University of Toronto
Faculty of Applied Science and Engineering
FINAL EXAMINATION, December 1998
First Year - Program 5
CSC181F Introduction to Computer Programming
Examiner - D.B. Wortman

One official 8 1/2 x 11 inch Aid Sheet permitted. Non programable calculators permitted. Write all answers in the examination book.
ANSWER ALL QUESTIONS. 7 questions, 150 marks total. 2 ½ hours (150 minutes).
WRITE LEGIBLY; unreadable answers can not be marked.
Clearly state any assumptions you've made in answering the questions in the exam book. You may use C++ instead of C in your answers if you wish.

DON'T PANIC *KEEP COOL* *DON'T PANIC*

1. [30 marks] A (slightly simplified) version of floating point constants in C can be defined as follows:

floating-constant	is	digit-sequence exponent
	or	dotted-digits exponent_{opt}
exponent	is	e sign-part_{opt} digit-sequence
	or	E sign-part_{opt} digit-sequence
sign-part	is	+
	or	–
dotted-digits	is	digit-sequence **.**
	or	digit-sequence **.** digit-sequence
	or	**.** digit-sequence
digit-sequence	is	digit
	or	digit digit-sequence
digit	is	any one of 0 1 2 3 4 5 6 7 8 9

Where the subscript *opt* indicates that a component is optional (i.e. it may be omitted).

Write a C function with the header

```
int isValidFloat( char * inFloat ) ;
```

This function returns true (non-zero) if its argument *inFloat* contains exactly one floating point constant satisfying the definition given above. Otherwise it returns false (zero).

Examples
isValidFloat("3.141659") -> 1
isValidFloat("123456E+7") -> 1
isValidFloat("12.34 Goodbye Cruel World") -> 0
isValidFloat("5678") -> 0

2. [15 marks] A CSC180F student was given the programming assignment:

> *Run Length Encoding (RLE)* is a technique for compressing information by replacing repetitive information by a *repetition count* and one copy of the information. Write a function with the header
>
> **char *** rlencode(**char *** inString) ;
>
> This function run length encodes repeated characters in inString by replacing the repeated characters with a biased count (the actual repetiton count plus 128) followed by a single instance of the repeated character. Example:
>
> AAAAAbbbbbbCCCCCdeeeeeeeeeeeef **->** ^{133}A^{134}b^{133}Cd^{140}ef
>
> Where the raised numbers represent the 1-character encoded counts.

The student wrote the C function shown below to solve this assignment.

```
1       char * rlencode( char * S ) {
2           char och  ;
3           char ch, *s1 = S, s2 = S ;
4           int cnt ;
5           while( ch = *S++ ) {
6               cnt = 0 ;
7               while( ch == och ) {
8               cnt++ ;
9               ch = *S++ ;
10              } ;
11              if( cnt ) {
12              *s1++ = 128 + cnt ;
13              *s1++ = och ;
14              } else
15              *s1++ = ch ;
16              och = ch ;
17          }
18          *s1++ = 0 ;
19          return  s2 ;
20      }
```

The numbers on the left are for reference and are not a part of the function.

For this question you are to **inspect** the function shown above.

[5 marks] Describe any improvements you would make in the **programming style** used in this function.

[10 marks] Describe any **problems** or **errors** that you found in this function.

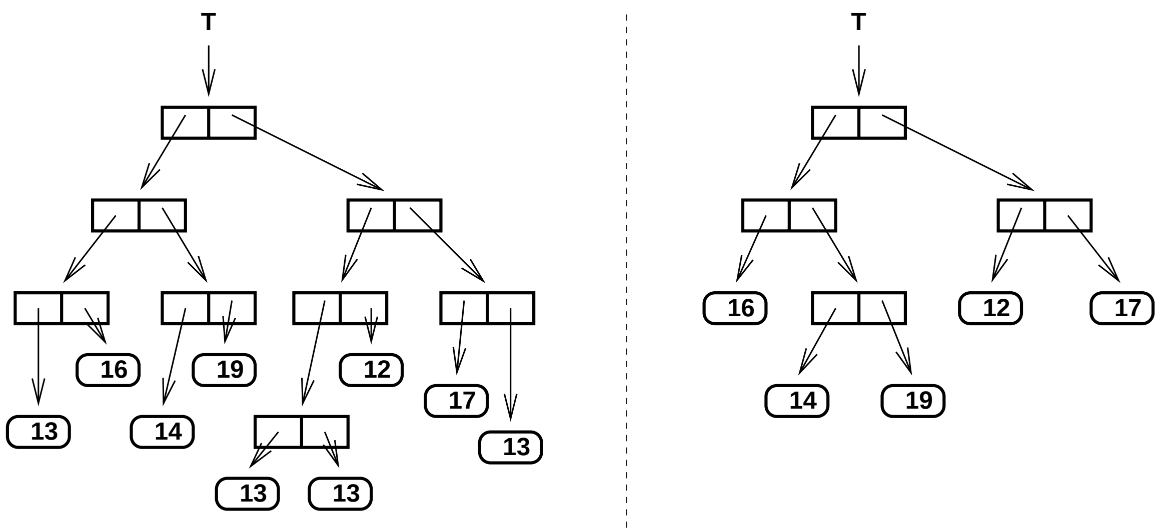
3. [30 marks] Assume that the data structures show below are used to implement binary trees of integers.

```
typedef struct treeNode * treePtr ;
typedef enum {  leaf, branch } nodeKinds ;
struct treeNode {
        nodeKinds nodeType ;        /* type of node */
        int value ;                 /* value of leaf */
        treePtr left, right ;       /* links for branches */
} ;
```

Write a C function with the header

treePtr pruneTree(treePtr *inTree* , **int** *val*) ;

This function removes all leaves that have the value *val* from the tree *inTree*. It returns a pointer to the modified tree. Any branches that become unnecessary due to leaf removal should also be removed. Example of pruneTree(T , 13) .



4. [20 marks] Write a C function with the header

double nthroot(**double** X , **int** N) ;

For N >= 3 and X >= 0.00 this function computes the Nth root of X using Newton-Raphson iteration. You may assume that four iterations are sufficient to produce the desired accuracy. You do **not** have to do any scaling of the argument X.

5. [25 marks] Assume that the data structures shown below were used to implement the Poly class in Assignment 6. Assume that the nodes for a polynomial are kept in order from largest exponent to smallest and that a term with exponent of zero is the only one that can have a coefficient of zero.

```
struct polyNode {
        double coeff ;   // coefficient
        int expon ;      // exponent
        polyNode * next;     // link to next term
    } ;
```

Write another member function for the Poly class with the prototype:

```
Poly::derivative( ) ;
```

This function computes the first derivative of the invoking polynomial and returns this derivative as its result.

6. [10 marks] Design some difficult test cases to test the polynomial derivative function that you wrote in Question 1.

   [5 marks] Specify 5 test polynomials. For each test case, explain in one sentence why you've choosen this test case.

   [5 marks] Specify one test polynomial that will CRASH all but the most carefully written polynomial derivative functions.

7. [20 marks] Assume the same data structures as in Question 5, write a C utility function with the header:

```
int isGoodPoly( polyNode * poly ) ;
```

This function checks the correctness of its argument polynomial *poly* and returns non-zero (true) if and only if the following conditions hold.

a) The terms are ordered by exponent value, largest to smallest.

b) There is no term with a non-zero exponent and a coefficient of 0.00.

c) No two terms have the same exponent value.

If any of these conditions fails to hold, the function returns zero (false).