NCSU ECE 209 Sections 602 and 604
UNCA CSCI 373 Section 002
Final Exam Fall 2010
9 December, 2010

This is a closed book and closed notes exam. Calculators, PDA's, cell phones, and any other electronic or communication devices may not be used during this exam.

This exam is to be turned in by 2:30 with the following exception: The five students who are taking the MAE 310 exam on the morning of 9 December may take this exam from 1:00 to 4:00.

Please read and sign the following statement:

> I have neither given not received unauthorized assistance on this test.
>
> Name: _____

If you want partial credit for imperfect answers, explain the reason for your answer!

Problem 1 (3 points)
Complete the following C statement so that the variable `myName` points to a character string containing your *clearly spelled* name.

```
char *myName =
```

Problem 2 (20 points)
Below is a sequence of C statements defining some variables

```
int    cnum = 100 ;
char   clet = 'C' ;  /* ASCII code for 'C' is 67 */
double cthd =  33.3 ;
char  *cstr = "C#" ; /* ASCII code for '#' is 35 */
```

The table below, which continues on the following page, contains two columns. The leftmost column is a C expression. In the rightmost column write the value of the expression as a C literal. If the value is a `double`, be sure to use the C syntax for `double` literals. Some of these are tricky, but none require complex arithmetic.

`++cnum`	
`cnum++`	
`cnum *= 2`	
`cnum = 5`	
`--clet`	
`cnum = clet`	
`cthd = 5`	
`&cstr[1] - &cstr[0]`	
`cstr[1]`	

`cstr[2]`	
`*cstr`	
`*cstr+1`	
`(*cstr)+1`	
`*(cstr+1)`	
`strlen(cstr)+1`	
`strlen(cstr+1)`	
`! strcmp("ab", "a")`	
`*&clet`	
`(int) 3.16`	
`1 + 2 * 3`	
`1 * 2 + 3`	
`3 % 2`	
`3 % 5`	
`100000 * 0.0`	
`1 * 3.0 / 2`	
`1 * 3 / 2`	
`1 / 2 * 3`	
`(int) 3.3 + 2.5`	
`(double) 3 / 2`	
`5 + 3`	
`5 && 3`	
`5 & 3`	
`5 || 3`	
`5 | 3`	
`5 == 3`	
`! 5 + 3`	
`5 > 4 > 3`	
`5 > 4 == 4 > 3`	
`(4+1) && (3+2)==5`	
`1 ? 3 : 5`	

Problem 3 (6 points)
Each of the following three `for` or `while` loops, which are sometimes proceeded by a few initialization statements, print numbers. For each loop, write in the three boxes below the loop the first three lines printed by loop. If the loop prints less than three lines, fill in a box for each line that is printed. Assume that `i` and `j` have already been declared as C `int` variables and that `x` has been declared as a C `double`.

```
for (x=3.5; x<5.0; x = x + 0.6) {
  printf("%4.1f\n", x) ;
}
```


```
i = 3 ;
while (i) {
  i = -i ;
  printf("%d\n", i) ;
}
```


```
for (i=1, j=0; i<10; j += i, i += 2) {
  printf("%d\n", j) ;
}
```


```
i = 4 ;
x = 6.0 ;
while (i<5.5 && x > 5.5) {
  ++i ;
  x = x - 0.6 ;
  printf("%d, %4.1f\n", i, x) ;
}
```


Problem 4 (9 points)
We're going to try fix the problematic Problem 4 of Exam 2 here.

Start off with a collection of variable definitions. Most of these involve pointers.

```
int v1 ;
int *v2 ;
int **v3 ;
const int *v4 ;
int v5[7] ;
int v6[3][5] ;
```

In the following list there are twelve assignment statements involving these variables. Because C is so accepting of clearly erroneous assignments between incompatible types, most C compilers let off bad assignments with a warning, perhaps about “incompatible pointers” or “assigning a pointer to an integer”.

For each of the following C assignment statements, state if the assignment *does* violate C's typing structure and *should* result in a warning. If a warning should be issued, write the programming mistake that is being made.

`v1 = 5 ;`
`v1 = *v2 ;`
`v2 = &v1 ;`
`v2 = &5 ;`
`v2 = v4 ;`
`v2 = v5 ;`
`v2 = v6[1] ;`
`v3 = v2 ;`
`v3 = v6 ;`
`v4 = v2 ;`
`*v4 = 5 ;`
`v5 = &v1 ;`

The remaining problems are short programming exercises based on this term's programming assignments. Your programming examples should be neatly written, indented, and adequately commented.

Problem 5 (8 points)

On the right there is a big Y that has been written using the '`.`' and '`*`' characters. Your task is to write a C *function* that implements the following prototype:

```
void printBigY(int N) ;
```

and, when called, prints a similar big Y that is `N` characters in width and height if `N` is an odd and is greater than 5, and prints nothing (not even an error message) if `N` is even or less than 5. Consequently, the following call should generate the big Y on the right.

```
PrintBigY(9) ;
```

```
*.......*
.*.....*.
..*...*..
...*.*...
....*....
....*....
....*....
....*....
....*....
```

Here's a hint to help you out with the math needed for this problem. The top part of the Y is printed in `N/2` lines. Let's index those lines with a variable `i` starting from 0. With this numbering the `i`'th line consists of `i` dots, followed by one star, followed by `N-2-2*i` dots, followed by one star, followed by `i` dots, and terminated with a newline character. The remaining lines are `N/2` dots, followed by one star, followed by `N/2` dots, and terminated with a newline.

The answer does have a lot of short `for`-loops nested inside a couple of big `for`-loops. It would be a good idea to comment your code.

```
void printBigY(int N) {
```

Problem 6 (4 points)
Suppose you are given a function which returns the number of unique values that occurs within the first `N` elements of an integer array `V`. The prototype of this function is:

```
int uniqueValues(int *V, int N) ;
```

Now write a small section of C code that examines a five element array declared as follows:

```
int Dice[5] ;
```

and prints "`legit Yahtzee`" if and only if the five values in `Dice` are all the same *and* that common value is between 1 and 6.

Only loop-free solutions will receive full credit.

Problem 7 (8 points)
Implement a function conforming to the following prototype:

```
int NumberSpaceyLines(FILE *inFILE) ;
```

that is passed a file pointer (`FILE *`) of a file already opened for input. Your function should read the entire file and returns the number of lines that consist of nothing by white space. (An empty line should also be included in the count. This makes the problem a little easier.)

You can implement the function using a simple state machine that "remembers" if it has "seen" any white space characters in the line it is presently reading.

```
int NumberSpaceyLines(FILE *inFILE) {
```

Problem 8 (8 points)
Use the function `NumberSpaceyLines` that you created for the last problem to write a C application to solve the following problem.

1) This application will receive an argument from the command line (using `argc` and `argv`).
2) The application's single argument will be the name of a file that the application should open for input.
3) If the file open fails, the application should exit after printing a helpful message.
4) The opened file pointer for the *opened input* file is passed to `NumberSpaceyLines`.
5) The value returned by `NumberSpacyLines` is written (using `printf`) to standard output in a message similar to the following:

```
There are XXX blanks lines in the file FFF.
```

where ***XXX*** is the value returned by `NumberSpaceyLines` and ***FFF*** is the name of the input file.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {









    return (EXIT_SUCCESS);
}
```

Problem 9 (12 points)

Assume that V is an array of integer of size 75 and that all the numbers stored in V are between 0 and 30. Also assume that H is an array of 31 integers.

```
int V[75] ;
int H[31] ;
```

Start by writing a section of C code to store, in H, a histogram of the values stored in V. That is, H[*i*] will be equal to the number of times the value *i* occurs in V. This is similar to the task of creating a histogram of die values in the second programming assignment.

Now use the contents of the histogram to create a *stem and leaf plot* (one of the latest "in" ways of displaying collections of data). Your stem and leaf plot will look something like the following:

```
 0:  5 6 6 7
10:  2 3 3 4 4 4 4 8 8
20:  3 3 3 5 5 5 5 7 7 7 8
30:  0 0
```

Each row corresponds to the number of values with in V that lie within a particular decade. The decade is the label before the colon. Consider the second row:

```
10:  2 3 3 4 4 4 4 8 8
```

The row is labeled with 10, so it corresponds the values of V that are between 10 and 19. The nine numbers following the label tell us that 12 appears one time, 13 appears two times, 14 appears four times, and 18 appears two times in V. Note that all the information needed to generate the stem and leaf plot appears in the histogram H so you do not need to use the original value array V to generate the plot.

If you need more room for your answer, ask the instructor for a sheet of lined paper or use the top of the next page.

For the problems of the next three pages, use the following `struct` declaration and `typedef`

```
struct LetterNode {
  char               value ;
  struct LetterNode *next ;
}
typedef struct LetterNode *LetterObj ;
```

Problem 10 (4 points)
Write a C function conforming to the following C prototype

```
LetterObj AllocLetterNode(char L) ;
```

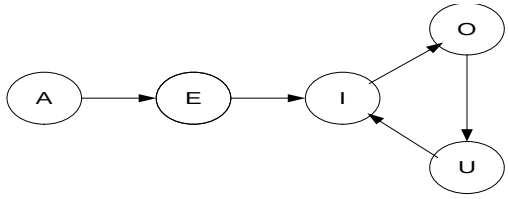
that uses dynamic storage (that is, storage allocated with `malloc`) to create, and return a `LetterObj` where the `value` field is copied from the `L` parameter. If the allocation of dynamic storage fails, `AllocLetterNode` should return `NULL`.

```
LetterObj AllocLetterNode(char L) {
```

There's a lot more space here than you should need. I just wanted the next two problems to be on the same page.

Problem 11 (4 points)

A linked list that eventually cycles back on itself is called a *lollipop*. (I'm really not making this up.) A lollipop of vowels is shown on the right.



The *starting* node of this lollipop is `A` and the *cycle entry* node is `I`.

Define an appropriate top-level structure for lollipops of `LetterNode`s that uses two pointer fields: one for the starting point and the other for the cycle entry point. Use a `typedef` to define `LetterLollipop` as a pointer to this structure. The structure definition and `typedef` should be very similar to the ones you used to define `WordLoop` in Homework 10 Part 1.

Problem 12 (6 points)

Now create the vowel lollipop of Probem 11 using the definitions you created for that same problem. First, use the `AllocLetterNode` function (Problem 10) to create the five nodes of the vowel lollipop. Next, link the five nodes into the lollipop according to the diagram. Finally create a `LetterLollipop` object that points to the starting and cycle entry nodes of the lollipop.

Problem 13 (8 points)
Finally, write a function that corresponds to the following prototype

```
int CountNodes(LetterLollipop P) ;
```

that returns the number of nodes in the lollipop.

Big hint: Go through the linked list just like you traversed the hash lists of the `WordMap` of Homework 10 Part 2. Count the number of nodes you encounter until you reach the cycle entry node the second time. For example, in the vowel lollipop, two nodes are traversed to reach the cycle entry node the first time and an additional two to reach it the second time. Add in one more for the cycle entry node and that makes a total of 5.

Be sure your function will work in the simplest case: The one node lollipop with a single node that points to itself.

```
int CountNodes(LetterLollipop P) {
```