COMPILER CONSTRUCTION

THE INNOVATORS

ARHAM SHARIF EB21102022
HAMZA ALAM HASHMI EB21102031
MUHAMMAD RIAZ EB21102077

INDEX

DATE	TITLE	SIGNATURE

LANGUAGE MAKING

Basic Syntax

Action	Command	Description	Example

Main	Main(){}	This is The Main Method	Main() {
			// rest of code
			}

Function/Method	func name(parameters){
	// rest of the body
	}
		This is the function method parameters can also be set to default values	func add(first: Int, second: Int) {
			return first + second
			}
			add(5,6) //output ->11

Constants Variable	let	the value can not be changed after defining	let name = “XYZ” name = “Hello” // error
Dynamic Variable	var	the value can be changed dynamically	var name = “Dynamic” name = “New Value”

Static Properties	static var
	/
	static func name()
	{}
		Static Properties will not initialized at launch of code unless they are called at first time, after calling it first time then they will remain loaded into memory until program terminates	class A {
			static let xyz = “xyz”
			static var ijk - “ijk”
			static func abc() {}
			}
			A.abc()
			A.xyz = “” // Error
			A.ijk = “Can be changed”

Array / Multi-Dimensional Array	[value, value]	list of elements of any data type but should be same data type	let a = [1,2,3] let b = [“”, “hehe”, “43”] let c = [User(), User()]
Dictionary	[key => value, key => value]	list of key-value pair, keys should be unique	let dict = [“Hello” : “World”, “Key” : “Value”]
Splitter	,		
Quotes	“\”Quote””	the first and last double quoted comma is string and the first and last double quoted comma after backslash (\) is double quoted commas	“ \”this is a quote defined in a string” said by xyz ”
Comment	// or	single line comment	// let a = 1
Comment Body	/* comment body */	Comments Multiple Lines	/* let b = 2 let c = 3 */

Loops

Action	Command	Description	Example
for loop	for i in range(0,1) { }	loop will execute two times i = 0 i = 1	same as command

while loop	while (i <= 2) {
	}
		loop will execute until true condition	i = 0
			while (i <= 2) {
			i += 1 or i = i + 1
			print(i)
			}
			output
			1,2,3

semicolon	;	ends statement	let a = 1;

return	return	return the value
		or end the function
			func abc() {
			return 0;
			}

print	print()	prints the output	print("Hello World")
exit	exit()	terminates program execution	exit();
Conditions			
Action	**Command**	**Description**	**Example**

1. if
2. else if or elif
3. else
	if (condition) {
	}
	else if (condition) {
	}
	else {
	}
		if then some condition and then the body of if and so on for else if and else	// only if
			if (i == 0) {
			}
			// if-else
			if (i == 0) {
			}
			else {
			}
			// if - else if - else
			if (i == 0) {
			}
			else (if i == 1) {
			}
			else {
			}

Operators			
Action	**Command**	**Description**	**Example**
Add	var = var + num var += num	adds a value to variable	a += 1 a = a + 1
Subtract	var = var - num var -= num	subtracts a value from variable	a -= 1 a = a - 1
Multiply	* OR *=	multiply a value from variable	a *= 5 a = a * 5
Divide	/	divide a value from variable	a = a / 5
Inc	++		
Dec	--		
Concat	string + string or var += var concat(arg1, arg2)		print("Hello" + "World") var a = "Hello" a += "World" print(a) -> HelloWorld
And	&&		
Or	||		
Equal Comparision	== OR ===		0 == 0 -> true

Not Equal Comparision	!= OR !==		0 != 1 -> true
Greater Than OR Greater Than Equal To	< OR <=		10 >= 10 -> true
Less Than OR Less Than Equal To	> OR >=		10 <= 15 -> true
Power	1. ** 2. pow(base, exponent)		let a = pow(2,4) // 16
Square Root	sqrt(num)		let a = sqrt(4) -> 2
Modulus	% or %=		let isEven = a%2
Not	!		if(!notCompleted)
OOP			
Action	**Command**	**Description**	**Example**
Class	class name {}	the name of the class	class CompilerUBIT {}
Object	ClassName()	the object of the class	let object = ClassName()
Constructor	init() { }		
Destructor	deinit() { }		
Public Method (Default)	public func name() {}	by default if not specified method will be public	public func compile() {}
Protected	protected func name() {}	protected function can be called at public but can not be override	protected func compile() {}
Private	private func name() {}	neither override nor called on public only accessed within class methods	private func compile() {}
inheritance	class A: B {}	Multiple Inheritance not allowed	class Compiler: Construction {}

super	super.method()
	super.init()
		when child class function is override and but dev wants to run the child func also he calls super.functionName() or for constructor he calls super.init()	class A {
			func calculate() {}
			}
			
			class B:A {
			override func calculate() {
			// continue your method before super
			super.calculate() // this line will execute class A method
			// continue your method
			}
			}

Polymorphism	func abc(a: Int, b: Int){}	functions can be of same name but parameters can be different	let a = A() a.abc(1,2) a.abc(1.1, 2.2)

abstract class	abstract class A {
	abstract func abc()
	}
		Abstract Class Methods must be override by its child classes, if class is inherited by abstract class and no override method is called error will be shown in the line of class inheritance	abstract class A {
			abstract func abc()
			}
			class B: A {
			override func abc() {
			}
			}

this	init(name: str) {this.name = “Doe”}	this must be called to avoid same class variable attributes conflicting with function parameter names	func add(a: Int) { this.a = a}
Datatype			
Action	**Command**	**Description**	**Example**
int	0, 10, -5	integer data type	let a = 1
float	0.1 , 9.99	float numbers data type	let a = 1.5
char	"a" or 'a'	single character	let a = 'b'
str	"hello world"	two or more characters	let a = "hello world"
bool	true or false	boolean conditions	let isCompleted = true
null	null	Null Data type	let a = null
Functions			
Action	**Command**	**Description**	**Example**
replace	replace(replacing input, to replace, from string/array)	Replace the given word with user input from the string/array	let a = "hello world" replace(" ", "-", a)
find	find(input for find, from string/array)	Find the given word from given string/array	let a = "hello world" find("hello", a)
len	len(string/array)	returns the length of given string/array	let a = "hello" print(len(a)) // 5
Special Characters			
Action	**Command**	**Description**	**Example**
&double;	"&double;"	Add Double Quote	let newline = "a&double;b" return "a"b"
&single;	"&single;"	Add Single Quote	let newline = "a&single;b" return "a'b"
\n	"\n"	Goes to next line	let newline = "line1\nline2" return "line1 line2"
** **	" "	Add Space	let newline = "a b" return "a b"
&back;	"&back;"	Add Back Slash	let newline = "a&back;b" return "a\b"

Context-Free Grammar (CFG)

Variable Declaration
<variable_dec> -> <keyword> <datatype> <identifier> <data> ;

<keyword> -> let | var
<datatype> -> <type> | E
<data> -> E | = <dataFull>
<dataFull> -> <constant> | <func_call> | <expression> | <identifier>
<type> -> int | str | ..... | bool

Inc Dec
<inc_dec> -> <identifier> <inc_dec_op> ;

<inc_dec_op> -> ++ | --

Variable Assignment
<variable_ass> -> <identifier> <op> <data> | <inc_dec>;

<data> -> <constant> | <func_call> | <expression> | <identifier>
<op> -> = | += | ..... | -=

If Elif Else
<if> -> if (<condition>) <block> <else_if>
<else_if> -> <elseif_keywords> (<condition>) <block> <else_if> | else <block> | E
<block> -> {<body>}
<elseif_keywords> -> else if | elseif | elif

<condition> -> <data> <comparison_operator> <data> <logical>| <term> <logical>
<logical> -> E | <logical_operator> <condition>
<data> -> <term> | <expression>
<term> -> <identifier> | <constant>
<comparison_operator> -> == | != | ... | < | > | <= | >=
<logical_operator> -> && | ||

For Loop
<for_loop> -> for (<variable_dec>; <condition>; <variable_ass>) <block>

<block> -> {<body>}

While Loop
<while_loop> -> while (<condition>) <block>

<block> -> {<body>}

Break
<loop_body> -> {...<break>...}

<break> -> return true; | E

Continue
<loop_body> -> {...<continue>...}

<continue> -> return false; | E

Function Call
<func_call> -> <identifier> (<params>);

<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <term> | <expression>
<term> -> <identifier> | <constant>

Function Declaration
<func_dec> -> <datatype> func <identifier> (<params>) <block>

<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <variable_ass> | <identifier> | <constant>
<datatype> -> <type> | E
<type> -> int | str | ..... | bool
<block> -> {<body>}

Context-Free Grammar (CFG) - OOP

Class Declaration
<class> -> class <identifier> <extend> <block> | abstract class <identifier> <block>

<extend> -> E | : <identifier>
<block> -> {<body>}

Constructor
<constructor> -> init(<params>) <block>

<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <variable_ass> | <identifier> | <constant>
<block> -> {<body>}

Destructor
<destructor> -> deinit(<params>) <block>

<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <variable_ass> | <identifier> | <constant>
<block> -> {<body>}

Function Declaration
<oop_func_dec> -> <other_var> <access_modeifiers> <datatype> func <identifier> (<params>) <block>

<access_modeifiers> -> public | protected | private | E
<other_var> -> static | override | abstract | E
<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <variable_ass> | <identifier> | <constant>
<datatype> -> <type> | E
<type> -> int | str | ..... | bool
<block> -> {<body>}

Object or Function Call or Variable Call

<oop_func_var_call> -> <object>.<identifier> (<params>) | <object>.<identifier>;

<object> -> super | <objName_identifier>() | this
<params> -> E | <data> <more_args>
<more_args> E | , <params>
<data> -> <term> | <expression>
<term> -> <identifier> | <constant>

Variable Declaration

<variable_dec_oop> -> <other_var> <access_modeifiers> <keyword> <datatype> <identifier> <data> ;

<access_modeifiers> -> public | protected | private | E
<other_var> -> static | override | abstract | E
<keyword> -> let | var
<datatype> -> <type> | E
<data> -> E | = <dataFull>
<dataFull> -> <constant> | <func_call> | <expression> | <identifier>
<type> -> int | str | ..... | bool