Semester Project

COS 301: Programming Languages

Fall 2018

Introduction

This course has a semester-long individual project in which you will learn a new language and analyze it based on what you are learning in the course. The language will be one with which you are not familiar. Part of the learning process will be completing some simple programming assignments using the language.

Although there will be some programming, this is primarily a written project, where you will hone your writing skills. The project is divided into 6 parts. The first is just to choose a programming language, while the other 5 involve writing the parts of the final project paper:

- Part 2: Introduction – including an overview and history of the language
- Part 3: Syntax, scope, primitive data types, and operators
- Part 4: Data types, expressions, and assignment
- Part 5: Control constructs, subprograms, and recursion
- Part 6: Discussion and conclusion + integration into a final paper

Each of these parts will be substantial. The process for each part will be:

1. Write and turn in a draft of the part. Note that this should not be your *first* draft of the part; that one will be a rough draft that you will then revise, at least once, to create a polished draft to turn in.
2. Peer editing. During this, you will edit (usually 2) of your classmates' papers, while yours is being edited by (usually 2) classmates. Editing is an important skill in itself for both academic and industry jobs, and one with which this will give you practice.
3. Revise your draft based on comments from your peer editors and turn in the revised draft for grading.

The assigned and due dates are listed in this document as well as on the syllabus/web.[1] You do *not* have to wait until a part is assigned to begin working on it, but you may find it useful to get feedback from your peers on a prior part before beginning.

[1] Since these may change as the semester progresses, in the event of a discrepancy between the two, the one on the website will be the correct one.

Each part will be graded, although the preponderance of the grade for the written portion of the project will come from the grade given on the final draft. This means that you won't be penalized for a poor draft that you later clean up and improve. *However*, if you do not make a substantial effort on one or more of the parts, including the editing, your final grade will suffer substantially.

Part 1: Choosing a programming language

Assigned: 9/5
Due: 9/14

After reading Chapter 1 of the text, do some research on the web and select a language on which to base this project. The web sites listed in the syllabus are good starting points. Also select an alternate language in case your first selection is not approved. You must select languages with which you have no (or *very* little) experience. The languages should not be too specialized (e.g., PostScript, SQL), and they must have a real-world purpose (i.e., no joke/toy languages such as WhiteSpace, BF, etc.)

You are strongly encouraged to select a language that is so far removed from your current experience that it will be a true learning experience (i.e., "the weirder the better"). C and Java are *not* acceptable languages for this project, since they are covered in the curriculum.

Free compilers and/or interpreters are available for almost all languages; however, make sure that you are able to obtain what you need before committing to a language!

Turn in: For this assignment, submit a PDF or plain text file (no Word, etc., files) via Blackboard that answers these questions:

1. Which language is your first choice?
2. Which language is your alternate choice?
3. Why did you choose these languages? This should be a well-written paragraph or two explaining the rationale for your selection. (This and all written assignments in this course will be graded for style, grammar, and spelling.)

Part 2: Introduction

Assigned: 9/14
Initial draft due: 9/24
Editing: 9/24–9/28
Revised draft due: 10/5

Write the introduction for your paper.[2] This should give a overview of the language as well as its history.

Some points that might be covered in the above topics are when and where the language originated, major influences on the language, the purpose of the language, main features, unusual features, strengths and weaknesses, acceptance by the software community, future prospects (if any) or reasons for the lack of future prospects. Remember to support your opinions and statements with logical arguments and citations of other authors.

[2] Note that it is *very* likely that you will revise this introduction as you go through the semester based on the focus of your paper and what you learn about your language. That's fine and to be expected.

The length of this part of the paper is *somewhat* up to you. I would suggest you shoot for 5 or more pages, since you will want to cover the language in sufficient detail. If it is too short, and hence, too sketchy or superficial, then you may lose points. Oddly, the difficult part of writing this paper is to keep it short.[3] The goal is to summarize (and highlight the interesting points), not to provide a reference manual for the language or to unduly cover things that you will cover later in the paper.

[3] "I have made this longer than usual because I have not had time to make it shorter." –Blaise Pascal

Format and LaTeX

Your document needs to have standard margins of 1 inch on all sides, and it must include a title page that includes the title, your name, the date, and an abstract.

You must use the LaTeX document preparation system to format your project paper so that you gain familiarity with this valuable tool. This means that during this part of the project, you will have to learn LaTeX if you don't already know it. This is a good part in which to learn LaTeX, however, since there are unlikely to be many unusual text-formatting needs in an introduction.

If you have never used LaTeX before, you will need to install it on your computer or use an online tool such as ShareLaTeX (sharelatex.com). Versions are readily available for all three of the major operating systems (Windows, MacOS, and Linux). Although LaTeX is a markup language for documents, there are several free WYSIWYG LaTeX editors available as well, which can be useful for novices.

For information about LaTeX, including resources for installing and learning it, see the " LaTeX information" page under the "Course Documents" menu item on the course website.

The `article` document style is a reasonable one to use; correct margins can be obtained using the `\geometry` package. To see information about a package on a Unix-like system (e.g., MacOS, Linux, etc.), you can usually use the `texdoc` command, e.g.:

```
texdoc geometry
```

Bibliography

In this part of the project, you will begin developing an *annotated bibliography*. This will each work you reference in the body of your paper as well as notes about that reference, in particular, why the reference is useful. You will add to this bibliography throughout the other parts of the project, and with each part, you will turn in the entire bibliography up to that time. You should only include references to works you actually cite in the paper (either the current part or a prior part). Every bibliography entry *must* have an annotation describing it.

You will use the BibTEX program, which works in conjunction with LATEX, to process your references. This means you will use a `.bib` file in which to store bibliographic information about the references, and you will use the LATEX `\cite` macro (or some variant of it) to insert references in your paper. Documentation for BibTEX is available online and on the course website (on the " LATEX information" page).

For the citations and references, you may use one of the IEEE citation style, the APA citation style, or the natural sciences style. BibTEX has style files for most common citation/reference styles; for instance, the natural sciences style is implemented by the `natbib` BibTEX style file.

Programming assignment

Write a program in your language that interactively obtains two floating point numbers `x` and `y` from the user and computes and displays: the integer and fractional parts of each number; the sum; the difference; the product; and the quotient. Also try division by zero to find out how your language handles it. (For this, don't use any of the exception-handling facilities of the language other than its default mechanism.)

You do not need to be concerned with determining whether input is correct; you can assume that it is. Some languages may not be well suited to interactive input and the assignment may be modified for non-interactive input if necessary; ask me or the TA if you have questions.

Good heavens, we hope the language can do arithmetic!

If you are working with a web scripting language you can use HTML for input. An example form is given below.

Example HTML Page for Web Scripting Languages:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>COS 301 Fall 2016 Project 1</title>
  </head>
  <body>
    <h1>COS 301 Fall 2016 Project 1</h1>
    <form id="form1" name="form1" method="get" action="">
      <p>
Enter x:&nbsp;
<input type="text" id="textx" name="textx" style="text-align: right;" />
<br/>
Enter y:&nbsp;
<input type="text" id="texty" name="texty" style="text-align: right;" />
<br/>
<input type="submit" value="Submit" name="submit" id="submit" />
      </p>
    </form>
  </body>
</html>
```

Other issues

- Remember that the suggested page count doesn't include the cover page or the bibliography.
- Remember to review the paper format checklist on the website before submitting!
- An amusing and interesting website for programming languages is the "99 Bottles of Beer" site, www.99-bottles-of-beer.net, which has the song implemented in 1500 different programming languages. For popular languages you can usually find at least half-a-dozen different programs varying widely in style. In particular, see the "bottled" version of the Perl program.

Turn in via Blackboard

1. A **PDF** of your paper—this is the only format acceptable! See the instructions on Blackboard for how you must name your file.
2. A zipfile or gzip'd tarfile of your LaTeX and BibTeX files.
3. Your program and its output.

Part 3: Syntax, scope, primitive data types, and operators

Assigned: 10/5
Initial draft due: 10/15
Editing: 10/15–10/19
Revised draft due: 10/26

In this part of the project, you will be learning and describing fundamental parts of your language. This section of the document will discuss:

- basic syntactic structure, including statement terminators or separators, block structure, syntactic peculiarities, etc.;
- units or levels of scope and the nature and type (runtime or compile-time) of name bindings within the different levels of scope;
- primitive data types available, including any range limitations, etc.; and
- operators for primitive data types and their precedence and associativity.

For this part of the project, a "primitive" data type is one that is not created from other data types. What is primitive in one language may not be primitive in another. For example, many languages will include strings as a primitive data type; however, some languages, such as C, do not (since they are defined as arrays of characters).

Advice on the length of this part of the paper is the same as for Part 2.

Bibliography

You will continue developing your annotated bibliography. This will include works you reference in the body of your paper as well as notes about those references. This bibliography will be added to throughout the other parts of the project, and with each part, you will turn in the entire bibliography up to that time. You should only include references to works you actually cite in the paper (either the current part or a prior part). Every bibliography entry *must* have an annotation describing it.

Programming assignment

Using your language, write a lexical analyzer ("lexer", scanner) and a recursive descent parser for a small grammar. The recursive descent parser will simply be a recognizer that determines whether or not a string is in the language; it will *not* construct an actual parse tree. Note that the calling sequence corresponds exactly to the parse tree. Output should be similar to the output from the example parser in the text (see below).

The grammar for Boolean expressions. Note that `||` is a token in the language (i.e., meaning logical OR), while `|` is a symbol used in the grammar specification denoting alternatives. E.g.,

```
a == 3 & x == 4 || y == 3
```

would be a valid string in the language, meaning "a is equal to 3 and x is equal to 4 or y is equal to 3".

```
 <bool_expr> ::= <and_term> { || <and_term> }
<and_term> ::= <bool_factor> { & <bool_factor> }
<bool_factor> ::= <bool_literal> | !<bool_factor> |
                ( <bool_expr> ) | <relation_expr>
<relation_expr> ::= <id> { <relop> <id> }
<id> ::= letter { letter | digit }
<bool_literal> ::= true | false
```

Constructing the lexical analyzer is actually the more difficult part of this assignment. You can always construct the parser first by creating an input stream of tokens (e.g., 1 = Left Paren, 2 = Right Paren, etc); then create a lexer that simply returns the next integer token. Then worry about actually parsing characters in a "program" later. Include output for the following expressions in an appendix:

```
foo & !( a2 > bar & w < foo || x < y)
A1 & B1 || A2 & B1 || (! C || A <> B )
```

Example output from the Sebesta parser (see the text):

```
Next token is: 25 lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 11 lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 21 lexeme is +
Exit <factor>
Exit <term>
Next token is: 10 lexeme is 47
Enter <term>
Enter <factor>
```

Turn in via Blackboard

1. A **PDF** of your entire paper up to this point, i.e., the introduction and this part. Only this part will be edited or graded, but having

the information in the first part will make editing and grading easier. PDF is the only format acceptable. See the instructions on Blackboard for how you must name your file.

2. A zipfile or gzip'd tarfile of your LaTeX and BibTeX files.
3. Your program and its output.

Part 4: Data types, expressions, assignment

Assigned: 10/26
Initial draft due: 11/5
Editing: 11/5-11/9
Revised draft due: 11/16

Write a paper that addresses the following topics for your selected language:

1. Data types available **in the language** beyond primitive data types (such as ints, floats, etc.). This could include arrays, vectors, structures, strings, objects, and classes. You should include higher-level structures, such as hashs, stacks, etc., if they are part of the language itself and not just part of a library.
2. Summary of standard libraries or classes that extend the language's type system.
3. Semantics of expression evaluation.
4. Coercions/implicit type conversions that are performed in expression evaluation.
5. Semantics of assignment statements, if unusual.

The same comments about paper length as for Part 2 apply here as well.

Programming

Write a function that accepts an HTML document (a string) and a keyword (also a string). The function will find all occurrences of the keyword in the HTML string after the `<body>` element unless the keyword appears within an HTML tag, then surround the string found with tags to "highlight" the keyword. For example,

```
<span style="background-color: blue; color: white">keyword</span>
```

You will have to be careful not to highlight strings occurring within an HTML tag. For example, if the keyword is "table", you wouldn't want to mark up this:

```
<table width="100%" border="0">
```

The result would hardly be valid HTML!

There are at least two approaches to this problem:

1. String processing. Treat the HTML as one long string and program a lexer that returns the next chunk, which would either be a tag, text, or a comment. (An HTML comment looks like this: `<!--hi there-->`.) If it is a tag or a comment, then just append it to the output. If it is text, then search the string for the keyword and apply the new markup if found.

   This may not be the best way to approach the problem. HTML and XHTML can be quite complex to parse. In addition, if we were *really* doing this right, the text between certain tag pairs would need to be skipped, for example, `<script></script>`, etc.;

we won't worry about that for this assignment, however. The approach also treats one abstraction (HTML) as another (a string), which may not be the best approach.

If you *do* use the string approach, don't start within the `<head>` tag, but start searching after the `<body>` tag.

2. DOM document processing. An HTML document is handled by browsers and other document processors by transforming the text into a tree structure defined by the W3C Document Object Model (DOM). For our purposes, each node in a DOM document is either an HTML element, a comment, a script, or text. (We'll ignore other things, like CDATA nodes.) To handle string markup, traverse the DOM tree and check the text nodes. You can also use an XPath expression to do actual string search and return a set of nodes. Either way, if you find a string to mark up, remove the text node, then create and append a new subtree (left text, markup with child text, and right text).

   Although this sounds difficult, it's actually pretty easy to do with the DOM parser libraries that can be obtained for virtually any language. Just search the Web for "[language name] DOM parser".

You can use either approach. If you are not familiar with HTML and the DOM, then the string processing method is likely easier in the short run. However, the DOM approach is much easier in general, and if you make the effort to understand the DOM, you will acquire a foundation of skills that will be useful for a long time to come.

Write a program that demonstrates and tests your function. If your language is a web scripting language, you can simply use a form that posts to your program, which will then output the HTML back to the browser. Or you can write a program that reads data from an HTML file and outputs the revised HTML to a different file.

Turn in via Blackboard

1. A **PDF** of your entire paper up to this point, i.e., the introduction and this part. Only this part will be edited or graded, but having the information in the first part will make editing and grading easier. PDF is the only format acceptable. See the instructions on Blackboard for how you must name your file.
2. A zipfile or gzip'd tarfile of your LaTeX and BibTeX files.
3. Your program and its output.

Part 5: Control constructs, subprograms, and recursion

Assigned: 11/16
Initial draft due: 11/26
Editing: 11/26–11/30
Revised draft due: 12/7

In this part of the project, you will write about: (1) control flow constructs in your language (conditionals, loops, gotos, etc.); (2) how subprograms are defined and called, including scoping rules, parameter passing, recursion, and whether subprograms themselves can be passed as parameters; and (3) how recursion is done in the language, if it is.

Programming

Write two versions of a non-trivial program in your language, one using iteration and the other using recursion. What is "non-trivial", you ask? That's up to you to figure out! (I think you have a pretty good idea.)

Turn in via Blackboard

1. A **PDF** of your entire paper up to this point, i.e., the introduction and this part. Only this part will be edited or graded, but having the information in the first part will make editing and grading easier. PDF is the only format acceptable. See the instructions on Blackboard for how you must name your file.
2. A zipfile or gzip'd tarfile of your LaTeX and BibTeX files.
3. Your program and its output.

Part 6: Discussion and Conclusion + Integration

Assigned: 12/7
Editing: None for this one
PROJECT FINAL DRAFT DUE: 12/14

The part of the paper written here will a discussion and conclusion section. It should present and argue for *your* impressions and evaluation of the language—e.g., is it a good language for general-purpose programming? Is it a specialty language, and if so, is it good for the specialty? Who should/shouldn't use the language? Is it easy to learn? Easy to use? What are its strengths and limitations? Etc.

In this part of the project, you will also combine all prior parts into a coherent paper about your language. This is a chance to redeem yourself on prior work if you didn't get a grade you liked. You will turn in your entire project as a single paper. Your complete annotated bibliography will be at the end. Any appendices (other than code) will come after the bibliography.

You may either just reuse what you turned in before with no changes to the writing, **or** you may edit what you turned in and improve it. You should let me know on the title page which you've done, either a note saying "as is" or "edited" for each of the parts 1–4, where "as is" means there have been at most minor changes since the part was graded by me.

Programming

There is no programming for this section. However, you need to turn in all prior programs (but not their output) as a single, well-formatted and easy to navigate appendix to the paper.

Turn in via Blackboard

1. A **PDF** of your entire paper. PDF is the only format acceptable. See the instructions on Blackboard for how you must name your file.
2. A zipfile or gzip'd tarfile of your LaTeX and BibTeX files.