DATA STRUCTURE & ALGO

1.0 Binary (0,1) → Language of computer
(0,1 → ✱ low / High Voltage)

Que We have two hourglass of 7hr & 4hr. How to measure 9mins?
In 4 hours, the hourglass II will be empty, further we will flip the hourglass II for 3 hr till hourglass I is empty. Now 7 hours have elapsed and we have 1 hr sand in hourglass II, let the sand flow & flip hourglass I. After eighth hour, hourglass II will be empty, again we will flip hourglass I & II. When hourglass I will be empty, 9 hours will be elapsed.

1.1 PROGRAM: Set of instructions that tell a computer what to do.

1.2 Compiler: Computer program that translates HLL to Machine language. (Intermediate Code)

1.3 Interpretor: Executes the instructions directly & removes the process of converting to intermediate code

1.4 C++ → Cross-platform language, Developed by Bjarne Stroustrup.

Que 3 light bulbs in a room, 3 button outside, can go inside once, how to find the light & it's switch

First, turn on a light for 30 mins, turn it off & enter the room,

- Warm Light → First Switch
- Glowing Light → Second Switch
- Off Light → Third Switch

Que 3ltr jug + 5ltr jug. How to measure 4ltr

- Fill 3ltr jug (0, 3)
- Pour 3ltr to 5ltr & fill 3ltr again (3, 3)
- Pour 3ltr to 5ltr again (5, 1)
- Empty 5ltr & pour 3ltr to 5ltr (1, 0)
- Fill 3ltr (1, 3)
- Pour 3ltr to 5ltr (4, 0)

2.0 VARIABLES: Container used to hold data.

Each var should have a unique name

Data Type	Meaning	Size
int	Integer	4
float	Floating-point	4 → Upto 7 Decimal
double	Double Floating pt.	8 → Upto 15 Decimal
char	Character	1 → Inside single quote
wchar_t	Wide Character	2
bool	Boolean	1 → false/true
void	Empty	0

2.1 Data Type Modifiers

↳ Used to modify the fundamental DataType.
↳ signed, unsigned, long, short

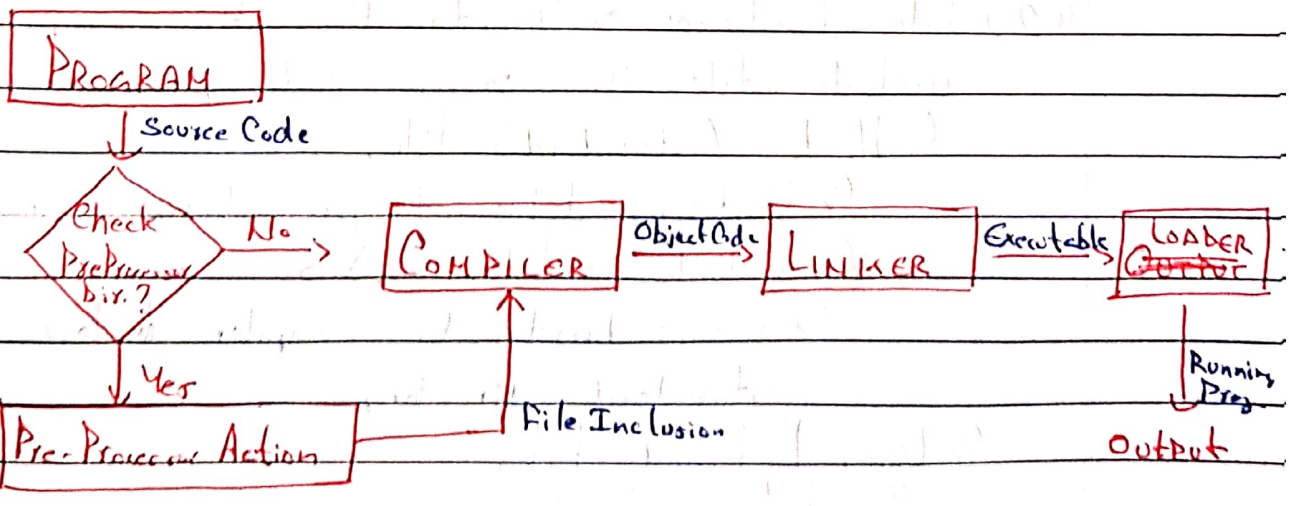
iostream : Input/Output stream in C++ used for input as well as output of a particular value.
↳ cout → Standard Output
cin → Standard Input
cerr → Standard Error

3.0* Comment : Displays important info about prog.

Compiler skips the comment line.
// Single Line /* Multi Line Comment */

When & Why Comments?
↳ Make our code more readable
↳ Provide description to algorithm

4.0* Pre-Processor: Programs that process source code before compilation.



↳ Pre-processor begins with # symbol.

4.1) Macro → Piece of code which is given some name.
↳ Whenever the name is encountered, compiler replaces name with code.

```
# include <iostream>
using namespace std;
# define LIMIT 5
int main() {
    cout << LIMIT << endl;
    return 0;
}
```

→ The limit var defined will replace whenever called.

Note: There is no semicolon (;) at the end of macro defination

↳ We can also pass arguments to macro & make it as function.

```
# define AREA(l, b) (l*b)
AREA(5, 2)
```

→ Here, the macro is sep by space b/w arguments & return value, act like function.

4.2) File Inclusion → Instructs the compiler to include file in source code program.
↳ Header (Standard) files
↳ Contains defn of pre-defined functions
include <filename>
↳ <> → Instructs compiler to look files in standard directory.
↳ User Defined files
include "filename"
↳ " " → Instruct compiler to look in current directory.

4.3) Header File Inclusion

↳ Used in large programs where a single user-defined header is in use by multiple files.

#ifdef MACRO
Used in case of Macro
#endif

```
#ifndef --HEADER--
#define --HEADER--
        HEADER FILE CONTENT
#endif
```

→ Used in header files so that linker doesn't link the content multiple times.

(github.com/kshitizsaini413/L2-Emulator)

↳ Here, if the macro is defined, the compiler will skip block of code else will simply execute it.

4.4) #undef → Used to undefine a macro.

```
#undef MACRO.
```

5.0 Why not use "using namespace std"

↳ It is suggested to use scope resolution operator (::) to import specific functions rather than importing entire std namespace.

```
using std::cout;
```

↳ std namespace is huge & has hundreds of defn so user may ~~refi~~ redefine & result in compilation error.

6.0 main()

↳ main() → Entrypoint of program.

↳ Every C++ program must have a main() function.

7.0 Basic Input / Output

↳ C++ Input / Output are known as streams
↳ Input Stream: Keyboard → Main Memory
↳ Output Stream: Main Memory → Screen

iostream : (Standard input / output) Contains defn of cin, cout, cerr, etc.
iomanip : (Input/Output manipulators) Contains methods for manipulating streams.
fstream : (File Stream) Handler data from file as input or output

↳ cout : Produce o/p on the stream (screen) using the insertion operator (<<)
↳ cin : Read i/p from the keyboard using the extraction operator (>>)
↳ cerr : Standard error stream used to o/p error message immidiately. & don't store for later.
↳ clog : Error is first inserted to buffer and then displayed.

8.0 bits/stdc++.h

↳ using <bits/stdc++.h>
  ↳ Not available in all compilers
  ↳ Reduces time of writing include
  ↳ Increases compilation time.

9.0 NAME SPACE

↳ Allow us to group named entities & remove them from global namespace to reduced namespace.

```
namespace NAMESPACE {

}
```

10.0 endl v/s \n

↳ endl → Inserts a new line & flushes the stream.

↳ \n → Only inserts a new line

cout << endl; ≡ cout << '\n' << flush;

\n is performance-wise better unless flushing of stream is reqd.

11.0 preIncrement v/s postIncrement

↳ preIncrement can be used as left value but postIncrement can never be used as as a left value.

```
++a = 20   ✓
a++ = 20   → ERROR
```

↳ ++a returns a refrence to variable to which we can further assign whereas a++ holds the value.

```
int &r = ++a  → ✓
int &r = a++  → ERROR
```

a++ = 20 Explained

a = 10

a++ = 20 Illegal

↳ 10 = 20

a = a+1

12.0 Precision for Numbers

↳ 12.1) floor()
↳ Rounds off to closest integer less than given value.

↳ 12.2) ceil()
↳ Rounds off to closest integer more than given value.

↳ 12.3) trunc()
↳ Removes digits after decimal points

↳ 12.4) round()
↳ Rounds number to closest integer

↳ 12.5) setprecision()
↳ Provides precision to floats when used with fixed keyword.

```
int x = 1.1311.3
cout << floor (x); → 1
cout << ceil (x); → 2
cout << trunc (x); → 1
cout << round (x); → 1
cout << fixed << setprecision (2) << x; 1.13
```

13.0 if-elif-else

↳ Used to specify the code to be executed according to some conditions.

```
if (CONDITION) {
} else if (CONDITION) {
} else {
}
```

Que Check if even /odd number.

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cin >> n;
    if (n % 2 == 0) {
        cout << "Even";
    }
    else {
        cout << "Odd";
    }
    return 0;
}
```

→ Condition for checking if a number is even or odd by deviding it by 2 and checking reminder.

Que Max - Min from two numbers.

```
void max_min (int n1, int n2) {
    if (n1 > n2) {
        cout << "Max" << n1;
        cout << "Min" << n2;
    }
    else (n2 > n1) {
        cout << "Max" << n2;
        cout << "Min" << n1;
    }
}
```

Que Max from 3 numbers

```
int max_three (int n1, int n2, int n3) {
    if (n1 > n2) {
        if (n1 > n3) {
```

```
        cout<< return n1;
    }
    else {
        return n3;
    }
}
else {
    if (n2>n3){
        return n2;
    }
    else{
        return n3;
    }
}
```

Que Check the type of triangle.

```
if (s1 == s2 && s2 ==s3){
    cout << "Equiletral " << endl;
}else if (s1 == s2 || s2 == s3 || s3 == s1){
    cout << "Isoscales" << endl;
}
else{
    cout<< "Scalene" << endl;
}
```

14.0 Loops

↳ Loops are used for executing a block of statement repetedly until a condition is satisfied.

- ↳ Entry Controlled : for, while
- ↳ Exit Controlled : do-while.

14.1) for loop

```
for (Initialization; Condition; Modification)
{
    // Statements
}
```

14.2) while loop

```
while (Condition)
{
    // Statement
}
```

14.3) do-while loop

```
do
{
    //Statement
} while (Condition);
```

15.0 Jumps (loops) (Control Flow)

↳ Jumps are used to control the flow of loop.

- ↳ break
- ↳ continue

15.1) continue
↳ used to skip to next iteration of loop.

15.2) break
↳ used to terminate the current loop.

Que Calculate sum of first n naturals.

```
for (int i = 1; i <= n; i++) {
    sum += i;
}
cout << sum;
```

Here the sum must be initialized to 0 & n must be declared first.

Que Print table of a number

```
for (int i = 0; i <= 10; i++) {
    cout << "n << " * " << i << " = " << n*i;
}
```

Que Add numbers till +ve are encountered.

```
int sum () {
    int n = 0;
    int sum = 0;
    while (n >= 0) {
        sum += n;
        cin >> n;
    }
    return sum;
}
```

Que Print odd till n

```
#include <iostream>
using namespace std;
void odd (int n){
    for (int i = 1; i<=n; i++){
        if (i % 2 == 0)
            continue;
        cout << i << " ";
    }
}
```

Que Check prime number

```
bool prime_check (int n) {
    for (int i=2; i <=sqrt(n); i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

math.h needs to be included for the function to work.

16.0 SWITCH-CASE

- Substitute for long if that compare a variable to multiple values
- Switch is a multiway branch statement which provides an easy way to dispatch execution to different part of code, based on value.
  - → No duplicate case allowed.
  - → Optional break & default statement.
  - → Two cases can't be same.
  - → Constant expression must be used instead of variables in case.

Some interesting programs about switch-case,

16.1) Using var as case,

```
const int i = 0;
switch (in)
{
    case i:          ] → Here case will
        //STATEMENT      work as it is
        break;         provided constant
    ...                value.
}
```

16.2) Find largest b/w two no using switch case

```
switch (n1 < n2)
{
    case 0:          ] → We use case 0 to
        //STATEMENT      denote false case
        break;
    default:         ] → We use default because
        //STATEMENT      every thing except 0
        break;           is true.
}
```

Que Simple Calculator

```
int calc (int n1, int n2, char op) {
    switch (op) {
        case '+':
            return n1 + n2;
        case '-'
            return n1 - n2;
        case '*'
            return n1 * n2;
        case '/'
            return n1/n2;
```

17.0 OPERATORS

17.1) Arithmetic Operators
- +, -, *, /, %, ++, --

Performs some arithmetic operations.

17.2) Relational Operators
- ==, !=, >, <, >=, <=

Define relation b/w entities, result in true/false.

17.3) Logical Operators
- &&, ||, !

Connect multiple expressions / conditions.

17.4) Bitwise Operators
- &, |, ^, ~, <<, >>

Operate on bits & perform bit-by-bit operation.

Postfix → Unary → Multiplicative → Additive → Shift → Relational → Equality → Bitwise AND → Bitwise XOR → Bitwise OR → logical AND → logical OR → Ternary → Assignment

Que Reverse a number

```
int reverse (int num)
{
    int revNum = 0;
    for (; num > 0; num /= 10)
    {
        revNum *= 10;
        revNum += num % 10;
    }
    return revNum;
}
```

Que Armstrong Number Check

```
bool isArmstrong (int num)
{
    int count = 0;
    for (int temp = num; temp > 0; temp /= 10)
    {
        count++;
    }
    int arm = 0;
    for (int temp = num; temp > 0; temp /= 10)
    {
        arm += pow (temp % 10, count);
    }
    return num == arm;
}
```

math.h needs to be imported.

18.0 FUNCTIONS

↳ Function is a block of code that perform a specific task.

→ Easy maintainence, more readibility, easy to understand.

```
returnType functionName (par1, par2, ...)
{
    // FUNCTION BODY
}
```

→ returnType: Datatype of var that function returns.
→ functionName: Unique name of function.
→ parameters: The values that function accept, along with dataType.

Note: It is recommended to declare a function before it is being used.

Que Calculate factorial

```
int factorial (int num)
{
    int fact = 1;
    for (int i=1; i<=n; i++)
    {
        fact *= i;
    }
    if (n >= 0) {
        return fact;
    } else {
        return -1;
    }
}
```

Que Print fibonacci

```
void fibonacci (int n1, int n2), int n)
{
    count << n1 << " " << n2 << " ";
    for (int i=1; i<n-1; i++)
    {
        int temp = n1;
        int n1 = n2;
        n2 = n2 + temp;
        cout << n2 << " ";
    }
    cout << endl;
}
```

Que Calculate sum of first n naturals

```
int sumOfNaturals (int n)
{
    int sum = (n) * (n-1) / 2;
    return sum;
}
```

Que Check pythagorean triplet

```
bool calculatePythogorean (int s1, int s2, int s3)
{
    return (s1*s1 == s2*s2 + s3*s3);
}
bool checkPythogorean (int s1, int s2, int s3)
{
    int max_side = max (s1, max (s2, s3));
    bool isPythogorean = false;
    if (max_side == s1)
        isPythogorean = calculatePythogorean (s1, s2, s3);
    else if (max_side == s2)
        isPythogorean = calculatePythogorean (s2, s3, s1);
    else
        isPythogorean = calculatePythogorean (s3, s1, s2);
    return isPythogorean;
}
```

Here, checkPythogorean() function will be called which in-turn will call calculatePythogorean() function.

Que Binary to Decimal

```
int binaryToDecimal(int binary)
{
    int decimal = 0;
    for (int i=0; binary>0; binary /= 10, i++)
    {
        if (binary % 10)
        {
            decimal += pow(2,i);
        }
    }
    return decimal;
}
```

Needs to import math.h

Que Octal to Decimal

```
int octalToDecimal(int octal)
{
    int decimal = 0;
    for (int i=0; octal>0; octal /= 10, i++)
    {
        decimal += octal % 10 * pow(8,i);
    }
    return decimal;
}
```

Que HexaDecimal to Decimal

```
int hexadecimalToDecimal(string hexadecimal)
{
    int ans = 0;
    int x = 1;
    for (int i = hexadecimal.size()-1; i>=0; i--)
```

```
{
    if (hexadecimal[i] >= '0' && hexadecimal[i] <= '9')
        ans += x * (hexadecimal[i] - '0');
    else if (hexadecimal[i] >= 'A' && hexadecimal[i] <= 'F')
        ans += x * (hexadecimal[i] - 'A' + 10);
    x *= 16;
}
return ans;
}
```

Que Decimal to Binary

```
int decimalToBinary(int decimal)
{
    int binary = 0;
    int x = 1;
    while (x <= decimal)
    {
        x *= 2
    }
    x /= 2;
    while (x > 0)
    {
        int last = decimal/x;
        decimal -= last * x;
        x /= 2;
        binary *= 10;
        binary += last;
    }
    return binary;
}
```

Que Decimal to Octal

```
int decimalToOctal(int decimal)
{
    int octal = 0;
    int x = 1;
    while (x <= decimal)
    {
        x *= 8;
    }
    x /= 8;
    while (x > 0)
    {
        int last = decimal / x;
        decimal -= last * x;
        x /= 8;
        octal *= 10;
        octal += last;
    }
    return octal;
}
```

Que Decimal to HexaDecimal

```
string decimalToHexadecimal(int decimal)
{
    string hexaDecimal = "";
    int x = 1;
    while (x <= decimal)
    {
        x *= 16;
    }
    x /= 16;
```

```
while (x>0)
{
    int last = decimal%x;
    decimal -= last * x;
    x /= 16;
    if (last<10)
    {
        hexaDecimal += to_string(last);
    }
    else
    {
        char ch = 'A' + last - 10;
        hexaDecimal += ch;
    }
}
return hexaDecimal;
}
```

19.0 Space & Time Complexity

19.1) Time Complexity

↳ Amount of time taken by algorithm to run as function of length of input.

O → O-notation → Worst Case
Ω → Omega-notation → Best Case
Θ → Theta-notation → Average Case.

$O(n^n) > O(n!) > O(n^3) > O(n^2) > O(n\log n) > O(n\log(\log(\log n)))$
$> O(n) > O(\text{sqrt}(n)) > O(\log n) > O(1)$

19.2) Space Complexity
↳ Quantifies the amount of ~~time~~ space taken by prog. to run as function of length of input. Directly prop. to largest mem. acquired during the running time.

Que Add binary numbers,

```
int addBinary(int n1, int n2) {
    int ans=0, carry=0;
    while (n1>0 && n2>0) {
        if (n1%2==0 && n2%2==0) {
            ans = ans*10 + carry;
            carry = 0;
        }
        else if ((n1%2==0 && n2%2==1) || (n1%2==1 && n2%2==0))
        {
            if (carry==1) {
                ans = ans*10 + 0;
                carry = 1;
            }
            else {
                ans = ans*10 + 1;
                carry = 0;
            }
        }
        else {
            ans = ans*10 + carry;
            carry = 1;
        }
        num1 /= 10;
        num2 /= 10;
    }
```

```
while (n1 > 0) {
    if (carry == 1) {
        if (n1 % 2 == 1) {
            ans = ans * 10 + 0;
            carry = 1;
        }
        else {
            ans = ans * 10 + 1;
            carry = 0
        }
    } else {
        ans = ans * 10 + (n1 % 2);
    }
    num1 /= 10;
}
while (n2 > 0) {
    if (carry == 1) {
        if (n2 % 2 == 1) {
            ans = ans * 10 + 0;
            carry = 1;
        } else {
            ans = ans * 10 + 1;
            carry = 0;
        }
    } else {
        ans = ans * 10 + (n2 % 2);
    }
    num2 /= 10;
}
if (carry == 1)
    ans = ans * 10 + 1;
return reverse(ans);
```