Mio Programming Language

- By
  - **B19102104 Muhammad Umar Anzar**
  - **B19102048 Izhan Nadeem**
  - **B19102004 Abdul Muqsit**
- University Of Karachi
- UBIT department of computer science
- Subject Compiler Construction

Description

Project: Front End Compiler Project

Language New Keywords

Java/C#	Mio	Java/C#	Mio
if	if	int	int
else	else	double	point
while	loop till	char	char
do	do till	byte	excluded
for	excluded	float	excluded
forEach	loop thru	string	str
switch	shift	boolean	bool
case	state	long	excluded
default	default	dynamic /var	val
continue	cont	comment	@comment
break	stop	comment	@@multi-line comment@@
return	ret	line terminator;	;
void	implicit	var	var
main	begin		

Java/C#	Mio	Java/C#	Mio
class	Class	try	test
interface	excluded	catch	except

Java/C#	Mio	Java/C#	Mio
abstract	Abstract	finally	finally
extends	Class name(inherited class)	throw	raise
final	const	throws	raises
implements	no interface		in
this	Self		
super	Parent		
public	implicit		
private	$$identiferName		
protected	$identiferName		
static	static		
new	new		

Classification of Lexemes

Operator

Same:

- Precedence
- Associativity
- Syntactically replaceable

Relation Operator	inc-dec	not	pm	mdm	power
<	++	!	+	*	^
>	--		-	/	
<=				%	
>=					
!=					
==					

simple assignment	compound assignment	and	or
=	+=	&&	||
	*=		
	-=		
	%=		

Keyword

Same:

- Syntactically replaceable

DT	String	var
int	str	var
point		
char		
val		
bool		

if	else	loop	till	thru	do	class
if	else	loop	till	thru	do	Class

switch	case	break	continue	return	static
switch	case	stop	cont	ret	static

final	main	in	default	new	abstract	super	this
const	begin	in	default	new	abstract	Parent	Self

Three ps	try	catch	finally	throw	throws
public	test	except	finally	raise	raises
private					
protected					

Syntax

Comment

```
Syntax:

@ Single Line Comment

@@ Multi
Line
comment @@
```

Main Function

```
Syntax:

Begin {
    block of code
}
```

Variable Definition and Initialization

```
Syntax:

declaration:
dataType variableName;

assignment:
variableName = Constant;

declaration and assignment:
dataType variableName = Constant;
```

Type Casting

```
Syntax:
DT <- value

int x = 2 * int<-3.9        @ Here 3.9 is converted into 3 then * 2

int x = int<- 2.9 * 32      @ Here 2.9 * 32 then converted into integer
```

Conditional Statement

- IF AND ELSE

```
Syntax:

if (condition) {
    block of code
```

```
}


if (condition) {
    block of code
} else {
    block of code
}

if (condition) {
    block of code
} else if (condition) {
    block of code
}
```

- SWITCH CASE

```
Syntax:

shift(expression) {
    state x:
        code block
        stop;
    state y:
        code block
        stop;
    default:
        code block
}
```

Loop

- While Loop

```
Syntax:

loop till (condition) {
     code block
}

int i = 0;
loop till (i < 5) {
  block of code
  i++;
}
```

- For Loop (iterator)

```
Syntax:

loop  thru ( DT i in (initial, final, incremental) ) {
    block of code
}
loop  thru ( int i in (0,5,1) ) {
```

```
    block of code
}
```

Function

- Procedure

```
Syntax:

def functionName(parameters,comma,separated) {
    block of code
}

def foo(int a, point b) {

}
```

- Function
  - returnType is dataType

```
def returnType Identifer(parameters,comma,separated) {
    block of code
    ret value;
}

def str foo(str x) {
    block of code
    ret x;
}
```

Class

- Class Structure

```
Syntax:

Class className(){

}
Class className(inheritedClass){

}
Class className(inheritedClass1,nheritedClass2){

}
Class className <Parameter> (inheritedClass1,nheritedClass2){

}
```

- Abstract class and function

```
abstract Class Shape{
    abstract draw();
```

```
}
```

- Example code

```
Syntax:

Class Car(){

    dataType attribute;         @public
    static dataType attribute;  @static
    dataType $attribute;        @private
    dataType $$attribute;       @protected

    @This function is public and void
    def foo(dataType variable1,dataType variable2) {
        @block of code
    }

    @This function is private and returnType is there
    def dataType $foo1(dataType variable1,dataType variable2) {
        @block of code
        ret variable1+variable2;
    }

    @This function is protected and returnType is there
    def dataType $$foo2(dataType variable1,dataType variable2) {
        @block of code
        ret variable1+variable2;
    }
}
```

- Parametric class

```
Syntax:

begin
{
      Polygone<A> x = new Box<A>();
      Polygone<str> x_alpha = new Box<str>();

      x.add(new A(10));
      x_alpha.add(new str("Hello World"));
}

Class Polygone<T>(){
   T $t; @private variable

   add(T t) {
      this.t = t;
   }

   T get() {
      return t;
   }
}
```

- Example code of multiple inheritances

```
Syntax:

Class JDM(){
    static dataType attribute;
    dataType $$attribute;
}

Class Supra(Car, JDM){    //Car and JDM classes are inherited
    dataType attribute;
}

Class UnitedBravo(Car){    //Car classe are inherited
    dataType attribute;
}
```