

Part I – Programming

You will create a program that will perform the function of a postfix calculator. This means that it will perform in a similar manner to that of a Hewlett Packard calculator.

As input you will expect to receive three things in the following order: first number, second number, and arithmetic operation (addition, subtraction, multiplication, and division). You must prompt the user for all of these pieces of information and continually prompt the user for the information until valid input is received. Valid input for the first two numbers can be assumed; however, valid input for the operation cannot be assumed and you must perform error checking. The following characters constitute valid input: “+” for addition, “-” for subtraction, “*” for multiplication, and “/” for division. When valid input is received as stated, then that arithmetic operation is performed.

You will then perform the arithmetic operation and display the result. After displaying the result the user must be prompted with a message asking if they would like to continue and perform an operation. There are only two valid inputs for this prompt, all other inputs will terminate the program. Yes, or “Y” will repeat the aforementioned process while “N,” or no, will terminate the program. Input at this stage is not case sensitive and an input of “Y” is equivalent to an input of “y.”

In summary, you ask for the first number, then the second number, the operation to apply to those two numbers (and continually ask until a valid input as defined is received), display the results, then ask the user if they would like to do the whole process over again with new numbers and a new arithmetic operation. You may use ONLY use “GetDec,” and “PutDec.” You may ONLY use the interrupts for displaying information to the screen, and as always no pre-defined procedures or macros unless you have created them. The only two macro exceptions for this project are “Cls,” and “SetCsrPos.”

Grade distribution:

Part 1	
Addition	5%
Subtraction	5%
Multiplication	5%
Division	5%
Operation	15%
Looping	15%
Part 2	
Question 1	10%
Question 2	10%
Following Instructions	10%
Comments	
Program Description	5%
Code Comments	5%
Misc./Efficiency	10%

Part II – Questions and Answers

1) Translate the following C code into Assembly language

```
If ((foo == 3) || (bar != 2))  
{  
    bar=1;  
    foo=foo*foo;  
}
```

```
cmp foo, 3  
jz dostuff  
cmp bar, 2  
jz done  
dostuff:  
    mov bar, 1  
    mov ax, foo  
    cwd  
    imul foo  
    mov foo, ax  
done:  
    nop
```

2) What is the range of a regular JMP instruction in the small model?

-128 ~ 127

comment ~ *****

Name: Bubba
SSN: 123-45-6789
Section: 09422
Date: 3/2/2004
Assignment: Program 2

Description:

Postfix calculator. This program takes in two numbers
The first number is requested from the user and is converted to
binary and is binary equivalent is displayed to the screen.
The program then asks for another number which it raises the second
number to (an exponent) and displays the result

Input:

None

Output:

None

~ *****

.model small

.586

.stack 100h

.nolist

```
;*****  
;Includes  
;*****  
include bios.inc  
includelib util.lib
```

.list

.data

```
CR equ 0Dh  
LF equ 0Ah  
PLUS equ "+"  
MINUS equ "-"  
DIVIDE equ "/"  
MULTIPLY equ "*"  
YES1 equ "Y"  
YES2 equ "y"  
  
menu db CR,LF  
db " ***** ",CR,LF  
db " * ",CR,LF  
db " * Postfix Calculator * ",CR,LF  
db " * ",CR,LF  
db " ***** ",CR,LF  
  
menulen equ $ - menu  
prompt1 db CR,LF,"First number:"  
promptlen1 equ lengthof prompt1  
prompt2 db "Second number:"  
promptlen2 equ lengthof prompt2  
prompt3 db "Operation:"  
promptlen3 equ lengthof prompt3  
response db CR,LF,"Result:"  
reslen equ lengthof response  
errormes db CR,LF,"Invalid selection, choose again",CR,LF  
errorlen equ lengthof errormes  
choice db "Perform more operations (Y or y to ",  
"continue, press any other key to stop):"  
choicelen equ lengthof choice  
crlf db CR,LF  
number1 sword ?  
number2 sword ?  
character db ?
```

.code

```
displaymsg proto c m:word, l:word  
keyread proto c k:word  
EXTERN GetDec:near  
EXTERN PutDec:near  
  
main proc  
    .startup  
    invoke displaymsg, addr menu, menulen ;display menu banner  
again:  
    invoke displaymsg, addr prompt1, promptlen1 ;prompt user for first number  
    call getdec ;get first number from the user  
    mov number1, ax ;move the number into memory  
    invoke displaymsg, addr prompt2, promptlen2 ;prompt user for second number  
    call getdec ;get second number the user  
    mov number2, ax ;move the number into memory  
operation:  
    invoke displaymsg, addr prompt3, promptlen3 ;prompt user for the action to take  
    invoke keyread, addr character ;read the response from user
```

```

        mov al, character                ;move the character into AL
        cmp al, PLUS                    ;compare AL to "+"
        jz doadd                        ;if equal, jump to addition label
        cmp al, MINUS                  ;compare AL to "-"
        jz dosub                        ;if equal, jump to subtraction label
        cmp al, MULTIPLY               ;compare AL to "*"
        jz domul                        ;if equal, jump to multiplication label
        cmp al, DIVIDE                 ;compare AL to "/"
        jz dodiv                        ;if equal, jump to division label
        invoke displaymesg, addr errormes, errorlen ;fall through to display error message
        jmp operation                  ;jump to operation and ask again

doadd:
        mov ax, number1                ;move first number into AX
        mov bx, number2                ;move second number into BX
        add ax, bx                      ;add AX and BX, store in AX
        jmp choose                      ;jump to choose label

dosub:
        mov ax, number1                ;move first number into AX
        mov bx, number2                ;move second number into BX
        sub ax, bx                      ;subtract BX from AX, store in AX
        jmp choose                      ;jump to choose label

domul:
        mov dx, 0                      ;clear out DX register
        mov ax, number1                ;move first number into AX
        mov bx, number2                ;move second number into BX
        imul bx                         ;signed multiplication of AX and BX, store in AX
        jmp choose                      ;jump to choose label

dodiv:
        mov ax, number1                ;move first number into AX
        mov bx, number2                ;move second number into BX
        cwd                             ;convert AX word to double word
        idiv bx                         ;signed divide AX by BX, store in AX

choose:
        invoke displaymesg, addr response, reslen ;display result message
        call putdec                     ;display result of arithmetic operation to stdout
        invoke displaymesg, addr crlf, 2   ;display carriage return, linefeed
        invoke displaymesg, addr choice, choicelen ;prompt user for choice to continue
        invoke keyread, addr character     ;read user selection from keyboard
        cmp character, YES1              ;compare user choice to "Y"
        jz again                         ;if equal, jump to again
        cmp character, YES2              ;compare user choice to "y"
        jz again                         ;if equal, jump to again
        .exit

main endp

```

```

;*****

```

```

;Function: displaymesg
;Description:
;   This function takes a pointer to a string
;   as input and the length of the string and
;   prints the result to stdout. It does this
;   by using the OS int 21h and function 040h.
;   BX holds the device (1), cx holds the
;   length of string, dx holds the address of
;   the string to be displayed. Flags and
;   the 32-bit registers are pushed and popped
;   of the stack.
;Input:
;   m - a pointer to the string which is to be
;       displayed to stdout
;   l - an short integer indicating the length
;       of the string to display
;Output:
;   None
;Return:
;   None
;*****

```

```

displaymesg proc near c uses ax bx cx dx, m:word,l:word

```

```

    pushf
    pushad
    mov ah, 040h
    mov bx, 1
    mov cx, l
    mov dx, m
    int 21h
    popad
    popf
    ret

```

```

displaymesg endp

```

```

;*****

```

```

;Function: keyread
;Description:
;   This function takes a pointer to a string

```

```

;      as an inputbuffer and attempts to read the
;      input from stdin. The keystrokes are stored
;      as characters in the inputbuffer
;      It does this by using the OS int 21h and
;      function 0Ah. Dx holds the address of
;      the string to be displayed. Flags and
;      the 16-bit registers (AX, DX are pushed and
;      popped
;      of the stack.
;Input:
;      input - a pointer to the string which is
;              to be modified
;Output:
;      input - a pointer to the string which has
;              been modified
;Return:
;      None
;*****
keyread proc near c uses ax dx, k:word
    pushf                ;push the flags onto the stack
    mov ah, 01h          ;put function number 0AH into AH
    int 21h             ;request service from OS
    mov bx, k
    mov BYTE PTR [bx], al; ;put the address of inputbuffer into DX
    popf                ;pop flags off of stack
    ret                 ;return control flow to the caller
keyread endp

```