CMSC 212 Midterm #2 (Fall 2005)
ANSWERS AND GRADING KEY

Discussion Section Time (circle one): 12:00 1:00 2:00 3:00 4:00 5:00
Elena    Sorelle    Morgan

(1) This exam is closed book, closed notes, and closed neighbor. No calculators are permitted. Violation of any of these rules will be considered academic dishonesty.

(2) You have 75 minutes to complete this exam. If you finish early, you may turn in your exam at the front of the room and leave. However if you finish during the last ten minutes of the exam please remain seated until the end of the exam so you don't disturb others. Failure to follow this direction will result in points being deducted from your exam.

(3) Write all answers on the exam. If you need additional paper, we will provide it. Make sure your name is on any additional sheets.

(4) Partial credit will be given for most questions assuming we can figure out what you were doing.

(5) Please write neatly. Print your answers, if that will make your handwriting easier to read. If you write something, and wish to cross it out, simply put an X through it. Please clearly indicate if your answer continues onto another page.

(6) The CMSC 212 Final Exam is scheduled for Thursday, December 15(4:00-6:00pm) Location: H.J. Patterson 0226.

<table>
<thead>
<tr>
<th>Page</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1.) [25 points] Define and explain the following as requested:

a) Assuming that you want to dynamically link a library and use the functions described in that library into the function you are writing, describe the code that would need to appear within the current function so that the functions of that library could be called. Make sure you include all steps that would be used if only this function needs to access those library functions. (Note: describe the steps since you don’t have enough specifics to write the code)

[5 points possible]
1 point for mention of the dlopen
1 point for mention of testing the return value of dlopen

1 point for each of the following (up to 2 points)
for the mention of the dlsym
function pointers to the functions defined in the library
for mentioning checking the return value of the dlsym

1 point for the dlclose

b) Write all of the pieces of code that would be necessary so that the one line z = callit(x,y) could be used to call the add function if the numbers are both positive or the multiply function if either (or both) of the integers are non-positive.

```c
int add(int a, int b)
{ return a+b; }
```

```c
int multiply(int a, int b)
{ return a*b; }
```

/* make sure you write both type definitions and code as needed*/

[5 points possible]

typedef int (*fpntr)(int, int);  [not necessary if they do it inside]

```c
int funct(int x, int y){
int z;
/* write the code that would go here so that z gets the correct value as described above when the one line shown below the blank space is executed */

fpntr callit;  [2 points for the declaration of the type of the callit function]

if (x > 0 && y > 0)
callit = add;
else
callit = multiply;

z = callit(x,y);
```
c) Given that the following line is used to define the library flags in your make file:

\[ \text{LIBFLAGS} = \text{-nostdlib -shared -fPIC -Wl,-soname,$@.1} \]

Describe the purpose of the $@ that appears near the end of the line. Tell when this would be useful.

**[3 points possible]**

The $@ will be replaced by the name of the target.

It is useful if there are several libraries that need to be compiled within the same makefile; it allows the same LIBFLAGS to be used for all of them because it makes the macro independent of the name of the target even though that name needs to appear within one of the switches.

---

Comment [JP3]: they must state that the $@ gets replaced (this is worth 3 points) they must say that it makes it possible to use the same LIBFLAGS (this is worth 2 points)

---

d) Answer the following questions about using lcov.

1. What needs to appear in the Makefile so that lcov can be accessed?

**[3 points possible]**

All of the files which are supposed to be analyzed by lcov should be compiled with flags that say they are supposed to have added "arcs" or connections and added "coverage".

or

- \text{-fprofile-arcs -fprofile-coverage}

or

The file profile needs to be set appropriately when compiling all files that need to be viewed by lcov.

---

Comment [JP4]: any method of wording an explanation --they don't need to specify the exact options -- they should use either the word "profile" or the "arcs/coverage" for full credit

---

2. What steps do you need to take at the shell prompt in order to use lcov?

**[3 points possible]**

[1 point for each of the following to a max of 3 points]

1. compile so that lcov options are included
2. tap lcov
3. run the application (or create the extra data file used by lcov)
4. type \text{lcov -c -d . -o cov.out}
   \hspace{1cm} (or type command telling it where to put the lcov output)
5. \text{genhtml cov.out}
6. view the html file created

---

Comment [JP5]: any 3 of these 6 (or their meaning) for full credit

- they don't need to have any certain sequence or order
- 1 point per to max of three – if they have 4 but 1 is wrong – cross it out but they still get full credit
e) Compare and contrast Whitebox testing and Blackbox testing. Compare and contrast means to specify both similarities and differences.

[4 points possible]

compare[1 point]: both of these testing methods try to test all aspects of the program

contrast[3 points]: In whitebox testing, you can see the implementation and base your test cases on what code you know is there and needs to be tested, but in blackbox testing, you only know the definition of the API and you have to test the functionality of that API without knowing what the code implementing it actually looks like.

or

The whitebox testing can consider issues such as path coverage/branch coverage because the tester can see the code and consider all of the paths available in that code, while in blackbox testing the tester has to try to test all kinds of input for conditions such as covering the extremes and exceptional cases because the tester can’t see the actual code.

f) Write, in hex, the two’s complement representation of the 32-bit value “-9”.

[2 points possible] FFFFFFF7
2.) [23 points] Write the complete main that would exit setting status to -1 and with a printed error message if there are not at least two arguments (in addition to the application which is in argv[0]). If there are at least two arguments, it should count how many would be positive integers (containing only numeric digits) and how many must only be considered as strings. It should then print out both counts. It then also prints the sum of the positive integers.

You may use the functions:

```c
int isdigit(char);             int atoi(const char *);
```

For example:
```
$ ./countthem a b 12 4 c
Numeric: 2    non-Numeric: 3         Sum: 16
$ ./countthem
There are not at least two arguments
$ ./countthem 12 23 13 1324a
Numeric: 3    non-Numeric: 1          Sum: 48
```

[23 points possible]

```c
int main(int argc, char * argv[]){
    int total = 0, curr=0;
    int picount = 0, strcount=0;
    if (argc < 3){
        printf("There are not at least two arguments\n");
        return -1;
    }
    for (curr = 1; curr < argc; curr++){
        if (isdigit(argv[curr])){
            picount++;
            total += atoi(argv[curr]);
        } else{
            strcount++;
        }
    }
    printf("Numeric: %d  non-Numeric: %d     Sum: %d\n", picount, strcount, total);
    return 0;
}
```

Comment [JP8]: This is broken among the concepts noted below

Comment [JP9]: the test to make sure there are at least 2 other values after name of the executable is worth 4 of the 23 points

Comment [JP10]: Correctly tracking the argument vector is worth 7 of the points of the 23
  ■ If they included the executable in the list -2
  ■ If they did not correctly convert and sum the value into the total up to -3
  ■ If they did not correctly increment the string counter it is up to -2
  ■ If they did not correctly increment the positive integer counter it is up to -2

Comment [JP11]: printing the result is worth 3 of the 23 points (one for each of the three values to be taken care of

Comment [JP12]: They need to have some way to determine if that argument is indeed all numeric
  This portion is worth 9 of the 23 points
  ■ It does not have to be done in a helper function
  ■ If they did write a helper function they don’t need to have a prototype or any certain order between the two functions
  ■ If they correctly allowed negative integers to be summed in -2
  ■ If they accepted anything with a – (anywhere in the number accepting that as numeric) -4
  ■ If they did not take care of the zero case (it should not be counted as a positive integer) -1
  ■ If they used isdigit incorrectly (to assume it did the whole string -5
3.) [15 points] Answer the following questions about multiple processes from within a single C source file.

   a) When the fork function is called, what value does the child process have as the return value of that function?

      [3 points possible]
      0

   b) When the fork function is called, what values could the parent process have as the return value of that function?

      [6 points possible]
      3 for saying the process ID of the child or saying a positive value
      3 for saying the value -1 in the case that the forking of a child was not possible

   c) If the value of a variable named parentvar is set to 5 before the fork command is called and the following line appears in the child portion of the code, circle the letter of the statement that best describes what will happen.

      printf("%d\n", parentvar);

      a) The child will declare it’s own variable named parentvar with a random initial value.
      b) The child will declare it’s own variable named parentvar with a 0 as the initial value.

   d) If the child changes the value of parentvar to 7, after the fork but before the child finishes, select the line that best describes what will happen if the following line appears after the wait in the parent process (so this line is running after the child has completed). Assume the parent did nothing to modify the value of this variable since it was originally set before the fork in the previous question.

      printf("%d\n", parentvar);

      a) The parent will print the value 7 since they share the same variable space.

   b) The parent will print the value 5 since the child changed its own copy of the variable. [3 points]

      c) The parent will print a random value since the space got corrupted -by the child’s attempt to change the value.
      d) There will be a compilation error because they can not both have access to the same variable.
      e) There will be a runtime error as the child accesses the parent’s space so this line won’t be executed at all.
      f) None of the above.

Comment [JP13]: They must say 0 here – basically all or nothing

Comment [JP14]: 3 points for each
   -- what happens when the fork is successful – at least they must say a positive integer (they don’t have to say that it is the PID of the child)
   -- what happens when the fork is not successful

Comment [JP15]: all or nothing

Comment [JP16]: all or nothing
4.) [21 points] Project 4 as defined used the following types. Write function called “distorttree” which will make a new tree that has all of the arithmetic operators changed to their opposite (plus becomes minus, minus becomes plus, mult becomes div and div becomes mult). The function returns the distorted tree through the return value of the function. You may write helper functions as needed. Names have been reduced to save some writing time.

typedef enum { operatorNode, variableNode, constantNode } nodeType;
typedef enum { plusOp, minusOp, multOp, divOp, equalOp } operatorType;

typedef struct _node {
    nodeType type;
    struct _node *left;
    struct _node *right;
    int value;
    operatorType operator;
    char *name;
} ASTnode;

ASTnode *createOperatorNode(operatorType op, ASTnode *left, ASTnode *right); /* assume these */
ASTnode *createConstantNode(int constant); /* functions have been written*/
ASTnode *createVariableNode(char *name); /* and tested already for you */
ASTnode *copyTree(ASTnode *node); /* you do not need to write them*/

---------------------------------------------------------------------------

ASTnode *distortTree(ASTnode *node); /*This is the prototype for the function you must implement*/

[21 points possible]

ASTnode * distortTree(ASTnode *root){
    ASTnode* newtree = copyTree(root);
    modtree(newtree);
    return newtree;
}

don'ttree(ASTnode *root){
    if (node == NULL) return;
    if (root -> type == operatorNode){
        if (root -> operator == plusOp) root->operator = minusOp;
        else if (root -> operator == minusOp) root->operator = plusOp;
        else if (root -> operator == multOp) root->operator = divOp;
        else if (root -> operator == divOp) root ->operator = multOp;
    }
    modtree(root->left);
    modtree(root->right);
}

------------- or -------------

Comment [JP17]: two basic ways – more people did the harder way
- they could use the copyTree function and then distort the copied tree
- they could make a new form of copy that id all of the copying and distorting

Comment [JP18]: 9 points for correctly copying the structure of the tree

Comment [JP19]: 9 points for correctly distorting the tree

Comment [JP20]: 3 points for returning the pointer to the root of their new tree as the return value of the function
ASTnode * distortTree(ASTnode * root){
    ASTnode* temp = NULL;
    if (root != NULL) {
        temp = (ASTnode *) calloc(sizeof(ASTnode),1);
        if (!temp) exit(-1);
        copycontent(temp, root);
        temp->left = distortTree(root->left);
        temp->right = distortTree(root->right);
    }
    return temp;
}

void copycontent(ASTnode *dest, ASTnode *source){
    operatorType swaparr[] = {minusOp, plusOp, divOp, multOp, equalOp};
    dest -> type = source -> type;
    dest -> value = source -> value;
    if (source ->name) dest -> name = strdup(source ->name);
    dest -> operator = swaparr[source ->operator];
    return;
}
5.) [16 Points] Recall the heap manager discussed in class, which allocates space using 8-byte chunks. Assume a total heap space of 64 bytes, and that the interface to the heap manager consists of `malloc()` and `free()` with the usual semantics. The shown heap does not contain the header discussed in class to manage the blocks – just the blocks allocated directly from the request are shaded (assume the management information is stored elsewhere). Assuming each square represents an 8-byte chunk, we can pictorially represent the state of the heap at some point as follows:

This shows the heap being broken into three blocks. The center of those three blocks is allocated and the two on the ends are free.

a) [5 points] Assuming a **first-fit** allocation strategy, that the memory is completely unallocated before starting, and the sequence of malloc() and free() operations below, annotate the following diagram by shading allocated regions to indicate the state of the heap immediately after each line of the program.

```plaintext
p1 = malloc(15);
p2 = malloc(20);
p3 = malloc(16);
free(p2);
p4 = malloc(4);
free(p3);
```

Comment [JP21]: 5 points for the first fit trace
- 2 for showing partial 8-byte chunks as allocated but the end of the 8-byte chunk as non allocated
- 3 for showing partial 8 byte chunks as allocated and starting the next allocation inside of that partial 8-byte chunk
- 1 for any allocation/deallocation that did not all or remove the right squares – but be careful of double jeopardy – once points are removed on one line for an error, as it continues down they should not lose points for maintaining the space allocated/deallocated by that error
(Pete – how do you want to handle if an error is mysteriously corrected – this can either be viewed as another error, or it can be accepted assuming we think they realize they did something wrong)

b) [5 points] Repeat the above assuming **best-fit**.

```plaintext
p1 = malloc(15);
p2 = malloc(20);
p3 = malloc(16);
free(p2);
p4 = malloc(4);
free(p3);
```

Comment [JP22]: same as for the first fit trace
-- yes they lose the 2 or 3 points again if they did it both places even though they are making the same conceptual error

Comment [JP23]: 3 points for the definition of internal fragmentation
it must include something that indicates “granularity of the chunks allocated” or “size of the chunk actually allocated as compared to the size of the space requested”

Comment [JP24]: 3 points for the definition of external fragmentation
it must include that it refers to spaces left available after a “free” takes place between two allocated portions in some way

c) Define internal and external fragmentation.

[3 points for definition of internal fragmentation] due to extra space allocated beyond what was requested in order to keep the allocated blocks at some specific granularity - example in the first line 2 whole sections were allocated (16 bytes) when only 15 were requested.

[3 points for definition of external fragmentation] due to the pattern of allocation and deallocation - when there are several small non-contiguous blocks of unallocated space