Midterm #1

Instructions: This is a closed book, closed notes exam. Do all 5 problems. Do your best to communicate your ideas *clearly* and *succinctly*. Draw a picture if it is useful to explain something. Show all work on these pages. Good luck!

NAME:

On problem	you got	out of
1		20
2		20
3		20
4		20
5		20
Σ		100

1 Definitions [20 points]

In a sentence (or two) describe each of the following concepts.

A. Abstract Data Type (ADT).

B. Memoization.

C. Associative Array.

2 Short Answers [20 points]

A. Reference types.

Write a C++ procedure `swap` which will swap its integer arguments. I should be able to say `int a=10; int b=20; swap(a,b);` and, following execution of the last statement, `a` should now have value 20 and `b` should now have value 10.

B. Infix to Postfix.

If you use the procedure described in class and in your text, what postfix expression would you find to be equivalent to the following infix one? (Assume the C++ rules of operator precedence.)

$$1 + 2 * (4 + 3) * 5$$

C. Big-O, Theta, Omega.

Identify each of the following statements as **TRUE** or **FALSE**. You must get all three correct to earn credit.

$$n^2 \lg n = O(n^2).$$

$$10n^3 = \Omega(n^2).$$

$$n^2 + 10n + 3 \log n + 50 = \theta(n^2).$$

3 Strings [20 points]

A. Think back to the Knuth-Morris-Pratt (KMP) algorithm described in class and in your book. Recall that this algorithm locates the first occurrence of a pattern p in a string s. The algorithm uses a "failure function," $f(i)$, whose values depend only on the pattern p.

Fill in the values of f for the given pattern p =acbacbabac: (*Note: you may use either the numbering convention I used in class,* **or** *the one used in the text.*)

i	1	2	3	4	5	6	7	8	9	10
$p[i]$	a	c	b	a	c	b	a	b	a	c
$f(i)$										

B. Use the values of $f(i)$ computed above. Suppose you're running the KMP algorithm on a string s and the pattern p. You've just compared $s[100]$ and $p[8]$ (p is indexed as above). You got a mismatch: $s[100] \neq p[8]$. The KMP algorithm tells you to resume comparisons by comparing

$s[\quad]$ and $p[\quad]$.

(Fill in the subscripts.)

4 Using Stacks [20 points]

Programming language **W** groups its statements using the pairs of keywords:

do ... **od**

begin ... **end**

repeat ... **until**

For example, if we ignore everything except the occurrences of these keywords,

do begin begin end end od repeat until

is a properly grouped program; while

do begin until od

is not.

Write a function `int properlyNested()` which returns 1 if the sequence of *tokens* corresponding to the above keywords are properly nested; and which returns 0 otherwise. Use the token `EOF` to indicate that there are no more tokens to read. As an example, if your function sees the sequence of tokens:

```
DO BEGIN BEGIN END END OD REPEAT UNTIL EOF
```

it should return 1 (everything is properly nested); but if your program sees the sequence of tokens

```
DO BEGIN UNTIL OD EOF
```

then it should return 0.

Assume you have the following available to you:

```
enum Tokens {DO, OD, BEGIN, END, REPEAT, UNTIL, EOF};
Tokens getNextToken() { /* ... */ } ;      //Returns next token

class TokenStack {  // Complete *definition* of  a TokenStack
  /* private part */
public:
  TokenStack() { /* ... */};                // constructor
  ~TokenStack() { /* ... */ };              // destructor
  void push(Tokens t) { /* ... */ };        // push token onto the stack
  Tokens pop() { /* ... */ };               // return token at TOS
  int empty() { /* ... */ };                // return 1 if stack empty, else 0
}
```

Do *not* implement the member functions of TokenStack, nor the getNextToken() function. Only fill in the following function.

```
// Returns 1 if sequence of tokens returned by getNextToken() are properly
// paired (DO - OD; BEGIN - END; REPEAT - UNTIL); returns 0 otherwise.
// Token EOF marks that the last token has been read.
int properlyNested()
{



}  // properlyNested
```

5 Dequeues [20 points]

A *dequeue* (**d**ouble-**e**nded **queue**) is an abstract data type that is like a queue except that it supports insertions and deletions at *both* the front and the rear. Consider a dequeue of `int`s. It supports:

```
void insert_front (int n); // add n at the front of the queue
void insert_rear (int n);  // add n to the rear of the queue
int delete_front();        // delete and return whatever is at front
int delete_rear();         // delete and return whatever is at the rear
```

A. Describe *with a picture* a *representation* of a dequeue. Your representation must support the above operations in $O(1)$ time. You do *not* know a bound on how many `int`s your dequeue might contain.

B. Give the complete *declaration* of a dequeue. Include the private data members. Include a constructor and destructor. If you implment in terms of other classes, include their declaration, too.

C. Give a complete *implementation* of the `int delete_rear()` routine. Note that it is an *error* to call `delete_rear()` on an empty dequeue. Call `error()` if this happens.

D. Explain how your destructor will work. Give its $\Theta(f(n))$ running time, where n is the number of elements in the dequeue.