Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

GNG1106/1506
Final Examination
December 11th, 2007
Solution

Time allowed: 3 hours
Closed book examination
Non-programmable calculators are allowed

Attempt all questions
Questions carry the weights indicated
The total number of points for the examination is 65

Answer the questions in the spaces provided
Use both sides of these sheets if necessary

The French version of the exam can be found on the back of the pages.

Name: _____ **Student Number:** _____

Question 1:	20	
Question 2:	15	
Question 3:	15	
Question 4:	15	
Total:	65	

⇐ Do not write in this box!

Question 1 *Short answer problems*

(20 points – 2 points per question) **Answer briefly.**

Note that a variable of type `int` requires four bytes of memory; a variable of type `float` requires four bytes of memory; and a variable of type `double` requires eight bytes of memory.

a) Is the following a structure type definition or a structure variable declaration?

```
#define MAXVALUES 17
struct valuesStr
{
   int num;
   double values[MAXVALUES];
};
```

Structure Type Definition

b) In C, which character is used to terminate a character string?
The nul character, '\0', ASCII code 0, decimal value 0.

c) Given a pointer declared as `struct valuesStr *vPtr` (see question a) and that `vPtr` contains 10400, what value does `vPtr` contain after the following statement is executed?

```
vPtr++;
```

The value 10540 (10400 + 4 for int + 17*8 for array)

d) What expression provides access to the 5th element of the `values` member of the structure referenced by `vPtr` presented in question c.
`vPtr->values[4]`

e) When the name of the structure variable is used as an argument in a function call, will its value or its reference be passed to the called function?
Its value.

f) What is the effect of the following statements to the file system (assume no error occurs during execution):

```
FILE *fPtr;
fPtr = fopen("exam.txt","w");
fclose(fPtr);
```

To create the file exam.txt if it does not exist, or erase the contents of the file exam.txt if it does exist.

g) Given that the ASCII file `exam.txt` contains the following contents:

```
First line of text.
Second line of text.
Third line of text.
```

What will the character array `string` contain after the execution of the following instructions?

```
FILE *fPtr;
char string[10];
fPtr = fopen("exam.txt","r");
fgets(string, 10, fPtr);
```

Please show ALL characters loaded into the array.

First lin'\0'

h) Consider the following code segment.

```
int *zPtr;
int *sPtr=NULL;
int integer, i;
int z[5] = {1, 2, 3, 4, 5};
sPtr = z;
for (i = 0; i <= 6; i++)
     printf("%d ", *(sPtr+i));
```

Find and describe the error in the loop structure.

There are only 5 elements in the array z. The pointer is incremented past the end of this array, that is, 7 elements. The index i should only have been incremented to 4 instead of 6 (`i <= 4`) in the for loop.

i) Give the function call used to write to the open binary file, referenced by `fPtr` (of type `FILE *`), the contents of the 1-D array, `values`, of type `double` that contains 20 elements.

`fwrite(values,sizeof(double),20,fPtr);`

j) Pointer arithmetic applies to pointer variables. To what other data type can be applied pointer arithmetic?

Pointer arithmetic can also be used with arrays, that is, array names.

Question 2 – Programming Model (15 Points).

For the program on the following page:

(a) Show how the contents of the working memory are changed by the execution of the program. Record successive assignments to a variable/parameter as follows:

Variable name	~~2~~, ~~6~~, ~~4~~, 10

(b) On the terminal screen, show the output of the program, i.e., what is printed on the screen.

Note that the numbers in parentheses give addresses of memory locations;

Program Memory

```
#include <stdio.h>
#include <stdlib.h>

#define FILENAME "exam.txt"
#define MAXVALUES 5
#define MAXLINE 20
struct valuesStr
{
   int num;
   float values[MAXVALUES];
};
/* Prototype */
void getValues(char *, struct valuesStr *, int );
/*-----------------------------------------------*/
int main()
{
    FILE *fPtr;
    char line[MAXLINE];
    struct valuesStr valStruct;
    int i;

    fPtr = fopen(FILENAME,"r"); /* fopen returns 35456 */
    if(fPtr != NULL)
    {
       fgets(line, MAXLINE, fPtr);
       getValues(line, &valStruct, MAXVALUES);
       for(i=0 ; i < valStruct.num ; i++)
          printf("Value %d is %f\n",i,valStruct.values[i]);
       fclose(fPtr);
    }
    else printf("Could not open file %s\n", FILENAME);
    return 0;
}
/*-----------------------------------------------*/
void getValues(char *chPtr, struct valuesStr *vPtr, int maxVals)
{
    float *fPtr = vPtr->values;
    vPtr->num = 0;
    while(*chPtr != '\n' && *chPtr != '\0' && vPtr->num != maxVals)
    {
       *fPtr = atof(chPtr);
       vPtr->num++;
       fPtr++;
       while(*chPtr != '\0' && *chPtr != '\n' && *chPtr != ' ') chPtr++;
       while(*chPtr == ' ') chPtr++;
    }
}
/*-----------------------------------------------*/
```

Terminal Screen

Value 0 is 2.3
Value 1 is 57.3 **(3 points)**
Value 2 is 134.6

Contents of exam.txt file

2.3 57.3 134.6’\n’

Working Memory

valStruct (10222)

num (10222) **~~0~~, ~~1~~, ~~2~~, 3** **(1 point)**

i **~~0~~, ~~1~~, ~~2~~, 3** **(1 point)**

values (10226) **2.3**

(10230) **57.3** **(1 point)**

(10234) **134.6**

(10238)

(10242)

fPtr **35456** **(1 point)**

line (10300) **2.3 57.3 134.6 ‘\n’’\0’** **(2 points)**

maxVals **5** **(1 point)**

vPtr **10222** **(1 point)**

fPtr **~~10226~~, ~~10230~~, ~~10234~~, 10238** **(2 points)**

chPtr **~~10300~~, ~~10301~~, ~~10302~~, ~~10303~~, ~~10304~~, ~~10305~~, ~~10306~~, ~~10307~~, ~~10308~~, ~~10309~~, ~~10310~~, ~~10311~~, ~~10312~~, 10314** **(2 points)**

Question 3 – C Coding (15 Points).

The following pseudo-code specifies an algorithm for counting the occurrence of a word pattern in an ASCII file. Note that the search is case sensitive (uppercase characters are different from lowercase characters) and the word pattern found within other words are counted (for example "the" in therefore would be counted as an occurrence of the word "the"). The algorithm is divided into three subprograms:

- The subprogram *numWords* uses three parameters; *fileNamePtr*, *wordPtr*, and *wordLen* that define the pointer to a file name, the pointer to a word and the length of the word respectively. The *numWord* subprogram reads each line in the file referenced by *fileNamePtr* and uses the *substring* program to find the occurrences of the word referenced by *wordPtr* in each line. It counts up all occurrences of the word in the file and returns this count. If it encounters an error occurs, it returns -1.
- The *substring* subprogram uses three parameters; *strPtr*, *subPtr*, and *len*. It finds the first occurrence of the sub-string pointed to by *subPtr* in the search string pointed to by *strPtr*. The *len* parameter receives the length of the sub-string. The length of the sub-string must be less or equal to the length of the search string. It returns the address of the location where the sub-string was first found in the search string, and NULL if no occurrence of the sub-string is found.
- The *strEqual* subprogram also uses three parameters; *str1Ptr*, *str2Ptr* and *len*. If all *len* characters referenced by *str1Ptr* are equal to the first *len* characters referenced by *str2Ptr*, the subprogram returns TRUE, otherwise it returns FALSE.

Translate the pseudo-code into C functions. Please note that *fgets* and *strlen* are standard C functions.

```
numWord(fileNamePtr, wordPtr, wordLen)
        Open fileNamePtr for reading with pointer fPtr
        If fPtr equals NULL
                Assign -1 to num
        Otherwise
                Assign 0 to num
                Repeat while fgets(line, MAXLINE, fPtr) not equal to NULL
                        If strlen(line) greater or equal to wordLen
                                Assign line to ptr
                                Assign substring(ptr, wordPtr, wordLen) to ptr
                                Repeat while ptr not equal to NULL
                                        Increment num
                                        Assign ptr+wordLen to ptr
                                        If strlen(ptr) greater than or equal to wordLen
                                                Assign substring(ptr, wordPtr, wordLen) to ptr
                                        Otherwise
                                                Assign NULL to ptr
                Close the file with pointer fPtr
        Return num

substring(strPtr, subPtr, len)
        Assign NULL to foundPtr
        Repeat while content pointed by (strPtr-1+len) not equal nul character AND foundPtr equals NULL
                If strEqual(strPtr, subPtr, len) equals TRUE
                        Assign strPtr to foundPtr
                Otherwise
                        Increment strPtr
        Return foundPtr
```

strEqual(str1Ptr, str2Ptr, len)

Assign 0 to cnt

Repeat while content pointed by strPtr1 equals content pointed by strPtr2 AND cnt less than len

Increment strPtr1

Increment strPtr2

Increment cnt

If cnt equals len

Assign TRUE to retValue

Otherwise

Assign FALSE to retValeue

return retValue

0.5 point declaration of symbolic constants

```
#define TRUE 1
#define FALSE 0
#define MAXLINE 200

char *substring(char *, char *, int);
void test(char *, char *);
int strEqual(char *, char *, int);
int numWords(char *, char *, int );

int numWords(char *fileNamePtr, char *wordPtr, int wordLen)
{
    FILE *fPtr;
    int num;
    char line[MAXLINE];
    char *ptr;

    fPtr = fopen(fileNamePtr, "r");
    if(fPtr == NULL)
       num = -1;
    else
    {
       num = 0;
       while(fgets(line,MAXLINE,fPtr) != NULL)
       {
          if(strlen(line) >= wordLen)
          {
             ptr = line;
             ptr = substring(ptr,wordPtr,wordLen);
             while( ptr != NULL)
             {
                num++;
                ptr = ptr+wordLen;
                if(strlen(ptr) >= wordLen)
                   ptr = substring(ptr,wordPtr,wordLen);
                else
                   ptr = NULL;
             }
          }
       }
       fclose(fPtr);
    }
    return(num);
}
```

No marks assigned for prototypes given that not a complete program is being provided.

0.5 point function defnition

0.5 point variable declarations

1 point opening file and if construct for dealing with error in opening file

0.5 point for loop in reading lines from the file.

0.5 point for test of line length

0.5 point for first substring search

0.5 point for loop structure

0.5 point for counting of words

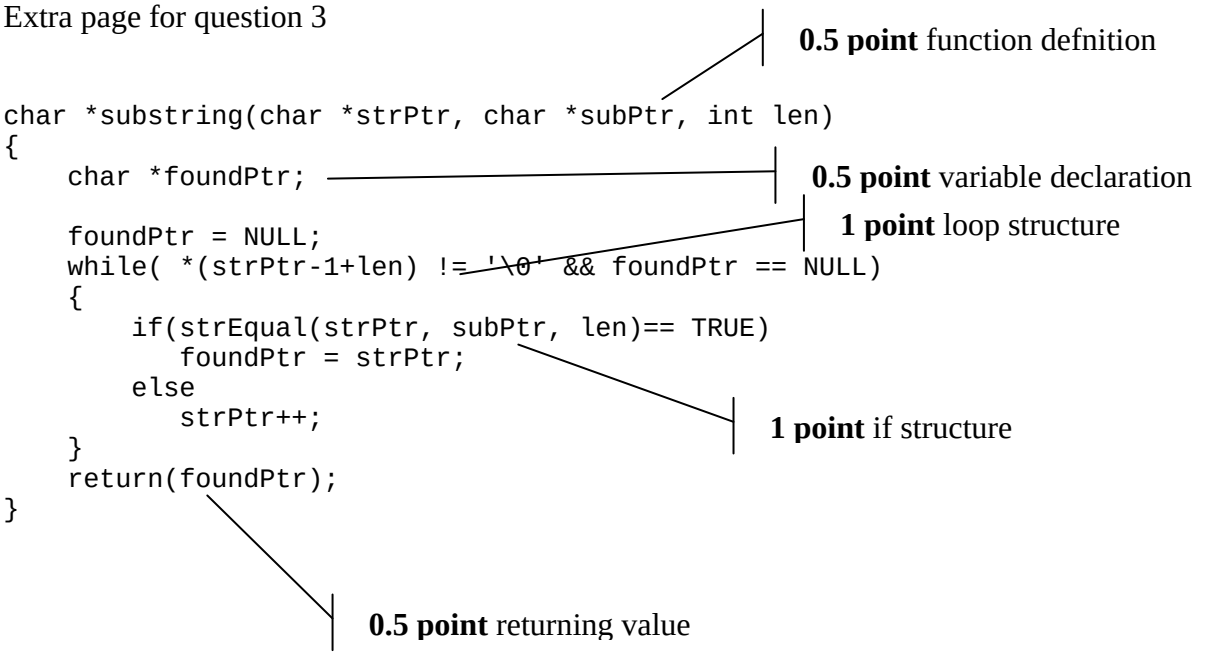
0.5 point for subsequent substring searchs

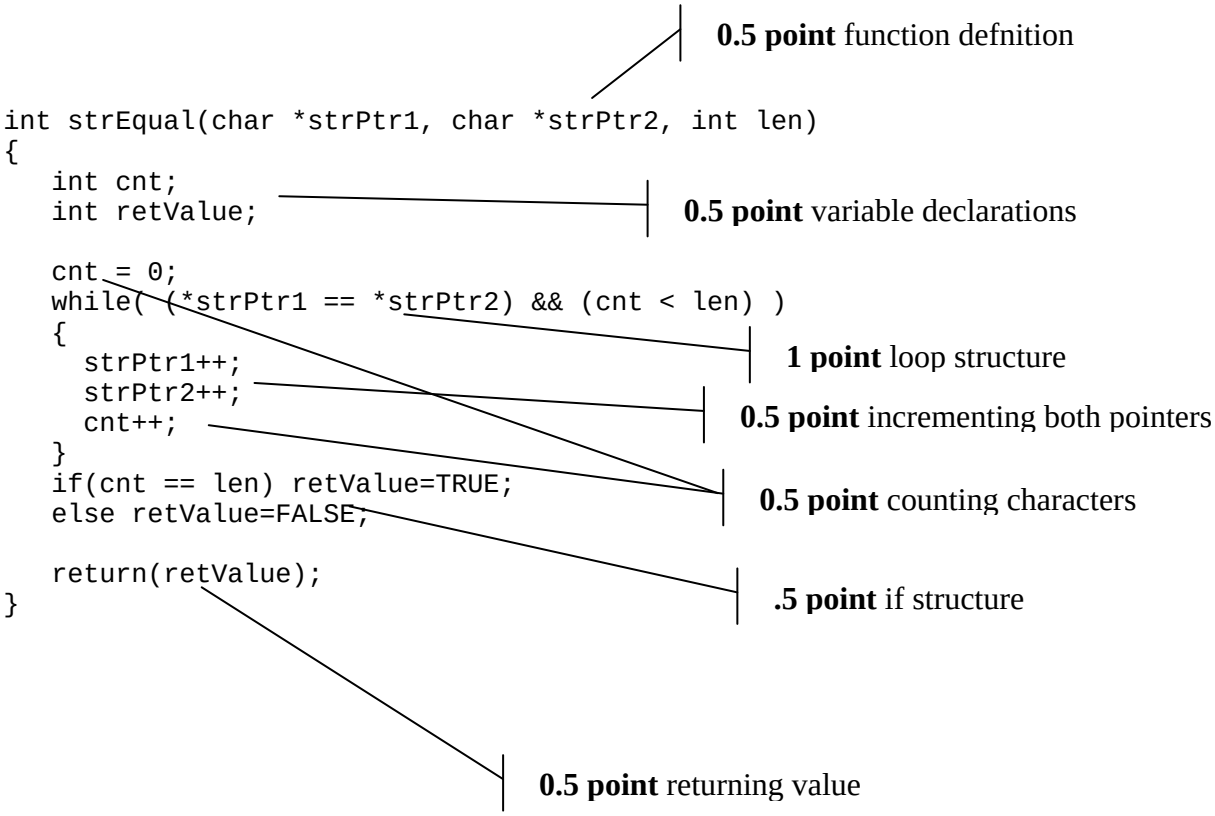
1 point for if structure

0.5 point for closing file

0.5 point for returning number of words found.

Extra page for question 3





Question 4 – Problem Solving Method (15 Points).

Aviation safety is a subject of the highest priority, with aircraft position of paramount importance. Consider a single plane. Two different sensors keep a continuous record of the aircraft location every quarter second. Assume that location is recorded in (*x*, *y*, *z*) coordinates, measured relative to the air traffic control tower. The *x*-coordinate is the east-west location (east positive), the *y*-coordinate is the north-south location (north positive), and the *z*-coordinate is the altitude. All values are in kilometres. The first sensor takes its data from a GPS (global positioning system) and stores it in a binary file saved to the aircraft "black box". Refer to this binary file as *file1*. The second sensor takes its data from the air traffic control RADAR system and saves it to a binary file in the NAV Canada database (In Canada, air traffic control is provided by NAV Canada, a private corporation). Refer to this binary file as *file2*. These two binary files are later compared by a program called *check* to determine data consistency, flagging all times where there is a difference in location greater than some user-defined value.

Assume that the data consist of sequential floating-point values, beginning with the *x*-coordinate, then the *y*-coordinate and then the *z*-coordinate, repeating for an unknown number of time intervals. To compare the two files, the program *check* must prompt the user for a tolerance value, *tolerance*, a floating-point value. Note that *tolerance* is actually a measure of the permissible "distance" between the two recorded locations. The expression for calculating the distance between two locations is

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The *check* program reads the coordinates of a location from each of the two files. It checks whether the calculated distance between the two locations is greater than the specified tolerance. Whenever the difference between the two does exceed the tolerance, the program must record, in the ASCII file *ERR.LOG*, the time at which the tolerance was exceeded, and the difference in location (i.e., the "distance" between the data values), together on one line.

Note that the two files may not be the same size. Your solution should accommodate this possibility.

Answer the following questions, as per our "Problem solving method", presuming that the program *check* consists of a single main program (no other sub-programs are used – standard C functions are permitted and expected).

(a) *Problem Identification and Statement*

Briefly state what objective your software should accomplish.

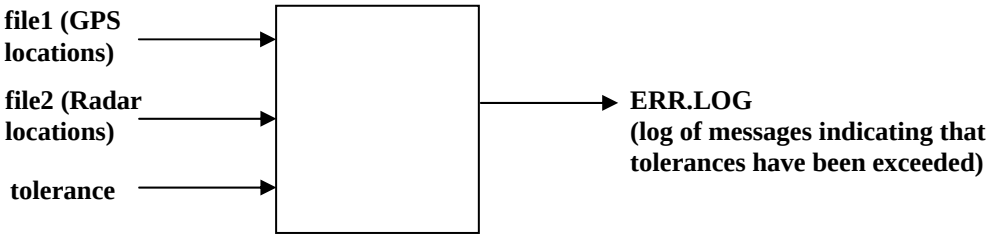
(1 point)

To create the program that compares the corresponding coordinate locations from each of the files *file1* and *file2*. When the distance between the locations from each file exceeds a tolerance value provided by the user, a message is written into a log file to provide the time and the distance between the locations.

(b) *Gathering of Information and Input/Output Description*

Summarize the critical information and provide an input/output diagram.

Distance between locations is computed as $\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ **.** **(1 point)**



(1 point)

(c) *Test Cases (Hand-Solved Examples) and Algorithm Design*

(i) As a test case, provide the content of two short files of different length for *file1* and *file2*. Provide a specific tolerance such that the values in the files illustrate the two different logical options that your software must handle. Provide the resulting output expected in the ERR.LOG file.

(2 points: 1 point for contents of file and option where tolerance is not exceeded, and 1 point for contents of ERR.LOG and option where tolerance is exceeded).

Given a tolerance of 0.35:

Contents of file1			**Contents of file2**			**Time index**	**Difference**
2	4	5	2.1	4.1	5.3	0	0.331662
2.1	4.2	5	2.15	4.25	5.2	0.25	0.212132
2.4	4.4	5	2.2	4.55	5.3	0.5	0.390512
2.5	4.6	5	2.51	4.55	5.1	0.75	0.11225
			2.65	4.71	5.2	1.0	

The contents of ERR.LOG will be:
Time 0.5, distance 0.390512

(ii) Describe the algorithm in your own words, without using pseudocode or C code.
Mention ALL logical steps the software must undertake.

(4 points: 0.5 for (a), 0.5 for (b), 1.5 for (c), 1.5 for (d) – answer needs to address all info from each point not necessarily in the same point form).

a) **Open the files *file1* and *file2* for reading and file *ERR.LOG* for writing.**
b) **Request from the user a value for tolerance.**
c) **Read 3 coordinates from *file1* and the corresponding 3 coordinates from *file2*. Compare the locations defined by these three coordinates by calculating the distance between the two locations using the distance formula. If the If the distance is greater than the tolerance value, then write the time index and the distance to the ERR.LOG file**
d) **Repeat step (c) for each of the coordinates in the files until no more coordinates are available from either file. The time index for each set of coordinates is defined by setting a time variable to 0 and incrementing by 0.25 seconds each time a new set of coordinates is read.**

(iii) Express your algorithm in detail, using PSEUDOCODE ONLY (credit will not be given for C code)

(6 points as shown)

```
Assign TRUE to flag
Open "file1" for reading with f1Ptr
If f1Ptr equals NULL
        Print "Cannot open file1 for reading"
        Assign FALSE to flag
If flag equals TRUE
        Open "file2" for reading with f2Ptr
        If f2Ptr equals NULL
                Print "Cannot open file2 for reading"
                Assign FALSE to flag
If flag equals TRUE
        Open "ERR.LOG" for writing with fEPtr
        If fEPtr equals NULL
                Print "Cannot open ERR.LOG for writing"
                Assign FALSE to flag
If flag equals TRUE
        Print "Please provide a tolerance value: "
        Read value into tolerance
        Assign 0 to time
        Repeat
                If fread(address of x1, sizeof(x1), 1, f1Ptr) equals -1
                        Assign FALSE to flag
                Otherwise
                        fread(address of y1, sizeof(y1), 1, f1Ptr)
                        fread(address of z1, sizeof(z1), 1, f1Ptr)
                If fread(address of x2, sizeof(x2), 1, f2Ptr) equals -1
                        Assign FALSE to flag
                Otherwise
                        fread(address of y2, sizeof(y2), 1, f2Ptr)
                        fread(address of z2, sizeof(z2), 1, f2Ptr)
                If flag equals TRUE
                        Assign sqrt((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2) to distance
                        If distance greater than tolerance
                                Write "Time ", time, "distance ", distance to fEPtr
                        Assign time+0.25 to time
        While flag is TRUE
```

1 point for dealing with errors in the program. The solution uses a flag to indicate an error condition that is changed to FALSE when an error is encountered (when flag is FALSE no other action is taken). Another possibility is to use Otherwise clauses in the if structure.

1 point opening files. Need to deal with files that cannot be opened.

1 point for obtaining tolerance value from the user. Note that invalid values are not checked.

1 point for generating the time index

1 point for reading in the coordinate values. Need to check if end of file is met in both files. Assumption has been made that no errors exist in the file so if the x coordinate can be read, the y and z coordinates will be present. Some mechanism to terminate the loop is required. The solution uses the error *flag* to for this purpose.

1 point for calculating the index, comparing to the tolerance and writing to the ERR.LOG file.

The *fread* call has been used to represent the reading of the contents of the binary files (mimics the use of standard functions as used in Question 3 pseudo-code). Use of any wording that conveys the call is also acceptable..