1. Matching. Indicate the letter of the best description. (3 pts. each)

- **L** actual parameter
- **K** formal parameter
- **C** .data section
- **A** .rodata section
- **J** .text section
- **G** "stack" section
- **H** "heap" section
- **O** .align 4
- **M** a₀, a₁, a₂, ...
- **N** a₀, a₁₀, a₂₀, ...
- **E** call-by-reference
- **F** call by value-result

A. read-only memory region for initialized data
B. write-only memory region for initialized data
C. read-write memory region for initialized data
D. copy value of actual parameter into space allocated for formal parameter.
E. copy address of actual parameter into space allocated for formal parameter
F. copy value of actual parameter into space allocated for formal parameter, and copy final value of formal parameter back into actual parameter
G. memory region for stack frames
H. memory region for dynamically allocated data
I. memory region for assembly language source
J. memory region for machine instructions
K. parameter defined in subroutine header
L. parameter passed to subroutine from caller
M. row-major memory order for an array
N. column-major memory order for an array
O. memory alignment directive for words
P. directive to move location counter to 4

11. Show the addressing arithmetic expression for calculating the byte address of element p[i] in an array declared in C as "int p[200]". Note that for the purposes of this question, sizeof(int) is 4. (No ARM code is necessary). Use the address "p" as the base address of the array and use 0-origin indexing.) (4 pts.)

\[ p + 4^*i \]

12. Identify three items in a generic stack frame. (6 pts)

parameters, local variables, registers to save
13. Consider this program and subroutine

```c
int a = 5; /* global variable */

int subr( int b, int c )
{
    a = 4*a;
    b = b + 3;
    c = c + 2;
    return( a + b + c );
}

void main(void){
    int d = 1,
    int e;
    e = subr(a, d);
}
```

Show final values after calls to subr() for the variables listed below, by column, according to the specified parameter passing methods. (18 pts. total)

<table>
<thead>
<tr>
<th></th>
<th>b: call by value</th>
<th>b: call by value-result</th>
<th>b: call by reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>14. a</td>
<td>20</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>15. d</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16. e</td>
<td>31</td>
<td>31</td>
<td>49</td>
</tr>
</tbody>
</table>
17. Consider the following ARM code for the subroutine fn2: (3 pts each)

```
.global fn2
fn2:  
  add sp, sp, #0
  mov r5, sp
  ldr r6, [r0]
  add r3, r1, r3
  ldr r0, [r2, #40]
  add r0, r0, r3
```

a. The parameter in r0 is passed by reference. How can this be observed from the ARM code?
   a load instruction was needed to access the value

b. The parameter in r1 is passed by value. How can this be observed from the ARM code?
   it is used directly; no extra load

18. Give the ARM code for the following C functions. Use m4 defines to provide symbolic register names and use these symbolic names in your code. (20 pts.)

```
void set_a(int a[], int i, int j){
  a[i] = j;
}

.set_a:
  .global set_a
  .type set_a, %function

set_a:
  push {r4, lr}
  mov r4, r1, lsl #2
  str r2, [r0, r4]
  pop {r4, pc}  @ OR pop {r4, lr}  @ bx lr
```
Problem 18 continued

```c
void init_a( int x[], int n, int val ){
    int i, sum = 0;         .global init
    for( i=0; i<n; i++ ){
        x[i] = val;
    }
}
```

```
.globl init_a
.type init_a, %function

.init_a:
    push {r4, r5, lr}
    mov r4, #0
    cmp r4, r1             @ check to see if r4 == n
        bge finish
    loop:
        mov r5, r4, lsl #2
        str r2, [r0, r5]
        add r4, r4, #1
        cmp r4, r1             @ check to see if r4 == n
        blt loop
    finish:
        pop {r4, r5, pc}
```
19. Give ARM assembly code that implements the following C function: (15 pts.)

```c
int descend(int *a, int *b)
{
    if(*a < *b)
    {
        int temp = *a;
        *a = *b;
        *b = temp;
    }
}
```

```assembly
.global descend
.type descend, %function

descend:
    @ addr a is in r0
    @ addr b is in r1
    push {lr} @ save return address
    ldr r2, [r0] @ move a into r2
    ldr r3, [r1] @ move b into r3
    cmp r2, r3
    bge done
    str r3, [r0] @ store indirectly into a
    str r2, [r1] @ store indirectly into b

done:
    mov r0, #0
    pop {pc}
```