COP 3223 Fall 2010 Final Exam Form A Solutions

Answer Key

1) B	6) A	11) E	16) C	21) C
2) C	7) C	12) D	17) C	22) D
3) E	8) B	13) C	18) A	23) B
4) D	9) A	14) A	19) A	24) D
5) E	10) C	15) E	20) A	25) A

Solutions

1) (B) The first assignment statement changes a to 2*7 + 2 = 16, while b and c remain the same (7 and 2 respectively.) First output statement is:

```
a = 16, b = 7, c = 2
```

2) (C) The second assignment statement changes b to 2*2 + 16 = 20, while a and c remain the same (16 and 2 respectively.) First output statement is:

```
a = 16, b = 20, c = 2
```

3) (E) The third assignment statement changes c to 2*16 + 20 = 52, while a and b remain the same (16 and 20 respectively.) First output statement is:

```
a = 16, b = 20, c = 52
```

This doesn't match any answer choice to E is the answer.

4) (D) There are two separate if statements here. The first if statement triggers because b < 3*a is true and it's part of an or, so A prints. The second if statement must print either B or C since it's an if-else. The boolean condition is an and with the first item false (2*a is 6 but b is 7), so C prints. Thus the output is:

```
AC
```

5) (E) This question is testing the matching-else problem. The else matches the inner if (second one), so when we find out that a > b is false (since a is 4 and b is 6), **we skip the whole body of that if statement, which is the whole if-else statement.** This means that we complete the outer if printing nothing. The following statement prints C, so the correct output is

```
C
```

This doesn't match any answer choice to E is the answer.

6) (A) There are four if statements. The first three have boolean conditions that are true, so DCB prints. The last if statement's boolean condition is false, so F prints. The final output is

DCBF

7) (C) The values added into sum over the course of the loop are 3, 6 and 9 (since we start at 3 and increment by 3 each time), the sum of these values is 18. Also, since 9 < 10, i will increment by 3 again to equal 12. Thus the output of this code segment is:

i = 12, sum = 18

8) (B) This question is testing the empty statement (extra semicolon on the line with the for). The only statement **INSIDE OF** the for loop is the empty statement. This means that i does increment from 3 to 6 to 9 to 12 and then the loop ends without ever affecting sum. The statement `sum += i;` executes once right after the loop completes, changing the value of sum to 12. Thus the desired output is

i = 12, sum = 12

9) (A) We actually just have to trace the whole thing to answer this question, and then we can answer 10, 11 and 12 for free. The array starts like this:

6 2 17 5

When we run the line of code array[i+1] = array[i+1] + array[i]; with i = 0, we get

6 8 17 5

Then, when we run the line of code array[i] = array[i+1] – array[i]; with i=0 we get

2 8 17 5

We must repeat these two statements for 2 more loop iterations with i = 1 and i = 2. In essense, an iterations adds the two values in question together and puts them in the next array index and then "moves" the old value of the next array index to the previous one. So after completing the code for i = 1, we have

2 17 25 5

Finally, after completing the iteration for i = 3 we have

2 17 5 30

Thus, the first line of output prints 2, choice A.

10) (C) From above, the second line of output prints 17.

11) (E) From above, the third line of output prints 5. This does not correspond to any of the given choices.

12) (D) From above, the fourth line of output prints 30.

13) (C) Let's draw this out. In main we pass 3 and 7, respectively to f. Here is the picture in f:

f
a 3
b 7

Now, after the first assignment statement we have:

f
a 3
b 7
temp 11

After the second assignment statement we have:

f
a 8
b 7
temp 11

After the third assignment statement we have:

f
a 8
b 1
temp 11

At this point we print a = 8, b = 1.

14) (A) Pick up where the trace above leaves off. The function will return 8*(1+2)%11 = 2 to main and in main this is assigned to variable b and main's a remains unchanged. The picture in main is:

main
a 7
b 2, thie corresponds to choice A

15) (E) In main, we pass the values a+b+2 = 11 and a-b+1 = 6 to function f, so the beginning of function f looks like:

f
a 11
b 6

Now, after the first assignment statement we have:

f
a 11
b 6
temp 49

After the second assignment statement we have:

f
a 38
b 6
temp 49

After the third assignment statement we have:

f
a 38
b 32
temp 49

At this point we print a = 38, b = 32. There is no answer choice that corresponds to this so the correct choice is E.

16) (C) Take over from above. The function will now return 38*(32 + 2)%49. (So I apologize this is gross to do by hand. It's 38*34, which I happen to know is $36^2 – 2^2 = 1296 – 4 = 1292$, but most people would probably just multiply it out by hand. Then divide 49 to get a quotient of 26 and a remainder of 18. The value returned to main is 18. Keep in mind in main that previously, a=7 and b=2, and that we will reset a to this value of 18. Thus, in main, we'll have

a = 18, b = 2

17) (C) You really have to draw these out. Here is a basic sketch of what memory looks like right after the first function call is made:

<u>main</u> <u>f</u>
a 7 <----------b
b 3 <----------a

So, a in f is pointing to the box with 3 and b in f is pointing to the box with 7. First line of code does this:

<u>main</u> <u>f</u>
a 7 <----------b
b 3 <----------a
temp 11

Second line of code does this:

<u>main</u> <u>f</u>
a 7 <----------b
b 8 <----------a
temp 11

Third line of code does this:
<u>main</u> <u>f</u>
a 1 <----------b
b 8 <----------a
temp 11

At this point we print *a and *b which are equal to 8 and 1, respectively. This is choice C.

18) (A) Take the picture above, and continue, the function will now return

8*(1 + 2)%11, which is 24%11, which is 2. Thus, in main, b gets set to 2. Here is the picture:

<u>main</u>
a 1
b 2, this corresponds to choice A

19) (A) Now, let's draw the picture after the function call to f begins:

<u>main</u> <u>f</u>
a 1 <----------a
b 2 <----------b

So, a in f is pointing to the box with 3 and b in f is pointing to the box with 7. First line of code does this:

<u>main</u> <u>f</u>
a 1 <----------a
b 2 <----------b
temp -1

Second line of code does this:

<u>main</u> <u>f</u>
a -2 <----------a
b 2 <----------b
temp -1

Third line of code does this:
<u>main</u> <u>f</u>
a -2 <----------a
b -4 <----------b
temp -1

At this point we print *a and *b which are equal to -2 and -4, respectively. This is choice A.

20) (A) Take the picture above, and continue, the function will now return

(-2)*(-4 + 2)%11, which is 4%(-1), which is 0. Thus, in main, a gets set to 0. (Note: Anything mod 1 or -1 is 0.) Here is the picture:

<u>main</u>
a 0
b -4, this corresponds to choice A

21) (C) Choice A has an illegal identifier name, 2num, choice B doesn't define a struct but looks like it's an incorrect function signature, choice C is fine, and choice D is trying to define struct f in terms of itself, which is not allowed.

22) (D) Since p is a pointer, to access its components, use the array. To actually change the component the ++ shorthand will do it. Choices A and B are just expressions, but don't specify an action to change the value of a variable like choices C and D. C is incorrect since p is a pointer.

23) (B) Both malloc and calloc would work but malloc is the only funciton explicitly listed.

24) (D) NULL is a reserved constant in C.

25) (A) For fun =)

Fall 2010 COP 3223 Section 3
Final Exam Free Response Solutions

1) (10 pts) Arup has decided that he's going to create his own newspaper to rival the in town Sentinel. You are working in his subscriptions department and he's asked you to write a program that calculates the cost of a subscription. The cost of a Sunday paper is $2.00 while the paper costs $1.25 on all other days. All subscriptions start on Monday. Your program should prompt the user with the number of days they want for their subscription and output the total cost of their subscription. (For example, a 7 day subscription costs $9.50 and an 8 day subscription costs $10.75.) Fill in the program below to complete this task.

```
#define REG_COST 1.25
#define SUN_COST 2.00
#define REGDAYS_WEEK 6

int main() {

    int sub_length;
    double total_cost = 0;
    printf("How long do you want to subscribe?\n");
    scanf("%d", &sub_length);



    // Calculate the number of full weeks and the cost of one week.
    int fullweeks = sub_length/7;
    double week_cost = REGDAYS_WEEK*REG_COST + SUN_COST; // 3 pts

    // Cost of the last (incomplete) week, could be 0.
    double last_week = (sub_length%7)*REG_COST;  // 4 pts

    // Add these two components to get the answer.
    total_cost = fullweeks*week_cost + last_week; // 3 pts









    printf("Your subscription will cost $%.2lf.\n", total_cost);
    return 0;
}
```

2) (10 pts) Write a segment of code that reads in a positive integer from the user greater than 1 and determines whether or not that value is prime. Part of the code is provided:

```
int n, prime = 1;
printf("Enter a positive integer greater than 1.\n");
scanf("%d", &n);

int i;                          //  1pt

// Go through each divisor.
for (i=2; i<n; i++) {               // 3 pts

    // Not prime...
    if (n%i == 0) {                 // 3 pts
        prime = 0;                  // 3 pts
        break;
    }
}

if (prime)
    printf("%d is prime.\n", n);
else
    printf("%d is NOT prime.\n", n);
```

3) (10 pts) Write a function that takes in an array of stock values for a set of consecutive days and determines the greatest change between two successive days. The two parameters to the function will be the array of values and the length of the array. (For example, if the array contained 9, 8.75, 9.25, 8, and 8.50, the function should return 1.25, because the change between day 2 and day 3 was | 8 – 9.25 | = 1.25.) The array length is guaranteed to be greater than 1.

```
double maxChange(double values[], int length) {

    int i;
    double max = 0;                             // 1 pt

    // Go through each successive pair.
    for (i=0; i<length-1; i++) {                // 3 pts

        // Calculate difference.
        double temp = values[i] - values[i+1]; // 2 pts
        if (temp < 0) temp = -temp;             // 1 pt

        // Update max difference if necessary.
        if (temp > max)                         // 1 pt
            max = temp;                         // 1 pt
    }
    return max;                                 // 1 pt

}
```

4) (10 pts) Write a function that takes in a pointer to the front of a linked list and **exchanges the values** stored in the first and last nodes of the list. If the list has fewer than 2 nodes, nothing should be done.

```
struct ll {
    int data;
    struct ll* next;
};

void swapFirstLast(struct ll* list) {


    // Do nothing in these cases.
    if (list == NULL || list->next == NULL)    // 1 pt
        return;

    // Save the front.
    struct ll* front = list;                   // 1 pt

    // Go to the last node.
    while (list->next != NULL)                 // 3 pts
        list = list->next;                     // 2 pts

    // Swap the two values.
    int temp = list->data;                     // 1 pt
    list->data = front->data;                  // 1 pt
    front->data = temp;                        // 1 pt



}
```

5) (10 pts) A pack of dice is characterized by two values: the number of dice, and the number of sides on each of the dice. In the game of Monopoly, for example, we use 2 dice with 6 sides each. (We assume the sides are labeled 1 through n, where n is the number of sides on each dice.) In this question you will write a couple functions related to the pack of dice, which will be stored in a struct shown below. Assume that the random number generator has already been seeded and that calls to rand() will produce random numbers in between 0 and 32767. Write the two functions according to the specifications given.

```
struct packofdice {
    int numdice;
    int numsides;
};

// Returns the sum of the dice on a simulated roll of all
// of the dice pointed to by thispack.
int roll(const struct packofdice* thispack) {

    int i;

    // Add up each random roll.
    int sum = 0;                                // 1 pt
    for (i=0; i<thispack->numdice; i++)         // 1 pt
        sum = sum + rand()%thispack->numsides + 1; // 4 pts

    // Return the value.
    return sum;          // 1 pt

}

// Returns the maximum possible value that could arise from
// the sum of all the dice in the pack pointed to by thispack.
int maxval(const struct packofdice* thispack) {

    return thispack->numdice*thispack->numsides; // 3 pts

}

// Note: Just one point off for using . instead of ->
```