University of Toronto
Faculty of Applied Science and Engineering

FINAL EXAMINATION, December 1999
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 programmable calculators permitted.
Write all answers in the examination book.
ANSWER ALL QUESTIONS. 8 questions, 150 marks total. $2\frac{1}{2}$ 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

Useful Data Structures

```
/* singly linked lists */
typedef struct listNode * ListPtr ;
struct listNode {
     int value ;
     /* link to next node */
     listPtr next ;
} ;
typedef struct listNode LISTNODE ;
#define LNODESIZE ( sizeof( LISTNODE ) )
```

```
/* binary trees */
typedef struct treeNode * TreePtr ;
struct treeNode {
     int value ;
     /* left and right branches */
     treePtr left, right ;
} ;
typedef struct treeNode TREENODE ;
#define TNODESIZE ( sizeof( TREENODE ) )
```

1a. **[1 mark]** Write your name LEGIBLY on the front cover of the exam book.
1b. **[1 mark]** Write your student number LEGIBLY on the front cover of the exam book.

2. **[20 marks]** Write a C function with the header

```
char *  findLongestWord( const char * wordlist ) ;
```

Where *wordlist* is a string containing *words* separated by white space. A *word* is a sequence of upper and/or lower case letters. *White space* is one or more blanks. This function finds the longest word in *wordlist* and returns it as a null-terminated string. If there is more than one word in wordlist that has the longest length, the function returns the first one. Examples:

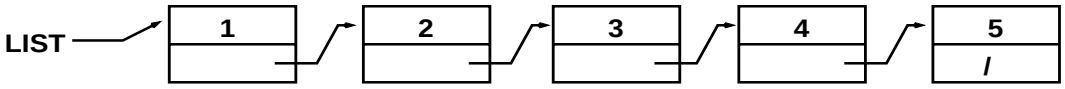
findLongestWord("engineers rule the world")	→	"engineers"
findLongestWord("Physics Chemistry Math English Computers")	→	"Chemistry"
findLongestWord("AbRaCadabra")	→	"AbRaCadabra"
findLongestWord("")	→	""

3. [20 marks] Assume the definition for singly linked lists given at the start of this exam. Given the functions

```
ListPtr apply ( ListPtr P , int F ( int val ) ) {
    if ( P == NULL )
        return NULL ;
    else {
        ListPtr newNode ;
        newNode = (ListPtr) malloc( LNODESIZE ) ;
        assert ( newNode ) ;
        newNode -> value = (*F)( P -> value ) ;
        newNode -> next = apply( P -> next , F ) ;
        return newNode ;
    }
}
```

```
int G ( int arg ) {
    static int old = 0 ;
    old += arg ;
    return old ;
}
```

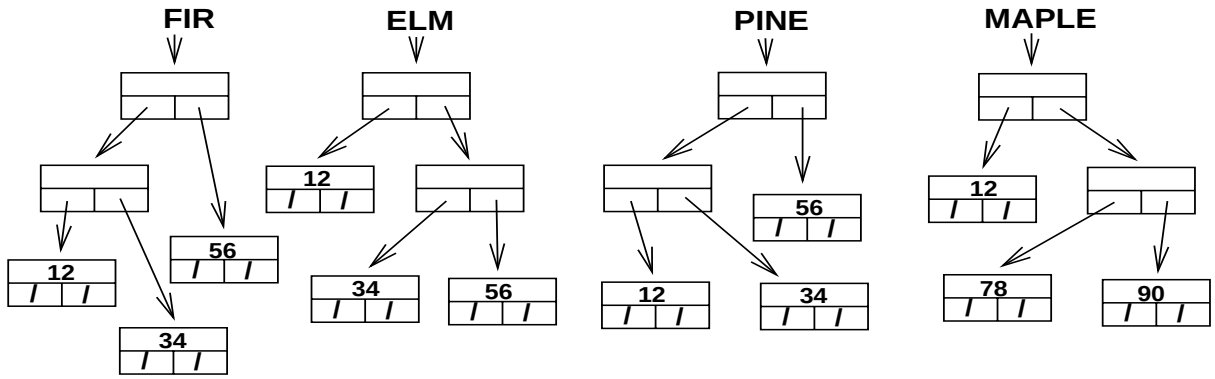
And the list



What is the result of the call `apply( LIST , G ) ;`

4. [30 marks] Examples of 4 small binary trees are given below. We say that two trees are *equal* if and only if:

a) they have exactly the same branch structure.
b) the values of corresponding leaves are equal.



In the example trees above, the trees fir and pine are equal, trees elm and maple are not equal because they have different leaves, trees fir and elm are not equal because they have different structures. For this question you are to write a C function:

```
int isEqual( treePtr treeA , treePtr treeB  ) ;
```

When started at the root of two trees, this function returns a non-zero (true) value if the two trees are equal and zero (false) otherwise.

5. **[30 marks]** An *N-ary* tree is a data structure similar to the binary tree defined above *except* that there can be an **arbitrary** number of branches at each node. For simplicity assume that an N-ary tree consists of nodes that contain only branches and leaves that contain data (the type of data really doesn't matter for this question). The branches at each node are indexed by an integer value starting at zero.
For this question you are to design some parts of a package for creating and manipulation N-ary trees.

a) Design one or more data structures (C struct declarations) for representing leaf and branch nodes in an N-ary tree. Describe clearly how N-ary nodes are implemented.

b) Assume that *nTree* is a pointer to the data structure you described in part a. Write a C function with the header

nTree branchAccess(nTree treeNode , int branchSelect) ;

The function branchAccess returns the branch of *treeNode* selected by *branchSelect*. branchAccess should return NULL if the selected branch doesn't exist at treeNode.

c) Assume that *nTree* is a pointer to the data structure you described in part a. Write a C function with the header

void addBranch(nTree treeNode , nTree newBranch , int branchSelect) ;

This function adds the branch pointed to by *newBranch* to the tree node pointed to by *treeNode*. The index of *newBranch* in treeNode is specified by the value of *branchSelect*. It there is already a branch with that index at treeNode it is replaced.

6. **[15 marks]** Assume that the *magnitude* of a sparse vector is defined as the square root of the sum of the squares of the elements in the vector. That is

$$\text{magnitude(V)} = \sqrt{\sum_{i=0}^{i=N-1} V_i^2}$$

Write a new member function for the sparseVector class with the header

```
double sparseVector::magnitude() ;
```

This function computes the magnitude (as defined above) of the invoking sparseVector and returns the magnitude as its result. The magnitude of an empty sparse vector is 0.0. Assume any data structure you wish for representing sparse vectors, but describe this data structure clearly as a part of your answer.

7. **[25 marks]** Statistics about the relative occurrences of letters are often useful in the design of text manipulation programs (e.g. compression, encryption). For this question you are to write a program that gathers information about the number of occurrences of letters and pairs of letters in ordinary test. Assume that your input (from stdin) consists only of *words* (sequences of lower case letters 'a' .. 'z') and white space (blanks and newlines). Your program should read its input until end of file and tabulate the number of times each letter and each pair of letters occurs in the input. The output from your program should be a table showing the number of times each letter and each pair of letters occurred. Pair counting starts over at the beginning of each word and includes all pairs within the word, e.g. (1st & 2nd letter), (2nd & 3rd letter), (3rd & 4th letter), etc. Hint: $26^1 = 26, 26^2 = 676$.
Example: if the input file contained one line "minimum pumpkin" then the output from your program might look like:

Letter	Count	Pair	Count
i	3	im	1
k	1	in	2
m	4	ki	1
n	2	mi	1
p	1	mk	1
u	2	mu	1
		ni	1
		pu	1
		um	2

8. **[8 marks]** Describe **briefly** how you would design a program like the one in the previous question if you wanted to record the number of occurrences of all sequences of 5 consecutive letters in words. Hint $26^5 = 11,881,376$.