Data Structures in C

Prof. Georg Feil

Stack & Queue ADTs with Pointers

Summer 2017

Acknowledgement

- These lecture slides are partly based on slides by Professor Simon Hood
- Additional sources are cited separately

Reading Assignment (required)

- Data Structures (recommended textbook)
  - Chapter 5

(Note the textbook does a few things we might consider poor style, for example one-letter variable names and using int for Boolean values.)

Stack implementation using pointers

- Using pointers and malloc/free to implement data structures like linked lists, stacks, and queues makes them more useful and dynamic (compared to the array impl.)
  - The data structure can grow to any desired size
  - Operations like inserting and deleting items in the middle become very efficient (why?)
- The stack behaviour is as before
  - LIFO structure (last in, first out)
  - Two main operations: push a new item, and pop the top item off
  - Two other operations: initStack and isEmpty

Stack implementation using pointers

- Our stack variable will be allocated on the heap using malloc
- Each item on the stack will also be allocated using malloc
- In the array implementation the 'top pointer' was just an array index (integer), now 'top' is an actual C pointer
  - Each item on the stack also has a 'next' pointer like a linked list
- There will be no need for a constant like MAXSTACK (why?)
  - Our stack can't overflow unless we run out of heap memory
- An empty stack will be identified using a null 'top' pointer

Stack implementation using pointers

- Download the source files Stack.c and Stack.h from SLATE (week 9)
- This stack implementation is structed as an ADT (Abstract Data Type), *unlike* the linked list
  - All relevant code is in a separate module
  - An API (list of functions to call in .h) is provided to work with it
  - We can create more than one "instance" of the stack
- Examine the code and answer the following questions

1. What indicates whether the stack is empty?
2. Why is the variable 'tmp' needed in the pop() function?

Exercise 1: Stack with integers

- Write a main function in a separate .c file that creates a stack and does some operations on it as described below
  - Recall: If you're using the Dev-C++ IDE you'll have to create a project to compile more than one .c file together
  - Choose 'Empty Project' and make sure to choose 'C Project'
  - Right-click on the project name to add new or existing files
- Do this in your main function to test the stack:
  - Initialize the stack using initStack()
  - Push the numbers 36, 15, 52, 23
  - Pop all the numbers and print them
    - Do the numbers come out in the expected order?

Queues

Using Pointers

Queue implementation using pointers

- Now let's look at a queue implementation using pointers instead of arrays
- This type of queue is the same as a linked list!

- The properties of the queue are the same as before
  - FIFO structure (first in, first out)
  - Two main operations: enqueue an item at the back , and dequeue the front item
  - Two other operations: initQueue and isEmpty

Queue implementation using pointers

- Our queue variable will be allocated on the heap using malloc
- Each item in the queue (node) will also be allocated using malloc
- In the array implementation the head/tail pointers were array indexes (integers), now they are actual C pointers
  - Each item in the queue also has a 'next' pointer like a linked list
- There will be no need for a constant like MAXQ (why?)

Queue implementation using pointers

- Download the source files Queue.c and Queue.h from SLATE (week 9)
- Create a Dev-C++ project, or compile it using command line
- Examine the code and answer the following questions

1. What indicates whether the queue is empty?
2. This queue implementation does not need logic for 'circular' behavior, why?

Exercise 2: Queue of integers

- Write a new main function in a separate .c file that does the following:
  - Create a queue as a local variable using initQueue()
  - Enqueue the following numbers in the order shown: 36, 15, 52, 23
  - Dequeue all the characters and print them. Use a loop and write your code so that it would work with any number of items in the queue.
- Do the numbers come out in the expected order?

Exercise 3: Queue of characters

- Create a new Dev-C++ project with a separate copy of the queue ADT implementation
- Modify the queue so that it holds characters instead of integers
- Write a main function in a separate .c file that creates a queue and does the following operations on it:
  - Create a queue using initQueue()
  - Enqueue the characters of your name
  - Dequeue all the characters and print them on one line

Exercise 4: Queue as a global/module var

Starting with your program from Ex 2,

- Modify the queue variable declaration so that it's a module variable (static global var, not a local var)
- Create two separate functions, one that contains the code to add items to the queue, and another that contains the code to remove and print the items. For example:

```
void addQ(int[] values, int len) { … }
void removeQ() { … }
```

- In the main function, initialize the queue and call the add and remove functions