CSE 303 – Winter 2008
Midterm Key

1. *[2 points]*
   Give a Unix command line that will list all (and only) files that end with '.h' in the current working directory.

   *Full credit: ls *.h*
   *Extra credit: ls –a *.h (although this actually doesn't help find a file with name like .a.h)*

2. *[2 points]*
   Support the claim that the shell's command line interface is strictly more powerful than a double-click-to-launch interface (e.g., the Windows desktop). Give a significant category of things you can do with the former that you cannot do (at least not even close to easily) with the latter.

   Any of a number of things: can easily specify arguments and switches; can pipe output of one into another; can use history.

3. *[4 points]*
   Suppose you're designing an application you intend to be run by users of Unix shell interfaces, and you're interested in allowing those users to customize individual runs of your program. One way you could support that is by implementing some set of command line switches; the user could specify the switches each time the program is used to change its default behavior.
   Name two other ways you could implement that would allow users to customize each invocation.

   1. environment variable settings
   2. read settings in a startup file (e.g., emacs reads ~/.emacs)

4. [3 points]
   What is the purpose of .h files in C programs? Be specific. (For example, what function do they have at compile, link, and run times?)

   Compile time: allow type-checking of the code in the .c file
   Link time: no function
   Run time: no function

5. *[2 points]*
   Suppose my current working directory has (only) these files in it:
   `addOne addTwo addAll deleteOne deleteTwo deleteAll`
   All are executable.
   What could I type to launch `addTwo` (the one in this directory) that requires the minimal number of keystrokes possible to get it launched? That is, give a minimal length string of keystrokes that will launch it. (For the lawyers among us, we're assuming nothing special has been done in advance to help with this. Use only features available from the

shell. I said "my" current working directory on purpose.)

*./a*o*

Full credit for answers like: ./a[tab]T[tab]

Partial credit for answer that indicated knowledge of some shortcut approach

6. [8 points]
The following C function is supposed to return a free()-able string whose value is equal to a (i.e., any) longest string in the null terminated array of strings it is given as an argument. If the array is empty (has no strings), a null string is returned.
Fix all errors you see in this code. (Hint: I think there are four.)
You can write next to or on the existing code, but try to make sure I can make out what the final code is, after your modifications.

#include <stdlib.h>
#include <string.h>
#include "maxString.h"

```
char* maxString( char* stringVec[] ) {  // should be char*
    char* pResult;
    int   index;
    int   nextLen;
    int   maxLen;
    int   maxIndex;

    maxLen = -1;
    for (index =0; stringVec[index]; index++ ) {
        if ( (nextLen=strlen(stringVec[index])) > maxLen ) {
            maxLen = nextLen;
            maxIndex = index;
        }
    }

    ~~if (maxLen < 0 ) return "";~~// that literal is not free()-able
    if (maxLen < 0 ) {
             pResult = malloc(1 * sizeof(char) );
             *pResult = '\0';
             return pResult;
    }

    pResult = malloc((maxLen+1)*sizeof(char));  // space for '\0'
    ~~strcpy( stringVec[maxIndex], pResult );~~
    strcpy(pResult, stringVec[maxIndex]);  // args reversed

    return pResult;
}
```

+1 point for indicating a desire to cast the result of malloc(), even though the compiler doesn't mind a bit without the cast.

[8 points]
Write a function that is given a string as an argument and reverses the string. That is, if the string passed is in "Panama" it changes it to "amanaP".
In an attempt to make clearer what I mean, I'm giving the type signature for the function.
You should NOT use malloc() (or anything malloc()-like) in this answer.
(You do not need to specify any #include's in your answer.)

```
void reverseString( char* pString ) {
   int    startIndex, endIndex;
   char   temp;

   for ( endIndex = 0; pString[endIndex]; endIndex++ );
   endIndex--;

   for ( startIndex= 0; startIndex<endIndex; startIndex++ ) {
       temp = pString[startIndex];
       pString[startIndex] = pString[endIndex];
       pString[endIndex] = temp;
       endIndex--;
   }
}
```

[14 points]
This question asks you to write an entire program in C. It is a question about C programming, not program design. Because we have limited time, the design tries to minimize the amount you have to write – we wouldn't write it this way outside of an exam.
The program takes a single argument, the name of a file containing up to 60 lines of grade data for one student. The program is invoked like this:

```
    $ ./a.out gradeFile
```

Each line has three fields, two giving the course and one the grade. A sample input line is

```
     CSE 303 4.0
```

Once the file has been read, the program sits in a loop reading from stdin. A "command" is just a department abbreviation, e.g., CSE. The program finds all courses that were taken in that department and prints the average grade of those courses. If no courses were taken, it prints "No courses taken." Input ends with EOF.

The program is implemented as two files. Write the main.c portion on this page. Write the gradeStore.h and gradeStore.c portions on the next page. Some boiler plate is supplied to save you some writing.
Because this is a midterm, you can assume there are no errors in the input file, or the stdin input, and that no call to standard library routines ever fail. You can assume that input strings will be of reasonable lengths.

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

  FILE* fin;
  char  dept[20];
  int   course;
  float grade;
```

```
  fin = fopen(argv[1], "r");
  while ( fscanf(fin, "%s %d %f", dept, &course, &grade) != EOF ) {
    addGrade( dept, course, grade);
  }
  fclose(fin);

  while ( scanf("%s", dept) != EOF ) {
    grade =  getAverage(dept);
    if ( grade >= 0 ) {
      printf( "Avg: %f\n", grade );
    } else {
      printf( "No courses taken\n" );
    }
  }

  return 0;
}
```

File gradeStore.h:

```
#ifndef GRADESTORE_H
#define GRADESTORE_H

extern void addGrade( char* dept, int course, float grade);
extern float getAverage( char* dept);


#endif // GRADESTORE_H
```

File gradeStore.c:

```
#include <string.h>
#include "gradeStore.h"

typedef struct {
  char  dept[20];
  int   course;
  float grade;
} gradeEntry;

static gradeEntry gradeVec[60];
static int        nGrades=0;

void addGrade( char* dept, int course, float grade ) {
  strcpy( gradeVec[nGrades].dept, dept);
  gradeVec[nGrades].course = course;
  gradeVec[nGrades].grade = grade;
  nGrades++;
}

float getAverage( char* dept ) {
```

```
 float total = 0.0;
 int   num = 0;
 int   index;

 for ( index=0; index<nGrades; index++ ) {
   if ( !strcmp(gradeVec[index].dept, dept) ) {
     num++;
     total += gradeVec[index].grade;
    }
 }

 if  (num == 0 ) return -1;

 return total/num;
}
```