ECE 220: Computer Systems and Programming

Spring 2021 – Final Exam

May 10, 2021

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
7. **Your CBTF time stamp is checked against your Gradescope submission time stamp. Any unusual activity will be reported to the college for infraction of academic integrity.**

Question 1 (24 points): ________________

Question 2 (30 points): A)______; B)______; C)______

Question 3 (16 points): ________________

Question 4 (12 points): ________________

Question 5 (18 points): A)______; B)______; C)______; D)______

Total Score: ________________

Write down your answers in the following format. Each question should be on a separate page. **Tag each question on your Gradescope submission.**

Name: NetID:

Q1 (**write down the # and entire line of code highlighted in yellow**)

Q2 (**write down the # and entire line of code highlighted in yellow**)
Part A; Part B; Part C

Q3 (**write down the # and entire line of code highlighted in yellow**)

Q4 (**write down the # and entire line of code highlighted in yellow**)

Q5
Part A; Part B; Part C; Part D

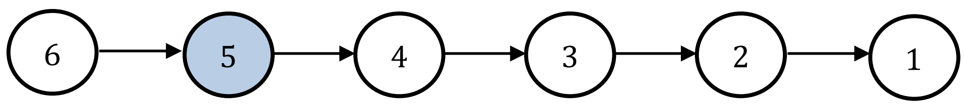
Problem 1 (24 points): Linked List

Given the head of a linked list, and an integer k. Write a C function that removes the kth node (the list is 1-indexed) from the head and insert the removed node as the (2k)-th node of the linked list. Assume k > 1 and the length of the linked list is larger than k. If the length of the linked list is smaller than 2k, please insert the removed node to the end of the linked list.

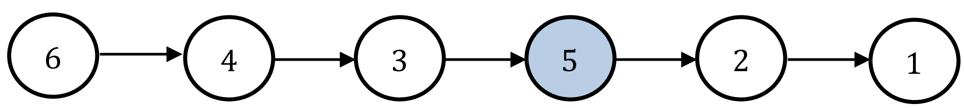
Example

Input:

[6,5,4,3,2,1], k=2



Output:
[6,4,3,5,2,1]



```c
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct ListNode node;

struct ListNode {
    int data;
    struct ListNode *next;
};

void KthNode(node* head, int k){
    node *cur = head;
    int i;
    /* traverse the list to find the (k-1)-th node */
    for(i=1; i < k - 1; i++){
        (1)_________________;
    }

    /* save the pointer that points to the (k)-th node */
    node *k_ptr = (2)__________________;

    /* update pointer to "remove" the (k)-th node */
    cur->next = (3)__________________;

    /* continue to traverse the list to find the right place for
       inserting the (k)-th node */
    i = 0;
    while(cur->next != NULL){
        cur = (4)__________________;
        i = (5)___________;
        if(i == k)
            (6)___________;
    }

    /* insert the (k)-th node after 'cur' node */
    node *tmp = cur->next;
    cur->next = (7)__________________;
    k_ptr->next = (8)__________________;
}
```

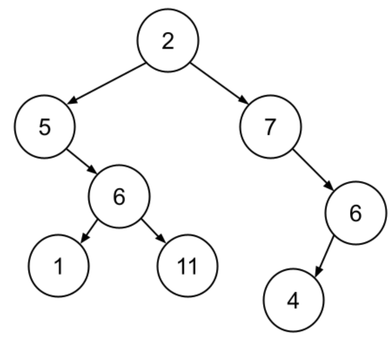
Problem 2 (30 points): Binary Tree

This problem has Parts A, B, and C.

Part A: Given a binary tree, write a recursive C function to find its largest node and return a pointer to it. Please note that the tree does not have duplicate nodes.

Example:

For the given binary tree, the function should return a pointer to the node contains the data '11'.

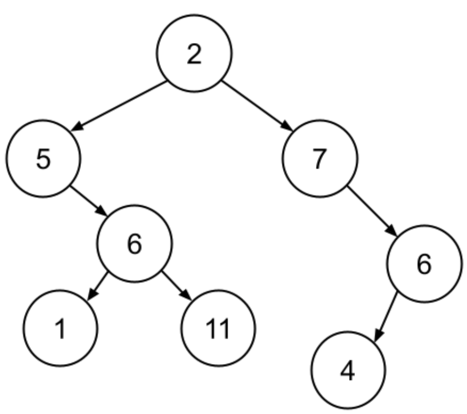


Part B: Write a C function to insert a new node as the left child of the largest node (found in Part A). If there already exists a left child, insert the new node as the right child. If the left and right child already exists, insert the new node as a left child anyway. Move the existing left subtree of the largest node to be the left child of the new node that is inserted.

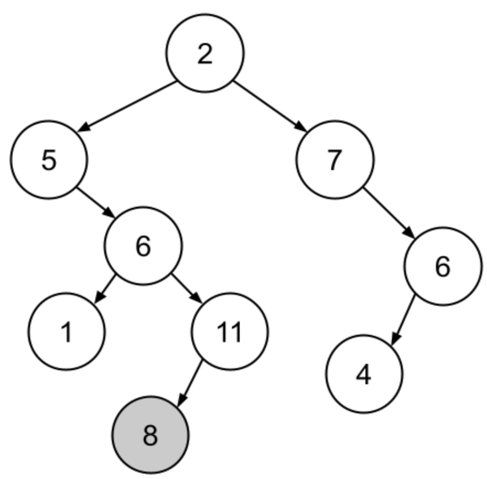
Examples:

In all the examples below, the maximum value found from part A is 11 and the new node to insert is 8.

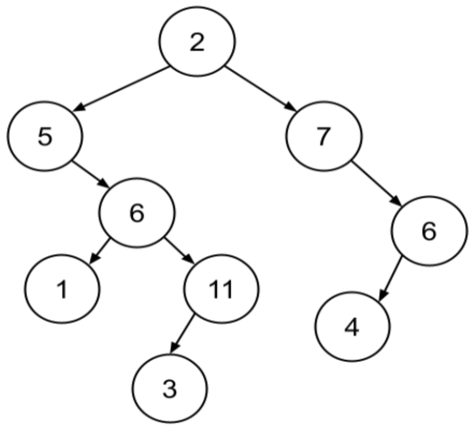
1. Initial Tree:



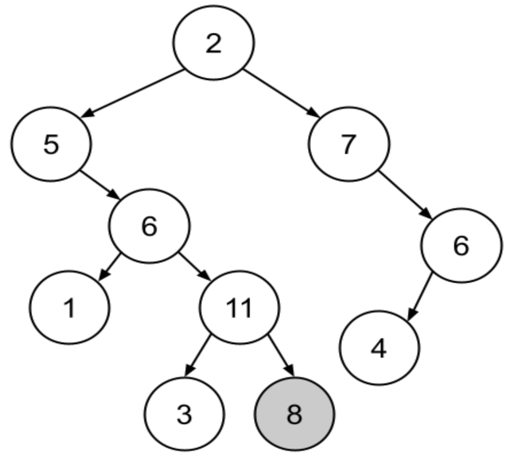
After Insertion:



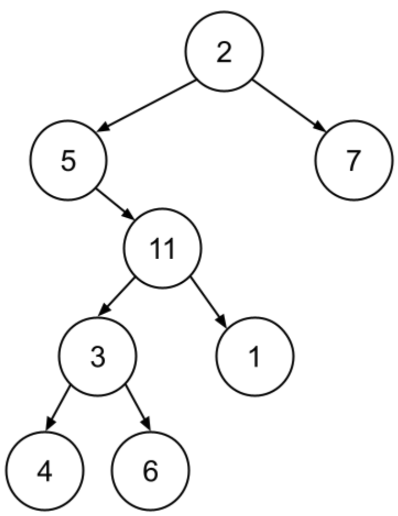
2. Initial Tree:



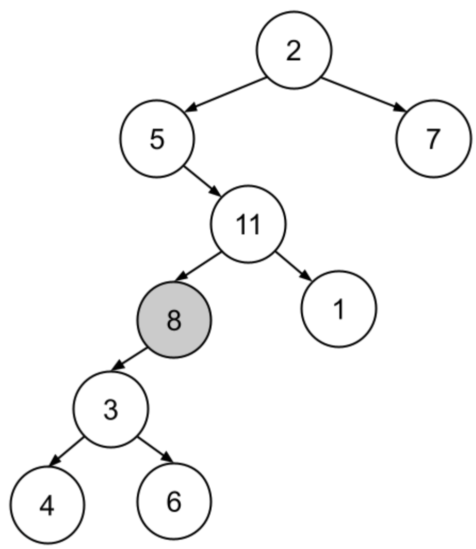
After Insertion:



3. Initial Tree:



After Insertion:



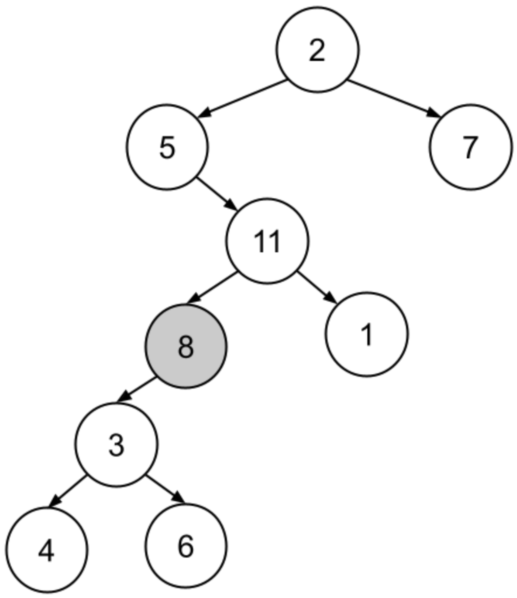
For Part A and Part B, fill in the blanks below.

```
/* tree node */
struct Node {
     int data;
     struct Node *left, *right;
};
```

```
/* Part A: find the maximum node */
struct Node* findMax(struct Node* root){
     /* Base case */
     if ((1)____________)
          return NULL;

     int lres=0;
     int rres=0;

     /* Return maximum of 3 values: 1) Root's data, 2) Max
     in Left Subtree, 3) Max in right subtree */

     int res = root->data;

     /* Recursion for the left subtree */
     struct Node* left_max = (2)____________;
     if (left_max!=NULL)
          lres=left_max->data;

     /* Recursion for the right subtree */
     struct Node* right_max = (3)____________;
     if (right_max!=NULL)
          rres=right_max->data;

     /* Compare lres, rres and res then return the ptr to
        the max node */
     /* if left subtree max node’s data is greater than
        that of right subtree’s max and root */
     if ((4)____________){
          return left_max;
     }
     /* if right subtree max node’s data greater than that
        of root */
     if ((5)____________){
          return right_max;
     }
     return root;
}
```

```
/* Part B: Inserts a new node */
/* Assume helper function 'struct Node* newnode(int val);' is
given and can be used to dynamically allocate a new node, set its
data to val, and left and right pointers to NULL */
void insertNode(struct Node* max_node, int val){
     if ((6)___________){
          printf("Cannot Insert New Node!");
          return;
     }

     /* Case 1: max_node doesn't have a left child */
     if ((7)___________){
          (8)___________;
          printf("Node inserted as left child.\n");
          return;
     }

     /* Case 2: max_node has a left child but no right child */
     if ((9)___________){
          (10)___________;
          printf("Node inserted as right child.\n");
          return;
     }

     /* Case 3: max_node has both left and right children */
     struct Node* temp = max_node->left;
     max_node->left = (11)___________;
     (12)____________________________;
     printf("Node inserted as left child. Inserted node now has a
     left subtree.\n");
}

/* Main Function */
int main(){
     /* creation of the tree is omitted*/

     /* Function call for part A. Assume root to be the root of
     the binary tree */
     struct Node* max_node=findMax(root);

     if (root!=NULL)
     printf("Maximum node is %d \n", max_node->data);

     /* Function call for part B. Assume 8 is to be inserted */
     insertNode(max_node, 8);
     return 0;
}
```

Part C: Suppose the binary tree after insertion looks like this:



What sequence would an in-order traversal (left, root, right) produce?

Your Answer: ___________________________________

Problem 3 (16 points): C to LC-3 Conversion

In the following, **alternateList(struct Node* head)** represents a C function that prints alternate nodes of the given Linked List, first from head to tail, and then from tail to head. If Linked List has an even number of nodes, then **alternateList()** skips the last node.

Examples:
If given a linked list: 1->2->3->4->5, it prints 1 3 5 5 3 1.
If given a linked list: 1->2->3->4->5->6, it prints 1 3 5 5 3 1.

Replicate the given C function **alternateList(struct Node* head)** using LC3. Code snippet is provided. You just need to fill in the blanks.

```
_______________________ C Code __________________________

/* A Linked List Node */
struct Node{
    int data;
    struct Node *next;
};

/* Recursive function - this function has no local variables */
void alternateList(struct Node* head){
  if(head == NULL)
    return;
  printf("%d  ", head->data);

  if(head->next != NULL )
    alternateList(head->next->next);
  printf("%d  ", head->data);
}
```

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

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

You **MUST** use and conform to the **LC-3 calling conventions** we have described **in class**. You may assume each data type will only occupy one memory location in LC-3.

```
; R6 is the stack pointer. R5 is the frame pointer.
; R7 contains the return address.

ALTERNATE_LIST
; complete callee set-up (push bookkeeping information and
; local variables) and comment on each line
(1)_____________________;reserve spaces (update stack pointer)
(2)_____________________;store return address
(3)_____________________;store caller's frame pointer
(4)_____________________;update frame pointer

; Load head to R0
(5)_____________________
; Check if head is NULL
(6)_____________________
(7)BR_ FINISH

; printing omitted

; Check if head->next is NULL
(8)_____________________ ; load head->next to R2
(9)BR__ PRINT

RECURSIVE
(10)_____________________ ; load head->next->next to R4
; complete caller set-up (push arguments)
(11)_____________________;store argument to run-time stack
(12)_____________________;update stack pointer
; make recursive call
(13)_____________________
; caller tear-down omitted

PRINT
; printing omitted

FINISH
; callee teardown
(14)_____________________;restore caller's frame pointer
(15)_____________________;restore return address
(16)_____________________;update stack pointer
RET
```

Problem 4 (12 points): C++

The following simple C++ program declares a class, creates some objects, prints them, and performs a "+" operation. Fill out the blanks in main.cpp for the program to match the provided output.

The expected output should look like the following:

```
This house has 5 rooms.
This house has 10 rooms.
This house has 15 rooms.
```

main.cpp

```
#include<iostream>
using namespace std;
class house {
    int roomCount;
    public :
    house(){}
    house(int r){

     ____________________________________________________________
    }

    int getRoomCount(void){

     ____________________________________________________________
    }

    void setRoomCount(int newRoomCount){

     ____________________________________________________________
    }

    void houseInfo() {
      cout << "This house has "<<roomCount<< " rooms."<<endl;
    }

    house operator + (const house & other){

     ____________________________________________________________

     ____________________________________________________________

     ____________________________________________________________

     }
};
```

```
int main() {
   house a(5);
   house b(10);
   a.houseInfo();
   b.houseInfo();
   house c = a+b;
   c.houseInfo();
   return 0;
}
```

Problem 5 (18 points): Concepts

Part A
Consider the following code:

```
char *name;
name = (char *)malloc(32*sizeof(char));
realloc(name, 16*sizeof(char));
free(name);
```

Assume memory allocations are successful and each char is 8-bit long, how many bytes of memory is deallocated when we call free?

Your Answer:____________________________ bytes

Part B
What would cause a memory leak?

(a) Run out of space in the run-time stack

(b) Run out of space in the heap

(c) Lost a pointer to a block of dynamically allocated memory

(d) Access memory location that is not allowed

Your Answer: A B C D

Part C
Determine whether the following statement is true or false:

A stack can be implemented by using an array or a linked list.

Your Answer: True False

Part D
Determine whether the following statement is true or false:

In C++, a virtual function table in each class stores its virtual functions.

Your Answer: True False

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

Instr.	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