EECE.2160: ECE Application Programming

Spring 2016

Exam 3
May 6, 2016

Name: _____

Section (circle 1): **201** *(8-8:50, P. Li)* **202** *(12-12:50, M. Geiger)*

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, PDAs) are prohibited. If you have a cell phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 3 questions for a total of 100 points. 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:

- Question 3 has three parts, but you are only required to complete two of the three parts.
  - You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**
- For each part of Question 3, you must complete a short function. I have provided comments to describe what your function should do and written some of the code for you.
  - 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 program 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 using that function.**
- You can solve each part of Question 3 using only the variables that have been declared, but you may declare and use other variables if you want.

You will have 3 hours to complete this exam.

Q1: Multiple choice	/ 20
Q2: Structures	/ 40
Q3: Strings	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

1. (20 points, 4 points per part) ***Multiple choice***
For each of the multiple choice questions below, clearly indicate your response by circling or underlining the one choice you think best answers the question.

a. Say you have a file named "myfile.txt" that initially contains the following data:

```
Have a great summer vacation.
```

What does the following program print?

```
int main() {
   char ch;
   FILE *fp;

   fp = fopen("myfile.txt", "a");
   fprintf(fp, " Bye.");
   fclose(fp);

   fp = fopen("myfile.txt", "w");
   fprintf(fp, " Just kidding.");
   fclose(fp);

   fp = fopen("myfile.txt", "r");
   while ((ch = fgetc(fp)) != EOF)
      fprintf(stdout, "%c", ch);
   fclose(fp);

   return 0;
}
```

i. `Have a great summer vacation. Bye. Just kidding.`

ii. `Have a great summer vacation. Just kidding.`

iii. `Have a great summer vacation. Bye.`

iv. `Just kidding.`

v. This program doesn't actually compile.

1 (continued)

b. You have a program that contains an array declared as:

```
int arr[40];
```

Which of the following code snippets would correctly read the contents of this array from a file?

i.
```
FILE *fp = fopen("input.txt", "rb");
fread(arr, sizeof(int), 40, fp);
```

ii.
```
FILE *fp = fopen("input.txt", "rb");
fread(fp, sizeof(int), 40, arr);
```

iii.
```
FILE *fp = fopen("input.txt", "r");
fscanf(fp, "%lf", arr);
```

iv.
```
FILE *fp = fopen("input.txt", "rb");
fwrite(arr, sizeof(double), 40, fp);
```

1 (continued)

c. You are writing a program that should repeatedly read input characters until a space is entered, then read the space and the rest of the line that follows (up to 49 total characters) into an array:

Which of the following code sequences correctly read this input?

i.
```
char c;
char inp[50];
while ((c = getchar()) != ' ')
     ;    // Do nothing—empty loop
fgets(inp, 50, stdin);
```

ii.
```
char c;
char inp[50];
while ((c = getchar()) != ' ')
     ;    // Do nothing—empty loop
ungetc(c, stdin);
fgets(inp, 50, stdin);
```

iii.
```
char c;
char inp[50];
putchar(c);
fputs(inp, stdout);
```

iv.
```
char c;
char inp[50];
printf("%c %s\n", c, inp);
```

1 (continued)

d. Which of the following choices dynamically allocates a <u>two-dimensional</u> array of 300 integers?

i.
```
p = (int *)malloc(300 * sizeof(int));
```

ii.
```
p = (int *)calloc(300 * sizeof(int));
```

iii.
```
p = (double **)malloc(20 * sizeof(double *));
for (i = 0; i < 20; i++)
     p[i] = (double *)malloc(15 * sizeof(double));
```

iv.
```
p = (int **)malloc(30 * sizeof(int *));
for (i = 0; i < 30; i++)
     p[i] = (int *)malloc(10 * sizeof(int));
```

v. None of the above

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

i. "Thanks for the free points."

ii. "I don't REALLY have to answer the last two questions, do I?"

iii. "This is the best final exam I've taken tonight."

iv. None of the above.

2. (40 points) ***Structures***

For each short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary. Assume all necessary libraries are included.

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 that I can easily recognize your final answer.

a. (12 points) *See the provided extra sheet for the `Fish` structure definition*

```
void func1(Fish *f) {
    printf("%s %d %.2lf\n", f->name, f->age, f->length);
}

Fish *func2(Fish arr[], int size) {
    int i, flag = 0;

    for (i = 0; i < size-1; i++) {
        if (arr[i].length < arr[i+1].length)
            flag = i+1;
    }
    return &arr[flag];
}

void main() {
   Fish myFishTank[3] = { {"Cod", 2, 2.543},
                          {"Shark", 3, 22.738},
                          {"Guppy", 4, 1.731} };
   int i;

   for (i = 0; i < 3; i++)
      func1(&myFishTank[i]);

   func1(func2(myFishTank, 3));
}
```

2 (continued)

b. (14 points) *See the provided extra sheet for the `Auto` and `AutoPart` structure definitions*

```
void printPart(AutoPart *part) {
   printf("%s\n", part->name);
   printf("%s\n", part->type.autoMake);
   printf("%d\n", part->type.year);
   printf("%d\n\n", part->inStock);
}

int main() {
   int i;
   AutoPart partList[4] = {  {"engine", {"Chevy", 1989},    1},
                             {"AC",     {"Porsche", 2016},  0},
                             {"brake",  {"Ford", 2010},     1},
                             {"tire",   {"Stroller", 2007}, 0} };

   AutoPart *partPtr = partList;
   printPart(partPtr);

   for (i = 0; i < 4; i++) {
      if (partList[i].type.year > 2000 && partList[i].inStock)
         printPart(&partList[i]);
   }

   partPtr = &partList[i-1];
   partPtr->type.year = 1999;
   partPtr->inStock = 1;
   printPart(partPtr);
}
```

2 (continued)

c. (14 points) *See the provided extra sheet for the `LLnode` structure definition, as well as the `addNode()` and `printList()` function definitions*

```
LLnode *func(LLnode *list) {
   LLnode *prev = NULL;
   LLnode *cur  = list;
   LLnode *next = NULL;
   while (cur != NULL) {
      next = cur->next;
      cur->next = prev;
      prev = cur;
      cur = next;
   }
   return prev;
}

void main() {
   LLnode *head = NULL;
   char myStr[] = "Exam 3";
   char temp;
   int i = 0;
   while ( (temp = myStr[i++]) != 0 ) {
      if (temp != ' '){
         printf("%c\n", temp);
         head = addNode(head, temp);
      }
   }
   printList(head);
   printList(func(head));
}
```

f. (40 points, 20 per part) ***Strings***

For each part of this problem, you are given a short program to complete. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the spaces provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

Remember, you must write all code required to make each function work as described—**do not simply fill in the blank lines for full credit.** Also, remember that each example provided is only applicable in one specific case—**it does not cover all possible results of using that function.**

a. `int mystrncmp(char *s1, char *s2, int n);`

Write your own version of the `strncmp()` function. Remember that this function does a character-by-character comparison of the first `n` characters of strings `s1` and `s2`. If all of those characters match, the function returns 0. Otherwise, the return value (positive or negative) is based on the first pair of characters that differ—for example:

- If `s1 = "ABC"` and `s2 = "ABD"`, the function returns a negative value.
- If `s1 = "This"` and `s2 = "That"`, the function returns a positive vale.

```
int mystrncmp(char *s1, char *s2, int n) {
   int i;
   // Loop to go through first n characters of strings
   for (_____________________________________________________) {
      // Test for s1 not equal to s2. Return negative value if s1
      // "less than" s2, positive value if s1 "greater than" s2
      if (_____________________________________________________) {









      }
      // Reached end of both strings, so they must be equal
      else if (_____________________________________________) {
         return ______;
   }
   // If you reach this point, first n characters must be equal
   return _____;
}
```

3 (continued)
b. `void followCmd(char *cmd, char *str);`

This function takes two strings as input; `cmd` is a command describing how to handle `str`. Valid commands take one of two formats:

- `"Read n characters"`, where `n` is replaced by a single digit integer. If `cmd` holds this command, the function should read up to `n` characters, including spaces, from the standard input into `str`.
- `"Print n characters"`, where `n` is replaced by a single digit integer. If `cmd` holds this command, the function should print the first `n` characters of `str` to the standard output.

All other commands are invalid; the function should print an error message in those cases. You may assume that if the first word of the command matches either `"Read"` or `"Print"`, then the rest of the command is formatted appropriately.

Note: It may help to recall that the ASCII value of '0' is 48.

```
void followCmd(char *cmd, char *str) {
   int i;       // Index variable

   // Test for and handle "Read" command--read up to n characters
   //   from standard input into str
   if (_____________________________________________________) {









   }

   // Test for and handle "Print" command--print first n
   //   characters of str to standard output
   else if (_____________________________________________________) {









   }

   // In all other cases, print an error message
   else
      printf("Error: invalid command\n");
}
```

3 (continued)

c. `void removeChar(char *str, char c);`

This function should remove all occurrences of the character `c` from the string `str`. Remember that, since the string is passed by address, changes made to `str` inside the function are seen outside the function. For example, if the string `s1 = "This is a message"`:

- After calling `removeChar(s1, 'i')` → `s1 = "Ths s a message"`
- After calling `removeChar(s1, ' ')` → `s1 = "Thisisamessage"`
- After calling `removeChar(s1, 's')` → `s1 = "Thi i a meage"`

Hint: to access part of a string (substring) after the first character, use the address of the desired starting character. For example, given the original `s1` above, `&s1[5]` can access the substring `"is a message"`. Using substrings with built-in string functions can simplify your solution.

```
void removeChar(char *str, char c) {
   int i;       // Index variable

   // Initialize variables as needed




   // Loop that ensures you cover all characters in str
   while (_____________________________________________________) {


      // If current character matches c, remove it
      if (_____________________________________________________) {









      }
      // Otherwise, update i to move on to the next character
      else {



      }
   }
}
```