Reg. No. : | | | | | | | | | | | | |

07/06/18

Question Paper Code : 40027

B.E./B.Tech. DEGREE EXAMINATION, APRIL/MAY 2018

Second Semester

Computer Science and Engineering

CS 8251 – PROGRAMMING IN C

(Common to : Computer and Communication Engineering/ B.Tech. Information Technology)

(Regulations 2017)

Time : Three Hours | Maximum : 100 Marks

Answer ALL questions

PART – A (10×2=20 Marks)

1. What is external storage class ?
2. How does a preprocessor work ?
3. What is an array ? Write the syntax for multi-dimensional array.
4. Design a C program for compare any two string.
5. List the advantages of recursion.
6. When null pointer is used ?
7. State the meaning of the root word struct.
8. Specify the use of typedef.
9. How can you restore a redirected standard stream ?
10. What does argv and argc indicate in command-line arguments ?

PART – B (5×13=65 Marks)

11. a) i) Explain the different types of operators used in C with necessary program. (8)

ii) Write a C program to check the integer is Palindrome or not. (5)

(OR)

b) Describe the decision making statements and looping statements in C with an example. (13)

12. a) Write the C program to multiply two matrices (two-dimensional array) which will be entered by a user. The user will enter the order of a matrix and then its elements and similarly input the second matrix. If the entered orders of two matrices are such that they can't be multiplied by each other, then an error message is displayed on the screen. **(13)**

(OR)

b) i) What are the different types of string function ? Describe with their purpose. **(5)**

ii) Write the C program to find the number of Vowels, Consonants, Digits and white space in a string. **(8)**

13. a) i) Explain the purpose of a function prototype. And specify the difference between user defined function and built-in functions. **(8)**

ii) Write the C program to find the value of sin(x) using the series up to the given accuracy (without using user defined function) also print sin(x) using library function. **(5)**

(OR)

b) What is difference between pass by value and pass by reference ? Write the C coding for swapping two numbers using pass by reference. **(13)**

14. a) Define structure in C. Also specify the pointer and structure with example. **(13)**

(OR)

b) i) Write a C program for accessing structure member through pointer using dynamic memory allocation. **(6)**

ii) Write a short note on singly linked list and specify how the node are created in singly linked list. **(7)**

15. a) Explain the types of file processing with necessary examples. **(13)**

(OR)

b) Write the C coding for finding the average of number stored in sequential access file. **(13)**

PART – C

(1×15=15 Marks)

16. i) Write the case study of "How sequential Access file is differ from Random Access file". **(10)**

ii) Write a C program to write all the members of an array of structures to a file using fwrite (). Read the array from the file and display on the screen. **(5)**

40027

B7002 4.2
1+1

APPROVED / NOT APPROVED
QP
KEY

B.E/ B.Tech DEGREE EXAMINATIONS, APRIL/MAY 2018

COMPUTER SCIENCE & ENGINEERING/ COMPUTER & COMMUNICATION ENGINEERING/ INFORMATION TECHNOLOGY

SEMESTER 2

CS8251 – PROGRAMMING IN C

(Regulation 2017)

Time: 3 Hours **Answer Key** Max. Marks 100

PART-A (10 x 2 = 20 Marks)

1. A variable declared with the **extern storage-class** specifier is a reference to a variable with the same name defined at the **external** level in any of the source files of the program. The internal **extern** declaration is used to make the **external**-level variable definition visible within the block.
2. The C preprocessor is a macro processor that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define macros, which are brief abbreviations for longer constructs
3. An array is a collection of data items, all of the same type, accessed using a common name. A one-dimensional array is like a list; A two dimensional arrayis like a table; The C language places no limits on the number of dimensions in an array, though specific implementations may.

```
data_type  array_name[size1][size2]....[sizeN];
```

data_type: Type of data to be stored in the array.
Here data_type is valid C/C++ data type
array_name: Name of the array
size1, size2,... ,sizeN: Sizes of the dimensions

4. **String Compare**

```
int main()
{
  char a[100], b[100];
  printf("Enter a string\n");
  gets(a);
  printf("Enter a string\n");
  gets(b);
  if (strcmp(a,b) == 0)
    printf("The strings are equal.\n");
  else
    printf("The strings are not equal.\n");
  return 0;
}
```

5. **Advantages**
   - Reduce unnecessary calling of function.
   - Through Recursion one can Solve problems in easy way while its iterative solution is very big and complex.
6. **The null pointer is used in three ways:**
   - To stop indirection in a recursive data structure.
   - As an error value.
   - As a sentinel value.
7. **"Struct"** The **root** word **struct means** "Build" and this **root** word has many prefixes. Knowing this **root** word will allow you to find out what many words **mean**. Such as, destruct, construct, structure, instruct, obstruct, and instructor. Destruct **means** "The act of destroying something that was built."

8. The purpose of typedef is to assign alternative names to existing types, most often those whose standard declaration is cumbersome, potentially confusing, or likely to vary from one implementation to another.
9. Using the standard C library functions named dup() and fdopen(), you can restore a standard stream such as stdout to its original state.
   - The dup() function duplicates a file handle. You can use the dup() function to save the file handle corresponding to the stdout standard stream.
   - The fdopen() function opens a stream that has been duplicated with the dup() function.
10. argc is the number of command line arguments and argv is array of strings representing command line arguments. This gives you the option to react to thearguments passed to the program. If you are expecting none. you might as well use int main

PART – B (5 x 16 = 80 Marks)

11. A(i)Types Of Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators Example

```
int main()
{
   int a = 9,b = 4, c;
   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c=a/b;
   printf("a/b = %d \n",c);
   c=a%b;
   printf("Remainder when a divided by b = %d \n",c);
   return 0;
}
```

Relational Operators

```
int main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d = %d \n", a, b, a == b); // true
   printf("%d == %d = %d \n", a, c, a == c); // false
   printf("%d > %d = %d \n", a, b, a > b); //false
   printf("%d > %d = %d \n", a, c, a > c); //false
   printf("%d < %d = %d \n", a, b, a < b); //false
```

```
    printf("%d < %d = %d \n", a, c, a < c); //tru
    printf("%d != %d = %d \n", a, b, a != b); //false
    printf("%d != %d = %d \n", a, c, a != c); //true
    printf("%d >= %d = %d \n", a, b, a >= b); //true
    printf("%d >= %d = %d \n", a, c, a >= c); //false
    printf("%d <= %d = %d \n", a, b, a <= b); //true
    printf("%d <= %d = %d \n", a, c, a <= c); //true
    return 0;

}
```

Logical Operators

```
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) equals to %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) equals to %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) equals to %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) equals to %d \n", result);

   result = !(a != b);
   printf("!(a == b) equals to %d \n", result);

   result = !(a == b);
   printf("!(a == b) equals to %d \n", result);

   return 0;
}
```

(ii) Program to Check Palindrome

```
#include <stdio.h>
int main()
{
   int n, reversedInteger = 0, remainder, originalInteger;

   printf("Enter an integer: ");
   scanf("%d", &n);

   originalInteger = n;
```

```
// reversed integer is stored in variable
while( n!=0 )
{
    remainder = n%10;
    reversedInteger = reversedInteger*10 + remainder;
    n /= 10;
}

// palindrome if orignalInteger and reversedInteger are equal
if (originalInteger == reversedInteger)
    printf("%d is a palindrome.", originalInteger);
else
    printf("%d is not a palindrome.", originalInteger);

return 0;
}
```

(OR)

B(i) FOUR different ways to take decisions which are as follows :

- if Statement
- if-else Statement
- Conditional Operators
- Switch Statement

Program :

```
#include main()
{ int var1=5, var2=20;
if(var1 > var2)
{
printf(“Variable 1 is bigger ”); }
else
{ printf("Variable 1 is smaller"); } }
```

Program Switch :

```
#include main()
{
int marks1, marks2, marks3, total, average;
printf("Enter marks for three subjects: ");
scanf("%d%d%d",&marksl,&marks2,&marks3);
total = marksl+marks2+marks3;
average = total/3;
switch (average/10)
{
case 10:
case 9:
case 8: printf("Distinction");
break;
case 7:
case 6:
printf("First Division");
break;
case 5: printf("Second Division");
```

```
        break;
        case 4: printf("Pass");
        break;
        default: printf("Fail"); } }
```

Types of Loops.

There are three type of Loops available in 'C' programming language.

- while loop
- for loop
- do..while

While loop example.

```
        void main()
        {
        int i;
        clrscr();
        i=1;
        while(i<5)
        {
        printf("\n%d",i);
        i++;
        }
        getch();
        }
```

For loop example:

```
        void main()
        {
        int i;
        clrscr();
        for(i=1;i<5;i++)
        {
        printf("\n%d",i);
        }
        getch();
        }
```

Do-While loop Example:

```
        void main()
        {
        int i;
        clrscr();
        i=1;
        do
        {
        printf("\n%d",i);
        i++;
        }
        while(i<5);
        getch();
        }
```

12. A.(i) **Matrix multiplication**

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

```
int main()
{
 int m, n, p, q, c, d, k, sum = 0;
 int first[10][10], second[10][10], multiply[10][10];
 printf("Enter number of rows and columns of first matrix\n");
 scanf("%d%d", &m, &n);
 printf("Enter elements of first matrix\n");
 for (c = 0; c < m; c++)
  for (d = 0; d < n; d++)
   scanf("%d", &first[c][d]);
printf("Enter number of rows and columns of second matrix\n");
scanf("%d%d", &p, &q);

if (n != p)
 printf("The matrices can't be multiplied with each other.\n");
else
{
 printf("Enter elements of second matrix\n");

 for (c = 0; c < p; c++)
  for (d = 0; d < q; d++)
   scanf("%d", &second[c][d]);

 for (c = 0; c < m; c++) {
  for (d = 0; d < q; d++) {
   for (k = 0; k < p; k++) {
    sum = sum + first[c][k]*second[k][d];
   }
   multiply[c][d] = sum;
   sum = 0;
  }
 }
 printf("Product of the matrices:\n");

 for (c = 0; c < m; c++) {
  for (d = 0; d < q; d++)
   printf("%d\t", multiply[c][d]);
  printf("\n");
 }
}
return 0;}
```

(OR)

B(i) **Types of string function**

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.

4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

(ii) C program to find the number of Vowels, Consonants, Digits and white space in a String.

```
#include <stdio.h>
int main()
{
   char line[150];
   int i, vowels, consonants, digits, spaces;
   vowels =  consonants = digits = spaces = 0;
   printf("Enter a line of string: ");
   scanf("%[^\n]", line);
   for(i=0; line[i]!='\0'; ++i)
   {
      if(line[i]=='a' || line[i]=='e' || line[i]=='i' ||
        line[i]=='o' || line[i]=='u' || line[i]=='A' ||
        line[i]=='E' || line[i]=='I' || line[i]=='O' ||
        line[i]=='U')
      {
        ++vowels;
      }
      else if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
      {
        ++consonants;
      }
      else if(line[i]>='0' && line[i]<='9')
      {
        ++digits;
      }
      else if (line[i]==' ')
      {
        ++spaces;
      }}
    printf("Vowels: %d",vowels);
    printf("\nConsonants: %d",consonants);
    printf("\nDigits: %d",digits);
    printf("\nWhite spaces: %d", spaces);
    return 0;
}
```

13. A.(i) **What is the purpose of a function prototype?**

- The Function prototype serves the following purposes –

1) It tells the return type of the data that the function will return.
2) It tells the number of arguments passed to the function.
3) It tells the data types of the each of the passed arguments.
4) Also it tells the order in which the arguments are passed to the function.

- Therefore essentially, function prototype specifies the input/output interlace to the function i.e. what to give to the function and what to expect from the function.
- Prototype of a function is also called signature of the function.

What if one doesn't specify the function prototype?

Output of below kind of programs is generally asked at many places.

```
int main()
{
  foo();
  getchar();
  return 0;
}
void foo()
{
  printf("foo called");
}
```

Built-in Functions

C has many built-in functions that you can use in your programs. So far we have learned 15 built-in functions:

main( )	gets( )	strlen( )	strcat( )	isdigit( )
printf( )	puts( )	strcmp( )	strstr( )	isupper( )
scanf( )	strcpy( )	stricmp( )	isalpha( )	islower( )

User-Defined Functions

If you have a special set of instructions that aren't in a built-in function, you can create a user-defined function. Here are the steps:

1. give your function a name that isn't already used in C (by built-in functions, types of variables, keywords, etc.)
2. create a **function header**, which contains three things:
   - the type of variable (int, char, double, etc.) that the function will produce (return)
   - the name of the function, which can be one or more words (but put underscores _ or CapitalLetters connecting these words, because no spaces are allowed)
   - the parameters of the function, which are the names and types of variables inside your function
3. create a **function body**, which contains the operations to be completed when you call the function to run

(ii) Write the c program to find the value of sin(x) using the series up to the given accuracy (without using user defined function) Also print sin(x) using library function

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
void main()
{
int n, x1;
float  acc, term, den, x, sinx=0, sinval;
clrscr();
printf("Enter the value of x (in degrees)\n");
scanf("%f",&x);
x1 = x;
/* Converting degrees to radians*/
```

```
x = x*(3.142/180.0);
sinval = sin(x);
printf("Enter the accuary for the result\n");
scanf("%f", &acc);
term = x;
sinx = term;
n = 1;
do
{
 den = 2*n*(2*n+1);
 term = -term * x * x / den;
 sinx = sinx + term;
 n = n + 1;
} while(acc <= fabs(sinval - sinx));
printf("Sum of the sine series        = %f\n", sinx);
printf("Using Library function sin(%d) = %f\n", x1,sin(x));
}
```

(OR)

B(i) Difference between Call by Value and Call by Reference

call by value	**call by reference**
In *call by value*, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.	In *call by reference*, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.
In call by value, actual arguments will remain safe, they cannot be modified accidentally.	In *call by reference*, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.

C Program to swap two numbers using pass by reference

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

void swap(int *num1, int *num2);

void main() {
  int x, y;

  printf("\nEnter First number : ");
  scanf("%d", &x);

  printf("\nEnter Second number : ");
  scanf("%d", &y);

printf("\nBefore Swaping x = %d and y = %d", x, y);
swap(&x, &y); // Function Call - Pass By Reference
```

```
  printf("\nAfter Swaping x = %d and y = %d", x, y);
  getch();
}

void swap(int *num1, int *num2) {
  int temp;
  temp = *num1;
  *num1 = *num2;
  *num2 = temp;
}
```

14. A(i) Structure in C

- A **struct** in the C programming language (and many derivatives) is a composite data type (or record) declaration that defines a physically grouped list of variables to be placed under one name in a block of memory, allowing the different variables to be accessed via a single pointer, or the struct declared name which returns the same address.
- The struct can contain many other complex and simple data types in an association, so is a natural organizing type for records like the mixed data types in lists of directory entries reading a hard drive (file length, name, extension, physical (cylinder, disk, head indexes) address, etc.), or other mixed record type (patient names, address, telephone... insurance codes, balance, etc.).
- **Syntax**

```
struct tag_name {
  type member1;
  type member2;
  /* declare as many members as desired, but the entire structure size must be known to the compiler.
*/
};
```

Pointers to struct

- Pointers can be used to refer to a struct by its address. This is particularly useful for passing structs to a function by reference or to refer to another instance of the struct type as a field. The pointer can be dereferenced just like any other pointer in C, using the * operator. There is also a -> operator in C which dereferences the pointer to struct (left operand) and then accesses the value of a member of the struct (right operand).

```
struct point {
  int x;
  int y;
};
struct point my_point = { 3, 7 };
struct point *p = &my_point;  /* To declare and define p as a pointer of type struct point,
                                 and initialize it with the address of my_point. */

(*p).x = 8;          /* To access the first member of the struct */
p->x = 8;            /* Another way to access the first member of the struct */
```

- C does not allow recursive declaration of struct; a struct can not contain a field that has the type of the struct itself. But pointers can be used to refer to an instance of it:

```
typedef struct list_element list_element;
struct list_element {
  point p;
  list_element * next;
};
list_element el = { .p = { .x = 3, .y =7 }, };
list_element lc = { .p = { .x = 4, .y =5 }, .next = &el };
```

(OR)

B(i)

```
#include <stdio.h>
#include <stdlib.h>
struct person {
  int age;
  float weight;
  char name[30];
};

int main()
{
  struct person *ptr;
  int i, num;

  printf("Enter number of persons: ");
  scanf("%d", &num);

  ptr = (struct person*) malloc(num * sizeof(struct person));
  // Above statement allocates the memory for n structures with pointer personPtr pointing to base address */

  for(i = 0; i < num; ++i)
  {
    printf("Enter name, age and weight of the person respectively:\n");
    scanf("%s%d%f", &(ptr+i)->name, &(ptr+i)->age, &(ptr+i)->weight);
  }

  printf("Displaying Infromation:\n");
  for(i = 0; i < num; ++i)
    printf("%s\t%d\t%.2f\n", (ptr+i)->name, (ptr+i)->age, (ptr+i)->weight);

  return 0;
}
```

(ii) **Singly Linked List**

- A **linked list** is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a **node**.
- A node is a collection of two sub-elements or parts. A **data** part that stores the element and a **next** part that stores the link to the next node.

Declaring a Linked list :

- In C language, a linked list can be implemented using structure and pointers .

```
struct LinkedList{
  int data;
  struct LinkedList *next;
};
```

Creating a Node:

- Let's define a data type of struct LinkedListto make code cleaner.

```
typedef struct LinkedList *node; //Define node as pointer of data type struct LinkedList

node createNode(){
```

```
    node temp; // declare a node
    temp = (node)malloc(sizeof(struct LinkedList)); // allocate memory using malloc()
    temp->next = NULL;// make next point to NULL
    return temp;//return the new node
}
```

- typedef is used to define a data type in C.
- malloc() is used to dynamically allocate a single block of memory in C, it is available in the header file stdlib.h.
- sizeof() is used to determine size in bytes of an element in C. Here it is used to determine size of each node and sent as a parameter to malloc.

15 . A.(i) File Input/Output in C

- A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure

Opening a File or Creating a File

- The fopen() function is used to create a new file or to open an existing file.
- **General Syntax:**

  *fp = FILE *fopen(const char *filename, const char *mode);
- Here, *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.
- filename is the name of the file to be opened and mode specifies the purpose of opening the file.

Closing a File

- The fclose() function is used to close an already opened file.
- **General Syntax :**

  int fclose(FILE *fp);
- Here fclose() function closes the file and returns zero on success, or EOF if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.

(OR)

B(i) finding the average of number stored in sequential access file

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

int main(void)
{
int value, total = 0, count = 0;
FILE * fileptrIn, * fileptrOut;
errno_t err = 0, err1 = 0;
char filenameIn[50], filenameOut[50];

printf("Please enter an input filename (use path if needed):\n");
scanf_s("%s", filenameIn, 50);
printf("Please enter an output filename (use path if needed):\n");
scanf_s("%s", filenameOut, 50);
err = fopen_s(&fileptrIn, filenameIn,"r");
err1 = fopen_s(&fileptrOut, filenameOut,"w");
```

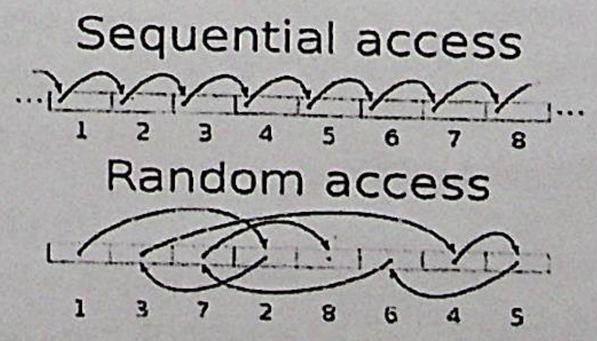
```
if(err !=0 )
{
printf("\nError opening %s for reading.\n", filenameIn);
exit (1);
}
else
printf("\nOpening %s for reading is OK.\n", filenameIn);
if(err1 != 0)
{
printf("Error opening %s for writing.\n", filenameOut);
exit (1);
}
else
printf("Opening %s for writing is OK.\n", filenameOut);
printf("\nThe data:\n");
while(EOF != fscanf_s(fileptrIn, "%i", &value))
{
printf("%d ", value);
total += value;
++count;
}
printf("\nThe total: %d\n", total);
printf("\n\nCalculate the average...\n");
fprintf(fileptrOut, "Average of %i numbers = %f \n", count, total/(double)count);
printf("Average of %i numbers = %f \n\n", count, total/(double)count);
printf("Check also your %s file content\n", filenameOut);

if(fclose(fileptrIn) == 0)
printf("%s closed successfully\n", filenameIn);
if(fclose(fileptrOut) == 0)
printf("%s closed successfully\n", filenameOut);
return 0;
}
```

PART – C (1 x 15 = 15 Marks)

16.A.(i) **Difference between Sequential and Random Access operations**

- Comparing **random** versus **sequential operations** is one way of **assessing application efficiency** in terms of disk use.
- Accessing data sequentially is much faster than accessing it randomly because of the way in which the disk hardware works. The **seek operation,** which occurs when the disk head positions itself at the right disk cylinder to access data requested, takes more time than any other part of the I/O process.
- Because reading randomly involves a higher number of seek operations than does sequential reading, random reads deliver a lower rate of throughput. The same is true for random writing.



You might find it useful to examine your workload to determine whether it accesses data randomly or sequentially. If you find disk access is predominantly random, you might want to pay particular attention to the activities being done and monitor for the emergence of a bottleneck.

(ii)

```
#include <stdio.h>
struct student
{
  char name[50];
  int height;
};
int main(){
   struct student stud1[5], stud2[5];
   FILE *fptr;
   int i;

   fptr = fopen("file.txt","wb");
   for(i = 0; i < 5; ++i)
   {
      fflush(stdin);
      printf("Enter name: ");
      gets(stud1[i].name);

      printf("Enter height: ");
      scanf("%d", &stud1[i].height);
   }

fwrite(stud1, sizeof(stud1), 1, fptr);
fclose(fptr);

fptr = fopen("file.txt", "rb");
fread(stud2, sizeof(stud2), 1, fptr);
for(i = 0; i < 5; ++i)
{
   printf("Name: %s\nHeight: %d", stud2[i].name, stud2[i].height);
}
fclose(fptr);}
```

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 80093

B.E./B.Tech. DEGREE EXAMINATIONS, APRIL/MAY 2019.

Second Semester

Computer Science and Engineering

CS 8251 — PROGRAMMING IN C

(Common to Computer and Communication Engineering/Information Technology)

(Regulation 2017)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Differentiate between formatted and unformatted input statements. Give one example for each.
2. What is the use of preprocessor directive?
3. Define an array.
4. Write a C function to compare two strings.
5. What is the need for functions?
6. What is the output of the following code fragment?

```
int x= 456, *p1, **p2;
p1=&x; p2=&p1;
printf ("Value of x is : %d\n", x);
printf("Value of *p1 is : %d\n", *p1);
printf ("Value of *p2 is : %d\n", *p2);
```

7. Compare and contrast a structure with an array.
8. What is the output of the following code fragment?

```
struct point
{
int x;
int y;
};
struct point origin, *pp;
main ( )
{
pp = & origin;
printf (" origin is (%d% d)\n", (*pp).x,pp → y);
}
```

9. Why files are needed?
10. What is the use of command line argument?

PART B — (5 × 16 = 80 marks)

11. (a) (i) What is the purpose of a looping statement? Explain in detail the operation of various looping statements in C with suitable examples. (12)

(ii) Write a C program to find the sum of 10 non-negative numbers entered by the user. (4)

Or

(b) (i) What is a storage class? Explain the various storage classes in C along with suitable example. (12)

(ii) Write a C program to find the largest among 3 numbers entered by the user. (4)

12. (a) Explain binary search procedure. Write a C program to perform binary search and explain. (16)

Or

(b) Discuss how you can evaluate the mean, median, mode for an array of numbers. Write the C program to evaluate the mean, median and mode for an array of numbers and explain. (16)

13. (a) What is recursion? Explain the procedure to compute $\sin(x)$ using recursive functions. Write the C code for the same. (16)

Or

(b) What is pass by reference? Explain swapping of 2 values using pass by reference in 'C'. (16)

14. (a) What is dynamic memory allocation? Explain various C functions that are used for the same with examples. (16)

Or

(b) What is a self-referential structures? Explain with suitable examples. (16)

15. (a) Explain in detail various operations that can be done on file giving suitable examples. (16)

Or

(b) Explain in detail random access in files along with the functions used for the same in C. Give suitable examples. (16)

———————