Name _____ UW ID# _____

There are 9 questions worth a total of 120 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed laptops, closed twitter, closed telepathy, closed Facebook, etc.

If you have a question, please raise your hand and stay seated.

Please wait to turn the page until everyone is told to begin.

Score _____ / 120

1. _____ / 13   6. _____ / 12

2. _____ / 16   7. _____ / 14

3. _____ / 10   8. _____ / 9

4. _____ / 20   9. _____ / 6

5. _____ / 20

Note: Please write your answers only on the specified pages. Pages with only questions and explanations will not be scanned for grading, and you should feel free to remove them from the exam.

Question 1. (13 points) Preprocessor. Consider the following program:

```
#include <stdio.h>
#define SQUARE(x) x * x
#define DOUBLE(x) 2 * x
int main() {
  int a = 2;
  int b = 3;
  int x = DOUBLE(a);
  int y = SQUARE(a+b);
  int z = DOUBLE(1+a+b);
  printf("x = %d, y = %d, z = %d\n", x, y, z);
  return 0;
}
```

(a) (10 points) Show the output produced by the preprocessor (`cpp -P`) when it reads and processes this C program, which is the first step that happens before the compiler can translate the code to generate a `.o` file. Ignore the `#include <stdio.h>` line – it will insert the declarations from `stdio.h` and do nothing further. Otherwise, your answer should show all of the output from the preprocessor. There are no preprocessor errors in this program, and the resulting program compiles and executes without errors.

(b) (3 points) Fill in the blanks to show the numbers printed when this program is executed:

x = _____, y = _____, z = _____

Question 2. (16 points) Making things. Suppose we have a project with multiple source files that have the `#include` dependencies shown below.

```
* * * * * * * * * * * * * *          * * * * * * * * * * * * * *
* foo.h         *                    * bar.h         *
* * * * * * * * * * * * * *          * * * * * * * * * * * * * *
#ifndef _FOO_H_                      #ifndef _BAR_H_
#define _FOO_H_                      #define _BAR_H_
...
#endif                               #include "foo.h"
                                     ...
                                     #endif

* * * * * * * * * * * * * *
* foo.c         *
* * * * * * * * * * * * * *          * * * * * * * * * * * * * *
#include "foo.h"                     * bar.c         *
...                                  * * * * * * * * * * * * * *
                                     #include "bar.h"
                                     ...

* * * * * * * * * * * * * *
* main.c        *
* * * * * * * * * * * * * *
#include "foo.h"
#include "bar.h"

int main() {
  ...
}
```

If we're being lazy, we could create an executable program from these files by executing the command `gcc -Wall -g -std=c11 -o main *.c`.

On the next page, create a `Makefile` that builds an executable program named `main` from these files as done by the `gcc` command above, but only recompiles and relinks the minimum number of files needed after any changes are made to the source files. Your answer should be done in two steps:

- First, construct the dependency graph that shows the dependencies between the source files, the compiled `.o` files, and the final executable file `main` that is created by linking the `.o` files.
- After drawing the dependency graph, write the final `Makefile`.

Write your answers on the next page. You should remove this page from the exam. **Do not write on this page.** It will not be scanned for grading.

Question 2. (cont.) (a) (8 points) Draw the dependency graph (diagram) showing dependencies between the files on the previous page, the `.o` files created by compiling the `.c` files, and the final executable program `main`.

(b) (8 points) Give the contents of a `Makefile` that will build the program as described by the dependency diagram above. The default target that is built if we just type `make` with no arguments should be the executable program `main`.

Question 3. (10 points) git. Two of our programmers, Bart and Lisa, are using git to manage a project. Both of them have been working on various parts of the code. Bart has made some changes, tested them, and now wants to commit and push his changes to the shared repository. A transcript of the terminal session follows. Answer questions about it below.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   main.c
no changes added to commit (use "git add" and/or "git commit -a")
$ git add main.c
$ git commit -m "change output message"
[master 2413774] change output message
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git push
To gitlab.cs.washington.edu:cse374-17au-students/cse374-17au-xa.git
 ! [rejected]       master -> master (fetch first)
error: failed to push some refs to 'git@gitlab.cs.washington.edu:cse374-17au-
students/cse374-17au-xa.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
$
```

(a) (2 points) Something went wrong when Bart tried to do the `git push`. What is the explanation of the problem? (Don't just repeat the message printed above – explain what we can actually conclude from the message about the underlying cause of the failure or what the likely conflict is, etc., but be brief.)

Question 3. (cont.) Not quite sure what to do next, Bart tries entering the `git pull` command suggested in the original error message. Here is the output from that:

$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.cs.washington.edu:cse374-17au-students/cse374-17au-xa
   7809b75..226da1e master   -> origin/master
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
$

(b) (8 points) Well, that didn't quite work, apparently. Explain specifically what the problem is now and then give a complete list of the specific steps that Bart needs to do to clear up the problem(s) and successfully push changes to the git repository.

Question 4. (20 points) Another trieing question. In HW5 we used a trie to store the words in a dictionary based on the digit sequences that encoded the words. There are several possible ways to represent trie nodes, but for this problem, assume that a node is defined as follows:

```
struct tnode {              // one node in the trie:
  char * word;              //  C-string if this node has a
                            //  word attached, otherwise NULL
  struct tnode *next[10];   //  Subtrees.  next[2] to next[9]
                            //  are subtrees for digits 2-9;
                            //  next[0] is the synonym ('#') link.
};                          //  For 0<=i<10, next[i]==NULL if
                            //  next[i] is an empty subtree.
```

For this problem, write a function that produces an exact copy of a trie. In other words, `clone(r)` should return a pointer to a new trie that is an exact copy of the original one, including copies of all of the nodes and strings in the original.

You should assume that all necessary header files have already been `#include`d and you do not need to add any `#include`s. You may assume that `malloc` always succeeds and you do not need to check the result for `NULL`.

Hints: recursion really, *really*, **really** is your friend. The clone of an empty trie is empty.

A bit of (maybe) useful reference information about strings and memory:

Some string library functions:
  - `char* strncpy`(*dest*, *src*, *n*), copies exactly *n* characters from *src* to *dst*, adding '\0's at end if fewer than *n* characters in *src* so that *n* chars. are copied.
  - `char* strcpy`(*dest*, *src*)
  - `char* strncat`(*dest*, *src, n*), append up to *n* characters from *src* to the end of *dest*, put '\0' at end, either copy from *src* or added if no '\0' in copied part of *src*.
  - `char* strcat`(*dest*, *src*)
  - `int    strncmp`(*string1*, *string2*, *n*), <0, =0, >0 if compare <, =, >
  - `int    strcmp`(*string1*, *string2*)
  - `char* strstr`(*string, search_string*)
  - `int    strnlen`(*s, max_length*)
  - `int    strlen`(*s*)

Basic C memory management functions:
  - void * **malloc**(size_t size)
  - void **free**(void *ptr)
  - void * **calloc**(size_t number, size_t size)
  - void * **realloc**(void *ptr, size_t size)

Write your answer on the next page. You should remove this page from the exam. **Do not write on this page.** It will not be scanned for grading.

Question 4. Write your implementation of the trie `clone` function below. Additional space is provided on the next page if you need it.

```
// Return a pointer to the root of a new trie that is an
// exact copy of the trie with root r.  If r is NULL
// (i.e., empty), return NULL.
struct tnode *clone(struct tnode *r) {



}
```

Question 4. (cont.) Additional space for your answer, if needed.

Question 5. (20 points) A bit of memory management. Recall from HW6 that we can represent the free list for the getmem/freemem storage allocator as a linked list of blocks. The beginning of each block is described by the following C struct:

```
struct free_node {            // node on free list:
  uintptr_t size;             //   number of bytes in this
                              //     block, not including the
                              //     size of this header
  struct free_node *next;     //   next block on free list or
                              //     NULL if none
};
```

(We will assume that the `size` in the `free_node` struct includes only the number of bytes of data following the header and does not include the header, although this won't really matter for this problem.)

One of the operations needed in the memory manager is to insert a new block in the free list in the correct position so that the blocks on the free list have ascending memory addresses. On the next page, implement the function `insert_free_node` that does this. You should assume that there is a global pointer variable that stores the address of the first block on the free list, and your code can access that variable without further declaration:

```
struct free_node *free_list;  // free list blocks; NULL
                              //   if free_list is empty
```

For full credit, your solution should not need to traverse the free list more than once (i.e., it should work in a single pass, not multiple passes over the list).

Simplifying assumptions: your code only needs to insert the block in the list. It does not need to detect adjacent blocks or combine blocks.

You can assume that the new block being inserted is not currently on the free list and does not overlap any blocks already on the free list.

You should assume that all necessary header files have already been `#included` and you do not need to add any `#includes`.

Write your answer on the next page. You should remove this page from the exam. **Do not write on this page.** It will not be scanned for grading.

(continued on next page)

Question 5. (cont.) Write your implementation of `insert_free_node` here. The heading of the function is written for you.

```
// Insert node b in correct place on the global free_list.
// Nodes are stored in order based on node addresses.
void insert_free_node(struct free_node *b) {




}
```

Question 6. (12 points) Testing. Describe three different *black-box* tests for the `insert_free_node` function from the previous problem. For full credit, the three tests must verify different things about the implementation, and must describe the specific input or setup for the test and the expected results. In your description of test inputs and outputs you can specify that a particular block is located at a particular address or that the free list contains blocks located at specific addresses, without having to explain how those blocks would be created at those specific addresses. Also, to simplify the question, you do not need to worry about whether block addresses are multiples of 16.

(a) Test input/setup:

Expected result:

(b) Test input/setup:

Expected result:

(c) Test input/setup:

Expected result:

Question 7. (14 points) C++. To explore a bit more C++, we've designed a simple class to represent complex numbers. A complex number is represented by a pair of doubles, holding the real and imaginary parts of the number respectively. (Don't worry if you aren't familiar with complex numbers – this is a programming problem and all the details needed are described below.)

The class declaration in file `Complex.h` looks like this:

```
#ifndef _COMPLEX_H_
#define _COMPLEX_H_

class Complex {
public:
  // Construct Complex x+y*i
  Complex(double x, double y);

  // accessors - return parts of this Complex number
  double real() const;
  double imag() const;

  // addition: return the value this + other
  Complex plus(Complex other) const;

private:
  // Representation of a Complex number: re+im*i
  double re;  // real part
  double im;  // imaginary part
};

#endif   // _COMPLEX_H_
```

On the next page, give implementations for the four functions (constructor, `real` and `imag` accessor functions, and `plus`) that are declared but not implemented above.

Reminder: The sum of two complex numbers (a+bi)+(c+di) is the complex number (a+c)+(b+d)i – in other words, simply add the real and imaginary parts together to get the real and imaginary parts of the result.

You should remove this page from the exam. **Do not write on this page.** It will not be scanned for grading.

Question 7. (cont.) Provide your implementation of the Complex class declared on the previous page as it would be written in the implementation file Complex.cc.

```
#include "Complex.h"

// write your implementation of class Complex below.
```

Question 8. (9 points) Concurrency. Consider the following code, which maintains a list of integer values.

```
struct int_node {          // nodes for an integer list:
  int val;                 //   value in this node
  struct int_node *next; //   next node or NULL if none
};

struct int_node *list = NULL;  // list of nodes; NULL if empty

// Insert new_node on the front of the list

void add_node(struct int_node *new_node) {

  new_node->next = list;

  list = new_node;

}
// remove first list node and return it; NULL if list empty

struct int_node *delete_node() {

  struct int_node *temp = list;

  if (list == NULL) return NULL;

  list = list->next;

  return temp;

}
```

Suppose we use this code in a program with two threads t1 and t2, both of which add and delete nodes from the (shared) `list` by calling `add_node` and `delete_node`.

(a) (5 points) This code is not thread-safe. Give a brief explanation of what can go wrong if the two threads are using these functions concurrently. (If it matters, assume that no other code modifies the list nodes or otherwise changes the data contained in the list.)

(b) (4 points) Good news! It turns out that the C standards committee has added an atomic statement to C, as described in lecture. A sequence of statements s1; …; sn; will be executed atomically if we surround them with atomic as follows: atomic{s1; …; sn}.

Add atomic{ … } statements to the above code so the resulting code is thread-safe.

Question 9. (6 points) Concurrency and locks. We have a program with three
concurrent threads and three shared objects X, Y, and Z. Here are two possible locking
strategies for handling access to the shared objects:

Strategy 1	Strategy 2

Thread 1
```
Acquire lock for X
Acquire lock for Y
Modify X and Y
Release lock on Y
Release lock on X
```

Thread 1
```
Acquire lock for X
Acquire lock for Y
Modify X and Y
Release lock on Y
Release lock on X
```

Thread 2
```
Acquire lock for Z
Acquire lock for X
Modify Z and X
Release lock on X
Release lock on Z
```

Thread 2
```
Acquire lock for X
Acquire lock for Z
Modify Z and X
Release lock on Z
Release lock on X
```

Thread 3
```
Acquire lock for Y
Acquire lock for Z
Modify Y and Z
Release lock on Z
Release lock on Y
```

Thread 3
```
Acquire lock for Y
Acquire lock for Z
Modify Y and Z
Release lock on Z
Release lock on Y
```

Which one of these strategies can cause a deadlock? Show a sequence of operations that
lead to the deadlock, and explain why the other strategy will never cause a deadlock.

Have a great winter break! The CSE 374 Staff