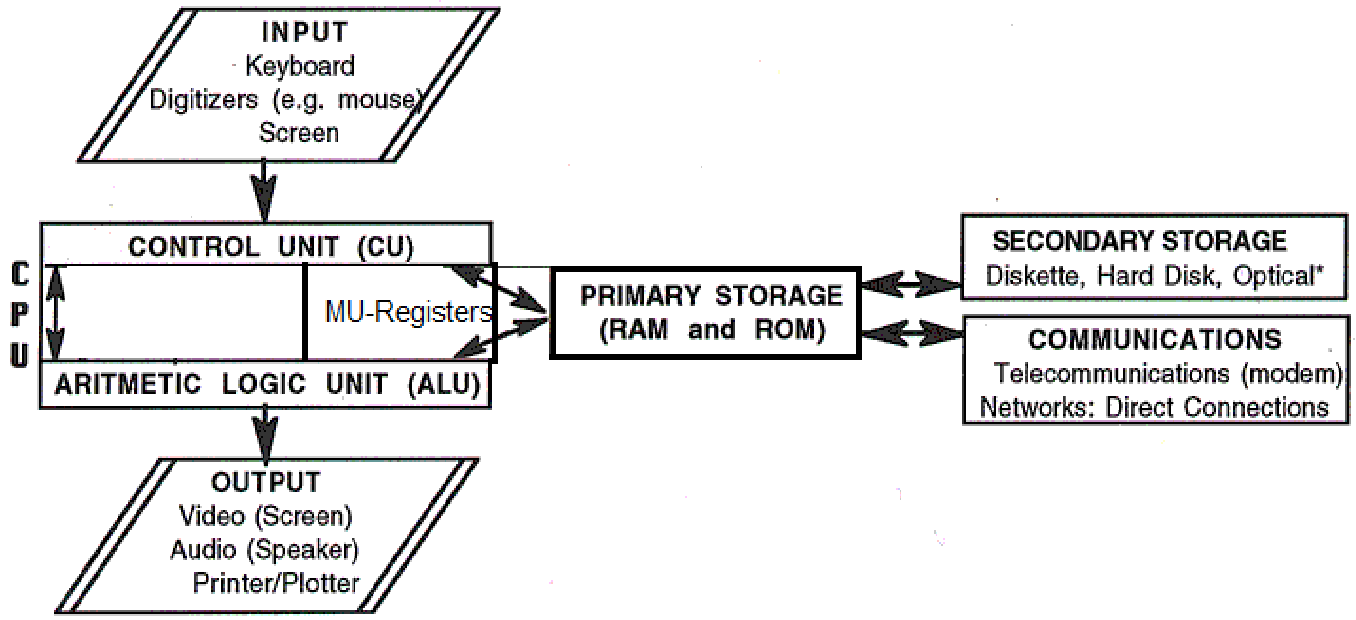
Model Answer/Solution of VTU Question-Jun/Jul 2019
C Programming For Problem Solving (18CPS23)-2nd Sem BE
Dr. P. N. Singh, Professor(CSE)

Q1a. With a neat block diagram of computer explain its components. 10M

Ans:



Input unit: The input unit that links the external environment to input data & tasks with the computer system to execute. Data are entered in different forms through different input devices. Keyboard is used for characters input. Mouse is used in GUI (Graphic User Interface). Internally data is processed in machine readable form.

Output Unit: Output/result is displayed, printed & transmitted to outside world. There are many output devices: monitor, printer/plotters, display boards, speaker etc.

Storage unit: The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. The storage unit is Primary Memory (RAM) & Secondary (permanent storage devices: disks, tapes)

CPU (Central processing Unit): It is the main unit which controls all events within computer. The CPU has 3 internal units: CU, ALU & Registers:
CU(Control unit): By selecting, interpreting, and seeing to the execution of the program instructions, the control unit is able to maintain order and directs the operation of the entire system. the control unit acts as a central nervous system for the other components of the computer. It manages and coordinates the entire computer system. It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.

ALU (Arithmatic & Logic Unit): The arithmetic and logic unit (ALU) is the part where actual computations take place. It consists of circuits that perform arithmetic operations (e.g. addition, subtraction, multiplication, division over data received from memory and capable to compare numbers (less than, equal to, or greater than).

MU (Memory Unit/Registers): Registers are built-in memory with CPU having less storage space in bits. Registers are a group of cells used for memory addressing, data manipulation and processing. Instruction Registers, Address registers, Program Counters, Accumulators are example of registers. ALU takes data from here inside the CPU.

RAM(Random Access Memory): RAM is the memory - primary storage where our data & programs are stored temporarily. It is volatile in nature. After switching off the system everything will be vanished from RAM.

ROM(Read Only Memory): ROM is storage medium/"firmware" where some code of manufacturer is permanently hardwired in chip which always executes automatically when we start the system. The process is known as POST(Power on Self test). Booting preceeds POST.

1B. Classify the following into input & output devices: 5 M

Monitors, Visual Display Unit, Track balls, Bar Code Reader, Digital Camera, Film Recorder, Microfiche, OMR, Electronic White Board, Plotters

Ans:

Input Devices	**Output devices**
Track Balls	Monitors
Bar Code Reader	Visual Display Unit
Digital Camera	Plotter
OMR (Optical Mark Reader)	Electronic White Board
	Film Recorder
	Microfiche

1c. Define the terms Network, LAN, WAN, MAN and Network Topology 5 M

Ans:

Network: Collection of devices/computers connected together is called a Network.

LAN(Local Area Network): A group of computers & devices connected together, usually within the same building or in same campus is LAN.

MAN(Metropolitan Area Network): MAN is a larger network that usually spans several buildings in the same city or town. The Dish TV network is an example of a MAN.

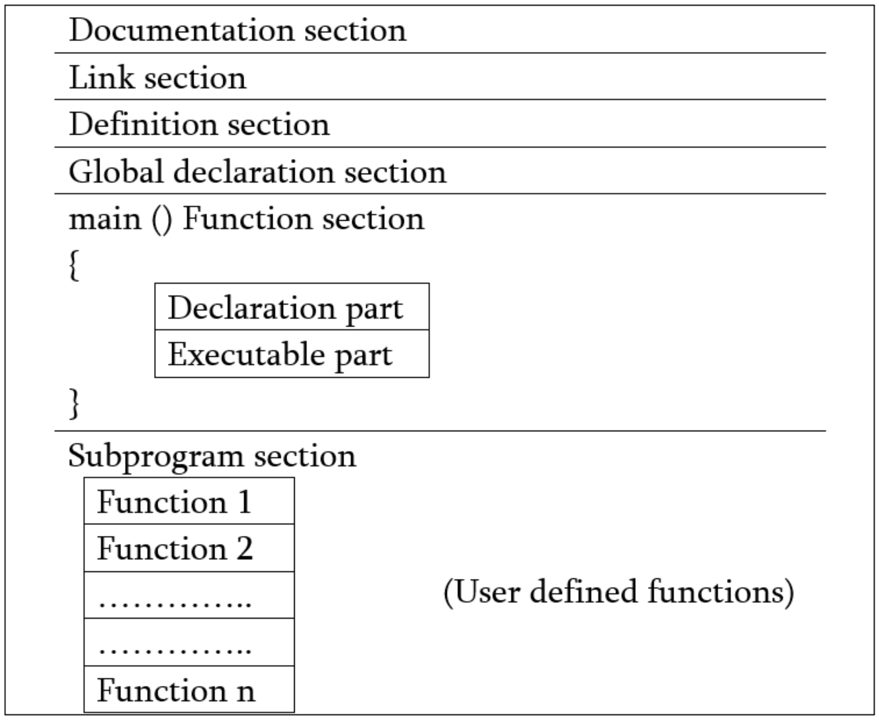
WAN(Wide Area Network): A **WAN** (wide area network), in comparison to a MAN, is not restricted to a geographical location, although it might be confined within the bounds of a state or country. A WAN connects several LANs, and may be limited to an enterprise (a corporation or an organization) or accessible to the public. The technology is high speed and relatively expensive.

Network Topology: Network topology is topological structure of a network where nodes may be connected in different ways. Examples of Network Topologies are : Bus Topology, Ring Topology, Star Topology & Mesh topology.

2a. Write the basic structure of a C Program. Explain each section briefly with suitable examples. 9 M

Ans:

Structure of a C program



- **Documentation Section:** This section is used to write Problem, file name, developer, date etc in comment lines within /*….*/ or separate line comments may start with // . Compiler ignores this section. Documentation enhances the readability of a program.
- **Link section :** To include header and library files whose in-built functions are to be used. Linker also required these files to build a program executable. Files are included with directive # include
- **Definition section:** To define macros and symbolic constants by preprocessor directive #define
- **Global section:** to declare global variables – to be accessed by all functions
- **main() i**s the user defined function which is recognized by the compiler first. So, all C program must have user defined function main() { ………… }. It should have declaration part first then executable part.
- **Sub program section:** There may be other user defined functions to perform specific task when called.

```
/* Example: a program to find area of a circle – area.c
            Dr. P. N. Singh              - Documentation Section*/
#include <stdio.h>                      /* - Link/Header Section */
#define PI 3.14                    /* definition/global section*/
int main()                         /* main function section */
{
     float r, area;               /* declaration part */
     printf(“Enter radius of the circle : “); /* Execution part*/
     scanf(“%f”, &r);
     area=PI*r*r; /* using symbolic constant PI */
          printf(“Area of circle = %0.3f square unit\n”, area);
     return (0);
}
```

2b. Define Operator. Explain any 6 operators with suitable examples. 7 M

Ans:

i. Arithmetic Operators :

+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus) returns Remainder: 10%3 will return 1

ii. Relational Operators:

< (less than)
<= (less than or equal to)
> (greater than)
>= (greater than or equal to)
==(equal to)
!=(not equal to)

iii. Logical Operators:

&& (and) Both of the conditions must be satisfied for true.
|| (or) Either of the condition should be satisfied for true
! (Not) Negation of true is false & vice-versa

iv. Increment & Decrement:

++, -- (c++ is equivalent to c = c+1)

v. Assignment Operator :

=, +=, -=, */, /=, %=

vi. Pre-processor Operator : #

vii. Member operator: . and ->

viii. Bitwise Operators : & (Bit-wise AND), | (Bit-wise OR), ! (Bit-wise NOT), ^ (Bit-wise XOR),
>> (Bit-wise right shift), <<(Bit-wise left shift)

ix. Ternary Operator(Conditional if): ? :

x. Address of operator: &

xi. Pointer (dereference) operator: *

xii. Comma Operator: ,

xiii. Statement terminator operator: ;

Examples of Bit-wise operators & Ternary operators

& - Bit-wise AND : 1 only when both inputs are 1
printf(“%d”, 20 & 25); /*10100 & 11001 = 10000 will give output 16*/
| - Bit-wise or : 1 when any of the input is 1. It gives 0 when both inputs are 0
printf(“%d”, 20 | 25); /*10100 | 11001 = 11101 will give output 29*/
^ - Bit wise XOR (Exclusive or): 1 when only one input is 1. If both are 1 or 0, it gives 0
printf(“%d”, 20 ^ 25); /*10100 ^ 11001 = 01101 will give output 13*/
~ - Bit-wise compliment : Difference in highest & given. In Binary 1 will be 0 and 0 will be 1. If unsigned int has 2 bytes (16 bits) size with highest value 65535 then
printf(“%u”, ~ 5); /*~ 0000000000000101 = 1111111111111010 will give output 65530*/
>> - Bit wise right shift : Halves the value in integer
printf(“%u”, 10>>2); /*1010 will 101 in first right shift & again 10 in 2nd right shift so 2 */
<< - Bit wise left shift : doubles the value(within range)
printf(“%u”, 11<<2); /*1011 will 10110 in first left shift & again 101100 in 2nd left shift so 44 */

Ternary Operator(Conditional if): ? :
Syntax:
<condtion> ? <statement for true> : < statement for false>
int salary = 50000;
Example1:
bonus = salary > 40000 ? 10000 : 20000;
Here salary is greater than 40000 so, bonus will be 20000

2c. State whether following are valid identifiers or not: integer, float. I am, 123_AbC
Ans:

A valid identifier may contain a-z, A-Z, 0-9 & _ (underscore). It must not start with a digit and it should nt be a keyword.

integer:	Valid
float:	In-valid (It is a keyword)
I am	Invalid(contains space/blank)
123_AbC	Invalid(starts with a digit)

3a. Define and write classification of input and output statements in C. Write a Program that prints the following output:

```
“ I am
  an”        ‘Engineering
  Student’
```
6 M

Ans:

Input Statements of C		Output Statements of C	
Formatted Input stmts	Unformatted Input stmts	Formatted Output stmts	Unformatted Output stmts
scanf( )	gets( )	printf( )	puts( )
fscanf( )	getchar( )	fprintf( )	putchar( )
sscanf( )	fgetc( )	sprint( )	fputc( )

Formatted input/output statements provides certain features that can be effectively exploited to control the alignment, spacing & width specification.
Unformatted input/output statements are limited with their built-in features only.
Formatted Input statements:
scanf(): to read values from keyboard (by default) as formatted input:
Syntax:

scanf (“control string”, &arg1, &arg2, ..., &argn);

The control string specifies the field format in which the data is to be entered and the arguments arg1,arg2.....,argn specify the address of locations where the data is stored. Actually arguments are pointers which indicate where the data items are stored in computer’s memory.
Control string contains field specification which direct the interpretation of input data. It may include:

- Field (or format) specification, consisting of the conversion character %, a data type character (or type specifier, and an optional number, specifying the **width (Maximum width Specification).**
- Blanks, tabs, or new lines.
- The arguments are written as variables or arrays, whose type match the corresponding character groups in the control string. Each variable name must be preceded by an ampersand (&). Array names should not begin with an ampersand.
  Examples:

scanf(" %d%s%d", &roll, name, &marks);
scanf("%2d%2d%2d",&marks1, &marks2, &marks3); /* only 2 digits will be scanned */

fscanf( ) : Fromatted data from files/standard input, one more argument for file pointer/standard input

f**scanf (fp, "control string", &arg1, &arg2, ..., &argn);**

sscanf( ) : Formatted data from a string, one more argument as pointer to string

f**scanf (char *ptr, "control string", &arg1, &arg2, ..., &argn);**

Unformatted input statements:

gets(): to read a line terminated by new line character. String with blanks can be read.

Syntax/example:

gets(str);

getchar( ): To read a single character from keyboard:

example:

char c;
getchar(c);

Formatted Output statements:

printf() – Library function for formatted output:

printf ("control string", arg1,arg2, ..., argn);

Control string of printf function consists of three types of item :

- Character that will be printed on the screen as they appear.
- Format specification/Minimum width specification that define the output format for display of each item.
- Escape sequence characters such as \n,\t, and \b.

Examples:

printf("%5d%20s%5d\n", roll, name, marks);

for roll minimum 5, for name minimum 20 & for marks minimum 5 characters space will be used & right aligned.

```
   65                Ankur   89
123451234567890123456789012345
                1        2
```

fprintf( ) : Foomatted data from files/standard input, one more argument for file pointer/standard input

f**scanf (fp, "control string", &arg1, &arg2, ..., &argn);**

sscanf( ) : Formatted data from a string, one more argument as pointer to string

f**scanf (char *ptr, "control string", &arg1, &arg2, ..., &argn);**

Unformatted input statements:

gets(): to read a line terminated by new line character. String with blanks can be read.

Syntax/example:

gets(str);

getchar( ): To read a single character from keyboard:

example:

char c;
getchar(c);

Q 3c. Evaluate:

```
i=1;
L: if(i>2)
  {
    printf("Saturday");
    i=i+1;
    goto L;
```

```
   }
   printf("Sunday");
```
Explain your result briefly. 4 M

Ans:
It will print Sunday.
L is a label followed by a colon for goto statement.
i=1; so if(i>2) is false & it will not go inside of if statement
Finally it will print Sunday (which is followed by if statement)

Q 4a. State the drawback of ladder if-else. Explain how do you resolve with suitable example. 8M
Ans:

Ladder if-else statement is multi-way decision statement
Syntax of ladder if-else statement:
```
if(condition1)
   statement(s);
     else if(condition2)
           statement(s);
         else if(condition3)
              statement(s);
               ……….
               else if(conditionn)
                       statement(s);
               else
                       statement(S);
```

if any of the condition1, condition2 & condition3 is evaluated true then corresponding statements are executed and the control comes out entire of the ladder if-else statement. If all conditions are false then last statement will be executed.

Drawback of ladder if-else statement:
a. Multiple if-else conditions are little tough to understand & check to modify for correct output.
b. As depth of ladder increases, readability of program decreases.
c. Each condition and decision may evolve expressions

Instead of this we can go for switch statement.
Syntax of switch statement;
```
      switch (expression)
      {        case  condition1
                        statement1;
                        statement2;
                        ……
                        break;
               case condition2
                        statement1;
                        statement2;
                        …….
                        break;
               …..
```

```
        default:
                statement1;
                statement2;
                .......
    }
```

Example:
Problem definition:
Bonus of employee in a company is based on grade as following:
Grade ‘a’ employees gets bonus equal to their salary
Grade ‘b’ and ‘c’ employees get bonus salary + 5000
Grade ‘d’ and ‘D’ employees get bonus salary + 10000
Other than these they get bonus salary + 15000.

```
/* Example of switch */
#include <stdio.h>
int main()
{
     int salary,bonus;
     char grade:
     printf("Enter grade : ");
     scanf("%c", &grade);
     printf("Enter salary : ");
     scanf("%d", &salary);
     switch (grade)
     {
           case 'a':
           case 'A':  bonus=salary;
                      break;
           case 'b':
           case 'B':
           case 'c':
           case 'C':  bonus=salary+5000;
                      break;
           case 'd':
           case 'D':  bonus=salary+10000;
        break;
            default :  bonus=salary+15000; /*lower grade-more bonus*/
     }
     print("Bonus = %d\n", bonus);
     return (0);
}
```

Q 4b. Write a C program to get triangle of numbers as a result:

```
1
1 2
1 2 3
1 2 3 4
```

Ans:

```
#include <stdio.h>
int main()
{
     int a,b;
          for(a=1;a<=4;a++)
          {
                for(b=1;b<=a;b++)
                     printf("%2d",b);
                     /* inner loop from 1 to a & prints b taking 2
                     space*/
                printf("\n");
                }
           return (0);
      }
```

Q 4c. Write a C program to check whether a given number is prime or not 6 M

Ans:

```
           #include <stdio.h>
           #include <math.h>
      int main()
      {
           int n,d, prime=1;
                printf("Enter a number (>1) : ");
                scanf("%d", &n);
                if(n<2)
                  { printf("Not a prime number\n"); return (99); }
                /* if number > 1 is not divisible from 2 to square
                root of number then it is prime*/
                for(d=2;d<=sqrt(n);d++)
                     if(n%d==0)
                     {
                     prime=0;  break;
                     }
                if(prime)
                     printf("Yes, %d is a prime number\n",n);
                else
                     printf("No, %d is not prime\n",n);

           return (0);
      }
```

```
Expected O/p:
Enter a number (>1) : 17
Yes, 17 is a prime number.
```

Q 5a. Define an array. Explain with suitable example how do you declare & initialize 1D array. 10 M

Ans:

An array is an identifier that refers to the collection of data items which all have the same name. The individual data items are represented by their corresponding array elements. Address of an element (subscript) ranges from 0 to n-1, where n is total number of elements.

Declaration of one-dimensional array:

syntax:

```
type variable[size];
```

Examples:

```
float height[50];
  int batch[10];
  char name[20];
```

Initialization of one-dimensional array may be done directly or indirectly running a loop.
Examples of direct initialization:

```
int marks[5] = {76,45,67,87,92};
static int count[ ] = {1,2,3,4,5};
static char name[5] = {‘S’, ‘I’, ‘N’, ‘G’, ‘H’, ‘\0’};
```

Example of indirect initialization:

```
int  a[10], x;
for(x=0; x<10;x++)
{
        printf(“Enter element %d : “, x+1);
        scanf(“%d”, &a[x]); /* reading values from keyboard for xth element */
}
```

Q 5b. Write a C program to search an element using linear & binary techniques. 10 M
Ans:

```
/* Linear & Binary  search in given array of N elements */
#include <stdio.h>

int main()
{
  int a[1000],x, y,temp,N, first, last, mid, skey, found;
  printf("Enter number of elements : ");
  scanf("%d", &N);
  for(x=0;x<N;x++)
  {
     printf("Enter element %d : ", x+1);
     scanf("%d",&a[x]);
  }
  printf("\nEnter search key integer for linear search : ");
  scanf("%d",&skey);

  for(x=0;x<N;x++)
    if(a[x] == skey)
    printf("Found at %dth Position\n",x+1);

  /* now sorting - because Binary search requires sorted elements */
  for(x=0;x<N;x++)
     for(y=0;y<N-x-1;y++)/* Bubble Sort */
       if(a[y] > a[y+1])
       {
          temp=a[y];
          a[y]=a[y+1];
          a[y+1]=temp;
       }
  printf("The sorted array:\n");
  for(x=0;x<N;x++)
     printf(" %d",a[x]);

  printf("\nEnter search key integer : ");
  scanf("%d",&skey);
  first=0;
  last=N-1;
  found=0;
  while(first <= last && ! found)
  {
     mid=(first+last)/2; /* integer value of mid */
     if(a[mid] > skey)
    last=mid-1;
     else if (a[mid] < skey)
    first=mid+1;
     else
    found=1;
  }
  if(found)
     printf("Found at %dth position\n",mid+1);
  else
     printf("Not found\n");

  return (0);
}
```

Expected Ouput:

```
Enter number of elements : 11
Enter element 1 : 44
Enter element 2 : 3
Enter element 3 : 55
Enter element 4 : 2
Enter element 5 : 11
Enter element 6 : 22
Enter element 7 : 33
Enter element 8 : 7
Enter element 9 : 8
Enter element 10 : 19
Enter element 11 : 27

Enter search key integer for linear search : 7
Found at 8th Position
The sorted array:
 2 3 7 8 11 19 22 27 33 44 55
Enter search key integer : 44
Found at 10th position
```

Q 6a. Define a string. Explain any 4 string library functions with syntax & example. (10 marks)

Ans:

A string is array of characters terminated with NULL '\0' character.
When a string is entered by default one NULL character is added at the end.

String declaration and initialization:

<u>Declaration:</u>

char str[size];
example:
char str[50];

<u>Initialization</u>

char name[ ]='SINGH';
char name[5]={'S','I','N','G',H','\0'};

String Library functions:

Header file: **string.h**

i. strlen(str) – Returns length of string in number of bytes.
Syntax:
size_t strlen (const char * str);
Example:
printf(“Length of string Dr. P. N. Singh is %d characters\n”, strlen(“Dr. P. N. Singh”));
Output:
Length of string Dr. P. N. Singh is 15 characters

ii. strev(str) – Reverses the string
Syntax:
char *strrev(char *str);
Example:
char str[] = “Kuchi Kuchi Rakma”
strrev(str);
printf(“%s”,str);
output:
amkaR ihcuK ihcuK

iii. strcat(str1, str2) – Concatenates(appends) str2 to str1 at the end.
Syntax:
char* strcat (char* strg1, const char* strg2);
Example:
If two strings are:
char name[]=”Roma”;
char surname[]=”Ritambhara”;
Then to append “Ritambhara” with name “Roma”
strcat(name,surname);
printf(“%s”, name);
Output: Roma Ritambhara

iv. strcmp(str1, str2) – Compares two string and returns:
0 if both strings are identical
else returns difference based on ASCII value of first mismatch.
Syntax:
int strcmp (const char* str1, const char* str2);
Example:
printf(“%d”, strcmp(“SINGH”, “SINHA”);
output: - 1

Q 6b. Write a C program to copy a string (combination of digits & alphabets) to another string (only alphabet)
10 M

Ans:
```
/* copy string only alphabets (not numbers) */
#include <stdio.h>
int main()
{
   char str1[50], str2[50];
   int x,y;
   printf("Enter alphanumeric string : ");
   gets(str1);
   y=0;
   for(x=0;str1[x]!=NULL;x++)
     if( !(str1[x]>='0' && str1[x] <= '9')) /* non-numeric */
       {
          str2[y]=str1[x];
          y++;
       }
   str2[y]=NULL; /* last character of copied string */
   printf("Copied string(only alphabets) %s\n",str2);
   return (0);
}
```

Expected O/P 1:
Enter alphanumeric string : Pay me 465 rupees
Copied string(only alphabets) Pay me rupees
Expected O/P 2:
Enter alphanumeric string : 1CR18CS001
Copied string(only alphabets) CRCS

Q 7a. Define a function. List & explain the categories of user defined functions. 10 M
Ans:

"A function may or may not have arguments"
"A function may or may not return a value"

i. Function without argument & does not return a value
ii. Function without argument & returns a value
iii. Function with argument & does not return a value
iv. Function with argument & returns a value
v. Recursive functions

i. Function without argument & does not return a value

```
void gao()
{
   printf("Kudi punjaban dil chura ke lay gayee – sona sona – dil mera sona\n");
}
```

ii. Function without argument & returns a value

```
int houserent()
{
    long int salary, hra;
    printf("Enter Salary : ");
    scanf("%ld", &salary);
    hra=salary*30/100;   /* hra 30 % of salary */
    return hra;
}
```

iii. Function with argument & does not return a value

```
    void houserent(long int salary)
    {
            long hra;
             if(salary > 50000)
```

```
                    hra=salary*40/100;
            else
                    hra=salary*30/100;
            printf("Hra = Rs. %ld\n",hra);
      }
```

iv. Function with argument & returns a value

```
      void houserent(long int salary)
      {
           long hra;
            if(salary > 50000)
                    hra=salary*40/100;
            else
                    hra=salary*30/100;

            return (hra);
      }
```

v. Recursive function: A process where a function calls itself repeatedly

```
long int factorial(int n)
{
            If(n==0 || n==1)
                    return (1);
            else
                    return (n*factorial(n-1));
}
```

Q 7b. Write a C program evaluating the Binomial coefficient using a function Factorial(n). 10 M

Ans:

```
/* Binomial coefficient using factorial */
#include<stdio.h>

long int fact(int n)
{
    if (n==0 || n==1)
       return (1);
    else
       return (n*fact(n-1));
}

int main()
{
      int n,r;
      printf("Enter n and k : ");
      scanf("%d%d",&n,&r);
      printf("Binomial coefficient of %d & %d = %ld\n",
             n,r,fact(n)/(fact(r)*fact(n-r)));

      return 0;
}
```

Output:

```
Enter n & k : 8 4
Binomial coefficient of 8 & 4 = 70
```

Q 8a. Define a recursion. Write a C recursive function for multiplying two integers where a function call is passed with two integers m &n. 10 M

Ans:

Recursion is a process where a function calls itself directly or indirectly. The simplest example of recursion is finding factorial of a number. The corresponding function is called as recursive function.
factorial of n = n * factorial of (n-1)
Using recursive algorithm, certain problems can be solved quite easily.

```
/* multiplication using recursion */
#include <stdio.h>

int multiply(int m, int n)
{
   if (n==0)
      return (0);
   else
      return (m+multiply(m,n-1)); /* using recursion - adding number n times */
}

int main()
{
   int m,n;
   printf("Enter value of m & n : ");
   scanf("%d%d",&m,&n);
   printf("%d",multiply(m,n));
   return (0);
}
```

Q 8b. Differentiate (i) user defined & built in function (ii) Recursion & iteration 10 M

Ans:

A function is a method to perform a given task. Functions are of 2 types: built-in function (library function) & user defined function

(i) user defined & built in function:

Built-in function:

- Provided by C library
- Already defined in a library file
- One library/header file may have several related library functions defined
- For example math functions like sqrt(), sin(), cos() are defined in library file math.h
- Accessed by writing function name with required list of arguments
- Related header file should be included with pre-processor directive(#include <math.h>)

User defined function (or programmer defined functions):

- Built in functions may not be enough or programmer needs/asked to defined similar function then programmer/user has to define some function themselves for relevant tasks. These functions are called user/programmer defined function.
- A program may be sub-divided in smaller meaningful segments, these segments are called user defined functions.
- Sometimes they may be called as routine/sub-routine, module/sub-module, procedure etc.

- This is known as modular approach where program becomes easy to check & debug.
- User defined functions are also called building-blocks of a program
- Parameters & types are also decided & well defined by user/programmer.

ii) Recursion & iteration

Recursion	Iteration
A statement in the function's body calls the function itself.	Allows the execution of a sequential set of statements repetitively using conditional loops.
A recursive function must comprise of at least one base case i.e. a condition for termination of execution.	There are loops with a control variable that need to be initialized, incremented or decremented and a conditional control statement that continuously gets checked for the termination of execution.
The function keeps on converging to the defined base case as it continuously calls itself.	The value of the control variable continuously approaches the value in the conditional statement.
Stack memory is used to store the current state of the function.	A control variable stores the value, which is then updated, monitored, and compared with the conditional statement.
If there is no base case defined, recursion causes a stack overflow error.	Infinite loops keep utilizing CPU cycles until we stop their execution manually.
The execution of recursion is comparatively slower.	The execution of iteration is comparatively faster.
A statement in the function's body calls the function itself.	Allows the execution of a sequential set of statements repetitively using conditional loops.
A recursive function must comprise of at least one base case i.e. a condition for termination of execution.	There are loops with a control variablethat need to be initialized, incremented or decremented and a conditional control statement that continuously gets checked for the termination of execution.

long int factorial(int n) /* A recursive function to find factorial*/
{
    If(n==0 || n==1)
        return (1);
    else
        return (n*factorial(n-1))
}
	/* A program using loop to for factorial */
	int main()
	{
	  long int fact = 1;
	  int x, n;
	  printf("Enter number : ");
	  scanf("%d", n);
	  for(x=n;x>1;x--)
	     fact=fact*x;
	  printf("Factorial = %ld\n", fact);
	  return (0);
	}

Q 9a. Define structures. Explain how do you declare, initialize & represent the memory for structure variable.

10 M

Ans:

Structure is a data structure whose individual elements can differ in type. It may contain integer elements, character elements, pointers, arrays and even other structures can also be included as elements within a structure. struct is keyword to define a structure.

Syntax:

```
struct  tag
{
            type member1;
            type member2;
            type member3;
            ......;
             type member n;
};
```

New structure type variables can be declared as follows:

```
    struct tag var1, var2, var3, ....... varn, var[10];
```

Declaration for structure variable:

```
struct student
{
    int roll;
    char name[30];
    int marks;
};
struct student s1, s2, s[10],*sp;
```

Here s1 and s2 are structure variables, s[10] is array of structure and *sp is pointer to structure

Initialization for structure variable:

i. Initialization with declaration

```
struct student
{
    int roll;
    char name[30];
    int marks;
}s1, s2 = {111, "Paras", 78};
```

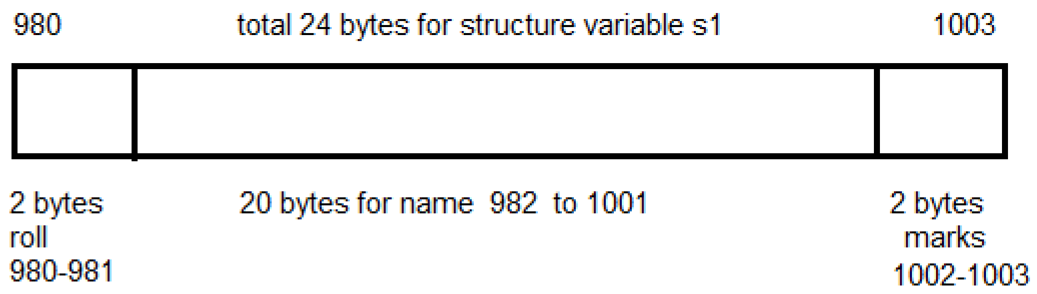
ii. Initialization for individual members

```
s1.roll = 222
strcpy(s1.name, "Kumar Anand");
s1.marks=89;
```

Memory representation structure variable:

```
struct student
{
    int roll;
    char name[30];
    int marks;
}s1;
```

In Memory s1 will occupy 24 bytes (2 bytes for integer roll, 20 bytes for name & 2 bytes for marks)


980
total 24 bytes for structure variable s1
1003
2 bytes
roll
980-981
20 bytes for name 982 to 1001
2 bytes
marks
1002-1003

/* by program */
```
#include <stdio.h>
struct student
{
   int roll;
   char name[20];
   int marks;
}s1;
int main()
{
  printf("size of s1=%d\n",sizeof(s1));
  printf("Address of s1=%u\n",&s1);
  printf("Address of roll=%u, Size of roll=%d bytes\n",
                           &s1.roll,sizeof(s1.roll));
  printf("Address of name=%u, Size of name=%d bytes\n",
                           &s1.name,sizeof(s1.name));
  printf("Address of marks=%u, Size of marks=%d bytes\n",
                           &s1.marks,sizeof(s1.marks));
  return (0);
}
```

Expected Output:
```
size of s1=24
Address of s1=980
Address of roll=980, Size of roll=2 bytes
Address of name=982, Size of name=20 bytes
Address of marks=1002, Size of marks=2 bytes
```

Q 9b. Write a C program that accepts a structure variable as a parameter to a function from a function call. 10 M
Ans:

Structure variable can be passed as copy of the structure or address of the structure variable. If address is passed then called function may corrupt/modify the value of members of the structure.

/* program to structure variable as argument */
```
#include <stdio.h>
struct student
{
   int roll;
   char name[20];
   int marks;
}s1={111,"paras",76};

/* Simple passing copy of structure
   here marks of calling function will not change */
void displaystruct(struct student s)
{
  printf("roll = %d name = %s marks= %d\n",s.roll,s.name,s.marks);
  s.marks=95;
}
```

```
/* passing address the marks in calling function will be changed*/
void printstruct(struct student *s)
{
  printf("roll = %d name = %s marks= %d\n",s->roll,s->name,s->marks);
  s->marks=95;
}

int main()
{
  struct student *sp;
  sp=&s1;
  printf("roll = %d name = %s marks= %d\n",s1.roll,s1.name,s1.marks);
  displaystruct(s1);
  printf("roll = %d name = %s marks= %d\n",s1.roll,s1.name,s1.marks);
  printstruct(sp); /* passing address */
  printf("roll = %d name = %s marks= %d\n",s1.roll,s1.name,s1.marks);
  return (0);
}
```

Output:

```
roll = 111 name = paras marks= 76
roll = 111 name = paras marks= 76
roll = 111 name = paras marks= 76
roll = 111 name = paras marks= 76
roll = 111 name = paras marks= 95
```

Q 10a. Define Pointers. Explain pass by value and pass by reference with C statements and an example. 10 M
Ans:

A pointer keeps address of data. A pointer is variable that represents the location (rather than the value) of a data item, such as a variable or an array element.
Pointers is an important feature of C and frequently used in C. They have a number of useful applications.
Pointer operator is asterisk (*).

To declare a pointer:	*datatype *pointervariable;*
To initialize:	pointervariable = &data;

& is address of operator.

Example1:

```
int n;
int *p;  /* here p is pointer variable & can be used to keep address */
p=&n;  /* here p keeps address of n */
```

Example2:

```
int a[5] = { 10,20.30,40,50};
int *p;
p = a;   /* It is equivalent to  p = &a[0];   Here it keeps address of first element */
```

Pass by value: A function is called passing value. Value of the variable of the calling function will not change
Pass by reference: A function is called passing reference (address of a variable) and received as pointer. The value of the variables of the calling function may be changed by the called function.

```
/*Pass by value – Function is called by passing value. It will not effect the value of calling function */
int area( int l, int w) { return (l*w);
int main()
{
   int len, wid;
```

```
    printf("Enter length and width of rectangle : ");
   scanf("%d%d",&len,&wid);
   printf("Area of rectangle = %d square unit\n", area(len,wid));
  return (0);
}
```

/*Pass by reference When a function is called by passing address then called function may change value of the variable because it receives address. Thus value of the variable of calling function will change also */

```
int swap(int *a, int *b) { int temp=*a; *a=*b; *b=temp; }
int main()
{
   int a=20, b=30;
   printf(a=%d b=%d\n",a,b); /* a=20 b=30 */
   swap(&a,&b); /* pass/call by reference */
   printf(a=%d b=%d\n",a,b); /*a=30 b=20 */
   return (0);
}
```

Q 10b. Define Pre-processor directives. Write C program that finds addition of two squared by defining macro for Square(x). 10 M

Ans:

A pre-processor represents pre-compilation process. These directives are preceded with # (**and there is no semicolon at the end**).

C has the special feature of pre-processor directives. The pre-processor processes the C source code before it passes to the compiler. Pre-processor directives are executed before the C source code passes through the compiler.

Pre-processor directives can be categorized in 2 groups:

- **General Directives : These are written in header section of the program and there will not be control of compiler over it. They are processed before compilation of program starts.**
  Examples: #include, #define, #undef, #pragma

- **Compiler control directives: These are conditional directives and may be written within program also. For example whether a macro is defined or not we can check it within program and redefine (general directives also) if required. Mostly #if family directives are of this category.**
  Examples: #if, #endif, #else, #ifdef, #ifndef

```
/* macro for adding squares */
#include <stdio.h>
#define Square(x)  (x)*(x)

int main()
{
   int a,b,sumsq;
   printf("Enter two numbers : ");
   scanf("%d%d",&a,&b);
   sumsq=Square(a)+Square(b);
   printf("Sum of squares = %d\n",sumsq);
   return (0);
}
```

Output:

```
Enter two numbers : 5 7
Sum of squares = 74
```