Flow++

Introduction

The purpose of this project is to design and implement an application oriented programming language able to interpret commands to produce flowcharts.
As a novice programmer it is desired to be able to easily create flowcharts. Flow++ aims to help novice programmers in their first steps towards the programming world. All basic flowchart features are included and with simple commands you can create, edit and view a flowchart.

Language development

The Flow++ language is composed of a lexical analyzer, a parser and the intermediate code. The lexical analyzer takes the code and divides it in tokens. If there are any invalid tokens the lexical analyzer produces an error and terminates. If there are no errors the tokens are then passed to the parser. The parser verifies if the code, divided in tokens, follows the grammar rules of the language. If the parser encounters any grammatical errors it produces an appropriate error message, for example if a parenthesis is missing or a command is missing a parameter it produces a missing token error. On the contrary if the grammar is correct the Flow++ code is translated to a corresponding java code.
The lexical analyzer was developed in the NetBeans development environment. The parser was created in the Windows command line using the JavaCC tool, which generates a parser based on a grammar specification. The intermediate code was developed in the Eclipse environment. The code was shared through a Github repository among team members.
To test the lexical analyzer a code generator was created to generate 2,000 lexemes to be fed to the lexical analyzer. The parser was tested by creating various test cases, ones which followed the grammar rules and used most of the commands and other that did not follow the grammar rules. Some of the cases that did not follow the grammar rules were missing parenthesis, incorrect number of parameters, missing quotes, missing initial Genesis command, additional newlines.
To test the translator a user interface was created with a console to display errors and four codes were developed with different commands for testing. All the codes consisted on displaying the flowchart to see graphically if the commands worked, if the variables were translated to their respective value and to check if the flowcharts were being positioned properly. The first code used was:

```
Genesis(Test)
a='x=10'
b='a=input'
c='input<x'
d='y=0'
e='y++'
```

```
g='y==10'
Insert(a,Start,d)
Insert(d,a,b)
Insert(b,d,c)
InsertIf(c,b,e,b)
Insert(e,c,g)
Insert(g,e,End)
ShowItToMe(Test)
```

This code tested if the commands produced the expected shape, if they pointed to the correct value and if the ShowItToMe worked and opened a new window. It also checked if a single loop was possible without causing errors. The next code used is the same but with the Fatality command included.

```
Genesis(Test2)
a='x=10'
b='a=input'
c='input<x'
d='y=0'
e='y++'
g='y==10'
Insert(a,Start,d)
Insert(d,a,b)
Insert(b,d,c)
InsertIf(c,b,e,b)
Insert(e,c,g)
Insert(g,e,End)
Fatality(Test2)
ShowItToMe(Test2)
```

This was used to see if Fatality actually destroyed the flowchart and all it components. When attempting to use ShowItToMe after Fatality the flowchart should not be able to appear.

```
Genesis(Test3)
a='x=10'
b='a=input'
c='input<x'
d='y=0'
e='y++'
g='y==10'
Insert(a,Start,d)
Insert(d,a,b)
Insert(b,d,c)
InsertIf(c,b,e,b)
```

```
Insert(e,c,g)
Insert(g,e,End)
Smash(c)
GetOverHere(b,e)
ShowItToMe(Test3)
```

The code above is the same as the first but this time Smash and GetOverHere are used. This was to test if Smash did delete the node given and if GetOverHere changed the next value of the given node.

```
Genesis(TestIf)
a='x==0'
b='x<0'
c='o=0'
d='p=0'
e='u=0'
InsertIf(a,Start,c,b)
Insert(c,a,End)
InsertIf(b,a,d,e)
Insert(d,b,End)
Insert(e,b,End)
ShowItToMe(TestIf)
Genesis(IFT)
a='x==0'
b='x<0'
c='o=0'
d='p=0'
e='u=0'
f='u>o'
InsertIf(a,Start,c,b)
Insert(c,a,f)
InsertIf(f,c,e,End)
InsertIf(b,a,d,e)
Insert(d,b,End)
Insert(e,b,End)
ShowItToMe(IFT)
```

This final code with two flowcharts was used to see how well it translated when an if pointed to another if. It also tested how it would respond to two nodes that pointed to the same node and if the arrows or shapes would overlap. Additionally it tested if two flowcharts could be created in the same code.

Conclusion

In this project we managed to design and implement a programming language able to produce flowcharts. In the process of creating the Flow++ language we learned how tools like JavaCC are very helpful for creating parsers. From specifications in BNF JavaCC creates a parser complete with error messages. Through the course of the project the grammar had to change so it could better represent our vision of what the language could do, for example create multiple flowcharts in one program. Also some tokens were added to help structure how the language should be written, such as the newline token to limit one statement per line. With Flow++ the user can create, edit and view one or more flowcharts that contain the basic flowchart elements utilizing simple commands.