CSC215

Math and Computer Science

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Stacks

- Last In First Out
    - LIFO

Everyday Stacks

- Stack of plates at all you can eat buffet
- Stack of cups at buffet
- Dixie cup dispenser in your bathroom
- Deposit slips at bank
- ?? Any more

Implementation

- Stack is not ordered by programmer
- All inserts and deletes take place from the same location
  - This is what makes it a LIFO structure
  - No matter what, an insertion will take place at the same location as the deletion does

Stack Operations – Safe Version

- isEmpty() – returns a boolean if the stack is empty or not
- push(item) – adds the new item to the top of the stack.
    - Returns true if the operation was successful, false otherwise.
- pop(item) – removes the item on top of the stack.
    - Copies the data into item.
    - Returns true if the operation was successful, false otherwise
- top(item) – allows the user to peek at the top item of stack
    - Copies the data into item if stack is not empty
    - Returns true if the operation was successful, false otherwise.

Stack Operations

- Size() – returns the number of items in the stack

For debugging purposes only.

- Print() – prints the stack to the screen.

Stack - ADT

```
class mystack
{
    public:
        mystack();
        ~mystack();

        bool push( int item );
        bool pop( int &item );
        bool top( int &item );
        bool isEmpty();
        int size();

    private:
        // Comming
};
```

Private Data

- Could use an array
- Could use a linked list – singly linked
- Could use the STL Vector
- Could use the STL List

- We just need to control where data is inserted into the list.

Array Based

- Assume array of size 10
- Integer to keep track of how many items are in the array
  - Gives us the index of next insertion point
  - Subtract one and it gives the spot for remove

```
int theData[10];
int index;
```

- To Create an empty stack, set index to zero

Linked List - Singly

- Program a singly linked list
- Insert at front to avoid traversals
- Must remove from the front also

```
struct node
{
    int value;
    node *next;
};
node *headptr;
```

STL Vector

```
vector<int> theData;
```

Push	Pop	Top
push_back	pop_back	back
Insert	Erase	back or front

STL List

```
list<int> theData;
```

Push	Pop	Top
push_back	pop_back	back
push_front	pop_front	front
Insert	Erase	Front or back (which end)

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY