CS240 - Systems Programming in C
Exam #1

2005 Feb 15

"It is against the grain of modern education
to teach children to program.
What fun is there in making plans,
acquiring discipline in organizing thoughts,
devoting attention to detail,
and learning to be self-critical?"
– Alan Perlis

Instructions: Read carefully through the entire exam first, and plan your time accordingly. Note the relative weights of each segment, as a percentage of the total exam score.

This exam is **closed book, closed notes**. You may *not* refer to any books or other materials during the exam. Calculators or other electronic devices are also prohibited unless explicitly stated otherwise.

Hats should be removed, or worn brim backwards, to ensure that the proctors have an unobstructed view of your eyes.

Write your answers on this exam. You may use both sides of the page.

When you are done, present your completed exam and your student identification to the instructor or proctors at the head table. If leaving before the exam period is concluded, please leave as quietly as possible as a courtesy to your neighbors.

Name:

Student Number:

Signature:

1. (Count characters and lines - 15%)

   Write a function `countline()` which MUST use `getchar()` to read characters until `EOF` is reached. The function should count the number of characters seen, ("C",) and also the number of lines seen, ("L".) When done, output "`Characters = `C`, Lines = `L", with the correct counts. The newline character should be counted in your total number of characters, but the `EOF` should not.

```
/**
 * countline() - reads characters from standard input, counting the
 *               number of characters and lines seen.
 */
void countline()
{

    char c = 0;
    int chars = 0, lines = 0;
    while ((c = getchar()) != EOF)
    {
        chars++;
        if (c == '\n')   {   lines++;   }
    }
    printf("Characters = %d, Lines = %d\n", chars, lines);
}
```

2. (Convert Hexadecimal Numbers - 15%) Write a `htoi()` function, which takes a character string of hexadecimal integers and returns the equivalent integer. The allowable digits are `0` through `9`, `a` through `f`, and `A` through `F`. The function should return an error value if the string contains disallowed characters or does not start with a hex prefix. You may *not* use any of the string library functions.

```
/**
 * htoi() -    Given an array of characters as input that spell out a
 *             hexadecimal number, return the corresponding integer.
 * Parameters: char s[] - a null-terminated string of characters.
 * Returns:    The integer value corresponding to the input string, or
 *             -1 if string does not correspond to a properly formatted
 *             hexadecimal number.  We assume that the value fits into
 *             an integer.  Both upper and lower case hex digits are
 *             allowed, and the hex prefix can be either "0x" or "0X".
 */
int htoi(char s[])
{

   int value = 0, loop = 0;
   // Check that the proper prefix of "0x" or "0X" is here.
   if (s[0] != '0')  {  return -1;  }
   if ( (s[1] != 'x') && (s[1] != 'X') )   {  return -1;  }
   // Loop until the end of the string, adding digits.
   for (loop = 2; s[loop] != '\0'; loop++)
   {
      // Check for digits 0-9.
      if ( (s[loop] >= '0') && (s[loop] <= '9') )
      {
          value = value * 16 + s[loop] - '0';
      }
      // Check for digits 'a' to 'f'.
      else if ( (s[loop] >= 'a') && (s[loop] <= 'f') )
      {
          value = value * 16 + 10 + s[loop] - 'a';
      }
      // Check for digits 'A' to 'F'.
      else if ( (s[loop] >= 'A') && (s[loop] <= 'F') )
      {
          value = value * 16 + 10 + s[loop] - 'A';
      }
      else return -1;      // Must be an error.
   }
   return value;
}
```

3. (String Match - 15%) Write a function `strmatch(s1, s2)`, which counts the number of times that string `s2` occurs as a substring in `s1`. You may *not* use any of the string library functions.

```
/**
 * strmatch() - Given two strings as input, returns number of times second
 *             string occurs in the first string, or -1 if no match occurs.
 * Parameters: char s1[] - a null-terminated string of characters.
 *             char s2[] - another null-terminated string of characters.
 * Returns:    The number of occurences of s2 in s1, or
 *             -1 if s1 string does not contain s2 as a substring.
 */
int strmatch(char s1[], char s2[])
{

   int loop1 = 0, loop2 = 0, count = 0;
   // Loop through the length of s1.
   for (loop1 = 0; s1[loop1] != '\0'; loop1++)
   {
      // At each character in s1, start looping through s2.
      for (loop2 = 0;
           (s1[loop1 + loop2] != '\0') && (s2[loop2] != '\0');
           loop2++)
      {
          // If any of the characters don't match, move on.
          if (s1[loop1 + loop2] != s2[loop2])
             break;
      }
      // If we reached the end of s2, there was a substring.
      if (s2[loop2] == '\0') count++;
   }
   if (count == 0) return -1;
   else return count;
}
```

4. (Reverse - 15%)

   Write a function that reverses a character string in place. Take care that the null terminator is still at the end of the string when you are done.

```
/**
 * strrev() - Reverse a string in place.
 * Parameters: char s[] - a null-terminated string of characters.
 * Returns: Void.  However, the string passed in as a parameter is
 *              reversed.
 */
void strrev(char s[])
{

   int low = 0, high = 0;
   char c = 0;

   for (high = 0; s[high] != '\0'; high++)
        ;
   high--;
   while (high > low)
   {
      c = s[high];
      s[high] = s[low];
      s[low] = c;
      high--;
      low++;
   }
}
```

5. (SysID - 15%) The gcc compiler predefines several constants for the C preprocessor when it is invoked. For example, on Sparc Solaris machine “Remus”, (our lab machine in PHYS008,) the C preprocessor constant “`sparc`” is `#define`d. On Pentium Linux workstation “Elsinore”, the constant “`linux`” is `#define`d. On the PowerPC XServe machine “Davros”, running Mac OS X, the preprocessor recognizes the constant “`darwin`”.

   Write a complete program below that when compiled and run prints either “`Linux`”, “`Solaris`”, “`Mac OS X`”, or “`Unknown`”, depending on the machine.

```
#include<stdio.h>

int main()
{
#ifdef linux
        printf("Linux\n");
#elif sparc
        printf("Solaris\n");
#elif darwin
        printf("Mac OS X\n");
#else
        printf("Unknown\n");
#endif
}
```

6. (Queens: 25%) Consider a non-standard, rectangular chessboard with 80 squares, organized into 8 rows and 10 columns. Write a program that finds all combinations of 8 queens on the board such that no queen can attack any other. A main program is already provided for you, and you may assume that there is a `printboard()` function already defined in a separate file.

```
#include<stdio.h>

#define ROWS 8
#define COLS 10

void printboard(int board[]);
void solve(int board[], int row);

int main()
{
   int board[ROWS];
   int i = 0;
   // Initialize board to all zeros.
   for (i = 0; i < ROWS; i++)  {  board[i] = 0;  }
   solve(board, 0);
}
```

Write a recursive `solve()` function, and any helper functions you like, in order to finish the solution:

```
/**
 * checkboard() returns false if the queen at row is attacked by
 *              any other queen in a previous row, else returns true.
 */
int checkboard(int board[], int row)
{
  int i = 0;
  for (i = 0; i < row; i++)
    {
      if (board[i] == board[row])
        return 0; /* Same column. */
      if (abs(board[i] - board[row]) == abs(i - row))
        return 0; /* Same diagonal. */
    }
  return 1;
}
```

```
/**
 * solve() - Recursively solves queens problem.
 */
void solve(int board[], int row)
{
   int i = 0;
   // Base case - all rows are filled.
   if (row >= ROWS)
   {
       printboard(board);
       return;
   }
   // Recursive case - try a queen in each column.
   for (i = 0; i < COLS; i++)
   {
      board[row] = i;
      if (checkboard(board, row))
      {   solve(board, row + 1);   }
   }
}
```