Tutorial 5

1 Queue

- A queue is a container of data that enforces a **FIFO** policy on adds/removes
  - e.g. order in Tim Hortons
- Definition of stack: A stack is an ordered container of data that enforces a **LIFO** policy on adds/removes
  - e.g. back/forward button on browser
- implementation of queue using vector(sapce inefficient)
- implementation of queue using linked list

2 C++ arrays

- Static (as in C)
  - Storage allocated on the stack
  - Array bound (N) must be a compile-time constant i.e., it can be determined by just looking at the code and not running the program
- C++ style dynamic arrays
  - Storage is allocated on the heap via a call to new
  - Array bound can be a run-time value (positive integer)
  - Must delete when done, need to say "[ ]"!

3 Testing

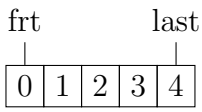
- Black-box Testing
  - Test a class/method against what it is supposed to do, but without looking at how the code achieves it
- White-box Testing
  - Try to exercise all parts of the underlying implementation
  - Test against how the code is built, rather than what it is supposed to do

4 exercises

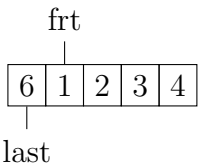
1. Implement queue using the struct below where
   - arr has fixed length "size"
   - frt stores the index of first element
   - last stores the index of last element
   - the next element of arr[size - 1] is arr[0]

```
struct Queue {
  int size;
  int frt;
  int last;
  int *arr;
};
void init(Queue &q, int size);
//if queue is full, replace the oldest element in the queue
void add(Queue &q, int val);
void remove(Queue &q);
void print(Queue &q);
```

For example,



add(q, 6)



2. Write test cases for the following Queue operations:

   - enqueue
   - dequeue
   - containsElement
   - isEmpty

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;


typedef vector<string> Queue;


void enqueue(Queue q, string data);
void dequeue(Queue& q);
bool isEmpty(Queue& q);
bool containsElement(Queue& q, string s);

int main() {
// Put your tests here
}

bool isEmpty(Queue& q) {
return q.empty();
}

void enqueue(Queue q, string data) {
q.push_back(data);
}

void dequeue(Queue& q) {
if(!isEmpty(q)) {
q.pop_back();
}
}

bool containsElement(Queue& q, string s) {
```

```
for(string next : q) {
return next == s;
}

return false;

}
```