**Lab Hard 2**

**Assembly Language**

1. Write a program that lets the user type some text, consisting of words separated by blanks, ending with a carriage return, and displays the text in the same word order as entered, but with the letters in each word reversed.
For example, "this is a test" becomes "siht si a tset".
2. To sort an array A of N elements by the bubblesort method, we proceed as follows:

*Pass 1.* For *j* = 2 … *N*, If A[*j*] < A[*j*-1] then

swap A[*j*] and A[*j*-1].

This will place the largest element in position *N*.

*Pass 2.* For *j* = 2 … *N*-1, if A[*j*] < A[*j*-1] then

swap A[*j*] and A[*j*-1].

This will place the 2nd largest element in position *N*-1.

…

*Pass N-1.* If A[2] < A[l], then swap A[2] and A[l].

At this point the array is sorted.

**Demonstration**

initial data 7 5 3 9 1

pass 1 5 3 7 1 9

pass 2 3 5 1 7 9

pass 3 3 1 5 7 9

pass 4 1 3 5 7 9

Write a procedure BUBBLE to sort a byte array by the bubblesort algorithm. The procedure receives the offset address of the array in SI and the number of elements in BX. Write a program that lets the user type a list of single-digit numbers, with one blank between numbers, calls BUBBLE to sort them, and prints the sorted list on the next line. For example,

?2 1 6 5 3 7

1 2 3 5 6 7

Your program should be able to handle an array with only one element.

1. Modify procedure INDEC so that it will check for overflow.
2. A problem in elementary algebra is to decide if an expression containing several kinds of brackets, such as, [,],{,},(,), is correctly bracketed. This is the case if (a) there are the same number of left and right brackets of each kind, and (b) when a right bracket appears, the most recent preceding unmatched left bracket should be of the same type. For example,

(a+[b-{cx(d-e)}]+f) is correctly bracketed, but

(a+[b-{cx(d-e)]}+f) is not

Correct bracketing can be decided by using a stack. The expression is scanned left to right. When a left bracket is encountered, it is pushed onto the stack. When a right bracket is encountered, the stack is popped (if the stack is empty, there are too many right brackets) and the brackets are compared. If they are of the same type, the scanning continues. If there is a mismatch, the expression is incorrectly bracketed. At the end of the expression, if the stack is empty the expression is correctly bracketed. If the stack Is not empty, there are too many left brackets.

Write a program that lets the user type in an algebraic expression, ending with a carriage return, that contains round (parentheses), square, and curly brackets. As the expression is being typed in, the program evaluates each character. If at any point the expression is incorrectly bracketed (too many right brackets or a mismatch between left and right brackets), the program tells the user to start over. After the· carriage return is typed, If the expression is correct, the program displays "expression Is correct." If not, the program displays "too many left brackets". In both cases, the program asks the user if he or she wants to continue. If the user types 'Y', the program runs again. Your program does not need to store the input string, only check it for correctness.

**Sample execution:**

ENTER AN ALGEBRAIC EXPRESSION:

(a + b)] TOO MANY RIGHT BRACKETS. BEGIN AGAIN!

ENTER AN ALGEBRAIC EXPRESSION

(a + [b - c] x d)

EXPPESSION IS CORRECT

TYPE Y IF YOU WANT TO CONTINUE:Y

ENTER AN ALGEBRAIC EXPRESSION:

[a + b x (c - d) – e}BRACKETS MISMATCH. BEGIN AGAIN!

ENTER AN ALGEBRAIC EXPRESSION:

((a + [b - {c x (d - e) } ] + f)

TOO MANY RIGHT BRACKETS. BEGIN AGAIN!

ENTER AN ALGEBRAIC EXPRESSION:

I'VE HAD ENOUGH

EXPRESSION IS CORRECT

TYPE Y IF YOU WANT TO CONTINUE:Y