16.216: ECE Application Programming

Fall 2012

Exam 3 Solution

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 choice you think best answers the question.

a. You are given the following pointer declaration: `double *dptr;`
Which of the following statements correctly allocates an array of 25 values of type `double` and stores a pointer to that array in `dptr`?

i. `dptr = double arr[25];`

ii. `dptr = (double *) malloc(25);`

iii. `dptr = (double *) malloc(double arr[25]);`

iv. ***`dptr = (double *) malloc(25 * sizeof(double));`***

v. `malloc(dptr, 25 * sizeof(double));`

b. The code below dynamically allocates space for a two-dimensional array:

```
double **arr;
arr = (double **) malloc(10 * sizeof(double *));
for (i = 0; i < 10; i++)
   arr[i] = (double *) malloc(5 * sizeof(double));
```

How many values are stored in this array, and how are they organized?

i. 15 values, organized as one array of 10 values and a separate array of 5 values.

ii. 50 values, organized as an array with 10 rows and 5 columns.

iii. 50 values, organized as an array with 5 rows and 10 columns.

iv. 100 values, organized as an array with 10 rows and 10 columns.

v. 216 values, and they aren't organized at all.

1 (continued)

c. Given the code sequence below:

```
int *list = (int *)calloc(5, sizeof(int));
list[1] = 7;
list[3] = -5;
list[4] = 17;
for (i = 0; i < 5; i++)
   printf("%d ", list[i]);
```

What will this program print?

i. `7 -5 17`

ii. `0 7 -5 17`

iii. `0 0 0 0 0`

iv. ***`0 7 0 -5 17`***

v. Program prints nothing—since `list[0]` is not initialized, program will crash as soon as this value is accessed inside the loop.

d. You have the following variables:

```
char *str;     // Dynamically allocated character array
int n;         // Current size of array
```

Assuming `str` points to a dynamically allocated array holding `n` characters, which of the following statements will reallocate space for this array so that the size of the array doubles, but all values currently stored in the array remain the same?

i. `str = (char *)malloc(n);`

ii. `str = (char *)malloc(n * 2);`

iii. `str = (char *)realloc(str, n);`

iv. ***`str = (char *)realloc(str, n * 2);`***

v. `str = new int [n*2];`

1 (continued)

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. "I know we're dropping the lowest program score from our final grade, but I think we should drop the lowest exam score, too."

iv. None of the above.

<u>Any answer is "correct."</u>

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.

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. (14 points)

```
typedef struct {
   int p1;
   int p2;
   char g[3];
} exam;

void main() {
   exam X, Y;
   exam Z = {55, 35, "A-"};      Z.p1 = 55; Z.p2 = 35; Z.g = "A-"

   X = Z;                        X.p1 = Z.p1 = 55
                                 X.p2 = Z.p2 = 35
                                 X.g = Z.g = "A-"

   Y.p1 = X.p1;                  Y.p1 = X.p1 = 55
   Y.p2 = 20;                    Y.p2 = 20
   strcpy(Y.g, "C");             Y.g = "C"
   X.p1 = X.p1 - 10;             X.p1 = X.p1 – 10 = 55 – 10 = 45
   X.g[0] = 'B';                 X.g[0] = 'B' → X.g = "B-"

   printf("%d + %d = %d (%s)\n", X.p1, X.p2, X.p1 + X.p2, X.g);
   printf("%d + %d = %d (%s)\n", Y.p1, Y.p2, Y.p1 + Y.p2, Y.g);
   printf("%d + %d = %d (%s)\n", Z.p1, Z.p2, Z.p1 + Z.p2, Z.g);
}
```

OUTPUT:

```
45 + 35 = 80 (B-)
55 + 20 = 75 (C)
55 + 35 = 90 (A-)
```

2 (continued)
b. (14 points)

```
typedef struct {
   int x;
   int y;
} partB;

void main() {
```

When the array below is initialized, the 5 different structures within that array are initialized. You are given 5 pairs of values—the first is the "x" value for that element; the second is the "y" value. For example, arr[0].x = 0; arr[0].y = 4.

```
   partB arr[5] = { {0, 4}, {1, 2}, {4, 3}, {2, 0}, {3, 1} };
   int i;

   for (i = 0; i < 5; i++)
      arr[i].x = arr[arr[i].y].x;
```

Each iteration of the loop above accesses one array element and overwrites the x value stored in that element with the x value stored in another element. To determine which element holds the new x value, look at the y value of the current element:

arr[0].x = arr[arr[0].y].x = arr[4].x = 3

arr[1].x = arr[arr[1].y].x = arr[2].x = 4

arr[2].x = arr[arr[2].y].x = arr[3].x = 2

arr[3].x = arr[arr[3].y].x = arr[0].x = 3 (set during 1st iteration)

arr[4].x = arr[arr[4].y].x = arr[1].x = 4 (set during 2nd iteration)

```
   for (i = 0; i < 5; i++)
      printf("%d %d\n", arr[i].x, arr[i].y);
}
```

OUTPUT:

```
3 4
4 2
2 3
3 0
4 1
```

2 (continued)

c. (12 points) *Hint: I strongly recommend drawing a diagram to keep track of where each pointer points.*

```
typedef struct n {
   int v;
   struct n *p1;
   struct n *p2;
} node;

void printNode(node *p) {
   printf("%d\n", p->v);
}

void main() {
   node n1 = {1, NULL, NULL};
   node n2 = {2, NULL, NULL};
   node n3 = {3, NULL, NULL};
   node *p = &n3;

   n1.p1 = &n2;
   n1.p2 = &n3;
   n2.p1 = &n1;
   n2.p2 = &n3;
   n3.p1 = &n1;
   n3.p2 = &n3;

   printNode(n1.p2);

   printNode(n3.p2);

   printNode(p);

   printNode(n1.p1->p1);

   printNode(p->p1->p1);
}
```

Given a pointer to a node, this function prints the integer v stored within that node

Each node is given an initial value for the integer v, as well as pointers p1 and p2

n1.p2 = &n3
→ This line prints n3.v, which is 3

n3.p2 = &n3
→ This line prints n3.v, which is 3

p = &n3
→ This line prints n3.v, which is 3

n1.p1 = &n2
Therefore, n1.p1->p1 = n2.p1 = &n1
→ This line prints n1.v, which is 1

p = &n3
Therefore, p->p1 = n3.p1 = &n1
Therefore, p->p1->p1 = n1.p1 = &n2
→ This line prints n2.v, which is 2

OUTPUT:

```
3
3
3
1
2
```

3. (40 points, 20 per part) ***General I/O***

For each part of this problem, you are given a short function 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.**

a. `int readLines(char list[][100], int n, FILE *fp);`

This function takes the following arguments:

- `char list[][100]`: A 2D character array; each row of this array is therefore a string of at most 100 characters.
  - To access an entire row of a 2D array, specify only the first dimension—for example, `strcpy(list[0], "test")` copies `"test"` into the first row of the array.
- `int n`: The number of rows in the array.
- `FILE *fp`: A pointer to an open file from which you will read input. This file should remain open when the function ends.

Complete this function so it reads the file pointed to by `fp` one line at a time (including spaces or newlines), and stores each line in the array `list[][100]` as a null-terminated string. If the file contains more than `n` lines, just read the first `n` lines. Assume each line is at most 99 characters.

The function should return the number of lines that were read.

Students had to complete the boldface, underlined lines of code.

```
int readLines(char list[][100], int n, FILE *fp) {
   int i = 0;        // Loop index

   // SET UP LOOP THAT RUNS AS LONG AS END OF FILE IS NOT
   //   REACHED, AND LESS THAN n LINES HAVE BEEN READ
   while (!feof(fp) && (i < n)) {

      // READ EACH LINE FROM FILE AND STORE IN list, THEN
      // UPDATE VARIABLE TRACKING NUMBER OF LINES READ
      fgets(list[i], 100, fp);
      i++;
   }

   // RETURN NUMBER OF LINES READ
   return i;
}
```

3 (continued)

b. `void fillArray(int arr[], int n, char *fn, char *fm);`

This function takes the following inputs:

- `int arr[]`: An integer array to be filled.
- `int n`: The number of integers to be stored in `arr[]`.
- `char *fn`: The name of the file to be opened.
- `char *fm`: The mode to be used when opening the file—will be either `"rt"` or `"rb"`.

The function should open the appropriate file and then read `n` values from that file, using the appropriate method, and store them in `arr[]`. For example:

- `fillArray(x, 10, "in1.bin", "rb")` will open the binary file `in1.bin`, and then read 10 integers into the array `x` using unformatted I/O.
- `fillArray(y, 30, "in2.txt", "rt")` will open the text file `in2.txt`, and then read 30 integers into the array `y` using formatted I/O.

```
void fillArray(int arr[], int n, char *fn, char *fm) {
  FILE *fp;    // Pointer to be used when opening file
  int i;       // Loop index

  // OPEN FILE WITH NAME fn AND MODE fm
  fp = fopen(fn, fm);

  if (fp != NULL) {  // TEST TO ENSURE FILE IS OPEN

    // IF USING BINARY FILE, USE UNFORMATTED I/O TO FILL ARR[]
    if (fm[1] == 'b') {
      fread(arr, sizeof(int), n, fp);
    }

    // OTHERWISE, FILL ARRAY USING FORMATTED I/O
    else {
      for (i = 0; i < n; i++)
        fscanf(fp, "%d", &arr[i]);
    }

    // CLOSE FILE
    fclose(fp);
  }
}
```

3 (continued)

c. `double readDouble(FILE *fp);`

This function should read a double-precision value from the file referenced by `fp` (which you may assume points to a valid, open file). Note that:

- The whole part (`w`) and the fractional part (`f`) of the value are stored separately in this function. For example, if user enters 123.456, `w == 123` and `f = 0.456`.
  - Note: the algorithm for adding each digit to `w` (which is partially given) is not the same as the algorithm for adding each digit to `f`.
- The `isdigit(char c)` function returns a non-zero value if `c` represents a number.
- `'0'` has the ASCII value 48, so `'0' – 48 == 0`, `'1' – 48 == 1`, and so on.

```
double readDouble(FILE *fp) {
  char c;           // Input character
  double w = 0;     // Whole part of double
  double f = 0;     // Fractional part of double
  double div = 10;  // Divisor used with fractional part

  // WHOLE PART--LOOP WHILE CHARACTER READ IS A DIGIT

  while (isdigit(c = getc(fp))) {
    w = (w * 10) + (c - 48);      // ADD DIGIT TO W
  }
  // TEST IF CHARACTER THAT ENDS LOOP ABOVE IS A DECIMAL POINT
  //  IF SO, NUMBER MUST HAVE FRACTIONAL PART--READ IT INSIDE if

  if (c == '.') {

    // FRACTIONAL PART--LOOP WHILE CHARACTER IS A DIGIT

    while (isdigit(c = getc(fp))) {

      // ADD EACH DIGIT TO FRACTIONAL PART AND UPDATE DIVISOR
      f = f + ((c - 48) / div);
      div = div * 10;
    }
  }
  // PUT LAST CHARACTER BACK IN INPUT FILE, THEN RETURN
  //   FINAL VALUE OF NUMBER (COMBINATION OF W AND F)
  ungetc(c, fp);
  return w + f;
}
```