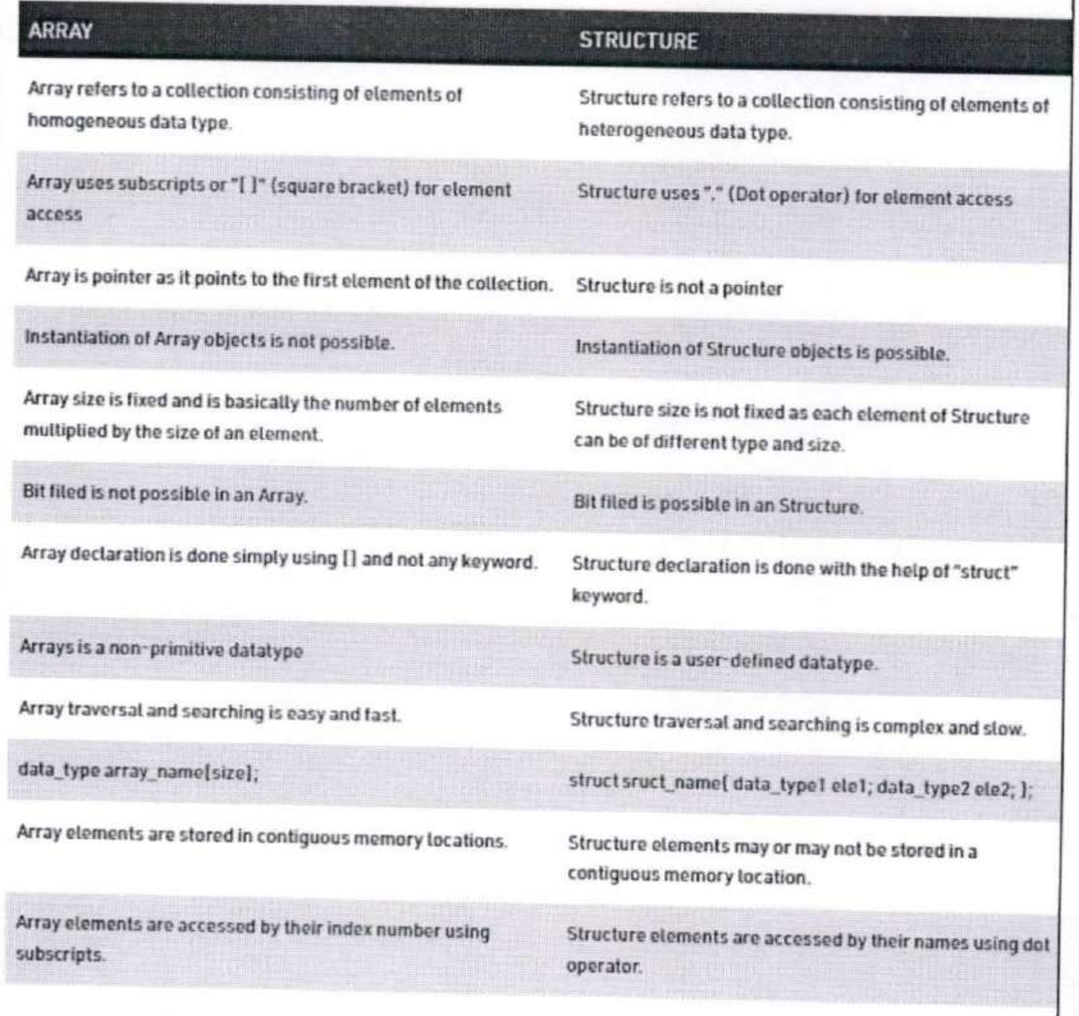
fwrite(pointer, size_of_each, no. of elements, fptr)

- writes an entire record at a time.

4. Close the file using fclose()
   Syntax : **fclose(fptr);**

**Q.4. Attempt all**

a) **Explain the difference between array and structures.** 4 3 4



| ARRAY | STRUCTURE |
|---|---|
| Array refers to a collection consisting of elements of homogeneous data type. | Structure refers to a collection consisting of elements of heterogeneous data type. |
| Array uses subscripts or "[ ]" (square bracket) for element access | Structure uses "." (Dot operator) for element access |
| Array is pointer as it points to the first element of the collection. | Structure is not a pointer |
| Instantiation of Array objects is not possible. | Instantiation of Structure objects is possible. |
| Array size is fixed and is basically the number of elements multiplied by the size of an element. | Structure size is not fixed as each element of Structure can be of different type and size. |
| Bit filed is not possible in an Array. | Bit filed is possible in an Structure. |
| Array declaration is done simply using [] and not any keyword. | Structure declaration is done with the help of "struct" keyword. |
| Arrays is a non-primitive datatype | Structure is a user-defined datatype. |
| Array traversal and searching is easy and fast. | Structure traversal and searching is complex and slow. |
| data_type array_name[size]; | struct sruct_name{ data_type1 ele1; data_type2 ele2; }; |
| Array elements are stored in contiguous memory locations. | Structure elements may or may not be stored in a contiguous memory location. |
| Array elements are accessed by their index number using subscripts. | Structure elements are accessed by their names using dot operator. |

4 4 3

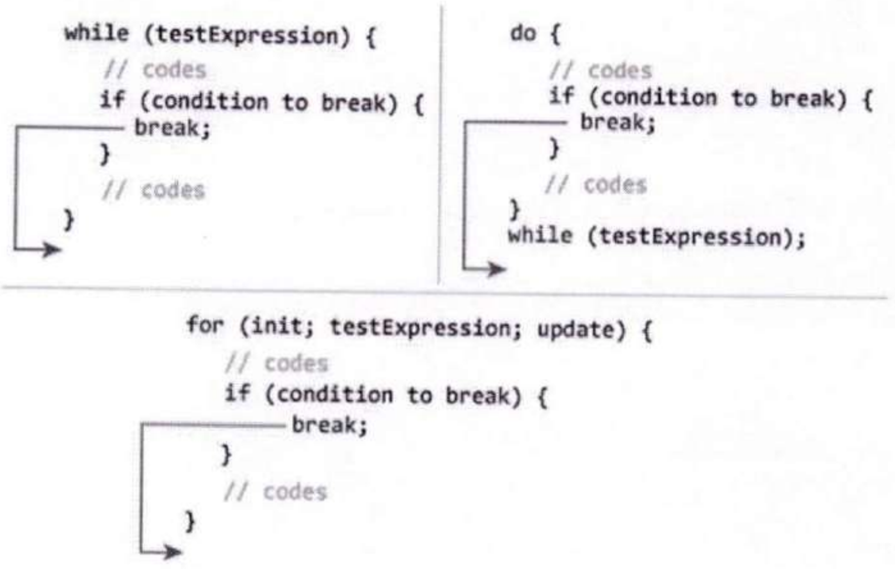
PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

b) **Show how break and continue statements are used in a C program, with example.**

The primary difference between break and continue statement in C is that the break statement leads to an immediate exit of the innermost switch or enclosing loop. On the other hand, the continue statement begins the next iteration of the while, enclosing for, or do loop.

**break statement:**

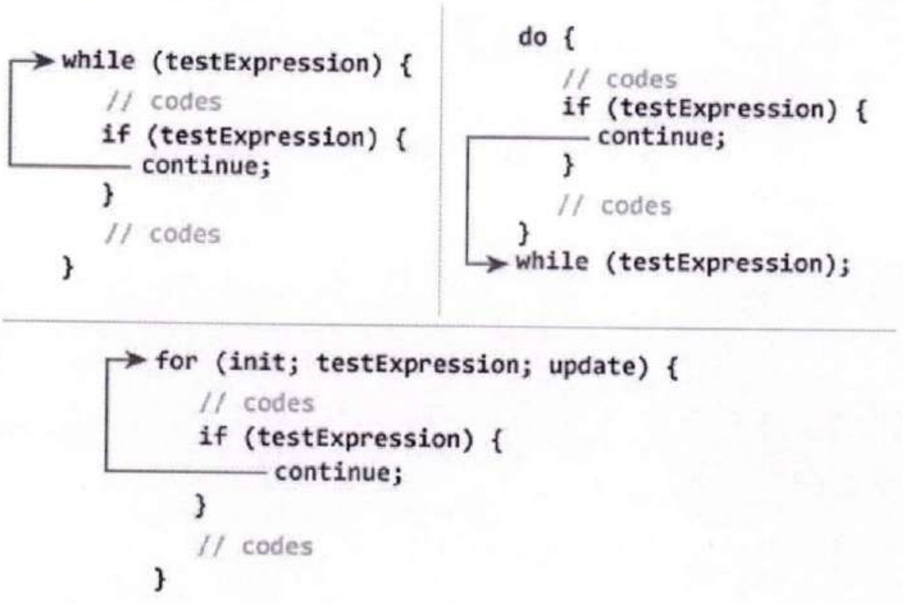


```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

**continue statement:**



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

any valid example

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

c) **Explain the concept of recursive function using the example program to find the factorial of a given positive number.**

Recursion - The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Syntax for recursive function:

```
void recursiveFunc()
{
if(base case)
            //something
      else
            //something
            recursiveFunc();      //recursive call
}
```

**program:**

```
#include<stdio.h>

int factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}

void main()
{
  int number;
  int fact;

  printf("Enter a number: ");
  scanf("%d", &number);

  fact = factorial(number);

  printf("Factorial of %d is %d\n", number, fact);

  return 0;
}
```

Output:

```
Enter a number: 6
Factorial of 5 is: 720
```

6 | 3 | 4

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| d) | **Write a C program to compute transpose of a given matrix.** (code below) | 6 | 6 | 4 |

```c
#include <stdio.h>

int main()
{
  int a[10][10], transpose[10][10], r, c;

  printf("Enter rows and columns: ");
  scanf("%d %d", &r, &c);

  // asssigning elements to the matrix
  printf("\nEnter matrix elements:\n");
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < c; ++j)
  {
    printf("Enter element a%d%d: ", i + 1, j + 1);
    scanf("%d", &a[i][j]);
  }

  // printing the matrix a[][]
  printf("\nEntered matrix: \n");
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < c; ++j)
  {
    printf("%d  ", a[i][j]);
    if (j == c - 1)
    printf("\n");
  }

  // computing the transpose
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < c; ++j)
  {
    transpose[j][i] = a[i][j];
  }

  // printing the transpose
  printf("\nTranspose of the matrix:\n");
  for (int i = 0; i < c; ++i)
  {  for (int j = 0; j < r; ++j)
    {
      printf("%d  ", transpose[i][j]);
    }
      printf("\n");
  }

  return 0;
}
```

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

```
OUTPUT :

Enter rows and columns: 2
3

Enter matrix elements:
Enter element a11: 1
Enter element a12: 4
Enter element a13: 0
Enter element a21: -5
Enter element a22: 2
Enter element a23: 7

Entered matrix:
1  4  0
-5  2  7

Transpose of the matrix:
1  -5
4  2
0  7
```

**CO1:** Understand the basic terminology used in computer programming.

**CO2:** Use different data types, operators and keywords to write programs.

**CO3:** Able to logically code using control statements and loops.

**CO4:** Use the concepts of arrays, strings, functions and Structures to structure complex programs.

**CO5:** Use of pointers to access different user defined data types like arrays, Strings and Structures.

**CO6:** Use different data structures and open/create/update basic data files.

**BT Levels**: - 1 Remembering ,2 Understanding, 3 Applying,4 Analyzing, 5 Evaluating, 6 Creating.

M-Marks, **BT**- Bloom's Taxonomy, **CO**-Course Outcomes.

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| d) | **Explain dynamic allocation using calloc, malloc, realloc, free.**<br><br>**Dynamic Memory Allocation**<br>An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it. 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. Library functions defined in the <stdlib.h> header file used are :<br><br>**malloc()**<br>Stands for memory allocation<br>It reserves a block of memory of the specified number of bytes.<br>It returns a pointer of void which can be casted into pointers of any form.<br>Syntax : ptr = (castType*) malloc(size);<br>Example : ptr = (float*) malloc(100 * sizeof(float));<br><br>**calloc()**<br>Stands for contiguous allocation.<br>The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.<br>Syntax:<br>ptr = (castType*)calloc(n, size);<br>Example:<br>ptr = (float*) calloc(25, sizeof(float));<br><br>**realloc()**<br>To change the size of previously allocated memory.<br>Syntax:<br>ptr = realloc(ptr, newSize);<br><br>**free()**<br>Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.<br>Syntax:<br>free(ptr); | 6 | 4 | 5 |
| Q.3. | **Attempt all** | | | |
| a) | **What is a pointer? Explain how the pointer variable is declared and initialized?**<br>Pointer is a variable which stores the address of another variable. The size of the pointer is 2 bytes irrespective of its datatype.<br>**Declaring a pointer :** int *a;<br>Example:<br>int number=50;<br>int *p;<br>p=&number; **//initializing pointer** | 4 | 1 | 5 |

QP CODE 210393

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| c) | **Write a C program to read N integers into an array A and to find the (i)sum of odd numbers,(ii) sum of even numbers,(iii) average of all numbers. Output the results computed with appropriate headings.** | 6 | 6 | 4 |

```
#include<stdio.h>

void main()
{
     int n, a[10], sum_odd=0, sum_even=0,i;
     clrscr();

     printf("Enter n = ");
     scanf("%d",&n);

     printf("Enter elements : ");
     for(i=0; i<n; i++)
     {
           scanf("%d",&a[i]);
     }

     //calculate sum
     for(i=0; i<n; i++)
     {
           if(a[i] % 2 == 0)
                 sum_even = sum_even + a[i];
           else
                 sum_odd = sum_odd + a[i];
     }

     printf("\nSum of even terms = %d",sum_even);
     printf("\nSum of odd terms = %d", sum_odd);
     printf("\nAverage of all terms = %f", (float)(sum_even
+ sum_odd)/n);
     getch();
}

/*
OUTPUT :

Enter n = 5
Enter elements : 1
2
3
4
5

Sum of even terms = 6
Sum of odd terms = 9
Average of all terms = 3.000000

*/
```

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

b)

**Explain the array of pointers with an example? or Explain how pointers and arrays are related with examples.**

An array of pointers is an array that consists of variables of pointer type, which means that the variable is a pointer addressing to some other element.

```
int *ptr[5];      // array of 5 integer pointers.
```

In the above declaration, we declare an array of pointers named as ptr, and it allocates 5 integer pointers in memory.

```
int a; // variable declaration.
ptr[2] = &a;
```

In the above code, we are assigning the address of 'a' variable to the third element of an array 'ptr'.

We can also retrieve the value of 'a' by dereferencing the pointer.

```
*ptr[2];
```

Ex:

```
int main()
{
 char *names[5] = {"john",  "Peter", "Marco",  "Devin",  "Ronan"};
 for(int i=0;i<5;i++)
   {
     printf("%s\n", names[i]);
   }
   return 0;
}
```

OUTPUT:

```
john
Peter
Marco
Devin
Ronan
```

In most contexts, array names decay to pointers. In simple words, array names are converted to pointers. That's the reason why you can use pointers to access elements of arrays. However, you should remember that pointers and arrays are not the same.

An array is represented by a variable that is associated with the address of its first storage location. A pointer is also the address of a storage location with a defined type, so D permits the use of the array [ ] index notation with both pointer variables and array variables. For example, the following two D fragments are equivalent in meaning: p = &a[0];

```
printf(a[2]);
printf(p[2]);
```

4 | 3 | 5

| | | | | |
|---|---|---|---|---|
| c) | **Define a string. Explain any 4 string library functions with syntax and example.**<br>A string can be defined as a one-dimensional array of characters terminated by a null charater('\0').<br>**String functions - string.h (any 4)**<br>strlen(s1) - returns the length of string s1.<br>strcpy(s1,s2) - copy string s2 to s1.<br>strncpy(s1,s2,n) - copy n characters from s2 to s1.<br>strcat(s1,s2) - joins string s2 at the end of s1.<br>strncat(s1,s2,n) - joins n characters from s2 at the end of s1.<br>strcmp(s1,s2) - compares string s1 & s2, returns<br>= 0 if s1 = s2<br>> 0 if the first non-matching character in s1 is greater (in ASCII) than that of s2.<br>< 0 if the first non-matching character in s1 is lower (in ASCII) than that of s2.<br>strcmpi(s1,s2) - ignores uppercase & lowercase for comparison (same as strcmp(s1,s2)).<br>strrev(s1) - returns reverse string.<br>strlwr(s1) - returns string characters in lowercase.<br>strupr(s1) - returns string characters in uppercase.<br>strchr(s1,ch) - to find the first occurrence of a specified character in string s1.<br>strrchr(s1,ch) - to find the last occurrence of a specified character in string s1.<br>strstr(s1,s2) - to find the first occurrence of substring s2 in string s1.<br>strrstr(s1,s2) - to find the last occurrence of substring s2 in string s1. | 6 | 1 | 4 |
| d) | **What is a file? Explain with an example the file handling functions.**<br>A file is a collection of bytes which is stored on a secondary storage device like a hard disk to store data permanently.<br>**File handling:**<br>1. Declare FILE pointer variable<br>Syntax : FILE *fptr;<br>where FILE is a predefined structure which is defined in the header file <stdio.h>.<br>2. Open file using **fopen()**<br>Syntax : fptr = fopen("filename.txt", "mode");<br>where "mode" specifies in which mode (read/write) file is to be opened.<br>3. Process file using suitable read/write functions.<br>**Reading from file**<br>fgetc(fptr)<br>- returns a single character from the file. It returns EOF at the end of file.<br>fscanf(fptr, "format specifiers", address of variables)<br>- is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.<br>fgets(str1,n,fptr)<br>- reads a line of characters from file. It gets string from a stream.<br>fread(pointer, size_of_each, no. of elements, fptr)<br>- reads the entire record at a time.<br>**Writing to file**<br>fputc(ch, fptr) | 6 | 1 | 6 |

- it is used to write a single character into file.

fprintf(fptr, "format string", list of variables)

- it is used to write set of characters into file. It sends formatted output to a stream.

fputs("string",fptr)

- writes a line of characters into file. It outputs string to a stream.

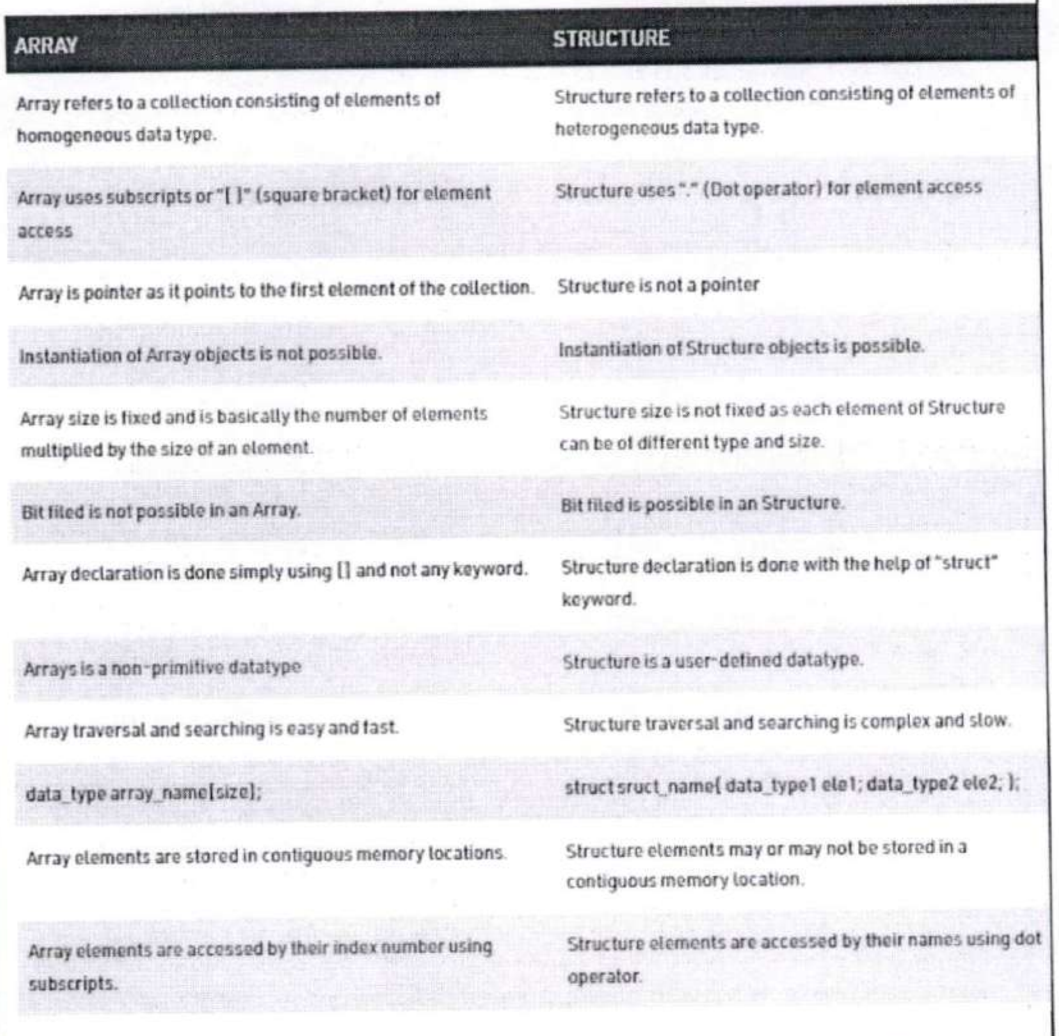
fwrite(pointer, size_of_each, no. of elements, fptr)

- writes an entire record at a time.

4. Close the file using fclose()
   Syntax : **fclose(fptr);**

Q.4. **Attempt all**

a) **Explain the difference between array and structures.** 4 3 4



| ARRAY | STRUCTURE |
|---|---|
| Array refers to a collection consisting of elements of homogeneous data type. | Structure refers to a collection consisting of elements of heterogeneous data type. |
| Array uses subscripts or "[ ]" (square bracket) for element access | Structure uses "." (Dot operator) for element access |
| Array is pointer as it points to the first element of the collection. | Structure is not a pointer |
| Instantiation of Array objects is not possible. | Instantiation of Structure objects is possible. |
| Array size is fixed and is basically the number of elements multiplied by the size of an element. | Structure size is not fixed as each element of Structure can be of different type and size. |
| Bit filed is not possible in an Array. | Bit filed is possible in an Structure. |
| Array declaration is done simply using [] and not any keyword. | Structure declaration is done with the help of "struct" keyword. |
| Arrays is a non-primitive datatype | Structure is a user-defined datatype. |
| Array traversal and searching is easy and fast. | Structure traversal and searching is complex and slow. |
| data_type array_name[size]; | struct sruct_name{ data_type1 ele1; data_type2 ele2; }; |
| Array elements are stored in contiguous memory locations. | Structure elements may or may not be stored in a contiguous memory location. |
| Array elements are accessed by their index number using subscripts. | Structure elements are accessed by their names using dot operator. |

4 4 3

QP CODE 210393

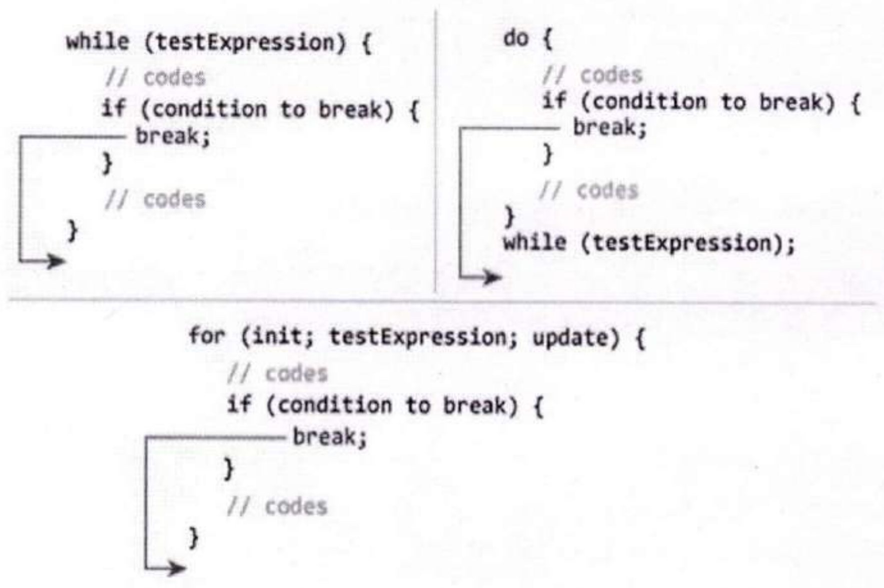
PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

b) **Show how break and continue statements are used in a C program, with example.**

The primary difference between break and continue statement in C is that the break statement leads to an immediate exit of the innermost switch or enclosing loop. On the other hand, the continue statement begins the next iteration of the while, enclosing for, or do loop.

**break statement:**

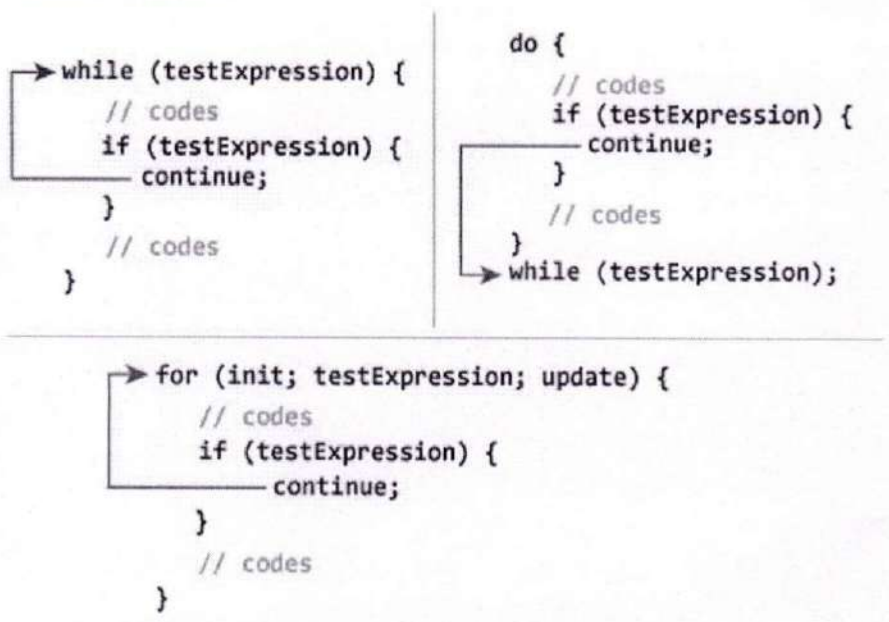


```
while (testExpression) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

**continue statement:**



```
while (testExpression) {
   // codes
   if (testExpression) {
     continue;
   }
   // codes
}
```

```
do {
   // codes
   if (testExpression) {
     continue;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (testExpression) {
     continue;
   }
   // codes
}
```

any valid example

c) **Explain the concept of recursive function using the example program to find the factorial of a given positive number.**

Recursion - The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Syntax for recursive function:

```
 void recursiveFunc()
{
if(base case)
            //something
      else
            //something
            recursiveFunc();      //recursive call
}
```

**program:**

```
#include<stdio.h>

int factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}

void main()
{
  int number;
  int fact;

  printf("Enter a number: ");
  scanf("%d", &number);

  fact = factorial(number);

  printf("Factorial of %d is %d\n", number, fact);

  return 0;
}
```

Output:

```
Enter a number: 6
Factorial of 5 is: 720
```

6 | 3 | 4

QP CODE 210393

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

| | | | | |
|---|---|---|---|---|
| d) | **Write a C program to compute transpose of a given matrix.**<br>`#include <stdio.h>`<br><br>`int main()`<br>`{`<br>`  int a[10][10], transpose[10][10], r, c;`<br><br>`  printf("Enter rows and columns: ");`<br>`  scanf("%d %d", &r, &c);`<br><br>`  // asssigning elements to the matrix`<br>`  printf("\nEnter matrix elements:\n");`<br>`  for (int i = 0; i < r; ++i)`<br>`  for (int j = 0; j < c; ++j)`<br>`  {`<br>`    printf("Enter element a%d%d: ", i + 1, j + 1);`<br>`    scanf("%d", &a[i][j]);`<br>`  }`<br><br>`  // printing the matrix a[][]`<br>`  printf("\nEntered matrix: \n");`<br>`  for (int i = 0; i < r; ++i)`<br>`  for (int j = 0; j < c; ++j)`<br>`  {`<br>`    printf("%d  ", a[i][j]);`<br>`    if (j == c - 1)`<br>`    printf("\n");`<br>`  }`<br><br>`  // computing the transpose`<br>`  for (int i = 0; i < r; ++i)`<br>`  for (int j = 0; j < c; ++j)`<br>`  {`<br>`    transpose[j][i] = a[i][j];`<br>`  }`<br><br>`  // printing the transpose`<br>`  printf("\nTranspose of the matrix:\n");`<br>`  for (int i = 0; i < c; ++i)`<br>`  {  for (int j = 0; j < r; ++j)`<br>`    {`<br>`      printf("%d  ", transpose[i][j]);`<br>`    }`<br>`      printf("\n");`<br>`  }`<br><br>`  return 0;`<br>`}` | 6 | 6 | 4 |

PILLAI COLLEGE OF ENGINEERING, NEW PANVEL
(Autonomous) (Accredited 'A+' by NAAC)

END SEMESTER EXAMINATION
SECOND HALF 2021(Supplementary)
BRANCH: FE (COMP / IT / ECS)

```
OUTPUT :

Enter rows and columns: 2
3

Enter matrix elements:
Enter element a11: 1
Enter element a12: 4
Enter element a13: 0
Enter element a21: -5
Enter element a22: 2
Enter element a23: 7

Entered matrix:
1  4  0
-5  2  7

Transpose of the matrix:
1  -5
4  2
0  7
```

**CO1:** Understand the basic terminology used in computer programming.

**CO2:** Use different data types, operators and keywords to write programs.

**CO3:** Able to logically code using control statements and loops.

**CO4:** Use the concepts of arrays, strings, functions and Structures to structure complex programs.

**CO5:** Use of pointers to access different user defined data types like arrays, Strings and Structures.

**CO6:** Use different data structures and open/create/update basic data files.

**BT Levels**: - 1 Remembering ,2 Understanding, 3 Applying,4 Analyzing, 5 Evaluating, 6 Creating.

M-Marks, **BT**- Bloom's Taxonomy, **CO**-Course Outcomes.