ECS 30 **Homework #5 (50 points)** **Winter 2013**

Due: Wednesday, February 13th. Written: 4pm in 2131 Kemper. Programs: 11:55pm using handin to cs30, p5 directory.
Filenames: morse.c, sort.c.

Written (4 points)

pp. 442-443

1. Identify an error in the following C statements:
```
int x[8], i;
for (i = 0;  i <= 8;  ++i)
    x[i] = i;
```
   Will the error be detected? If so , when?

2. Declare an array of type double values called **exper** that can be referenced by using any day of the week as subscript, where 0 represents Sunday, 1 represents Monday, and so on.

3. The statement marked /* this one */ in the following code is valid. True or false?
```
int counts[10], i;
double x[5];
printf(“Enter an integer between 0 and 4> “);
i = 0;
scanf(“%d”, &counts[i]);
x[counts[i]] = 8.384;  /* this one */
```

4. What are the two common ways of selecting array elements for processing?

Zyante: 5.1 – 5.6, 11.1, 11.2, 11.4

Programming (46 points)

All programs should be able to compile with no warnings when compiled with the –Wall option, e.g. gcc –Wall taxes.c. Beginning this week we will be taking points off for warnings. You should put your name(s) in a comment on the first line of each file. When you are to write your own functions, since main() should be the first function in each file, you will need to provide a prototype above main() for each function you write. You will find my executables as well testing files in ~ssdavis/30/p5. The prompts and output format of each program must match the examples exactly. User inputs are in **bold**.

#1. Filename: morse.c (33 points, 35 minutes)

“Beginning in 1836, the American artist Samuel F. B. Morse, the American physicist Joseph Henry, and Alfred Vail developed an electrical telegraph system. This system sent pulses of electric current along wires which controlled an electromagnet that was located at the receiving end of the telegraph system. A code was needed to transmit natural language using only these pulses, and the silence between them. Morse therefore developed the forerunner to modern International Morse code.…In his earliest code, Morse had planned to only transmit numerals, and use a dictionary to look up each word according to the number which had been sent. However, the code was soon expanded by Alfred Vail to include letters and special characters, so it could be used more generally. Vail determined the frequency of use of letters in the English language by counting the movable type he found in the type-cases of a local newspaper in Morristown. The shorter marks were called "dots", and the longer ones "dashes", and the letters most commonly used were assigned the shorter sequences of dots and dashes.” from http://en.wikipedia.org/wiki/Morse_code

You are to write a program that is able to translate from English into Morse code, and from Morse code to English based on the user’s choice.

Letter	Code
A	.-
B	-…
C	-.-.
D	-..
E	.
F	..-.
G	--.
H	….
I	..
J	.---
K	-.-
L	.-..
M	--
N	-.
O	---
P	.--.
Q	--.-
R	.-.
S	…
T	-
U	..-
V	…-
W	.--
X	-..-
Y	-.--
Z	--..

Specifications:

1. main() should contain a loop that calls get_choice(), read(), and write(), and not much else.
2. The 26 Morse codes should be stored in a global const char array of strings. The start of the declaration of this array will be: `const char *codes[26] = {".-", "-...",` …
   a. Because the codes are stored as C strings, their last character will be a '\0'. This will be useful for sentinel controlled searches.
3. One space character will separate Morse letter codes.
4. When reading or writing Morse code, two consecutive spaces will indicate a space character in English.
5. get_choice() will range check the int entered. You may assume that the user will enter an integer.
6. You should use macros for 0, 1, 2.
7. You may not #include string.h, nor may you use "%s" in anywhere in your code. You should be using "%c" everywhere but in get_choice().
8. If either a "Morse" code is incorrect, or an English message contains non-alphabetic non-space character(s), your program should print a '?'.
9. Hints:
   a. As demonstrated below, you can check your work by copying the output of your write() function into the input of your read() function. Therefore it is best (and much easier) to write the write() function before the read() function.
   b. You will find isalpha() and toupper() of ctype.h useful.
   c. When reading Morse code, once you find a matching code in your codes array, to print the corresponding letter based on the index in the array is very simple.
   d. Printing a Morse code based on a letter is also very simple once you have corrected for 'A' being in the zeroth position in the array.

```
[ssdavis@lect1 p5]$ morse.out
Morse Menu
0. Done.
1. Read Morse code.
2. Write Morse code.

Your choice: 2
English message: This is a test
- .... .. ...  .. ...  .-  - . ... -
Morse Menu
0. Done.
1. Read Morse code.
2. Write Morse code.

Your choice: 1
Morse code: - .... .. ...  .. ...  .-  - . ... -
THIS IS A TEST
Morse Menu
0. Done.
1. Read Morse code.
2. Write Morse code.

Your choice: 2
English message: This & that!
- .... .. ...  ? - .... .- - ?
Morse Menu
0. Done.
1. Read Morse code.
2. Write Morse code.

Your choice: 1
Morse code: ... ---. --- ..-- .. $
S?O?I?
```

```
      Morse Menu
      0. Done.
      1. Read Morse code.
      2. Write Morse code.

      Your choice: 3
      That is not between 0 and 2.
      Please try again.

      Morse Menu
      0. Done.
      1. Read Morse code.
      2. Write Morse code.

      Your choice: 0
[ssdavis@lect1 p5]$
```

#2 Filename: sort.c (13 points, 15 minutes)

The binary search algorithm that follows may be used to search an array when the elements are in order. Here is the algorithm:

1. Let *low* be the subscript of the initial array element, and *high* be the subscript of the last array element.
2. Repeat as long as *low* is not greater than *high*, and the element is not found.
   a. Let *mid* be the subscript halfway between *low* and *high.*
   b. If the element at position *mid* is the element sought, then return its position.
      else if the element at position *mid* is greater than the element sought, then let *high* = *mid* – 1.
      else (the element at position *mid* is less than the element sought) let *low* = *mid* + 1.
3. If *low* > *high*, then return -1 to indicate that the element sought was not found.

Write a function binary_search() that implements the algorithm.In addition to implementing binary search in the function binary_srch(), you should implement insertion sort in the function insertion_sort().

In general, the best simple algorithm for sorting is insertion sort. Like selection sort, insertion sort makes N – 1 passes through the array, where N is the number of elements in the array. For pass p =1 through N -1, insertion sort ensures the elements in positions 0 through p are in sorted order. The sort makes use of the fact that elements in positions 0 through p - 1 are already sorted. Here is the algorithm:

1. For each value, V, of array in positions 1 to N:
2. Copy V to a temporary variable.
3. Start i at the position to the left of V and move toward the zeroth position, copying values one position to their right until a value is found which is less than V.
4. Copy V in the vacated position just to the right of the smaller valued position.

Insertion sort is explained below.

Your program will read from a file named data.txt that contains up to 20 elements. Once the elements are read into an array, your program should call insertion_sort(). insertion_sort() will print out the contents of the array after each pass from right to left through the array (once per value of i). Once the array is sorted, your program should the prompt the user for a value, and print out the position in the array of that value. If the value is not in the array, the program should indicate so. The program will terminate when the user enters -1.

```
[ssdavis@lect1 p5]$ sort.out
Pass  1:  12  78  13  43  89   3  76  45  34   2   7  83  14  66  42
Pass  2:  12  13  78  43  89   3  76  45  34   2   7  83  14  66  42
Pass  3:  12  13  43  78  89   3  76  45  34   2   7  83  14  66  42
Pass  4:  12  13  43  78  89   3  76  45  34   2   7  83  14  66  42
Pass  5:   3  12  13  43  78  89  76  45  34   2   7  83  14  66  42
Pass  6:   3  12  13  43  76  78  89  45  34   2   7  83  14  66  42
Pass  7:   3  12  13  43  45  76  78  89  34   2   7  83  14  66  42
Pass  8:   3  12  13  34  43  45  76  78  89   2   7  83  14  66  42
Pass  9:   2   3  12  13  34  43  45  76  78  89   7  83  14  66  42
Pass 10:   2   3   7  12  13  34  43  45  76  78  89  83  14  66  42
Pass 11:   2   3   7  12  13  34  43  45  76  78  83  89  14  66  42
Pass 12:   2   3   7  12  13  14  34  43  45  76  78  83  89  66  42
Pass 13:   2   3   7  12  13  14  34  43  45  66  76  78  83  89  42
Pass 14:   2   3   7  12  13  14  34  42  43  45  66  76  78  83  89

Please enter a value (-1 = done)> 2
2 is located at position 0 in the array.

Please enter a value (-1 = done)> 76
76 is located at position 11 in the array.

Please enter a value (-1 = done)> 55
55 is not in the array.

Please enter a value (-1 = done)> -1
[ssdavis@lect1 p5]$
```