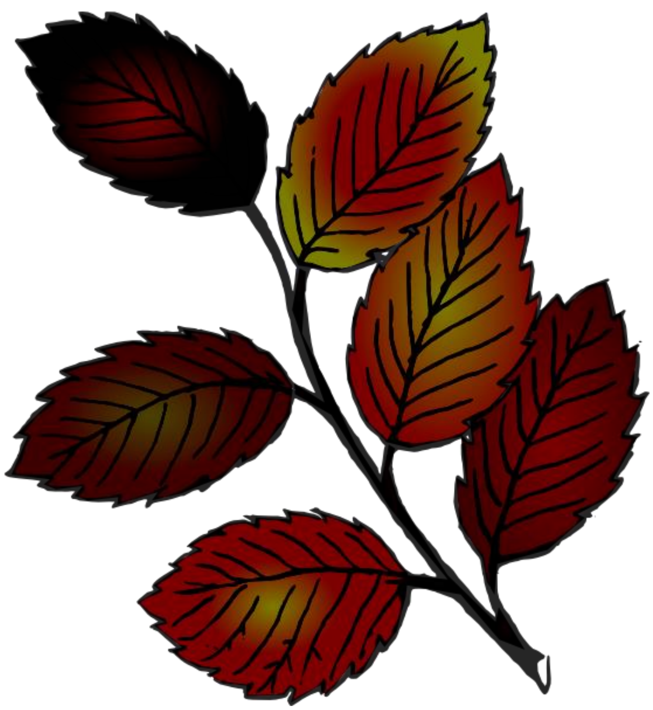
Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

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  = 66;
int     *b  = &a;
int    **c  = &b;
float    f  = 9.23;
float   *p  = &f;
char    S[] = "Sally";
char    *q  = &(S[0]);
float    g  = 8.26;
double   z  = 917.25;
```

4000								
4008								
4016								
4024								
4032								
4040								
4048								
4056								
4064								

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

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

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

```
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`, `k`, and `y` reside in **$4, $5,** and **$6**, respectively. Modify only registers **$6** and **$7**.

```
int y = A[k][j][4];
```

Label	Instruction	Comment

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

Consider the following C code fragment.

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

struct_t myStruct1= {10,20,0x2F};
struct_t myStruct2;
int j = 42;
struct_t * p = &myStruct1;
int *q = &myStruct1.A;

myStruct2.A = j;
(*p).B = 10;
p->C = 0x2A;
```

Part A (15 points) Assuming a **32-bit system with 32-bit memory interface and 32-bit addresses**, fill in the table with the values of the given expressions or U if they are unknown. Each expression should be evaluated independently only given the above code. Please assume variables are allocated beginning at address 1000.

Expression	value
`&myStruct2`	
`p->A`	
`*(q+1)`	
`myStruct1.C+1`	
`(p+1)->A`	

Part B (15 points) Consider the following function:

```
int foo(char *s,char *d){
       int cnt=0;
       while (*s){
          *d++ = *s++;
          cnt++;
      }
      *d = 0;
      return cnt;
}
```

B.1 What does the function do?
B.2 Describe two things that must be true for this function to execute correctly. Hint: (allocation and assumptions about the data).
B3. What is the size of foo's activation frame?

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

Consider the following C code fragment:

```
typedef struct {
   int Start;
   int End;
} trip_info_t;

int TripAdvisor() {
   int         odometer = 981005;
   int         Gallons[] = {16, 6};
   trip_info_t TI;
   int         rate;
   int         Update(trip_info_t, int [], int *);
   TI.Start    = 180;
   TI.End      = 420;
   rate = Update(TI, Gallons, &odometer);
   return(odometer);
}

int Update(trip_info_t Trip, int G[], int *OD) {
   int         miles, MPG;
   miles       = Trip.End - Trip.Start;
   MPG         = miles/G[1];
   *OD        += miles;
   return(MPG);
}
```

Part A (18 points) Suppose `TripAdvisor` has been called so that the state of the stack is as shown below. Describe the state of the stack <u>just before</u> `Update` deallocates locals and returns to `TripAdvisor`. Fill in the unshaded boxes to show `TripAdvisor's` and `Update's` activation frames. Include a symbolic description and the actual value (in decimal) if known. 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**	
SP, TripAdvisor's FP	9892	**RV**	
	9888		
	9884		
	9880		
	9876		
	9872		
	9868		
	9864		
	9860		
	9856		
	9852		
	9848		
	9844		
	9840		
FP: ________	9836		
SP: ________	9832		
	9828		

Part B (22 points) Write MIPS code fragments to implement the subroutine `Update` 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 `Update`'s frame pointer and to allocate space for `Update`'s local variables and initialize them if necessary.

label	instruction	Comment
`Update:`		

`# miles = Trip.End – Trip.Start;`

label	instruction	Comment

`# MPG = miles/G[1];`

label	instruction	Comment

`# *OD += miles;`

label	instruction	Comment

`# return(MPG);   (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