ECE312: Software Design and Implementation Final Exam
Dr. Pedro Santacruz April 27, 2022

Name: **ID:**

Final Exam

Instructions: This exam is **closed notes**, but you are allowed to use **two sheets (two sides)** as reference sheets. You are not allowed to have any kind of communication with anyone else (except the instructor) during the exam.

Write down your process for solving questions and intermediate answers that **may** earn you partial credit. **Simplify all answers as much as possible**.

There is scratch paper at the end. If your actual answer is on the scratch paper, state that fact clearly and legibly in the space below the question. When code is required, write C/C++ code, as specified. You may break problems up into smaller methods. (In other words you may add helper methods wherever you like.) You may not use functions that are not in the standard C/C++ library, except for the ones you write, of course.

Assume 32-bit machines, so pointers and `int`s are 32-bits wide. Style is not evaluated when grading, but neatness and legibility are required. Use indentation to help the graders read your code. The proctors will not answer most questions. Write down any assumptions that you need to make to resolve your doubts.

You have **120 minutes** to complete the exam. The maximum possible score is 100.

Here are some function signatures etc:

```
void* malloc (size_t size);

void free (void* ptr);

void* realloc (void* ptr, size_t size);

char* strcpy ( char* destination, const char* source );

int strcmp(const char *str1, const char *str2); //returns 0 if equal, else non-
    zero

size_t strlen(const char *str) //returns the size of a string

double abs (double x); // absolute value

printf -- %d (int), %u (unsigned int), %g (double/float), %s (string)

myStructType* myStruct; //myStruct->xx is the same as (*myStruct).xx

//scanf usage example:
//   int i;
//   scanf("%d", &i);  \\stores read value on i
```

Good luck!

Here are some function signatures for the string class, map class, etc.

- `string` class overloads most of the operators that one may expect, including (==. !=, ¡)
- default constructor: `string();`
- copy constructor: `string(const string &str);`
- constructor from c-string: `string(const char* s);`
- `char& operator[] (size_t pos);`
- `string& operator= (const string &str);`
- `string& operator= (const char* s);`

General function in std.

```
int std::stoi (const string& str, 0, 10);
//convert string to int, if string is a decimal integer such as "14"
```

Map class functions

```
map<K, V>(); // default constructor, where K is the key type and V is the value type.
std::map& operator[] (const key_type& k);
```

- If `k` matches the key of an element in the container, the function returns a reference to its mapped value.
- If `k` does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value

```
map<K, V>::iterator <iterator_name> = <map_object>.find(<k>)
```

- Searches `map_object` for an element with a key equivalent to `k` and returns an iterator to it if found, otherwise it returns an iterator to `map_object` called `map_object::end`. You can use `==` to compare the iterator returned by find to `map_object.end()`.

How to use an iterator:

```
map<int, string> m;
for(map<int,string>::iterator it = m.begin(); it != m.end(); ++it) {
    cout << it->first << endl;
    cout << it->second << endl;
}
```

Vector class functions

`operator[]` Access element (public member function)
`size` Return size (public member function)
`push_back` Add element at the end (public member function)
`pop_back` Delete last element (public member function)

Problem 1: Code Trace [20 points]

Assume `stdio.h` or `<iostream>` has been imported. Showing your work might get you partial credit if the answer itself is wrong. There are no compile or runtime errors. If the answer is blank, write ”Nothing”, without quotes.

a) [**5 points**] What is the output of the following program:

```cpp
#include <iostream>
using namespace std;

int funcA(int x) {
    x--;
    return x;
}

int funcB(int& x) {
    x++;
    return x;
}
int main() {
    int a = 4;
    int temp;
    funcA(a);
    funcB(a);
    cout << a << endl;
    temp = funcB(a);
    cout << temp << " " << a << endl;
    temp = funcA(a);
    cout << temp << " " << a << endl;
    return 0;
}
```

Answer:
5
6 6
5 6

b) [**5 points**] What is the output of the following program:

```
int main(){
    int a[4]={2,3,1,4};
    int i,j,t;
    i=0;
    j=3;
    while(i<j){
        t=*(a+i);
        *(a+i)=*(a+j);
        *(a+j)=t;
        i++;
        j--;
    }
    for(i=0;i<4;i++){
        printf("%d ",*(a+i));
    }
}
```

Answer: 4 1 3 2

c) [**5 points**] Give the Big-O worst-case tight-bound complexity. Also give a short explanation (1-2 sentences).

```
void bigo(int n) {
  int k = 0;
  while (k < n) {
    k = k + 1;
    if (k == n / 2) {
      k = 0;
      n = n - 2;
    }
  }
}
```

Answer: $O(n^2)$

Explanation: The total number of executions of the loop is $\frac{n}{2}+(\frac{n}{2}-1)+(\frac{n}{2}-2)+...+1$ **which is equal to** $\frac{(n/2)(n/2+1)}{2}$ **which is** $O(n^2)$**.**

d) [**5 points**] What is the output of the following program:

```
#include <map>
using namespace std;

int main(){
    map<int, int> numbers;
    numbers[1] = 10;
    numbers[2] = 100;
    numbers[3] = 1000;
    numbers[5] = 100000;

    map<int, int>::iterator myIter;
    int howMany = 0 ;
    for(myIter = numbers.begin(); myIter != numbers.end(); myIter++){
        if (myIter->first%10 == 0){
            howMany++;
        }
    }
    cout<< howMany << " " << numbers.size() << endl;
}
```

Answer: 0 4

Problem 2: Classes and Structures [30 points]
We are planning to implement a data structure that will hold non-negative integers using a hash table and acts like a set. Our data structure will consists of two classes `Set` and `Node`. The index of the location of a node with value `val` in the hash table is simply `val%k`, where `k` is the number of buckets in the hash table. Assume separate chaining using linked lists at each index, as we did in class. You are not allowed to use the `std::list` library.

Assume you start with the given code:

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int val;
    Node* next;

    Node(){
      val = 0;
      next = NULL;
    }

    Node(int val, Node* prevHead){
      this->val = val;
      next = prevHead;
    }
};

class Set {
public:
  int k; // number of buckets
  Node** buckets;

  Set(){
    k = 0;
    buckets = NULL;
  }

  Set(int k); // Part a: write this
  ~Set();  //Part b:  write this
  Set(const Set& other); //Assume this is written for you

  void addToSet(int val); //Part c: write this
  int findMax();//Part d: write this

};
```

a) [**5 points**] Write the constructor `Set(int k)` for `Set`. Your function should allocate space in the heap for `k` buckets (i.e. `k` Node pointers, one for each bucket, since each bucket will be a linked list), initialize each of the pointers to `NULL`, and initialize all other fields in `Set`.

```
//Solution

Set::Set(int k) {
    this->k = k;
    buckets = new Node*[k];
    for(int i=0; i<k; i++){
        buckets[i] = NULL;
    }
}
```

b) [5 points] Write the destructor ∼`Set`. The destructor must ensure that no memory is leaked. Every Node in the data structure must be freed. Keep in mind each bucket is a Linked List.

```
//Solution

Set:~Set(){
    for(int i=0; i<k; i++){
        Node* prev = NULL;
        Node* curr = buckets[i];

        while(curr!=NULL){
            prev = curr;
            curr = curr->next;
            delete prev;
        }
    }
}
```

c) [10 points] Write the function `void Set::addToSet(int val)`. This function must add `val` to the set by creating a Node with the appropriate fields. It must add it to the correct bucket, remember the index of the location of a Node with value `val` in the hash table is simply index `val%k`. Since we are implementing a Set, if a value is already in the set, we **do not add a duplicate**. You are allowed to add the new value to the beginning of the linked list in the correct bucket. You do not have to worry about having to resize the table.

```
//Solution

void Set::addToSet(int val) {
    int index = val%k;

    Node* prev = NULL;
    Node* curr = buckets[index];

    while (curr!=NULL){
        if (curr->val == val){
            return;
        }
        prev = curr;
        curr = curr->next;
    }

    Node* toAdd = new Node(val, buckets[index]);
    buckets[index] = toAdd;
}
```

d) [10 points] Implement the `int Set::findMax()` function. It should return the largest integer in the set.

```
//Solution

int Set::findMax() {
    int max = 0;

    for (int i=0; i<k; i++){
        Node* prev = NULL;
        Node* curr = buckets[i];

        while(curr!=NULL){
            if (curr->val > max){
                max = curr->val;
            }
            prev = curr;
            curr = curr->next;
        }
    }
    return max;
}
```

Problem 3: Recursion [20 points]

Rules: do not use loops, global or static variables, malloc, free, new, delete, or any other function. Do not use helper functions.

a) [**10 points**] Write a function to calculate find the sum of the digits that make up an integer. For example, give the number `101`, the result would be `2` since $1 + 0 + 1 = 2$.

```
//Solution

int sumItUp(int x){
    if (x == 0)
        return 0;
    else {
        int sum;
        int last = x % 10;
        sum = last + sumItUp(x / 10);
        return sum;
    }
}
```

b) [**10 points**] Given a binary tree using the implementation shown below, write a recursive function that counts the number of leaves in the tree. You can assume that `overallRoot` points to a properly allocated binary tree. A tree made up of just one node has 1 leaf.

```
struct Node {
  int val;
  Node* left;
  Node* right;
};

struct BTree{
private:
  Node* overallRoot;
  int countLeaves(Node*);  //Implement this

public:
  int countLeaves(){ //This is the public function that will be called by the user
    countLeaves(overallRoot);
  }
};
```

```
//SOLUTION

int BTree::countLeaves(Node* root){
    if (root==NULL)
        return 0;
    if (root->left==NULL && root->right == NULL){
        return 1;
    }

    return countLeaves(root->left)+ countLeaves(root->right);
}
```

Problem 4: Trees [20 points]
Consider an expression tree with the following implementation:

```

struct ExpNode {
  string data;
  ExpNode* left;
  ExpNode* right;

  /* other stuff that is less relevant to the question */

  };

class ExpTree {
private:
  ExpNode* overallRoot;

  void printInOrder(ExpNode* root){
    if (root == NULL)
      return;

      printInOrder(root->left);
      std::cout << root->data << " ";
      printInOrder(root->right);
  };

  void printPreOrder(ExpNode* root); //Part a:  write this

public:
  void printInOrder(){
    printInOrder(overallRoot);
  };

  void printPreOrder(){
    printPreOrder(overallRoot);
  };

/* other stuff that is less relevant to the question */
};
```

a) [**5 points**] Write the following function to give the expression in prefix notation. You should not print parenthesis, as prefix notation order of operations is unambiguous.

```
//SOLUTION

void ExpTree::printPreOrder(ExpNode* root) {
    if (root == NULL)
        return;

    std::cout << root->data << " ";
    printPreOrder(root->left);
    printPreOrder(root->right);
}

```

b) [**5 points**] Draw and evaluate an expression tree for the following prefix notation expression:
`+ - * / 16 8 4 2 1`

Tree:

Result: 7

c) [**5 points**] Draw and evaluate an expression tree for the following prefix notation expression:
`- 1 - 1 - 1 - 1 1`

Tree:

Result: 1

d) [**5 points**] Draw and evaluate an expression tree for the following prefix notation expression:

```
- - 1 - - 1 1 1 1
```

Tree:

Result: 1

Problem 5: General Programming [10 points]
Write a program in C++ that takes a vector of `int`s and returns the most frequent `int`. If there are multiple answers, return one of them. The input vector is guaranteed to have at least one element. An $O(N)$ solution guarantees full credit. The smallest `int` that could be present in the vector is `MIN_INT`, and the largest `MAX_INT`.

Sample input: [1, 4, -8, -8, 10, 10, -8] Returns: -8.

Hint: In the front of the exam, you can see information about the map class.

```cpp
//SOLUTION

int most_frequent(const vector<int>& list) {
  map<int, int> counts;
  int freq_num = list[0];
  int maxCount = 1;
  for(int i=0; i<list.size(); i++) {
    counts[list[i]]++;
    if (counts[list[i]] > maxCount) {
      freq_num = list[i];
      maxCount = counts[list[i]];
    }
  }
  return freq_num;
}
```

Scratch Paper

Scratch Paper