Homework $8\frac{3}{4}$

Due Wednesday, 2 May

Handout 25
CSCI 334: Spring 2018

Notes

This homework has three types of problems:

Self Check: You are strongly encouraged to think about and work through these questions, and may do so with a partner. Do not submit answers to them.

Problems: You will turn in answers to these questions.

Pair Programming: This part involves writing both Scala and C++ code. You are required to work with a partner for this section. I will assume that you plan to work with the same partner you worked with on the last assignment unless you email me.

Turn-In Instructions

You must turn in your work for this assignment in two parts, one for the Problems part and one for the Pair Programming part. You will be assigned two GitHub repositories.

Note that this assignment should be anonymized: neither your name nor your partner's name should appear in any of the files submitted with this assignment except the file "`collaborators.txt`" (see below).

Problems: Problem sets should be typed up using LaTeX and submitted using your `cs334_hw8point75_<username>` repository. For example, if your GitHub username is `dbarowy`, then your repository will be called `cs334_hw8point75_dbarowy`. Be sure that your work is committed and pushed to your repository by Wednesday, May 2 at 11:59pm.

If you discuss the problem set with your partner or with a study group, please be sure to include their names in a `collaborators.txt` file in your repository.

Pair Programming: Programming solutions should be typed up and submitted using your partner repository. For example, if your GitHub username is `dbarowy` and you are working with a partner whose username is `wjannen`, look for a repository called `cs334_hw8point75_dbarowy-wjannen` (usernames will be in alphabetical order). Be sure that your work is committed and pushed to your repository by Wednesday, May 2 at 11:59pm.

Reading

1. **(Required)** Mitchell, Chapters 12
2. **(Required)** Bendersky, Understanding lvalues and rvalues in C and C++ (on website)
3. **(As Needed)** Scala and C++ Resources

Self Check

S1. .... Pointers in C

The following questions are intended to familiarize you with pointers in C.

(a) Write a program that demonstrates the fact that arrays themselves are not passed to functions, but a pointer to the first element of the array is what is passed. Hint: The `sizeof` operator will be useful.

(b) According to the C standard, `arr[0]` is actually syntactic sugar for `*(arr+0)`. Write a program that prints all the elements of an integer array using this alternative notation.

(c) Create a simple function `void print_addr(int x)` whose sole purpose is to print the address of the integer `x` passed to it. Create an integer variable in `main`, print out its address, and then pass that variable to `print_addr`. Compare the results. Is this the expected behavior?

(d) Create a function `int *new_integer()` that declares and initializes an integer inside the function and returns the address of that integer. Print out the integer value associated with this memory address in `main`. Is this legal C? Does your complier display warnings? Is this a safe operation?

(e) If your compiler displayed no warnings for the previous question, try compiling it using the `-fsanitize=address` option for `clang`. Does the compiler help you track down the problem?

(f) Create a function `void print_array(...)` that prints out all values of an integer array. What parameters must the function have in order to work for <u>any</u> integer array?

(g) Write a program which uses an `ARR_SIZE` constant via `#define ARR_SIZE 10`. Create a function that allocates memory for `ARR_SIZE` integers, assigns the 0th integer to 0, the 1st integer to 1, and so on, and returns the address of the allocated space. Use the `print_array` function from the previous question to print out each element of the allocated array. After you have successfully made the program, try altering the value of `ARR_SIZE`.

Problems

Q1. (30 points) .... Assignment and Derived Classes

Consider the following C++ code:

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

class Vehicle {
public:
  int x;
  virtual void f();
  void g();
};

class Airplane : public Vehicle {
public:
  int y;
  virtual void f();
  virtual void h();
};

void inHeap() {
  Vehicle *b1 = new Vehicle;
  Airplane *d1 = new Airplane;
  /* location 1 */
  b1->x = 1;
  d1->x = 2;
  d1->y = 3;
  b1 = d1;
  /* location 2 */
}

void onStack() {
  Vehicle b2;
  Airplane d2;
  /* location 3 */
  b2.x = 4;
  d2.x = 5;
  d2.y = 6;
  b2 = d2;
  /* location 4 */
}

int main() {
  inHeap();
  onStack();
}
```

(a) Draw the state of memory (stack, heap, and vtables) after pointers `b1` and `d1` have been assigned to objects in the call to `inHeap` (`location 1`).

(b) Redraw the relevant parts of memory from part (a) to reflect the changes that result after the assignment `b1=d1` occurs (`location 2`).

(c) Draw the state of memory (stack, heap, and vtables) after objects `b2` and `d2` have been allocated in the call to `onStack` (`location 3`).

(d) Redraw the state of memory from part (c) to reflect the changes that result after the assignment `b2=d2` occurs (`location 4`).

(e) Why isn't all of `d2`'s member data copied into `b2`?

(f) Why isn't `b2`'s vtable pointer changed by the assignment in `onStack`?

(g) Explain the difference between the code executed by the call `b1->g()` in `inHeap` and the code executed by the call `b2.g()` in `onStack`. Why do you think C++ works this way?

(h) Attempting to provide a definition for "`void Airplane::g()` results the in a compiler error that states `error: out-of-line definition of 'g' does not match any declaration in 'Airplane'`". Why can't we provide a definition for `g` in `Airplane`?

(i) If we change the definition of

```
class Airplane : public Vehicle
```

to

```
class Airplane : Vehicle
```

the program no longer compiles. Why?

Your homework repository has a working version of this code if you would like to experiment with it. The starter version also includes a couple virtual methods that you may find helpful to understand what happens in the `inHeap` and `onStack` functions. To compile, write `clang++ q1_starter.cpp` and then run the executable `a.out` with the command `./a.out`.

Q2. (15 points) .... Templates and Overloading

Consider the following C++ code:

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

void printIt(int a, int b) {
  int c = a + b;
  cout << "You gave me " << a << " and " << b << "." << endl;
  cout << "Together they make " << c << "." << endl;
}

int main() {
  int a1 = 1.0;
  int a2 = 2.0;
  printIt(a1,a2);

  double b1 = 1.0;
  double b2 = 2.0;
  printIt(b1, b2);

  string c1 = "Oh ";
  string c2 = "noes!";
  printIt(c1, c2);
}
```

(a) The code above does not compile. Rewrite the `printIt` function as a C++ template function so that the code compiles and runs properly.

(b) In your working code, if we replace `printIt(c1, c2)` with `printIt("Oh ", "noes!")` the function no longer compiles. Why not?

Your finished program should be named `q2.cpp`. Answer the question about the code not working in a separate LaTeX file.

Your homework repository has a version of this code if you would like to experiment with it. To compile, write `clang++ q2_starter.cpp` and then run the executable `a.out` with the command `./a.out`.

Q3. (15 points) .... C++ Lambda Expressions

C++11 added lambda expressions to the language. They are especially concise when combined with the `auto` keyword, which instructs the C++ compiler to infer the type of the expression.

A C++ lambda has three parts: (1) a parameter list, (2) a function body, and (3) a "capture list". You should already be familiar with the first two items, but the third is probably new to you. A capture list instructs the compiler which variables used in the expression are free variables (i.e., are not bound to the parameter list) and must be "captured" from the environment that defines the lambda.

The following C++ lambda expression has its parts labeled:

`auto myFunction = [`③`](`①`){`②`};`

Such a function can be called like any other ordinary C++ function:

```
myFunction(...);
```

Your homework repository contains starter code for this question. You are given the following `swap` lambda, and you need to fill in the code for the `swapper` function.

```
  auto swap =
    [](int *a, int *b) {
      int tmp = *a;
      *a = *b;
      *b = tmp;
    };
```

Your `swapper` function must:

(a) be a recursive function, and

(b) produce exactly the same result as the "ugly" `for` loop.

Note that the supplied `swap` function is side-effecting (in fact, it returns `void`).

To compile, write `clang++ q3_starter.cpp` and then run the executable `a.out` with the command `./a.out`. Your finished program should be named `q3.cpp`.

Hint: A pointer to C-style array is just a pointer to the array's first element. Assuming that an array has more than 2 elements, moving the pointer to the second element is a pointer to the "subarray." Note, however, that there is no way to "query" a C-style array as to its length; you must keep track of this information yourself.

Pair Programming

P1. (20 points) .... Continuation Passing Style in C++

Many languages lack high-level support for continuations. For example, C++ does not have first-class continuations. However, it does have first-class functions, which means that you can still write functions in continuation passing style (CPS). In this question, you will convert an ordinary recursive function in C++ to its equivalent in CPS.

The following code performs a bubble sort in C++.

```
List<int> *bubble_inner(List<int> *xs) {
  if (!xs || !xs->tail) {
    return xs;
  } else {
    int a = xs->head;
    int b = xs->tail->head;
    List<int> *rest = xs->tail->tail;
    if (a < b) {
      return bubble_inner(rest->cons(b))->cons(a);
    } else {
      return bubble_inner(rest->cons(a))->cons(b);
    }
  }
}

List<int> *bubble_outer(List<int> *xs) {
  if (!xs) {
    return xs;
  } else {
    int x = xs->head;
    List<int> *xs2 = xs->tail;
    return bubble_inner(bubble_outer(xs2)->cons(x));
  }
}

List<int> *bubblesort(List<int> *xs) {
  return bubble_outer(xs);
}
```

Convert this program to CPS. Note that you must convert both `bubble_inner` and `bubble_outer` to CPS to get full credit. Make sure that your CPS implementation produces the same output as the given implementation.

Note that the supplied starter code provides a generic, recursive `List` implementation. You may find the following definitions helpful:

```
template <typename T>
auto Identity = [](T *x) { return x; };

typedef std::function<List<int>*(List<int>*)> Continuation;
```

Your finished program should be named `p1.cpp`.

P2. (60 points) .... Random Sentence Generator

The goals of this problem are to:

(a) write a class hierarchy in Scala, and
(b) utilize the Composite Design Pattern.

Random Sentence Generator. The "Random Sentence Generator" creates random sentences from a grammar. With the right grammar you could, for example, use this program to generate homework extension requests:

- Wear down the Professor's patience: I need an extension because I used up all my paper and then my dorm burned down and then I didn't know I was in this class and then I lost my mind and then my karma wasn't good last week and on top of that my dog ate my notes and as if that wasn't enough I had to finish my doctoral thesis this week and then I had to do laundry and on top of that my karma wasn't good last week and on top of that I just didn't feel like working and then I skied into a tree and then I got stuck in a blizzard on Mt. Greylock and as if that wasn't enough I thought I already graduated and as if that wasn't enough I lost my mind and in addition I spent all weekend hung-over and then I had to go to the Winter Olympics this week and on top of that all my pencils broke.
- Plead innocence: I need an extension because I forgot it would require work and then I didn't know I was in this class.
- Honesty: I need an extension because I just didn't feel like working.

Grammars. The program reads in grammars written in a form illustrated by this simple grammar file to generate poems:

```
<start> = The <object> <verb> tonight
;

<object> =
  waves
| big yellow flowers
| slugs
```

```
;

<verb> =
  sigh <adverb>
| portend like <object>
| die <adverb>
;

<adverb> =
  warily
| grumpily
;
```

The strings in brackets (<>) are the non-terminals. Each non-terminal definition is followed by a sequence of productions, separated by `'|'` characters, and with a `';'` at the end. Each production consists of a sequence of white-space separated terminals and non-terminals. A production may be empty so that a non-terminal can expand to nothing. There will always be whitespace surrounding the `'|'`, `'='`, and `';'` characters to make parsing easy.

Here are two possible poems generated by generating derivations for this grammar:

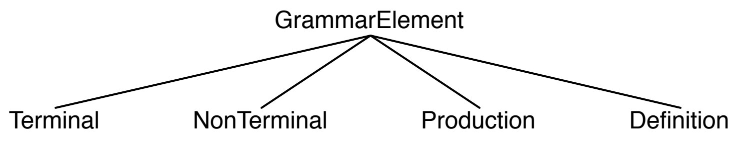
```
The big yellow flowers sigh warily tonight

The slugs portend like waves tonight
```

Your program will create a data structure to represent a grammar it reads in and then produce random derivations from it. Derivations will always begin with the non-terminal `<start>`. To expand a non-terminal, simply choose one of its productions from the grammar at random and then recursively expand each word in the production. For example:

```
   <start>
-> The <object> <verb> tonight
-> The big yellow flowers <verb> tonight
-> The big yellow flowers sigh <adverb> tonight
-> The big yellow flowers sigh warily tonight
```

System Architecture. A grammar consists of terminals, non-terminals, productions, and definitions. These four items have one thing in common: they can all be expanded into a random derivation for that part of a grammar. Thus, we will create classes organized in the following class hierarchy to store a grammar:



The abstract class `GrammarElement` provides the general interface to all pieces of a grammar. It is defined as follows:

```
abstract class GrammarElement {

  /**
   * Expand the grammar element as part of a random
   * derivation.  Use grammar to look up the definitions
   * of any non-terminals encountered during expansion.
   */
  def expand(grammar : Grammar) : String;

  /**
   * Return a string representation of this grammar element.
   * This is useful for debugging.  (Even though we inherit a
   * default version of toString() from the Object superclass,
   * I include it as an abstract method here to ensure that
   * all subclasses provide their own implmementaiton.)
   */
  def toString() : String;
}
```

The `Grammar` object passed into `expand` is used to look up the definitions for non-terminals during the expansion process, as described next.

The `Grammar` Class. A `Grammar` object maps non-terminal names to their definitions. At a minimum, your `Grammar` class should implement the following:

```
class Grammar {

  // add a new non-terminal, with the given definition
  def +=(nt : String, defn : Definition)

  // look up a non-terminal, and return the definition, or null
  // if not def exists.
  def apply(nt : String) : Definition

  // Expand the start symbol for the grammar.
  def expand() : String

  // return a String representation of this object.
  override def toString() : String
}
```

The `toString` method is useful for debugging.

Subclasses. The four subclasses of `GrammarElement` represent the different pieces of the grammar and describe how each part is expanded:

- `Terminal`: A terminal just stores a terminal string (like "`slugs`"), and a terminal `expand`s to itself.
- `NonTerminal`: A non-terminal stores a non-terminal string (like "`<start>`"). When a non-terminal `expand`s, it looks up the definition for its string and recursively expands that definition.
- `Production`: A production stores a list of `GrammarElement`s. To `expand`, a production simply expands each one of these elements.
- `Definition`: A definition stores a vector of `Production`s. A definition is expanded by picking a random `Production` from its vector and expanding that `Production`.

This design is an example of the Composite Design Pattern. The hierarchy of classes leads to an extensible design where no single `expand` method is more than a few lines long.

Implementation Steps.

(a) Download the starter code from the handouts web page. Once compiled with `scalac` (or `fsc`), you will run the program with a command like

```
scala RandomSentenceGenerator < Poem.g
```

You will need to use Scala's generic library classes. In particular, you will probably want to use both Lists and Maps from the standard Scala packages. The full documentation for these classes is accessible from the cs334 links web page.

(b) Begin by implementing the four subclasses of `GrammarElement`. Do not write `expand` yet, but complete the rest of the classes so that you can create and call `toString()` on them.

(c) The next step is to parse the input to your program and build the data structure representing the grammar in `RandomSentenceGenerator.scala`. The grammar will be stored in the instance variable `grammar`.

I have provided a skeleton of the parsing code. The parser uses a `java.util.Scanner` object to perform lexical analysis and break the input into individual tokens. I use the following two `Scanner` methods:

i. `String next()`: Removes the next token from the input stream and returns it.

ii. `boolean hasNext(String pattern)`: Returns true if and only if the next token in the input matches the given pattern. (If pattern is missing, this will return true if there are any tokens left in the input.)

When parsing the input, it is useful to keep in mind what form the input will have. In particular, we can write an EBNF grammar for the input to your program as follows:

```
<Grammar>     ::= [ Non-Terminal '=' <Definition> ';' ]*
<Definition>  ::= <Production> [ '|' <Production> ]*
<Production>  ::= [ <Word> ]*
```

where `Non-Terminal` is a non-terminal from the grammar being read and `Word` is any terminal or non-terminal from the grammar being read. Recall that the syntax `[ Word ]*` matches zero or more `Word`s.

The parsing code follows this definition with the following three methods:

```
protected def readGrammar(in : Scanner): Grammar
protected def readDefinition(in : Scanner): Definition
protected def readProduction(in : Scanner): Production
```

Modify these methods to create appropriate `Terminal`, `NonTerminal`, `Production`, `Definition`, and `Grammar` objects for the input. You may wish to print the objects you are creating as you go to ensure the grammar is being represented properly. You will need to complete the definition of `Grammar` at this point as well.

(d) Once the grammar can be properly created and printed, implement the `expand` methods for your `GrammarElement`s. Scala provides a random number generator that can be used as follows:

```
val number = Random.nextInt(N);  // number is in range [0,N-1].
```

Change `RandomSentenceGenerator` to create and print three random derivations after printing the grammar.

(e) You may also submit new a new grammar if you like. It can be as simple or complicated as you like. We may award points for creativity.

A few details about producing derivations:

- The grammar will always contain a `<start>` non-terminal to begin the expansion. It will not necessarily be the first definition in the file, but it will always be defined eventually. I have provided some error checking in the parsing code, but you may assume that the grammar files are otherwise syntactically correct.

- The one error condition you should catch reasonably is the case where a non-terminal is used but not defined. It is fine to catch this when expanding the grammar and encountering the undefined non-terminal rather than attempting to check the consistency of the entire grammar while reading it. The starter code contains a

  `RandomSentenceGenerator.fail(String msg)`

  method that you can call to report an error and stop.
- When generating the output, just print the terminals as you expand. Each terminal should be preceded by a space when printed, except the terminals that begin with punctuation like periods, comma, dashes, etc. You can use the `Character.isLetterOrDigit` method to check whether a character is punctuation mark. This rule about leading spaces is just a rough heuristic, because some punctuation (quotes for example) might look better with spaces. Don't worry about the minor details— we're looking for something simple that is right most of the time and it's okay if is little off for some cases.

Your finished program should be named `p2.scala`.

P3. (20 points) .... Scala Streams

A Scala `Stream` is like a `list` except that its elements are computed lazily. Owing to this fact, a `Stream` can be long... like infinitely long!

Just as you can create a list in Scala by using the cons operation,

```
val xs = 1 :: 2 :: 3 :: List.empty
```

you can also create a stream in Scala using the stream cons operation,

```
val ys = 1 #:: 2 #:: 3 #:: Stream.empty
```

For a `Stream[T]` the element on the left of `#::` must be of type `T` and the element on the right must be of type `Stream[T]`. Since a stream can be infinitely long, you can produce a `Stream` of infinite length by stream cons'ing an element to a call to a recursive function call. For example,

```
def numbers = {
  var n = 0
  def more : Stream[Int] = {
    n += 1
    n #:: more
  }
  0 #:: more
}
```

I can use the function as follows:

```
scala> numbers.head
res0: Int = 0

scala> numbers.tail.head
res1: Int = 1

scala> numbers.tail.tail.head
res2: Int = 2

scala> numbers.take(5).toList
```

```
res4: List[Int] = List(0, 1, 2, 3, 4)

scala> numbers.takeWhile(_ < 10).toList
res5: List[Int] = List(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

You may find the Scala `Stream` documentation to be a useful reference: `https://www.scala-lang.org/api/2.12.3/scala/collection/immutable/Stream.html`

For this question, you will modify your "Random Sentence Generator," using an infinite stream, so that it will continue to provide homework extension request excuses as long as the user presses the `Enter` key. When the user types `fine!`, the program prints `Thanks professor!!!` and terminates. Your program should do something like the following:

```
$ scala RandomSentenceStream Extension-request.g
I need an extension because my karma wasnt good last week
sooo...? > [Enter]
and as if that wasnt enough I lost my mind
sooo...? > [Enter]
and then I didnt know I was in this class
sooo...? > fine!
Thanks professor!!!
```

Your finished program should be named `p3.scala`.

P4. (15 points) .... Challenge Problem: Templated Lambdas in C++

This question asks you to alter your CPS conversion from the "Continuation Passing Style in C++" question. Polymorphism in C++ is achieved by using templates, which are a compile-time facility that generate code for a specific datatype while allowing the programmer to write code generically.

You may find the following definitions useful:

```
template <typename T>
using TContinuation = std::function<List<T>*(List<T>*)>;

template <typename T>
using Comparison = std::function<bool(T,T)>;
```

Your code should be able to sort the following list of C++ strings, `xs`:

```
string arr[] = { "the", "quick", "brown", "fox", "jumped",
                 "over", "the", "lazy", "dog" };
List<string> *xs = List<string>::init(arr, sizeof(arr)/(sizeof(*arr)));
```

Your finished program should be named `p4.cpp`.