GLASS

Milestone Three Progress Report

Team: Tommy Galletta, Alexander Lockard

Faculty Advisor: Dr. Stansifer

Milestone Three Task Matrix

Task	Completion %	Tommy	Xander	Todo
Syntax specification file interpretation	100%	85%	15%	
Basic macro interpretation / XML manipulation	50%	10%	40%	Implement basic macros. Continue implementation of macro interpreter.
Continued research of parser generators	100%	50%	50%	
Begin documentation	90%	45%	45%	Find a way to easily put documentation on website

Faculty Advisor Feedback

- Advisor continues to be happy with the progress made on the project.
- Advisor decided it would be best for us to ditch XML as an intermediate output.
    - We plan to still allow for XML-ized output to be generated via the macro system.
- Advisor sees the separation of the grammar (syntax specification) and the “piggybacking” (macro system) to be the biggest novel feature of our project.
- Advisor hopes we will continue working on the project over the summer (that is our plan).

Task Discussion (continued)

Begin Documentation

- The beginnings of our documentation have been created
- Currently within a Google Doc
  - We plan to transfer all of the contents to be hosted on our project website by the end of the next milestone.

Token regex is simply a regex pattern that will be used to identify whether or not some input matches a given token.

For example, in the following token declaration:

```
visible A_SEQUENCE = /a+/
```

States that any string that contains a series of a's (i.e. `aaaaa`) could be considered to be an `A_SEQUENCE` token.

Alternatively, tokens of the same type (`visible` or `invisible`) can be declared within a block, where the type only needs to be typed once, as shown below:

```
visible {
     FRUIT_ONE: /apple/
     FRUIT_TWO: /banana/
     FRUIT_THREE: /cherry/
}
```

Task Discussion

Syntax Specification File Interpretation

- Fully implemented
- Built off of the parser generator system
- Demo later on

```
while (nodeStack.size() > 0) {
    ParserNode currentNode = nodeStack.removeFirst();
    String currentName = currentNode.getName();

    if (currentName.equals("VISIBLE")) currentTokenType = TokenType.VISIBLE;
    if (currentName.equals("INVISIBLE")) currentTokenType = TokenType.INVISIBLE;
    if (currentName.equals("TOKEN_DATA")) currentIdentifierUse = IdentifierUse.TOKEN_NAME;
    if (currentName.equals("PARENT_PRODUCTION")) {
        if (currentIdentifierUse == IdentifierUse.PRODUCTION_RIGHT) {
            Production newProd = new Production(currentProductionLeft, currentProductionRight,
                                                generatedLexer, currentProductionID);
            generatedParser.addProduction(newProd);
            definedProductions.add(newProd);
            currentProductionRight.clear();
        }
        currentIdentifierUse = IdentifierUse.PRODUCTION_LEFT;
        currentProductionRight = new ArrayList<String>();
    }
    if (currentName.equals("TERM_NONTERM_LIST")) currentIdentifierUse = IdentifierUse.PRODUCTION_RIGHT;

    if (currentName.equals("IDENTIFIER")) {
        String nodeContents = ((TerminalNode) currentNode).getSequence();
        if (currentIdentifierUse == IdentifierUse.TOKEN_NAME)
            currentTokenName = nodeContents;
        else if (currentIdentifierUse == IdentifierUse.PRODUCTION_LEFT)
            currentProductionLeft = nodeContents;
        else if (currentIdentifierUse == IdentifierUse.PRODUCTION_RIGHT) {
            currentProductionRight.add(nodeContents);
        }
    }
```

Task Discussion (continued)

Basic macro interpretation

- Solidified “alpha” macro specification file grammar
- Started work on macro specification file interpreter
- Discussed basic macro ideas

XML has been mostly scrapped for now

```
path {
    .replace(regex, "new")
    .replace("new")
}

F IDENTIFIER {
    a = .find(regex)
    a + 5
    .replace(a, "new")
}

E E' T T' F E {
    a = .evaluate("T T' STAR")
    if a:
        macroC(E E' T T' F)
        macroC(this.parent.path)
}

macroC() {
    .replace("new")
}
```

Task Discussion (continued)

Continued research of parser generators

- Brief Investigation of JQ complete (not a parser generator, more like a macro system)
- Brief Investigation of Visual BNF complete (not a parser generator, but useful for inspiration for our GUI)
- We will likely continue with a few more investigations

Demo Time!

Milestone Four Plan

Task	Tommy	Xander
Add options and ease of use features	Implement debugging features	Add settings for syntax specification files
GUI-based syntax specification system	Debug and assist with creating GUI system	Implement main GUI system
Continue writing documentation and host it on project website	Proofreading/editing documentation, getting documentation on web	Writing documentation
Continued research of other tools	Investigate 1-2 parser generator tools	Investigate 1-2 parser generator tools
Debugging currently implemented systems	Both team members will work to debug all implemented features	

Disclaimer: Some, if not most of these tasks will be tackled during the summer. The actual Milestone Four plan come fall will likely be different.

Discussion of Planned Tasks

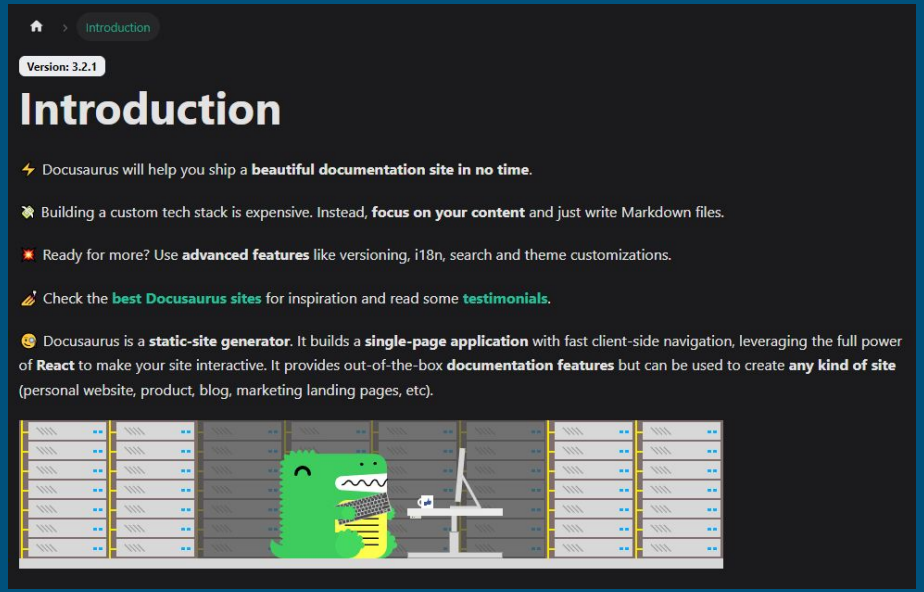
Add Options and Ease Of Use Features

- To help with making our tool easier to use in “common” cases, we will included certain features which are enabled by default
    - These features will be able to be enabled or disabled at will via the syntax specification file
- An optional debug argument will be added to GLASS, allowing for a debug report to be written out when the program is run

Discussion of Planned Tasks

Continue writing documentation and host it on project website

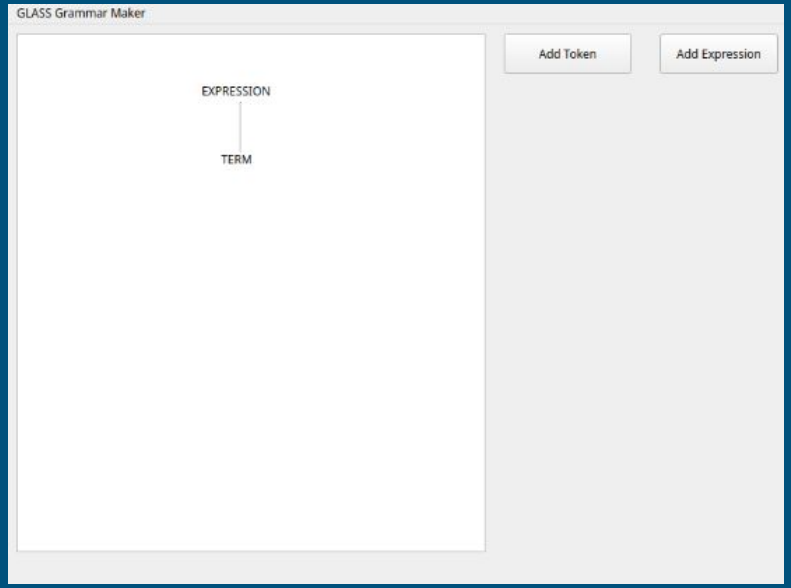
- We want to work continuously to ensure that our all features of our tool are well documented
- We will find a tool (potential option on the right) that will allow us to easily transfer our Google Doc documentation to our website



Discussion of Planned Tasks (continued)

GUI-based Syntax Specification System

- To allow for easier grammar creation, we will create a GUI based tool with drag-and-drop style features to create productions by joining together “symbol nodes”
- After the user has created a grammar, a syntax specification file containing their grammar will be generated



Discussion of Planned Tasks (continued)

Continued research of other tools

- Same as last milestone, we feel that this continued research will help us ensure that we are implementing the right features while also maintaining a level of simplicity and avoiding confusing notation.

Debugging currently implemented systems

- Simply put, we don’t want our program to unexpectedly crash. Thorough testing will be done to ensure that our program rarely crashes, and in the event that it does, some useful error message is printed to the user.

Questions?