CSE 490c – Autumn 2004
Midterm 1

Please do not read beyond this cover page until told to start.

Name: ____

There are 10 questions. Questions are worth varying numbers of points. The number of points roughly reflects an estimate of the time required to answer and double-check your answer.

TOTAL: __________________ / 50

1. [4 points]
   Explain briefly what happens when each of the following commands is typed to the shell.
   (a) *echo **

   The names of the files in the current directory (that do not begin with '.') are printed.

   (b) *echo ~*

   The full path name of the user's home directory is printed.

2. [4 points]
   Shell commands like *echo*, *ls*, *cat*, and many others are simply programs; that is, there is no code in the implementation of the shell that knows anything about them specifically. On the other hand, the command *export* is part of the shell implementation, and not a separate program. (Here is an example use of *export: export CEBWIN="white;600;800"*.)

   (a) Give a brief argument about why it's a good thing that *echo, ls,* and the others are not part of the shell implementation.

   It simplifies the implementation of the shell and makes it easier to add entirely new *"commands" or to modify existing ones.*

(b) Why, then, is *export* part of the shell implementation?

export operates on environment variables. Each process has its own set of environment variables (initialized originally with values inherited from its parent). If export were a normal command, it would execute in a process distinct from the shell, and any manipulation of environment variables it performed would apply to its own set, not that of the shell. That would be useless – any changes would be lost when export terminated.

3. [3 points]
Here is some output from a session on attu:

```
[attu1] ~> ls
-bash: ls: No such file or directory
```

Give a plausible explanation of what the problem might be. ("There is no ls executable on attu." is among the implausible answers.)

$PATH doesn't include the directory where the ls executable can be found.

4. [3 points]
Why does the world need *sed*? (That is, what does it do that most editors do not do?)

sed is a stream editor – the input to be edited comes from stdin and the result goes to stdout. That makes it useful in scripts, as it can be easily combined with other programs using pipes without requiring any temporary files to be created (as you would need with most other editors).

5. [4 points]
I want to create a file whose content is the list of full path name of all files whose name begins *src* in the file system subtree rooted at /u5/zahorjan. How can I do that? Be specific. At least some of your answer should be code (or command(s)) I can execute to get the result.

find /u5/zahorjan –name 'src' –print >filenames*

(The –print is optional, as printing is the default operation. Some kind of quoting around src is required – if you don't have it the shell will do substitution before handing the arguments to find. If there are no files with names beginning 'src' in the current directory, the result of the expansion will be 'src*', and everything will be fine. But if there are any files that match the pattern, their names will be substituted for the pattern itself.*

6. [6 points]
(a) In the source for the editor of HW3, what is the purpose of the following line:
#include <ncurses.h>

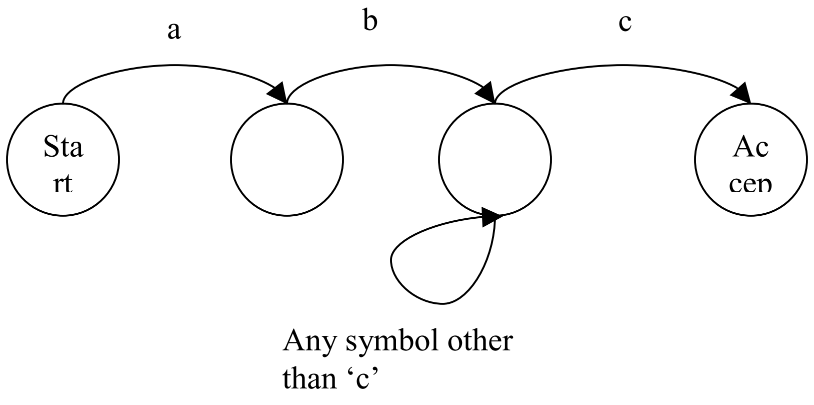
To include the declaration of the ncurses interface so that the compiler can do type checking of your code. This lets it warn you of potential type problems while its translating your C code into hardware instructions. (Note that it needs only the interface declaration of the interface to do this, not the ncurses implementation.)

(b) What is purpose of *–lncurses* in *gcc *.c –lncurses* (again, for the HW3 code)?

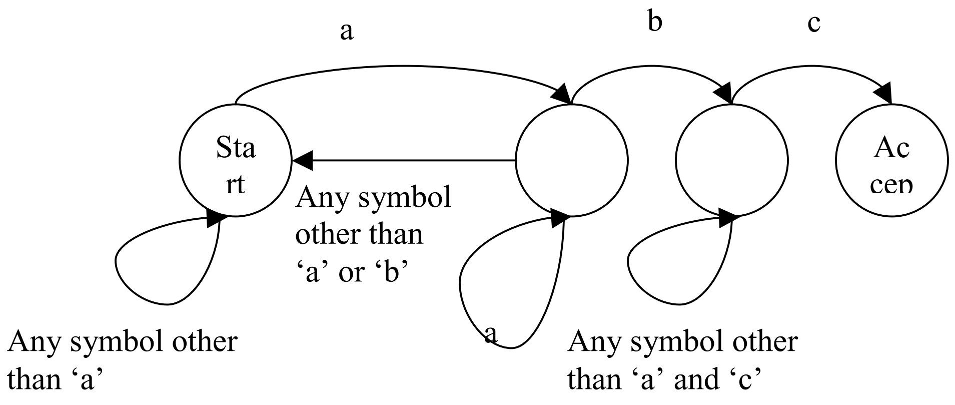
To inform the linker that it should look in the ncurses library (/usr/lib/libncurses.a) as one place it might find the compiled (hardware instruction) version of methods that have been called by your code but are not implemented by your code. This is necessary because the ncurses library is not looked in by default (unlike the standard library, /usr/lib/libc.a).

7. [4 points]
When typed to *grep*, the regular expression *ab+.*c* mean "any string that is an 'a' followed by 1 or more 'b's and then eventually followed by a 'c'.
Draw an FSA that corresponds to that regular expression.

Most people answered this in a somewhat non-grep way, giving an FSA that reached an accepting state if the input string had a prefix that matched the regular expression. Here's a version of that, with the assumption that any input symbol not listed takes us to an undrawn error state (where we stay).



grep actually looks for any match of the pattern within the file, though. Here's an FSA that does that:


a
b
c
Sta rt
Ac cep
Any symbol other than 'a' or 'b'
Any symbol other than 'a'
a
Any symbol other than 'a' and 'c'

8. [4 points]
Explain what the bug(s) is(are) in the following bit of C code:

```
 int          myInt;
char        myStr[24];
…
 scanf( "%d%s", myInt, myStr );
```

scanf() requires call-by-reference arguments, which in C means passing pointers to arguments. Because 'myStr' is an array, it's already a pointer. The argument 'myInt' should be '&myInt', though.

9. [6 points]
Give the code for a C subroutine that takes two "C strings" as parameters and returns 0 if the strings are identical and 1 if they are not. (Do NOT use the string library for this.)

```
int strcompare( char* str1, char* str2 ) {
    for (;;) {
        if ( *str1 != *str2 ) return 1;
        if ( *str1 == '\0' ) return 0;
        str1++;
        str2++;
    }
}
```

10. [12 points]
The function below is intended to take two NxN matrices, A and B, as inputs. It returns a NEW matrix, C, where c[i][j] = max(a[i][j], b[i][j]).
Complete the implementation.

```
float** matMax( int N, float** A, float** B ) {

    float**   C;
    int       row;
    int       col;

    C = (float**)malloc( sizeof(float* )* N );
    for ( row=0; row<N; row++ ) {
        C[row] = (float*)malloc( sizeof(float) * N );
        for ( col=0; col<N; col++ ) {
            C[row][col] = A[row][col] > B[row][col] ? A[row][col] : B[row][col];
        }
    }
    return C;

}
```