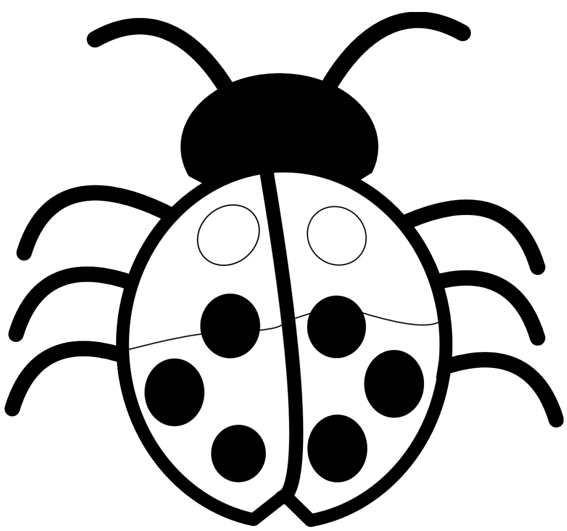
Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

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
30	30	40	100



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

Part A (16 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 the double word 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., M[0]). Note: int and float are 32-bits.

```
int     a  = 5;
int     b  = 25;
int    *c  = &b;
double  d  = 9.23;
double *e  = &d;
char   S[] = "Harry";
char   *f  = &(S[3]);
float   g  = 8.1;
double  z  = 17.25;
```

4000								
4008								
4016								
4024								
4032								
4040								
4048								
4056								
4064								

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

```
int A[32][16][8] = {...};
int *q = A;
```

B.1 Complete the assignment statement below using only `q` to assign to `x` the value of `A[20][10][2]`.

`int x = *(q + ________________________________);//an expression is ok`

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 **$3** and variables **`j, i,`** and **`y`** reside in **$4, $5,** and **$6**, respectively. Modify only registers $1 and $2.

`int y = A[0][j][i];`

Label	Instruction	Comment

Problem 2 (4 parts, 30 points) **Accessing Structs, Activation Frame Allocation**

Consider the following C code fragment.

```
typedef struct {
   int   age;
   float height;   // in meters
   float weight;   // in kilograms
} patient;

float Calc_and_Print_BMI(int A, float H, float W) {
   patient   Pat;
   float     BMI;

   _______________________________________________// part A

   _______________________________________________// part A

   _______________________________________________// part A

   BMI = Compute_BMI(&Pat);
   printf("BMI: %f\n", BMI);
   return(BMI);
}

/* BMI = weight divided by height squared (units: kg/m2). */
float Compute_BMI(patient *P) {

   float height_squared = ___________________________________________; // part B


   float BMI = ___________________________________ / height_squared;   //part B

   return(BMI);
}
```

Part A (6 points) Fill in the blanks in `Calc_and_Print_BMI` with statements that assign the inputs `A`, `H`, and `W` to the `age`, `height`, and `weight` fields of `Pat`, respectively.

Part B (6 points) Fill in the blanks in `Compute_BMI` with 1) a statement that squares the `height` of the `patient` pointed to by `P` and 2) a statement that computes the body mass index (`BMI`) of the `patient` pointed to by `P` by dividing the patient's `weight` by `height_squared`.

Part C (12 points) Assuming a 32-bit system, give the total number of bytes allocated for the activation frame of `Calc_and_Print_BMI` and the number of bytes for the activation frame of `Compute_BMI`.

Size of activation frame of `Calc_and_Print_BMI` (in bytes):	**bytes**
Size of activation frame of `Compute_BMI` (in bytes):	**bytes**

Part D (6 points) What does the following code fragment print?

Code:	**Answer:**

`char Name[] = "Fred";`
`char *p = Name;`
`while (*(++p))`
`   printf("%c\n", *p);`	

Problem 3 (1 part, 40 points) **Activation Frames**

The function `Bar` (below left) calls function `Foo` after completing `code block 1`. Write MIPS assembly code that properly calls `Foo`. 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 (below right).

```
int Foo(int Set[], int x, int *y, int z) {
   <body of Foo>
}

int Bar() {
   int          A[] = {40, 45, 50};
   int          B =  10;
   int          i;

   <code block 1>

   A[i] = Foo(A, A[2], &B, 0);

   <code block 2>

}
```

	address	label	value
Bar's FP	9620	XXX	XXX
	9616	A[2]	50
	9612	A[1]	45
	9608	A[0]	40
	9604	B	10
SP	9600	i	assigned in code block 1
	9596		
	9592		
	9588		
	9584		
	9580		
	9576		
	9572		

label	instruction	comment
		# Allocate activation frame
		# Preserve bookkeeping info
		# Push inputs
	jal Foo	# Call Foo
		# Restore bookkeeping info
		# Read return value
		# Store return value in A[i]: # load i, scale it, and add it # to base address of A to # compute where to store return # value, and store RV there.
		# Deallocate activation frame

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