Assignment 2

Due 23:00, Saturday, February 10, 2018

CS ID 1: **x0k1b**

CS ID 2: **t1v0b**

Section (circle one):

Monday	L2D 10:00–12:00	L2S 12:00–14:00	L2E 14:00–16:00
	L2U 18:00–20:00		
Tuesday	L2Q 09:00–11:00	L2G 11:00–13:00	L2A 13:00–15:00
	L2M 15:00–17:00	L2R 17:00–19:00	
Wednesday	t1v0b, L2F 10:00–12:00	L2B 14:00-16:00	L2P 17:00–19:00
Thursday	L2L 09:00–11:00	L2K 14:00–16:00	L2C 16:00–18:00
	L2T 18:00–20:00		
Friday	L2N 10:00–12:00	L2H 12:00–14:00	x0k1b, L2J 14:00–16:00

Each individual or pair will be submit one completed copy of this homework onto gradescope. You may either print it and write your solutions in the space provided, or complete the assignment in latex. Show your work where necessary for full credit. If you require additional space, please indicate in the question space that you are writing on the last blank page, and also indicate on the blank page which question the work solves.

You must upload the completed document, including this page and the last page, to GradeScope, using your 4- or 5-character CS ID.

1. **[Miscellaneous Choices! – 36 points].**

   In each of the multiple choice problems below, bubble the best answer. In a few cases, more than one response is correct. Mark them all! You are welcome to explore some of the coding problems using a compiler, but be careful–sometimes the system gives you an unreasonable, reasonable answer. A good way to solve these problems is to speculate on a response *without* using the compiler, and then check to see if the code behaves as you predicted.

MISC1 (1.5pts)

```
1 int main(void) {
2         int *array = new int[4];
3         for (int i = 0; i < 4; i++){
4               array[i] = rand() % 10;
5         }
6         delete array;
7         return 0;
8 }
```

What is the problem with this code?

○ `array` should be declared as `int[]`.
○ We should declare `array` outside of `main`.
○ `array` is missing a constructor and a destructor.
○ A line of the code is leading to undefined behavior.
○ This code has a compiler error on line 6.
○ This code will seg-fault on line 4.
● None of the other answers is true.

MISC2 (1.5pts)

```
1
2  class Coffee{
3    public:
4      bool awesome;
5      void setName(int);
6      char *getName() { return name; }
7      Coffee() : awesome(true), name(new char[100]) {}
8      ~Coffee() { delete[] name; }
9    private:
10     char *name;
11 };
12
13 void Coffee::setName(int type) {
14     // given type, set name to something meaningful
15 }
16
17 int main(void) {
18     Coffee latte;
19     latte.setName(1);
20     if (latte.awesome) {
21         Coffee caffelatte(latte);
22         cout << caffelatte.getName() << endl;
23         caffelatte.setName(2);
24     }
25     cout << latte.getName() << endl;
26     return 0;
27 }
```

Identify all the problems in this code.
○ When `setName` is declared the parameter name is omitted.
○ The `name` member in `caffelatte` is incorrectly initialized.
○ The `getName` function returns a dangling pointer.
○ The instance `latte` is not initialized.
○ None of the above are correct.

MISC3 (1.5pts)

What error(s) might be caused by the code above?
● Double free error.
○ Memory leak.
○ C++ compilation errors.
○ Segmentation fault.
○ None of the other answers is true.

MISC4 (1.5pts)

How would you fix the error(s)?
○ Explicitly implement a copy constructor.
○ Declare `caffelatte` outside the `if` statement.
○ Declare `name` as a public member in `Coffee`.
○ Define `setName` inside `Coffee`'s class member list (with the parameter name!)
○ None of the other answers is true.

MISC5 (1.5pts)

```
1  class LegoMov {
2    public:
3      bool *evIsAwes;
4      void setEvIsAwes(bool b) { *evIsAwes = b;}
5      LegoMov() { evIsAwes = new bool(false); }
6  };
7
8  int main(void) {
9      LegoMov movie;
10     movie.setEvIsAwes(true);
11     return 0;
12 }
```

Are there any errors in the code?
○ Yes, `movie` is not initialized so we cannot call `setEvIsAwes`.
○ Yes, the value that `evIsAwes` points to is not modified after we call `setEvIsAwes`.
○ Yes, the code won't compile because of other unstated reasons.
● Yes, there might be a memory leak.
○ No, it looks good to me.

MISC6 (1.5pts)

Consider this simple example

```
class Bear {
    public:
        Bear() { cout << "Growl "; }
        void roar() { cout << "Roar "; }
        ~Bear() { cout << "Stomp stomp stomp "; }
};

int main(void) {
    Bear beary;
    beary.roar();
    cout << "Run! ";
    return 0;
}
```

What is the result of compiling and executing this code?

- ○ `Roar Run!`
- ○ `Run!`
- ● `Growl Roar Run!  Stomp stomp stomp`
- ○ `Growl Roar Run!`
- ○ This code does not compile.

MISC7 (1.5pts)

```
class MessageBlock {
    public:
        MessageBlock(const string & str) : msg(str) {}
        /* PART A */ (int position)
            { return msg[/* PART B */]; }
    private:
        string msg;
};
```

```
MessageBlock privet("3apabctbynte");
for (int i = 0; i < 12; i += 1)
    cout << privet(i);
cout << endl;
```

Which of the following statements complete the code above so that the output is `etnybtcbapa3`?

- ○ `char & operator()` and `msg.length()-position`
- ○ `char & operator[]` and `position`
- ○ `char & operator()` and `position`
- ○ `char & operator[]` and `msg.length()-position-1`
- ○ `char & operator()` and `msg.length()-position-1`

MISC8 (1.5pts)

Consider this simple code that protects your backyard. How many times is `plant`'s copy constructor called? Note that similar to the case of the default constructor, compilers may explicitly generate `plant`'s copy constructor and assignment operator.

```
1  plant * imitater(plant & orig) {
2      plant * tater = new plant(orig);
3      return tater;
4  }
5
6  int main(void) {
7      plant peashooter;
8      plant *repeater;
9      repeater = imitater(peashooter);
10     return 0;
11 }
```

○ Never, but the code executes with no errors.
○ Never, because this code has a compiler error.
● One time.
○ Twice.
○ Three times.

MISC9 (1.5pts)

How many times are all the `plant` constructors called in the example above?

○ Never, but the code executes with no errors.
○ Never, because this code has a compiler error.
○ One time.
● Twice.
○ Three times.

MISC10 (3pts)

The code in MISC8 gives us heebie jeebies because it R eturns the M emory A ddress of a variable that G oes O ut O f S cope !!!

MISC11 (1.5pts)

```
1 void foo(int &bar, int *baz) {
2     // FILL IN THIS LINE
3 }
```

What is the statement for assigning the sum of the value referred by `bar` and the value pointed to by `baz` to `bar`?

- ○ `bar = bar + baz;`
- ● `bar += *baz;`
- ○ `&bar = &bar + *baz;`
- ○ `*bar += baz;`
- ○ `*bar = bar + *baz;`
- ○ None of the other options.

MISC12 (1.5pts)

What about assigning the sum to `baz`?

- ○ `*baz += &bar;`
- ● `*baz = bar + *baz;`
- ○ `*baz = &bar + *baz;`
- ○ `&baz += *bar;`
- ○ `baz = *bar + &baz;`
- ○ None of the other options.

MISC13 (1.5pts)

```
1 int shrub = 10;
2 int *bush = &shrub;
3 cout << *bush << " ";
4 foo(shrub, bush);
5 cout << shrub << " " << *bush << endl;
```

Following **MISC12**, if we assign the sum to `baz` after calling `foo(int &, int *)`, which of the following options below would be printed out by the program?

- ○ 10 20 10
- ● 10 20 20
- ○ 10 10 10
- ○ 10 10 20
- ○ I can't find the answer because my code won't compile.

MISC14 (1.5pts)

Which of the following `vector` ADT implementations gives us an $O(n)$ time for `push_back`, i.e inserting an element at the end of the list, assuming we do not copy any data?

● A singly circular-linked list with only a `head` pointer.
○ A doubly circular-linked list with only a `head` pointer.
● A doubly-linked list with only a `head` pointer.
○ A doubly-linked list with `head` and `tail` pointers.
○ None of the other options is correct.

MISC15 (1.5pts)

In a sorted singly linked list, what will be the time required to insert at the middle position of the list?

○ $O(1)$
○ $O(\log \log n)$
○ $O(\log n)$
○ $O(n)$
○ $O(n \log n)$

MISC16 (1.5pts)

Suppose you are given a pointer to some node in a singly linked list, but NOT the head pointer. Is it possible to delete the node from the list?

○ Yes, we can find a way to delete any of the nodes.
○ Yes, but we can only delete a node if it's not the last node.
○ No, because we need to change the `next` field in the previous node.
○ No, because we cannot delete the pointer to the node.
○ None of the above is correct.

MISC17 (1.5pts)

Suppose we have implemented the Queue ADT as a singly linked list with `head` pointer and no sentinels, nor `tail` pointer. Which of the following best describe the tightest running times for the functions `enqueue` and `dequeue`, assuming there are $O(n)$ items in the list, and that the front of the queue is at the `head` of the list?

○ $O(1)$ for both.
○ $O(n)$ for both.
○ $O(1)$ for `enqueue` and $O(n)$ for `dequeue`.
○ $O(n)$ for `enqueue` and $O(1)$ for `dequeue`.
○ None of the options is correct

MISC18 (1.5pts)

We have implemented the Queue ADT as a circular array of size n, and we denote the head, or exit from the queue, as h. Every time the array is full, you resize the array creating a new array that can hold four times as many elements as the previous array and copy values over from the old array. For each of the elements in the new array with index i, you'd assign the element at index [ ] in the old array.

○ i
○ $n - 4i$
○ $4n - i$
○ $i - h \mod 4n$
○ $h + i \mod n$
○ $i - h \mod n$
○ $4i + h \mod n$

MISC19 (1.5pts)

Suppose we have an non-empty stack `std::stack<int> s`. What is the result of executing the following code snippet, assuming all required libraries are included?

```
1  int foo(std::stack<int> &s) {
2      int top = s.top();
3      s.pop();
4      if (s.empty()) return top;
5      else {
6          int bar = foo(s);
7          s.push(top);
8          return bar;
9      }
10 }
```

○ An arbitrary element in `s` is returned.
○ The smallest element in `s` is returned, assuming arbitrary `s`.
○ The largest element in `s` is returned, assuming arbitrary `s`.
○ The element at the bottom of `s` is returned.
○ The element at the top of `s` is returned.

MISC20 (1.5pts)

What is the result of executing the following code snippet? Assume `foo(std::stack<int> &)` is the same as defined in **MISC18**.

```
void fun(std::stack<int> &s) {
    if (s.empty()) return ;
    int e = foo(s);
    fun(s);
    s.push(e);
}
```

○ `s` remains the same.
○ The elements of `s` are reversed.
○ The elements in `s` are sorted.
○ The elements in `s` are shuffled in a random order.
○ The top half of `s` contains even elements and the bottom half of `s` contains odd elements

MISC21 (1.5pts)

Consider the following function definition and suppose that 1) the `node` class consists of an integer `data` element, and a `node` pointer called `next`, and 2) variable `head` is the address of a linked list of such `node`s.

Which of the following statements are correct after the while loop ends?

```
void bar(node * curr) {
   while (curr->next->next != NULL) {
      curr = curr->next;
      if (curr->next == NULL) {
         baz(curr);
      }
    }
}

node * head = NULL;
// we insert at least three nodes into the chain
bar(head);
```

○ `baz` has been invoked and `curr` points to the second last element.
○ `baz` has never been invoked and `curr` points to the last element.
○ `baz` has never been invoked and `curr` points to the second last element.
○ `baz` has been invoked and `curr` points to the last element
○ The program aborted before the while loop ends.
○ None of the other options is correct.

MISC22 (1.5pts)

In implementing Queue ADT, using which of the following data structure gives best asymptotic runtime for `enqueue` and `dequeue`? (Assume best possible implementation for queue using provided data structure and that the front of the queue is at the `head` of the list)

○ Singly linked list with head pointer only.
○ Singly linked list with head and tail pointer.
○ Doubly linked list with head pointer only.
○ Singly circular-linked list with tail pointer only.
○ Doubly circular-linked list with head pointer only.

MISC23 (1.5pts)

Consider a class `list` that is implemented using a singly circular-linked list with a `head` pointer. Given that representation, which of the following operations could be implemented in $O(1)$ time, assuming you *are* allowed to copy data?

○ Insert an item before the head pointer
○ Insert an item after the head pointer
○ Delete the item before the head pointer
○ Delete the item after the head pointer
○ None of them

2. **[Inoha Sort – 22 points].**

   Inoha has many books stored in one pile in a box and would like to sort them by weight in ascending order. She can only take out one book each time, so she borrows another box from her friend Hanoi in order to facilitate this task. Now, she can take out the book on the top of her box and (optionally) put it into the other box, or set it on the table. The other box holds as many books as the original, but she can only put one book on the table. Can you design an algorithm that sorts the books into one of the boxes?

   (a) (4 points) Show how to sort the books in Ihona's box step by step. The sorted books will be in Hanoi's box. Remember you don't need to immediately move the book from one box to the other. Instead, store it in temp!

Inoha's box	Hanoi's box	temp	operation
`1 5 3 7]`	`]`		`temp=i.pop()`

(b) (6 points) You are given the code below. Implement the `ihonaify` function. **Notice:** `stack::pop()` will not return the element on the top.

```
1  void inohaify(stack<int> &inoha, stack<int> &hanoi) {
2     // WRITE YOUR CODE HERE
3
4
5
6
7
8
9
10 }
11
12 int main(void) {
13
14        int numbers[] = {3, 5, 1, 7, 4};
15
16        stack<int> hanoi;
17        stack<int> inoha;
18
19        for (int i = 0; i < 5; i += 1) {
20              inoha.push(numbers[i]);
21        }
22
23        inohaify(inoha, hanoi);
24
25        return 0;
26 }
```

(c) (4 points) What are the worst case time and space complexity for this algorithm? Can you justify that?

Time: [ ] Space: [ ]

(d) (6 points) Now Inoha wants to find the book with the least weight (and possibly take it out) as quick as possible, but she would also like to add new books to these (infinitely big) boxes.

- (2 points) An ADT that satisfies Inoha's goal is called a *priority queue*. We are implementing this ADT with two (name another ADT) [ ]. In our application, the book with the least weight should be stored on the top of one of the two boxes, and thus taking out the book with least weight can be done in [ ] time.
- (4 points) The books in Inoha's box are sorted, and we'd like to maintain this status. However, when Inoha tries to add another book, she faces a challenge. We can solve this in [ ] time by:

(e) (2 points) Briefly explain how Inoha can get the k-th heaviest book from her box.

3. **[Raincouver – 30 points].** In case you hadn't noticed, Vancouver is a very rainy city! Suppose the daily rainfall data for n consecutive days is stored in an array `R`, so that `R[d]` is the number of centimeters of rain that fell on day `d`. Often, we are interested in looking at how much total rain fell between two given days. In this problem we will build a structure to help us answer such queries in constant time.

   (a) (4 points) Assuming that $0 \leq i \leq j < n$, prove the correctness of the following algorithm by arguing that it returns the sum of daily totals between `R[i]` and `R[j]`, inclusive, for any $0 \leq i \leq j < n$.

```
1 int getSum(const vector<double> & R, int i, int j){
2   if (i == j)
3     return R[i];
4   else
5     return R[j] + getSum(R, i, j-1);
```

   i. Base case: argue briefly that the base case is correct.

   ii. State an appropriate inductive hypothesis:

   iii. Prove the inductive case:

   (b) (2 points) What is the worst case running time of the algorithm in the previous part?

(c) (2 points) Suppose we want to pre-compute the sum of rain in *all possible* consecutive sets of days. Exactly how many sums do we need to compute for an array of size n?

(d) (4 points) We propose the following algorithm to compute some of the entries of a 2D array S:

```
int main(void) {
    vector <double> R;
    ... // initialize R to contain n daily rainfall amounts
    vector <vector <double>> S;
    for (int j = 0; j < n; j++) {
        vector <double> temp;
        for (int i = 0; i < j; i++)
            temp.pushBack(getSum(R,i,j))
        S.pushBack(temp);
    }
    return 0;
}
```

What is the worst case running time of this algorithm to build the query structure, and how much space does it require in terms of n?

Time: Space:

(e) (2 points) What is the running time to use `S` to answer queries like "what is the total rainfall between day i and day j? (inclusive)" for any $1 \leq i \leq j \leq n$?

(f) (4 points) Let's be clever about this... suppose we build an array `A` whose size is the same as `R`, but whose entries are cumulative total rainfalls from the start day. What is the running time to build such a structure, and how much space does it use? (We are not asking you to show us the pseudocode solution to this problem, but you should sketch it out in order to answer the question!)

Time: Space:

(g) (4 points) Give an expression in terms of the new array `A` that can be used to answer queries like "what is the total rainfall between day 0 and day j (inclusive)?" Also tell us the running time for answering the query.

Expression: Time:

(h) (4 points) Give an expression in terms of the new array `A` that can be used to answer queries like "what is the total rainfall between day i and day j (inclusive)?" Also tell us the running time for answering the query.

Expression: [ ] Time: [ ]

(i) (4 points) Finally, we are going to design an algorithm to build *another* structure that will allow us to answer queries of the form "what was the rainiest stretch of k days?" for any $1 \leq k \leq n$.

Our goal, in the end, will be to have an array `V`, whose entry `V[k]` gives the last day number of the rainiest `k` day stretch, for $1 \leq k \leq n$. For example, if `V[3] == 37`, then the rainiest 3 day stretch is the set of days 35, 36, and 37.

Suppose you have accurately built array `A`. Assume, in addition, you have (perhaps magically) completed iteration j of the algorithm to compute `V`, so that `V[k]` indicates the rainiest interval of length `k` within the stretch 0 to day $j-1$. This is an invariant on the state of `V`.

Complete the code below to maintain the invariant on array `V` in the next iteration of a loop. To do this, think about how the entries of `V` might change, given `A[j]`.

```
1  void updateRainiest(const vector<double> & A, vector<int> & V, int j) {
2      //assumes A[0..n-1] contains correct cumulative sums
3      //assumes V[1..n] has been correctly updated up to iteration j.
4
5
6
7
8
9
10
11
12
13
14
15     //end condition: V[1..n] has been correctly updated for iteration j
16     return;
17 }
```

Blank sheet for extra work.