EECE.2160: ECE Application Programming

Fall 2017

Exam 3
December 16, 2017

Name: _____

Lecture time (circle 1): *8-8:50 (Sec. 201)* *12-12:50 (Sec. 203)* *1-1:50 (Sec. 202)*

For this exam, you may use only one 8.5" x 11" double-sided page of notes. All electronic devices (e.g., calculators, cell phones) are prohibited. If you have a cell phone, please turn off your ringer prior to the start of the exam to avoid distracting other students.

The exam contains 4 questions for a total of 100 points, plus a 10 point extra credit question. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer. In particular, note that:

- Questions 1b and 2b require you to complete short functions. We have provided comments to describe what each function should do and written some of the code.
    - Note that each function contains both lines that are partially written (for example, a `printf()` call missing the format string and expressions) and blank spaces in which you must write additional code. **You must write all code required to make each function work as described—do not simply fill in the blank lines.**
    - Each function is accompanied by one or more test cases. Each test case is an example of how the function should behave in one specific case—**it does not cover all possible results of running that function.**
    - You can solve each of these questions using only the variables that have been declared, but you may declare and use other variables if you want.
- For this exam, unlike Exams 1 and 2, you may attempt the extra credit problem even if you have not at least partially completed all other problems on the exam.
- The extra pages accompanying the exam hold structure and function definitions, as well as test cases for functions you must write—info needed for questions 1a, 1b, 2b, and 5.

You will have 3 hours to complete this exam.

Q1: Structures	/ 34
Q2: File I/O	/ 25
Q3: Character & line I/O	/ 17
Q4: Dynamic allocation & linked lists	/ 24
TOTAL SCORE	/ 100
Q5: EXTRA CREDIT	/ 10

1. (34 points) ***Structures***

a. (14 points) Show the output of the short program below exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so I can easily recognize your final answer.

The `Struct1` structure definition is in the extra document provided with the exam.

```
void f1(Struct1 *s1ptr, int a, int b) {
   s1ptr->v1 = a + b;
   s1ptr->v2 = a * b;
}

void f2(Struct1 *s1ptr) {
   printf("%d %d\n", s1ptr->v1, s1ptr->v2);
}

int main() {
   Struct1 x = {5, 3};
   Struct1 y, z;

   f1(&y, x.v2, x.v1);
   f1(&z, y.v2, x.v2);
   f2(&x);
   f2(&y);
   f2(&z);

   return 0;
}
```

1 (continued)

b. (20 points) Complete the function described below:

```
double semesterGPA(GradeList *glp);
```

This function reads a `GradeList` structure, pointed to by argument `glp`, and calculates and returns the grade point average, in which each grade is weighted by the number of credits per class. Each grade is represented by a letter (A, B, C, D, F)—no + or – grades—and is worth the following number of points: A = 4 points, B = 3 points, C = 2 points, D = 1 point, F = 0 points.

The `GradeList` structure definition and test cases for this function are in the extra document provided with the exam.

```
double semesterGPA(GradeList *glp) {
   int i;       // Loop index
   double GP;   // Total grade points
   int cred;    // Total credits

   // Initialize variables



   // Go through all elements of arrays in grade list

   for (_____________________________________________________) {
      // Add credits for current class


      // Add grade points for current class

      switch (_____________________________________________) {









      }
   }
   // Return final weighted GPA

   return _____________________________________________________;
}
```

2. (25 points) ***File I/O***

a. (5 points) A program must read a binary file, `f1.bin`, which contains a set of integer values, store those values in an array, and print the array contents to the screen. The file has at least 1 value in it and may hold up to 200 values.

Which of the choices below will always perform the operations described above correctly, regardless of the file size? Assume the file always opens with no errors.

Clearly indicate your response by marking the one choice you think best answers the question.

i.
```
int x;
FILE *fp = fopen("f1.bin", "rb");
fread(&x, sizeof(int), 1, fp);
printf("%d", x);
```

ii.
```
int i;
int arr[200];
FILE *fp = fopen("f1.bin", "rb");
fread(arr, sizeof(int), 200, fp);
for (i = 0; i < 200; i++)
   printf("%d\n", arr[i]);
```

iii.
```
int i, n;
int arr[200];
FILE *fp = fopen("f1.bin", "rb");
n = fread(arr, sizeof(int), 200, fp);
for (i = 0; i < n; i++)
   printf("%d\n", arr[i]);
```

iv.
```
int i, n;
int arr[200];
FILE *fp = fopen("f1.bin", "rb");
n = fread(arr, sizeof(int), 200, fp);
for (i = 0; i < n; i++)
   fprintf(fp, "%d\n", arr[i]);
```

v.
```
int i, n;
int arr[200];
FILE *fp = fopen("f1.bin", "rb");
n = fwrite(arr, sizeof(int), 200, fp);
for (i = 0; i < n; i++)
   printf("%d\n", arr[i]);
```

2 (continued)

b. (20 points) Complete the function described below:

`TicTacToe(char *inName, char *outName);`

This function takes two arguments: strings representing the names of an input (`inName`) and output file (`outName`). The input text file represents moves in a game of tic tac toe—each of the 9 lines contains a letter (`'X'` or `'O'`) and a row and column number in the 3x3 tic tac toe board.

The function should read each input line, storing the letter into the row and column specified, and then print the whole 3x3 board to the named output file. You may assume both files always open without errors. Test cases for this function are in the extra pages supplied with the exam.

```
void TicTacToe(char *inName, char *outName) {
   FILE *inFP, *outFP;    // Input/output file pointers
   char board[3][3];      // Tic tac toe board
   int i, j, k;           // Array/loop indexes
   char XO;               // Input letter--'X' or 'O'

   // Open input file


   // Read each row from file, then store letter in board

   for (_____________________________________________________) {

      fscanf(_____________________________________________________);

   }
   // Open output file


   // Print board to output file—one board row per line of output

   for (_____________________________________________________) {

      for (_____________________________________________________) {

         fprintf(_____________________________________________________);
      }

      fprintf(_____________________________________________________);
   }
   // Close files


}
```

3. (17 points) ***Character & line I/O***

a. (12 points) Show the output of the short program below exactly as it will appear on the screen, assuming the user types the following input at the command line (including a newline character after the last digit (6)):

```
97-8-216
```

Be sure to clearly indicate spaces between characters when necessary. You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so I can easily recognize your final answer.

```
int main() {
   char c1, c2, c3;

   while ((c1 = getchar()) != '\n') {
      c2 = fgetc(stdin);
      if (c2 == '\n')
         ungetc(c2, stdin);
      else if (c2 != '-') {
         c3 = getchar();
         if (c3 == '\n')
           ungetc(c3, stdin);
      }

      printf("%c %c %c\n", c3, c2, c1);
   }

   return 0;
}
```

3 (continued)

b. (5 points) You are given the short program below:

```
int main() {
   char s1[10];
   char s2[10];

   scanf("%s", s1);
   fgets(s2, 10, stdin);

   printf("%s\n", s1);
   printf("%s\n", s2);

   return 0;
}
```

What will the program print if the user enters the line below, including a newline at the end?

```
   You're almost done
```

Clearly indicate your response by marking the <u>one</u> choice you think best answers the question.

i.
```
You're almost done
```

ii.
```
You
're almost
```

iii.
```
You're
 almost done
```

iv.
```
You're
 almost d
```

v.
```
You'll be done soon
```

4. (24 points) ***Dynamic allocation & linked lists***

a. (14 points) Show the output of the short program below exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so I can easily recognize your final answer.

```
int main() {
   int i;
   int *p;
   int s[3] = {4, 7, 5};

   p = (int *)calloc(s[0], sizeof(int));
   for (i = 0; i < s[0]; i++)
      printf("%d ", p[i]);
   printf("\n");

   p = (int *)realloc(p, s[1] * sizeof(int));
   for (i = 0; i < s[1]; i++) {
      p[i] = s[i%3];
      printf("%d ", p[i]);
   }
   printf("\n");

   p = (int *)realloc(p, s[2] * sizeof(int));
   for (i = 0; i < s[2]; i++) {
      p[i] = p[i] / 2;
      printf("%d ", p[i]);
   }
   printf("\n");

   return 0;
}
```

4 (continued)

b. (5 points) Say you have a linked list built using the structure definition below for each node:

```
typedef struct node {
     int value;           // Data
     struct node *next;   // Pointer to next node
} LLnode;
```

Assume you have a pointer, `list`, which points to the first node in the linked list. You are also given two other variables, which are declared as follows:

```
LLnode *p;           // General node pointer
int n = 0;           // Counter
```

Which of the following code snippets will set `n` equal to the total number of nodes in the list? Clearly indicate your response by marking the <u>one</u> choice you think best answers the question.

i.
```
p = list;
n = sizeof(p);
```

ii.
```
p = list;
while (p == NULL) {
   p = p->next;
   n++;
}
```

iii.
```
p = list;
while (p != NULL) {
   n += p->val;
   p = p->next;
}
```

iv.
```
p = list;
while (p != NULL) {
   p = p->next;
   n++;
}
```

4 (continued)

c. (5 points) Circle one (or more) of the choices below that you feel best "answers" this "question."

i. "Thanks for the free points."

ii. "This is the best final exam I've taken today."

iii. "It's a good thing we have this exam today—I couldn't think of anything better to do on a Saturday afternoon."

iv. None of the above.

5. (10 points) ***EXTRA CREDIT***

Everyone may attempt this problem, **even if you have not at least partially solved the rest of the exam.** Remember, you can earn partial credit for a partial solution to this problem.

Show the output of the short program below exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary. *(Hint: The final output is easily readable, but you must show at least some work to get credit.)*

Definitions for the `Node` structure, `add1()`, and `add2()` are on the extra handout.

```
int p4b() {
   int i;
   Node arr[17] = { {3,  'u', NULL}, {16, 'a', NULL}, {7,  'o', NULL},
                    {10, 'b', NULL}, {1,  'y', NULL}, {14, 'e', NULL},
                    {0,  ' ', NULL}, {18, 'k', NULL}, {2,  'y', NULL},
                    {8,  'r', NULL}, {0,  'o', NULL}, {12, 'r', NULL},
                    {6,  'j', NULL}, {9,  ' ', NULL}, {1,  'n', NULL},
                    {20, '!', NULL}, {2,  'E', NULL} };
   Node *first = NULL;
   Node *n;

   for (i = 0; i < 17; i++) {
      if (i % 2 == 0)
         first = add1(first, &arr[i]);
      else
         first = add2(first, &arr[i]);
   }

   n = first;
   while (n != NULL) {
      printf("%c", n->ch);
      n = n->next;
   }
   printf("\n");

   return 0;
}
```

5 (continued) ***ADDITIONAL SPACE TO SOLVE EXTRA CREDIT PROBLEM***