Birla Institute of Technology & Science, Pilani – K K Birla Goa Campus
First Semester: 2016-2017
Assignments (Part III)

August 24, 2016

***Course No.*: CS F301** ***Course Title*: Principles of Programming Languages**
***Instructor-in-Charge*: Prof. Santonu Sarkar. *Instructor*: Soumyadip Bandyopadhyay**
***E-mails*:** santonus@goa.bits-pilani.ac.in; soumyadipb@goa.bits-pilani.ac.in

Preface

The assignments given here are meant to improve your knowledge on programming language, concepts and programming styles. In the Internet era we live in, it is impossible that you will NOT find these assignments and their answers or their variants somewhere. So it is very easy for you to copy them and show it to us. You can also copy the solution from your friend. Please note that I have no time or interest to check who is taking such a shortcut or who is genuinely trying. Remember, if you take a short-cut, it won't lead you anywhere; you will only deceive yourself, not me.

1 Control Flow

In the class we discussed the notion of short-circuiting to be an efficient compiler technique to evaluate a boolean expression. In this tutorial, write a small program that has the following boolean condition:

```
int a, b, c;
if (a && b || c && d) { // do something }
```

Create a makefile to compile the program with -S option. Open the assembly file to observe the short-circuiting. Then write a simple case statement and compile the program with -S option (add another line in the same makefile). Open the assembly file to observe the code.

2 Fastest Word fast

The goal of this experiment is to effectively use regular expressions for a business problem. This experiment involves making a Java Form based game which has the following format.

1. A Java web based application which will initialize a game between two opponents.
2. A countdown begins with each player given an English Alphabet.(Same alphabet for both players and chosen at random)
3. The players have to write as many words as possible that start with the alphabet given.
4. The players have a time of 60 seconds at which the game stops. The player with the highest valid entries(points) is declared the winner.

Only valid words will be counted for each player. Invalid words will be highlighted and no points will be awarded for invalid words. Guidelines for the validity of a word are as follows:

1. Each word entered by the player must be at least 5 characters long.
2. The words entered should be present in a dictionary. Use any online word spell checker API for this purpose.
3. Once a valid word is entered, all words that are the derivatives of the first word will not be valued even if they are valid words. For example, words like something, somewhere, somebody, someone must be counted as 1 point only.

The logic building for this will have extensive use of regular expression matching.

Challenge Assignment: The same assignment to be made as an Android App. Please take care of disabling any kind of cheat methods that can be employed here, like copy-paste of pre-populated words etc.

3 Fancy Iterator- STL Style

In the class you learnt the notion of a generator and iterator. These are cleaner idiom for iterating over a sequence without a need to know how the sequence is generated. Consider a generator that generates string of size l over a set of alphabets a, b, c, d in a lexicographically sorted order. Implement this in C++ and Java. Your implementation MUST NOT store an explicit representation of the entire sequence of strings. Call this class as LexGen. The created object should have constant size, independent of l.

For C++ code **1.** The class should be such that new LexGen(len) creates an object that represents the lexicographically sorted sequence of all strings over the above alphabet of `length=len`.

2. The class should provide methods `begin()` and `end()`. Both return an iterator of type `LexGen::iterator`. The iterator returned by `begin()` points to the first element in the sequence of strings, the one returned by `end()` points beyond the end of the sequence of strings.

3. The iterator class itself should provide a dereferencing operator and an increment operator.

The usage of this iterator should be:

```
LexGen l(3);
for (LexGen::iterator it = l.begin(); it != l.end(); ++it)
    cout << *it << endl;
```

Similarly, for the Java code: **1.** The class should be such that new LexGen(len) creates an object that represents the lexicographically sorted sequence of all strings over the above alphabet of `length=len`.

2. The class should provide a method `elements()`, which returns an object of type Enumeration that implements the Enumeration interface. As such LexGen needs to provide methods `hasMoreElements()` and `nextElement()`, which test whether there are more elements in the sequence to be traversed and return the next element in the sequence if there are any, respectively.

The usage would be:

```
LexGen l = new LexGen(3);  Enumeration e = l.elements();
while (e.hasMoreElements())
   System.out.println(e.nextElement());
```

4 OO Design-1

Complicated data structures are hard to implement. One of the reasons is that missing parts are often represented using null pointers; any operation on the data structure then needs to test for such null pointers, in order not to dereference them. The *Null Object pattern* can often be used to eliminate these tests, thereby reducing the chance for errors and increasing the elegance of your code. Search the internet to read more about Null Object Pattern. Implement a doubly-linked list class in C++ that supports the following operations:

1. `l.create` (implemented as the list's constructor): Creates a new doubly- linked list.
2. `l.destroy` (implemented as the list's destructor): Destroys the list l, deallocating all its nodes.
3. `l.head`: Returns l's head node or null if l is empty. (Here, null means null, not a pointer to the null object.)
4. `l.tail`: Returns l's tail node or null if l is empty. (Once again, null means null, not a pointer to the null object.)

5. `l.insertbefore(n, x)`: Given a node n of l (you do not need to check whether n is indeed a node of l) and an element x, create a new node n before n in the list and store element x in n. The method should return n.
6. `l.insertafter(n, x)`: Same as insertbefore, only the new node n suc- ceeds n.
7. `l.delete(n)`: Delete node n from l.
8. `n.pred`: Return the predecessor of node n in the list or null if n is the head of the list.
9. `n.succ`: Return the successor of node n in the list or null if n is the tail of the list.
10. `n.element`: Return the element stored at node n.

Assume the elements to be stored in the list are integers. Proper use of the Null Object pattern allows you to eliminate all checks for null pointers and all checks whether a given node is currently the head or tail of the list[1]. Your goal should be to completely remove `if` and `switch` statement. If that happens you can be sure that you have employed the Null Object pattern to its full extent.

5 OO-Design-II

Write simple programs to test the following:

1. Write a Java Program to implement nested classes and call a method inside the nested class from the main method.

2. Does the innerClass instance get created when the outerClass instance is created?

3. Can we create an object of the OuterClass type inside a method of the nested class.

Think: **1.** Can a method inside the Inner Class access the private members of the outer class?

2. Can a method of a private nested class access the members of the outer class?

Dirty Cloth catcher In the class you have seen a running example of a game in Java for Android platform. Revisit some OO concepts.

- Make a list of all possible entities that will get created while creating the game.(Except those inherent to the Android Framework)
- Categorize them into Classes Interfaces and Events/EventHandlers.
- Identify where we can apply “Hiding and Grouping” and “Reuse and Extend”.

6 Parallel Programming

7 Functional Programming

8 Scripting

To reduce the likelihood of typographic errors, the digits comprising most credit-card numbers are designed to satisfy the so-called Luhn formula, standardized by ANSI in the 1960s, and named for IBM mathematician Hans Peter Luhn. Starting at the right, we double every other digit (the second-to-last, fourth-to-last, etc.). If the doubled value is 10 or more, we add the resulting digits. We then sum together all the digits. In any valid number the result will be a multiple of 10. For example, 1234 5678 9012 3456 becomes 2264 1658 9022 6416, which sums to 64, so this is not a valid number. If the last digit had been 2, however, the sum would have been 60, so the number would potentially be valid.

Write a html page with embedded Javascript such that once you enter the credit card number and press "validate" button the Javascript should validate if the entered number is correct.

[1] You need to use more than a Null Object in the strict sense. The interpretation of Null Object you should use here is: the object does what should be done if this was a null pointer.

Challenge: Many sophisticated scripts would also check if the card is a Masters or a Visa card. Improve the Javascript that detects whether the card number belongs to Masters or Visa, and puts the appropriate logo on the HTML page.

9 Smart Memory Allocation

Objective-C uses smart pointers to eliminate the ownership problem that generally creates trouble to C/C++ code. It holds the last reference to a dynamically allocated object and thus should delete it when it is no longer needed. Objective-C's approach is to associate a reference count with every allocated object. When allocating an object, it's reference count is initialized to 1. When assigning a reference to an object to another variable, the programmer has the option to explicitly increase the reference count of the object. The Objective-C equivalent of C's `free()` function decrease the given object's reference count by 1 and frees the object only if the reference count is now 0. Thus, every increase of the reference count needs to be matched by a corresponding call to free() to avoid memory leaks. C does not support reference counts. One of the goals of this class is to make you think about simulating useful language features in languages that do not have this feature. In this question, you are to equip C with support for reference counts. In particular, you should provide implementations for the following three functions.

1. `refmalloc(int)` is the equivalent of C's standard malloc(int) function in that it allocates a memory chunk of the given size and returns a pointer to this chunk to the caller. Different from malloc, however, this function also stores a reference count for the allocated chunk.

2. `refincr(void*)` takes a pointer to a memory chunk allocated using refmalloc and increases the chunk's reference count by 1 to indicate that the caller also holds a reference to this chunk.

3. `refrelease(void*)` takes a pointer to a memory chunk allocated using refmalloc and decreases the chunk's reference count by 1 to indicate that the caller is about to destroy its last reference to this chunk. If this decreases the chunk's reference count to 0, the chunk is freed. Thus, this is in a sense the equivalent to C's standard `free()` function.

An important requirement is that any function other than these three can use a pointer returned by refmalloc in exactly the same way as a pointer returned by standard malloc. Also read your code and evaluate if you are making any implicit assumptions (which are usually satisfied). Explain these assumptions.

Instructor-In-Charge, CS F301
Santonu Sarkar