# Data Structures

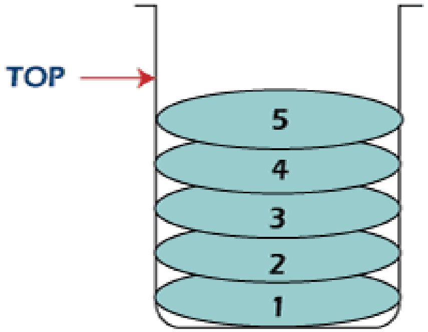
Fall 2023

## 8. Stack

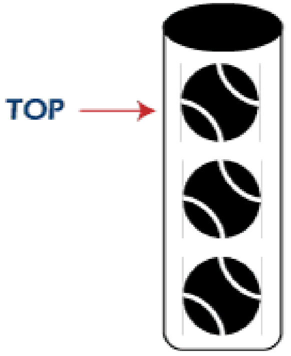
# Stack

- A stack is a special kind of list in which all insertions and deletions are made at one end, called top
- Last In First Out (LIFO)

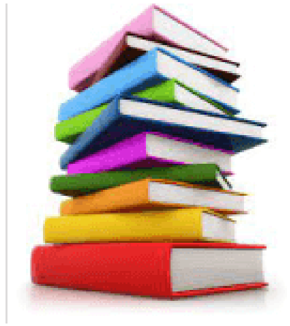
Stack of Coins



Stack of Plates



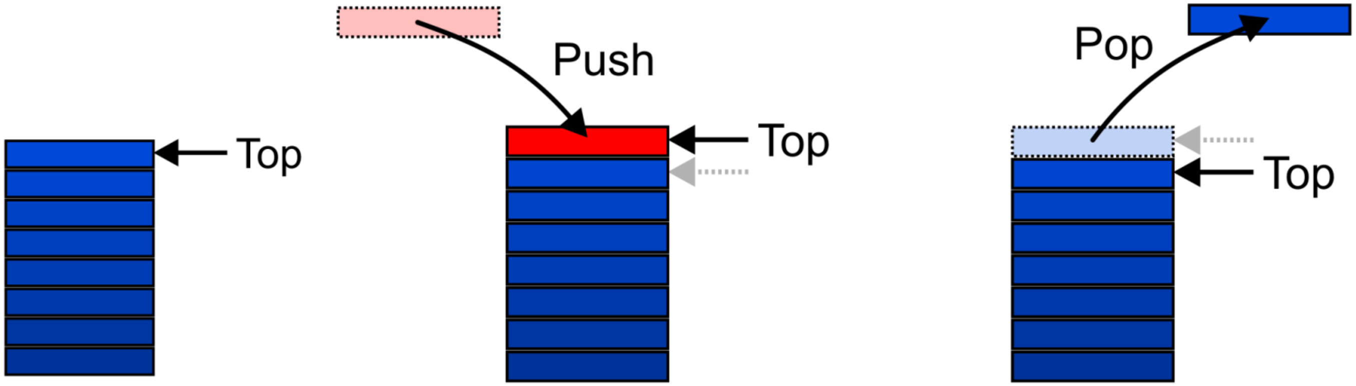
Can of Tennis Balls



Stack of Books

# Stack ADT – Operations (1)

- The fundamental operations on a stack
  - **Push**: Insert at top
  - **Pop**: Delete from top
- Graphically, the stack operations are viewed as follows:



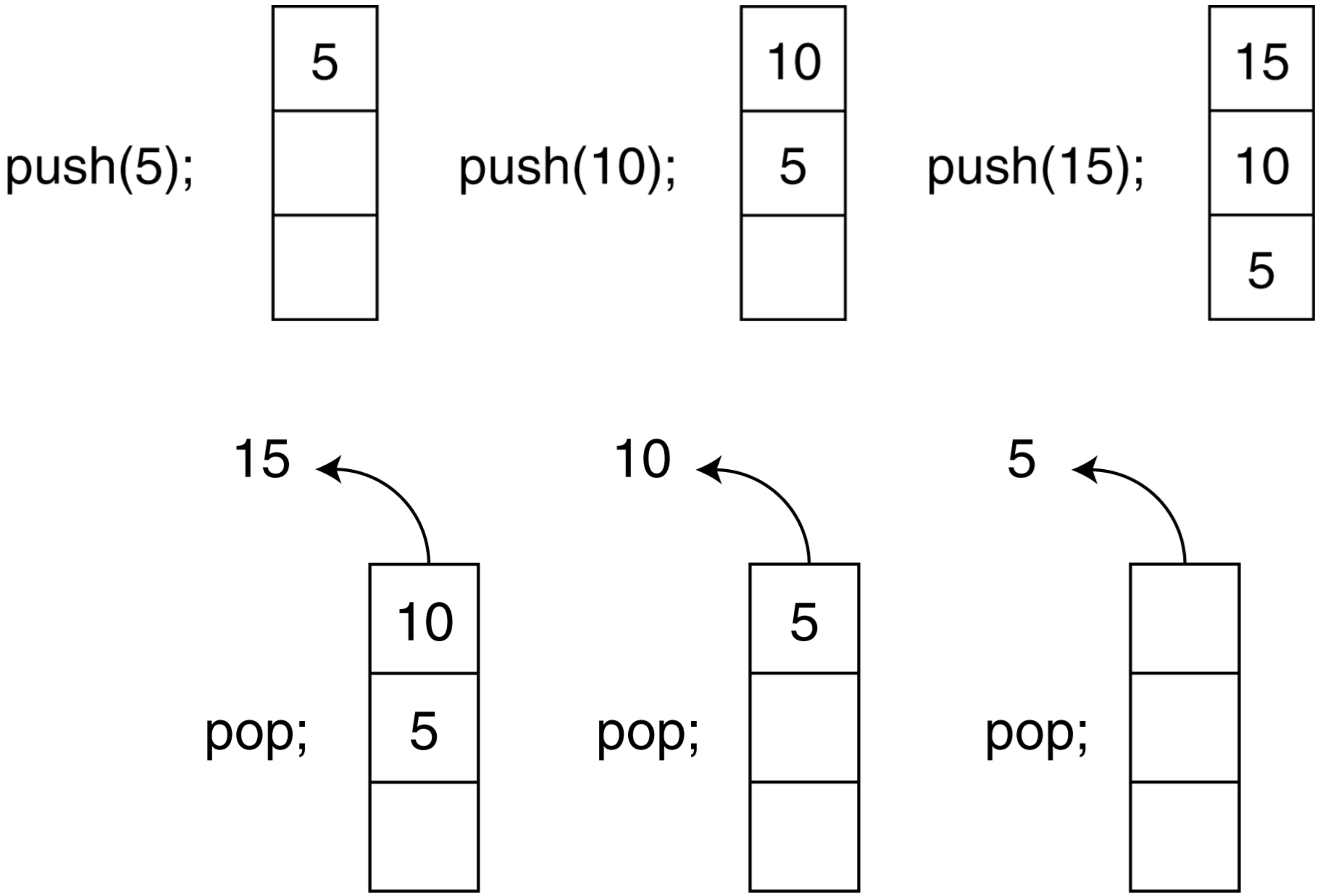
# Stack ADT

- Stack ADT emphasizes specific operations
  - Uses an explicit linear ordering
  - Insertions and removals are performed individually
  - Inserted objects are pushed onto the stack
  - Top of the stack is the most recently pushed object onto the stack
  - When an object is popped from the stack, the current top is erased

# Stack ADT – Operations

- MAKENULL(S)
  - Make Stack S be an empty stack
- TOP(S)
  - Return the element at the top of stack S
- POP(S)
  - Remove the top element of the stack
- PUSH(S,x)
  - Insert the element x at the top of the stack
- EMPTY(S)
  - Return true if S is an empty stack and return false otherwise

# Push and Pop Operations of Stack



push(5);

push(10);

push(15);

pop;

pop;

pop;

# Static and Dynamic Stacks

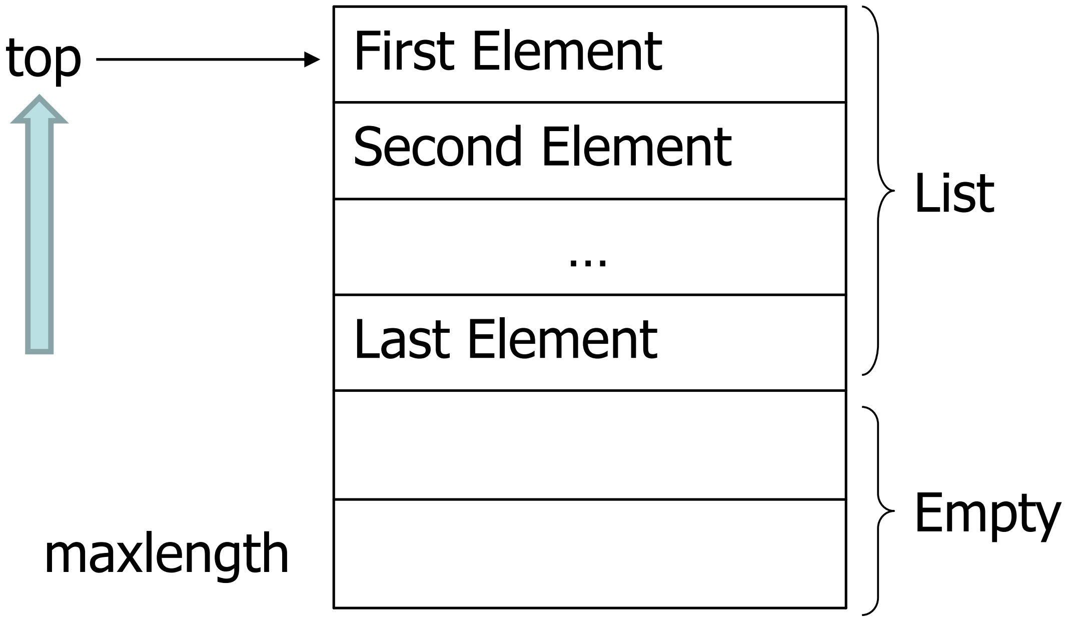
- Two possible implementations of stack data structure
  - **Static** (fixed size) implementation using arrays
  - **Dynamic** implementation using linked lists

# Array-based Implementation

# Array Implementation – First Solution (1)

- Elements are stored in contiguous cells of an array

- New elements can be inserted to the top of the list



top

First Element

Second Element

…

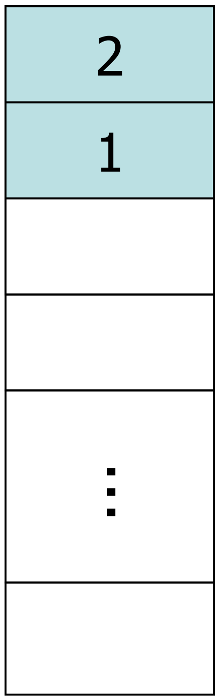
Last Element

List

Empty

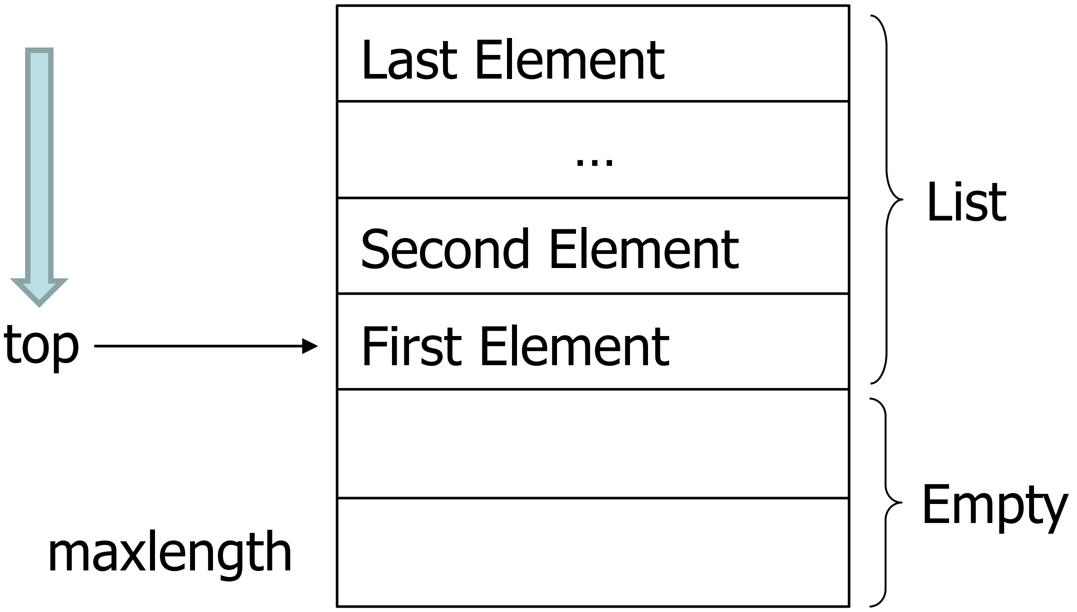
maxlength

# Array Implementation – First Solution (2)



- Problem
  - Every PUSH and POP requires moving the entire array up and down

# Array Implementation – Better Solution (2)



Idea: Anchor the top of the stack at the empty end of the array

# Array Implementation – Code (1)

```
class Stack
{
    private:
        int *stackArray;
        int stackSize;
        int top;

    public:
        Stack(int size) {
            stackArray = new int[size];
            stackSize = size;
            top = -1;
        }
};
```

# Array Implementation – Code (2)

- `isFull` function

```
bool isFull()
{
        bool status;
        if (top == stackSize - 1)
                status = true;
        else
                status = false;
        return status;
}
```

- `isEmpty` function

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

# Array Implementation – Code (3)

- push function inserts the argument v onto the stack

```
void push(int v)
{
    if (isFull())
    {
        cout << "The stack is full.\n";
    }
    else
    {
        top++;
        stackArray[top] = v;
    }
}
```

# Array Implementation – Code (4)

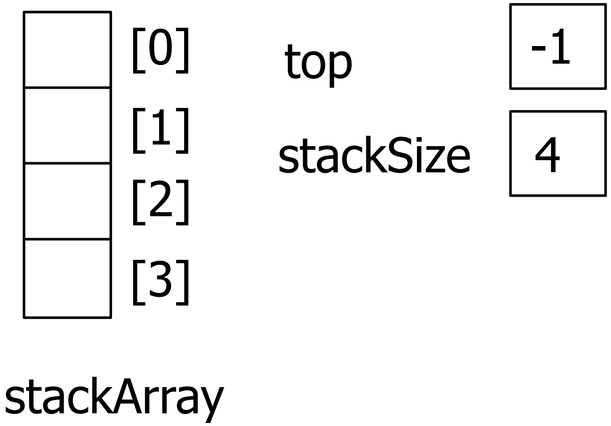
- pop function removes the value from top of the stack and returns it

```
int pop()
{
    int v = -1;
    if (isEmpty())
    {
        cout << "The stack is empty.\n";
    }
    else
    {
        v = stackArray[top];
        top--;
    }
    return v;
}
```

# Using Stack (1)

```
void main()
{
   Stack stack(4);

}
```



[0]

[1]

[2]

[3]

stackArray

top -1

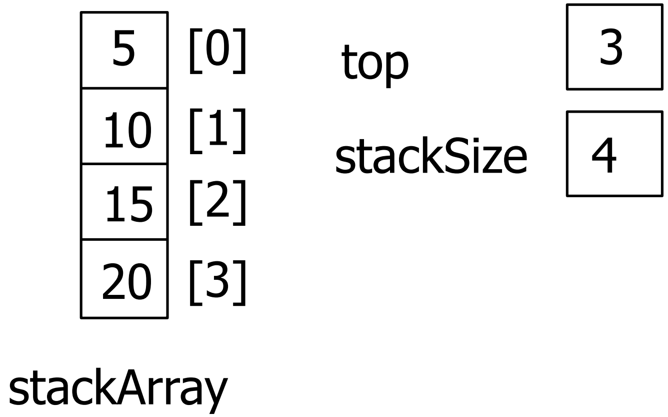
stackSize 4

# Using Stack (2)

```
void main()
{
   Stack stack(4);

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);

}
```



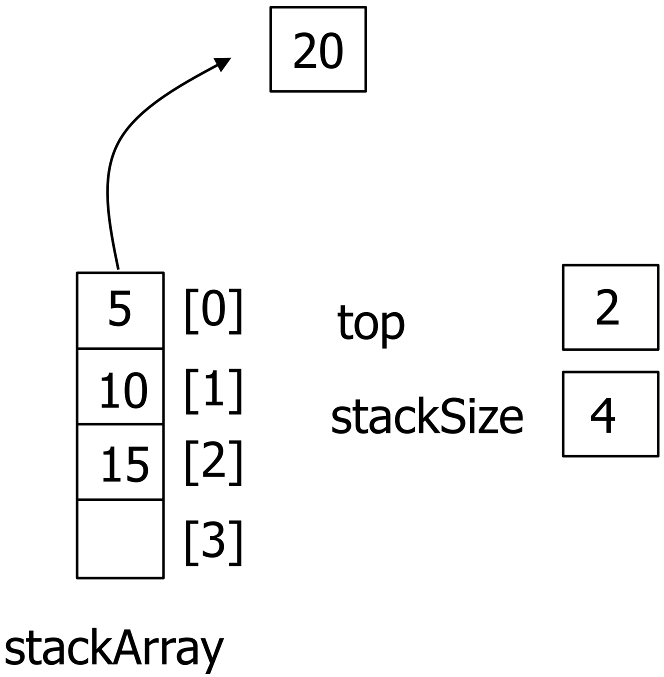
# Using Stack (3)

```
void main()
{
   Stack stack(4);

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);

   cout << "Popping...\n";
   cout << stack.pop() << endl;

}
```



# Using Stack (4)

```
void main()
{
   Stack stack(4);

   cout << "Pushing Integers\n";
   stack.push(5);
   stack.push(10);
   stack.push(15);
   stack.push(20);

   cout << "Popping...\n";
   cout << stack.pop() << endl;
   cout << stack.pop() << endl;
   cout << stack.pop() << endl;
   cout << stack.pop() << endl;
}
```

**Output:**

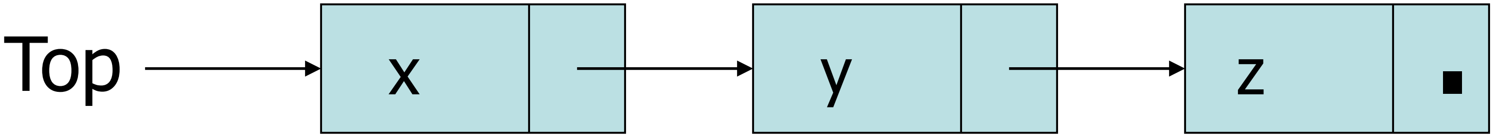
```
Pushing Integers
Popping...
20
15
10
5
```

# Pointer-based Implementation

# Pointer-based Implementation of Stacks

- Stack can expand or shrink with each push or pop operation

- Push and pop operate only on the header cell, i.e., the first cell of  the list



# Pointer Implementation – Code (1)

```
struct node
{
      int data;
      node *next;
};

class Stack
{
      private:
            node *top;
}
```

# Pointer Implementation – Code (2)

- `isEmpty` function returns true if the stack is empty

```
bool isEmpty()
{
    if (top == NULL)
           return true;
    else
           return false;
}
```

# Pointer Implementation – Code (3)

- push function inserts a node at the top(head) of the stack

```
void push(int v)
{
   node *newptr;
   newptr = new node;

   newptr->data = v;
   newptr->next = top;

   top = newptr;
}c
```

# Pointer Implementation – Code (4)

- pop function deletes the node from the top of the stack and returns it

```
int pop()
{
   int v = -1;
   if (isEmpty())
      cout << “Stack is empty \n”;

   temp = top;
   v = top->data;
   top = top->next;

   delete temp;
   return v;
}
```

# Any Question So Far?