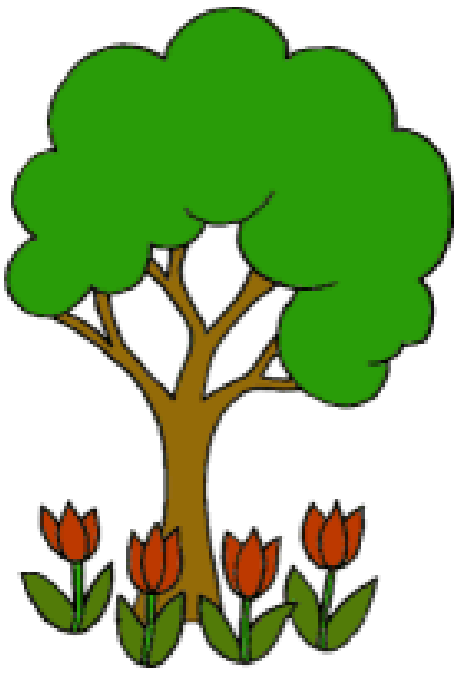
Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

Your Name (***please print***) _____

This exam will be conducted according to the Georgia Tech Honor Code. I pledge to neither give nor receive unauthorized assistance on this exam and to abide by all provisions of the Honor Code.

Signed _____

1	2	3	total
35	27	38	100



Problem 1 (2 parts, 35 points) **Storage Allocation and Pointers**

Part A (15 points) Assuming a **64-bit system with 64-bit memory interface and 64-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 memory. Label the box with the variable name. Label each element of an array (e.g., M[0]). Note that int and float are still 32-bits.

```
double G;
float  M[] = {1.44, 3.69, 4.64};
double *P = &G;
int    z = 4;
float  *Q = M;
double H;
```

4000				
4004				
4008				
4012				
4016				
4020				
4024				
4028				
4032				
4036				
4040				
4044				
4048				
4052				

Part B (20 points) For this part, assume a 32-bit system, such as MIPS-32 that we have worked with.

```
int X = 5;
int V = 7;
char S[] = "hello!";
double Z = 7.25;
double *P = &Z;
int *R = &X;
```

Question:	Answer:
How much space (in bytes) is allocated for R?	**bytes**
How much space (in bytes) is allocated for P?	**bytes**
What is printed by this statement? `printf("%d\n", *(R+1));`	
What is printed by this statement? `printf("%c\n", *(S+4));`	
What is printed by this statement? `printf("%f\n", *P);`	

Problem 2 (4 parts, 27 points) **Struct and Array Access**

Consider the following declarations.

```
int  A[][] = {{2, 6, 10, 14},
              {5, 15, 25, 35},
              {7, 11, 18, 22}};
typedef struct {
   int height;
   int base;
} triangle;
triangle T;
triangle *P = &T;
```

Part A (3 points) Array `A` contains an element of value **11** (shown in bold). Write a single C statement that overwrites that element with the value **100**.

_____.

Part B (3 points) Write a C statement that sets the **`base`** field of **`T`** to **16**, using the variable **`T`**.

_____.

Part C (3 points) Write a C statement that sets the **`height`** field of **`T`** to **20**, using the variable **`P`**.

_____.

Part D (18 points) Write the MIPS code implementation of the dynamically allocated array access below in the smallest number of instructions. A pointer to the array **`Icons`** (declared below) is stored in `$3`. Variables **`I`**, **`Y`**, **`X`**, and **`Cell`** reside in `$4`, `$5`, `$6`, and `$2` respectively. Modify only registers `$1` and `$2`. Assume a 32-bit operating system.

```
int    Icons[8][12][16];          /* array declaration */
Cell = Icons[I][Y][X];            /* implement this */
```

Label	Instruction	Comment

Problem 3 (2 parts, 38 points) **Activation Frames**

Consider the following C code fragment:

```
int Average(int [], int);

int TrapezoidArea(int H) {
   int         Bases[] = {3, 5};
   int         N = 2;
   int         R;

   R = H * Average(Bases, N);
   return(R);
}

int Average(int A[], int S) {
   int         x, y, avg;
   x = A[0];
   y = A[1];
   avg = (x+y)/S;
   return(avg);
}
```

Part A (18 points) Suppose `TrapezoidArea` has been called with input **`H=10`** so that the state of the stack is as shown below. Describe the state of the stack <u>just before</u> `Average` deallocates locals and returns to `TrapezoidArea`. Fill in the unshaded boxes to show `TrapezoidArea's(TA's)` and `Average's` activation frames. Include a symbolic description and the actual value (in decimal). For return addresses, show only the symbolic description; do not include a value. *Label the frame pointer and stack pointer*.

address	description	Value
9900	**RA of TA's caller**	
9896	**FP of TA's caller**	
9892	**H**	**10**
SP,TrapezoidArea's FP 9888	**RV**	
9884		
9880		
9876		
9872		
9868		
9864		
9860		
9856		
9852		
9848		
9844		
9840		
9836		
9832		
9828		

Part B (20 points) Write MIPS code fragments to implement the subroutine `Average` 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 `Average`'s activation frame). You may not need to use all the blank lines provided. (You may assume that intermediate values your code stores in registers stay in the registers from one part to the next unless your code overwrites them.)

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

label	instruction	Comment
`Average:`		

`# x = A[0];`

label	instruction	Comment

`# y = A[1];`

label	instruction	Comment

`# avg = (x+y)/S;`

label	instruction	Comment

`# return(avg);  (store return value, deallocate locals, and return)`

label	instruction	Comment

MIPS Instruction Set (core)

instruction	*example*	*meaning*
arithmetic		
add	add $1,$2,$3	$1 = $2 + $3
subtract	sub $1,$2,$3	$1 = $2 - $3
add immediate	addi $1,$2,100	$1 = $2 + 100
add unsigned	addu $1,$2,$3	$1 = $2 + $3
subtract unsigned	subu $1,$2,$3	$1 = $2 - $3
add immediate unsigned	addiu $1,$2,100	$1 = $2 + 100
set if less than	slt $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if less than immediate	slti $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
set if less than unsigned	sltu $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if < immediate unsigned	sltui $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
multiply	mult $2,$3	Hi, Lo = $2 * $3, 64-bit signed product
multiply unsigned	multu $2,$3	Hi, Lo = $2 * $3, 64-bit unsigned product
divide	div $2,$3	Lo = $2 / $3, Hi = $2 mod $3
divide unsigned	divu $2,$3	Lo = $2 / $3, Hi = $2 mod $3, unsigned
transfer		
move from Hi	mfhi $1	$1 = Hi
move from Lo	mflo $1	$1 = Lo
load upper immediate	lui $1,100	$1 = 100 x 2^{16}
logic		
and	and $1,$2,$3	$1 = $2 & $3
or	or $1,$2,$3	$1 = $2 | $3
and immediate	andi $1,$2,100	$1 = $2 & 100
or immediate	ori $1,$2,100	$1 = $2 | 100
nor	nor $1,$2,$3	$1 = not($2 | $3)
xor	xor $1, $2, $3	$1 = $2 ⊕ $3
xor immediate	xori $1, $2, 255	$1 = $2 ⊕ 255
shift		
shift left logical	sll $1,$2,5	$1 = $2 << 5 (logical)
shift left logical variable	sllv $1,$2,$3	$1 = $2 << $3 (logical), variable shift amt
shift right logical	srl $1,$2,5	$1 = $2 >> 5 (logical)
shift right logical variable	srlv $1,$2,$3	$1 = $2 >> $3 (logical), variable shift amt
shift right arithmetic	sra $1,$2,5	$1 = $2 >> 5 (arithmetic)
shift right arithmetic variable	srav $1,$2,$3	$1 = $2 >> $3 (arithmetic), variable shift amt
memory		
load word	lw $1, 1000($2)	$1 = memory [$2+1000]
store word	sw $1, 1000($2)	memory [$2+1000] = $1
load byte	lb $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
load byte unsigned	lbu $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
store byte	sb $1, 1002($2)	memory[$2+1002] = $1 (byte modified only)
branch		
branch if equal	beq $1,$2,100	if ($1 = $2), PC = PC + 4 + (100*4)
branch if not equal	bne $1,$2,100	if ($1 ≠ $2), PC = PC + 4 + (100*4)
jump		
jump	j 10000	PC = 10000*4
jump register	jr $31	PC = $31
jump and link	jal 10000	$31 = PC + 4; PC = 10000*4