CMPSC 24: Lecture 10
Recursion Introduction & Abstract Data Type: Stacks

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

5/3/10 1

Lecture Plan

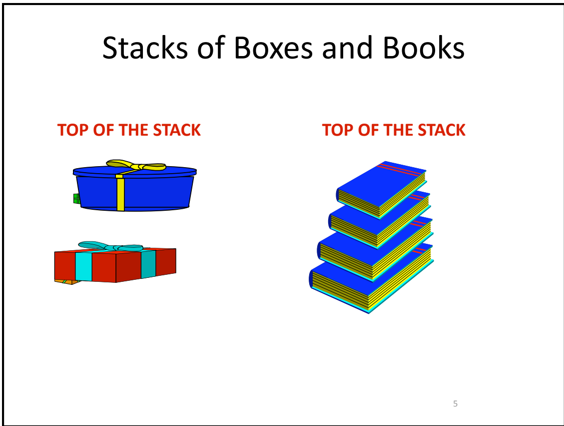
- Binary Search on Sorted Lists
- Stack ADT

2

Stack ADT

- Describe a stack and its operations at a logical level
- Demonstrate the effect of stack operations using a particular implementation of a stack
- Implement the Stack ADT, in both an array-based implementation and a linked implementation
- Applications
  - Matching parentheses
  - Efficient way to save current state
    - Call stack

3



Stacks

- *What do these composite objects all have in common?*

6

Stacks

Stack

An abstract data type in which elements are added and removed from only one end (LIFO)

7

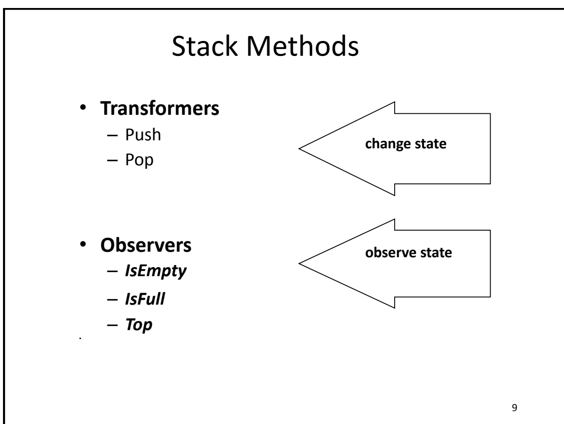
Stacks

•*What operations would be appropriate for a stack?*

8

Stack Methods

- **Transformers**
  - Push
  - Pop
- **Observers**
  - ***IsEmpty***
  - ***IsFull***
  - ***Top***



9

Stacks

```
class StackType
{
  public:
  StackType();
  bool IsEmpty() const;
  bool IsFull() const;
  void Push(ItemType
 item);
  void Pop();
  ItemType Top() const;
```

Logical Level

10

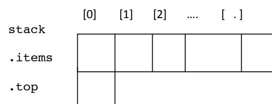
Array-Based Implementation

```
 private:
   int top;
   ItemType items[MAX_ITEMS];
 };
```

stack

.items

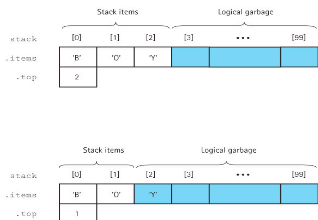
.top



Physical Level

11

Array-Based Implementation



Give a series of operations that could produce this situation

12

Array-Based Implementation

Before we code, we must consider error conditions

- **Stack overflow**
  - The condition that results from trying to **push** an element on to a full stack
- **Stack underflow**
  - The condition that results from trying to **pop** an empty stack

13

Array-Based Implementation

```
StackType::StackType()
{ top = -1; }

bool StackType::IsEmpty() const
{ return ( top = -1); }

bool StackType::IsFull() const
{ return ( top == MAX_ITEMS); }
```

What does ***const*** *mean?*

14

Array-Based Implementation

```
void StackType::Push(ItemType newItem)
{
  if (IsFull())
    throw FullStack();
  top++;
  items[top] = newItem;
}
```

What is ***FullStack()*** *?*

*Code for Top/**Pop**?*

15

Array-Based Implementation

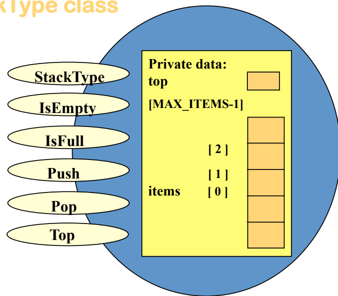
```
void StackType::Pop()
{
  if (IsEmpty())  throw EmptyStack();
  top- -;
}
ItemType StackType::Top() const
{
  if (IsEmpty()) throw EmptyStack();
  return (items[top]);
}
```

What is ***EmptyStack*** *?*

16

Class Interface Diagram

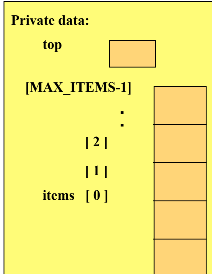
StackType class



17

Tracing Client Code

letter 'V'




```
char       letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
     charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
   charStack.Pop()}
```

18

Tracing Client Code

letter 'V'

Private data:
top -1
[MAX_ITEMS-1]
:
[2]
[1]
items [0]

```
char      letter = 'V';

StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'V'

Private data:
top 0
[MAX_ITEMS-1]
:
[2]
[1]
items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'V'

Private data:
top 1
[MAX_ITEMS-1]
:
[2]
[1] 'C'
items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter: 'V'

Private data:
top: 2

	items
[MAX_ITEMS-1]	
:	
[2]	'S'
[1]	'C'
[0]	'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter: 'V'

Private data:
top: 2

	items
[MAX_ITEMS-1]	
:	
[2]	'S'
[1]	'C'
[0]	'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter: 'V'

Private data:
top: 1

	items
[MAX_ITEMS-1]	
:	
[2]	'S'
[1]	'C'
[0]	'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'V'

Private data:

top 2

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

25

Tracing Client Code

letter 'V'

Private data:

top 2

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

26

Tracing Client Code

letter 'K'

Private data:

top 2

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'K'

Private data:

top 1

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'K'

Private data:

top 1

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code

letter 'C'

Private data:

top 0

[MAX_ITEMS-1]

:

[2] 'K'

[1] 'C'

items [0] 'V'

```
char      letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code


```
char    letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Tracing Client Code


```
char    letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

End of Trace

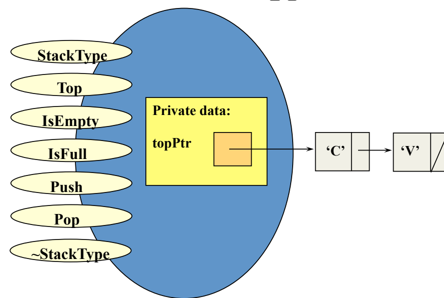

```
char    letter = 'V';
StackType  charStack;
charStack.Push(letter);
charStack.Push('C');
charStack.Push('S');
if ( !charStack.IsEmpty( ))
    charStack.Pop( );
charStack.Push('K');
while (!charStack.IsEmpty( ))
{ letter = charStack.Top();
  charStack.Pop()}
```

Another Stack Implementation

- One advantage of an ADT is that the kind of implementation used can be changed.
- The dynamic array implementation of the stack has a weakness -- the maximum size of the stack is passed to the constructor as parameter.
- Instead we can dynamically allocate the space for each stack element as it is pushed onto the stack.

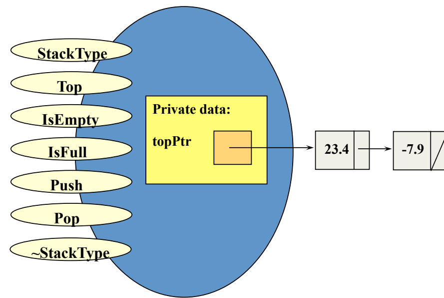
34

`ItemType is char`
`class StackType`

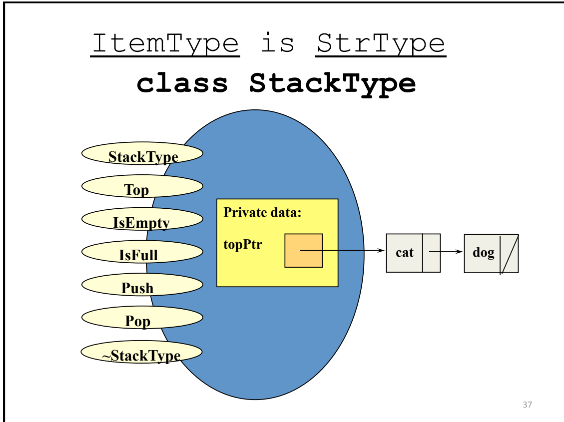


35

`ItemType is float`
`class StackType`



36




```
// DYNAMICALLY LINKED IMPLEMENTATION OF STACK
Struct NodeType;            //Forward declaration
class StackType
{
public:
//Identical to previous implementation
private:
   NodeType* topPtr;
};
.
.
Struct NodeType
{
  ItemType  info;
  NodeType* next;
};
```

38

Implementing Push

```
void StackType::Push ( ItemType newItem )
  // Adds newItem to the top of the stack.
{
   if (IsFull())
     throw FullStack();
   NodeType*  location;
   location = new  NodeType;
   location->info = newItem;
   location->next = topPtr;
   topPtr = location;
}
```

39

Implementing Pop / Top

```
void StackType::Pop()          // Remove top item from Stack.
{
   if (IsEmpty())
      throw EmptyStack();
   else
   {
      NodeType* tempPtr;
      tempPtr = topPtr;
      topPtr = topPtr ->next;
      delete tempPtr;
   }
}
ItemType StackType::Top()
// Returns a copy of the top item in the stack.
{
   if (IsEmpty())
      throw EmptyStack();
   else
      return topPtr->info;
}
```

40

Implementing IsFull

```
bool StackType::IsFull()  const
// Returns true if there is no room for another
// ItemType on the free store; false otherwise
{
   NodeType* location;
   try
   {
      location = new NodeType;
      delete location;
      return false;
   }
   catch(std::bad_alloc exception)
   {
      return true;
   }
}
```

Why is a destructor needed?

When a local stack variable goes out of scope, the memory space for data member topPtr is deallocated. But the nodes that topPtr points to are not automatically deallocated.

A class destructor is used to deallocate the dynamic memory pointed to by the data member.

42

Implementing the Destructor

```
stackType::~StackType()
// Post: stack is empty;
// All items have been deallocated.
{
  NodeType* tempPtr;

  while (topPtr != NULL)
  {
    tempPtr = topPtr;
    topPtr = topPtr-> next;
    delete tempPtr;
  }
}
```

43