

CMSC 212 FINAL EXAM (Spring 2005)

Name _____

Signature _____

Discussion Section Time (circle one):

12:00 1:00 2:00 3:00

Asad

Konstantin

- (1) This exam is closed book, closed notes, and closed neighbor. No calculators or other references are permitted. Violation of any of these rules will be considered academic dishonesty.
- (2) You have 120 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.

| Question | Possible | Score |
|----------|------------|-------|
| 1 | 20 | |
| 2 | 11 | |
| 3 | 15 | |
| 4 | 12 | |
| 5 | 10 | |
| 6 | 12 | |
| 7 | 20 | |
| Total | 100 | |

1.) [20 points]

a) Explain the purpose of C's preprocessor and then give two different preprocessor directives (with explanation of their purpose).

b) What is the purpose of each of these compiler switches? Your answer must be correct regardless of what other switches appear on the same compilation command line.

- 1. **-o** _____
- 2. **-O** _____
- 3. **-c** _____
- 4. **-g** _____

c) According to the code segment in the first column - write the type of each of the given expressions in the third column. (i.e. Integer, Integer pointer, address of an integer pointer, etc) If you can not tell from the code given, you must write the word "unknown". You do not need to worry about what value (or even if there is space allocated) - just what type it would be.

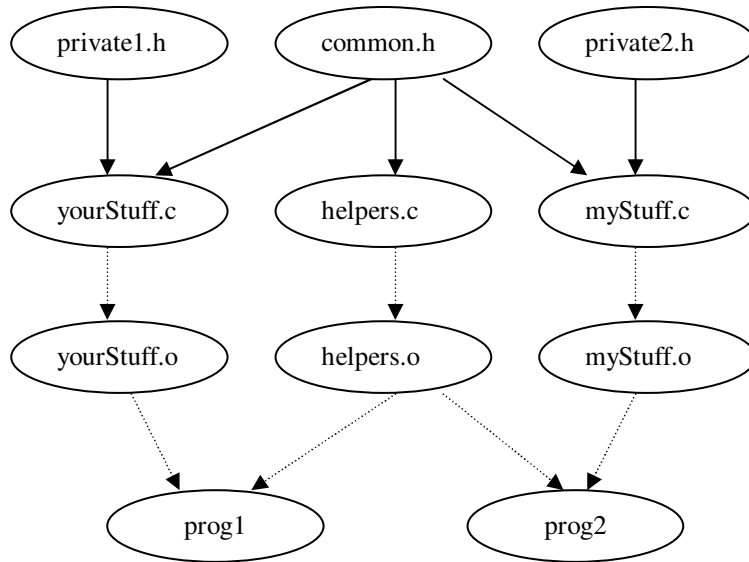
| | | |
|---------------------------------|-------|--|
| <pre>int a, b; int* t, q;</pre> | T | |
| | *(&a) | |
| | &b | |
| | q | |
| | *t | |

d) Given one example of a situation where optimizing wall time (vs. process time) makes more sense and one where optimizing process time (vs. wall time) makes sense.

- 3.) [15 points] Write the assembly code for the following function (following the stack conventions of the machine as discussed in class. You may pass parameters in the registers starting at register #2):

```
void foo(int x, int y)
{
    if (x > 0) {
        foo(x-1, y);
    } else if (y > 0) {
        printf("%d\n", y);
    }
}
```

- 4.) [12 points] Write a makefile that builds the programs with the following dependencies. A solid line indicates a file included by another file and a dash line indicates a file that should be built from one or more files. You must write the makefile in such a way that typing the command `make` at the command line will create both executable files based on the correct versions of the files in the current directory.



- 5.) [10 Points] Implement `strstr` without using any array notation or any string library functions. (searches for the second parameter within the first and returns a pointer to the beginning of that substring as the return value of the function). If either of the pointers passed as the parameters have the value `NULL`, the function should return the value `NULL`. Other than the case where the pointer itself is `NULL`, you may assume that the strings passed are valid C strings with appropriate termination. If the sub-string is not present in the full-string, the function should also return the value `NULL`.

```
char * strstr(const char * full, const char* sub);
```


7.) [20 points] Write the following routines:

```
void *myMalloc(int size);  
void *myCalloc(int num, int size);  
void myFree(void *ptr);
```

These routines must mimic the C library routines malloc, calloc and free respectively. You are given a function void *add10Meg() that grows the memory of your process by 10 megabytes at a time, and returns a pointer to the newly added space. The only C function you may call (that you don't write) is add10Meg(). The size in both the myMalloc and the myCalloc indicate the number of bytes in decimal. The num in the myCalloc parameter list indicates the quantity of the items of the size indicated.

You may use the following structure to implement it. Also, recall from class that it is convenient to make the amount of space allocated always a multiple of the size of your boundary structure.

```
typedef struct _boundary{  
    int size;  
    int free; /*indicates 1 for space that is currently free or 0 for space that is currently allocated*/  
    struct _boundary *next;  
} boundary;
```

```
boundary *list;
```

```
void *myMalloc(int size);
```



```
void *myCalloc(int num, int size);
```

```
void myFree(void *ptr);
```

```
/* myFree does not need to coalesce adjacent empty blocks and if a complete 10 meg unit  
becomes unneeded, myFree does not need to worry about returning it to the operating system */
```

This page intentionally blank - you may continue an answer here, but it must be clearly marked both here and where the answer is continued from.

Extended Project #1 Computer Instructions

This page should be removed from the exam to have it easily readable as you answer the questions pertaining to this topic. You do not need to reattach this page, and you should not submit it with your exam paper. Do not write any answers on this page or on the back of this page. This page must be kept on the top of your desk with your exam paper at all times - it can not be put into the chair next to you.

| | |
|--|--|
| Load <R ₁ > <R ₂ > <mem> | $R_1 \leftarrow \text{memory}[R_2 + \text{mem}]$ |
| Load <R ₁ > #<num> | $R_1 \leftarrow \text{num}$ |
| Store <R ₁ > <R ₂ > <mem> | $\text{Memory}[R_2 + \text{mem}] \leftarrow R_1$ |
| Move <R ₁ > <R ₂ > | $R_1 \leftarrow R_2$ |
| Add <R ₁ > <R ₂ > <R ₃ > | $R_3 \leftarrow R_1 + R_2$ |
| Halt | |
| Negate <R ₁ > | $R_1 \leftarrow -R_1$ |
| Bnn <R ₁ > <mem> | If ($r_1 \geq 0$) $PC \leftarrow \text{mem}$ |
| Branch <R ₁ > <R ₂ > <memory> | $R_1 \leftarrow PC; PC \leftarrow R_2 + \text{memory}$ |
| Input <R ₁ > | |
| Output <R ₁ > | |