Expt.No.....

C

Page No.....
Date .....

TYPES OF PROGRAMMING LANGUAGES:

Low level
Middle level
High level

dssembly language:

C ⇒ structure oriented - procedural language (what, how to do ?) (customization)

OOPS ⇒ Non-procedural (divides the code into small parts & run)

C, Java ⇒ static type language (mention datatype manually) [Fast]

python ⇒ dynamic [slower than static]

C ⇒ OS dependent language (System programming lan) ⇒ modularity ⇒ mother langu

ctype.h ⇒ c
string.h ⇒
conio.h ⇒ console i/p, o/p
stdlib.h ⇒ memory allocation.

Compiler process:

preprocessing → compiling → Assembling → Linking
(c code) (code expand) (assembly code) (exe file)
obj code ↑

Data type:

Int ⇒ -2^{31} to $2^{31}-1$ (4 bytes)
byte ⇒ -128 to 127 (1 bit)
short ⇒ (2 bytes)
Long ⇒ -2^{63} to $2^{63}-1$ (8 bytes)
Float ⇒ (4 bytes)
Double ⇒ (8 bytes)

char ⇒ ASCII key $[-2^7 \text{ to } 2^7-1]$ (1 byte) in C, in Java (2 byte) because of UNICODE)

Variable:

name given to a value. → one among identifier

Format Identifier:

int ⇒ %d	signed int ⇒ %i
char ⇒ %c	long ⇒ %ld
float ⇒ %f	short ⇒ %hd
double ⇒ %lf	

Variable:

Declaration, Initialization, Definition

2018 signed short, short, unsigned short, short int ⇒ 2 bytes.

Type casting:

Change from one datatype to another datatype.

→ Implicit TC

→ Explicit TC

ASCII key value

A = 65	a = 97	Number 0 = 48
Z = 90	z = 122	Number 9 = 57

Total characters in ASCII is 256

Eg: consider a char value 250, but char range (-128 to 127)

-128 ..... 0 ..... 127 (127 < 250)

∴ char 250 = -6

char -130 = 126

Expt.No.....

Page No.....
Date .....

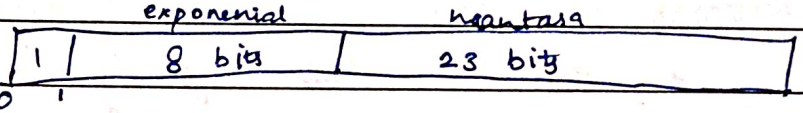
```
unsigned char a = 130;
char s = 130;
printf("Unsigned : %d \n", a);
printf("Signed : %d \n", s);
```

OUTPUT:

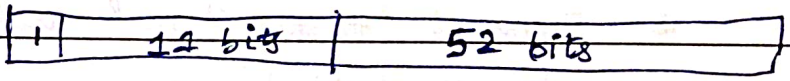
Unsigned : 130
Signed : -126

* Though we declare a char in unsigned defaultly it will consider unsigned as int. So, size = 4 bytes

Float (4 bytes = 32 bits)



Double (8 bytes =)



CODE : printf("%x %o %b", 30, 30, 30);

OUTPUT : 1e 36 11110

1. 127 :

Binary : $2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ ⇒ 01111111

1 1 1 1 1 1 1

Octal : $8^3\ 8^2\ 8^1\ 8^0$ ⇒ 0177

Decimal :

Hexadecimal : 0x7F

700 base 9 ⇒ 857.

784 base 3 ⇒ $3^7\ 3^6\ 3^5\ 3^4\ 3^3\ 3^2\ 3^1\ 3^0$ 243 81 27 9 3 1

⇒ 1 0 0 2 0 0 1

7541 base 6 ⇒ $6^5\ 6^4\ 6^3\ 6^2\ 6^1\ 6^0$

7776 1296 216 36 6 1.

5 4 4 8 6

```
7541
-6480
1061
 864
 197
 144
  53
  48
   5
```

CODE:
```
unsigned int a = -20;
printf("%d", a);      // output : -20
printf("%u", a);      // o/p : 4294967276
printf("%3d", 10);    // o/p : [ ]10   (len of string)
printf("%-10.4s", "Hello"); // o/p : Hell      // (6 spaces)
```

`#include <stdbool.h>` [1 bit] Header file to use boolean.

STRONG NUMBER:

1. Get the last digit
2. Factorial (Iteration & mul until 1 or >0)
3. Add all factorial values
4. Remove the last digit

FUNCTION:

Declaration [return_type function_name (arg data_type);
Definition
Calling

function types:

(i) with arg with ret

```
int sum(int a, int b)
{   int s = a+b;
    return s;
}
```

(ii) with arg without ret

```
void sum(int a, int b)
{   int s = a+b;
    Printf("%d",s);
}
```

(iii) without arg with return type

```
int sum()
{ int a=5, b=6; s = a+b;
    return s;
}
```

(iv) without arg without ret

```
void sum()
{ int a=5, b=6; s = a+b;
    printf("%d", s);
}
```

Expt.No.....

Page No.....
Date .....

Parameter - value pased inside () in function definition.
Argument - value passed inside() during function call.

Local scope, global scope:

Eg: How much percentage 60 in 200 $\frac{60}{200} \times 100 = 30\%$.

```
void percentage (int a, int b)
{ float percent = (((float) a/b) * 100);
  printf("%.2f%%.", percent);
}
```

22/8/24 PATTERNS: (*, 1, A):

1. Right triangle
2. Left triangle
3. Inverted right tri
4. Inverted left tri
5. Straight triangle
6. Inverted str tri
7. Hollow right triangle (for all 6 formats)
8. Diamond pattern, hallow, half diamon
9. Pyramid, hallow pyramid
10. Hour Glass pattern, hallow
11. Floyd's triangle
12. Numbers
```
1
1 2
1 2 3
```
13. Rhombus pattern

1. Dutch National Problem
2. Quick, Merge, Insert sort are efficient sorts. (0, 1, 2) most efficient 2. approach.

28/8

Try:

1. Find second largest element
2. Sort 0s, 1s, 2s in a 1D array
3. Find K^{th} largest element $3/4^{th}$ K smallest.

Practice problems in program wiz.

4. Move (-ve) to the beginning
5. Perfect square without math.h
6. Frequency of vowel in a char array.

24/08

int count[26] = {0} // then all index stores value 0.

In leetcode,

int * nums is ~~similar~~ equal to int nums[]

arr1, arr2 are given. Have to find the median of these.

→ size(arr1) = l_1, sizeof(arr2) = l_2 ; $l_3 = l_1 + l_2$;

→ arr3[l_3];

[1, 2, 5, 8, 12] [3, 6, 10, 20, 25] (i < l_1) (j < l_2)

Right rotation array:

[1, 2, 3, 4, 5], r = 3

rev ((r-1) = 2) rev

2, 1 5, 4, 3

rev

⇒ 3, 4, 5, 1, 2

Left rotation array:

Expt.No.....

Page No.....
Date .....

```
rotateRight (int arr[], int size int r)
{   r = r % size;
    rev (arr, 0, k-1)
    rev (arr, k, size-1)
    rev (arr, 0, size-1)
}
```

SERIES

1. Arithmetic series sum:

$$sum = \frac{no.of.terms}{2} \times (2 \times first\_term + (no.of.term-1) \times com\_diff)$$

$$sum = \frac{n}{2}(2a+(n-1)d)$$

Edge case ⇒ n<=0 ⇒ return 0;

2. Geometric series sum:

$$sum = \frac{first\_term \times (common\_ratio^{no.of\ terms} - 1)}{common\ ratio - 1} = \frac{a(r^n-1)}{r-1}$$

Edge case ⇒ n<=0 ⇒ return 0;

3. Harmonic series sum:

Harmonic series = sum from (n=1) to (n=∞) with terms $1/n$

$= 1/1 + 1/2 + 1/3 + 1/4 + 1/5 + \cdots + 1/n$

Edge case ⇒ n<=0 ⇒ return 0;

4. n fibonacci series

i/p: 5 ; o/p : 0 1 1 2 3

5. Factorial series :

Factorial = $1 \times 2 \times 3 \times \ldots \times n$

Edge case ⇒ n == 0 ⇒ return 1;

Factorial sum series :

Factorial inside for loop.

6. Power series ~~sum~~ :

math.h → power (base, exponent)

7. Taylor series for sin(x) :

$$\text{sum}\left(\frac{(-1)^n \times x^{2n+1}}{(2n+1)!}\right)$$

```
double taylor(double x, int n)
{ if (n<=0) return 0.0;
  double sum = 0.0;
  for (int i=0 ; i<n ; i++) {
    double term = (pow(-1,n) * pow(x, 2*n+1)) / factorial(2*n+1);
    sum += term ; }
  return sum;
```

8. Maclaurin series (for e^x) :

It is a power series that allows one to calculate an approximation of function f(x) for i/p values close to 0.

$$\text{Sum} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

```
double maclaurin(double x, int n)
{ if(n<=0) return 1.0;
  double sum = 1.0, term = 1.0;
  for(int i=01 ; i<n ; i++)
  { term *= x/i;
    sum += term;
  }
  return sum;
}
```

Expt.No.....

Page No.....
Date .....

9. Alternating series

```
int altSerSum(int firstSum, int commratio, int numTerms)
{   if (numTerms <= 0) return 0;
    int sum = 0, term = firstSum;
    for(int i = 0; i < numTerms; i++)
    {   sum += term;
        term *= commratio;
    } return sum; }
```

10. Infinite series: (Harmonic series with tolerance)

```
double infSeries(double tolerance)
{ double sum = 0.0, term = 1.0;
  int n = 1;
  while(term > tolerance)
  { sum += term;
    n++;
    term = 1.0/n;
  } return sum;
}
```

Series: 1/1 + 1/2 + 1/3 + ...
Tolerance : 0.001
O/p : 14.39273

PATTERN

1)
```
*
* *
* * *
* * * *
* * * * *
```

3)
```
    *
  * * *
* * * * *
* * * * * * *
* * * * * * * * *
```

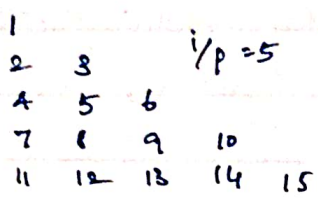
5)
```
* * * *
* * *
* *
*
* *
* * *
* * * *
```

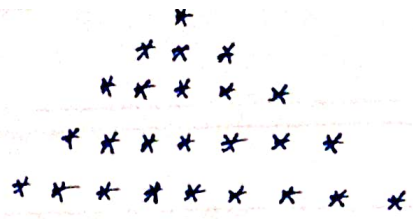
2)
```
* * * * *
* * * *
* * *
* *
*
```

4)
```
    *
  * * *
* * * * *
* * * * * * *
* * * * *
  * * *
    *
```
diamond

6) i/p : 5
```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

7)


8)


9)


10)


21/8/22 **Leetcode 121 - Best time to buy & sell stock**

```
int maxProfit(int *prices, int pricesSize){
{   int min = prices[0];
    int profit = 0;
    for(int i=1; i<=pricesSize; i++)
    {   if(prices[i] < min)
            min = prices[i];
        else if(prices[i]-min > profit)
            profit = prices[i] - min
    }
    return profit;
}
```

Leetcode 55 - Jump game

```
for{ if(i > max) return false;
   } if(max > i + nums[i]) max = i + nums[i];
   return true;
```

Leetcode 70 - Climbing stairs

Climbing stairs(n) = fibaSeries(n+2)

```
int climbStairs(int n)
{ int a=0, b=1, c;
  for(int i=3; i<=n+2; i++)
  {  c = a+b;
     a = b;
     b = c;
  }
  return c;
```

Climbs
1 → 1
2 → 2
3 → 3
4 → 5
5 → 8
6 → 13
7 → 21

Expt.No.....

Page No.....
Date .....

Task:

1. Union, intersection of 2 1D arrays.
2. Frequency char (256 char) → most occuring element.

2-D ARRAY

* Arrays of array
* → int arr1[3][3];
  → int arr2[3][3] = {1,2,3,4,5,6,7,8,9};
  → int arr3[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
  → int arr4[][3] = {1,2,3,4,5,6,7,8,9};

Task:

1. Sum of all elements
2. Search
3. Transpose
4. Addition
5. Multiplication
6. Diagonal sum
7. Antidiagonal sum
8. For 256 chars - freq array - find most occuring element

Binary search in 2D array:

```
while(i < row && j >= 0)
{ if (arr[i][j] == target)
        return 1;
  else if (arr[i][j] < target)
        i++;
  else
        j--;
}
return 0;
```

28/8

Antidiagonal sum of elements in a matrix:

```
void antidiagonalSum (int r1, int c1, int mat[r1][c1])
{ int sum = 0;
  for (int i=0; i< r1; i++)
      sum += matrix[i][r1-i-1];
  printf("sum : %d", sum);
}
```

SEARCHING (3rd approach - fully sorted 2d array / matrix:

FLATTENED APPROACH:

STEP 1 : 1D index calculation

2d position (i,j) = 1d position (i*n+j)

∴ index = i*n+j

STEP 2 : ACCESSING ELEMENTS.

Use flattened index i*n+j in 1d array.

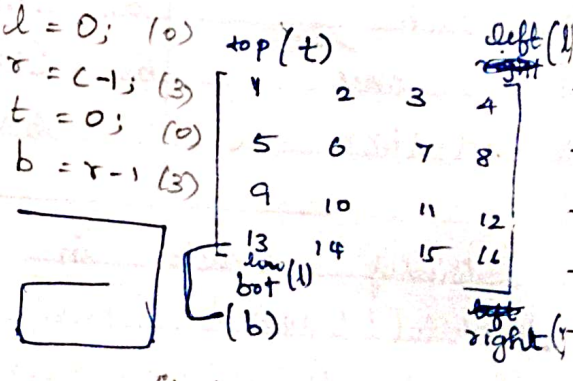
Spiral traversal of a matrix:

```
while[ l <= r && t <= b)
{ for (i = l; i <= r; ++i)
      printf("%d ", a[t][i];
  top++;
  for (i = t; i <= b; ++i)
      printf("%d ", a[i][r];
  r--;
if (t <= b)
{ for (i = r; i >= l; --i)
      printf("%d ", a[b][i];
  b--;
}
```

(i) (ii)

```
if (l <= r)
{ for (i = b; i >= t; --i)
      printf("%d ", a[i][l];
  l++;
}
```

(iii) (iv)



(i) l = 0

o/p: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Expt.No.....

Page No.....
Date .....

STRING : (reverse)

	i/p	o/p
1.	I love programming	- gnimmargorp evol I
2.	I ❤ "	- I evol gnimmargorp
3.	" " .	- Programming love I .

To get str i/p : scanf("%[^\n]%*c", str);

OPERATORS

associativity

1st priority - primary operators - ( ) (left to right)

2nd priority - unary operator - ++, --, &, *, ~, ! (right to left)

3rd priority - Type casting - (right - left)

4th priority - Multiplicative ope - *, /, %. (left to right)

5th priority - Additive ope - +, - (left to right)

6th - Shift ope - <<, >> (left to right)

left shift → a << b = a * 2^b (left to right)

right shift → a >> b = a / 2^b

7th priority - Relational ope - >=, <=, >, < (left to right)

8th priority - Equality ope - ==, != (left to right)

9th pri - Bitwise and (&) - & (")

10th pri - Bitwise XOR - ^ (")

11th pri - Bitwise OR - | (")

Task : 33 & 11 ^ 9

12th pri - Logical AND - && (")

13th pri - Logical OR - || (")

14th pri - Ternary operator - (condition ? true : false) (right to left)

15th pri - Assignment ope - =, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^= (right - left)

16th pri - Comma operator - , (left to right).

1) To convert 'ch' to upper case → ch = ch ^ 32;

Expt.No.....

Page No.....
Date .....

STRING:

⇒ char str[50] = "Hello World";

⇒ char str1[50] = {'H', 'e', 'l', 'l', 'o', '\0'};

⇒ char str2[50];

scanf("%s", str2); printf("%s", str2);

⇒ To give multiple words in input (sentence)

(i) scanf("%[^\n]s", str2);

(ii) ~~gets(str2);~~ [dangerous]

(iii) fgets(str2, 5, stdin); (5 = size)

⇒ Different methods to print string.

(i) printf("%s", str2);

(ii) puts(str2);

(iii) fputs(str2, 5); ~~stdout~~ default - stdout

⇒ str = "Hello"; sizeof(str[])/sizeof(str[0]); o/p = 6

29/8 ~~Anagram~~ : Isogram

str1 = "listen", str2 = "silent".

All chars in str1 also present in str2, then its anagram.

use count[256]; if count[] = 0 then anagram.

ISOGRAM:

str = "machine" → No chars in this words is repeated.

Logic: int count[26] = {0};

for(int i=0; str[i] != '\0'; i++)

{ if(isalpha(str[i])

{ int index = tolower(str[i]) - 'a';

if (count[index] > 0) return 0;

count index ++;

}

}

PANGRAM :

→ a sentence containing all letters in alphabet.

Logic : int alphabet[26] = {0};

index of alphabets present in sentences are '1'.

if any index in alphabet[] is 0, then not pangram

else pangram.

STRING ROTATION : K rotations

right rotation → (i) temp[i] = str[size - K + i];

(ii) str[i] = str[i-K];

(iii) str[i] = temp[i];

left rotation → (i) temp[i] = str[i];

(ii) str[i] = str[i+K];

(iii) str[size-K+i] = temp[i];

VALID PARANTHESES: (using stack in array):

Logic : char stack[1000], int top = -1, int size = strlen(str)

```
for (i to size)
{   char curr = str[i]
    if(curr=='(' || curr=='[' || curr=='{')  stack[++top] = curr;   // push
  else if(curr==')' || curr==']' || curr=='}')
   {  if (top < 0) return false;
      char topchar = stack[top--];   // pop
      if((curr==')' && topchar=='(') || ((curr=='}' && topchar==
                || (curr==']' && topchar=='['))
          return false;
}  }
return top == -1;
```

Expt.No.....

Page No.....
Date .....

INBUILD FUNCTIONS:

string.h → string theaader ✓

(i) strlen (char []) → length of char[]

(ii) strcpy (char1, char2) → char2 value is copied in char1

(iii) strncpy (chr1, chr2, 3) → length

(iv) strcat (c1 , c2) → c2 value is concatinated to c1.

(v) strncat (c1, c2, 3) → concates the limited length

(vi) strcmp (c1, c2) → returns the difference btw 2 strings.

(vii) strncmp (c1, c2, 4) → compares first 4 letters only.

(viii) strchr (r, 'u') → prints from u in the string.

(ix) strrchr (r, 'u') → print 'u' from the last occurance.

(x) strstr (str ,"sstr") → prints the occurance of substring in string

REMOVE DUPLICATES IN STRING:

```
for (i to size)
{   for ( j to i )
    {  if (str[i] == str[j] → break; }
  if(j==i) → str[index++ = str[i];
  str[index] = '\0';
```

30/8 sum of elements

SUBSTRING & SUBSEQUENCE: (2^n)

All substrings are subsequence but not all subsequence are substring

RECURSION :

factorial ⇒ if(n==1 || n==0) return 1;
return n * factorial(n-1);

Power(a^b) ⇒ if(b==0) return 1;
return b * pow(b, e-1);

Task:

Fib sequence
sum of all elements in array
Palindrome (string)
Solve any pattern in recursion.

POINTERS:

A variable that stores the address of another variable.
Format specifier - "%p" - hexadecimal ~~addr~~ address.

continued later

LITERALS : constant values that are assigned to the variables.

INTEGER PROMOTION : some datatypes like char, short int takes less number of bytes than 'int', these datatypes are automatically promoted to int or unsigned int, ~~this~~ when an operation is performed on them.

Q: When an operation is performed btw two short values the size will be (4) → integer promotion.

Q: Size of pointers for all datatypes is 8 bytes

Q: Difference between null and void pointer.

Expt.No..31/08/2024

Page No.....
Date .....

TYPES OF POINTERS

(i) Primitive datatype pointer:

int* p, float* q, double* r, long* v;

(ii) Array pointer:

Array base address = its first element address

Eg:
```
int arr[] = {1,2,3,4,5};
int * f = arr // no '&' used because array is a pointer
for(int i=0; i<5; i++)
    printf("%d ", arr[i]); // O/P: 1 1 1 1 1
```

(iii) Arithmetic pointer:

Performs arithmetic operations in pointers.

Eg: printf("%d ", *f); f++; // o/p: 1 2 3 4 5

(iv) Double pointer:

A pointer that stores address of another pointers.

Used while 2D array ~~is~~ using

(v) Null pointer:

* When the null pointer is de-referenced then the system will crash
* Can be assigned to any pointer
* Used to identify the termination
* Syntax: type ptrname = NULL; type ptrname = 0;

(vi) Wild pointer:

No value is assigned to the pointer.

(vii) Void pointer:

Can store the address of values of all datatypes.

Cannot be de-referenced directly, it should be typecasted

Eg: p = &a, *p; (Wrong)

printf("%d", *(int *)p); (Correct)

(viii) Function pointer:

&function_name — is called code reference.

```
Eg: int add(int a, int b) // function definition
        return a+b;

    main()
    { int (*ptr)(int, int) = &add; // using code ref to pointer
      (*ptr)(4, 3); // function call
    }
```

(ix) Dangling pointer:

Pointer that points to a memory location that has been deallocated or is no longer valid.

```
Eg: int *p = (int *) malloc(sizeof(int));
    printf("%d ", *p);
    free(p);
    printf("%d ", *p); // Dangling pointer.
```

(x) Constant pointer: (Address - constant)

the address once assigned it cannot be re assigned.

```
Eg: int * const p = &a;
```

(xi) Pointers to constant: It points to the constant variable.

```
Eg: const int c = 5;
    const int *q; // pointer used to store const value
    int *r;
```

(Value - constant)

Expt.No.....

Page No.....
Date .....

Storage class :

Auto , static , extern , register.

Auto - local variable (default storage)
- variables can be accessed only within the block.
- stored in stack memory.

Register :

Static : - Initialized only once & exist till the termination
No new memory is alloted because they aren't redeclared.
- Restricting the usage of function [in function]
- Used in

Extern : - Global variable
- Only declaration occurs definition will be in some other file.

STRUCTURE :

→ User - defined datatype
→ used to grp different datatypes into a single type.
→ Only declaration is done, the memory is allocated ~~when~~ only when it is defined.
→ Can be initialize the value in 3 methods.
(i) dot operator `stu1.name = "ABC";`
(ii) Using initializer list `stu2 = {"ABC", 12, 12345};`
(iii) Designated initializer list `stu3 = {.name = "ABC"};`
→ typedef - keyword - provide some meaningful names to the already existing variable in C program.

Nested structure:

→ structure within structure.

→ One structure can be declared inside another structure in the same way structure members are declared inside a structure.

structure pointer: the pointer which points to the address of the memory block that stores a structure.

Eg:
```
struct node
{   int data;
    struct node *next; // structure pointer
};
```

Task:

Write code to add, mul, 2 complex nums using struct.

PADING:

In C it adds the one or more empty bytes betw the memory addresses to align the data in memory.

UNION: (Shared memory allocation)

Similar to structure. (user defined data type).

But all the members in the C union are stored in the same memory location.

So only one member can be accessed at a time.

DYNAMIC MEMORY ALLOCATION :

→ stdlib.h → stored memory in heap.

Malloc - as per the given length of datatype the memory is alloted random.
- Return type void.

Calloc - also void return type. similar to malloc
- But continuous memory allocation.
- Initalize 0 in all memory not garbage value.

Realloc - → void pointer return type.
Used to extend / shrink the size of the memory created by malloc, calloc.

free - clear the memory created before.

POINTER ARITHMETIC :

→ Increament, decreament can be performed in Pointer

→ Operations which can be performed.

(i) Addition of an integer or Increament

(ii) Substraction of an integer or Decreament

(iii) Substraction of one pointer from another pointer (same type).

(iv) Relation operators can be used btw 2 pointers.