Principle of Data Abstraction:

Programmer(client side) should be able use instances of user defined data types without the knowledge of its data layout.

Data layout -> Name & TypeName of every data member in type definition == Data layout of instance of type.

(डिफाइन केलेल्या डेटा टाइप चे इंस्टेंसेस, डेटा टाइप चा लेआउट माहिती न करुन घेता वापरता यावेत)

```
struct Date{
    int day;
    int month;
    int year;
};
```

Knowing data layout of struct Date -> Having & Using knowledge that variable of struct Date (or instance of struct Date) has three data members viz. day, month and year and their types are int, int and int respectively.

```
int main(void)
{
    struct Date myDate = {8, 2, 2025};
    // FOLLOWING LINES VIOLATED THE DATA ABSTRACTION PRINCIPLE
    printf("%d / %d / %d\n", myDate.day, myDate.month, myDate.year);

    return (0);
}
```

//

Date.c

```
// SERVER SIDE START
struct Date
{
    int day;
    int month;
    int year;
};

void show(struct Date* pDate){
    printf("%d / %d / %d\n", pDate->day, pDate->month, pDate->year);
}

// SERVER SIDE END
```

```
// CLIENT SIDE START
int main(void)
{
    struct Date myDate = {8, 2, 2025};

    show(&myDate);

    return (0);
}

// CLIENT SIDE END
```

This version of C code DOES NOT VIIOLATE DATA ABSTRACTION PRINCIPLE

MODULAR PROGRAMMING

Date.h

```
#ifndef _DATE_H
#define _DATE_H

struct Date
{
    int day;
    int month;
    int year;
};

void show(struct Date* pDate);

#endif
```

Date.c

```
#include "Date.h"
#include <stdio.h>

void show(struct Date* pDate)
{
    printf("%d / %d / %d\n", pDate->day, pDate->month, pDate->year);
}
```

Date.h + Date.cpp = Date.dll or libdate.so (Date.lib)

```
TO client -> Date.h & Date.lib

useDate.c // CLIENT FILE

#include "Date.h"   # Furnished by server

int main(void)
{
    struct Date myDate = {25, 8, 2025};

    show(&myDate);

    return 0;
}

# cl /EHsc /Fe:app.exe useDate.c Date.lib

# gcc -o app Date.c -lDate

#-------------------------------------------

MODULAR PROGRAMMING SUMMARY:
    If you want to implement new Data type T then at server side create
    two files
    T.h and T.c

    T.h will contain data layout of type T (i.e struct T) and declarations
    of functions that will process instances of type T.
    (and any relevant symbolic constants nad typedefs)

    T.c
    will contain implementation of all functions in T.h

    DEPLOYMENT CONSIDERATIONS:
    T.h and T.c can be built into a static or dynamic link library
    Client should keep T.h in global include path or project relative
    include path and T.lib in global lib path or project relative lib path

    For creating and using instances of type T
    the client should
    #include Date.h
    #include <Date.h> // if Date.h is in global include path
    #include "Date.h"   // if Date.h is in project relative include path
```

```
   While linking -> Date.lib should be linked
   (Link command depends on OS and Compiler)

###################

CONCEPT:
   INITIALIZATION OF DATA OBJECT:
      ACT OF PUTTING DATA VALUE IN DATA OBJECT AT THE TIME OF ALLOCATING
      MEMORY TO IT IS CALLED AS INITIALIZATION.

int num = 10;    // WHILE ALLOCATING 4 BYTES -> YOU MUST VALUE 10 IN IT.
                 // OR AT LEAST YOU SHOULD BE ABLE PROVIDE SUCH EFFECT

// Global data definition statement int C

int num = 10;

ASSEMBLY CONVERSION

.section .data
   num:
   .int   10

ASSEMBLER OBJECT FILE
   [      1010]

from source code -> num goes to object file (4 bytes with 1010)
-> object file -> exe file carry forward
-> loader loads data section

[      1010] -> exe -> virtual address space -> physical address space
4 bytes allocation + initialization to 10 -> ATOMIC (INDIVISIBLE STEP)

void test()
{
   int num = 10;
}

.section .text
.globl test
.type test, @function

test:
```

```
    # PROLOGUE
    pushl  %ebp
    movl   %esp, %ebp

# int num = 10;
    subl   $4, %esp  # NUM LA MEMORY ALLOCATE HOTE
    movl   $10, -4(%ebp) # NUM HA 10 LA INITIALIZE HOTO

# printf("num = %d\n", num)
    # ASSEMBLY

#-------------------------------------

int num = 10;

1) POSSIBILITY 1 : It is really possible for compiler to set value 10 in four
bytes WHILE THEY ARE BEING ALLOCATED.

2) POSSIBILITY 2: Compiler allocates memory of 4 bytes in step 1 and assigns
value 10 to it in step 2

BUT IN BOTH CASES : C/C++ programmer should get the effect AS IF 10 was assigned
while memory was BEING ALLOCATED.

int num = 10;

// if num is accessed then its value must be 10

// In C
// BUILT IN TYPE | USER DEFINED DATA TYPE
// WE CAN ALWAYS INITIALIZE
// PROOF

int num = 10;

struct Date myDate = {8, 2, 2025};

// IN BOTH CASES C PROGRAMMER GETS THE FEELING OF INITIALIZATION

##############################
But due to data abstraction principle C++ programmer cannot get this
feature (or feeling of initialization) without special arrangement
```

```
// FOR C++ programmer

int main(void)
{
    int num = 10; // ok

    Date myDate = {10, 2, 2025}; // ERROR
                                 // BECAUSE day, month, and year of myDate
                                 // are not accessible in main() because
                                 // main() is external to class Date
}

class Date
{
    private:
        int day, month, year;
    public:
        void init(int init_day, int init_month, int init_year)
        {
            this->day = init_day;
            this->month = init_month;
            this->year = init_year;
        }

        void show()
        {
            cout << day << "/" << month << "/" << year << endl;
        }
};

int main(void)
{
    Date myDate;

    // C++ LEVEL LA : ALLOCATION AND SETTING VALUE -> DIFFERENT STEP
    myDate.init(8, 2, 2025);

    myDate.show();
    return 0;
}
```

```
// IF C++ PROGRAMMER WANTS TO INITIALIZE OBJECT CREATED FROM CLASS
// THEN WHAT HE/SHE WOULD WISH FOR??

int main(void)
{
    Date myDate(8, 2, 2025); // AT THIS STATE -> THIS IS A WISH LIST
                                // TOMORROW(NEXT SESSION) THIS WILL BE REALITY

    myDate.show();   // 8/2/2025 HE PRINT ZALA PAHIJE

    return (0);
}

// IF THIS CODE WORKS THEN WE CAN SAY THAT C++ PROGRAMMER CAN INITIALIZE
// USER DEFINED CLASS' OBJECT

4KB -> 0

RAM MADHALE CONTENT 0
CPU INVOLVE

1 DATA CYCLE -> MAX 16 BYTE

4096/16 =256
#----------------------------------

INSTRUCTION SET

READ / WRITE -> ATOMIC OPERATIOON

READ | ALU | WRITE -> ATOMIC INSTRUCTION

int num = 10;
// MEMORY ALLOCATION AND ITS INITIALIZATION


CISC -> Complex Instruction Set Computer

COMPUTER ARCHITECTURE -> ADVANCES

PIPELINE STAGES -> OUT OF ORDER EXECUTION ENGINE
```

[

INSTRUCTIONS -> MICRO OPERATIONS

]

RISC = REDUCED INSTRUCTION SET COMPUTER (MIPS)