Midterm Review

1 Input/Output

1. ```
   #include <iostream>
   #include <fstream>
   #include <string>
   using namespace std;

   int main(int argc, char *argv[]) {
     ifstream file;
     file.open(argv[1]);
     if(file) {
       return 1;
     }
     string s1, s2;
     while(file >> s1 >> s2) {
       cout <<s2 << endl;
     }
     file.close();
   }
   ```

2. 3
   3

2 Shell

1. all files
   *
   *
   ${HOME}
   /home/calvin
   "${HOME}" single quotes protect everything(including double quotes)
   '/home/calvin' double quotes protect everything except doublequote, backquote, and $VARS(suppresses globbing)

2. ls *.cc ¿ anssignlist.txt

3. cat *.cc — wc
   ```

3 Pointers & References

1. 1
   2
   2

2. differences: a pointer can change the object that it points to. Reference cannot
   similarities: mutate values

4 Linked List

```
Node *add(Node *n1, Node *n2) {
  bool add1 = false;
  Node *frt = nullptr;
  Node *cur = frt;
  while(n1 && n2) {
    if(!frt) {
      frt = new Node;
      cur = frt;
    } else {
      cur->next = new Node;
      cur = cur->next;
      cur->next = nullptr;
    }
    int digit = n1->val + n2->val;
    if(add1) {
      digit += 1;
    }
    if(digit >= 10) {
      add1 = true;
      digit %= 10;
    } else {
      add1 = false;
    }
    cur->val = digit;
    n1 = n1->next;
    n2 = n2->next;
  }
  if(add1) {
    cur->next = new Node;
    cur = cur->next;
    cur->next = nullptr;
    cur->val = 1;
```

```
  }
  return frt;
}
```

5 Stack

1. lecture slides. use linked List

2.
```
string reverse(string str) {
      Stack s;
      for(int i = 0; i < str.length(); i++) {
          push(s, str[i]);
      }
      string str_rev = "";
      for(int i = 0; i < str.length(); i++) {
          str_rev.push_back(top(s));
          pop(s);
      }
  }
```

3. to do List

4. forward: assert forward is not empty
   back: asert back is not empty

6 Queue

1.
```
struct Queue {
  int size;
  int frt;
  int last;
  int *arr;
};

void init(Queue &q, int size) {
  q.size = size;
  q.frt = -1;
  q.last = -1;
  q.arr = new int [size];
}
```

```
void add(Queue &q, int val) {
  if(q.frt == -1) {
    assert(q.last == -1);
    q.frt = 0;
    q.last = 0;
    return;
  }

  if(q.last == q.size - 1) {
    q.last = 0;
  } else {
    q.last++;
  }
  q.arr[q.last] = val;
  if(q.frt == q.last) {
    q.frt++;
  }
}

void remove(Queue &q) {
  assert(q.frt != -1 && q.last != -1);
  if(q.last == q.frt) {
    q.last = -1;
    q.frt = -1;
  } else if(q.frt + 1 == q.size) {
    q.frt = 0;
  } else {
    q.frt++;
  }
}

void print(Queue &q) {
  cout << "frt is " << q.frt << " last is " << q.last << endl;
  int i = q.frt;
  while(true) {
    cout << q.arr[i] << ' ';
    if(i == q.last) break;
    else if(i + 1 == q.size) i = 0;
    else i++;
  }
  cout << endl;
}
```

2. $O(\log n)$ time for all oeprations

7 Testing

8 Trees

1.
```
int height(Node *root, int h) {
  if(!root) return h;
  int hleft = height(root->left, h + 1);
  int hright = height(root->right, h + 1);
  if(hleft > hright) return hleft;
  else return hright;
}
```

2.
```
bool lookup(vector<int> &t, int val) {
  int i = 0;
  while(t.size() > i) {
    if(t[i] == val) return true;
    if(t[i] < val) i = 2 * i + 2;
    else i = 2 * i + 1;
  }
  return false;
}
```

3.
```
Insert O(log2 N)
i.e., insert new value at "end", then fix the tree so it's a heap again
 Insert a new node into the next empty slot
i.e., new left child of node with value 3
 Compare to parent, swap if child is bigger
 Continue to swap with parent until parent is bigger or reach root
 Delete O(log2 N)
i.e., return max value (at the root), then fix the tree so it's a heap again
 Move "last" node into root (and delete from its old position)
 Compare root to children, if not larger than both then swap with
largest child
 Continue until done (parent larger than both children or it's a leaf)
```