****

**De La Salle University • College of Computer Studies**

**Maestro**

Comparative Analysis of Multiple Languages

Name (last name first) : Fernandez, Ryan Austin

Poblete, Clarisse Felicia M.

San Pedro, Marc Dominic

Tan, Johansson E.

Section : G01

Date of Submission : January 28, 2016

**Similarities and Differences**

All five programming languages have their similarities in logic and share some common terms and elements, though there are still differences. Some make use of different terms and implementations, some have make their own additions, and some choose not to include other types and features, based on their own strengths.

For example, C++ puts more control into the hands of the programmer, which is why, unlike the other languages, it chose to not treat string as its own separate data type, and is able to look at it more like a sequence of characters. Python on the other hand, avoids low-level details and keeps its syntax minimalistic, and does so partially by removing “unnecessary” elements such as the char data type and the switch statement. While C# isn’t as “low-level” as C++, it does still choose to have those elements that Python decided not to include. Javascript is very different from the others partially because the variables are untyped and dynamic. Lastly, Scala as a functional programming language has syntax that follows a logic different from that of the other four languages which are purely procedural, but it does allow the developer to blend procedural with functional logic..

**Special Features**

C++

Support for OOP + Support for Functional Programming (Usage of Pointers)

Standard Template Library (STL)

C#  
OOP  
Used for writing Windows 8 and Windows Phone Apps  
Supports both static and dynamic binding

Can be used to create web apps via the .NET framework

Python

Makes use of new lines and spacing instead of brackets and parentheses in organizing code

Has minimalistic syntax

Is byte-code interpreted

Javascript

Can interact with HTML and CSS (and Server side through AJAX)

Dynamic Variable Typing

Different ways to declare objects (functions or JSON)

Function variables

Scala

Both explicit and implicit variable binding.

Support for Functional Programming

Support for Closures (functions referring to outside variables)

Pattern matching (Similar to switch but can handle multiple types)

**Maestro**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **C++** | **C#** | **Python** | **Javascript** | **Scala** | **Maestro** |
| **Data** | Data Types | char bool  int  wchar\_t  float  double | char  bool  int  String  float  double | int  float  long  complex  boolean  string  dictionary | Number  Boolean  String  Array  Function  Regex  Object | Byte/Short/Int  /Long  Float/Double  Char  String  Boolean  Unit  Any/Nothing/AnyRef | Note  Rest  Sync  Seq |
|  | Values | ‘C’  false  9430  L"A”  31415.9265  0.123456789 | ‘c’  true  24601  “Hello World”  3.14159265  5 | 1  1.0  1L  1j | complex(0, 1)  True | False  “Hello” | ‘Hello’  dictionary = {‘Key1’: ‘Value1’, ‘Key2’: ‘Value2} | 12 | 014 | 0x0C | 12.0 | 1.2e1  true | false  “Hello”  [1,2,3,4]  <see below>  /^[abc]{5}$/  {  name : “My Name”,  age : 12  } | 12 | 014 | 0x0C | 12L  12.0 | 1.2e1  ‘s’ | ‘\n’  “Hello” | “””Multi  Line”””  true | false  - | note(C#,3,1/4)  rest(1/4)  sync {  note(C#,3,1/4)  note(F,3,1/4)  note(G#,3,1/4)  }  seq {  note(C#,3,1/4)  note(F,3,1/4)  note(G#,3,1/4)  } |
|  | Constants and Variables | Explicit  Static | Implicit  Static | Implicit  Static | Implicit  Dynamic | Explicit/Implicit  Static | Implicit  Dynamic |
| **Process** | Branching | if...else  switch | if...else  switch | if...else | if...else  switch | if...else  pattern matching |  |
|  | Iteration | for  while  do...while | for  while  do...while  foreach | for  while | while  do...while  for  for...in | for  while  do...while | object \* number of iterations |
| **Subprograms** |  | <return type> <function name> (type var1, type var2, …) {  <body>  } | <return type> <function name> (type var1, type var2, …) {  <body>  } | def <function name>(param1, param2...):  <body> | function <function name> (var1, [var2,...]) {  <body>  } | def <function name>(param1: param1 type, [param2 : param2 type,...]) : <return type> = {  <body>  } | sync {  <body>  }  seq {  <body>  } |

**Table 1 - Comparison of Five Languages and Maestro**

**Sample Program**

play{

note(C#, 3, 1/4) -> x

rest(1/4)

x>> \* 2

x>4

chorus

seq{

note(C, 3, 1/4)

note(E, 3, 1/4)

note(G, 3, 1/4)

} -> seq1

sync{

note(A, 3, 3/4)

seq{

note(C, 3, 1/4)+4

note(E, 3, 1/4)+4

note(G, 3, 1/4)+4

}

seq1+8

}

seq{

seq1[1..2]

note(A#, 3, dot(3/4))+4

note(Gb, 3, dot(dot(dot(1/32)))) -> y

y++

y+2

} -> seq2

chorus++

seq2

y-4

}

seq{

note(C, 5, 1/4) \* 3

note(E, 5, 1/2)

note(G, 5, 1/4)

} -> chorus

**Tokenizer**

**Driver.java**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.PrintWriter;

import java.util.Iterator;

public class Driver {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(

new FileReader(

new File(args[0])));

String code = "";

String s;

do {

s = br.readLine();

if( s != null ) {

code += s + "\n";

}

} while(s != null);

br.close();

if( code.endsWith("\n") ) {

code = code.substring(0,code.length() - 1);

}

Tokenizer t = new Tokenizer(code);

t.tokenize();

Iterator itr = t.getTokens();

PrintWriter pw = new PrintWriter(

new BufferedWriter(

new FileWriter(

new File(args[1]))));

while(itr.hasNext()) {

pw.println(itr.next());

}

pw.close();

}

}

**Token.java**

public class Token {

private String token;

private String type;

public Token(String token, String type) {

this.token = token;

this.type = type;

}

public String token() {

return token;

}

public String type() {

return type;

}

public String toString() {

return (token.equals("\n") ? "\\n" : token) + " - " + type;

}

}

**Tokenizer.java**

import java.util.ArrayList;

import java.util.Iterator;

public class Tokenizer {

private ArrayList<Token> tokens;

private String code;

private int state;

private String currToken;

private static String[] reserved;

private static String[] operators;

public Tokenizer(String code) {

this.code = code;

tokens = new ArrayList<Token>();

state = 0;

currToken = "";

reserved = new String[] {

"play", "note", "rest", "seq", "sync", "dot"

};

operators = new String[] {

"->", "+", "++", "-", "--", ">", ">>", "<", "<<", "/", "..", "\*",

"(", ")", "{", "}", ",","[","]"

};

}

public void tokenize() {

for(int i = 0; i < code.length(); i++) {

char c = code.charAt(i);

// System.out.println(state + " " + currToken + " "

// + (c == '\n' ? "\\n"

// : (c == ' ' ? "space" : c)));

switch(state) {

//new token

case 0:

if(c == '/') {

state = 1;

currToken += c;

} else if(c == '\t' || c == ' ') {

continue;

} else if(c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z') {

state = 4;

currToken += c;

} else if( c == '\n' ) {

currToken += '\n';

addToken();

} else if (c >= '0' && c <= '9' ) {

state = 5;

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

currToken += c;

break;

}

}

}

break;

//slash read

case 1:

if( c == '/') {

state = 2;

currToken = "";

} else {

i--;

state = 3;

}

break;

//in comment

case 2:

if( c == '\n') {

state = 0;

currToken = "\n";

addToken();

}

break;

//operator check

case 3:

if(c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z' ) {

state = 4;

addToken();

currToken += c;

} else if(c >= '0' && c <= '9' ) {

state = 5;

addToken();

currToken += c;

} else if( c == ' ' || c == '\t' ) {

state = 0;

addToken();

} else if( c == '\n') {

state = 0;

addToken();

currToken += c;

addToken();

} else {

boolean foundStart = false;

//check if start of operator

for(String s: operators) {

if( s.startsWith(currToken + c)) {

currToken += c;

foundStart = true;

break;

}

}

if( !foundStart) {

addToken();

currToken += c;

}

}

break;

//word check

case 4:

if( c == ' ' || c == '\t' ) {

state = 0;

addToken();

} else if( c == '\n') {

state = 0;

addToken();

currToken += c;

addToken();

} else if( c == '/') {

state = 1;

addToken();

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

addToken();

currToken += c;

break;

}

}

if( state != 3 ) {

currToken += c;

}

}

break;

//num check

case 5:

//if non-numerical

if( c < '0' || c > '9' ) {

addToken();

if( c == ' ' || c == '\t') {

state = 0;

} else if( c == '\n') {

state = 0;

currToken += c;

addToken();

} else if(c >= 'A' && c <= 'Z' || c >= 'a'

&& c <= 'z') {

state = 4;

currToken += c;

} else if( c == '/') {

state = 1;

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

currToken += c;

break;

}

}

}

} else {

currToken += c;

}

break;

default:

}

}

addToken();

}

public void addToken() {

Token t = new Token(currToken,tokenType());

tokens.add(t);

// System.out.println("Added " + t);

currToken = "";

}

public String tokenType() {

if( currToken.equals("\n") ) {

return "newline";

}

for(String s: reserved) {

if( s.equals(currToken)) {

return s;

}

}

for(String s: operators) {

if(s.equals(currToken)) {

return s;

}

}

try {

int i = Integer.parseInt(currToken);

if( i > 0 ) {

return "int";

} else {

return "Invalid number, cannot be negative";

}

} catch(NumberFormatException nfe) {

if(currToken.matches("^[A-G](#|##|b|bb)?$")) {

return "pitch";

} else if( currToken.matches("^[A-Za-z][A-Za-z0-9\_]\*$") ) {

return "varname";

} else {

return "other";

}

}

}

public Iterator getTokens() {

return tokens.iterator();

}

}

import java.util.ArrayList;

import java.util.Iterator;

public class Tokenizer {

private ArrayList<Token> tokens;

private String code;

private int state;

private String currToken;

private static String[] reserved;

private static String[] operators;

public Tokenizer(String code) {

this.code = code;

tokens = new ArrayList<Token>();

state = 0;

currToken = "";

reserved = new String[] {

"play", "note", "rest", "seq", "sync", "dot"

};

operators = new String[] {

"->", "+", "++", "-", "--", ">", ">>", "<", "<<", "/", "..", "\*",

"(", ")", "{", "}", ",","[","]"

};

}

public void tokenize() {

for(int i = 0; i < code.length(); i++) {

char c = code.charAt(i);

switch(state) {

//new token

case 0:

if(c == '/') {

state = 1;

currToken += c;

} else if(c == '\t' || c == ' ') {

continue;

} else if(c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z') {

state = 4;

currToken += c;

} else if( c == '\n' ) {

currToken += '\n';

addToken();

} else if (c >= '0' && c <= '9' ) {

state = 5;

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

currToken += c;

break;

}

}

}

break;

//slash read

case 1:

if( c == '/') {

state = 2;

currToken = "";

} else {

i--;

state = 3;

}

break;

//in comment

case 2:

if( c == '\n') {

state = 0;

currToken = "\n";

addToken();

}

break;

//operator check

case 3:

if(c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z' ) {

state = 4;

addToken();

currToken += c;

} else if(c >= '0' && c <= '9' ) {

state = 5;

addToken();

currToken += c;

} else if( c == ' ' || c == '\t' ) {

state = 0;

addToken();

} else if( c == '\n') {

state = 0;

addToken();

currToken += c;

addToken();

} else {

boolean foundStart = false;

//check if start of operator

for(String s: operators) {

if( s.startsWith(currToken + c)) {

currToken += c;

foundStart = true;

break;

}

}

if( !foundStart) {

addToken();

currToken += c;

}

}

break;

//word check

case 4:

if( c == ' ' || c == '\t' ) {

state = 0;

addToken();

} else if( c == '\n') {

state = 0;

addToken();

currToken += c;

addToken();

} else if( c == '/') {

state = 1;

addToken();

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

addToken();

currToken += c;

break;

}

}

if( state != 3 ) {

currToken += c;

}

}

break;

//num check

case 5:

//if non-numerical

if( c < '0' || c > '9' ) {

addToken();

if( c == ' ' || c == '\t') {

state = 0;

} else if( c == '\n') {

state = 0;

currToken += c;

addToken();

} else if(c >= 'A' && c <= 'Z' || c >= 'a'

&& c <= 'z') {

state = 4;

currToken += c;

} else if( c == '/') {

state = 1;

currToken += c;

} else {

//check if start of operator

for(String s: operators) {

if( c == s.charAt(0)) {

state = 3;

currToken += c;

break;

}

}

}

} else {

currToken += c;

}

break;

default:

}

}

addToken();

}

public void addToken() {

Token t = new Token(currToken,tokenType());

tokens.add(t);

currToken = "";

}

public String tokenType() {

if( currToken.equals("\n") ) {

return "newline";

}

for(String s: reserved) {

if( s.equals(currToken)) {

return s;

}

}

for(String s: operators) {

if(s.equals(currToken)) {

return s;

}

}

try {

int i = Integer.parseInt(currToken);

if( i > 0 ) {

return "int";

} else {

return "Invalid number, cannot be negative";

}

} catch(NumberFormatException nfe) {

if(currToken.matches("^[A-G](#|##|b|bb)?$")) {

return "pitch";

} else if( currToken.matches("^[A-Za-z][A-Za-z0-9\_]\*$") ) {

return "varname";

} else {

return "other";

}

}

}

public Iterator getTokens() {

return tokens.iterator();

}

}