EECE.2160: ECE Application Programming

Fall 2017

Exam 3 Solution

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;
}
```

Sets v1 and v2 members of structure to sum and product of a and b

Prints structure contents

x.v1 = 5, x.v2 = 3

y.v1 = x.v2 + x.v1 = 3 + 5 = 8
y.v2 = x.v2 * x.v1 = 3 * 5 = 15

z.v1 = y.v2 + x.v2 = 15 + 3 = 18
z.v2 = y.v2 * x.v2 = 15 * 3 = 45

OUTPUT:

```
5 3
8 15
18 45
```

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
   GP = 0;
   cred = 0;

   // Go through all elements of arrays in grade list
   for (i = 0; i < glp->nClass; i++) {
      // Add credits for current class
      cred = cred + glp->nCred[i];

      // Add grade points for current class
      // NOTE: You can ignore 'F' since it's worth 0 points
      switch (glp->grd[i]) {
         case 'A':
            GP += 4 * glp->nCred[i];
            break;
         case 'B':
            GP += 3 * glp->nCred[i];
            break;
         case 'C':
            GP += 2 * glp->nCred[i];
            break;
         case 'D':
            GP += glp->nCred[i];

      }
   }

   // Return final weighted GPA
   return GP / cred;
}
```

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);
```
Always reads just one integer

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]);
```
Doesn't work if file contains <200 values

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]);
```
Tries to print results to input file through file pointer fp

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]);
```
Tries to write, not read, input file

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
   inFP = fopen(inName, "r");


   // Read each row from file, then store letter in board
   for (k = 0; k < 9; k++) {

      fscanf(inFP, " %c %d %d", &XO, &i, &j);
      board[i][j] = XO;
   }
   // Open output file
   outFP = fopen(outName, "w");


   // Print board to output file—one board row per line of output
   for (i = 0; i < 3; i++) {
      for (j = 0; j < 3; j++) {
         fprintf(outFP, "%c ", board[i][j]);
      }
      fprintf("\n");
   }
   // Close files
   fclose(inFP);
   fclose(outFP);
}
```

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;
}
```

Notes on solution:
Each loop iteration reads up to three characters, stopping if (1) any of the characters is a newline (which is placed back in the input so the loop condition becomes false), or (2) the second character is `'-'` (`c3` only changes if `c2 != '-'`). The three characters are printed in reverse order at the end of each iteration (`c3, c2, c1`)

The program takes four iterations to process all input characters:

- Iteration 1: c1 = '9', c2 = '7', c3 = '-'
- Iteration 2: c1 = '8', c2 = '-' (so nothing new is read into c3, which remains '-')
- Iteration 3: c1 = '2', c2 = '1', c3 = '6'
- Iteration 4: c1 = '\n' → loop ends

OUTPUT:

```
- 7 9
- - 8
6 1 2
```

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;
}
```

Reads up to first space → s1 = "You're"
Reads up to 9 characters, since there are >9 characters before newline → s2 = " almost d" (fgets() reads spaces!)

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 one 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));      p = {0, 0, 0, 0}
   for (i = 0; i < s[0]; i++)
      printf("%d ", p[i]);
   printf("\n");

   p = (int *)realloc(p, s[1] * sizeof(int)); p = {0, 0, 0, 0,
                                                  ?, ?, ?}
   for (i = 0; i < s[1]; i++) {
      p[i] = s[i%3];                  p[0] = p[3] = p[6] = 4
                                      p[1] = p[4] = 7
                                      p[2] = p[5] = 5
      printf("%d ", p[i]);
   }
   printf("\n");

   p = (int *)realloc(p, s[2] * sizeof(int)); p = {4, 7, 5, 4, 7}
   for (i = 0; i < s[2]; i++) {
      p[i] = p[i] / 2;                        p = {2, 3, 2, 2, 3}
      printf("%d ", p[i]);
   }
   printf("\n");

   return 0;
}
```

OUTPUT:

```
0 0 0 0
4 7 5 4 7 5 4
2 3 2 2 3
```

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 one choice you think best answers the question.

i.
```
p = list;
n = sizeof(p);
```
n = size of pointer, not whole list

ii.
```
p = list;
while (p == NULL) {
   p = p->next;
   n++;
}
```
Loop doesn't run unless list is empty (p == NULL), and program will crash when you attempt to access p->next

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

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.

All of the above are "correct."

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;
}
```

Solution notes: Each array entry is a linked list `Node`, with an integer value that may be used to place it in the list, and a character that will be printed when the list is traversed at the end. Each node's `next` pointer starts as NULL but is overwritten when the node is added to the list.

The two add functions are basically the `addNode()` and `addSortedNode()` functions we wrote in class, only the nodes have already been allocated and filled with data. `add1()` adds a node to the beginning of the list, while `add2()` uses the `val` field in the node to sort the new node being added. The `for` loop in `main()` alternates between the two add functions.

The next page shows how the list is updated, showing the node that's added each time, the function used to add it, and the set of characters in the list after tha.

Node being added	add func. used	Characters in list
{3, 'u', NULL}	add1 (unordered)	u
{16, 'a', NULL}	add2 (ordered)	ua
{7, 'o', NULL}	add1	oua
{10, 'b', NULL}	add2	ouba
{1, 'y', NULL}	add1	youba
{14, 'e', NULL}	add2	youbea
{0, ' ', NULL}	add1	youbea
{18, 'k', NULL}	add2	youbeak
{2, 'y', NULL}	add1	y youbeak
{8, 'r', NULL}	add2	y yourbeak
{0, 'o', NULL}	add1	oy yourbeak
{12, 'r', NULL}	add2	oy yourbreak
{6, 'j', NULL}	add1	joy yourbreak
{9, ' ', NULL}	add2	joy your break
{1, 'n', NULL}	add1	njoy your break
{20, '!', NULL}	add2	njoy your break!
{2, 'E', NULL}	add1	Enjoy your break!

The `while` loop at the end of the program simply traverses the list and prints every character, so the output is:

```
Enjoy your break!
```