Midterm I Exam

15-122 Principles of Imperative Computation
Frank Pfenning, Tom Cortina, William Lovas

September 30, 2010

Name: Andrew ID:

Instructions

- This exam is closed-book with one sheet of notes permitted.
- You have 80 minutes to complete the exam.
- There are 4 problems.
- Read each problem carefully before attempting to solve it.
- Consider writing out programs on scratch paper first.

	Searching & sorting	Stacks & queues	Linked lists	Modular arith. & JVM	
	Prob 1	Prob 2	Prob 3	Prob 4	Total
Score					
Max	40	50	30	30	150
Grader					

2 Stacks and Queues (50 pts)

Consider the following interface to stacks, as introduced in class.

```
typedef struct stack* stack;
stack s_new();                    /* O(1); create new, empty stack */
bool s_empty(stack S);            /* O(1); check if stack is empty */
void push(int x, stack S);        /* O(1); push element onto stack */
int pop(stack S);                 /* O(1); pop element from stack  */
```

In these problem you do not need to write annotations, but you are free to do so if you wish. You may assume that all function arguments of type `stack` are non-`NULL`.

Task 1 (10 pts). Write a function `rev(stack S, stack D)`. We require that D is originally empty. When `rev` returns, D should contain the elements of S in reverse order, and S should be empty.

```
void rev(stack S, stack D)
//@requires s_empty(D);
//@ensures s_empty(S);
{
  while (!stack_empty(S))
      push(pop(S), D));

}
```

As a variant on this question, consider a similar function, except that we want S to be in its orginal state at the end. Consider how we could write this code:

(a) Limited to the interface functions above.
(b) Using the underlying implemenation, as used previously.

Now we design a new representation of queues. A queue will be a pair of two stacks, in and out. We always add elements to in and always remove them from out. When necessary, we can reverse the in queue to obtain out by calling the function you wrote above.

```
struct queue {
  stack in;
  stack out;
};
typedef struct queue* queue;
```

Task 2 (10 pts). Write the enq function.

```
void enq(queue Q, int x) {
    push(Q->in, x);
}
```

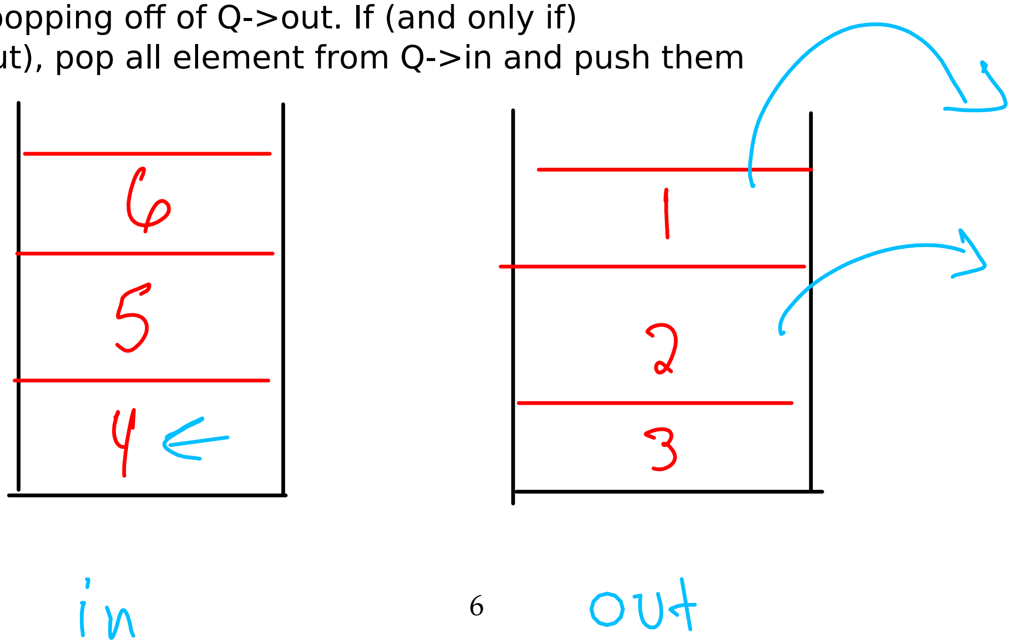
Task 3 (10 pts). Write the deq function. Make sure to abort the computation using an appropriate assert(_,_) statement if deq is called incorrectly.

```
int deq(queue Q) {
```

Idea: Dequeue by popping off of Q->out. If (and only if) stack_empty(Q->out), pop all element from Q->in and push them on Q->out.

Reason over why this maintains the correct order.

```
}
```



Midterm I Exam

15-122 Principles of Imperative Computation
André Platzer Ananda Gunawardena

February 23, 2012

Name: Andrew ID: Section:

Instructions

- This exam is closed-book with one sheet of notes permitted.
- You have 80 minutes to complete the exam.
- There are 4 problems.
- Read each problem carefully before attempting to solve it.
- Do not spend too much time on any one problem.
- Consider if you might want to skip a problem on a first pass and return to it later.
- And most importantly,

DON’T PANIC!

	Mod.arith.	Safari	Contracts	Big O	
	Prob 1	Prob 2	Prob 3	Prob 4	Total
Score					
Max	20	35	25	20	100
Grader					

1 Modular Arithmetic (20 pts)

$-x = \sim x + 1$

In C0, values of type `int` are defined to have 32 bits. In this problem we work with a version of C0 where values of type `int` are defined to have only 7 bits. In other respects it is the same as C0. All integer operations are still in two's complement arithmetic, but now modulo 2^7. All bitwise operations are still bitwise, except on only 7 bit words instead of 32 bit words.

Task 1 (10 pts). Fill in the missing quantities, in the specified notation.

1000000

-2^6 2^0

a. The minimal negative integer, in decimal: -64

b. The maximal positive integer, in decimal: 63 $2^6 - 1$ 0111111

c. -4, in hexadecimal: 0x 7C $\underbrace{111}_{7}\underbrace{1100}_{C}$

d. 44, in hexadecimal: 0x 2C $32+8+4$ | $010\underbrace{1100}$

e. `0x48`, in decimal: -56

$1001000 = -64+8$

Task 2 (10 pts). Assume `int x` and `int y` have been declared and initialized to unknown values. For each of the following, indicate if the expression always evaluates to `true`, or if it could sometimes be `false`. In the latter case, indicate a counterexample in the C0 dialect described here by giving a value for x and y that falsifies the claim. You may use decimal or hexadecimal notation.

a. `x >= x - 1` F int_min()

b. `~(x ^ (~x)) == -1` always false

c. `x+(y+1)-2*(x-1)-3 == -x+y` T

$x + y + 1 - 2x + 2 - 3 = -x + y$

d. `(x!=-x || y!=-y) || x==y` ________________

e. `x <= (1<<(7-1))-1` ________________

Although it is possible that we will have overflow in (c), we are guaranteed that overflow behaves predicatably according to modular arithmetic.
So, even if overflow is involved, it will affect both expressions and they will still evaluate to the same value.

4 Big-O (20pts)

Task 1 (10 pts). Define the big-O notation

$f(n) \in O(h(n))$ if and only if $n_0 \geq 0$, $c > 0$ s.t. $f(n) \leq c\,h(n)$

and briefly state the two key ideas behind this definition in two sentences and briefly explain why and how it captures those key ideas:

for all $n \geq n_0$

Task 2 (10 pts). For each of the following, indicate if the statement is true or false.

(a) $O(n^2 + 1024n \log n + 10^{10}) = O(5n^2 + 1)$ T

(b) $O(\underline{n * \log(n)}) \subset O(n)$ F

(c) $O(n) \subset O(n * \log(n))$ T

(d) $O(3 * \underline{\log_2} n) = O(2 * \underline{\log_3} n)$ T

(e) $O((10 \log n + 3 * n) * n^2) \subset O(\underline{\log(n^3)})$ F

$10n^2 \log n + 3n^3$