Introduction to Computer Science I

Winter 2021 - COM SCI31-1 - STAHL

Final Exam Review Solutions

Sample Problems:

In preparation for the Final Exam, please complete the following problems. We'll go over these problems during class Thursday to help you prepare for the Final Exam.

1. The class `Tire` represents a datatype that can be rolled and tracks it distance traveled. Consider the following:

```
class Tire
{
public:
    Tire( );

     void roll( int amount );
private:
     int myDistance;
};

Tire::Tire( ) : myDistance( 0 )
{ // empty... }

void Tire::roll( int amount )
{  myDistance += amount;  }
```

Based on this class `Tire`, create the class `Auto`. An `Auto` is made up of four tires, a front right, front left, rear right and rear left tire. Define the operation Auto::drive(int amount) which properly rolls each of its four tires.

```
class Auto
{
public:
  Auto( );
  void roll( );
private:
  Tire tires[4];
};

Auto::Auto( ) {  }
void Auto::roll( )
{
  for(int i = 0; i<4; i++)
  {
     tires[i].roll( );
  }
}
```

2. Define a structure `Candy` with the data fields `kind` (a string such as "M & M's"), `maker` (a string such as "Mar's") and `cost` (a double such as 0.99). Create a variable `c` of type `Candy` that represents a snicker's bar made by Mar's that costs $1.25. Print to `cout` each of the fields of the variable `c`. Create a pointer variable and give it the address of `c`. Print to `cout` each of the fields of this pointer variable. Will this pointer variable need to be `delete`'d when you are finished working with it

```
 string kind;
 string maker;
 double cost;
};

Candy c = { "Snicker's", "Mars", 1.25 };
cout << c.kind << c.maker << c.cost << endl;
Candy * ptr = &c;
cout << ptr->kind << ptr->maker << ptr->cost << endl;
// no delete needed!
```

3. When the compiler is provided many overloaded versions of the function foo, how does it decide which version of the method to invoke when driver code calls function foo?

C++ decides based on the types and number of the parameters. Each overload must be distinct and different to avoid ambiguity.

4. Consider the number line: **... -13 -12 -11 -10 -9** - 8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 **4 5 6 7 8 9 10 11 12 13...**

What would be the correct boolean expression that ensures an integer is in the **bolded blue** region of the number line above.

```
if ( i <-8 || i > 3 ) {        }
```

5. Consider the number line: ... -11 -10 -9 **-8** -7 -6 -5 -4 -3 **-2 -1 0 1 2** 3 4 5 6 7 8 9 10 **11 12 13 14 15...**

What would be the correct boolean expression that ensures an integer is in the **bolded green** region of the number line above.

```
if ( i ==-8 || i  > 10 || (i <=2 && i >=-2 )  {            }
```

6. When does C++ code use -> and when does C++ code use . ? Suppose we declare a pointer to a structure Candy you created in Question 2 above. Using that pointer variable, what portion is accessed with -> and what portion is accessed with . ?

objects work with .
pointers to object use ->. The thing it points needs to not be nullptr. When you use -> on a nullptr value, your code will crash.
structs do not support information hiding and are all public fields by default. So any part of the code can use . to access any data member.

7. What is the difference between an automatic variable and a dynamic variable? Who manages the lifespan of an automatic variable and when does its lifespan come to an end? Who manages the lifespan of a dynamic variable and when does its lifespan come to an end?

An automatic variable is fully managed by C++ and winds up on the stack. A dynamic variable is one we new and delete ourselves. Automatic variables live and die within the { } scope in which they have been declared. Dynamic variables can be new'ed in one part of the code and delete'd much later in a completely different pile of code. Dynamic variables don't die off until they are manually deleted.

8. How does std::string handle > and < comparisons? Will it consider the string's length or its dictionary-sorted order to determine this boolean value?

string's operator < and operator > use lexicographic sorting based on the collating sequence of our machine. In English, this means "dictionary sorted order" based on the ASCII table.

9. Suppose someone tries to store into a single integer a student number and a student percentage score for a certain exam. For example, the integer 10085 might be used to store the fact that student number 100 received the value 85 on a certain exam. For example, the integer 876597 might be used to store the fact that student number 8765 received the value 97 on a certain exam. Assuming no one scored more than 99 on a certain exam and assuming you will receive data with atleast three digits or more so the student number and percentage score always both exist, write code that reads in a single integer value and then breaks this packed integer into its two fields, the student number and the student percentage score.

```
int value;
cin >> value;
int studentNumber = value / 100;
int score = value % 100;
```

10. Related to Project 5, suppose we create the ability for flyers to earn bonus miles that would mean their account gets credited with twice the miles they flew. With this new feature in place, consider the following asserts:

```
    assert( std::to_string( a.getBalance() ) == "0.00000" );
    a.bonusMilesStarted( );
    a.addFlightToAccount( f );   // counts as 570 miles flown...
    a.bonusMilesEnded( );
    a.addFlightToAccount( f );
    assert( std::to_string( a.getBalance( ) ) == "855.000000" );
```

```
class FrequentFlyerAccount
{
public:
    FrequentFlyerAccount( string name );

    double getBalance( );
    string getName( );

    bool addFlightToAccount( PlaneFlight flight );
    bool canEarnFreeFlight( double mileage );
    bool freeFlight( string from, string to, double mileage, PlaneFlight & flight );

    void bonusMilesStarted( );
    void bonusMilesEnded( );

private:
    string mName;
    double mBalance;
    bool   mBonusMiles;
};

FrequentFlyerAccount::FrequentFlyerAccount( )
{
   mName = "";
   mBalance = 0;
   mBonusMiles = false;
}
```

```
void FrequentFlyerAccount::bonusMilesStarted( )
{
   mBonusMiles = true;
}
```

```
void FrequentFlyerAccount::bonusMilesEnded( )
{
   mBonusMiles = false;
}
```

```
bool   FrequentFlyerAccount::addFlightToAccount( PlaneFlight flight )
{
    bool result = false;
    if (flight.getName() == mName && flight.getCost() > 0 && flight.getFromCity() != "" &&
        flight.getToCity() != "" && flight.getFromCity( ) != flight.getToCity() )
    {
        mBalance += flight.getMileage();
        if (mBonusMiles)
        {
           // double the miles...
           mBalance += flight.getMileage();
        }
```

```
        result = true;
    }
    return( result );
}
```

changed to accommodate this change? Suppose we decide to allow three Players to play, not two. Which classes would need to be changed to accommodate this change?

For ten rounds of play, all we would need to change is the BeatThat class. It has a private constant int named MAX_NUMBER_OF_TURNS which currently equals five. Update it to ten and we will have a game that plays longer.

For three dies, the Player class would need to be updated to include a third Die data member. In addition to .largestDie( ) and .smallestDie( ), we would need to make .middleDie( ). Then in BeatThat, we would then have to update both versions of the humanPlay( ) and computerPlay( ) operations to use all three dies.

For a third player, the BeatThat class would need to be updated to include a third Player data member. Then in BeatThat, we would need to introduce a thirdPlayerPlay( ) operation.

12. What is the difference between int & and int *? What is the difference between int * and a declared array of int? How can you walk an array of int without using [ ] s?

int * is a pointer, an address of a location in memory. Any pointer might have the value nullptr.

int & is a reference variable we use to pass a variable by value from driver code. An int & is created when driver code passes a variable of the exact right matching type which in this case is int. An int & can never be nullptr.

An array variable holds the address of the zeroth element of the array. So an array variable is a pointer variable. But this pointer is a fixed arrow that cannot be moved. You can walk the arrow and change the box, but the arrow itself cannot be moved somewhere else. With pointer arithmetic, you can access any array element by offset off the array variable. array[5] is the same element as *(array + 5)

13. What portions of a class `Foo` can be called by a `public` method of the class `Foo`? What portions of a class `Foo` can be called by a `private` method of the class `Foo`? What portions of a class `Foo` can be called by using the object `f` by the operation `void Bar::b( Foo f )` ? What portions of a class `Foo` can be called by using the object `f` by the operation `void Bar::c( const Foo & f )` ?

public parts of the Foo class can access any part of the Foo class, as long as that public operation has not been marked const. A read-only const operation of Foo can only access other read-only const operations.

private parts of the Foo class can access any part of the Foo class, as long as that private operation has not been marked const. A read-only const operation of Foo can only access other read-only const operations.

code outside the Foo class can only use the public part of the Foo class.

A read-only Foo object can be used to only access the public read-only parts of the Foo class

Last modified: Thursday, 11 March 2021, 11:28 AM PST

◄ Final Exam Review

Jump to...

Worksheet 9 ►