Preprocessor

- Design the macros for the following, minimize the side effects wherever applicable
  - Sum of two numbers
  - Square/Cube of a number
  - Biggest of two numbers
  - Length of a 1D array
  - No.of rows, No.of columns in a 2D array
- Provide macros for (a) set (b) reset (c) flip (d) query Kth bit in an integer variable
- Try conditional compilation #if, #ifdef, #ifndef, #elif, #else, #undef
  Provide symbol definition via -D option of gcc
  What if a symbol is defined inside source code and also supplied via -D option?
- Write a small header file "**test.h**" with some prototypes, symbol definitions etc.
  include this header file in two more header files **a.h, b.h**
  now include both **a.h, b.h** in a source file and check for the conflicts.
  In case of conflicts, fix the problem using #ifndef technique or **#pragma once** option
- Explore #pragma directive, options supported by gcc compiler.
- Write a program to make use of special preprocessor symbols like
  __FILE__, __LINE__, __FUNCTION__, __TIMESTAMP__, __DATE,
  __TIME__ etc and #line directive
- Try concatenation of tokens and stringification of macro arguments.

```
#define CONCAT(a,b)        a##b
#define PRINT(str)         puts(#str)
```

- Write the macros

  a) to generate definition of square function which can take any type

  For eg:-

```
#define SQUARED(type,param) \
        type square(type parram) { \
        type res; \
        res=param*param; \
        retrurn res; \
    }
```

  SQUARED(int,x) (or) SQUARED(double,y)

should generate suitable definition of square function

b) to generate definition of sum function which can take two parameters of any type.

i.e. SUMD(int,x,float,y) (or) SUMD(double,p,double,q)

should generate suitable definition for sum function

c) to generate definition of sum function which can take three parameters of any type.

- Try out nesting of macro calls, usage of one macro in other, multi line macros

```
z=SQUARE(SUM(10,20));
```

```
#define QUAD(a,b)   SQUARE(a)+SQUARE(b)+PROD(2,PROD(a,b))
z=QUAD(x,y);
```

```
#define INC(a)      (a)++
#define COMP(a,b)   (a)<=(b)
#define PRINT(n)    while(COMP(i,n)) { \
      printf("%d\n",i); \
      INC(i); \
     }
```

Now invoke PRINT(12)

Miscellaneous

- Enumeration data types
- Sum of variable no.of integers/double values

```
int vsum(int n,...);
vsum(2,10,20);
vsum(3,10,20,12);
vsum(4,11,12,13,14);
```

- Design a function similar to printf, say miniprintf which can handle int, char data types

```
int miniprintf(const char*, …);
```

Dynamic Memory

- Allocate memory for single variable of different types, structure variables and access them.
- Allocating 1D array dynamically and access the elements
- Allocating 2D array (a) contiguous rows with fixed no.of cols,

  (b) Non contiguous rows with variable no.of columns
- Allocate memory for array of structure variables dynamically and access members of each element
- What if size value for malloc, calloc or realloc is zero
- What if old pointer is NULL in case of realloc
- When malloc may return NULL?
- Analyze the following heap problems with **valgrind** tool

```
struct student {
      int rollno;
      char* sname;
     double marks;  };
int *ptr=malloc(40);
struct student* ps=malloc(sizeof(struct student));
ps->sname=malloc(20);
//Access the elements
free(ptr);
free(ps->sname);
free(ps);
```

  - Checking for memory leaks (omission of any free in above code)
  - Double free problem (free(ptr) once again)
  - Invalid read/write detection (beyond block size, ptr[15], *(ptr+12) etc.)
  - Read/Write operations after free
  - free(ptr) after realloc(ptr,0)

I