Name _____

CS 251 Practice Final Exam R. Brown May 1, 2015

SHOW YOUR WORK—No work may mean no credit

Point totals will be adjusted to a 120 point scale later

I pledge my honor that I have neither given nor received assistance during this exam, and that I have seen no dishonest work.

Signed _____

☐ I have intentionally not signed the pledge (*check only if appropriate*)

(*6 pts*) 1. Write a definition of a C++ function `doublestring` that satisfies the following specification.

`doublestring`

1 Argument: A string (null-terminated array of characters).

State change: If *arg1* is not an empty string, a string with twice the length of *arg1* is allocated dynamically, and assigned to hold two complete copies of the characters in *arg1*.

Return: If *arg1* is not an empty string, return the doubled string that was allocated and assigned. Otherwise, return 0.

For example, the call `doublestring("hello")` should return a newly allocated string `"hellohello"`, and `doublestring("")` should return 0.

(*9 pts*) 2. Draw a memory diagram for the following complete C++ program, using the conventions presented in class, including output.
Note: No definition is provided for the method `makeString`. Assume that `makeString` satisfies the spec on the following page, and show any calls to `makeString`, but **omit the details** of those calls that could only be known by having an implementation.

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

class Name {
protected:
  char *name;

public:
  Name(const char *n) { name = makeString(n); }
  Name() { name = makeString(""); }
  char *getName() { return name; }
  char modify(int num) {
    name[0] += num; return name[0];
  }
  char *makeString(const char *str);
};

struct Item : public Name {
  double price;

public:
  Item(const char *n, double p) {
    name = makeString(n); price = p;
  }
  char modify(int num) {
    price = price * num;
    name[0] += num; return name[0];
  }
};
```

```
double func(double x, double &y) {
  double saved = y;
  y = y + x;
  return saved;
}

int main() {
  double x = 2.5;
  char *y = new char[5];
  y[1] = 'A';
  Item it("cat", 4.1);
  y[2] = it.modify(2);
  Name nm(it.getName());
  y[3] = nm.modify(3);
  cout << it.price << it.getName()
       << nm.getName() << endl;
}
```

(Additional answer space for memory diagram problem)

`makeString (method)`
1 Argument: A C-style string (null-terminated array of characters).
State change: A newly allocated copy of *arg1* is created.
Return: C-style string, that newly allocated copy of `str`.

(*6 pts*) 3. Answer **one** of the following questions. Summarize your answer in words; provide an example if desired. Clearly indicate which part you are skipping.

a) The key rule of dynamic memory management is that *every* `new` *should have exactly one* `delete`. What would go wrong if some call of `new` had no corresponding call of `delete`? What if a single call of `new` had more than one `delete` call?

or

b) What are some advantages of programming with subclasses? Include an example where a subclass of a class C would be more appropriate than having a state variable of type C.

(*12 pts*) 4. On the next pages, define and implement the class `Taxi` described by the accompanying spec.

- Define and implement the entire class `Taxi`. Implement the `Taxi` default constructor within the `Taxi` class definition (inline), and implement the `display()` method and the helper function `makeString` outside of that class definition. Other required members may be implemented within or outside that class definition.
- *Do not* define the superclass `Car` nor any of its members.
- Use C-style strings (`char *` type, null-terminated) with dynamic allocation and correct memory management.
- Use the helper function `makeString()` appropriately in the code you write.
- Call the `Car` assignment operator within your definition of the `Taxi` assignment operator, and call the `Car display()` method within your definition of the `Taxi display()` method.
- Use `const` appropriately. Use reference types whenever passing objects as arguments.

(Solution of implementation problem)

(Additional answer space for implementation problem)

Taxi, An example class representing a taxicab

R. Brown

Class *Taxi*

12/5/14 (R Brown): Initial entry.

Represents a taxicab.

Superclass: *Car*

State Variables for Class Taxi:

`driver` C-style string (null-terminated array of character) (e.g., `"Jim Doe"`), the driver for this taxicab.

`rate` Float (e.g., 2.20), the rate per mile of this taxicab.

Constructors for Class Taxi:

`Taxi` Constructor

Arguments:

`cr` *Car* object.

`dr` C-style string, driver for this instance of *Taxi*.

`rt` Float, rate per mile for this instance of *Taxi*.

State Change: This instance of *Taxi* is initialized by copying values from `cr` to the corresponding *Car* state variables in this *Taxi* object, with new allocation where appropriate, assigning a newly allocated copy of `dr` to the state variable `driver`, and assigning `rt` to the state variable `rate`.

`Taxi` Default constructor

Arguments: None.

State Change: This instance of *Taxi* is initialized by applying the default constructor for *Car* to assign values to the *Car* state variables in this *Taxi*, assigning a newly allocated empty string to the state variable `driver`, and assigning 0.00 to the state variable `rate`.

`Taxi` Copy constructor

Arguments:

`tx` An instance of *Taxi*.

State Change: This instance of *Taxi* is initialized by copying the *Car* state variables in this *Taxi* from the corresponding *Car* state variables in `tx`, and assigning the state variables `driver` and `rate` from the corresponding *Taxi* state variables in `tx`, with new allocation where appropriate.

Over...

Destructor for Class Taxi:

`~Taxi` Destructor

Arguments: None.

State Change: Memory is deallocated that was dynamically allocated for this instance of *Taxi* but not for the superclass *Car*.

Methods for Class Taxi:

`=` Assignment operator

Arguments:

`tx` Another instance of *Taxi*.

State Change: The state variables for this instance of *Taxi* are assigned copies of the values of the corresponding state variables in the argument `tx`, with new allocation where appropriate.

Return Value: This instance of *Taxi*.

`setRate`

Arguments:

`rt` Float.

State Change: The value `rt` is assigned to the state variable `rate`.

Return Value: None.

`setDriver`

Arguments:

`dr` C-style string.

State Change: A newly allocated copy of `dr` is assigned to the state variable `driver`.

Return Value: None.

`display`

Arguments: None

State Change: The state variables of this instance of `Taxi` are printed on standard output in the format

year make (*driver*, $*rate*)

Example: `2014 Ford (Jim Doe, $2.20)`

Return Value: None.

`makeString` Helper method

Arguments:

`str` A C-style string.

State Change: A newly allocated copy of `str` is created.

Return Value: C-style string, that newly allocated copy of `str`.

Car, An example class representing an automobile

R. Brown

Class *Car*

12/5/14 (R Brown): Initial entry.

Represents an automobile.

Superclass: *None*

State Variables for Class Car:

`make` C-style string (null-terminated array of character) (e.g., `"Ford"`), the manufacturer for this automobile.

`year` Integer (e.g., 2014), the model year of this automobile.

Constructors for Class Car:

`Car` Constructor

Arguments:

`mk` C-style string, manufacturer for this instance of *Car*.

`yr` Integer, model year for this instance of *Car*.

State Change: This instance of *Car* is initialized by assigning a newly allocated copy of `mk` to the state variable `make`, and assigning `yr` to the state variable `year`.

`Car` Default constructor

Arguments: None.

State Change: This instance of *Car* is initialized by assigning a newly allocated empty string to the state variable `make`, and assigning 1 to the state variable `year`.

`Car` Copy constructor

Arguments:

`cr` An instance of *Car*.

State Change: This instance of *Car* is initialized by assigning the state variables `make` and `year` from the corresponding *Car* state variables in `cr`, with new allocation where appropriate.

Destructor for Class Car:

`~Car` Destructor

Arguments: None.

State Change: Memory is deallocated that was dynamically allocated for this instance of *Car*.

Over...

Methods for Class Car:

`=` Assignment operator

Arguments:

`cr` Another instance of *Car*.

State Change: The state variables for this instance of *Car* are assigned copies of the values of the corresponding state variables in the argument `cr`, with new allocation where appropriate.

Return Value: This instance of *Car*.

`getMake`

Arguments: None.

Return Value: C-style string, the value of the state variable `make`.

`getYear`

Arguments: None.

Return Value: Integer value of the state variable `year`.

`setMake`

Arguments:

`mk` C-style string.

State Change: A newly allocated copy of `mk` is assigned to the state variable `make`.

Return Value: None.

`setYear`

Arguments:

`yr` Integer.

State Change: The value `yr` is assigned to the state variable `year`.

Return Value: None.

`display`

Arguments: None

State Change: The state variables of this instance of `Car` are printed on standard output in the format

year make

Example: `2014 Ford`

Return Value: None.