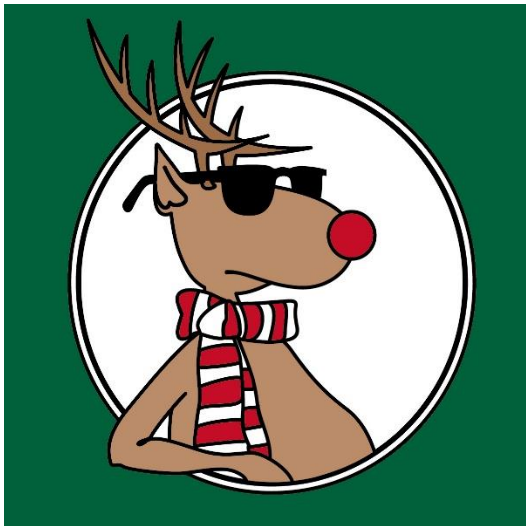
Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

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	4	5	6	total
20	40	20	25	30	40	175



Problem 1 (20 points) **Optimization**

Perform at least **five** standard compiler optimizations on the following C code fragment by writing the optimized version (in C) to the right. *Assume* ***cube, g,*** *and* ***h*** *are pure functions that each return an integer with no side effects to other data structures.*

```
int cube (int n) {
  return (n*n*n);}
int g (int k) { … }
int h (int i, int j) { … }
int slowcode(int a, int b) {
  int t = 100;
  int p = 1, s=0;
  do {                                  ⇨
      if (s)
        p += t*cube(p);
      else
        p += h(s+a*t, p+a*t);
      printf("t:%d, p:%d\n",t,p);
      t++;
  } while(t<g(p/256) + h(a, b));
  return h(p, t);
}
```

Briefly describe which standard compiler optimizations you applied *<u>and how they improve storage and/or execution efficiency in this code example</u>* (be specific; e.g., "replaces 2 MIPS operations with 1 operation on every loop iteration").

1.

2.

3.

4.

5.

Problem 2 (2 parts, 40 points) **MIPS and C Programming**

Part A (16 points) Given two arrays A and B, of 64 integers, write a C fragment that loops through the two arrays and computes the average difference of the corresponding elements (i.e., `A[i]-B[i]` for `i` from 0 to 63) and assigns the result to the variable `avgDiff`. The C fragment should also compute the minimum and maximum of the differences and assign them to the variables `minDiff` and `maxDiff`, respectively. ***For maximum credit, declare and initialize any necessary variables.*** ***NOTE****: A and B can be any 64-element integer arrays, not just the example given. Pay careful attention to how you initialize `minDiff` and `maxDiff` – the minimum difference could be greater than 0 and the maximum difference might be less than 0.*

```
int   A[64] = {-17, 2, 93, 9, ... -14, 7}; // given
int   B[64] = {-19, 5, 93, 7, ... -14, 8}; // given
int   avgDiff;
int   minDiff;
int   maxDiff;
```

Part B (24 points) Write MIPS code for the fragment in Part A. **Store the `avgDiff` computed in register $4, the `minDiff` in register $5, and `maxDiff` in register $6**. *For maximum credit use a minimum number of instructions.*

Label	Instruction	Comment
	`.data`	#
A:	`.word -17, 2, 93, 9, ... -14, 7`	# given
B:	`.word -19, 5, 93, 7, ... -14, 8`	# given
	`.text`	#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#
		#

Problem 3 (4 parts, 20 points) **Short Answer**

Part A (4 points) Write a **single** MIPS instruction that is equivalent to the original fragment. Assume *little endian* byte ordering.

Original:	Equivalent MIPS statement:
`lui  $4, 0xFF00`	
`lw   $3, 1000($0)`	
`and  $3, $3, $4`	
`srl  $3, $3, 24`	

Part B (4 points) Suppose the instruction "`jal Foo`" is at instruction memory address 2020 and `Foo` is a label of an instruction at memory address 4040. When this instruction is executed, what changes occur to the registers. List all registers that are changed (both general purpose and special purpose) and give their new values.

Register	Value

Part C (6 points) For each of the following, write a single MIPS instruction to implement the C fragment? Assume variables `A, B, C,` and `D` are of type `int` and are stored in registers `$1, $2, $3,` and `$4`.

`A = B & 7;`	
`C = D / 256;`	

Part D (6 points) Consider the MIPS code on the left which implements the array declaration and access on the right, where the variables **`Z`**, **`Y`**, **`X`**, and **`Value`** reside in `$4`, `$5`, `$6`, and `$7` respectively.

```
addi $1, $0, 48
mult $1, $4
mflo $1
sll $2, $5, 4
add $1, $1, $2
add $1, $1, $6
sll $1, $1, 2
sw $7, Array($1)
```

```
int Z, Y, X, Value;
…

int Array[_________][_________][_________];
…
Array[Z][Y][X] = Value;
```

What does this code reveal about the dimensions of Array? Fill in the blanks in the array declaration with the size of each dimension that can be determined from the code. If a dimension cannot be known from this code, put a "?" in its blank. Assume a 32-bit operating system.

Problem 4 (4 parts, 25 points) **Garbage Collection**

Below is a snapshot of heap storage. Values that are pointers are denoted with a "$". The heap pointer is **$6188**. The heap has been allocated contiguously beginning at **$6000**, with no gaps between objects.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
6000	8	6032	12	6064	16	6096	12	6128	8	6160	8
6004	33	6036	28	6068	4	6100	$6004	6132	60	6164	0
6008	40	6040	12	6072	55	6104	$6016	6136	75	6168	16
6012	12	6044	$6016	6076	8	6108	$6176	6140	16	6172	12
6016	0	6048	$6100	6080	6148	6112	12	6144	20	6176	$6132
6020	$6100	6052	$6116	6084	8	6116	$6032	6148	6046	6180	$6100
6024	$6088	6056	4	6088	4	6120	$6176	6152	80	6184	$6116
6028	8	6060	0	6092	40	6124	0	6156	26	6188	0

Part A (9 points) Suppose the stack holds a local variable whose value is the memory address **$6044**. No other registers or static variables currently hold heap memory addresses. List the addresses of all objects in the heap that are *not* garbage.

Addresses of Non-Garbage Objects: ______________________________

Part B (6 points) If a reference counting garbage collection strategy is being used, what would be the reference count of the objects at the following addresses?

Reference count of object at $6044 = ______________________________

Reference count of object at $6100 = ______________________________

Reference count of object at $6116 = ______________________________

Part C (6 points) If the local variable whose value is the address **$6044** is popped from the stack, which addresses from Part A will be reclaimed by each of the following strategies? If none, write "none."

Reference Counting:	
Mark and Sweep:	

Part D (4 points) What benefit does reference counting garbage collection provide that mark and sweep garbage collection strategy does not provide?

Benefit: ______________________________

Problem 5 (2 parts, 30 points) **Doubly Linked Lists**

Consider a doubly linked list that is implemented using the following struct definitions.

NOTE: These are the same as the structs used in Project 2-1, except the `data` field in `llnode_t` is of type `int` and the `DLinkedList` has no `size` field.

```
typedef struct llnode_t {
    int             data;
    struct llnode_t* previous;
    struct llnode_t* next;
}LLNode;

typedef struct dll_t {
    struct llnode_t* head;
    struct llnode_t* tail;
    struct llnode_t* current;
} DLinkedList;
```

Part A (12 points) Assume a **32-bit system** and consider the following `create_dlinkedlst` function:

```
DLinkedList* create_dlinkedlist() {
    DLinkedList* newList = (DLinkedList*)malloc(sizeof(DLinkedList));
    newList->head = NULL;
    newList->tail = NULL;
    newList->current = NULL;
    return newList;
}
```

A.1 What integer is passed to `malloc` when this function executes? ______________________________.

A.2 Which region of memory holds the variable `newList`? _____________________________________.

A.3 How much space (in bytes) is allocated for `newList` in this region of memory? ___________**bytes**.

A.4 How much space (in bytes) is allocated for the return value of `create_dlinkedlst`?

______________________________**bytes**.

Part B (18 points) Complete the C function **`Insert_Node_After`** that takes a pointer to an `LLNode` and inserts it **<u>after</u>** the `current` node in the doubly linked list pointed to by the input parameter `DLL`. Return 0 if the current node is `NULL`, otherwise return 1 (this code is already provided). Be sure to update the tail of `DLL` if `N` becomes the new tail. DLL's `current` field should not change.

```
int Insert_Node_After (LLNode *N, DLinkedList *DLL) {
  if(DLL->current == NULL){
    return 0;
  }else{






    return 1;
  }
}
```

Problem 6 (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		

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