CP ASSIGNMENT NO.2.

Q1) What is recursion? Write a program for finding LCM and GCD of two numbers using recursion.

In C, it is possible for the functions to call themselves. A function is called ‘recursive’ if a statement within the body of a function calls the same function. Sometimes called ‘circular definition’, recursion is thus the process of defining something in terms of itself.

Program for finding LCM and GCD:

```
#include <stdio.h>
int gcd(int x, int y);
int main()
{
 int num1, num2, hcf, lcm;
 printf("Enter two Integer Values:\n");
 scanf("%d %d", &num1, &num2);
 hcf = gcd(num1, num2);
 printf("GCD: %d", hcf);
 printf("\nLCM: %d", (num1 * num2) / hcf);
 return 0;
}
int gcd(int x, int y)
{
 if (y == 0)
 {
```

```
 return x;
 }
 else
 {
 return gcd(y, x % y); //calls itself
 }
}
```

Q2. Explain any five string library function with suitable example?

strlen( ):

This function counts the number of characters present in a string. Its usage is illustrated in the following program:

```
# include <stdio.h>
# include <string.h>
int main( )
{
char arr[ ] = "Bamboozled" ;
int len1, len2 ;
len1 = strlen ( arr ) ;
len2 = strlen ( "Humpty Dumpty" ) ;
printf ( "string = %s length = %d\n", arr, len1 ) ;
printf ( "string = %s length = %d\n", "Humpty Dumpty", len2 ) ;
return 0 ;
}
```

The output would be...

string = Bamboozled length = 10

string = Humpty Dumpty length = 13

strcpy( ):

This function copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function. Here is an example of strcpy( )

```
# include <stdio.h>
# include <string.h>
int main( )
{
char source[ ] = "Sayonara" ;
char target[ 20 ] ;
strcpy ( target, source ) ;
printf ( "source string = %s\n", source ) ;
printf ( "target string = %s\n", target ) ;
return 0 ;
}
```

And here is the output...

source string = Sayonara

target string = Sayonara

strcat( ):

This function concatenates the source string at the end of the target string. For example, “Bombay” and “Nagpur” on concatenation would

result into a string “BombayNagpur”. Here is an example of strcat( )

```
# include <stdio.h>
# include <string.h>
int main( )
{
char source[ ] = "Folks!" ;
char target[ 30 ] = "Hello" ;
strcat ( target, source ) ;
printf ( "source string = %s\n", source ) ;
printf ( "target string = %s\n", target ) ;
return 0 ;
}
```

And here is the output...

```
source string = Folks!
target string = HelloFolks!
```

strcmp( ):

This is a function which compares two strings to find out whether they are same or different. The two strings are compared character-by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, strcmp( ) returns a value zero. If they’re not, it returns the numeric difference between the ASCII values of the first non-matching pair of characters. Here is a program which puts strcmp( ) in action.

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

```
int main( )
{
char string1[ ] = "Jerry" ;
char string2[ ] = "Ferry" ;
int i, j, k ;
i = strcmp ( string1, "Jerry" ) ;
j = strcmp ( string1, string2 ) ;
k = strcmp ( string1, "Jerry boy" ) ;
printf ( "%d %d %d\n", i, j, k ) ;
return 0 ;
}
```

And here is the output...

```
0 4 -32
```

strlwr():

It changes all the characters of the string to lower case. Here's an example of strlwr().

```
#include<stdio.h>
#inlcude<string.h>
int main()
{
    char str[100];
    gets(str);
    strlwr(str);
    puts(str);
```

```
    return 0;
}
```

And here is the output:

HII THIS IS YASH

hii this is yash

Q3. Explain different types of storage classes with suitable example?

From C compiler's point of view, a variable name identifies some physical location within the computer where the string of bits representing the variable's value is stored. There are basically two kinds of locations in a computer where such a value may be kept— Memory and CPU registers. It is the variable's storage class that determines in which of these two types of locations, the value is stored.

Moreover, a variable's storage class tells us:

(a) Where the variable would be stored.

(b) What will be the initial value of the variable, if initial value is not specifically assigned.(i.e. the default initial value).

(c) What is the scope of the variable; i.e. in which functions the value of the variable would be available.

(d) What is the life of the variable; i.e. how long would the variable exist.

There are four storage classes in C:

(a) Automatic storage class

(b) Register storage class

(c) Static storage class

(d) External storage class

Automatic Storage Class:

The features of a variable defined to have an automatic storage class are as under:

Storage: Memory.

Default value: An unpredictable value, often called a garbage value.

Scope: Local to the block in which the variable is defined.

Life: Till the control remains within the block in which the variable is defined.

Following program shows how an automatic storage class variable is declared, and the fact that if the variable is not initialized, it contains a garbage value.

```
# include <stdio.h>
int main( )
{
auto int i, j ;
printf ( "%d %d\n", i, j ) ;
return 0 ;
}
```

The output of the above program could be...

1211 221

Register Storage Class:

The features of a variable defined to be of register storage class are as under:

Storage: CPU registers.

Default value: Garbage value.

Scope: Local to the block in which the variable is defined.

Life: Till the control remains within the block in which the variable is defined.

A value stored in a CPU register can always be accessed faster than the one that is stored in memory. Therefore, if a variable is used at many places in a program, it is better to declare its storage class as register. A good example of frequently used variables is loop counters. We can name their storage class as register.

```
# include <stdio.h>
int main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "%d\n", i ) ;
return 0 ;
}
```

Static Storage Class:

The features of a variable defined to have a static storage class are as under:196 Let Us C

Storage: Memory.

Default value: Zero.

Scope: Local to the block in which the variable is defined.
Life: Value of the variable persists between different
function calls.

```
#include <stdio.h>
void increment( ) ;
int main( )
{
increment( ) ;
increment( ) ;
increment( ) ;
return 0 ;
}
void increment( )
{
static int i = 1 ;
printf ( “%d\n”, i ) ;
i = i + 1 ;
}
```

The features of a variable whose storage class has been defined as external are as follows:
Storage: Memory.
Default value: Zero.
Scope: Global.
Life: As long as the program’s execution doesn’t come to an end.

External variables differ from those we have already discussed in that their scope is global, not local. External variables are declared outside all functions, yet are available to all functions that care to use them. Here is an example to illustrate this fact.

```
# include <stdio.h>
int i ;
void increment( ) ;
void decrement( ) ;
int main( )
{
printf ( "\ni = %d", i ) ;
increment( ) ;
increment( ) ;
decrement( ) ;
decrement( ) ;
return 0 ;
}
void increment( )
{
i = i + 1 ;
printf ( "on incrementing i = %d\n", i ) ;
}
void decrement( )
{
i = i - 1 ;
printf ( "on decrementing i = %d\n", i ) ;
```

}

Q4. Differentiate between Structures and Union?

A keyword Struct is used to define Structure.
A keyword Union is used to define a Union.

Each member within the structure is defined a unique storage area.
Memory allocated is shared with all the members of a Union.

Altering a single value doesn’t alter other values.
Altering a single value does alter other values.

Several members can be initialised at once.
Only the first member of a Union can be initialised