

EEL 3801
Introduction to Computer Engineering
Summer 2005

Home Work Schedule

Schedule of Assignments:

<u>HW#</u>	<u>Week</u> <u>Due</u>	<u>Points</u>	<u>Title</u>
1	07/05/05	3%	Memory dump in assembly
2	07/19/05	3%	Solve a Maze
3	08/02/05	4%	Company Records

WARNING:

I WILL NOT ACCEPT ANY HOMEWORK AFTER 8/5/05.

All homework must include the following information. If an assignment has more than one part then each part must include this information.

HW number

Name

Short description of the work done.

Source code with comments.

Program output.

Notes:

- Comments on the source code are to explain what the code does. This will help the grader (TA) as well as your self understand how your code works and what each function does. Each function must have comments indicating exactly what that function does.
- Homework assignments are due at the BEGINING of the first class meeting for the week it is due. Late assignments will be accepted but MAY be penalized (maybe not). While I generally do not penalize, I do reserve the right to penalize for late assignments (if it gets out of hand).
- Homework assignments are to be turned in to me (course instructor, Dr. Gonzalez) or put into my mailbox or under my office door (enr 211).

Homework Assignment #1
EEL 3801 Introduction to Computer Engineering
Topic: Assembly language.

Memory dump in assembly

Write an assembly program to display the contents of memory on the screen. The program is to ask the user for 3 numbers (the segment and offset of the starting address and the number of bytes to display). The program then displays the contents of memory starting at this address and displays the correct number of bytes as specified by the user. The output should be formatted to include 16 bytes per row with a space between each byte and an area displaying any character that matches an ASCII code.

The input:

Each number is entered as a 4 digit hex number all upper case. You must prompt the user for this data. Use int 21 function 9 to output a string telling the user what to enter and how (4 characters, hex uppercase).

Use int 21 function 1 to get characters from the keyboard.

You may use this code to input 4 digits and store them in here.

```
mov  ah,1
int  21h
mov  here,al
int  21h
mov  here + 1,al
int  21h
mov  here + 2,al
int  21h
mov  here + 3,al
```

The output:

The output must properly display the ASCII codes of all of the data in the memory block. It also has a section at the end of the row to display any character whose number represents an ASCII code. The format is as follows:

```
45 A3 25 37 37 37 48 65 6C 6C 6F 0D 0A 24 4D 65 *E %777Hello $Me*
30 B9 88 13 00 40 E9 8A 51 49 24 0B 6F 70 63 6F *0 @ QI$ opco*
```

Thing you will need:

Write a function that converts 4 ASCII codes stored in memory to a 16 bit value stored in register BX. This function will receive the address of the 4 bytes in register DX. It will read each byte and convert the ASCII code to a value then store the 4 bit number in its corresponding location within BX. To convert from ASCII to binary compare the number

to 'A', if it is less than 'A' then subtract '0' from the number otherwise subtract 'A' and add 10d to the number. To store the 4 bit number into the left half of an 8-bit register move it into the register and shift it to the left 4 times. Then to move a 4-bit number into the right half of the register, simply add it.

Write a second function to convert an 8-bit binary number stored in AL to ASCII (2 ASCII codes) and save it in memory using the address that is in BX.

Hints:

Store the segment portion of the source in register ES, the offset portion in register SI and the number of bytes to display in CX. BX will point to the location in the buffer that holds the numbers to display (the left side) and DI points to the next position in the right side of the row where the ASCII codes are displayed. You will need two loops. The inner loop executes 16 times to create a row of printable data. The outer loop runs the number of bytes requested to display divided by 16 since each row displays 16 bytes. You can divide by 16 by shifting the number to the right by 4 bits (4 times). After each inner loop display the buffer using int 21h function 9.

If a number does not represent an ASCII code then print a blank, ' ', in its place in the right side of the row. The valid ASCII codes you will print are the codes in the range from 20h to 7Dh. If a number is out of this range then print a blank.

Steps to take:

First write the 2 conversion functions and test them. To test the ASCII to binary function, create a 4 byte buffer and store 4 hex digits. Then store the address of this buffer into BX and call the function. After the function executes use the Borland Debugger to view the registers (R command) and verify the data in register BX is correct. You can verify and debug the other conversion function in a similar manner.

Then write the main part of the program but instead of asking the user for data simply create a buffer with some data and process this data. This way you know what is in the data and you can verify your program. Store the address of the buffer into the appropriate registers.

Finally, once the two parts are correctly working, connect the two parts together by using the address entered by the user to get the data.

Homework Assignment #2
 EEL 3801 Introduction to Computer Engineering
 Topic: Arrays, stacks, classes and algorithms.

Solve a Maze

Write a program to solve a maze. The maze is specified as a 8 X 13 two-dimensional array of character. The walls are marked by the character 'O' and the blanks are open space. You must start at the top left corner, position 1,1 in the maze, and end at the bottom right corner at position 8,13. You can only move forward, backwards, up or down. No diagonal movement is allowed.

The maze with the outside wall is:

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0		0		0	0	0		0			0	0	0
0													0
0		0		0	0	0	0	0		0			0
0		0			0			0					0
0		0	0	0		0	0			0	0	0	0
0				0		0							0
0	0		0		0		0	0	0			0	0
0	0			0			0						0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

You can define the initial maze using an array:

```
char maze[10][16] = {"0000000000000000",
                    "0 0 0 000 0 00",
                    "0                               0",
                    "0 0 0000000 0 0",
                    "0 0  0  0  0",
                    "0 000  00  0000",
                    "0  0 0  0",
                    "00 0 0 0 000 00",
                    "00  0  0  0",
                    "0000000000000000"};
```

Algorithm:

You move by checking the four neighboring positions around you for an empty spot. Use the following order.

	4	
3	You	1
	2	

First try to move forward. If you can't then try to move down. If you can't do that either then try back and if not try moving up. As you move mark each spot visited with a * character. If the spot you visited is a dead-end (all the 4 spots around are either blocked or marked as being already visited), then mark the current spot as a dead end position, 'D', so that you will never get into it again, and back track to the previous position you were at.

To back track you can use Push() and Pop() functions. As you move push each point you visited into the stack. When you need to back track simply pop the previous position from the stack.

The pseudo code:

Set the position variable to 1 and 1 for row and column.

repeat until the maze is solved (reach position 8,13) or the maze has no solution (can not back track any more).

 Try to move to an empty position using the pattern given.

 If an empty position is found

 push the current position index onto the stack.

 move to the new position by changing the position index.

 mark the new position as visited.

 else (if an empty position if not found)

 mark the current position as dead end.

 back track.

You need to print out the initial maze and the maze with your path after you solve it.

Turn in the source code and output of your program.

Notes:

- You may be creative and use the Borland C++ or visual C++ windows facility to display the maze using a nice graphical display. This is not necessary for this homework however.

Hints:

- So that you do not wonder off out of the maze, you must check to make sure that if you are at an edge of the maze you do not consider the position outside of the maze as an option. You may alternatively do this by building a wall around the outside of the maze. Define the maze as 10 by 10 and initialize the boarder as all walls 'O'.
- If you try to backtrack and the stack is empty this means the maze has no solution.
- You may use the following code to implement the stack.

```
// The convention is to have SP point to the next available spot. This way SP
// also contains the number of items in the stack.
```

```
int    stack[200], SP = 0; // Global data
```

```
void   push(int    x)
      {
        stack[SP++] = x;
      }
```

```
int    pop()
      {
        if (SP > 0)
            return stack[--SP];
        else
            return -1;
      }
```

Homework Assignment #3

EEL 3801 Introduction to Computer Engineering

Topic: Inheritance, polymorphism, and operator overloading.

Company Records

You are to implement a program to input employee records into an array of pointers to objects. Then you will sort the items and finally print a report.

- Step 1 Create a class called Person like the one I have in the notes. Include the person's name, address and phone number as private elements. The constructor must receive all of the information from the user and initialize the object. The class should have a virtual public member function to display all of the private data and a less than function < using operator overloading that compares the person's name. Use strcmp() to do the comparison.
- Step 2 Create a class called Employee that inherits class Person. Add the person's yearly salary to the private data. Add a member function to display the data. This function should call Person's display as well and it is to be declared virtual.
- Step 3 Create a class called Volunteer that inherits class Person. Add the person's hours per week of work to the private data. Add a member function to display the data. This function should call Person's display as well and it is to be declared virtual.
- Step 4 Create an array with 5 pointers to class Person. Initialize this array by traversing the array. For each component ask the user to enter all of the information the object needs (volunteer or employee, name, address, phone and salary or hours of work depending on what's appropriate). Then dynamically allocate a new object of type Employee or Volunteer depending on the type of person and send its constructor the data gathered from the user.
- Step 5 Sort this array using the bubble sort algorithm. For the comparisons you may use the operator < function you defined as part of class Person.
- Step 6 Print a report of all of the records by traversing the array and calling the objects display function.