Lecture 22

Today:

- Queue ADT
- An algorithm that uses a Queue
- Implementing a Queue (with a Linked List)
- Information Hiding & Encapsulation - Part 1
Queue ADT (Review)

Queue ADT: A *queue* is a sequence of data, but the insert and remove operations work on opposite ends of the sequence.

- order is first-in-first-out (FIFO)
- like a line-up

Used in simulations and modeling

- to model sequences of work and their processors, e.g., assembly lines

*Operations Research* (OR)
Queue-Based Searching (Breadth-First Search)

Problem: Find all locations that are reachable from the start, and compute their distance.

Algorithm:
Create an empty queue Q; enqueue start → Q
Initialize all distances ← -1 (unreachable), except distance(start) ← 0
while Q not empty {
    dequeue from Q → current
    if next is neighbour of current and distance(next) == -1 {
        distance(next) = distance(current) + 1
        enqueue next → Q
    }
}

Sample Map:

Rules:
• Numbers represent elevation
• You may only traverse to adjacent grid cells that differ by no more than 2

Distance:
Queue Implementation

Queue Interface:

- a sequence of data in FIFO order
- create()
- enqueue(x)
- dequeue()
- isEmpty()

Implement using a Linked List

- create() and isEmpty() are trivial
- for enqueue(x) and dequeue(), only issue is to decide which end of the list
Queue Implementation: Algorithms

create():
    return LLcreate();

isEmpty(Q):
    return (Q->head == NULL);

enqueue(Q, x):
    LLappend(Q, x);

decqueue(Q):
    return LLremoveHead(Q);

<table>
<thead>
<tr>
<th></th>
<th>head</th>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>remove</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>
typedef LL_t queue_t;

// Creates a pointer to a new empty queue.
// Returns NULL on failure.
queue_t * queue_create(void);

// Recycles a queue
void queue_destroy(queue_t * q);

// Returns 1 iff queue is empty
int queue_isEmpty(queue_t * q);

// Adds element to the back of the queue
void queue_enqueue(queue_t * q, int element);

// Removes element from the front of the queue.
// Undetermined behaviour if queue is empty
int queue_dequeue(queue_t * q);

Information Hiding in C

An invitation for disaster!
Encourages abuse or misuse by calling the linked list functions on the type queue_t *.

Better would be:
typedef struct _queue
queue_t;

which hides all information.
The naming implies that we would or should call these operations only on the type queue_t *.
Marrying Data and Functions

Encapsulation

● bundle related data and operations together

Forge a language construct that marries data and operations together

● use a struct!
● make the functions part of the data type explicitly
  ○ called methods
● similar idea to an object in Python

Adds another level of protection against misuse
typedef struct _queue {
    LL_t * intlist;
} queue_t;

queue_t * queue_create(void);
void queue_destroy(queue_t * q);
int queue_isEmpty(queue_t * q);
void queue_enqueue(queue_t * q, int element);
int queue_dequeue(queue_t * q);

queue_t * queue_create(void);
typedef struct _queue {
    LL_t * intlist;
} queue_t;

queue_t * queue_create(void);

void queue_destroy(queue_t * q);
int queue_isEmpty(queue_t * q);
void queue_enqueue(queue_t * q, int element);
int queue_dequeue(queue_t * q);

typedef struct _queue {
    LL_t * intlist;
    void queue_destroy(struct _queue * q);
    int queue_isEmpty(struct _queue * q);
    void queue_enqueue(struct _queue * q, int element);
    int queue_dequeue(struct _queue * q);
} queue_t;

queue_t * queue_create(void);
typedef struct _queue {
    LL_t * intlist;
    void ( * destroy)(struct _queue * q);
    int ( * isEmpty)(struct _queue * q);
    void ( * enqueue)(struct _queue * q, int element);
    int ( * dequeue)(struct _queue * q);
} queue_t;

queue_t * queue_create(void);
void queue_destroy(queue_t * q);
int queue_isEmpty(queue_t * q);
void queue_enqueue(queue_t * q, int element);
int queue_dequeue(queue_t * q);

typedef struct _queue {
    LL_t * intlist;
    void ( * destroy)(struct _queue * q);
    int ( * isEmpty)(struct _queue * q);
    void ( * enqueue)(struct _queue * q, int element);
    int ( * dequeue)(struct _queue * q);
} queue_t;

queue_t * queue_create(void);

typedef struct _queue {
    LL_t * intlist;
    void ( * destroy) (struct _queue * q);
    int ( * isEmpty) (struct _queue * q);
    void ( * enqueue) (struct _queue * q, int element);
    int ( * dequeue) (struct _queue * q);
} queue_t;

queue_t * queue_create(void);

Pointer to a function rather than the function itself

Caller’s notation:
Q->enqueue(Q, x);
A Look Ahead to C++

Motivated by these interface issues, C++ evolved out of C.

- formulated by Bjarne Stroustrop in 1978

Provides the syntactic sugar for:

- information hiding
- encapsulation of data and methods
- common code re-use situations

Migrate from `struct` → `class`