FINAL EXAMINATION
INTRODUCTION TO ALGORITHMS AND PROGRAMMING II
03-60-141-01

UNIVERSITY OF WINDSOR
SCHOOL OF COMPUTER SCIENCE
Winter 2014

Last Name: ....

First Name: ....

Student ID: ....

PLEASE READ CAREFULLY BEFORE YOU START

1. This is a CLOSED book test; no notes, textbooks, calculators or computer aids are allowed.
2. PRINT your name legibly and clearly with your Student ID in the space indicated above.
3. You will be asked to sign your name, once during the exam (sign-in) and once before leaving the exam room (sign-out).
4. Answer all the questions in the space provided. DO NOT REMOVE any pages or attach any papers to this test. If you need more space please request an additional exam booklet which MUST be returned with this exam paper with your name and ID clearly written on it.
5. You are not allowed to give or receive unauthorized help with your test. Any misconduct, as outlined by the Senate bylaw 31 article I, will be reported accordingly. Such misconduct includes any attempt, by any means whatsoever, to copy answers from other students, or to receive any unauthorized assistance answering any and all questions.
6. You have 3 hours to complete this test.
7. Final examination papers are not returned to students.
8. You may view your exam paper during the specified date and time to be posted by your instructor.
9. **It is recommended that you document your code to clarify ambiguities or partial answers.**

Good Luck!

MARKS:

Q1	/20
Q2	/15
Q3	/15
Q4	/15
Q5	/ 5
Q6	/30
TOTAL:	/100

Q1. [20 MARKS] For each part below, write a single C language statement that answers the question.

A. Write a statement that opens the file "**master.dat**" for reading and assigns the returned file pointer to **mfPtr**. Assume that the file contains only character data with variable length records. **[1 mark]**

```
mfPtr = fopen( "master.dat", "r" ) ;
```

B. Write a statement that reads a record from a file to which `cfPtr` points. Assume that the record consists only of an integer **Number** and string **Name**, both stated in character form and the inputs are to be stored in variables **Number (int)** and **Name (char *)**. **[1 mark]**

```
fscanf( cfPtr, "%d%s", &Number, Name ) ;
```

C. Convert the alphabetic character stored in the variable **C** (**char**) to an uppercase letter, storing the result in **C**. **[1 mark]**

```
C = toupper( C ) ;
```

D. Read an entire line of text into array of characters **S**, from the keyboard. You are not permitted to use either `scanf` or `fscanf`. **[1 mark]**

```
fgets( S, 200, stdin ) ; // NOTE: any reasonable value >= 100 is OK

gets( S ) ;   // This also is fine
```

E. Assuming that **Ptr** is a pointer to a character, assign to **Ptr** the location of the first occurrence of a character **C** in string **S**. **[2 marks]**

```
Ptr = strchr( S, C ) ;

for( Ptr=S ; C != *Ptr && *Ptr != '\0' ; Ptr++ ) ;
    // This is also OK for 1 mark only
    // Note that if C is not found, Ptr ends up set to '\0'
```

F. Assuming that both **S1** and **S2** are defined as string variables of appropriate length, append (or concatenate) no more than the first 10 characters from the string in **S2** to the string in **S1**. **[1 mark]**

```
strncat( S1, S2, 10 ) ;

// Note:  A for-loop approach might work, but is likely to be
complicated.  Watch for such answers and give part marks if they are
on the right track.
```

G. Assuming that both **S1** and **S2** are defined as string variables, compare the string in **S1** with the string in **S2**, within a **`printf`** function, so that the result of the string comparison is outputted. **[1 mark]**

```
printf( "%d", strcmp( S1, S2 ) ) ;
```

H. Assume that a float variable **Number** has been declared. Declare and initialize a pointer **nPtr** to **Number**. **[1 mark]**

```
float * nPtr = &Number ;
```

I. Define **`Part_t`** to be a synonym (i.e. user defined type) for the type **`struct Part`**. **[1 mark]**

```
typedef struct Part Part_t ;
```

J. Write a statement that opens the file "**DAmaster.dat**" for both reading and writing and assigns the returned file pointer to **`mfPtr`**. Assume that the file contains machine readable (i.e. binary) data with fixed length records. **[1 mark]**

```
mfPtr = fopen( "DAmaster.dat", "rb+" ) ;
```

K. Assuming that the direct access file "**DAmaster.dat**" has been opened for reading and writing and file pointer value has been assigned to **`mfPtr`**. Write a statement that outputs the structure **`Part`** to "**DAmaster.dat**". **[2 marks]**

```
fwrite( &Part, sizeof( Part ), 1, mfPtr ) ;
```

L. Assume the declarations: **`double A[10][20], *Aptr = &A[0][0];`**
Write a single **`for`** loop statement that stores the value **9.9** into the first column location in each row within the array **`A`** (i.e. **`A[0][k]`**), using only the pointer **`Aptr`** provided as stated. **[2 marks]**

```
for( ; (Aptr-&A[0][0])/(20*sizeof double);
       Aptr += 20*sizeof double ) *Aptr = 9.9 ;

// Watch for alternative, correct solutions.
```

M. Assume the declarations: **`char A, B;`** and further assume that the actual 8-bit values stored in **`A=01011010, B= 11111111`**. Write a single statement that operates on both **`A`** and **`B`** to produce the result **`10100101`**, which is stored in **`A`**. **[2 marks]**

```
A = A ^ B ;
```

N. Assume that **`Aptr`** has been declared and initialized as a pointer to a dynamically allocated structure. Write a single statement that deallocates the structure pointed to by **`Aptr`**. **[1 mark]**

```
free( Aptr ) ;
```

O. Assuming the definitions: **`int C; char S[256];`** write a single statement that counts the number of records inputted from **`stdin`**, storing the count result in **`C`** and assuming that no input record contains more than 255 characters. **[2 marks]**

```
for( C=0 ; feof( stdin ) != EOF ; C++ ) ;

// Watch for correct alternative solutions using gets() or fgets()
```

Q2. [15 MARKS]

Part A. [8 marks] Consider the definition of the string processing function **`strspn()`**, whose prototype is defined as:

> **`size_t strspn( const char * S1, const char * S2 );`**
> Definition: String S1 is searched, and returns the length of the initial substring segment in S1 that contains only characters found in S2.

As an example: **`printf("%d", strspn("The-cat-and-the-rat", "-echT") );`** produces output **5**.
Write a complete C function definition of **`strspn()`** that follows the definition above. **You <u>may not</u> use any C library functions!**

```
size_t strspn( const char * S1, const char * S2 ) {
   int N = 0 ;                                                // Decl. 2 mks
   char * p1 = S1, * p2 ;

   while( *p1 != '\0' ) {
      for( p2=S2 ; *p2 != '\0' ; p2++ )
         if( *p1 == *p2 ) {
            N++ ;
            break ;   // Break from for-loop
         }
      if( *p1 != *p2 ) break ;  // Break from while-loop
      p1++ ;
   }                                                          // Log. 5 mks

   return N ;                                                 // Ret. 1 mks
}

// I expect alternative solution strategies.  Be sure to trace the logic
to ensure correctness, or not.

Do not over-penalize minor syntactic errors.  If it is clear what the
student is trying to do, but their syntax is consistently wrong, you may
consider partial marks.
```

Question 2 (Continued)

Part B. [7 marks] Consider the definition of the string processing function **`strpbrk()`**, whose prototype is defined as:

> **`char * strpbrk( const char * S1, const char * S2 );`**
>
> Definition: Locates the first occurrence in S1 of any character found in S2, and returns a pointer to that position in S1. Otherwise a NULL value is returned.

As an example: **`printf("%s", strpbrk(("The quick crazy cat", "aktz") );`** produces the output string **`"k crazy cat"`**.

Write a complete C function definition of **`strpbrk()`** that follows the definition above. **You <u>may not</u> use any C library functions!**

```
char * strpbrk( const char * S1, const char * S2 ) {
   char * p1 = S1, * p2 ;                                     // Decl. 2 mks

   while( *p1 != '\0' ) {                                     // Log. 4 mks
      for( p2=S2 ; *p2 != '\0' ; p2++ )
         if( *p1 == *p2 ) {
            return p1 ;
         }
      p1++ ;
   }

   return NULL ;                                              // Ret. 1 mks
}

// I expect alternative solution strategies.  Be sure to trace the logic
to ensure correctness, or not.

Do not over-penalize minor syntactic errors.  If it is clear what the
student is trying to do, but their syntax is consistently wrong, you may
consider partial marks.
```

Q3. [15 MARKS]
Write and document a C function named **subStrCnt** that takes two parameters of type character pointer, **str1** and **str2**. The function **subStrCnt** counts the number of occurrences of the (sub)string **str2** in the string **str1** and returns the count, assuming that each parameter is a valid, initialized string.
[You <u>may</u> use any library function(s) as needed].

<u>Example</u>:
Given: `str1 = "The cat and the rat";`
  and `str2 = "at";`

```
printf("%d", subStrCnt(str1, str2)); // will print 2.
printf("%d", subStrCnt("aardvark", "a")); // will print 3.
```

```
int subStrCnt( const char * str1, const char * str2 ) { // Prot. 3 mks
   char * p1 = str1 ;                                  // Decl. 2 mks
   int Cnt = 0 ;

   while( *p1 != '\0' && p1 != NULL ) {                // Log. 8 mks
      if( p1=strstr( p1, str2 ) != NULL ) {
         Cnt++ ;      // If str2 is found, count it
         p1++ ;       //    and continue searching
      }
      else                                             // Ret. 1 mks
         return Cnt ; // If str2 not found, return
   }

   return Cnt ;                                        // Ret. 1 mks
}
```

```
// I expect alternative solution strategies.  Be sure to trace the logic
to ensure correctness, or not.  Note that my approach uses two return
statements to account for an empty str1.

My solution uses strstr() and this makes it quite straightforward.  If
students develop their own functions, or code substring search into their
solution, expect the code to be considerably more complicated, requiring
careful tracing of the logic.

Do not over-penalize minor syntactic errors.  If it is clear what the
student is trying to do, but their syntax is consistently wrong, you may
consider partial marks.
```

Q4. [15 MARKS]

Write a complete C program that satisfies the following requirements:

a. Defines a 2-dimensional array of integers, **A**, with number of rows and columns defined by MAXROW and MAXCOL respectively. These can be defined using any positive numbers greater than 50.
b. Populates **A** with random integer values.
c. Sorts all the values in **A** so that after the sorting is finished, the elements of **A** are arranged with the smallest integer in **A[0][0]** and the values increase across each row and continue to increase through all the rows.
d. Outputs the array values from smallest to largest in each row, and for all rows in the array.
e. When finished, exit the program.

As an example, consider the case below showing **A** before and after the sorting for a 3x3 array.

```
               Before                      After
          5    3    7                 1    2    3
A =       1    9    4                 4    5    6
          8    2    6                 7    8    9
```

```
//   Solution on next page.
```

```
#include <stdio.h>                                 // #includes and #defines
#include <stdlib.h>                                //       3 marks

#define MAXROW 75
#define MAXCOL 100

int main( ) {
   int A[ MAXROW ][ MAXCOL ] ;                          // Decl. 2 mks
   int j, k, L, Temp ;
   int * Ptr ;

   for( j=0; j<MAXROW ; j++ )                           // Enter random values
      for( k=0; k<MAXCOL ; k++ )                        //     2 mks
         A[j][k] = rand() ;

   L = MAXROW * MAXCOL ;                    // Sorting logic below - 5 mks .

   for( j=0; j<L-2 ; j++ ) {                       // Sort values - I use a pntr
      Ptr = &A[ MAXROW-1 ][ MAXCOL-1 ] ;           //   and view the 2D array as
      for( k=L-1; k>j ; k-- ) {                    //   a 1D list based on the
         if( *Ptr < *(Ptr-1) ) {                   //   row-major ordering in
            Temp = *Ptr ;                          //   memory.
            *Ptr = *(Ptr-1) ;
            *(Ptr-1) = Temp ;
         }
         Ptr-- ;
      }
   }

   for( j=0; j<MAXROW ; j++ ) {                    // Output the sorted array
      for( k=0; k<MAXCOL ; k++ )                   //   with 1 row per line
         printf( "%d ", A[j][k] ) ;                //      2 mks
      printf( "\n" ) ;
   }

   return 0 ;                                                // Ret. 1 mks
}

// I expect alternative solution strategies.  Be sure to trace the logic
to ensure correctness, or not.

Although I have used a pointer to simplify the sorting logic, students may
try other approaches using array indices and the modulus operation.  Check
these very carefully.

Do not over-penalize minor syntactic errors.  If it is clear what the
student is trying to do, but their syntax is consistently wrong, you may
consider partial marks.
```

Q5. [5 Marks] Consider the following situations:

A bank maintains a **Random Access** (also known as Direct Access or Binary) file called **'RAfile.dat'** for all the account holders in their system. Records for each account holder are kept in ascending order of the account numbers. You are writing parts of a C program to access those records for updating purposes.

For that, you have defined a C-structure called **'Account'** (short name for struct account), to temporarily store the records of an account holder inputted from the file. Right now you are not concerned about the number and types of the members of the structure, except for the first member, which is called **'AccountNum'** and has **'int'** type. The file is organized so that **'AccountNum'** is used as the record number.

In the **main()** function, assume that you have declared the following variables:

1. **'pFile'** – a FILE pointer currently pointing to the bank's account holders' file. (File name is not important). **'pFile'** has been opened with both input and output streams.

2. **'anAccount'** – a variable of type **'Account'**. (Currently having no significant values for the members).

3. **'numAcc'** – a variable of type **'int'**. This variable currently contains the account number you want to access.

Assume that the position of the read/write head is unknown.

a. Write the statement(s) needed to open a file called **'RAfile.dat'**, locate and read the records of the account holder having the account number **'numAcc'** from the file into **'anAccount'**. **[3 marks]**

```
pFile = fopen( "RAfile.dat", "rb+" ) ;                  // 1 mk
fseek( pFile, numAcc*sizeof( Account ), SEEK_SET ) ;    // 1 mk
fread( &anAccount, sizeof( Account ), 1, pFile ) ;      // 1 mk
```

b. Write the statement(s) needed to write back the same records from **'anAccount'** to the file (after possible modifications that you are not concerned with). Note that the position of the read/write head may have been changed by now. **[2 marks]**

```
fseek( pFile, numAcc*sizeof( Account ), SEEK_SET ) ;    // 1 mk

// Note:  can also use the following relative seek
fseek( pFile, -sizeof( Account ), SEEK_CUR ) ;  // Note minus sign


fwrite( &anAccount, sizeof( Account ), 1, pFile ) ;      // 1 mk
```

Q6. [30 MARKS] Write and document a complete **C program** to satisfy the following requirements. Follow the structure of the following pages in answering this question.

Given a **sequential text file** that has the following layout where each line is one record containing the following fields (separated by a single space):

int	**id**	// An unique positive integer.
string	**name**	// Name of the item. (Less than 20 characters, all alphabetic, first character a capital letter).
float	**price**	// Unit price for the item.

Example of a record in the file: **10 Apple 2.99**

The file may contain any number of records up to a MAXSIZE of 5000 (five thousand) records. Each record is in a separate line. You will need to define a self-referential data structure and use a singly linked list to store input record data.

You must write a complete C program to include the following features:

1. Design and declare all variables and functions as appropriate to do the processing as required for this program. (You may use a minimum set of global variables, but excessive use will be penalized).
2. Write all necessary code to:
   a) **Read()** all the records from the input file (called **"groceryList.dat"**) into a singly linked list.
   b) **Display()** the data of all **grocery** items on the screen, using the data in the linked list, according to the order of the inputs.
   d) **Write()** your results back into the file "**groceryList.dat**", thereby over-riding (i.e. destroying) its earlier contents. The records must be written to the file in <u>alphabetic order</u> by the **item name**, as shown in the example below. As each record is written to the file, the corresponding node of the linked list must be deallocated. When the last record has been written, the linked list must be completely empty.

The functions **Read()**, **Display()** and **Write()** must be defined, as described later in the question.

The following sample sequential text file is given to illustrate what your program must achieve:
Sample contents of file **"groceryList.dat"** <u>before</u> processing:

```
2 Tomato 2.50
4 Potato 1.75
1 Chicken 4.00
3 Beef 5.50
5 Fish 7.30
```

Sample contents of file **"groceryList.dat"** <u>after</u> processing:

```
3 Beef 5.50
1 Chicken 4.00
5 Fish 7.30
4 Potato 1.75
2 Tomato 2.50
```

Requirements:
- Your program should be able to handle a file with up to MAXSIZE **records**.
- Write all functions necessary and explain the process of your technique (Write your algorithm clearly and document your code!)
- Document and define your functions according to the instructions provided
- You should assume that all the records have unique **item id's.**

```
// #includes [1 Mark]

#include <stdio.h>
```

```
// structure and type definitions [5 Marks]

// Following are the Grocery Item structure and type definitions

struct Item {
   int Id ;
   char Name[20] ;
   float Price ;
   struct Item * Next ;
}

typedef struct Item Item_t ;
```

```
// function prototypes [4 Marks]

void Read( Item_t * ) ;

void Display( Item_t * ) ;

void Write( Item_t * ) ;
```

```
//int main ()[5 Marks]
int main(  )
{
/*declare required variables, including pointers for linked list
'GList' used to store input records */

   Item_t * GList = NULL ;


/*No menu required - just invoke the function calls to satisfy the
  requested program sequence, then exit the program.*/

   Read( GList ) ;
   Display( GList ) ;
   Write( GList ) ;

   return 0 ;

}
```

```
// Code all necessary functions below [15 Marks]
```

```
//Function Read [6 Marks].
```

```
/*This function reads 'grocery' records from the file
'groceryList.dat'. After creating a 'grocery' structure it inputs
each value of a record to the structure's members. It then adds the
structure to the linked list 'GList'.  The function returns after
reading the entire file.*/
```

```
void Read( Item_t * Root ) {
   FILE * fPtr ;
   Item_t * GLPtr, * Ptr ;

   fPtr = fopen( "groceryList.dat", "r" ) ;
   while( feof( fPtr ) != EOF ) {
      Ptr = (Item_t *) malloc( sizeof( Item_t ) ) ;
      fscanf( fPtr, "%d%s%f", &(Ptr->Id), Ptr->Name, &(Ptr->Price) ) ;
      Ptr->Next = NULL ;
      if( Root != NULL ) {
         GLPtr->Next = Ptr ;
         GLPtr = Ptr ;
      }
      else {
         Root = Ptr ;
         GLPtr = Ptr ;
      }
   }
   fclose( fPtr ) ;
   return ;
}
```

```
//Function Display [4 Marks]
/*Prints the values of all the records in 'GList' (one record in one
line) as "ID NAME PRICE" format.*/
```

```
void Display( Item_t * Root ) {
   Item_t * Ptr = Root ;

   if( Root == NULL )
      printf( "Empty grocery list!\n" ) ;   // Not required

   while( Ptr != NULL ) {
      printf( "%d %s %f\n", Ptr->ID, Ptr->Name, Ptr->Price ) ;
      Ptr = Ptr->Next ;
   }
   return ;
}
```

```
//Function Write [5 Marks]
/*This function writes all the elements in 'GList' into the file
called 'groceryList.dat'. The records must be written to the file
in alphabetic order by the item name. After each record has been
written, the node in 'GList' must be deallocated from the list,
leaving an empty list after the last record is written. */
```

```
void Write( Item_t * Root ) {
   FILE * fPtr ;
   Item_t * Ptr = Root, * Ptr2, * smallPtr ;

   fPtr = fopen( "groceryList.dat", "w" ) ;  // Destroy previous file

   if( Root == NULL ) {
      printf( "No grocery list to file!\n" ) ;
      return ;                                  // Not required
   }

   while( Root != NULL ) {    // Continue until no more nodes in list
      Ptr = Root ;
      smallPtr = Root ;
      Ptr2 = Ptr->Next ;

    // Locate the node with smallest item Name
      while( Ptr2 != NULL ) {
         if( strcmp( Ptr->Name, Ptr2->Name ) > 0 )
            smallPtr = Ptr2 ;
         Ptr2 = Ptr2->Next ;
      }

    // Output the record
      fprintf( fPtr, "%d %s %f\n",
                     smallPtr->ID, smallPtr->Name, smallPtr->Price ) ;

    // Deallocate the node
      if( smallPtr == Root ) {
         Root = Root->Next ;      // Deletion at start of list
         free( smallPtr ) ;
      }
      else {                      // Deletion in rest of list
         for( Ptr=Root ; Ptr->Next != smallPtr ; Ptr = Ptr->Next ) ;
         Ptr->Next = smallPtr->Next ;
         free( smallPtr ) ;
      }
   }

   fclose( fPtr ) ;     // Do not forget to close the file
   return ;
}
```

(This extra page is provided for rough work or continuation of answers)