Problem 1 (2 parts, 30 points) **Storage Allocation, Arrays, and Pointers**

Part A (16 points) Assuming a **32-bit system with 32-bit memory interface and 32-bit addresses**, show how the following global variables map into static memory. Assume they are allocated starting at address 4000 and are properly aligned. **For each variable, draw a box showing its size and position** in the memory shown below in which byte addresses increment from left to right. **Label the box with the variable name.** Label each element of an array (e.g., N[0]). Assume all alignment restrictions imposed by the hardware are obeyed, and the compiler does not add additional alignment restrictions. Note: `int` and `float` are 4 bytes, and `double` is 8 bytes.

```
char    c  = 'a';
char   *p  = &c;
char   N[] = "Now!";
int     x  = 5;
double  d  = 3.23;
double *e  = &d;
```

4000	c	s	s	s
4004	p	p	p	p
4008	N[0]	N[1]	N[2]	N[3]
4012	EOS	s	s	s
4016	x	x	x	x
4020	d	d	d	d
4024	d	d	d	d
4028	e	e	e	e
4032				
4036				

Part B (14 points) Assuming a 32-bit system, consider the following declaration:

```
int  A[][] = {{2, 6, 10, 14},
              {5, 15, 25, 35},
              {7, 11, 18, 22}};
```

B.1 Array `A` contains an element of value **25** (shown in bold). Write a single C statement that overwrites that element with the value **100**.

A[1][2] = 100 ;

B.2 Write the MIPS code implementation of the following assignment statement in the smallest number of instructions. A pointer to the array **A** is stored in **$1** and variables **`j`, `i`,** and **`y`** reside in **$2, $3,** and **$4**, respectively. Modify only registers $4 and $5.

```
int y = A[j][i];
```

Label	Instruction	Comment
	`sll $4, $2, 2`	`# j * 4`
	`add $4, $4, $3`	`# j * 4 + i`
	`sll $4, $4, 2`	`# scale by 4`
	`add $4, $4, $1`	`# add A base`
	`lw  $4, 0($4)`	`# y = A[j][i]`

Problem 2 (3 parts, 30 points) **Accessing structs, pointers, pre/post increment**

Consider the following C code fragment.

```
typedef struct {
   int    A;
   int    B;
   char   C[10];
} struct_t;

int foo() {
   struct_t S[2], *p;
   char *q;
   p = &S[1];
   q = &S[0].C[0];

   S[0].B = 42;

   //copies the string "yolo"
   // to the array S[0].C
   // beginning at C[0]
   S[0].C = "yolo";

   //This line implemented
   //in assembly for Part C
   p->B = ++(S[0].B);

   *(q++) ='n';
}
```

Foo's A.F.	Addr	Variable name	Value
	n+8:	Caller's RA	
	n+4:	Caller's FP (no inputs)	
Foo's FP:	n:	RV	
	n-4:	S[1].C[8-9]ss	
	n-8:	S[1].C[4-7]	
	n-12:	S[1].C[0-3]	
	n-16:	S[1].B	43
	n-20:	S[1].A	
	n-24:	S[0].C[8-9]ss	
	n-28:	S[0].C[4-7]	EOS
	n-32:	S[0].C[0-3]	~~y~~ nolo
	n-36:	S[0].B	~~42~~ 43
	n-40:	S[0].A	
	n-44:	p	n-20
	n-48:	q	~~n-32~~ n-31

Part A (15 points) Fill in the value of each expression after the code above is executed.

S[0].B	**43**
S[1].B	**43**
S[0].C[0]	**'n'**
S[0].C[4]	**EOS**
(--p)->C[0]	**'n'**

Part B (5 points) Assuming a 32-bit system, what is the total space allocated to `foo`'s local variables?

___48____ bytes

Part C (10 points) Write the assembly code for the line identified above. Hint: Remember they are local variables.

Label	Instruction	Comment
	lw $1, -36($30)	**# S[0].B**
	addi $1, $1, 1	**# S[0].B + 1**
	sw $1, -36($30)	**# S[0].B = S[0].B + 1 (++S[0].B)**
	lw $2, -44($30)	**# read p**
	sw $1, 4($2)	**# store S[0].B to p->B**

Problem 3 (2 parts, 40 points) **Activation Frames**

Part A (20 points) The function `Bar` (below left) calls function `Area` after completing `code block 1`. Write MIPS assembly code that properly calls `Area`. Include all instructions between `code block 1` and `code block 2`. Symbolically label all required stack entries and give their values if they are known.

```
typedef struct diamond_t {
  int    axis1;
  int    axis2;
} Diamond;

int Area(Diamond *K, int S[], int n){
  int a;
  int m = S[2];
  a = 2 * (K->axis1) * (K->axis2);
  S[1] = m + n;
  return a;
}

int Bar() {
  Diamond D = {5, 10};
  int     V[] = {4, 9, 16};
  int     x = 2;

  <code block 1>

  x = Area(&D, V, x);

  <code block 2>
}
```

Bar's FP	9620	XXX	XXX
	9616	D.axis2	10
	9612	D.axis1	5
	9608	V[2]	16
	9604	V[1]	9
	9600	V[0]	4
SP	9596	x	2
	9592	**RA**	**n/a**
	9588	**FP**	**9620**
	9584	**&D**	**9612**
	9580	**V**	**9600**
	9576	**X**	**2**
	9572	**RV**	**n/a**
	9568		

label	instruction	comment
	addi $29, $29, -24	# Allocate activation frame
	sw $31, 20($29)	# Preserve bookkeeping info
	sw $30, 16($29)	
	addi $1, $30, -8	# compute &D
	sw $1, 12($29)	# push it on stack
	addi $1, $30, -20	# compute V
	sw $1, 8($29)	# push it on stack
	lw $1, -24($30)	# read x
	sw $1, 4($29)	# push it on stack
	jal Area	# Call Area
	lw $31, 20($29)	# Restore bookkeeping info
	lw $30, 16($29)	
	lw $1, 0($29)	# Read return value
	sw $1, -24($30)	# Store return value in x
	addi $29, $29, 24	# Deallocate activation frame

Part B (20 points) Write MIPS code fragments to implement the subroutine `Area` by following the steps below. *Do not use absolute addresses in your code; instead, access variables relative to the frame pointer.* Assume no parameters are present in registers (i.e., access all parameters from `Update`'s activation frame). You may not need to use all the blank lines provided.

First, write code to properly set `Area`'s frame pointer and to allocate space for `Area`'s local variables and initialize them if necessary.

label	instruction	Comment
Area:	**addi $30, $29, 0**	**# set FP**
	addi $29, $29, -8	**# make room for 2 words**
	lw $1, 8($30)	**# read S**
	lw $2, 8($1)	**# read S[2]**
	sw $2, -8($30)	**# M = S[2]**

 # a = 2 * (K->axis1) * (K->axis2);

label	instruction	Comment
	lw $3, 12($30)	**# read K**
	lw $4, 0($3)	**# read K->axis1**
	lw $5, 4($3)	**# read K->axis2**
	mult $4, $5	**# multiply them**
	mflo $4	**# $4 gets product**
	sll $4, $4, 1	**# times 2**
	sw $4, -4($30)	**# write result to a**

 # S[1] = m + n;

label	instruction	Comment
	lw $3, -8($30)	**# read m**
	lw $9, 4($30)	**# read n**
	add $3, $3, $9	**# m + n**
	lw $1, 8($30)	**# read S**
	sw $3, 4($1)	**# S[1] = m+n**

 # return a; (store return value, deallocate locals, and return)

label	instruction	Comment
	sw $4, 0($30)	**# store a in RV slot**
	addi $29, $30, 0	**# deallocate locals**
	jr $31	**# return**