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		15
3		15
4		25
5		25
Σ		100

1 Describe [20 points]

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

A. **Abstract Data Type.**

B. What it means when function `fnct` is made a **friend** to a class `T`.

C. **L-value** (as opposed to an R-value).

D. The **Knuth-Morris-Pratt Algorithm** (what it does—not how it does it!).

2 Short Answer [15 points]

Alice has written a C++ program with a class `Matrix`. Somewhere in her program she has a declaration for a binary operator `operator+`. It looks like:

```
Matrix operator+ (const Matrix&);
```

Alice also has in her program a declaration that looks like

```
Matrix operator+ (double, const Matrix&);
```

A. What is the **signature** of a function? Do the two functions above have the same or different signatures?

B. If both of the functions above are **binary** operators, why does the first declaration list **one** argument and the second declaration list **two** arguments?

C. Why do you think Alice is writing `const Matrix&` instead of `Matrix`?

3 Stacks [15 points]

Use the procedure described in class and in your text to convert the following **infix** expression into an equivalent **postfix** expression. Assume the customary order of operator precedence: $*$ and / of equal precedence, grouping left-to-right; then + and − of equal and lower precedence, also grouping left-to-right; parenthesis used to modify all this. The $ is a special symbol used to indicate the end-of-input.

Draw the state of the stack at each of the indicated points, and show the postfix expression which results from your conversion.

$$a \quad + \quad b \quad * \quad c \quad / \quad (\quad d \quad + \quad e \quad / \quad f \quad) \quad \$$$

Equivalent postfix expression:

4 Analysis and Design [25 points]

Consider the function

```
int R (int n)
{
  if (n<=0) return n-1;

  int s = 1;
  for (int i=0; i<n; i++) {
    s = (s*2*  R(n-1)  +3)%47;
  }
  return s;
}
```

A. Write a **recurrence relation**[1] for $T(n)$, where $\Theta(T(n))$ is the time complexity of the above program:

$$T(n) \quad =$$

B. Now solve your recurrence relation to get an **explicit formula**:

$$T(n) \quad =$$

C. It took my SGI Indy **6.57 seconds** to compute `R(10)` using the code above. Estimate how long it will take my machine to compute `R(12)`. You need not "simplify" your answer; writing a correct expression is fine.

[1] Recall that a recurrence relation for $T(n)$ expresses $T(n)$ in terms of $T(n')$-values, for $n' < n$. When we evaluated the "Bhutan change making problem" we got a recurrence relation of $T(n) = T(n-1) + T(n-3) + T(n-8) + T(n-17)$, when $n > 0$; $T(n) = 1$ otherwise.

D. Now you are to make a change so the program runs **much** faster (still computing the correct value regardless of what it is called on.) I've repeated the old program; just pencil in your changes.

```
#include <iostream.h>

int R (int n)
{

  if (n<=0) return n-1;

  int s = 1;

  for (int i=0; i<n; i++) {

    s = (s*2*  R(n-1)  +3)%47;

  }

  return s;

}



int main()
{

  cout <<  R(12) << endl;

}
```

E. Let $T(n)$ be the running time of your modified program. Fill in with an explicit formula:

$$T(n) \;\; = \Theta(\qquad\qquad)$$

Explain why your running time is as claimed.

5 Dictionaries [25 points]

An entry in a dictionary consists of a word together with its definition. A dictionary entry can be represented by:

```
struct Entry {
  String word;        // A word in the dictionary (a nonempty string)
  String definition;  // Its definition (also a nonempty string).
};
```

(The above assumes a class `String`. You may assume `String` has whatever operations you need that are natural and convenient.)

A **Dictionary** is an ADT supporting the operations `Insert` and `Lookup`, described *informally* as follows:

- `Insert` – takes an entry `(wd, defn)`, where `wd` is not yet in the dictionary, and adds this entry into the dictionary.
- `Lookup` – takes a word `wd` and, if it already has a definition, return that definition.

A. Formalize what I have just said by *carefully* and *rigorously* defining a dictionary as an ADT. Give a comment on each operation which unambiguously specifies the effect of that operation.

ADT **Dictionary**:

// DATA:

// OPERATIONS:

B. Write the declaration for the **public** part of a class **Dictionary**. Include a constructor and destructor:

```
class Dictionary {
 private: // ...
 public:



};
```

C. Describe two substantially *different* ways to represent a dictionary. Give the **private** part of a class declaration for each implmentation. Do *not* assume any known bound on the maximal number of elements in your dictionary. Discuss, in a couple sentences, advantages and disadvantages of each representation.

```
class Dictionary {
  private:            // # 1


  public:  // ...
};
```

```
class Dictionary {
  private:            // # 2


  public:  // ...
};
```

D. Choose one of the above representations and indicate which. Now write the definition of `Insert` and `Lookup`.