Assignment :- 1 **Name :- Jayesh Sharma**

Lab :- Advances Data Structure **Roll No :- 23115045**

Assignment - 1 : Abstract Data Types

Abstract Data Type (ADT)

An Abstract Data Type (ADT) is a data structure model in which the implementation is concealed and only the interface or the behavior is revealed. It defines what can be done with the data, but not how that is being done.

Key Properties of ADT:

- **Focuses on what an object will do, rather than how it will do it**
- **Encourages modular construction**
- **Enhances maintainability and discourages direct code reuse**

What is Abstraction?

Abstraction in C++ is concealing the intricate implementation details and revealing just the required characteristics of an object. Abstraction reduces programming effort and complexity.

Real-Life Analogy:

You operate the car's steering, brakes, and pedals. You don't need to know about the internal combustion engine or how the fuel is ignited — this is abstraction.

C++ Example: Stack as an ADT

Let’s create a basic Stack class using arrays — a classical example of an Abstract Data Type.

Interface (exposed to the user):

- `push(x)`
- `pop()`
- `peek()`
- `isEmpty()`

Internal Implementation (hidden from the user):

- **Array-based data storage**

- **Top index tracking**

Explanation

The Stack class is an ADT: it exposes only necessary operations (push, pop, etc.) and hides internal data like the array and top index.

This illustrates abstraction in practice: the user of the Stack doesn't need to know how the data is internally managed — they simply use the interface provided.

```
#include <iostream>
using namespace std;

class Stack {
private:
    int arr[100];
    int top;

public:
    Stack() {
        top = -1;
    }

    void push(int x) {
        if (top >= 99) {
            cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = x;
    }
```

```cpp
    void pop() {
        if (top == -1) {
            cout << "Stack Underflow\n";
            return;
        }
        top--;
    }

    int peek() {
        if (top == -1) {
            cout << "Stack is empty\n";
            return -1;
        }
        return arr[top];
    }

    bool isEmpty() {
        return top == -1;
    }
};

int main() {
    Stack s;

    s.push(10);
    s.push(20);
    cout << "Top element: " << s.peek() << endl;

    s.pop();
    cout << "Top after pop: " << s.peek() << endl;

    s.pop();
    s.pop();

    return 0;
}
```