Midterm

Instructions: This is a closed book, closed notes exam. Do all 6 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.

Be careful not to make careless errors when showing the result of executing an algorithm described in class; in most cases we will not give partial credit.

Relax. Good luck!

Students taking the 10:30 – 11:50 exam: *Your signature certifies that you will not communicate any aspect of this exam with any person taking the later exam.*
Students taking the 12:10 - 1:30 exam: *Your signature certifies that you have not had any communication concerning this exam with anyone who took it already.*

NAME:

SIGNATURE:

On problem	you got	out of
1		30
2		30
3		30
4		20
5		30
6		30
Σ		170

1 Definitions [30 points]

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

A. Memoization.

B. Amortized running time.

C. Operator overloading.

2 Short Answers [30 points]

A. 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.)

$$8 * (5 + 2) - 4 * (4 + 3) * 5$$

B. Big-O.

Give a formal definition of $O(f(n))$.

C. Big-O, Theta, Omega.

Identify each of the following statements as being either **TRUE** or **FALSE**.

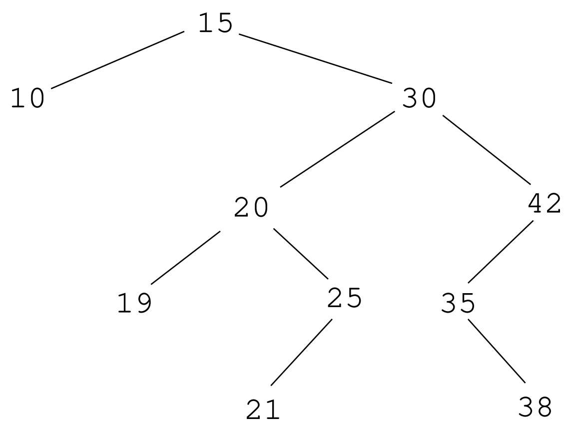
$$n^2 \log n = O(n^2).$$

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

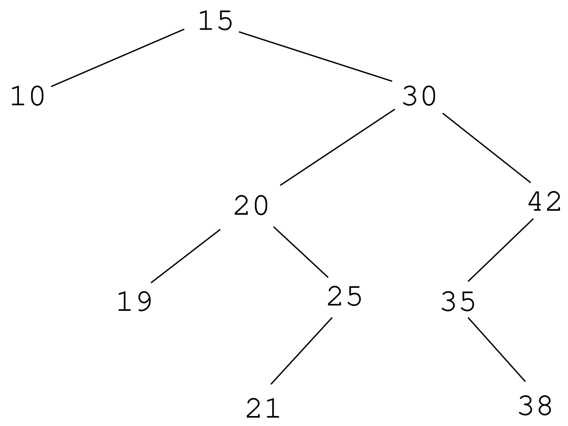
$$n^2 + 1000n + 3 \log n + 80 = \theta(n^2).$$

3 Binary Search Trees — Splay Trees [30 points]

A. Show the **binary search tree** that results if you **Delete** the element with key **30** from the following binary search tree. Assume an "ordinary" binary search tree (ie., not a splay tree, AVL tree, etc.) and use the algorithm described in class and in your text.



B. Show the **splay tree** that results (after all splaying is complete) when you do a **Find** on the element with key **38** from the following splay tree. *Show all intermediate work.*



4 String Matching [20 points]

A. In class we presented an algorithm to find the first occurrence of a pattern P within a text string T using a Finite Automaton. Consider the pattern:

$$P = \text{abbabaa}$$

Draw a picture representing the Finite Automaton associated to P. Assume the underlying alphabet is $\Sigma = \{\text{a,b}\}$. Make sure to indicate the start state, final state(s), and all transitions.

5 Using Stacks [30 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 definitions corresponding to the following declarations available to you. The operations have the obvious semantics.

```
enum Token {DO, OD, BEGIN, END, REPEAT, UNTIL, EOF}; //Tokens
Token getNextToken();

class TokenStack {
private:          // ...
public:
  TokenStack();
  ~TokenStack();
  void push(Token);
  Token pop();
  int empty();
}
```

Don't implement `getNextToken()` or the member functions of `TokenStack`. Only fill in the definition for the following function.

```
int properlyNested()
{

} // properlyNested
```

6 Thesaurus ADT [30 points]

A *slot* in a thesaurus consists of an *entry*, a (possibly empty) list of *synonyms* for that entry, and a (possibly empty) list of *antonyms* for that entry. The entry and each of its synonyms and antonyms are strings. For example, a 2-slot thesaurus might contain the information:

entry: cold. **syn:** chilly, freezing, gelid, icy. **ant:** hot, tepid, warm.

entry: beautiful. **syn:** exquisite, gorgeous, lovely, pretty. **ant:** grotesque, homely, ugly.

As an ADT, a thesaurus supports the following (informally described) operations:

- `Make_current_entry`. Takes a string `ce` and makes `ce` the "current entry." (If there is not yet a slot with entry `ce` then such a slot is created.) The "next synonym" and "next antonym" for `ce` is reset (made to be the first synonym and first antonym for `ce`).
- `Add_syn`. Takes a string `s` and adds `s` as a synonym for the current entry, `ce`.
- `Add_ant`. Takes a string `s` and adds `s` as an antonym for the current entry, `ce`.
- `Get_syn`. Returns the next synonym for the current entry, `ce`. If there are no more synonyms for `ce`, return the empty string.
- `Get_ant`. Returns the next antonym for the current entry, `ce`. If there are no more antonyms for the `ce`, return the empty string.

As an example, the thesaurus described above might be created by:

```
Thesaurus T;
T.Make_current_entry("cold");
T.Add_syn("chilly"); T.Add_syn("freezing"); T.Add_syn("gelid"); T.Add_syn("icy");
T.Add_ant("hot"); T.Add_ant("tepid"); T.Add_ant("warm");
T.Make_current_entry("beautiful");
T.Add_syn("exquisite");T.Add_syn("gorgeous");T.Add_syn("lovely");T.Add_syn("pretty");
T.Add_ant("grotesque"); T.Add_ant("homely"); T.Add_ant("ugly");
```

Having created the thesaurus above we could, for example, print the synonyms for "cold" by

```
T.Make_current_entry("cold");
do {
  s = T.Get_syn();
  if (s != "") cout << s << eoln;
} while (s != "");
```

A. Formalize what has just been said: carefully, rigorously, unambiguously define a thesaurus as an ADT. Give a comment on each operation which unambiguously specifies the effect of that operation. Resolve any ambiguities in the description we have given in whatever manner way you see fit. Do not write code; an abstract description is what is wanted.

ADT **Thesaurus**

// DATA:

// OPERATIONS:

B. For each scenario described below describe the data structure(s) you would choose to implement your Thesaurus, and why.

In **scenario 1** your goal is simplicity. You expect that there will only be a few slots in your dictionary, and each slot will contain a few synonyms and antonyms.

In **scenario 2** your goal is speed. You expect there will be approximately 200,000 slots, and each slot will contain a few synonyms and a few antonyms. You have no idea the order in which entries will be introduced into your Thesaurus (perhaps it will be alphabetical order).