GitHub: https://github.com/JuheonChu/lumber_84

Code Modernization and Documentation

Dickinson CS student research Group

Outline

- **Project 1:** Modernization of the corporate codebase via the conversion of C codebase into Java, Python, and .NET

  by Boosung, Maximo, William, Shahir, Fox.

- **Project 2:** Scan the corporate codebase and produce documentation and specifications that can facilitate training new developers.

  by John, Pranav, Ahnaf, Youssif

Project Scheme

- Aims to discover the optimal approach to initiate them.
- Divide the team into multiple subgroups to experiment with various approaches.

Subgroups

Subgroup 1: Maximo, Shahir, Fox

Design and implement a backend system to support code analysis.

Subgroup 2: John, Pranav, Ahnaf, Youssif

Construct an NLP pipeline to generate documentation based on codebases.

Subgroup 3: Boosung, William

Experimentation with existing open-source platforms such as Copilot.

Subgroup 1 (Maximo & Shahir & Fox)

- **Presentation:** We've started to create a presentation on the Back-End aspect that will explain its importance for 84 Lumber, how to implement it, tools available, and more for the next meeting.
- **API**: Had the computer trouble but I'll try and get the bare-bones API so that I can do a little demo on how it works with its basic operations for the next meeting.

API

- The API is based on .NET Core 7 using MySQL as the database engine. Idea is to have a basic CRUD (Create Read Update & Delete) API that can perform the basic operations on a demo database that stores basic user information. Both the database and the API are run locally but would have the same functionality on a server, the only difference with the cloud-base API is that it would connect back and forth with a server instead of being run locally.

Subgroup 2 (John, Pranav, Ahnaf, Youssif)

Approach

- Construct an NLP pipeline to generate documentation based on the input codebases.

Objective

- Address Project 1 & Project 2 details as denoted in Outline.

Motivation

- Delve into NLP, Programming Language Structure, and low-level programming languages (e.g. Assembly language, Low C/C++, BASIC)

Project settings

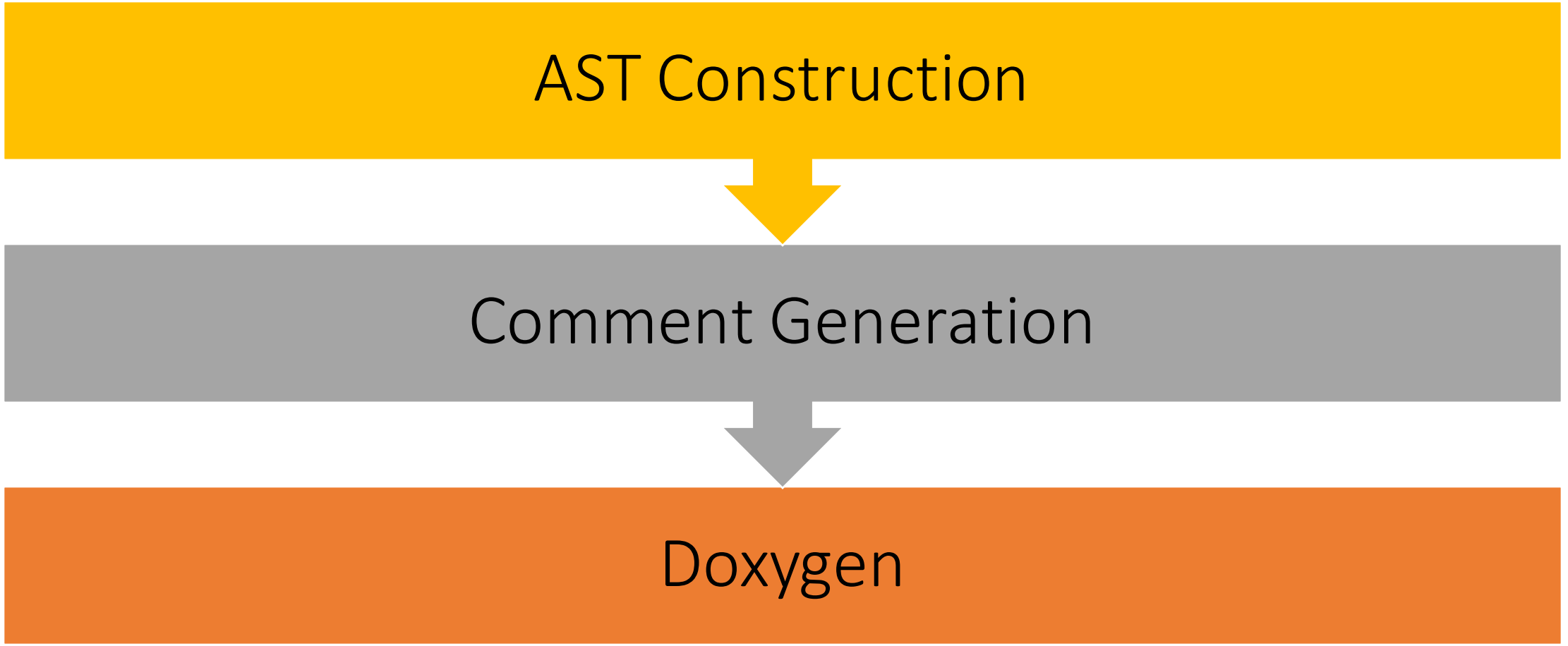
Software: WSL Ubuntu (John), Multipass (Ahnaf), Pranav & Youssif (Docker, Jupyter), NLTK, graphviz

Programming Languages: BASIC, C/C++, Python, Java, ANTLR4

Deep Learning models: Hugging Face Transformer language models (e.g. GPT, BERT)

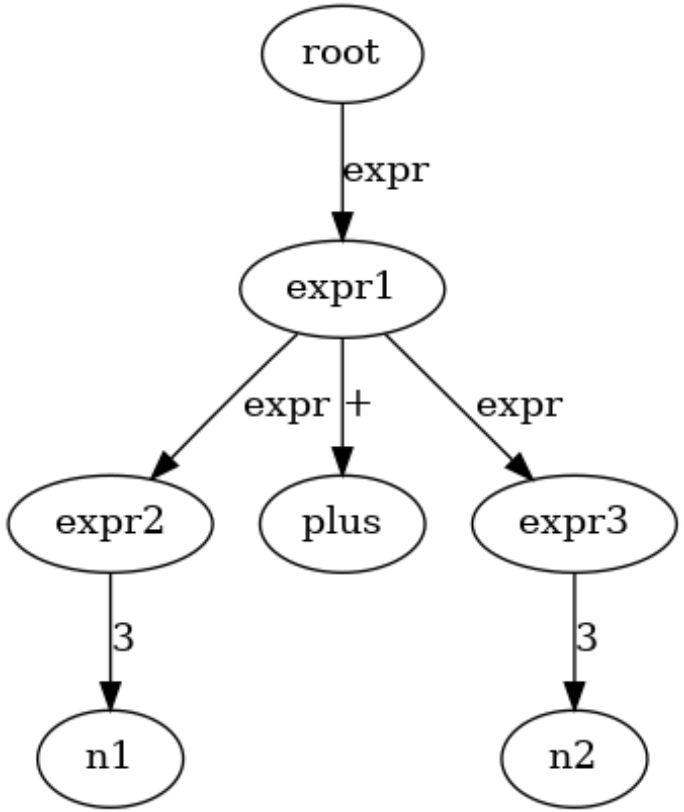
Subgroup Meeting: Weekly subgroup meeting to discuss the direction of our progress.

Project Workflow

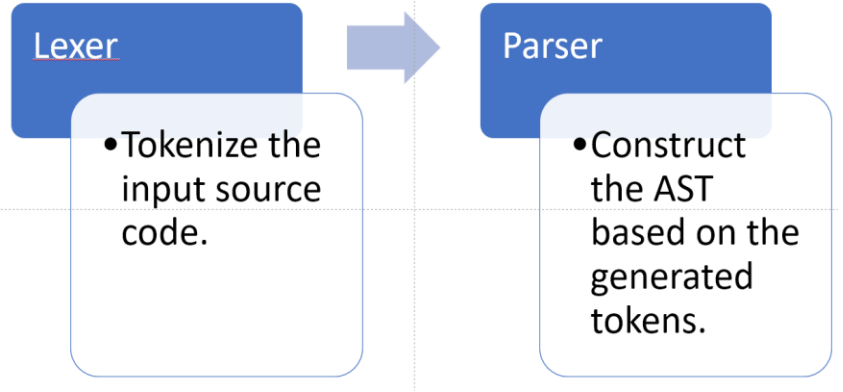


Abstract Syntax Tree (AST) Construction (John & Ahnaf)

- We aim to construct the Abstract Syntax Tree to store tokens that implicate pivotal program logics.



Abstract Syntax Tree



ANTLR4 Pipeline

Grammar Definition (BASIC.g4)

- Define a regular expression that can analyze the BASIC programming framework to construct the Abstract Syntax Tree.
- Experimented with a simple BASIC code.

```
1 REM A simple BASIC program to add two numbers
2 PRINT "Enter your name: ";
3 INPUT NAME$
4 PRINT "Hello, " + NAME$ + "! Let's add two numbers."
5 PRINT "Enter the first number: ";
6 INPUT NUM1
7 PRINT "Enter the second number: ";
8 INPUT NUM2
9 SUM = NUM1 + NUM2
10 PRINT "The sum of " + STR$(NUM1) + " and " + STR$(NUM2) + " is: " + STR$(SUM)
11 END
```

Example Code

Grammar

```
statement
    : remStatement
    | printStatement
    | inputStatement
    | assignmentStatement
    | endStatement
    ;

remStatement
    : REM .*? NEWLINE
    ;

printStatement
    : PRINT expression (';')?
    ;

inputStatement
    : INPUT variable
    ;

assignmentStatement
    : variable '=' expression
    ;

endStatement
    : END
    ;
```

Statement

```
variable : IDENTIFIER ('$')? ; // Optional $ for string variables

number : NUMBER ;

REM : 'REM' ;
PRINT : 'PRINT' ;
INPUT : 'INPUT' ;
END : 'END' ;
IDENTIFIER : [a-zA-Z] [a-zA-Z0-9]* ;
NUMBER : [0-9]+ ;
STRING : '"' .*? '"' ; // A simple string matcher, adjust as needed
WS : [ \t]+ -> skip ; // Whitespace
NEWLINE: '\r'? '\n' ;
```

Variable

```
expression
    : expression '+' expression        # Add
    | STRING                           # String
    | variable                         # VariableExpression
    | 'STR$' '(' expression ')'        # ConvertToString
    ;
```

Expression

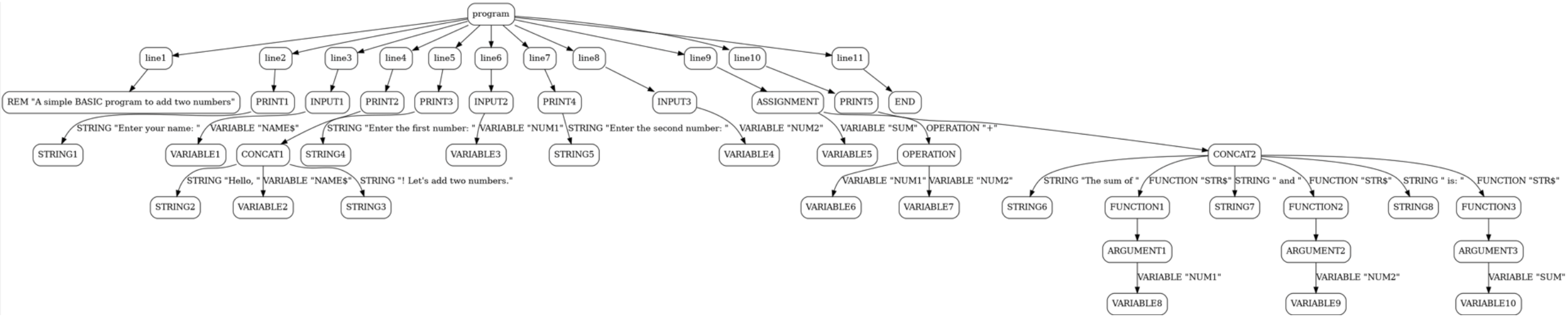
Tokenized AST

- As a result of ANTLR4 pipeline, we are able to get a tokenized Abstract Syntax Tree.

```
parsed_output = '''(program (line (number 1) (statement
(remStatement REM A simple BASIC program to add two numbers \n 2 PRINT
"Enter your name: " ; \n 3 INPUT NAME $ \n 4 PRINT "Hello, " + NAME $ +
"! Let's add two numbers." \n 5 PRINT "Enter the first number: " ; \n 6
INPUT NUM1 \n 7 PRINT "Enter the second number: " ; \n 8 INPUT NUM2 \n 9
SUM = NUM1 + NUM2 \n 10 PRINT "The sum of " + STR$ ( NUM1 ) + " and " +
STR$ ( NUM2 ) + " is: " + STR$ ( SUM ) \n 11 END \n)) \n) <EOF>)'''
```

Tokenized AST

AST Contsruction



Challenges

- Standardize the Python code to construct the AST based on how Pranav & Youssif formatted (Slide 17).
- The bullet point above will be our task next week.

Progress

- We read materials sent by Mark and have been studying another book that is helpful in understanding the programming structure.
- Based on the **"CBASIC Reference Manual,"** we are defining a grammar for CBASIC to tokenize and construct AST.
- We have been bridging the process between Step 1 & Step 2.

CBASIC Programming Language Structure

- **Programming rules, conventions, and labels**
- **Identifications, Numbers, Expressions**
- Statements
- Functions
- File I/O

```
grammar Expr;

// Parser Rules
program : (statement)* ;

statement : assignment | conditionalStatement | functionCall ;

assignment : variable '=' expression ;
conditionalStatement : 'IF' expression relationalOperator expression 'THEN' functionCall ;
functionCall : 'CALL' ID ;

expression
   : '-' expression                                 #UnaryMinusExpr
   | '(' expression ')'                             #ParensExpr
   | functionReference                              #FuncRefExpr
   | variable                                       #VarExpr
   | constant                                       #ConstExpr
   | expression arithmeticOperators expression      #ArithExpr
   | expression relationalOperator expression       #RelExpr
   | expression logicalOperator expression          #LogicExpr
   ;



functionReference : ID '(' expression ')'   // E.g.: ABS(150)
                  | ID                      // E.g.: TEMP.A
                  ;

variable : ID ( '.' ID )? ('$')? ;
constant : STRING | NUMBER ;

operator : arithmeticOperators
         | relationalOperator
         | logicalOperator ;

arithmeticOperators : '^' | '*' | '/' | '+' | '-' ;
relationalOperator : '<' | '<=' | '>' | '>=' | '=' | '<>' ;
logicalOperator : 'NOT' | 'AND' | 'OR' | 'XOR' ;

// Lexer Rules
ID : [a-zA-Z_][a-zA-Z0-9_]* ;
STRING : '"' ( ~["\r\n] )* '"' ;
NUMBER : [0-9]+ ('.' [0-9]+)? ;
WS : [ \t\r\n]+ -> skip ;
```

Expr.g4

Experimentation

- Though we need more testings with different codebase, the following example worked.

```
X = ABS(-150)
Y = 100 + 250
Z = X * Y / 3.5

IF X < Y THEN CALL DISPLAY.MESSAGE
IF ABS(X-Y) = 0 THEN CALL WARN.MSG

TEMP.A = 20.5
TEMP.B = TEMP.A + 15.3

IF TEMP.B > 35 THEN CALL COOLING.SYSTEM

VAL = ABS(TEMP.A - TEMP.B)

NAME$ = "John"
```

```
(program (statement (assignment (variable X) = (expression
(functionReference ABS ( (expression - (expression (constant
150))) ))))) (statement (assignment (variable Y) = (expression
(expression (constant 100)) (arithmeticOperators +) (expression
(constant 250))))) (statement (assignment (variable Z) =
(expression (expression (expression (functionReference X))
(arithmeticOperators *) (expression (functionReference Y)))
(arithmeticOperators /) (expression (constant 3.5)))))
(statement (conditionalStatement IF (expression
(functionReference X)) (relationalOperator <) (expression
(functionReference Y)) THEN (functionCall CALL DISPLAY))))
```

Tokenized AST

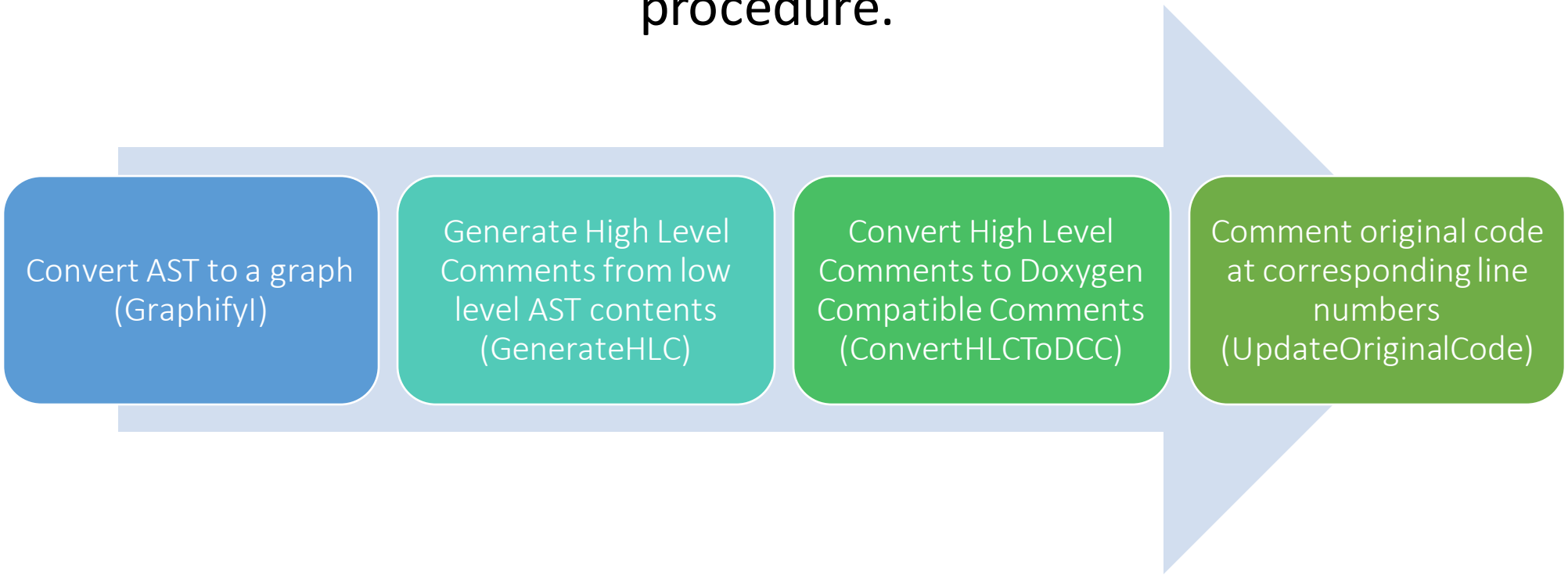
Bridging Process

```
+ class Node:
+     def __init__(self, value):
+         self.value = value
+         self.children = []
+
+ def parse(tokens):
+     stack = []
+     for token in tokens:
+         if token == '(':
+             stack.append(Node(None))
+         elif token == ')':
+             child = stack.pop()
+             if stack:
+                 stack[-1].children.append(child)
+             else:
+                 return child
+         else:
+             stack[-1].children.append(Node(token))
+     return stack[0]
+
+ def tokenize(input_string):
+     return input_string.replace('(', ' ( ').replace(')', ' ) ').split()
+
+ def traverse(node, line=1):
+     output = []
+     output.append((node.value, line))
+     for child in node.children:
+         line += 1
+         child_output, line = traverse(child, line)
+         output.extend(child_output)
+     return output, line
+
+
+
```

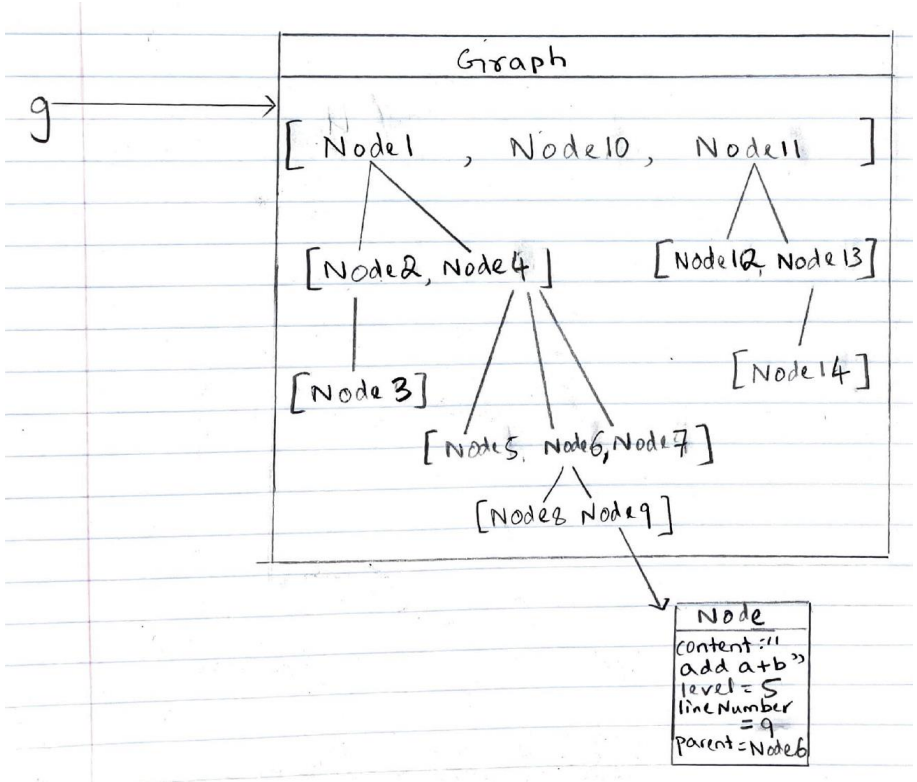
```
jc981073@DESKTOP-R3H5HUN:~/lumber_84/jchu98$ python3 Construct_AST.py
program,2
None,3
statement,4
None,5
assignment,6
None,7
variable,8
X,9
=,10
None,11
expression,12
None,13
functionReference,14
ABS,15
None,16
None,17
expression,18
-,19
None,20
expression,21
None,22
constant,23
150,24
None,25
```

Comment Generation (Pranav & Youssif)

Assuming the existence of an AST in a desired format (with proper indentation and line numbers), we seek to adopt the following procedure.



Step 1: Converting AST to a Graph (GraphifyI)



- Method named GraphifyI operates on an AST file in txt format, converting every line to a custom Node in the format

"__content,lineNumber" to node object Node(content, level, lineNumber, parent) (here, ___ denotes number of indentations done, which determines the level of the node).

- Subsequently, GraphifyI adds these nodes to a Graph object 'g', which is shown in the object diagram on the left

Code can be found at https://github.com/JuheonChu/lumber_84/tree/main/pranav

```java
package lumberTry;

public class SchoolClass {

	int studentNumber;
	int classRoom;
	String teacherName;

	public SchoolClass(int studentNumber, int classRoom, String teacherName) {
		this.classRoom = classRoom;
		this.studentNumber = studentNumber;
		this.teacherName = teacherName;
	}

	public void addStudent() {
		studentNumber ++;
	}

	public void classRoomNum(int classRoomNumber) {
		classRoom = classRoomNumber;
	}

	public static void main(String args[]) {
		SchoolClass CS = new SchoolClass(20,118,"Goble William");
		CS.addStudent();

	}

}
```

```
PackageName,1
ClassDeclaration,3
	StudentNumberDeclaration,5
	ClassRoomDeclaration,6
	TeacherNameDeclaration,7
	Constructor,9
		classRoomInitial,10
		studentNumberInitial,11
		teacherNameInitial,12
	methodAddStudent,15
		AddStatement,16
	methodClassRoomNum,19
		AddStatement,20
	methodMain,23
		NewSchoolClass,24
		addStudentCall,25
```

```
Level 1: PackageName (Line 1)
Level 1: ClassDeclaration (Line 3)
  Level 2: StudentNumberDeclaration (Line 5)
  Level 2: ClassRoomDeclaration (Line 6)
  Level 2: TeacherNameDeclaration (Line 7)
  Level 2: Constructor (Line 9)
    Level 3: classRoomInitial (Line 10)
    Level 3: studentNumberInitial (Line 11)
    Level 3: teacherNameInitial (Line 12)
  Level 2: methodAddStudent (Line 15)
    Level 3: AddStatement (Line 16)
  Level 2: methodClassRoomNum (Line 19)
    Level 3: AddStatement (Line 20)
  Level 2: methodMain (Line 23)
    Level 3: NewSchoolClass (Line 24)
    Level 3: addStudentCall (Line 25)
```

Currently, we have obtained successful results by testing GraphifyI on a dummy AST

SWOT Analysis of GraphifyI

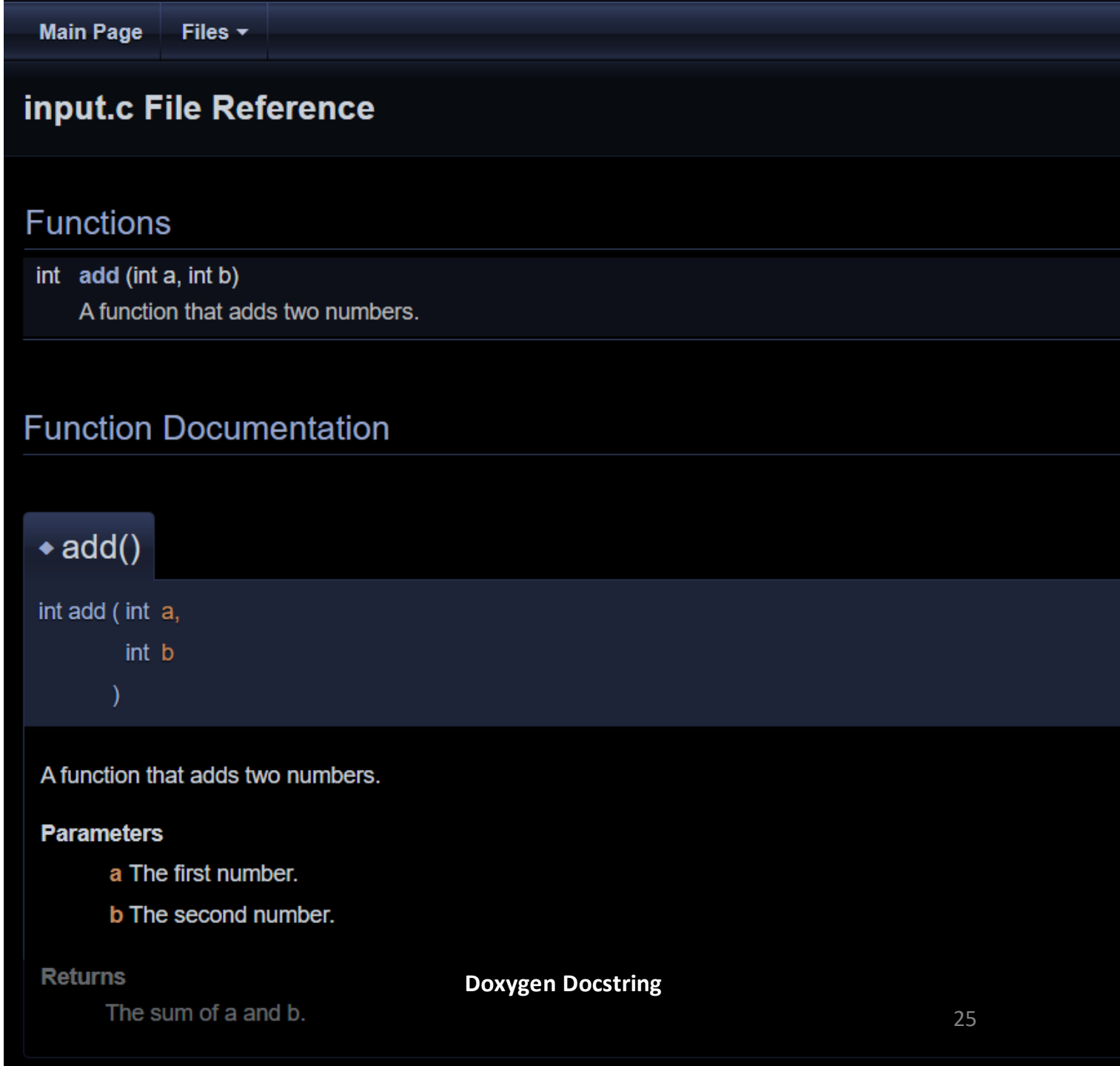
- Strengths: Captures both the hierarchy and sequence of program instructions via a custom data structure tailor-made for ASTs!
- Weaknesses: Requires AST to be formatted in a particular format
- Opportunities: Code for GraphifyI can be generalized for a wide range of files where both sequence and hierarchy are important
- Threats: Coming up with an efficient traversal method may be difficult

Step 2: Generating Comments

- We read through the CBASIC Manual and realized that explanation of CBASIC statements was missing in the manual. It discussed the concepts of the language more than the code itself.
- Thus, we are planning on using a manual for BASIC that has an explanation of the syntax, on the basis of the fact that CBASIC and BASIC are very similar languages
- Task for next week and beyond : to obtain a dictionary comprising the code statements as keys, and their explanations in the BASIC manual as the values.

Document Generation

- When DocString is generated at Step 2, we will be able to generate a documentation that outlines the code.



Doxygen Docstring

Progress

John & Ahnaf are researching regular expressions to define a grammar that tokenizes corporate BASIC codebases & constructs AST.

Pranav & Youssif are nailing down to construct the GraphifyI function and further investigating to apply Spacy library to generate Doxygen-formatted Docstrings.

Subgroup 3 (William and Boosung)

Subgroup 3 (William and Boosung)

Approach

- Propose a translation pipeline using Copilot or GPT4 specified to the use-case per codebase.

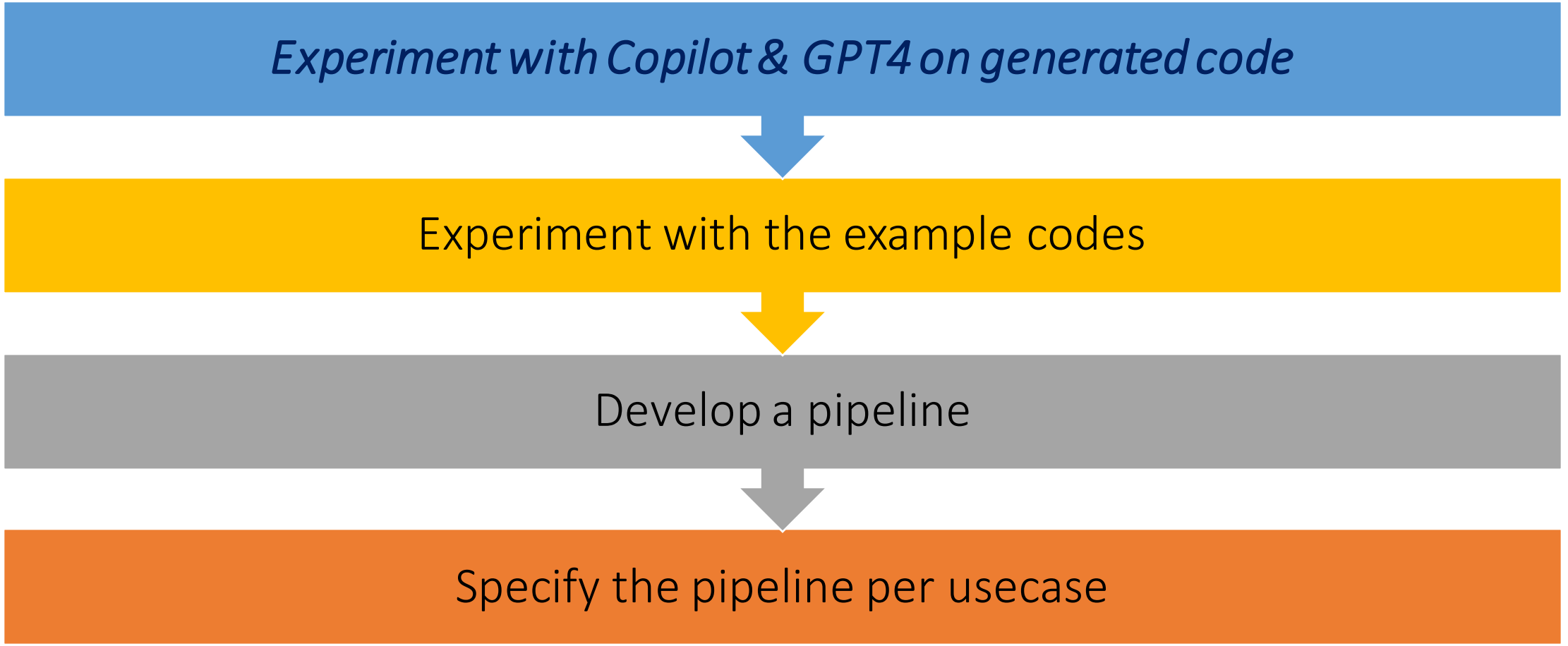
Objective

- Address Project 1 as denoted in Outline.

Motivation

- Microsoft Copilot is the standard used in the industry, used by over 1M developers and 20,000 organizations

Project Timeline



Weekly Updates

1. Execution of CBasic Code within a Virtualized Environment
   - Currently running into the compile errors
     - Specifics listed in the next slide
   - Downloaded supplementary helpers (emacs, gcc)
2. Experiment with GitHub Copilot
   - Successful run with a few sample codes (~20 lines long)

1. Execution of CBasic Code on VM

1. Attempted to compile the bac.c
2. Types of errors returned
   - incompatible implicit declaration of function
   - Previous declaration with type void()
   - Unmatching function name declaration
   - Static declarations follow non-static declaration
   - passing argument 1 of ‘_setjmp’ from incompatible pointer type
   - return type defaults to ‘int’ for the main
   - previous declaration of ‘drem’ with type ‘double(double, double)’

```
bac.c:20:27: error: static declaration of ‘round’ follows non-static declaration
   20 | static double pascal near round();
      |                           ^~~~~
In file included from /usr/include/features.h:486,
                 from /usr/include/x86_64-linux-gnu/bits/libc-header-start.h:33,
                 from /usr/include/stdio.h:27,
                 from bac.c:1:
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:301:1: note: previous declaration of ‘round’ with type ‘double(double)’
  301 | __MATHCALLX (round,, (_Mdouble_ __x), (__const__));
      | ^~~~~~~~~~~
bac.c:438:15: error: ‘drem’ redeclared as different kind of symbol
  438 | static double drem = 0.0 ;
      |               ^~~~
In file included from /usr/include/features.h:486,
                 from /usr/include/x86_64-linux-gnu/bits/libc-header-start.h:33,
                 from /usr/include/stdio.h:27,
                 from bac.c:1:
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:187:1: note: previous declaration of ‘drem’ with type ‘double(double,  double)’
  187 | __MATHCALL (drem,, (_Mdouble_ __x, _Mdouble_ __y));
      | ^~~~~~~~~~
bac.c:833:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
  833 | main(argc,argv)
      | ^~~~
bac.c: In function ‘main’:
bac.c:882:1: warning: implicit declaration of function ‘_mbsdim’; did you mean ‘_mbqsdim’? [-Wimplicit-function-declaration]
  882 | _mbsdim(&lckt_S,__lckt_S,20);
      | ^~~~~~~
      | _mbqsdim
bac.c:892:1: warning: implicit declaration of function ‘_mbprep’; did you mean ‘_mbpeek’? [-Wimplicit-function-declaration]
  892 | _mbprep(12,20,"ssss");
      | ^~~~~~~
      | _mbpeek
bac.c:893:1: warning: implicit declaration of function ‘_mbparm’; did you mean ‘_mbnorm’? [-Wimplicit-function-declaration]
  893 | _mbparm(&lapemp_S);
      | ^~~~~~~
      | _mbnorm
bac.c:900:30: warning: implicit declaration of function ‘_mbiend’; did you mean ‘_mbint’? [-Wimplicit-function-declaration]
  900 |   if (setjmp(_mbxfile) == 0) _mbiend();
      |                              ^~~~~~~
      |                              _mbint
bac.c:978:1: warning: implicit declaration of function ‘_mbfrread’; did you mean ‘_mbflread’? [-Wimplicit-function-declaration]
  978 | _mbfrread(1, (long) verloc_I,"s",&kzgarb_S);
      | ^~~~~~~~~
      | _mbflread
bac.c:986:1: warning: implicit declaration of function ‘_mbrfprnt’; did you mean ‘_mbscprnt’? [-Wimplicit-function-declaration]
  986 | _mbrfprnt(1, (long) verloc_I,"s",pver_S);
      | ^~~~~~~~~
      | _mbscprnt
```

2. GitHub Copilot Experimentation

Sample C code

```c
#include <stdio.h>

// Function to calculate GCD (Greatest Common Divisor)
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

// Function to calculate LCM (Least Common Multiple)
int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

int main() {
    int num1, num2;
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    printf("LCM of %d and %d is %d", num1, num2, lcm(num1, num2));
    return 0;
}
```

Translated Python Code By Copilot

```python
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

def lcm(a, b):
    return (a * b) // gcd(a, b)

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

print("LCM of", num1, "and", num2, "is", lcm(num1, num2))
```

For Next Week:

- Debug current errors with the CBasic file
  - If issues persist, look into potential virtual box instance solutions
  - Was the code sent to us as is?
  - What compiler is used to run the CBasic code?
- Experiment more with Copilot and GPT4
- Experiment with Copilot Labs and other pipeline tools