EECS 281: Data Structures and Algorithms

Stacks and Queues

A “list of type T”

- Logical sequential storage
  - an ordered collection of objects of type T
  - each object in the list is called an “element”
- Must have:
  - insert a new element
  - remove a previously inserted element
- Typically have:
  - construct an empty list
  - count the number of elements
- May have:
  - destroy or copy the list
  - reverse one list, concatenate two lists, etc. etc.

Why do you need a list?

- Remember things you have to do
  - list of assignments due this week
- Remember things you already know
  - list of all the people you’ve met at UM
- Break big tasks into smaller tasks
  - find next step in a graph, build up into paths
Oodles of lists

- Many different kinds of lists
  - they differ in the exact details of how insertion/removal work
- Lists we’ll talk about soon
  - stack: remove in reverse order of insertion
  - queue: remove in order of insertion
- Lists we’ll talk about later
  - sorted lists
  - priority queues

Definitions

Abstract Data Type: mathematical model of a data type and operations defined on the model that is accessed only through an interface

- separate the conceptual transformations that our programs perform on data from any particular data structure representation and algorithm implementation

Stack: Definition

- Pushdown stack is a list ADT
  - has one end that is special: “top” of the stack
- comprises five operations:
  - create: make an empty stack (of size N)
  - push: insert a new item on the top
  - pop: remove the item from the top if any
  - empty: returns true if empty, false otherwise
  - full: returns true if full, false otherwise
- last in, first out (aka LIFO)
Stack: Usage

- When you are working on physics, roommate reminds you of 281 homework
- 281 is clearly more important than physics 😊
- Put physics books on the "stack of stuff" on your desk, start working on 281
- When you are done with 281, grab your physics books from the top of the stack, and try to finish it

Stack: Usage

- Cached URLs on your web browser
- Undo operations maintained by editors
- Implementing function calls
  - Run time stack in C++

Usage: Evaluating Postfix Expressions

- Use a stack to evaluate postfix expressions:
  - If the next component of expression is an operand, push value onto the stack
  - If the next component is an operator, then pop its operands from the stack; perform operation; push result back onto the stack
Stack: Concrete Representation 1

First representation: array-based

- A stack is defined as:
  ```
  int size  // the capacity of the stack
  T A[size] // an array named A of type T that can hold size elements
  int next // moves with the top of the stack
  // is the smallest index of an empty slot
  // OR is size if the array is full
  ```

Stack: Concrete Representation 2

Second representation: pointer-based

- A node is defined as:
  ```
  node {
    T element;  // element held here
    node *next;  // NULL if this is last node
  }
  ```

- A stack is defined as:
  ```
  node *top;  // NULL if empty, else top node
  ```

Stack: List-based

```cpp
create() {
  top = NULL
}

bool empty() {
  return (top == NULL)
}

bool full() {
  return false
}

push(T elt){
  node *n = new node
  n->element = elt
  n->next = top
  top = n
}

T pop() {
  if empty()
  throw exception
  T result = top->element
  node *victim = top
  top = victim->next
  delete victim
  return result
}
```
Queue: Definition

- A Queue is a list ADT
  - both ends are special: “head” and “tail”
- comprises five operations
  - create: make an empty queue (of size N)
  - enqueue: put a new item on the tail of the queue
  - dequeue: remove the item from the head, if any
  - empty: return true if queue is empty, else false
  - full: return true if queue is full, else false
- first in, first out (aka FIFO)

Queue: Usage

- You plan to complete assignments in the order that they were assigned
- Each time an assignment is given, it goes on the end of your list
- Each time you complete an assignment, you cross it off and begin the next assignment on the list

Queue: Concrete 1

First representation: array-based

A queue is defined as:

```c
int size // the capacity of queue plus one
T A[size] // an array of T that can hold size elements
int tail // names next empty slot to fill
int head // names next full slot to remove
// if (head == tail) then queue is empty
// if (head == (tail+1)mod size), then queue is full
```
Queue: Array-based

```java
create(int N) {
    size = N+1;
    A = new T[size];
    tail = 0;
    head = 0;
}

bool empty() {
    return head == tail;
}

bool full() {
    return (((tail+1)%size) == head);
}

enqueue(T elt) {
    if full() 
        throw exception 
    A[tail++] = elt 
    tail %= size 
}

dequeue() {
    T result 
    if empty() 
        throw exception 
    result = A[head++]
    head %= size 
    return result 
}
```

Queue: Concrete 2

Second representation: pointer-based

- A node is defined just as before:
- A queue is defined as:
  ```
  node *head // NULL if empty, else head node
  node *tail // NULL if empty, else tail node
  ```

Double-ended Queues

- Also called **deque**
- Richer ADT than both the stack and the queue
- Allows insertion and deletion at both the front and the end of a queue
- Use doubly linked list to implement
Summary: Stacks and Queues

- Lists are a general, abstract concept
- Stacks are a specific type of list
  - insert (push) onto 'top'
  - remove (pop) from 'top'
  - can be implemented as array or linked list
- Queues are a specific type of list
  - insert (enqueue) onto 'bottom'
  - remove (dequeue) from 'top'
  - can be implemented as array or linked list