CSE 333 Midterm Exam 7/25/16

Name _____ UW ID# _____

There are 7 questions worth a total of 100 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 electronics, closed telepathy, open mind.

If you don't remember the exact syntax for something, make the best attempt you can. We will make allowances when grading.

Don't be alarmed if there seems to be more space than is needed for your answers – we tried to include more than enough blank space.

Relax, you are here to learn.

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

Score _____ / 100

1. _____ / 10

2. _____ / 9

3. _____ / 17

4. _____ / 22

5. _____ / 20

6. _____ / 20

7. _____ / 2

Question 1. (10 points) Preprocessor. Suppose we have the following two files:

defs.h:

```
#define DIV(a,b) a / b
#define INCDIV(c,d) DIV(c + 1, d + 1)
```

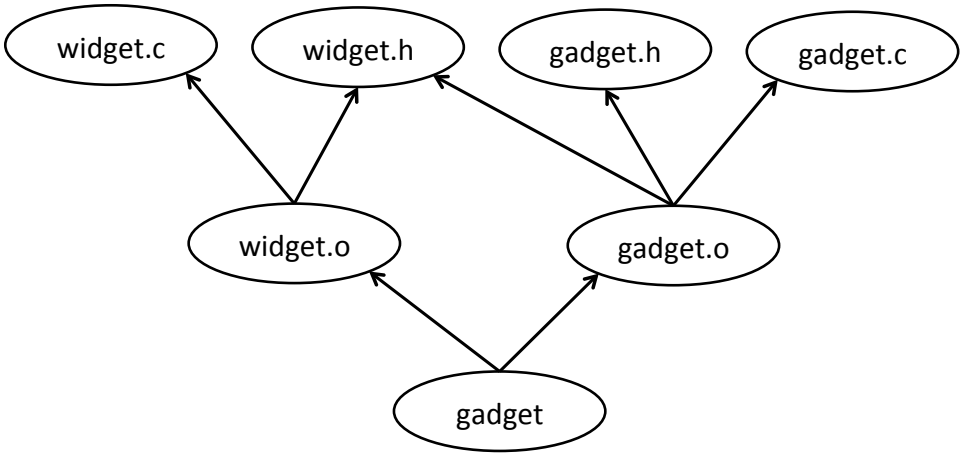
main.c:

```
#include <stdio.h>
#include "defs.h"
int main() {
  int n = INCDIV(5, 1);
  printf("%d\n", n);
  return 0;
}
```

(a) (7 points) Show the result produced by the C preprocessor when it processes file `main.c` (i.e., if we were compiling this file, what output would the preprocessor send to the C compiler that actually translates the program to machine code?). You should ignore the `#include <stdio.h>` directive since that includes library declarations that we do not have access to. Write the rest of the preprocessor output below.

(b) (3 points) What output does this program print when it is executed?

Make and related things. We have the following diagram that describes the dependencies between the various files that are involved in building an executable program named `gadget`.



Question 2. (9 points, 3 each) What can we conclude from the above diagram? In the statements below, **circle** the number of the *best* choice in each group of possible answers:

(a) The `widget.c` source file:

(i) **Definitely** contains a `#include "widget.h"` preprocessor directive.

(ii) **Might** contain a `#include "widget.h"` preprocessor directive, but might not.

(iii) **Does not** contains a `#include "widget.h"` preprocessor directive.

(b) The `gadget.c` source file:

(i) **Definitely** contains a `#include "widget.h"` preprocessor directive.

(ii) **Might** contain a `#include "widget.h"` preprocessor directive, but might not.

(iii) **Does not** contains a `#include "widget.h"` preprocessor directive.

(c) The `main` function for this program:

(i) **Definitely** is implemented (defined) in the `gadget.c` source file.

(ii) **Might** be implemented (defined) in the `gadget.c` source file, but might not.

(iii) **Is not** implemented (defined) in the `gadget.c` source file.

(more about this diagram on the next page)

Question 3. (17 points) Making stuff. Write an appropriate `Makefile` that will compile and link the files described by the dependency diagram on the previous page to build the executable program named `gadget`. Your `Makefile` should only recompile or relink files if needed to bring targets up to date. The default target in the `Makefile` should be the executable program `gadget`, i.e., if `make` is run with no arguments, the `gadget` program (target) should be built.

You should use `gcc` with appropriate options to compile the program. To save a little time and writing, you may omit `-std=c11`, but you should include `-Wall` and `-g` in appropriate places. In addition, you may find the following options to be useful:

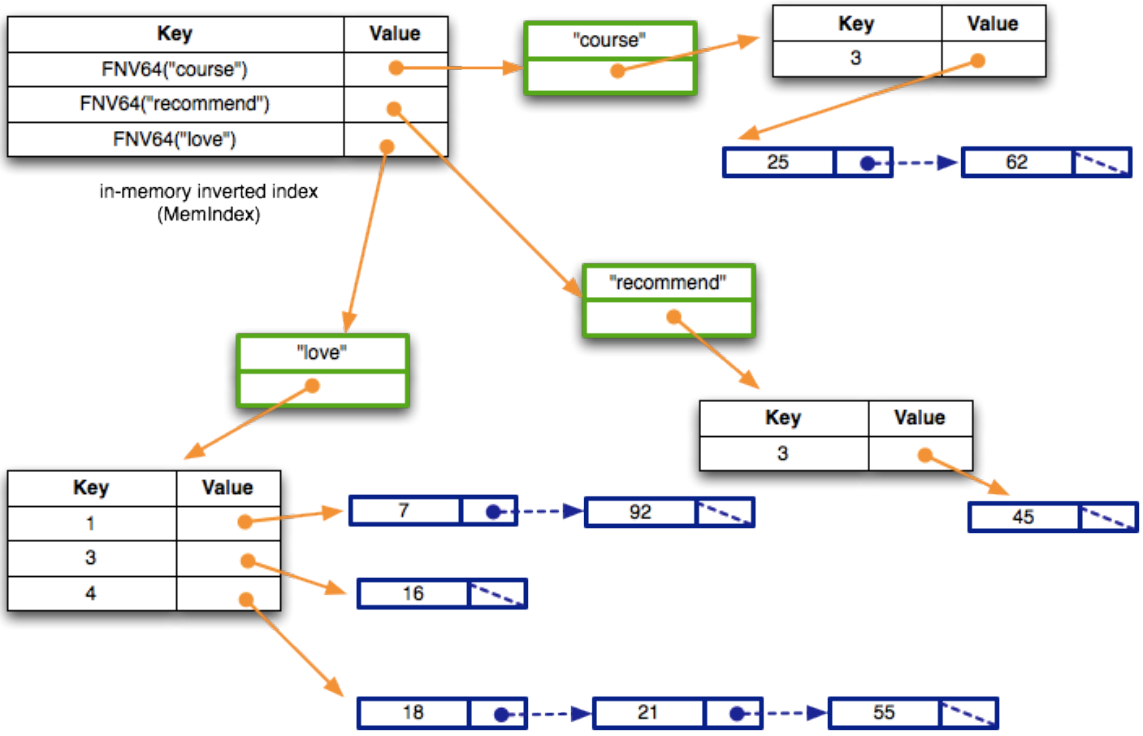
- `-o` *outputfilename* – specify gcc output file
- `-c` – only create a `.o` file and then stop without linking. (If `-c` is given, then the option `-o` *filename*.o is assumed as the default, where *filename*.c is the input file.)

Reminder: the format of a makefile rule is:

target: sources
command

Question 4. (22 points) C programming! This question involves adding a function to the code from HW2. Copies of the `HashTable.h` and `memindex.h` header files, which are the ones needed for this problem, are provided on separate pages. You should assume that those headers, and any other necessary header files and declarations, have already been included in the source files.

After building the inverted index data structures in HW2, we would like to scan the index and, for each word we find, print out the number of different documents where the word appears. For example, here is the inverted index diagram from the HW2 assignment:



The output of the function should list each word in the inverted index and the number of documents where it appears, using a separate line for each word. For the above data the output might be:

```
recommend  1
love  3
course  1
```

The words may appear in any order, and the exact spacing doesn't matter. The output should be written to `stdout` (i.e., feel free to use `printf`). On the next page, implement function `PrintWordDocCounts` to produce this output.

(Hint: Don't panic! The solution isn't particularly long – the sample solution is less than 15 or 20 lines of code. Of course, yours isn't required to be any particular length.)

Question 4. (cont.) Write your implementation of function `PrintWordDocCounts` below.

```
// For each word in MemIndex mi, print on stdout a line
// containing that word and the number of documents where
// it appears.

void PrintWordDocCounts(MemIndex mi) {
  Verify333(mi != NULL);
  // add your code here




}
```

Question 5. (20 points) Bugs 'R Us. One of your friends has been trying to learn C and has been experimenting with a linked list of strings, using dynamically allocated nodes and string values. The list allows duplicates and strings are not stored in any particular order. The problem is that the code is badly broken, and, in fact, doesn't even compile.

Make changes needed in the code below so that function `new_node` allocates a properly initialized new node and `insert` successfully adds it to the front of the list. If you want to rewrite the code instead of modifying it here, you can use the next page.

```
#include <stdlib.h>

struct strnode {          // node in a string linked list:
  char * str;             //   copy of the string (on the heap)
  struct strnode * next; //   next node or NULL if none
};


struct strnode * str_list; // global pointer to list of strings


// return a new node with copy of string s and next == NULL
struct strnode * new_node(char * s) {

  struct strnode * node;

  node->str = s;

  node->next = NULL;

  return node;

}


// add a new node with a copy of s to the front of str_list
void insert(char * s) {

  struct strnode p = new_node(s);

  str_list = p;

  p->next = str_list;

}
```

(You can use the next page if you want to write your answers there instead of editing the code on this page; if you do that you can remove this page from the exam.)

Question 5. (cont) Extra space for your answer, if you prefer to write it here.

Question 6. (20 points) Not quite the traditional what-does-it-print question. Consider the following C++ program, which, as is usual, compiles and executes with no errors.

```
#include <iostream>
using namespace std;

int f(int &n, int *pa, int &k, int *pb) {
  k = pb[1];
  pb[2] = pa[1];
  n = *pb**pa;
  return k+1;
}

int main() {
  int a = 1;
  int &b = a;
  int ray[4] = { 10, 11, 12, 13 };
  int *p = ray;
  int *q = &ray[1];

  *p = f(ray[2], p, b, q);

  cout<< "a = " << a << ", b = " << b << ", *q = " << *q << endl;
  cout<< "ray = ";
  for (int k = 0; k < 4; k++)
    cout << ray[k] << " ";
  cout << endl;

  return 0;
}
```

What output does this program produce when it runs? (You are not required to draw a boxes-n-arrows diagram, but you might find doing so to be very helpful, and it might help us if we need to assign partial credit to a not-completely-perfect answer.)

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there's something written here. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't here?

(b) (1 point) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

None of the above. My answer is _____.