CS31 Introduction to Computer Science Review

Luis Ángel Larios Cárdenas

June 5, 2015

Exercise 1

What is the output of the following code snippet? Consider that a boolean is outputted as `1` if `true`, and `0` otherwise. Assume all necessary libraries have been loaded, and that the `std` name space is in use.

```
int array[] = { 4, -2, 0 -5, 3, 3 };

int &a = array[3];
a++;

int *b = array;
b+=4;
*b = -a;

int c = b - &a;
array[c]*= -2;

bool d = *(array+1) == *(b-1);

int* &e = b;
e = array;

cout << a << *b << c << d << *e << endl;
```

Answer

(a) It won't run: there's a compilation error.

(b) Undefined behavior.

(c) `4-4114`

(d) `4-410-4`

(e) `44114`

(f) `-44104`

Exercise 2

The following function is intended to reverse a `C string` while, at the same time, transforms the string letters into uppercase. For example, if the input parameter is `"exam"`, it modifies the string into `"MAXE"`. Although the function compiles, there's a bug and one of the strings below will not be modified appropriately. What is such test string? Both `<cstring>` and `<cctype>` have been included to check the function.

```
void reverse( char* str )
{
    char* tail = &str[ strlen(str)-1 ];

    for( ; tail > str; str++, tail-- )
    {
        char temp = toupper(*tail);
        *tail = toupper(*str);
        *str = temp;
    }
}
```

Answer

(a) `"albatroz"`

(b) `"RISE OF ROME"`

(c) `"European"`

(d) `"Mediterranean"`

(e) `"deep lake"`

(f) `"A diligent 1st-class"`

(g) `"K Pop"`

Exercise 3

We are attempting to write a function that finds the *first occurrence* of a `C string` within another string. Our function should return a pointer to `s1` if `s2` is found. Thus, for example, if `s1` is `"Hey, you!"`, and `s2` is `"y, "`, `strInStr()` must return a pointer to the character at the subscript `2` of `s1`. If `s2` is not found in `s1`, a `nullptr` should be returned. Your task is to fix the function by fixing the pointer usage appropriately.

```
const char* strInStr( const char* s1, const char* s2 )
{
   const char* b = s2;

   while( s1 != 0 && s2 != 0 )
   {
      if( *s1 == *s2 )
      {
         const char a;
         for( a = s1; s1 == s2; *s1++, *s2++ )
            ;
         if( s2 == 0 )
            return a;

         s1 = a;
         s2 = b;
      }

      *s1++;
   }

   return nullptr;
}
```

Even after fixing the pointer notation in `strInStr()`, our function is not quite right yet. It has a bug that yields *undefined behavior* for some unfortunate input combinations. Which of the following input choices will make your function run into *undefined behavior*?

(a) `s1 = "A christmas tale"` and `s2 = ""`

(b) `s1 = "Hello, World"` and `s2 = "llo "`

(c) `s1 = "We are! we are!"` and `s2 = "are!"`

(d) `s1 = "Hey, bring pizza"` and `s2 = "bring piza"`

(e) `s1 = "Deadly sins"` and `s2 = "deadly"`

(f) `s1 = "Jurassic Park"` and `s2 = "Jurassic World"`

(g) `s1 = "La vida loca"` and `s2 = "loca"`

(h) `s1 = "Age of Empires 1"` and `s2 = "Age of Empires 2"`

(i) `s1 = ""` and `s2 = "Oh, no!"`

Exercise 4

Complete the definition of a class `Rational` to execute fractional arithmetics. The class contains two `private` data members, a `private` utility function to *simplify* the fraction, and `public` accessors to the *numerator* and *denominator*. The class must have two constructors: a default one that initializes both `num` and `den` to 1, and a constructor with parameters that stores the fraction in simplified form (i.e. by using the auxiliary `private` function `simplify()`) – for example, the rational number

$$\boxed{-\frac{2}{4}}$$

should be stored as `num = -1`, and `den = 2`. When a new `Rational` number is created, you should keep the fraction sign *always* in the numerator.

Besides the provided public functions, you must add the following methods (both prototype and definition):

(a) Add two `Rational` numbers (i.e. get as input the other summand), and return a simplified `Rational` number result. (Should not be larger than 5 lines of code.)

(b) Multiply two `Rational` numbers, and return a simplified `Rational` number product. (Should not be larger than 4 lines of code.)

(c) Print the `Rational` number in the format `"num/den"`. (Should not be larger than 1 line of code. Assume that `iostream` and `using namespace std` were properly included.)

```
class Rational
{
   private:
      int num;                    // Numerator.
      int den;                    // Denominator.

      void simplify()             // Utility function to simplify a fraction.
      {
          // TODO: Complete the definition. (Note: be aware that the numerator may
          // be negative!)









      }
```

```
public:
   Rational( int num, int den )   // Constructor with parameters.
   {
      // TODO: Assign num and den to numerator and denominator respectively.
      // Fractions such as 1/-2 should be stored as -1/2, -1/-2 as 1/2, and 2/4 as 1/2.
      // Use simplify() within this function.









   }

   Rational()                 // Default constructor.
   {
      // TODO: One liner.

   }

   int getNum() const         // Return the value of numerator.
   {
      // TODO: One liner.

   }

   int getDen() const         // Return the value of denominator.
   {
      // TODO: One liner.

   }

   // TODO: Declare and define here the 'add' function: it should
   // get as a parameter a reference to a constant Rational number
   // and return the result of adding that operand to this object.
```

```
    // TODO: Declare and define here the 'multiply' function: it should
    // get as a parameter a reference to a constant Rational number
    // and return the product of that number with the current object.









    // TODO: Declare and define here the 'print' function that
    // cout's this Rational number in the format "num/den". It should
    // not return anything.









};    // End of Rational class.
```

Test your class (and Math). What are the results of the `cout` statements in the following `main` function?

```
int main()
{
   Rational r1( 2, -4 );
   Rational r2( -1, 3 );

   r1.add( r2 ).print();        // What is outputted?
   r2.multiply( r1 ).print();   // What is outputted?

   return 0;
}
```

Exercise 5

We would like to create a `PlayList` class, which is made of up to 100 `Song` objects. The `Song` class holds important `string` properties such as `title`, `genre`, `singer`, and a pointer to the `PlayList` it belongs to. The only possible usefulness of a `Song` is to print a list of other similar songs (by *genre*) that reside in its `PlayList`.

On the other hand, a `PlayList` object can dynamically allocate up to 100 `Song` objects by calling its member function `addSong()`. Additionally, knowing that the user can get tired of the songs from a particular artist, the `PlayList` class offers the method `removeSongsBySinger`, which removes all songs corresponding to a given *singer*. Finally, the `PlayList` class also allows users to get a refined list of songs that match certain *genre* criterion.

Your job is to complete the definitions of the following class declarations. You may find the *comments* in the function prototypes helpful when defining their content. At the end of the classes there is a small `main()` function to test your classes and understanding of the task. It's recommendable to go over the `main()` function and its desired output first in order to better understand the expected behavior.

```
//////////////////////////////// Class declarations /////////////////////////////////////

const int MAX_SONGS = 100;

class PlayList;             // Forward declaration.

class Song
{
private:
   string title;            // Song's title.
   string genre;            // Song's genre.
   string singer;           // Song's singer.
   PlayList* playListPtr;   // Pointer to its playlist.

public:
   // Constructor.
   // Parameters: t => title, g => genre, s => singer, ptr => playListPtr.
   Song( string t, string g, string s, PlayList* ptr );

   // Accessors to private members.
   string getTitle() const;
   string getGenre() const;
   string getSinger() const;

   // Display (cout) the all songs in its playlist that match
   // "this" song's << genre >>.
   void getSimilarSongs() const;
};

class PlayList
{
private:
   int nSongs;                  // Effective number of songs.
   Song* songs[MAX_SONGS];      // An array of up to MAX_SONGS dynamically
                                // allocated Song objects.
public:
   // Constructor: just initializes nSongs to 0.
   PlayList();
```

```
   // Destructor: delete all remaining Song objects pointed to by the array songs[].
   ~PlayList();

   // Add a song.
   // Allocate a new dynamic song with the given properties. Upon success (e.g. there
   // still space in songs[]) return a pointer to the (constant) newly created Song
   // object; otherwise, return nullptr.
   const Song* addSong( const string title, const string genre, const string singer );

   // Populate the list[] array with pointers to songs that have the given genre.
   // Return the number of Song pointers you inserted in list[].
   int getSongsByGenre( const string genre, Song* list[] ) const;

   // Removing all songs that correspond to some artist.
   // Delete the dynamically created Song objects whose artist is singer.
   // Return the number of removed songs.
   int removeSongsBySinger( const string singer );
};

////////////////////////////// Class definitions for Song ////////////////////////////

Song::Song( string t, string g, string s, PlayList* ptr )
{




}

string Song::getTitle() const
{

}

string Song::getGenre() const
{

}

string Song::getSinger() const
{

}

void Song::getSimilarSongs() const
{







}
```

```
////////////////////////////// Class definitions for PlayList ////////////////////////////


PlayList::PlayList()
{

   cout << "Playlist has been created, and it's empty" << endl;
}


PlayList::~PlayList()
{



   cout << "Playlist has been destroyed." << endl;
}


const Song* PlayList::addSong( const string title, const string genre, const string singer )
{









   cout << "No more songs can be added" << endl;
   return nullptr;
}


int PlayList::getSongsByGenre( const string genre, Song* list[] ) const
{
   int count = 0;









   return count;
}
```

```
int PlayList::removeSongsBySinger( const string singer )
{
   int deleted = 0;









   return deleted;
}

////////////////////////////////// Testing the classes ////////////////////////////////

int main()
{
   PlayList pl;
   pl.addSong( "I promised myself", "Pop", "ATeens" );
   pl.addSong( "In the end", "Rock", "Linkin Park" );
   const Song* s1 = pl.addSong( "What you waiting for?", "Pop", "Gwen Stefani" );
   pl.addSong( "Call me baby", "KPop", "EXO" );
   pl.addSong( "Upside down", "Pop", "ATeens" );

   s1->getSimilarSongs();

   int r = pl.removeSongsBySinger( "ATeens" );
   cout << "Removed: " << r << " songs!" << endl;

   s1->getSimilarSongs();

   return 0;
}

////////////////////////////////// Desired Output ////////////////////////////////

Playlist has been created, and it's empty
'I promised myself' by ATeens
'What you waiting for?' by Gwen Stefani
'Upside down' by ATeens
Removed: 2 songs!
'What you waiting for?' by Gwen Stefani
Playlist has been destroyed.
```