Compiler Project

The Kool-Krab

Zack Nicholson

Thomas Widjaja

Leonardo Pangco

Alex Gomez

Ronald Mak

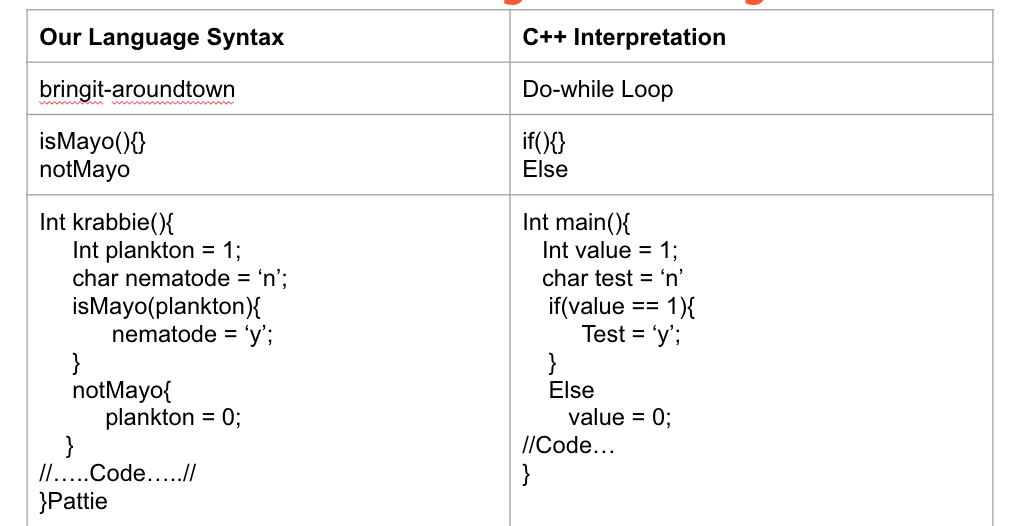
CMPE 152 Fall 2018

**Introduction**

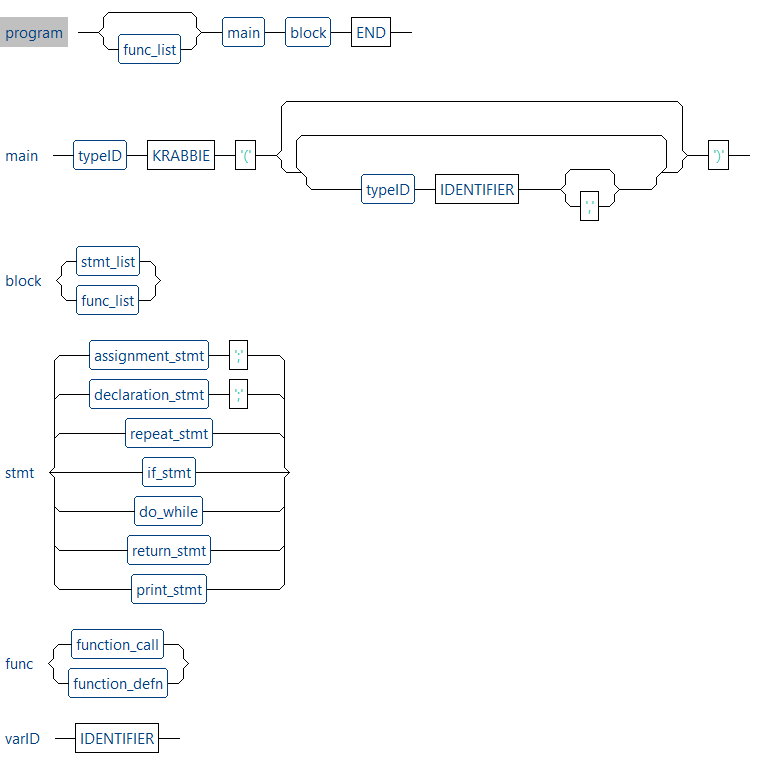
The goals of this term project for the compilers course was to create a unique programming language as well as to create a compiler for that created language; a culmination of all the concepts that were taught in this course. As a team we decided that created a simpler version of the C++ programming language that we were all used to was the first step in creating our unique language. As ideas progressed, we also wanted to have a little fun and be creative leading us to make the language a homage to Spongebob Squarepants where many key terms in the language would be references to the hit television series. In the making of the language and its compiler some of the new softwares introduced in the course would also be utilized such as ANTLR-4 to ease the workload in front-end design as it generated syntax diagrams and parse trees in real time as well as Jasmin assembly language to assist with the back-end. The project provided us with a better understanding of how a programming language is created and gave us a greater appreciation of the impact compilers have in coding. In this report, information about the Kool-Krab programming language and compiler, how the compiler was built and ran, and sample programs as well as outputs will be discussed.

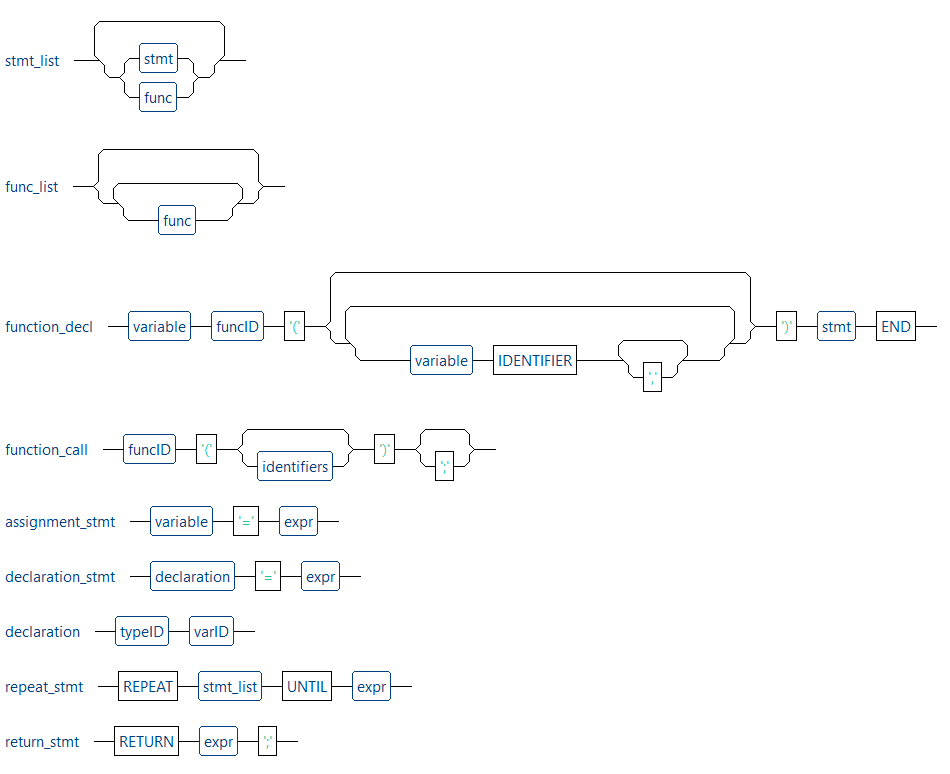
**Features**

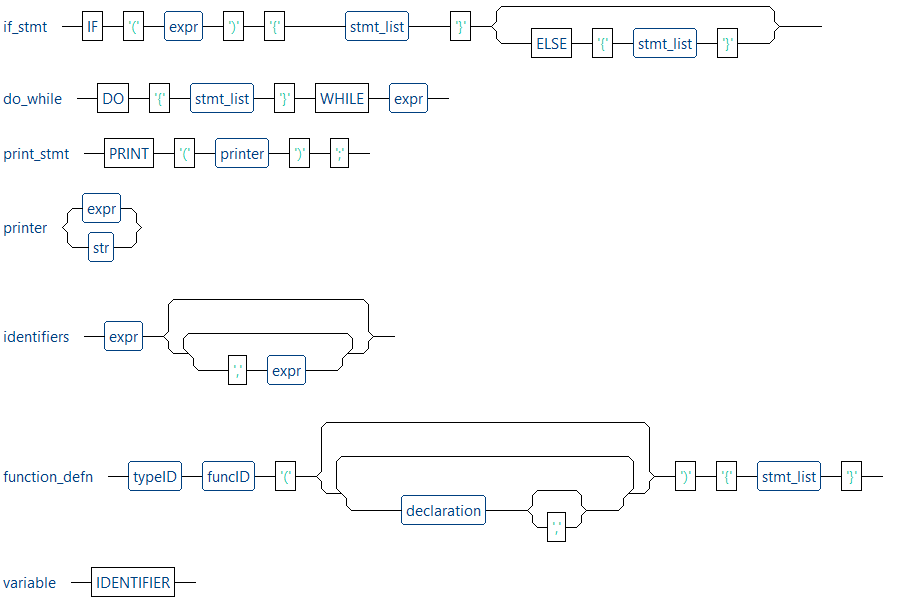
The language is comprised as a simpler version of C++. The language successfully computes basic signed/unsigned arithmetic, handles boolean conditions via if conditions and do-while loop conditions. However, as an homage to the great animated series Spongebob Squarepants, the syntax includes references to the show which have correlations to C++ syntax. The table below shows some of the examples of our language parallel to the C++ interpretation.

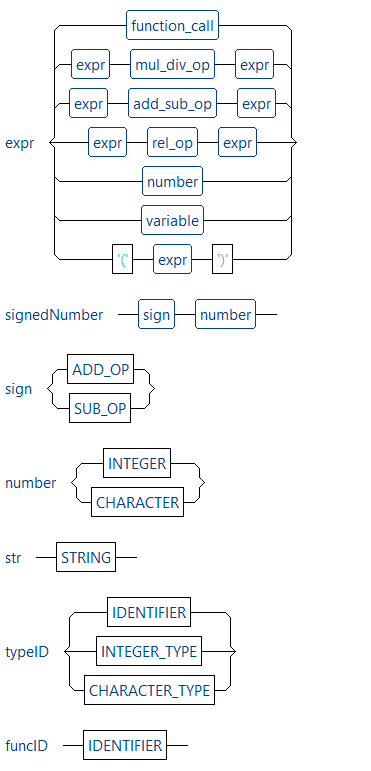
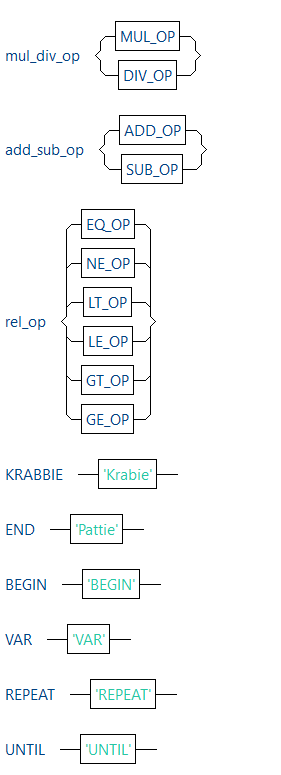
****

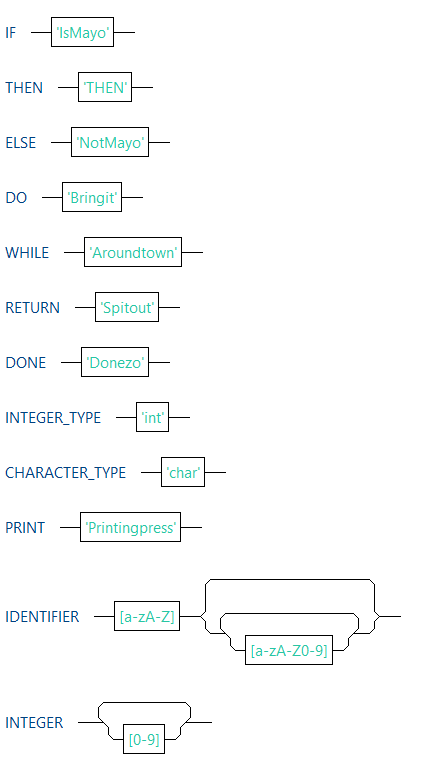
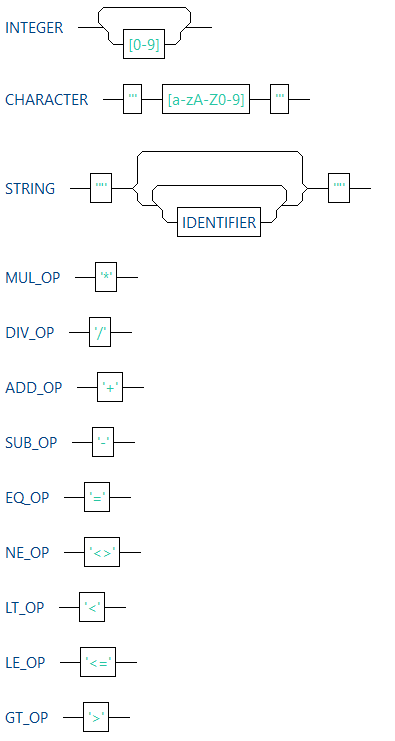
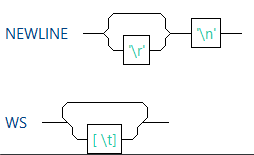
**Syntax Diagrams**

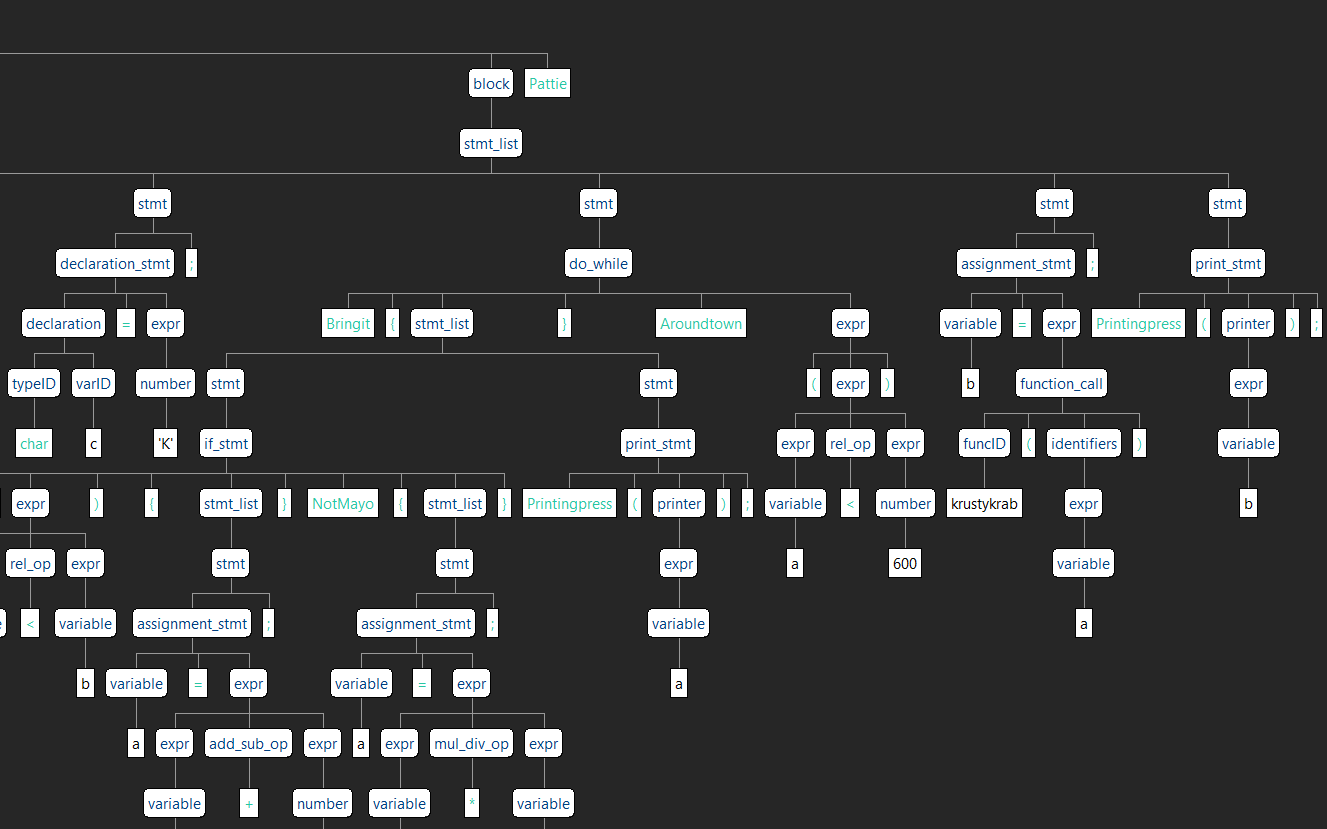
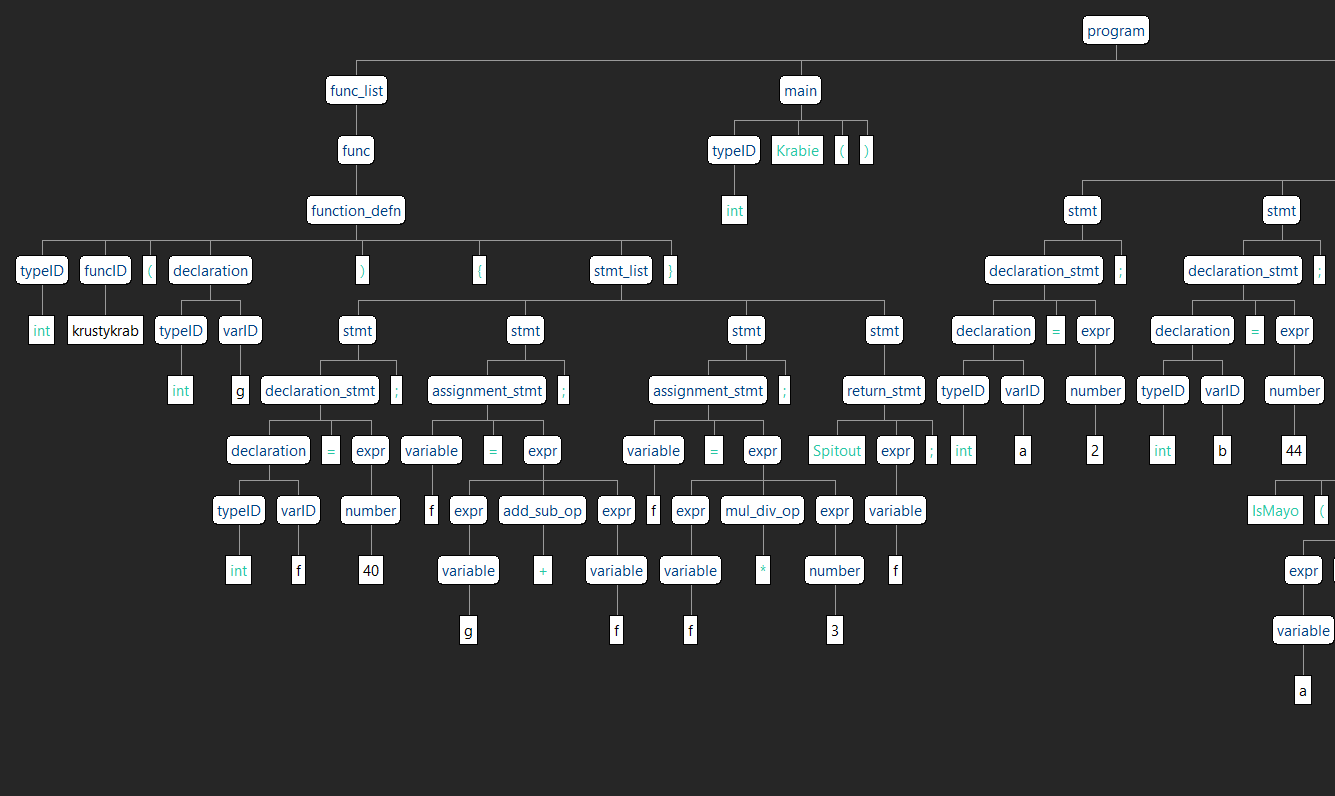
****

****

****

****

****

**Parse Trees**

**Code Templates**

|  |
| --- |
| **Printingpress()** |
| ; Printingpress(a);  getstatic Krabbie/a I  putstatic Krabbie/a I  getstatic java/lang/System/out Ljava/io/PrintStream;  ldc "a = %d\n"  iconst\_1  anewarray java/lang/Object  dup  iconst\_0  getstatic Krabbie/a I  invokestatic java/lang/Integer.valueOf(I)Ljava/lang/Integer;  aastore  invokestatic java/lang/String.format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;  invokevirtual java/io/PrintStream.print(Ljava/lang/String;)V  getstatic Krabbie/a I |

|  |
| --- |
| **Jasmin Output for Function Declaration** |
| goto krustykrab\_end  krustykrab:  astore\_1  ; intf=40;  ldc 40  putstatic Krabbie/krustykrab\_f I  ; f=g+f;  getstatic Krabbie/krustykrab\_g I  getstatic Krabbie/krustykrab\_f I  iadd  putstatic Krabbie/krustykrab\_f I  ; f=f\*3;  getstatic Krabbie/krustykrab\_f I  ldc 3  imul  putstatic Krabbie/krustykrab\_f I  ; Spitoutf;  getstatic Krabbie/krustykrab\_f I  ret 1  krustykrab\_end: |

|  |
| --- |
| **Jamin Output from Function Call w/ return to assignment statement** |
| ; b=krustykrab(a);  getstatic Krabbie/a I  putstatic Krabbie/krustykrab\_g I  jsr krustykrab  putstatic Krabbie/b I |

|  |
| --- |
| **Jasmin Output for “If Statement”** |
| ; IsMayo(a<b){a=a+2;}NotMayo{a=a\*b;}  Label\_1:  getstatic Krabbie/a I  getstatic Krabbie/b I  if\_icmplt Label\_2  ; a=a\*b;  getstatic Krabbie/a I  getstatic Krabbie/b I  imul  putstatic Krabbie/a I  goto Label\_3  Label\_2:  ; a=a+2;  getstatic Krabbie/a I  ldc 2  iadd  putstatic Krabbie/a I  Label\_3: |

|  |
| --- |
| **Jasmin Output for “Do While Statement”** |
| ; Bringit{a=a+2;Printingpress(a);}Aroundtown(a<30)  Label\_0:  ; a=a+2;  getstatic Krabbie/a I  ldc 2  iadd  putstatic Krabbie/a I  ; Printingpress(a);  getstatic Krabbie/a I  putstatic Krabbie/a I  getstatic java/lang/System/out Ljava/io/PrintStream;  ldc "a = %d\n"  iconst\_1  anewarray java/lang/Object  dup  iconst\_0  getstatic Krabbie/a I  invokestatic java/lang/Integer.valueOf(I)Ljava/lang/Integer;  aastore  invokestatic java/lang/String.format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;  invokevirtual java/io/PrintStream.print(Ljava/lang/String;)V  getstatic Krabbie/a I  ldc 30  if\_icmplt Label\_0 |

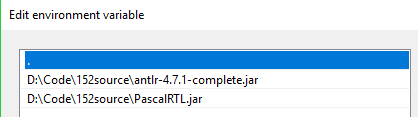
|  |
| --- |
| **Jasmin Output for loading Char into global** |
| ; charc='K';  ldc 75  putstatic Krabbie/c C |

**Instructions on how to build compiler**

1. Download Eclipse, boost, Antlr4.jar, Antlr4 zip, and Jasmin.jar
2. Setup Eclipse
   1. Create a new C++ project and drop all the files in our zip file into it.
   2. Go to project properties/Build/Settings. Change the Dialect to C++11. Include the path where all the antlr4 header files are. In my case it was *"D:\Code\152source\antlr4-4.7.1\runtime\Cpp\runtime\src"*. Also include the location of your boost headers.
   3. Under C++ Linker/Libraries include *antlr4-runtime*. In the library search path include the location of your .a file generated from the antlr4 zip file.
   4. Build the project.
   5. Now that an executable has been generated, under runtime configurations/arguments, type *input.bob*, which is the included source file written in our language.
3. Run the project.
4. Once the project is ran a file called *Krabbie.j* should have been generated (the file name is hardcoded in our visitors).
5. Assuming Jasmin is setup properly on your end, run the command *jasmin Krabbie.j*. A class file *Krabbie.class* should be generated. To run this command we setup a batch file jasmin.bat to run the following command *java -jar D:\Code\152source\jasmin-2.4\jasmin.jar %1 %2 %3 %4 %5*.
6. If you have any issues some of the screenshots in *Figure 1* - *Figure 3* may help.

**Instructions on how to run compiler**

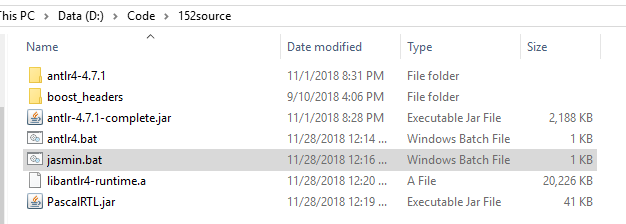
1. Once a class file has been successfully generated pass it to the JVM and run it. We set up some environment variables to allow us to run the class file by issuing the command *java Krabbie*. I’ve included some screenshots of
2. An output should appear in the terminal that shows the value of variable *a* as it goes through our do-while (*Bringit Aroundtown*), as well as the value of variable *b* after it was passed into a function. This functionality will make more sense if you look at *Figure 1* below.
3. If you have any issues some of the screenshots in *Figure 1* - *Figure 3* may help.



*Figure 1*: CLASSPATH Environment Variable



*Figure 2*: Path Environment Variable



*Figure 3*: Dependencies Directory

**Sample source programs written in our language**

|  |
| --- |
| **Input.bob** |
| int krustykrab(int g)  {  int f=40;  f=g+f;  f=f\*3;  Spitout f;  }  int Krabie()  int a=2;  int b=44;  char c='K';    Bringit {  IsMayo (a<b){  a=a+2;  }  NotMayo {a=a\*b;}  Printingpress(a);  }    Aroundtown (a<600)  b=krustykrab(a);  Printingpress(b);  Pattie |

|  |
| --- |
| **input2.bob** |
| int Krabie(int g)  int bob = 0;  int pat = 12;  char krabs='K';  char plankton = 'P';    IsMayo(krabs == plankton){  pat = bob;  }  NotMayo{bob = pat;}  Printingpress(bob);  Printingpress("Thank you for visiting Bikini Bottom");  Pattie |

**Sample Output**

|  |
| --- |
| **Input.bob Compiler Output** |
| === Pass1Visitor(): symtab stack initialized.  === visit Program:  === visit Function Definition: intkrustykrab(intg){intf=40;f=g+f;f=f\*3;Spitoutf;}  === visit Function ID: krustykrab  === visit Type Id: int  === visit Declaration: intg  === visit Type Id: int  === visit Var Id: g  === visit Declaration: intf  === visit Type Id: int  === visit Var Id: f  === visitUnsignedNumberExpr: 40  === visit Integer Constant: 40  === visit Add/Subtract Expression: g+f  === visit Variable Expression: g  === visit Variable Expression: f  === visit Multiply/Divide Expression: f\*3  === visit Variable Expression: f  === visitUnsignedNumberExpr: 3  === visit Integer Constant: 3  === visit Variable Expression: f  === visit Type Id: int  === visit Block:  === visit Declaration: inta  === visit Type Id: int  === visit Var Id: a  === visitUnsignedNumberExpr: 2  === visit Integer Constant: 2  === visit Declaration: intb  === visit Type Id: int  === visit Var Id: b  === visitUnsignedNumberExpr: 44  === visit Integer Constant: 44  === visit Declaration: charc  === visit Type Id: char  === visit Var Id: c  === visitUnsignedNumberExpr: 'K'  === visit Character Constant: 'K'  === visit Relation Op Expression: a<b  === visit Variable Expression: a  === visit Variable Expression: b  === visit Add/Subtract Expression: a+2  === visit Variable Expression: a  === visitUnsignedNumberExpr: 2  === visit Integer Constant: 2  === visit Multiply/Divide Expression: a\*b  === visit Variable Expression: a  === visit Variable Expression: b  === visit Variable Expression: a  === visit Paren Expression (a<600)  === visit Relation Op Expression: a<600  === visit Variable Expression: a  === visitUnsignedNumberExpr: 600  === visit Integer Constant: 600  === visitFuncCallExpr: krustykrab(a)  === visit Function ID: krustykrab  === visit Variable Expression: a  === visit Variable Expression: b  === visitProgram: Printing xref table.  ===== CROSS-REFERENCE TABLE =====  \*\*\* PROGRAM Krabbie \*\*\*  Identifier Line numbers Type specification  ---------- ------------ ------------------  a  Defined as: variable  Scope nesting level: 1  Type form = scalar, Type id = integer  b  Defined as: variable  Scope nesting level: 1  Type form = scalar, Type id = integer  c  Defined as: variable  Scope nesting level: 1  Type form = scalar, Type id = char  krustykrab  Defined as: FUNCTION  Scope nesting level: 1  krustykrab\_f  Defined as: variable  Scope nesting level: 1  Type form = scalar, Type id = integer  krustykrab\_g  Defined as: variable  Scope nesting level: 1  Type form = scalar, Type id = integer  ||==================================================||  ||=====================Pass 2=======================||  === visit Function Definition: krustykrab  === visit Declaration: intf=40  === Visit Assignment Statement  === Visit Add/Subtract Expression: g+f  === visiVariableExpr  === visiVariableExpr  === Visit Assignment Statement  === visiVariableExpr  === Visit Return Statement  === visiVariableExpr  === visit Declaration: inta=2  === visit Declaration: intb=44  === visit Declaration: charc='K'  === visitDo While Statement  === visit If Statement: IsMayo(a<b){a=a+2;}NotMayo{a=a\*b;}  === visitRelational Op Expression a<b  === visiVariableExpr  === visiVariableExpr  === Visit Assignment Statement  === visiVariableExpr  === visiVariableExpr  === Visit Assignment Statement  === Visit Add/Subtract Expression: a+2  === visiVariableExpr  === visitPrint\_stmt  === visiVariableExpr  === visitRelational Op Expression a<600  === visiVariableExpr  === Visit Assignment Statement  === Visit Function Call  === visiVariableExpr  === visitPrint\_stmt  === visiVariableExpr |

|  |
| --- |
| **Input.bob Runtime Output** |
| a = 4  a = 6  a = 8  a = 10  a = 12  a = 14  a = 16  a = 18  a = 20  a = 22  a = 24  a = 26  a = 28  a = 30  a = 32  a = 34  a = 36  a = 38  a = 40  a = 42  a = 44  a = 1936  b = 5928  0.03 seconds total execution time. |

|  |
| --- |
| **Input2.bob Runtime Output** |
| bob = 12  Thank you for visiting Bikini Bottom  0.01 seconds total execution time. |

**Reflection**

The process of creating a language from scratch is a very ambitious task, requiring many scrapped ideas over the 8 weeks of developing the language. In order to complete this project, we learned of many new softwares such as ANTLR4 and Jasmine, to create the proper diagrams and trees, while generating the .j files needed to run our code. However, the difficulties of using new software is the amount of time required setting up the environment and dependencies in order to properly use them in our implementation.

Implementation wise, the debugging is one of the more challenging aspects of this project. Most bug occurrences happen in highly-coupled program structures, which can ultimately affect many other parts of the program if left unnoticed. The development was a constantly changing iterative process. If the current status of our language was compared to our originally planned syntax, there would be significant differences in their logic and structure.

**Conclusion**

Needless to say: compilers are under appreciated. This course started off slowly with learning the fundamentals of a compiler and slowly transitioning to building a compiler of our own. We chose to have some fun with the overall project and decided to make our language based off the Sponge Bob Square Pants show. With that idea in mind, each week we had lab assignments that helped us to understand all the components that would be necessary for our project in the latter half of the semester. When it came to starting to work on our project, we had design ideas such as having no brackets or semicolons and using a newline in place of them. But as we attempted to implement these ideas, we ran into several problems that ended up having us go back to using brackets and semicolons. Those design problems along with some local environment setup issues and learning new software such as Antlr-4 had us rethink our design. Our design then became a mixture of a simplified version of C++ and a Sponge Bob theme. Noticeable references to the show are shown in our if-else statement and do-while loop, where there were changed to isMayo-notMayo and bringit-aroundtown. Other notable references include: using “Krabbie” in place of “main” , “Pattie” in place of our last bracket to the main function and using “PrintingPress” as our form of “cout”. Overall, we are satisfied with the resulting design to the language and its functionality and have developed a larger appreciation for compilers.