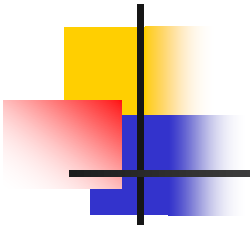
Stack and Queue

Lecture 07

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Adapted partially from Data Structures and Algorithms in Java, M.T. Goodrich, R.Tamassia and M. H. Goldwasser, Sixth Edition,Wiley; Data Structures and Algorithms in C++,Adam Drozdek, 4th Edition, Cengage Learning



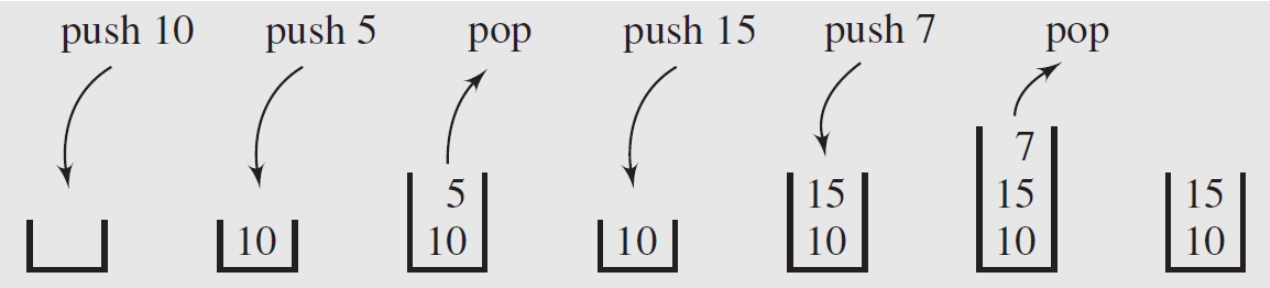
Stacks

- A ***stack*** ??
  - a restricted access **linear data structure**
    - only be accessed at one of its ends for adding and removing data elements
  - e.g., a stack of trays in a cafeteria
    - trays are removed from the top and placed back on the top
  - ***last-in first-out (LIFO)*** structure
- Restrictions
  - only remove items that are available
  - can't add more items if there is no room

} needs your attention during the implementation

Stacks (cont.)

- Stack operations:
  - ***clear( )***: clears the stack
  - ***isEmpty( )***: determines if the stack is empty
  - ***push(el)***: pushes the data item ***el*** onto the top of the stack
  - ***pop( ):*** removes the ***top element*** from the stack
  - ***topEl( )***: returns the value of the ***top element*** of the stack without removing it
- E.g., a series of pushes and pops



Stacks (cont.)

- Implementation of stack
  - natural implementation: ***array***

```
4   #ifndef STACK
5   #define STACK
6
7   #include <vector>
8
9   template<class T, int capacity = 30>
10  class Stack {
11  private:
12      vector<T> pool;
13  };
14
15  #endif
```

header guard: prevent header files from being included multiple times

ref: https://www.educative.io/answers/what-are--sharpifndef-and--sharpdefine-used-for-in-cpp

sequence containers representing arrays that can change in size

ref: https://cplusplus.com/reference/vector/vector/

class template: a class defines something that is independent of the data type

ref: https://www.geeksforgeeks.org/templates-cpp/

Stacks (cont.)

- Implementation of stack: ***array***
  - ***push(el)***: pushes the data item ***el*** onto the top of the stack

```
4  #ifndef STACK
5  #define STACK
6
7  #include <vector>
8
9  template<class T, int capacity = 30>
10 class Stack {
11 public:
12     void push(const T& el) {
13         pool.push_back(el);
14     }
15 private:
16     vector<T> pool;
17 };
18
19 #endif
```

vector::push_back: adds a new element at the end of the vector, after its current last element.

ref: https://cplusplus.com/reference/vector/vector/push_back/

Stacks (cont.)

- Implementation of stack: ***array***
  - ***pop( ):*** removes the top element from the stack

vector::back: returns a reference to the last element in the vector
ref: https://cplusplus.com/reference/vector/vector/back/

vector::pop_back: removes the last element in the vector
ref: https://cplusplus.com/reference/vector/vector/pop_back/

```
4  #ifndef STACK
5  #define STACK
6
7  #include <vector>
8
9  template<class T, int capacity = 30>
10 class Stack {
11 public:
12     T pop() {
13         T el = pool.back();
14         pool.pop_back();
15         return el;
16     }
17 private:
18     vector<T> pool;
19 };
20
21 #endif
```

Stacks (cont.)

- Implementation of stack: ***array***
  - ***topEl( )***: returns the value of the top element of the stack without removing it

```
4   #ifndef STACK
5   #define STACK
6
7   #include <vector>
8
9   template<class T, int capacity = 30>
10  class Stack {
11  public:
12      T& topEl() {
13          return pool.back();
14      }
15  private:
16      vector<T> pool;
17  };
18
19  #endif
```

Stacks (cont.)

- Implementation of stack: ***array***
  - ***isEmpty( )***: determines if the stack is empty

vector::empty: returns whether the vector is empty
ref: https://cplusplus.com/reference/vector/vector/empty/

```
4  #ifndef STACK
5  #define STACK
6
7  #include <vector>
8
9  template<class T, int capacity = 30>
10 class Stack {
11 public:
12     bool isEmpty() const {
13         return pool.empty();
14     }
15 private:
16     vector<T> pool;
17 };
18
19 #endif
```

Stacks (cont.)

- Implementation of stack: ***array***
  - ***clear( )***: clears the stack

vector::reserve: requests that the vector capacity be at least enough to contain capacity elements
ref: https://cplusplus.com/reference/vector/vector/reserve/

vector::clear: removes all elements from the vector
ref: https://cplusplus.com/reference/vector/vector/clear/

```
4  #ifndef STACK
5  #define STACK
6
7  #include <vector>
8
9  template<class T, int capacity = 30>
10 class Stack {
11 public:
12     Stack() {
13         pool.reserve(capacity);
14     }
15     void clear() {
16         pool.clear();
17     }
18 private:
19     vector<T> pool;
20 };
21
22 #endif
```

Stacks (cont.)

- Another implementation of stack: linked list

```
4  #ifndef LL_STACK
5  #define LL_STACK
6
7  #include <list>
8
9  template<class T>
10 class LLStack {
11 private:
12     list<T> lst;
13 };
14
15 #endif
```

lists are sequence containers that allow constant time insert and erase operations anywhere within the sequence, and iteration in both directions.

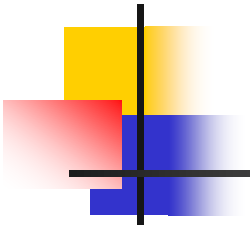
ref: https://cplusplus.com/reference/list/list/

Stacks (cont.)

- Another implementation of stack: linked list

list::member functions
ref: https://cplusplus.com/reference/list/list/

```
11  public:
12      LLStack() {
13      }
14      void clear() {
15          lst.clear();
16      }
17      bool isEmpty() const {
18          return lst.empty();
19      }
20      T& topEl() {
21          return lst.back();
22      }
23      T pop() {
24          T el = lst.back();
25          lst.pop_back();
26          return el;
27      }
28      void push(const T& el) {
29          lst.push_back(el);
30      }
```



Stacks (cont.)

- Particularly useful in situations
  - data have to be stored and retrieved in ***reverse order***
- Numerous applications:
  - balancing **delimiters** in program code, e.g., [, {, (
  - evaluating expressions and parsing syntax
  - etc…

Stacks (cont.)

- Balancing delimiters in program code, e.g., [, {, ((cont.)
  - open delimiters, e.g., '(', '[', '{',
  - close delimiters, e.g., ')', ']', '}'

- matching delimiters

```
a = b + (c - d) * (e - f);
g[10] = h[i[9]] + (j + k) * l;
```

- unmatching delimiters

```
a = b + (c - d) * (e - f));
g[10] = h[i[9]] + j + k) * l;
```

Stacks (cont.)

- A delimiter matching algorithm:
  - first opening delimiter must be matched with the last closing delimiter
  - delimiter can be separated
    - all delimiters following it and preceding its match have been matched
    - e.g.,

```
while (m < (n[8] + o))
```

  - e.g., s = t[5] + u / (v * (w+y));

```
while not end of file {
  if ch is '(', '[', or '{'
    push (ch);
  else if ch is ')', ']', or '}'
    if ch and popped off
       delimiter do not match
      failure;
  read next character ch;
}
if stack is empty
  success;
else
  failure;
```

Stacks (cont.)

- s = t[5] + u / (v * (w + y));
  - checking delimiters: open delimiters, e.g., '(', '[', '{';
    close delimiters, e.g., ')', ']', '}'

Stack	Nonblank Character Read	Input Left
empty		s = t[5] + u / (v * (w + y));
empty	s	= t[5] + u / (v * (w + y));
empty	=	t[5] + u / (v * (w + y));
empty	t	[5] + u / (v * (w + y));
[	[	5] + u / (v * (w + y));
[	5	] + u / (v * (w + y));
empty	]	+ u / (v * (w + y));
empty	+	u / (v * (w + y));
empty	u	/ (v * (w + y));
empty	/	(v * (w + y));
(	(	v * (w + y));

Stacks (cont.)

- s = t[5] + u / (v * (w + y));
  - checking delimiters: open delimiters, e.g., '(', '[', '{';
    close delimiters, e.g., ')', ']', '}'

Stack	Nonblank Character Read	Input Left
(	v	* (w + y));
(	*	(w + y));
((	(	w + y));
((	w	+y));
((	+	y));
((	y	));

Stacks (cont.)

- s = t[5] + u / (v * (w + y));
  - checking delimiters: open delimiters, e.g., '(', '[', '{';
    close delimiters, e.g., ')', ']', '}'

Stack	Nonblank Character Read	Input Left
(	)	);
empty	)	;
empty	;	

Stacks (cont.)

- s = t[5] + u / (v * (w + y));
  - checking delimiters
  - open delimiters, e.g., '(', '[', '{',
  - close delimiters, e.g., ')', ']', '}'

Stack	Nonblank Character Read	Input Left
empty		s = t[5] + u / (v * (w + y));
empty	s	= t[5] + u / (v * (w + y));
empty	=	t[5] + u / (v * (w + y));
empty	t	[5] + u / (v * (w + y));
[	[	5] + u / (v * (w + y));
[	5	] + u / (v * (w + y));
empty	]	+ u / (v * (w + y));
empty	+	u / (v * (w + y));
empty	u	/ (v * (w + y));
empty	/	(v * (w + y));
(	(	v * (w + y));
(	v	* (w + y));
(	*	(w + y));
(		
(	(	w + y));
(		
(	w	+y));
(		
(	+	y));
(		
(	y	));
(	)	);
empty	)	;
empty	;	