name: Lucius Guo

lab time: 5:00 pm

CS 152
Exam 2
Winter 2020

11:30am class

Short Answer	/40
Linked List Imp	/30
Tree Implementation	/ 30
Total	/100

1. No one may leave for any reason and come back to work on his/her test. If you need to go to the restroom, go now.

2. You may have 1 sheet of notes, hand-written, double-sided

3. Your backpack must be zipped up.

4. You may not use any electronic devices of any kind. Make sure your cell phone is turned off so it does not ring during the test.

5. You may not wear hats, hoods, sunglasses, nor do anything else that would obscure your eyes.

6. You may not sit near your partner if you are partnered in programming or lab (within 3 seats in any direction)

8. We will not answer questions during the exam.

9. Do not hold your test up. It must stay on your desk.

10. If you seem to be looking around, you will be moved to the front to allow you to look around without having other students' test in your field of vision.

1. Short Answer: (you may use string library functions)

enum position_tag { UC_EMPLOYEE, UC_STUDENT };
enum division_tag { UC_PSD, UC_BSD, UC_SSD, UC_HD, UC_PME };
typedef unsigned int uint;

```
typedef struct {
        char name[50]; 50
        uint year_hired; 8
        enum division_tag division; 5
        uint salary; 8          101
        char title[30]; 30
} ucemployee;
```

```
typedef struct {
enum division_tag division; 5
char name[50]; 50
uint year_entered; 8
char major[50]; 50
uint UCID; 8          121
} ucstudent;
```

```
union _ucnode {
        ucemployee emp;
        ucstudent stu;
};
typedef struct {
        enum position_tag position; 2
        union _ucnode data;
} ucmember;
```

a. (5) If a char and enumerated type are 1 byte each, and a uint is 8 bytes, how much space is allocated for a single ucmember struct? Show your work.

The uc member struct has an enum which is 2bytes big. We always store enough space for the bigger union which is ucstudent (5+50 + 8+50 +8 =121). So a single ucmember struct will be alloated
121+2 = $\boxed{\text{123 bytes}}$

b. (12) Given the declaration below, write three lines of code.
1) set the tag of mem so that it is designated as an employee, not a student
2) copy the string "Associate Professor" into the title
3) set the division to be UC_PSD

`ucmember mem;`

```
mem.position = UC_EMPLOYEE;                    Howdy
mem.data.title= "Associate Professor";
mem.data.division = UC_PSD;
```

c. (10) Draw the end state of string1 and string2 after the following commands:

```
char string1[50], string2[50];
strcpy(string1, "Howdy");
strcpy(string1+20, "Hi");
strcpy(string2, "Whoop");
strcat(string2, string1);
```

Final state

string 1: 'H' 'o' 'w' 'd' 'y' ... 'H' 'i' '\0'   20 spaces in the array

string 2: same as String 1;

d. (13) Write a generic compare function that compares two ucemployee structs by name.

```
int compare_ucemployee(void *v1, void *v2)
{
  ucemployee *e1 = (ucemployee *)v1;

  ucemployee *e2= (ucemployee *)v2;

int result = strcmp( e1->name, e2->name) ;

 if(result == 0)
    return 0;
 if(result < 0)
    return -1;
 else
    return 1;

}
#inclde <string.h>
```

```
int strcmp(const char *x, const char *y)
{
  while(*x)
  {
    if( *x != *y)
      break
    x++;
    y++;
  }
  return *(const unsigned char *) x -
         *(const unsigned char * )y;
}
```

2. Linked List Implementation

(30) Write a function that inserts the val at the index specified, where the head is index 0. Do not remove or replace anything in the list – instead, place val in between the current index-1 and index node. If the list has fewer the index-1 items, do not insert into the list. Return 1 if the insertion was successful and 0 if the list was too short. Use the following list definitions:

```
typedef struct _node node;
struct _node {
        void *item;
        node *next;
};
```

```
typedef struct _llist {
        node *head;
        node *tail;
} llist;
```

```
node * newNode (void* val)
{
  node *nPtr;
  nPtr = (node *) malloc (sizeof(node));
  nPtr-> item= val;
  nPtr-> next = NULL;
  return nPtr;
}
```

```
int insert ( llist *lst, void *val, int index )
{  //when list is empty
   if (lst!=NULL && index ==0)
   {  node *new = new Node (val);
      new ->next = lst->head;
      lst->head= new;    return1;
   }
  node *holder = lst -> head;

  int i;

  for(i=0 ; i < index ; i++)
  {
   holder = holder ->next;
  }

  node *holder 2 = lst-> head;

  for(i=0 ; i <= index ; i++);
  {
    holder 2 = holder 2->next
  }
  node * new = new Node (val);

  holder -> next = new;

  new -> next = holder 2;

  return 1;

}
```

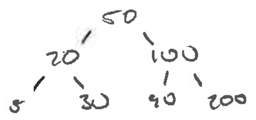
```
int length =0;
node *holder 1 = lst -> head;
while (holder 1 != lst -> tail)
{     length ++
      holder = holder -> next;
}
if ( length < index -1)
   return 0;
```

3. Tree Implementation

(30) Write an ***efficient*** function (fewest calls to comp) that prints out all **items in a BST** that are ***greater than*** the second parameter. comp returns -1 for less than, 0 for equal, 1 for greater than. You may continue your code onto the next page. Assume the following definitions:

```
typedef struct _bnode bnode;
struct _bnode {
    void  *item;
    bnode *lsub;
    bnode *rsub;
};
```

```
typedef struct {
    int (*compare)(void*, void*);
    void (*print_item)(void *);
    bnode *root;
} bst;
```



```
void bst_print_greater_than ( bst *tree,  void *low)
{   // if no tree, do nothing
    if (tree == NULL)
            return;
    else
            item_print(tree->root, low, tree->compare, tree->print_item);
}
                                           50
void item_print ( bnode *root,  void *low, int (*comp)(void *, void *), void (*print)(void *)  )
{
   if(root==NULL)
      return;
   if(comp (low, root->item >= 0)
   {
      item_print ( root->right, low, comp, print);
      print(root);;
      exit;
   }
   else
   {
      item_print (root -> left, low, comp, print);
   }
)
```

item print (root , 100, comp, print)