Unit #1: Abstract Data Types

CPSC 221: Algorithms and Data Structures

Lars Kotthoff[1]
larsko@cs.ubc.ca

[1]With material from Will Evans, Steve Wolfman, Alan Hu, Ed Knorr, and Kim Voll.

But first…Pointers

also see
http://www.cplusplus.com/doc/tutorial/pointers


£10
Bank of England
I PROMISE TO PAY THE BEARER ON DEMAND THE SUM OF
TEN
Pounds
EK07 443278
London
For the Govr. and Compa. of the BANK of ENGLAND
Chief Cashier
10
EK07 443278


I PROMISE TO PAY THE BEARER ON DEMAND THE SUM OF

Why care about references?

- You go skiing with a friend. You split a granola bar with him. He eats his half. Does it affect yours?
- You make a copy of your lecture notes for a friend. Her dog chews up her copy. Does it affect yours?

Why care about references?

- You go skiing with a friend. You have the hotel make a copy of your hotel key for your friend, so he can leave some stuff there. He trashes the room. Does it affect your room?
- Your parents get an extra credit card for you, on their account. You go wild on a shopping spree. Does this affect your parents' credit?

When does it matter?

Aliasing more than one pointer to the same object

Mutability object that is pointed to can be modified, but the pointer stays the same

In C++

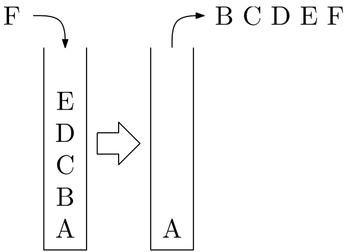
```
int a = 1; // primitive value
int *b = &a; // pointer to the memory location of a

// print a
cout << a << endl; //                                 1
// print the memory location of a
cout << b << endl; //                                 0x7ffce4439874
// print the value of the memory location of a (= a)
cout << *b << endl; //                                1

// modify a
a = 2;

// the pointer is the same...
cout << b << endl; //                                 0x7ffce4439874
// ...the value it point to has changed
cout << *b << endl; //                                2

// modify the value of the memory location
*b = 3;

// a has changed as well
cout << a << endl; //                                 3
```

. . . back to ADTs

Stack ADT

Stack operations

- create
- destroy
- push
- pop
- top
- is_empty



Stack property

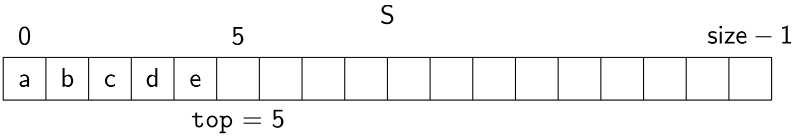
If x is pushed before y is pushed, then x will be popped after y is popped.

LIFO: Last In First Out

Stacks in Practice

- function call stack
- removing recursion
- balancing symbols (parentheses)
- evaluating Reverse Polish Notation
- depth first search

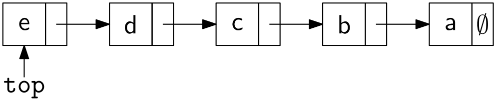
Array Stack Data Structure




```
void push(Object x) {
  assert(!is_full());
  S[top] = x;
  top++;
}

Object top() {
  assert(!is_empty());
  return S[top-1];
}
```

```
Object pop() {
  assert(!is_empty());
  top--;
  return S[top];
}
bool is_empty() {
  return(top == 0);
}
bool is_full() {
  return(top == size);
}
```

Linked List Stack Data Structure




```
void push(Object x) {
  Node *temp = top;
  top = new Node(x);
  top->next = temp;
}

Object top() {
  assert(!is_empty());
  return top->data;
}
```

```
Object pop() {
  assert(!is_empty());
  Object ret = top->data;
  Node *temp = top;
  top = top->next;
  delete temp;
  return ret;
}

bool is_empty() {
  return(top == NULL);
}
```

Deque ADT

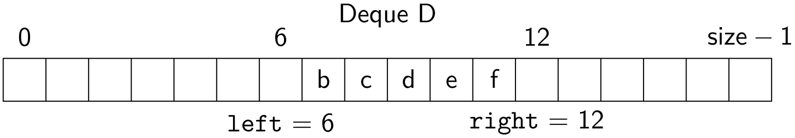
Deque (Double-ended queue) operations

- create/destroy
- pushL/pushR
- popL/popR
- is_empty

Deque property

Deque maintains a list of items.
push/pop adds to/removes from front(L)/back(R) of list.

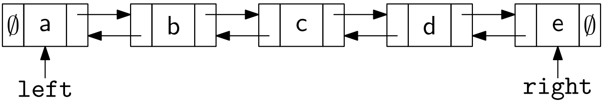
Circular Array Deque Data Structure




```
void pushL(Object x) {
  assert(!is_full());
  D[left] = x;
  left = (left - 1) %
      size;
}
Object popR() {
  assert(!is_empty());
  right = (right - 1) %
       size;
  return D[right];
}
...
```

```
bool is_empty() {
  return(left ==
      (right - 1) % size);
}
bool is_full() {
  return(left ==
      (right + 1) % size);
}
```

Linked List Deque Data Structure




```
void pushL(Object x) {
  if(is_empty())
    left = right = new
        Node(x);
  else {
    left->prev = new
        Node(x);
    left->prev->next =
        left;
    left = left->prev;
  }
}

bool is_empty()
{ return left==NULL; }
```

```
Object popR() {
  assert(!is_empty());
  Object ret = right->
      data;
  Node *temp = right;
  right = right->prev;
  if(right) right->next =
        NULL;
  else left = NULL;
  delete temp;
  return ret;
}
```

Data structures you should already know (a bit)

- Arrays
- Linked lists
- Trees
- Queues
- Stacks