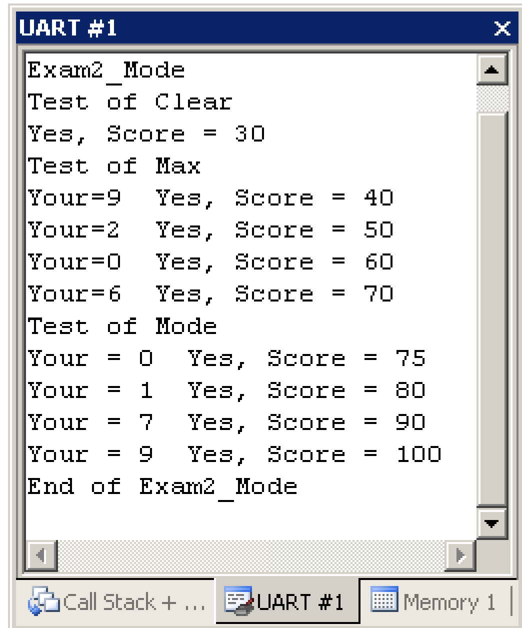
First:________________ Last:______________________

Scoring The correct output values are shown in the figure on the right.
Your grade will be based both on the numerical results returned by your program and on your programming style. In particular, write code that is easy to understand, easy to debug, easy to change. Please employ good labels, pretty structure, and good comments.



Performance Score= Run by TA at the checkout		TA:

I promise to follow these rules

This is a closed book exam. You must develop the software solution using the **Keil uVision** simulator. You have 35 minutes, so allocate your time accordingly. You are allowed to bring only some pencils (no books, laptops, cell phones, hats, disks, CDs, or notes). You will have to leave other materials up front. Each person works alone (no groups). You have full access to **Keil uVision**, with the **Keil uVision** help. You may use the Window's calculator. You sit in front of a computer and edit/assemble/run/debug the programming assignment. You do NOT have access the book, internet or manuals. You may not take this paper, scratch paper, or rough drafts out of the room. You may not access your network drive or the internet. You are not allowed to discuss this exam with other EE319K students until Thursday.

The following activities occurring during the exam will be considered scholastic dishonesty:

1) running any program from the PC other than **Keil uVision**, or a calculator,
2) communicating with other students by any means about this exam until Thursday,
3) using material/equipment other than a pen/pencil.

Students caught cheating will be turned to the Dean of Students.

Signed: ______________________ October 31, 2012

Procedure

First, you will log onto the computer and download files from the web as instructed by the TAs.

Web site xxxxxx
user: xxxxxx
password: xxxxxx

Unzip the folder placing it in a temporary folder. You are not allowed to archive this exam. Within **Keil uVision** open these files, put your name on the first comment line of the file **Exam2.s**. Before writing any code, please assemble and run the system. You should get output like the figure above (but a much lower score). You may wish create backup versions of your program. If you wish to roll back to a previous version, simply open one of the backup versions.

My main program will call your subroutines multiple times, and will give your solution a performance score of 0 to 100. *You should not modify my main program or my example data.* When you have written your subroutines, you should run my main program, which will output the results to the **UART#1** window. After you are finished, raise your hand and wait for a TA. The TA will direct you on how to complete the submission formalities. The TA will run your program in front of you and record your performance score on your exam cover sheet. The scoring page will not be returned to you. The submission guidelines are at the end of the exam handout.

Mode is an operation that finds the value which occurs most frequently in a list of numbers.

Part a) The first subroutine, called **Clear**, should initialize an array of 8-bit numbers (address to which is in Register R0). Each element of the array is stored in one 8-bit byte, located at sequential memory addresses. The size of the array is fixed at 10.

Input parameters: R0 has a pointer to an array of 8-bit numbers that should be initialized
Output: No formal return parameter

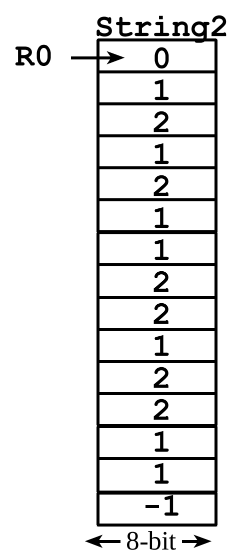
A typical calling sequence is

```
  LDR  R0,=Array1   ; pointer to array
  BL   Clear        ; all 10 entries become zero
```

Part b) Write a second assembly language subroutine, called **Max**, which returns the index of the largest element in the array. The first element is index 0 and the last one is index 9. There are 10 unsigned 8-bit numbers in the array, the address to which is passed in R0. Each element of the array is stored in one 8-bit byte, located at sequential memory addresses. The size of this array is fixed at 10 entries. There will be at least one nonzero value. The result is returned in Register R0.

```
Case1 DCB 1,2,3,4,5,6,7,8,9,10            ; returns R0=9 (the 10)
Case2 DCB 1,2,100,12,13,14,15,16,17,18    ; returns R0=2 (the 100)
Case3 DCB 200,2,3,10,100,12,13,14,15,16   ; returns R0=0 (the 200)
Case4 DCB 0,2,200,8,100,12,201,14,15,199  ; returns R0=6 (the 201)
```

Input parameters: R0 has a pointer to an array of ten 8-bit unsigned numbers
Output: R0 is the index of the largest value (0 to 9)

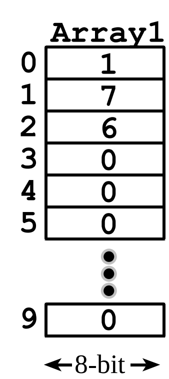


A typical calling sequence is

```
  LDR  R0,=Case2  ; pointer to array
  BL   Max        ; returns R0=2 because 100 is at index 2
```

Part c) Write a third assembly language subroutine, called **Mode**, which returns the mode of a variable length string of 8-bit numbers. The string only contains the numbers from 0 to 9 and is terminated by a -1. The mode is the element that occurs most frequently in the string. There are three steps to find the mode. First, initialize **Array1** to zero (call **Clear**). Second, iterate through the input string from start to finish. For each number in the input string increment a corresponding count in **Array1**. Lastly, search **Array1** for the largest count (call **Max**).

```
String1 DCB 0,-1                                ; mode is 0
String2 DCB 0,1,2,1,2,1,1,2,2,1,2,2,1,1,-1      ; mode is 1
String3 DCB 6,7,7,7,6,9,8,8,7,7,7,7,4,3
        DCB 7,7,9,7,7,7,-1                      ; mode is 7
String4 DCB 2,3,4,7,6,9,9,8,7,7,9,7,4,3
        DCB 7,7,9,7,7,9,9,9,9,9,9,-1            ; mode is 9
```



The figure shows a string with one 0, seven 1s, and six 2s. First, you clear **`Array1`** so all values are zero. The figure shows **`Array1`** after the scan. Basically you are counting the number of occurrences of each number. Using your **`max`** function you determine that the largest value (7) occurs at index 1. Index 1 means the number 1.

Input parameters: R0 has a pointer to a -1-terminated string
Output: R0 number (0 to 9) occurring most frequently in the string

A typical calling sequence is

```
LDR  R0,=String2 ; pointer to string
BL   Mode        ; returns R0=1 because it occurred most
```

__Important Notes__:

- Your subroutines should work for all cases shown in the starter file.
- Handle the simple cases first and the special cases last.

Note that calling your subroutine in part (a) in part (b) will greatly reduce the amount of code you will need to write.

__**Submission Guidelines**__:

- Log onto Blackboard and submit your Exam2.s source file into the Exam2 field. Be careful because only one submission will be allowed.

Memory access instructions

```
LDR    Rd, [Rn]        ; load 32-bit number at [Rn] to Rd
LDR    Rd, [Rn,#off]   ; load 32-bit number at [Rn+off] to Rd
LDR    Rd, =value      ; set Rd equal to any 32-bit value (PC rel)
LDRH   Rd, [Rn]        ; load unsigned 16-bit at [Rn] to Rd
LDRH   Rd, [Rn,#off]   ; load unsigned 16-bit at [Rn+off] to Rd
LDRSH  Rd, [Rn]        ; load signed 16-bit at [Rn] to Rd
LDRSH  Rd, [Rn,#off]   ; load signed 16-bit at [Rn+off] to Rd
LDRB   Rd, [Rn]        ; load unsigned 8-bit at [Rn] to Rd
LDRB   Rd, [Rn,#off]   ; load unsigned 8-bit at [Rn+off] to Rd
LDRSB  Rd, [Rn]        ; load signed 8-bit at [Rn] to Rd
LDRSB  Rd, [Rn,#off]   ; load signed 8-bit at [Rn+off] to Rd
STR    Rt, [Rn]        ; store 32-bit Rt to [Rn]
STR    Rt, [Rn,#off]   ; store 32-bit Rt to [Rn+off]
STRH   Rt, [Rn]        ; store least sig. 16-bit Rt to [Rn]
STRH   Rt, [Rn,#off]   ; store least sig. 16-bit Rt to [Rn+off]
STRB   Rt, [Rn]        ; store least sig. 8-bit Rt to [Rn]
STRB   Rt, [Rn,#off]   ; store least sig. 8-bit Rt to [Rn+off]
PUSH   {Rt}            ; push 32-bit Rt onto stack
POP    {Rd}            ; pop 32-bit number from stack into Rd
ADR    Rd, label       ; set Rd equal to the address at label
MOV{S} Rd, <op2>       ; set Rd equal to op2
MOV    Rd, #im16       ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2>       ; set Rd equal to -op2
```

Branch instructions

```
B    label   ; branch to label    Always
BEQ  label   ; branch if Z == 1   Equal
BNE  label   ; branch if Z == 0   Not equal
BCS  label   ; branch if C == 1   Higher or same, unsigned ≥
BHS  label   ; branch if C == 1   Higher or same, unsigned ≥
BCC  label   ; branch if C == 0   Lower, unsigned <
BLO  label   ; branch if C == 0   Lower, unsigned <
BMI  label   ; branch if N == 1   Negative
BPL  label   ; branch if N == 0   Positive or zero
BVS  label   ; branch if V == 1   Overflow
BVC  label   ; branch if V == 0   No overflow
BHI  label   ; branch if C==1 and Z==0  Higher, unsigned >
BLS  label   ; branch if C==0 or  Z==1  Lower or same, unsigned ≤
BGE  label   ; branch if N == V   Greater than or equal, signed ≥
BLT  label   ; branch if N != V   Less than, signed <
BGT  label   ; branch if Z==0 and N==V  Greater than, signed >
BLE  label   ; branch if Z==1 and N!=V  Less than or equal, signed ≤
BX   Rm      ; branch indirect to location specified by Rm
BL   label   ; branch to subroutine at label
BLX  Rm      ; branch to subroutine indirect specified by Rm
```

Interrupt instructions

```
CPSIE  I              ; enable interrupts  (I=0)
CPSID  I              ; disable interrupts (I=1)
```

Logical instructions

```
AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2    (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2    (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2    (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2) (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2) (op2 is 32 bits)
LSR{S} Rd, Rm, Rs      ; logical shift right Rd=Rm>>Rs  (unsigned)
LSR{S} Rd, Rm, #n      ; logical shift right Rd=Rm>>n   (unsigned)
ASR{S} Rd, Rm, Rs      ; arithmetic shift right Rd=Rm>>Rs (signed)
```

```
ASR{S} Rd, Rm, #n      ; arithmetic shift right Rd=Rm>>n  (signed)
LSL{S} Rd, Rm, Rs      ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n      ; shift left Rd=Rm<<n  (signed, unsigned)
```

Arithmetic instructions

```
ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12 ; Rd = im12 – Rn
CMP    Rn, <op2>       ; Rn – op2      sets the NZVC bits
CMN    Rn, <op2>       ; Rn - (-op2)   sets the NZVC bits
MUL{S} {Rd,} Rn, Rm    ; Rd = Rn * Rm       signed or unsigned
MLA    Rd, Rn, Rm, Ra  ; Rd = Ra + Rn*Rm    signed or unsigned
MLS    Rd, Rn, Rm, Ra  ; Rd = Ra - Rn*Rm    signed or unsigned
UDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         unsigned
SDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         signed
```

Notes `Ra Rd Rm Rn Rt represent 32-bit registers`

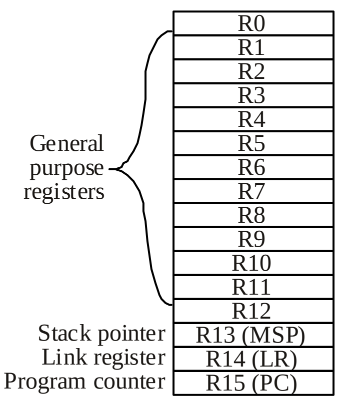
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 to 4095
label   any address within the ROM of the microcontroller
op2     the value generated by <op2>
```

Examples of flexible operand **<op2>** creating the 32-bit number. E.g., **Rd = Rn+op2**

```
ADD Rd, Rn, Rm         ; op2 = Rm
ADD Rd, Rn, Rm, LSL #n ; op2 = Rm<<n  Rm is signed, unsigned
ADD Rd, Rn, Rm, LSR #n ; op2 = Rm>>n  Rm is unsigned
ADD Rd, Rn, Rm, ASR #n ; op2 = Rm>>n  Rm is signed
ADD Rd, Rn, #constant  ; op2 = constant
```

, where **X** and **Y** are hexadecimal digits:

- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form **0x00XY00XY**
- in the form **0xXY00XY00**
- in the form **0xXYXYXYXY**



Condition code bits

N negative
Z zero
V signed overflow
C carry or
unsigned overflow