CMPT 135-D100 Mini Midterm
Spring 2024

First name (please write as legibly as possible within the boxes)

Last name

SFU ID number

This is a **20 minute closed book exam**: notes, books, computers, calculators, electronic devices, etc. are **not** permitted. Do not speak to any other students during their exam or look at their work. Please remain seated and **raise your hand** if you have a question.

Pointers and Memory Management

(5 marks) Write a function called `safe_delete(string* s, string* t)` that correctly deletes the values that `s` and `t` point to. Assume the following:

- It's possible that one, or both, of `s` and `t` could be the null pointer.
- If `s` and `t` do point to strings, then those strings are on the free store.
- `s` and `t` might point to the same string, or to two different strings.

Use this function header and write your answer under it:

```
void safe_delete(string* s, string* t) {
```

Sample Solution

There are various possible solution

```
// This tricky solution relies on the
// fact that calling delete on a
// nullptr is okay in C++ and does not
// nothing without causing a problem.
void safe_delete(string *s, string *t) {
    if (s == t) {
        delete s;
    } else {
        delete s;
        delete t;
    }
}
```

```
// This version clearly covers each
// case. It is not the shortest, but
// is probably the easiest to
// understand.
void safe_delete(string *s, string *t) {
    if (s == nullptr && t == nullptr) {
        return;
    } else if (s == nullptr) {
        delete t;
    } else if (t == nullptr) {
        delete s;
    } else if (s == t) {
        delete s;
    } else {
        delete s;
        delete t;
    }
}
```

Marking Scheme

- **+1 mark**: correct when s and t point to different strings
- **+1 mark**: correct when s and t point to the same string
- **+1 mark**: correct when s points to a string and t is the nullptr
- **+1 mark**: correct when s is the nullptr and t points to a string
- **+1 mark**: correct use of delete

Up to -1 mark deducted if the code is correct, but is very inefficient, does anything unnecessary, or is unusually difficult to read.

Notes

- If an error is very small, or maybe it's just a slip of the pen, you can take off no marks, or just a few marks. Forgetting one ";" is probably not worth taking any marks off. But repeatedly forgetting a ";" should be a deduction of at least 0.5.
- Sometimes answers can be different than what the marking scheme expects, and in that case you may need to "wing it". In such cases, first try to decide if it is a failing or passing answer. If it's failing, don't give more than 50%.

```
// This should be correct, but the if-statements
// within if-statements make it hard to read.
void safe_delete(string *s, string *t) {
    if (s != nullptr) {
        if (s == t) {
            delete s;
        } else {
            delete s;
            if (t != nullptr) {
                delete t;
            }
        }
    } else if (t != nullptr) {
        delete t;
    }
}
```

Object-oriented Programming and Inheritance

(5 marks) Create a class called `Circle` that stores the (x,y)-center and radius of a circle. Make these private, and call them `x`, `y`, and `radius`. In addition, add the following:

1. A **default constructor** that sets both `x` and `y` to 0, and the `radius` to 100.
2. A **copy constructor** that uses an **initialization list** to make a new `Circle` object that is a copy of a another `Circle` object.
3. A **destructor** that prints "done!".
4. A **setter** that lets the user change the `radius` of the circle. If a user tries to set `radius` to a value that is 0 or less, then the `radius` is *not* changed.

Sample Solution

```
class Circle {
    double x;
    double y;
    double radius;

public:
    Circle() : x(0), y(0), radius(100) {}

    Circle(const Circle& c)
    : x(c.x), y(c.y), radius(c.radius) {}

    ~Circle() { cout << "done!"; }

    void set_radius(double r) {
        if (r > 0) radius = r;
    }
};
```

Marking Scheme

- **+1 mark**: correct private variables
- **+1 mark**: correct default constructor that uses an initialization list
- **+1 mark**: correct copy constructor that uses an initialization list
- **+1 mark:** correct destructor
- **+1 mark:** correct setter

Up to -1 mark deducted if the code is very inefficient, or does anything unnecessary.

Notes

- If an error is very small, or you maybe just a slip of the pen, you might decide to take no marks off, or only a few marks. Forgetting one ";" is probably not worth taking any marks off. But repeatedly forgetting a ";" should have a deduction of at least 0.5.
- Sometimes answers can be different than what the marking scheme expects, and in that case you may need to "wing it". In such cases, first decide if it is a failing or passing answer. If it's failing, don't give more than 50%.

Multiple Choice

For each of the following questions, fill in **the one best answer** on the answer sheet.

Every correct answer is worth 1 mark. Incorrect answers, unanswered questions, questions with more than one answer, or questions with illegible answers, are worth 0.

1) Consider these statements:

   i) Any C++ for-loop can be re-written as a while-loop that does the same thing.
   ii) Any C++ switch statement can be re-written using just if-statements or if-else statements.

   A. i) and ii) are both true
   B. i) and ii) are both false
   C. i) is false and ii) is true
   D. i) is true and ii) is false

2) Consider these statements:

   i) Blackbox tests for a function can be written before the function's body is written.
   ii) Unit testing is a kind of whitebox testing.

   A. i) and ii) are both true
   B. i) and ii) are both false
   C. i) is false and ii) is true
   D. i) is true and ii) is false

3) Suppose `arr` points to an array of 10 doubles in the free store. What is the correct way to de-allocate `arr`?

   A.
```
for(int i = 0; i < 10) {
    delete arr[i];
}
```

   B.
```
for(int i = 0; i < 10) {
    delete arr[i];
}
delete[] arr;
```

   C.
```
delete arr;
```

   D.
```
delete[] arr;
```