Friday 19/07/19

Corman → Data structures book.

\# define Y 5 → replace Y as 5 in preprocessing.
\# define SUM (a,b) a+b → SUM → a+b.

Q- macro function that displays prod of 2 numbers

```
# include <stdio.h>
# define PRO(a,b) a*b
int main ( void )
{
  Printf ("Product is %d", PRO(2,3));
}
```

\#define MUL 1*2

Print (MUL)

stdio.h → input/o/p files ke liye
stdlib.h → numeric functions, etc
string.h →
mac.h.

Printf ("%d", PRO (2+3, 4+5));

2+3*4+5

2+12+5

⇒ 19.

C branch → files

If we have to concatenate

```
# include <stdio.h>
# define PRD(a,b) a*b
# define CONCAT(a,b) a##b
int main(void){
printf("%d", PRD(2+3, 4+5));
```

→ it converts it into string literal.

```
→ # define COUTMINE(i,n) while(i<n)
                          {printf("%d", i);}
                          return 0;
                          }
```

→ inline functions

```
# define square(x) x*x
int main(void){
int x = 36/square(6);
printf("%d", x);
}
```

O/P → 36

```
→ inline int square(int x){return x*x;}
```

⇒ Conditional Compilation

```
int main(void){
# if VAR >= 10
printf("u");
# endif
```

EPOC - timestamp

headerfiles
[define
ifndef.
ifdef]

Standard macros
__FILE__ → current file name
date of compilation
__DATE__
time of compilation
__TIME__
line no.
__LINE__

to remove macro already defined
#undef __

Open Ques 1
→ Why is compiler needed for C not for JS ?

to compile with warnings
gcc -wall hello_world.c -o helloworld. (→ -save-temps.)

→ Compilation 4 steps
1) Preprocessing → removal of comments, expansion of macros, expansion of
2) Compiling
3) Assembly
4)

gcc -wall -save-temps hello.c -o hello && ./hello.
cat hello.i
cat hello.s
cat hello.o

→ Diff b/w static linking & dynamic linking

filesize → ls -la

hello → 8304
hello.c → 61
hello.i → 17923
hello.o → 1552
hello.s → 479

Q- interprocess Comm. - User level & Kernel level

Q- PCB's why give command to CPU

Q- Diff b/w signal process & ~~signal probe~~ process interrupt.

\a - alarms → system beeps
\n → new line
\t → tab
\v - vertical tab
\\ → print \
\000 → Octal address.
\x## → hexadecimal
\0 → null.

tokens → identifiers, keywords, operators → unary, binary, ternary

Q- Virtual M/m & Physical M/m
TLB → in what m/m.

see branch → git branch
stathe
git reset

→ Pointer

1) int a=10;
int *ptr;
ptr = &a;

printf("%x", ptr);

→
2) int arr[] = {4,2,3,1};
*ptr = arr;

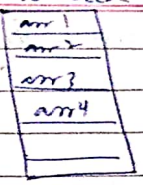
while (*ptr != '\0')
{
printf("%d", *ptr);
ptr++;
}

ptr[0] = *(ptr+0)
ptr[1] = *(ptr+1)

Q → reverse array using pointers by iterative & recursive method

vim pointerarr.c

2D array → linear array of pointer which store addresses of different arrays. and pointer points base address of different 1-D arrays.



Binary Search

```
#include <stdio.h>
int main(void){
int i, n;
scanf("%d", &n);
int arr[n];
for(i=0; i<n; i++)
{
scanf("%d", &arr[i]);
}
int first=0, last=n-1, search;
int middle = (first+last)/2;
scanf("%d", &search);
while (first <= last)
{
If (array[middle] < search)
{
first = middle+1;
}
else if(array[middle] == search)
{
printf("%d found at index %d", search, middle);
break;
}
else
{last= middle-1;}
middle = (last+first)/2 }
}
```

```
int main (int argc, char * argv[])
{
  int arr[argc-1], i, j=0;
  for (i=1; i<argc; i++)
  {
     arr[i] = atoi(argv[i]);
  }
 int *ptr;
 ptr = &arr[argc-1];
  reverse(ptr, argc-1);
 return 0;
 }
void reverse (int *ptr, int n)
 {
    if (n==0)
       return;
     else
     printf("%d\n", *ptr);
     reverse (ptr-1, n-1);
 }
```

~~switch~~ Branch array → git checkout array

time complexity for for loop O(n)

for {
  for {   O(n^2)

recursion
func def.
terminating con

123456

fibonacci series → iterative, recursion.

Q- search an ~~element~~ array in rotated array.
binary search.

Q- sorted - left or right rotate, search using binary search

Q- if array is rotated tell how many rotation and is left rotate or ~~left~~ right rotate

Q- array +ve & -ve all & then find largest continuous sub array
→ largest sum array.

-4 [13 -2 1] -1
— toye hoga.

sum & elements

-4 -2 [3 1] -1

i=0, i<n

1
2

-4 -2, -2 3, 3 1, 1 -1

-4 -2 3, -2 3 1, 3 1 -1

j=1; j<n; j++
i=0; i<n; i++ 1
j=1; j<n; j++ 2
{ a[i]+a[j]; } 3
{ k=2; k<n; k++
a[i]+a[j]+a[k]