similarity ?

difference ?

Announcements

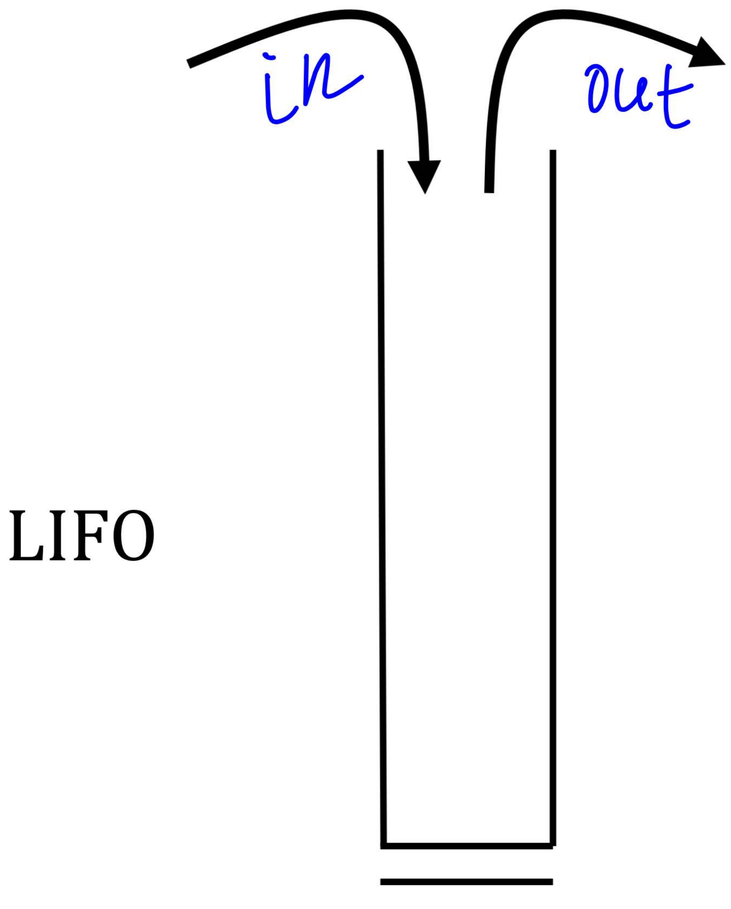
MP4 available, due 10/17, 11:59p. EC due 10/10, 11:59p.

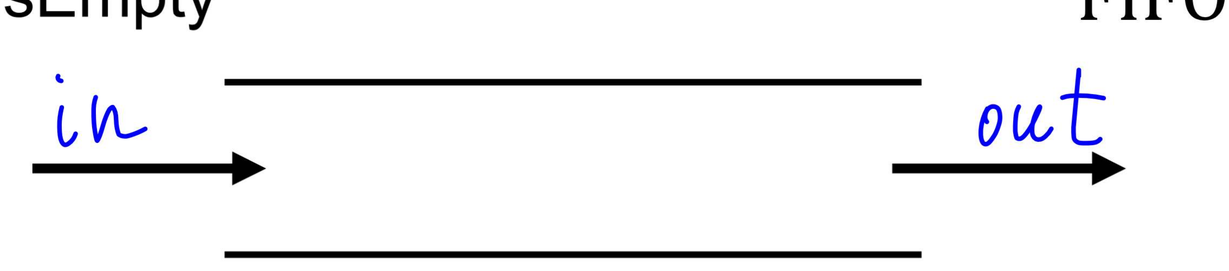
Stack ADT:

- push – in/insert
- pop – out/remove
- isEmpty

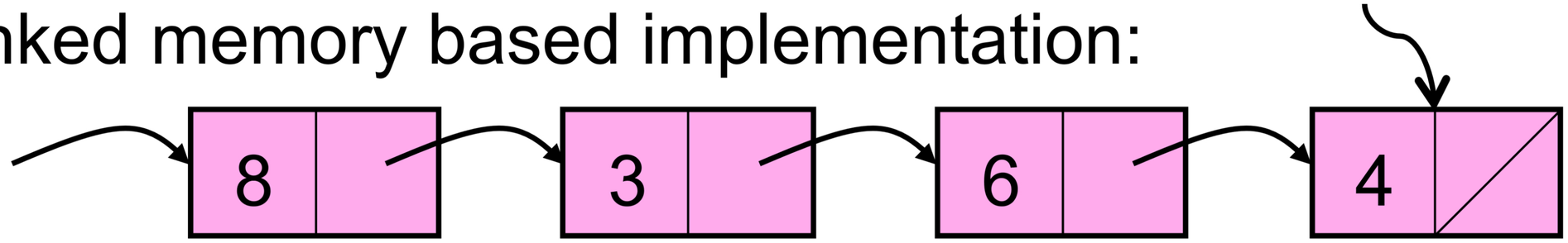
Queue ADT:

- enqueue – in/insert
- dequeue – out/remove
- isEmpty





Queue—linked memory based implementation:




```
template<class SIT>
class Queue {
public:
     // ctors dtor
     bool empty() const;
     void enqueue(const SIT & e);
     SIT dequeue();
private:
     struct queueNode {
          SIT data;
          queueNode * next;
     };
     queueNode * entry;
     queueNode * exit;
     int size;
};
```

Which pointer is “entry” and which is “exit”?

What is running time of enqueue?

What is running time of dequeue?

Queue array based implementation:

```
template<class SIT>
class Queue {
public:
     Queue();
     ~Queue(); // etc.
     bool empty() const;
     void enqueue(const SIT & e);
     SIT dequeue();
private:
     int capacity;//size of array
     int size;//number of items in array
     SIT * items; // array of items
     // maybe some other stuff…
};
```

```
template<class SIT>
Queue<SIT>::Queue(){
     capacity = 8;
     size = 0;
     items = new SIT[capacity];
}
```

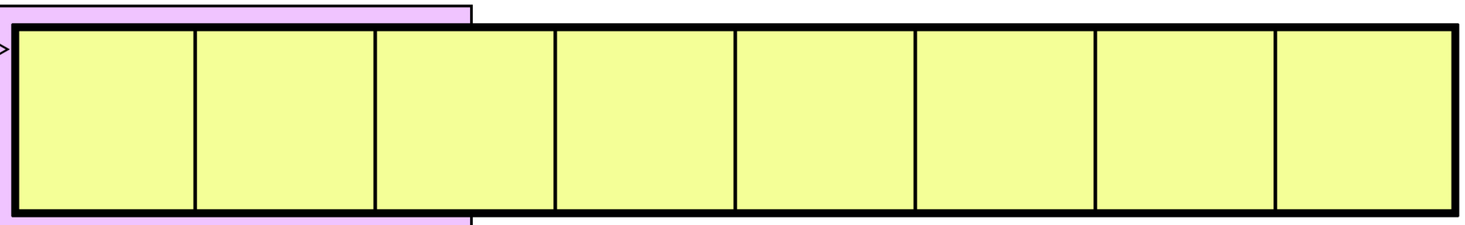
Are we worried about the finite size array?

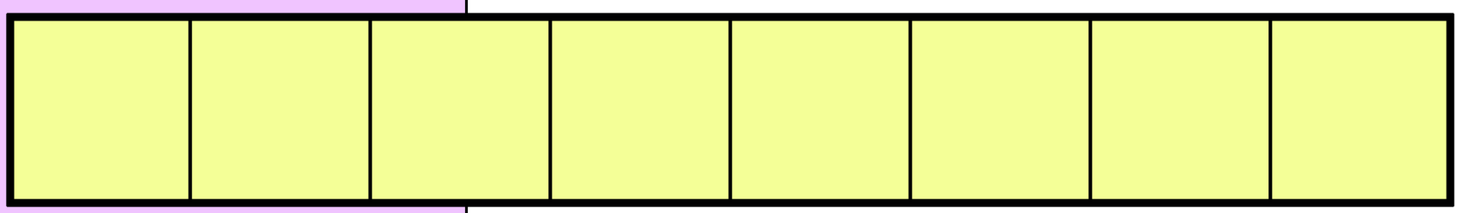
Strategy:

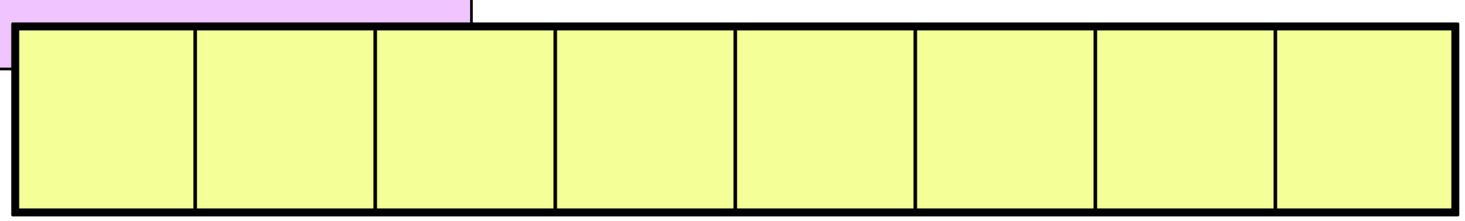
Queue array based implementation:

```
template<class SIT>
class Queue {
public:
     Queue();
     ~Queue(); // etc.
     bool empty() const;
     void enqueue(const SIT & e);
     SIT dequeue();
private:
     int capacity;
     int size;
     SIT * items;


};
```



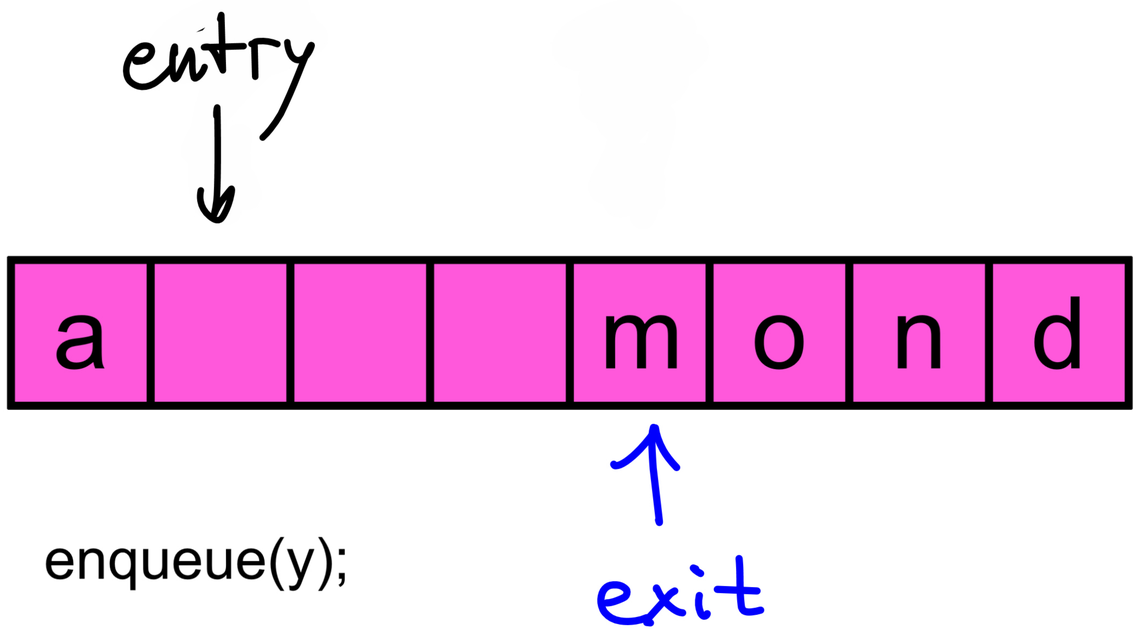




enqueue(3);
enqueue(8);
enqueue(4);
dequeue();
enqueue(7);
dequeue();
dequeue();
enqueue(2);
enqueue(1);
enqueue(3);
enqueue(5);
dequeue();
enqueue(9);

Queue array based implementation:

```
template<class SIT>
class Queue {
public:
     Queue();
     ~Queue(); // etc.
     bool empty() const;
     void enqueue(const SIT & e);
     SIT dequeue();
private:
     int capacity;
     int size;
     SIT * items;
     int entry;
     int exit;
     // some other stuff…
};
```



enqueue(y);

enqueue(i);

enqueue(s);

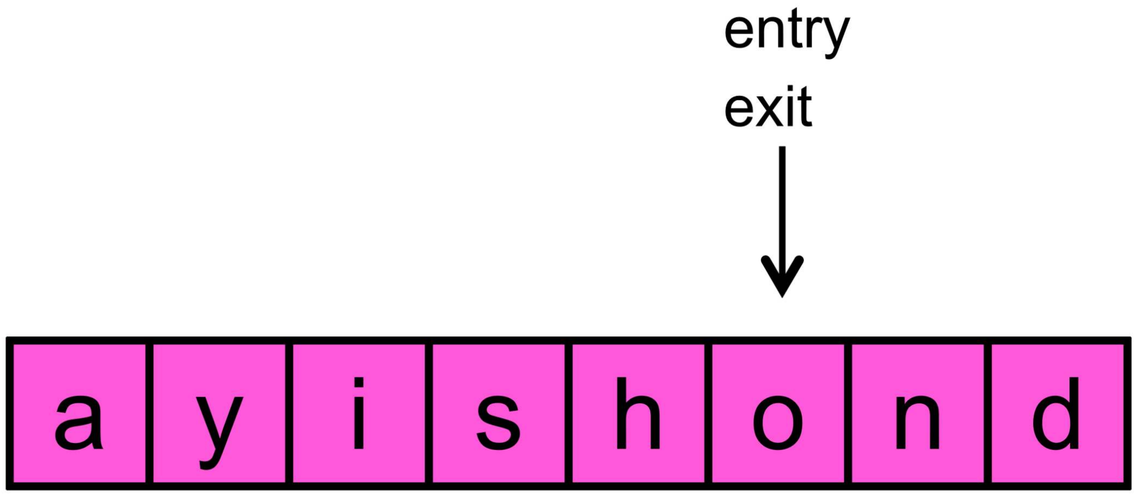
dequeue();

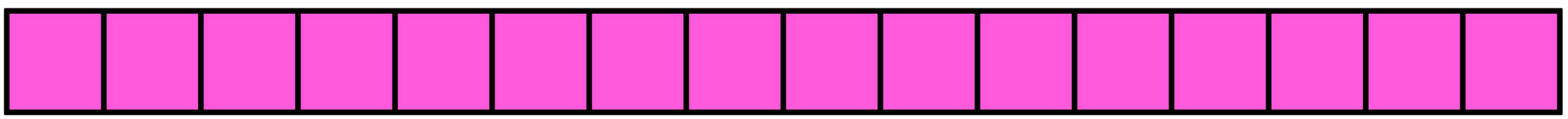
enqueue(h);

enqueue(a);

What if array becomes too sparse?

What if array fills?:





Running time:

enqueue:

dequeue:

Empirical performance proof?

Empirical Proof: Linked Lists vs Arrays

```
#include <iostream>
#include <ctime>
#include <iomanip>
using namespace std;

class StackList{
public:
  StackList(){
    top = NULL;
    size = 0;
  }
  void push(int a){
    listNode * tmp = new listNode(a);
    tmp->next = top;
    top = tmp;
  }
private:
  struct listNode{
    int data;
    listNode * next;
    listNode(int a): data(a), next(NULL){}
  };
  listNode *top;
  int size;
};
```

```
class StackArray{
public:
  StackArray(){
    size = 0;
    capacity = 1;
    items = new int[capacity];
  }
  void resize(){
    capacity = capacity*2;
    int *itemsNew = new int[capacity];
    for (int i = 0; i < size; i++)
      itemsNew[i] = items[i];
    delete []items;
    items = itemsNew;
  }
  void push(int e) {
    if (size == capacity){
      resize();
    }
     items[size] = e;
    size++;
  }
  void print() {
    for (int i = 0; i < size; i++)
      cout << items[i] << " ";
  }

private:
  int * items;
  int capacity;
  int size;
};
```

Empirical Proof: Linked Lists vs Arrays

```
int main() {

  StackArray a;
  StackList b;
  int numOp = 1000000;
  cout << "Number of elements to push: " << numOp << endl;

  cout << fixed;
  cout << setprecision(4);

  // ARRAY WITH RESIZING
  clock_t beginArray = clock();
  for (int i = 0; i < numOp; i++) {
    a.push(i);
  }
  clock_t endArray = clock();

  double timeArray =
    double(endArray - beginArray) / CLOCKS_PER_SEC;
  cout << "array" <<timeArray << endl;


  // LINKED LIST
  clock_t beginList = clock();
  for (int i = 0; i < numOp; i++) {
    b.push(i);
  }
  clock_t endList = clock();

  double timeList =
    double(endList - beginList) / CLOCKS_PER_SEC;
  cout << "list " << timeList << endl;
```