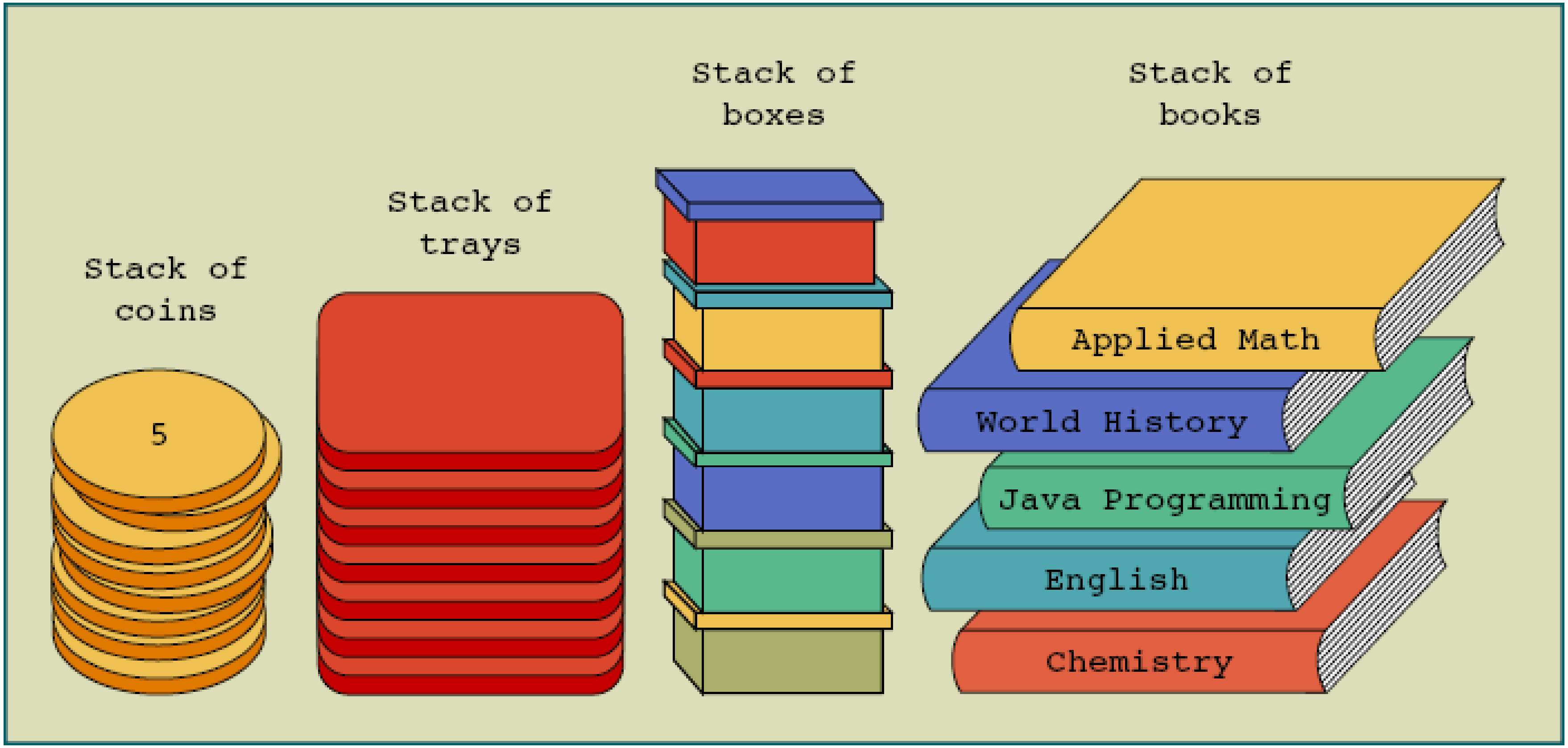
Data Structure
Lab 5
Stacks

CSCI207

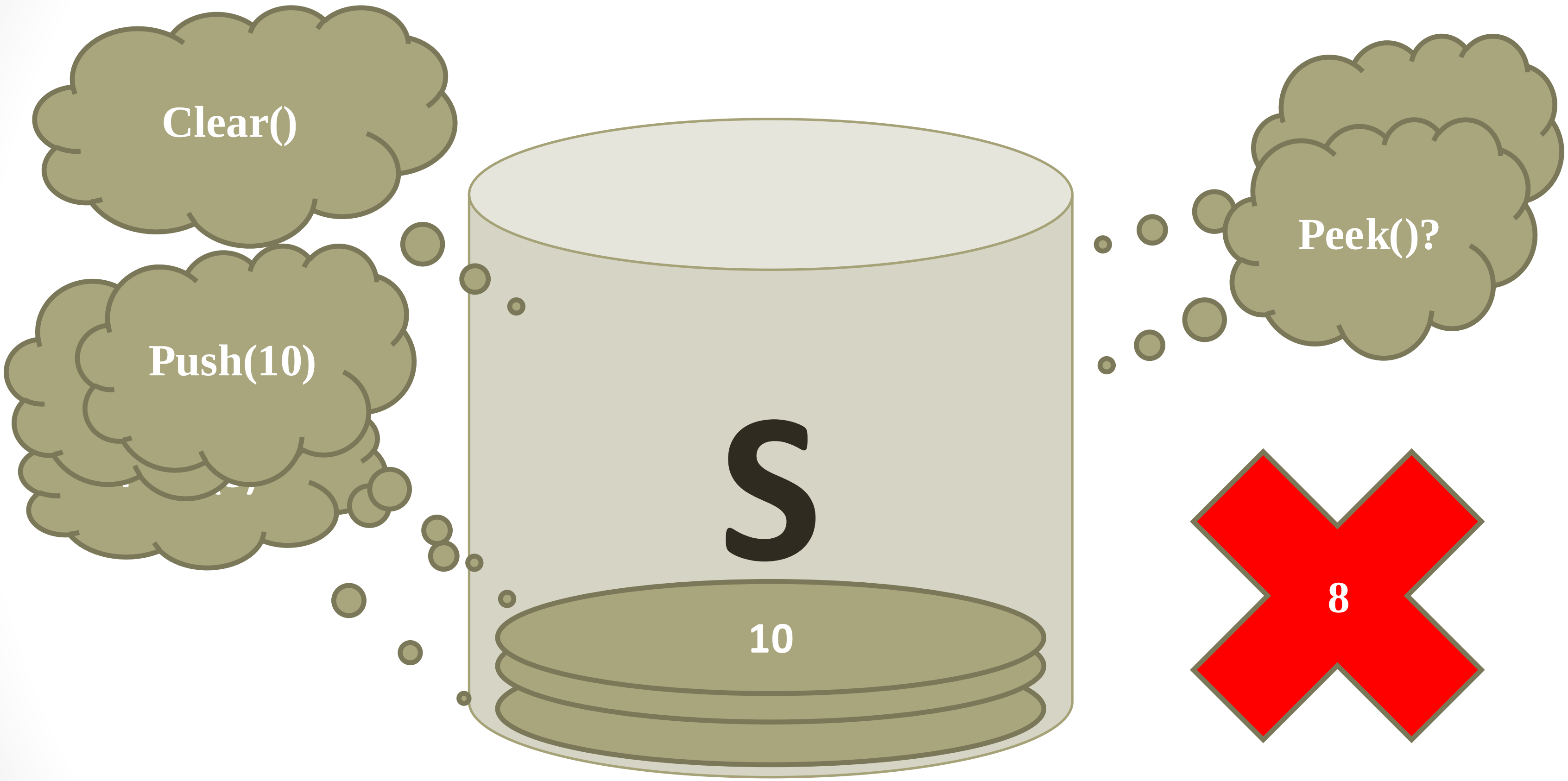
Stacks


Stack of
coins
5
Stack of
trays
Stack of
boxes
Stack of
books
Applied Math
World History
Java Programming
English
Chemistry

Stacks

- Stacks are also called **Last In First Out** (**LIFO**) data structures
- **Operations** performed on stacks
  - **Push**: adds an element to the stack
  - **Pop**: removes an element from the stack
  - **Peek**: looks at the top element of the stack

Stack Representation **LIFO**



Let’s implement

- First we need to create class stack that has members:
  - Size : int → the max size you need
  - Top : int → the last element in the stack
  - List : array of int values(or any data type we need)
- Default constructor
  - Set size = 100
  - Set top = -1 → that refer to the stack is empty
  - Create list of size 100

Let’s implement

- Parameterized constructor that take size parameter
  - Set size = the size you got from parameter
  - Set top = -1 → that refer to the stack is empty
  - Create list it's size that you got from parameter
- Initialization function → to initialize all element of list
- Push function → to add new element
- Pop function → return the last element and delete it
- Peek function → just return last element
- Clear function → remove all element

Let's implement

```
class Stack
{
    private:
        int maxStackSize;
        int stackTop;
        int *list;
    public:
        Stack()
        {
            maxStackSize = 100;
            stackTop = -1; //set stackTop to -1
            //create the array
            list = new int[maxStackSize];
        }
        Stack(int S)
        {
            maxStackSize = S;
            stackTop = -1; //set stackTop to -1
            //create the array
            list = new int[maxStackSize];
        }
```

Let's implement

```
void initializeStack()
{
    for (int i = 0; i < stackTop; i++)
        list[i] = NULL;
    stackTop = -1;
}
bool isEmptyStack()
{
    return (stackTop == -1);
}
bool isFullStack()
{
    return (stackTop == (maxStackSize-1));
}
void push(int newItem)
{
    if (isFullStack())
    {
        cout << "Cannot add->the stack is full" << endl;
          return;
    }
    stackTop++;              //increment stackTop
    list[stackTop] = newItem; //add newItem
}
```

Let’s implement

```
int pop()
{
    if (isEmptyStack())
      {
        cout << "The stack is empty !!!" << endl;
        return -1;
    }
      return list[stackTop--];
}
int peek()
{
    if (isEmptyStack())
    {
        cout << "The stack is empty !!!" << endl;
        return NULL;
    }
    return list[stackTop];
}
```

Let's implement

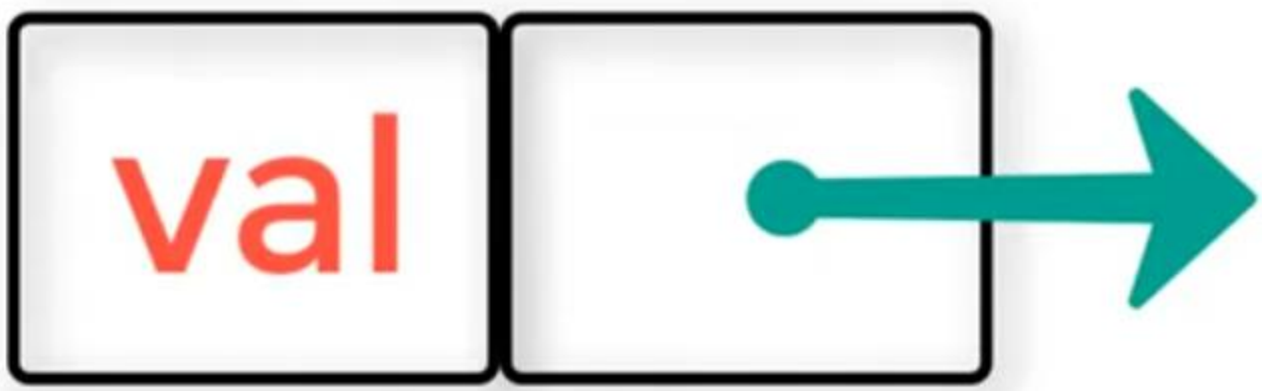
```
        void clear()
        {
            if (isEmptyStack())
            {
                cout << "The stack is empty !!!" << endl;
            }
            while(!isEmptyStack())
            {
                pop();
            }
        }
};
int main()
{
    Stack MyStack (10);  // makes new stack of size 10.
    MyStack.push(5);          // push items onto stack
    MyStack.push(8);       //  These are ‘logical’ operations!!!
    MyStack.pop();
    MyStack.push(10);
    cout<< MyStack.peek()<<endl
    MyStack.clear();

    return 0;
}
```

Stack using linked list

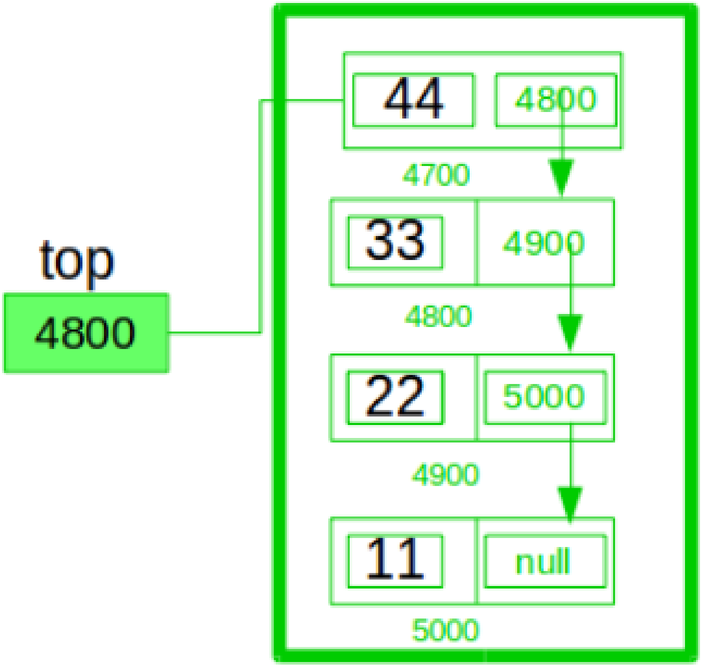
- **Drawbacks of Implementing a Stack Using Arrays**
  - **Fixed Size Limitation**
  - Inefficient Memory Management
  - Costly Resizing Operations
  - Must know max number of values at compile time
  - Arrays hold data of the same data type

Stack using linked list

- **Each item(node) in stack** contains **:-**
- A piece of data (any type)
- Pointer to the previous node in the stack
- 

Stack using linked list

- Pointer of the first node point to Null
- Each node contain the address of the previous node
- Top pointer contain the address of the last node in the stack



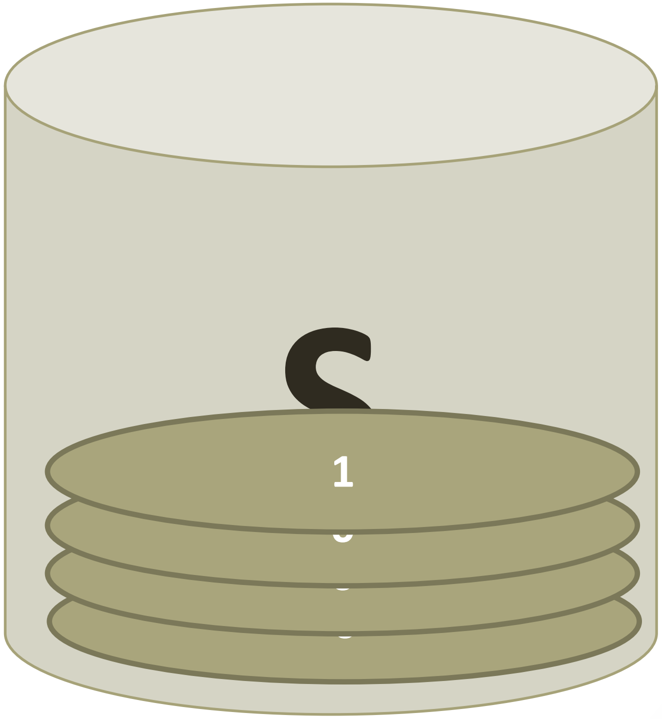
Stack Operations

- **Push**: adds an element to the stack
- **Pop**: removes an element from the stack
- **Peek**: looks at the top element of the stack

Check the Implementation.cpp file

Task : DECIMAL TO BINARY CONVERTER

- Implement a function that takes input parameters (decimal number) and works on it using Stack to convert it into a binary number.
- EX: to convert 8 to binary
  - 8 /2 → 4 with reminder 0
  - 4/2 → 2 with reminder 0
  - 2/2 → 1 with reminder 0
  - 1/2 → 0 with reminder 1
  - So 8 become 1000



THANK YOU