CS-218
DATA STRUCTURE

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

STACKS

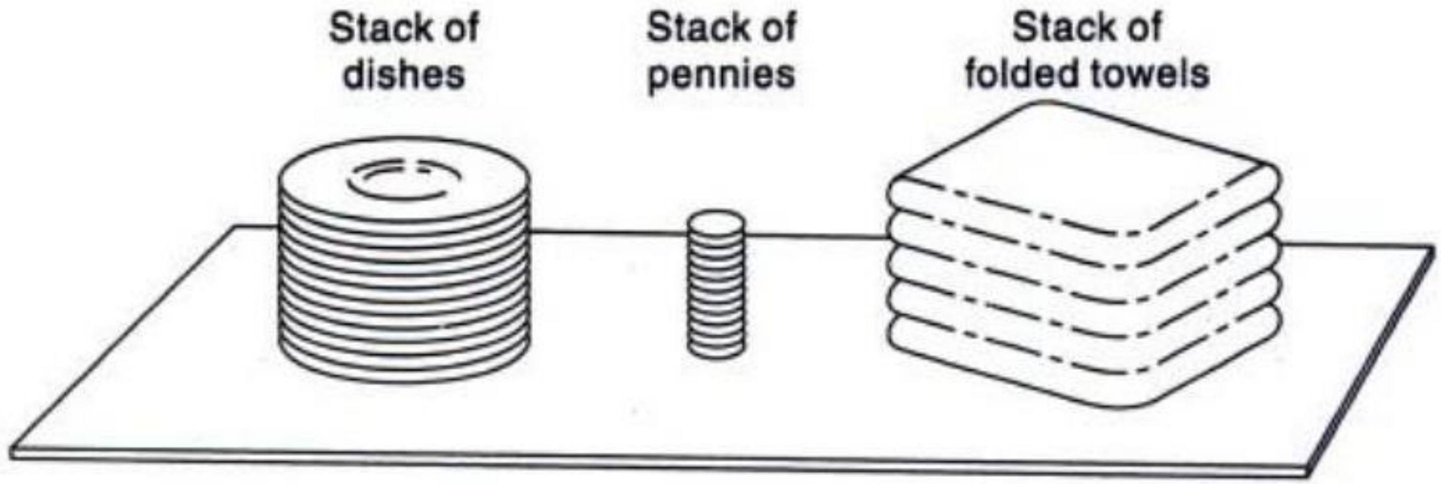
Stack

- "A ***Stack*** is a special kind of list in which all insertions and deletions take place at one end, called the ***Top***"
- Other Names
  - Pushdown List
  - Last In First Out (LIFO)

Stack

Examples:

- Folded towels on shelf
- Dishes on a shelf
- Pennies on shelf



Common Operations

1. **MAKENULL(S):** Make Stack S be an empty stack.
2. **TOP(S):** Return the element at the top of stack S.
3. **POP(S):** Remove the top element of the stack.
4. **PUSH(S):** Insert the element x at the top of the stack.
5. **ISEMPTY(S):** Return true if S is an empty stack; return false otherwise.

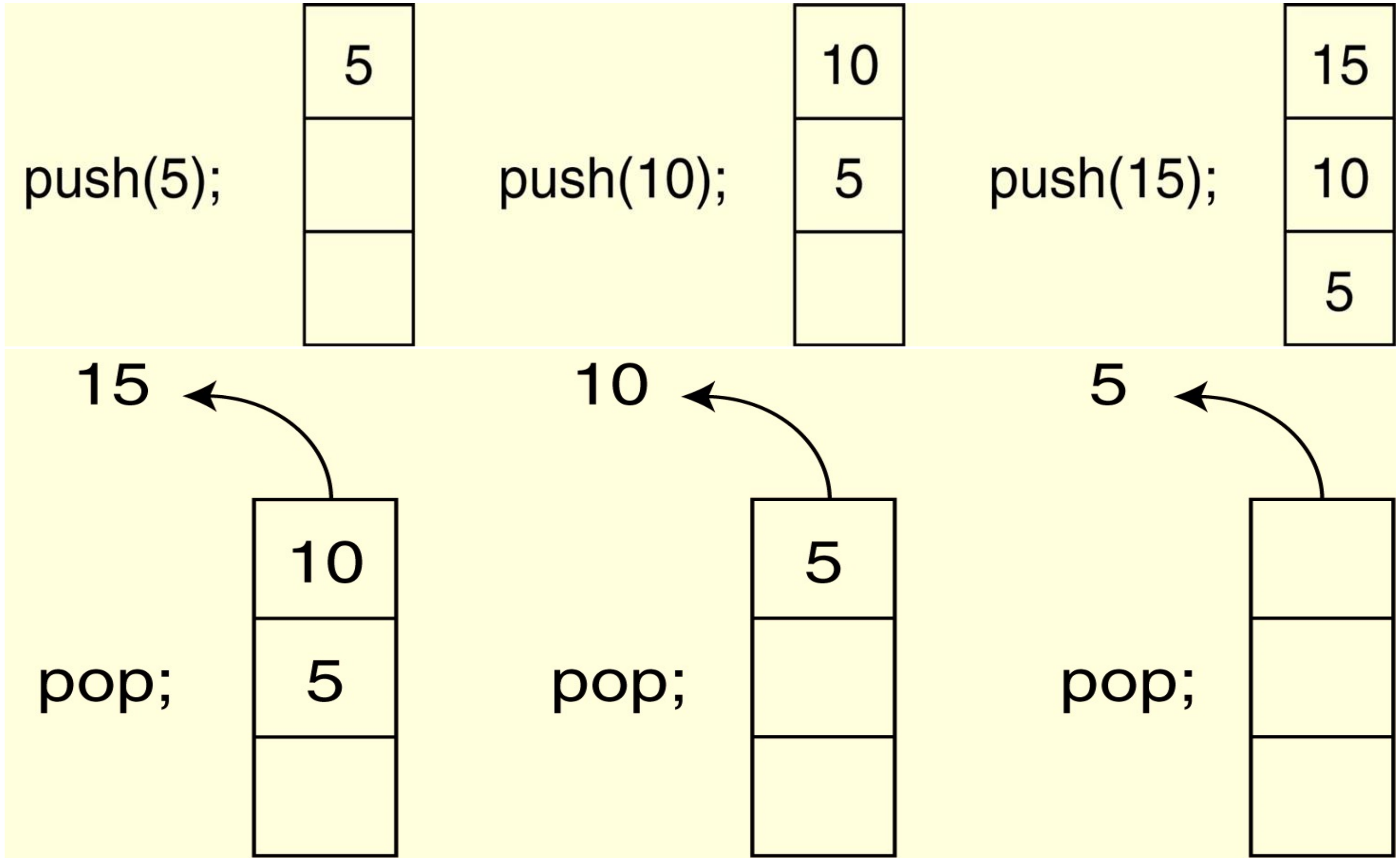
Static and Dynamic Stacks

- There are two kinds of stack data structure,

a) **Static**, i.e. they have a **fixed size,** and are *implemented as* **arrays.**

b) **Dynamic**, i.e. they **grow in size** as needed, and *implemented as* **linked lists**

Common Operations



ARRAY IMPLEMENTATION OF STACK

A Stack Class

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

public:
      IntStack(int);
      void push(int);
      void pop(int &);
      bool isFull(void);
      bool isEmpty(void);
};
```

Implementation

```
//*******************
//    Constructor   *
//*******************
    IntStack::IntStack(int size){
    stackArray = new int[size];
    stackSize = size;
    top = -1;
}
```

Implementation … isEmpty

```
//***********************************************
// Member funciton isEmpty returns true if the stack
// is empty, or false otherwise.*
//************************************************

bool IntStack::isEmpty(void){
      bool status;
      if (top == -1)
            status = true;
      else
            status = false;
      return status;
}
```

Implementation … **isFull**

```
//*****************************************************
// Member function isFull returns true if the stack *
// is full, or false otherwise.                     *
//*****************************************************

bool IntStack::isFull(void){
      bool status;
      if (top == stackSize - 1)
            status = true;
      else
            status = false;

      return status;
}
```

Implementation … **Push**

```
// Push function pushes the argument onto
// the stack.
void IntStack::push(int num){
        if (isFull())
                cout << "The stack is full.\n";
        else{
                top++;
                stackArray[top] = num;
        }
}
```

Implementation … **Pop**

```
// Pop function pops the value at the top
// of the stack off, and copies it into the variable
// passed as an argument.
void IntStack::pop(int &num){




}
```

Implementation ... Pop

```
// Pop function pops the value at the top
// of the stack off, and copies it into the variable
// passed as an argument.
void IntStack::pop(int &num){
        if (isEmpty())
                cout << "The stack is empty.\n";
        else{
                num = stackArray[top];
                top--;
        }
}
```

Implementation … main

```
int main(void){
      IntStack stack(5);
      int catchVar;

      cout << "Pushing 5\n";
      stack.push(5);
      cout << "Pushing 10\n";
      stack.push(10);
      cout << "Pushing 15\n";
      stack.push(15);
      cout << "Pushing 20\n";
      stack.push(20);
      cout << "Pushing 25\n";
      stack.push(25);

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

   stack.pop(catchVar);
   cout << catchVar << endl;

   return 0;
}
```

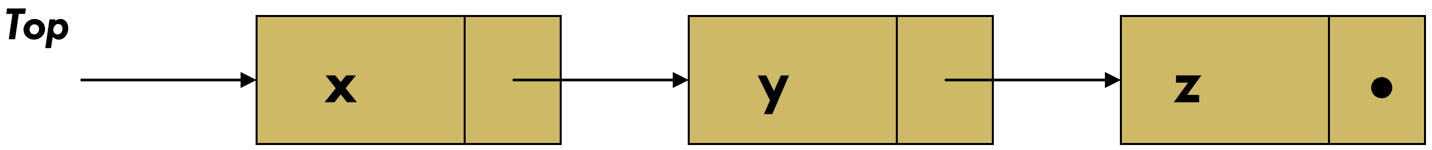
Implementation … output

```
Pushing 5
Pushing 10
Pushing 15
Pushing 20
Pushing 25
Popping...
25
20
15
10
5
```

LINKED LIST IMPLEMENTATION OF STACK

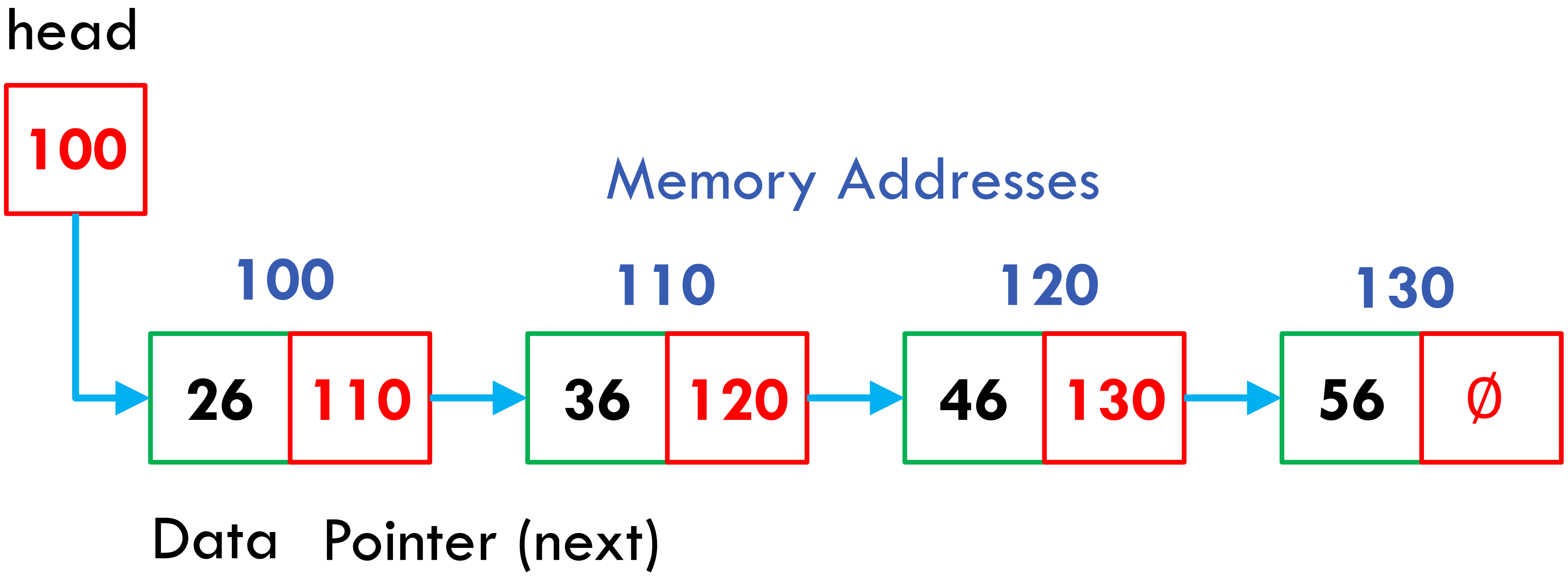
Implementation

- Stack can *expand* or *shrink* with each PUSH or POP operation.
- PUSH and POP operate only on the header cell and the first cell on the list.



Linked List

```
struct Node {
        int     data;           // data
        struct Node*  next;     // pointer to next
};
```

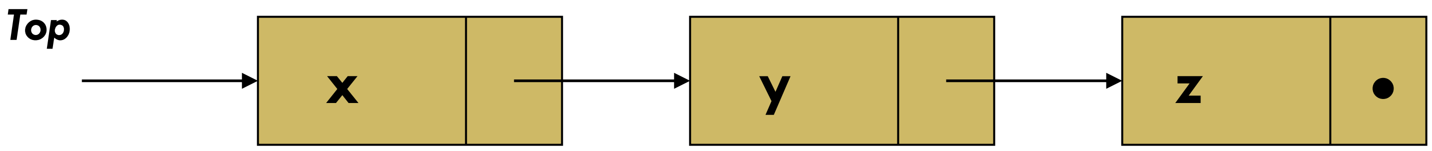


Implementation

```
class Stack{
        struct node{
                int data;
                struct node *next;
        };
node *top;
public:
        void Push(int newelement);
        int Pop(void);
        bool IsEmpty();
};
```

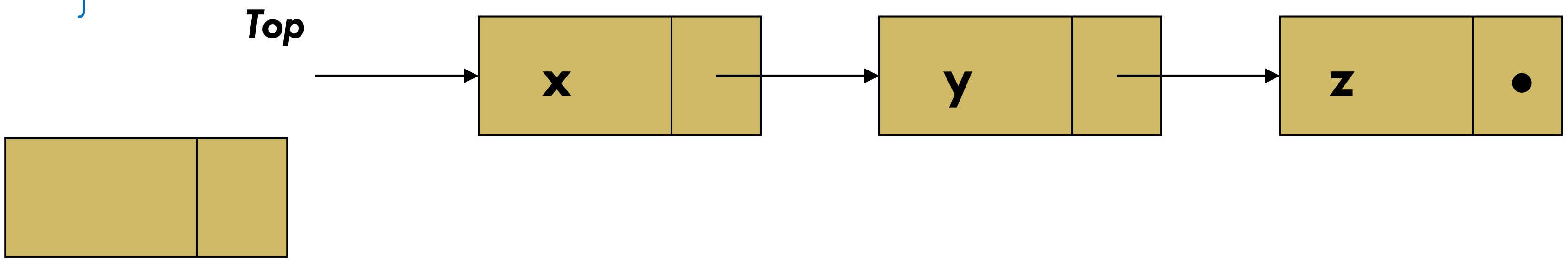
Implementation … isEmpty

```
void Stack::IsEmpty(){
     if (top==NULL)
          return true;
     else
          return false;
}
```



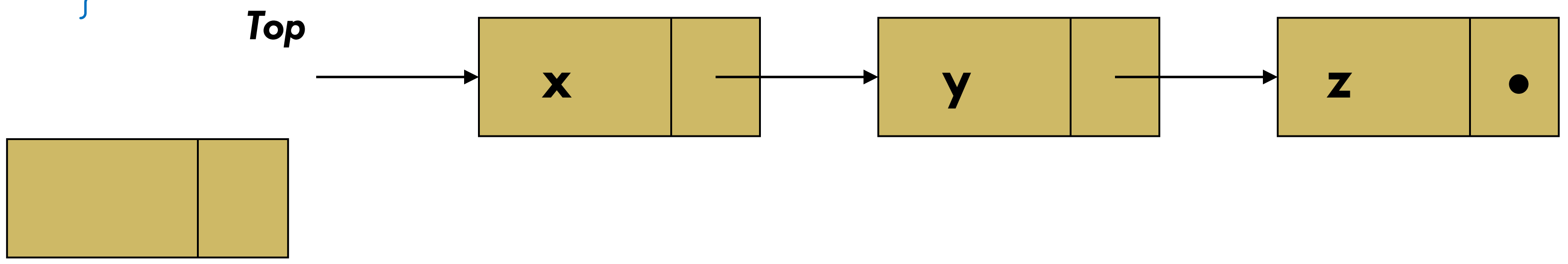
Implementation … Push

```
void Stack::Push(int newelement){



}
```



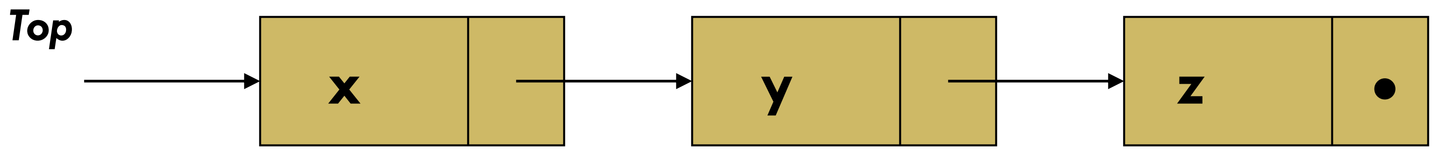
Implementation … Push

```
void Stack::Push(int newelement){
        node *newptr;
        newptr = new node;
        newptr->data = newelement;
        newptr->next = top;
        top = newptr;
}
```



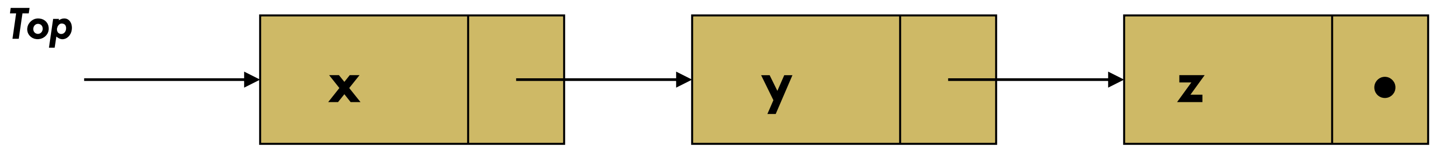
Implementation … Pop

```
int Stack:Pop(void){
      if (IsEmpty()) {
            cout<<"underflow error";
            return;
      }






}
```



Implementation … Pop

```
int Stack:Pop(void){
      if (IsEmpty()) {
            cout<<"underflow error";
            return;
      }
      tempptr = top;
      int returnvalue = top->data;
      top = top->next;
      delete tempptr;
      return returnvalue;
}
```



Implementation … main

```
int main(void){
        Stack stack;
        int catchVar;

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

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

cout <<"\nAttempting again... ";
stack.pop(catchVar);
return 0;
}
```

Implementation … output

```
Pushing 5
Pushing 10
Pushing 15
Popping...
15
10
5

Attempting to pop again... The stack is empty.
```

Reading Materials

- Schaum’s Outlines: Chapter # 6
- D. S. Malik: Chapter # 7
- Nell Dale: Chapter # 4
- Mark A. Weiss: Chapter # 3
- Chapter 7, ADT, Data structures and problem-solving using C++ , Larry Nyhoff.