ECE 220: Computer Systems and Programming

Fall 2020 – Final Exam

1. This is a closed-book, closed-notes exam
2. Absolutely no interaction between students is allowed
3. Illegible handwriting will be graded as incorrect
4. You must put your name and NetID on your submission page
5. **Use a separate page for each question**
6. Submission is only accepted through Gradescope

Question 1 (30 points): ______________

Question 2 (30 points): ______________

Question 3 (10 points): ______________

Question 4 (15 points): ______________

Question 5 (15 points): ______________

Total Score: ______________

Your answers should be in the following format. Each question should be on a separate page. Write out the entire line of code highlighted in yellow.

On each page, write down the following:

NetID

Q#
(1)
(2)
(3)
...

Problem 1 (30 points): Linked List

The below program, called “dotProd,” walks through two linked lists to perform a sparse vector dot product. Vectors are simply a 1-dimensional matrix, with only a single row in the case of our lists.

The vector header is defined as:
```
typedef struct vector{
   int length;
   node_t *head;
} vect_t;
```

Nodes in the list are defined as:
```
typedef struct node{
   double value;
   int col;
   struct node *next;
} node_t;
```

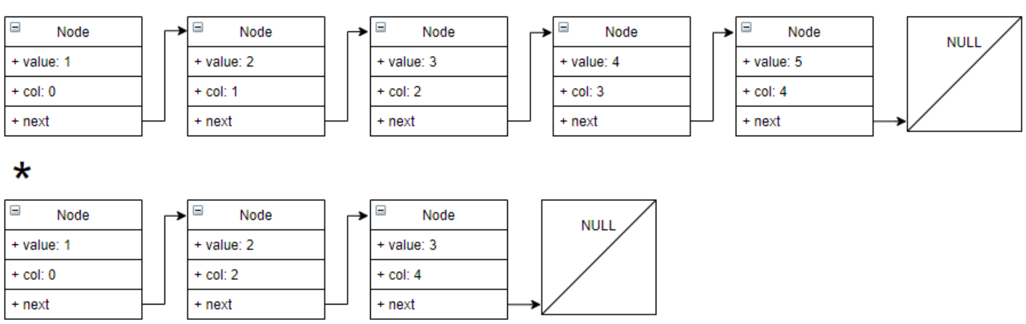
Where col is the column of the given entry. Assume that the list is in order, and that any column without an entry simply has a value of 0, as we assumed for sparse matrices.

The dot product is simply a series of multiply-accumulates. For every column, the values in both vectors are multiplied, and then added into the final result. This means that if a column does not exist in either list, then the product must be zero, and does not change the dot product at all.

Ex: [1 2 3 4 5] * [1 0 2 0 3]

[1 2 3 4 5]
*
[1 0 2 0 3]
= 1 + 0 + 6 + 0 + 15 = 22

In LL form:



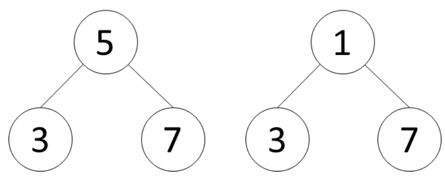
Fill in the blanks to complete the code

```
//sparse dot product
//multiply and accumulate both lists
//where missing elements are just zeroes
double dotProd(vect_t *v1, vect_t *v2){

   //return 0 if dot product cannot be performed
   //if either vector is nonexistent
   if( (1)___________________ )return 0;
   //if the lengths mismatch
   if( (2)___________________ )return 0;

   //start with no accumulated result
   double result = 0.0;

   node_t *h1 = v1->head;
   node_t *h2 = v2->head;

   //while both remaining lists are not NULL
   while( (3)______________ ){

      //if they match in col
      if( (4)_____________ ){
         //multiply and accumulate
         (5)___ += (6)_________________ ;
         h1 = h1->next;
         h2 = h2->next;
      }
      else if( (7)___________ ){ //if h1 precedes h2
         h1 = (8)_____ ; //move h1 forward
      }
      else{
         h2 = (9)_____ ; //move h2 forward
      }

  }
  return result;
}
```

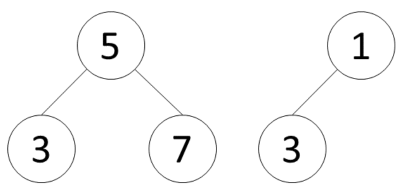
Problem 2 (30 points): Binary Search Tree

Part A.
The program, called "treematch," uses a post-order traversal of a tree to compare two trees and determine whether

- the trees are exactly the same, including their values
- the trees' structures are the same, but the values differ
- the trees' structures differ



Above is an example of two trees with different values, but the same structure. The root node for the left tree is 5, while the root node for the right tree is 1.



For this example, the structures differ, so the fact that the values differ does not matter. The root node of the left tree has two children, but the root node of the right tree has only a left child.

The returned value of treematch() is an enum, defined as:

```
typedef enum{
          SAME_ALL = 0
          SAME_STRUCT_DIFF_VALUE = 1
          DIFF_STRUCT = 2
}match_t;
```

SAME_ALL means that both the structure and the values are the same.
SAME_STRUCT_DIFF_VALUE means the structure is the same, but the values differ.
DIFF_STRUCT means that the structures differ (the values don't matter in this case).

The tree node is defined as:

```
typedef struct node{
        int value;
        struct node *parent, *left, *right;
}node_t;
```

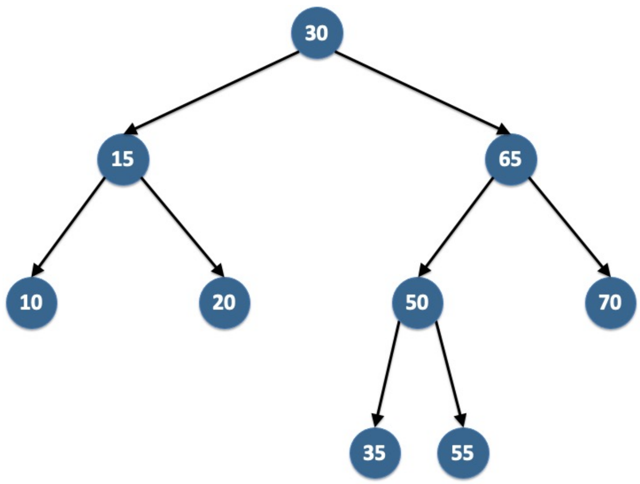
You may assume:

1. That the tree is formed correctly (e.g. that parent and child links are correct).

Fill in the blanks to complete the code

```
#include <stdio.h>
#include <stdlib.h>
#include "treematch.h"

match_t treematch(node_t *roota, node_t *rootb) {
    //if both null
    if ( (1)______________________ ) {
        //consider as having same value and structure
        return SAME_ALL;
    }
    //check if structure matches
    if ( (2)_______________________ ){
        //recursively check subtrees
        match_t mleft = treematch( (3)_____________________ );
        match_t mright = treematch( (4)____________________ );

        //if either subtree has a different structure
        if ( (5)__________________________________________ ) {
            return DIFF_STRUCT ;
        }
        //if either subtree has a different value
        else if ( (6)_____________________________________ ) {
            return SAME_STRUCT_DIFF_VALUE ;
        }
        //remaining: the subtrees are the same
        //check the values in the current nodes
        if ( (7) _________________________________ ) {
            return SAME_ALL;
        }
        else {
            return (8)_______________________ ;
        }
    }
    //remaining case: one pointer exists, the other doesn't
    return (9)_______________________ ;
}
```

Part B.
Given the binary search tree below, write out the **sequence of nodes visited during a pre-order traversal.**



Your Answer:

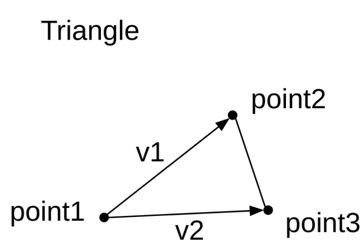
Part C.
Where should a new node with the value ‘62’ be inserted into the binary search tree shown in Part B? **Draw the entire tree for your answer.**

Your Answer:

Problem 3 (10 points): C++

In this problem, we are going to implement some geometric computation with the concept of OOP.
Point is a class contains 2 data members in public: x and y in double, that is the 2D coordinate.
There is a base class Polygon that contains a member function get_area, which is going to compute the area for the polygon.
The class Triangle, public derived from the class Polygon, that contains 3 data members (3 Point): point1, point2, point3 in private. Please complete the class Triangle for its constructor, copy constructor, access function, and the get_area function.
List of the functions you need to complete:
Default constructor: initialize point1 as (0, 0), point2 as (1, 0), point3 as (0,1).
Copy constructor: copy the points value from the given triangle.
Constructor: initialize the points' value with the given points, tp1, tp2, tp3.
Access function: let the public domain be able to access the private data: point1, point2, point3.
get_area function: compute the area of the triangle by computing the cross product of two vectors.

- Assign v1 as a vector from p1 to p2 by subtracting p1's coordinates from p2; v2 as a vector from p1 to p3 by subtracting p1's coordinates from p3.
- Compute the cross product of v1 and v2. (Hint: cross product: $(x_1, y_1) \times (x_2, y_2) = x_1 y_2 - x_2 y_1$)
- Get the absolute of this product and divide it by 2.



```
#include <iostream>
using namespace std;
class Point {
 private:
   double x, y;
 public:
   void set_val(const double & tx, const double & ty){
     x = tx;
     y = ty;
   }
   Point():x(0),y(0){};
   Point(const Point & p):x(p.x()), y(p.y()){};
   Point(const double tx, const double ty):x(tx), y(ty){};
   ~Point(){};
   const double x() const{ return x; }
   const double y() const{ return y; }
};

class Polygon{
 private:
 public:
   virtual double get_area() const = 0 ;
};

class Triangle: public Polygon {
 private:
   Point point1, point2, point3;
 public:
   const Point p1() const;
   const Point p2() const;
   const Point p3() const;
   Triangle();
   Triangle(const Triangle &t);
   Triangle(const Point & p1, const Point & p2, const Point & p3 );
   ~Triangle(){};
   double get_area() const;
};
Triangle::Triangle(){ // default ctor
 // initial p1 = (0,0), p2 = (1, 0), p3 = (0,1)
 (1)________________
 (2)________________
 (3)________________
}
Triangle::Triangle(const Triangle & t){ // copy ctor
 (4)________________
 (5)________________
 (6)________________
```

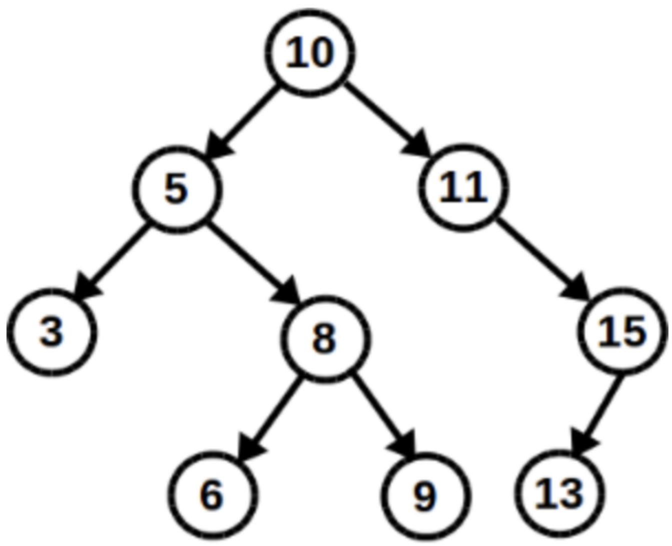
```
}
Triangle::Triangle(const Point & p1, const Point & p2, const Point & p3 ){ // ctor w/ given inital value
(7)________________
 (8)________________
 (9)________________
}
//Access function
const Point Triangle::p1() const{
 return (10)________________
}
const Point Triangle::p2() const{
 return (11)________________
}
const Point Triangle::p3() const{
 return (12)________________
}
// compute area function
double Triangle::get_area() const{
 Point v1( (13)________________________ ); // vector p1 --> p2
 Point v2( (14)________________________ ); // vector p1 --> p3
 double cross_product = (15)________________________
 if (cross_product < 0) { // compute the absolute value
   cross_product = -cross_product;
 }
 return 0.5*cross_product;
}

int main(){
 Polygon* ptr;

 Triangle tri0;
 ptr = & tri0;
 std::cout << "tri 0 area :"<<ptr->get_area()<<std::endl;

 Point p1(1,1);
 Point p2(4,4);
 Point p3(4,1);
 Triangle tri1(p1, p2, p3);
 ptr = &tri1;
 std::cout << "tri1 area :"<<ptr->get_area()<<std::endl;

 Triangle tri2(tri1);
 ptr = &tri2;
 std::cout << "tri2 area :"<<ptr->get_area()<<std::endl;

 return 0;
}
```

Problem 4 (15 points): C to LC-3 Conversion

In this problem, you will translate a recursive C function **SumPostOrder**, which traverses the binary tree in post-order and adds the value of each node in the tree and prints the current sum, to LC3.

You can assume each memory location stores 16 bits. Both integers and pointers are also 16 bits.

Example:



Input: root of binary tree **Output:** 80

C code:

```
typedef struct Node{
	int value;
	Node* left;
	Node* right;
}

int SumPostOrder(Node* node) {
	int sum = 0;
	/* Base case */
	if (node == NULL){
		return sum;
	}

	/* Recursive calls */
	sum += SumPostOrder((node->left);
	sum += SumPostOrder(node->right);
	sum += node->value;
	return sum;
}
```

Recalled that a function's activation record has the following format:

Local Variables
Caller's Frame Pointer
Return Address
Return Value
Arguments

Register map for this problem:

R0	NODE
R1	SUM
R2	TEMP
R5	FRAME POINTER
R6	STACK POINTER
R7	RETURN ADDRESS

Requirements:
You **MUST** follow the run-time stack convention presented in lectures.
You **MUST** finish your code within the number of provided lines.
You **MUST** use the register map as specified above. No other registers may be used.

```
SUM_POST_ORDER
; callee set-up
(1) _______________
(2) _______________
(3) _______________
(4) _______________
;
; initialize sum
(5) _______________
;
; check base case
(6) _______________          ; load NODE
(7) _______________          ; check NODE==NULL
;
; SumPostOrder(node->left)
(8) _______________          ;  load NODE->left to TEMP
(9) _______________          ; update stack pointer
(10) _______________         ; push argument
JSR SUM_POST_ORDER
(11) _______________         ; store return value in TEMP
(12) _______________         ; add TEMP to SUM
(13) _______________         ; stack teardown
;
; SumPostOrder(node->right)
; second recursive call omitted for simplicity
;
; add current node's value to sum
(14) _______________         ; load NODE->value to TEMP
(15) _______________         ; add NODE->value to SUM
;
DONE
(16) _______________         ; store return value
; callee tear-down
; omitted here for simplicity
RET
```

Problem 4 (15 points): Concepts

Part A & B

Recall in MP8 - 2048, we worked with a dynamically allocated game struct:

```
typedef int cell;
typedef struct{
	int rows;
	int cols;
	cell* cells;
	int score;
} game;
game* make_game(int rows, int cols){
	game* mygame = (game*)malloc(sizeof(game));
	mygame->cells = malloc(rows * cols * sizeof(cell));
	// rest of the function is omitted
}
```

Part A: Why is it necessary that the following function is executed after each game finishes?

```
void destroy_game(game* cur_game){
	free(cur_game->cells);
	free(cur_game);
	return NULL;
}
```

Your Answer (use no more than 20 words):

Part B: It is not possible to adjust the size of the **`cells`** array after it has been created. (True or False)

Your Answer: **True** **False**

Part C

Given the C++ classes below, answer the question in Part C. Note that the code is different from what was provided for MP12.

```
1 class Number{
2    public:
3         double magnitude, phase;
4         Number();
5         // more functions omitted
6         virtual Number operator + (const Number& arg) = 0;
7 };
8 class RealNumber : private Number{
9    public:
10        double real_component;
11        RealNumber();
12        RealNumber operator + (const RealNumber& arg);
13        // more functions omitted
14 };
```

In line 12, how is the argument **`arg`** passed to the operator overload function? Choose the correct answer from the following options.

1. By value **2. By pointer** **3. By reference**

Your Answer for Part C:

Table E.2 The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

LC-3 Instructions

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

Opcode	Encoding	Assembly	RTL
ADD	0001 | DR | SR1 | 0 | 00 | SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001 | DR | SR1 | 1 | imm5	ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101 | DR | SR1 | 0 | 00 | SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101 | DR | SR1 | 1 | imm5	AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc
BR	0000 | n | z | p | PCoffset9	BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)
JMP	1100 | 000 | BaseR | 000000	JMP BaseR	PC ← BaseR
JSR	0100 | 1 | PCoffset11	JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111 | 0000 | trapvect8	TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]
LD	0010 | DR | PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 | DR | PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 | DR | BaseR | offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 | DR | PCoffset9	LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001 | DR | SR | 111111	NOT DR, SR	DR ← NOT SR, Setcc
ST	0011 | SR | PCoffset9	ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011 | SR | PCoffset9	STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111 | SR | BaseR | offset6	STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR

End of ECE 220 Final Exam