

# COMPILER CONSTRUCTION

## THE INNOVATORS

ARHAM SHARIF EB21102022  
HAMZA ALAM HASHMI EB21102031  
MUHAMMAD RIAZ EB21102077

---

## INDEX

[illegible]

## 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

<b>while loop</b>	<pre>while (i &lt;= 2) {</pre>	loop will execute until true condition	<pre>i = 0 while (i &lt;= 2) { i += 1 or i = i + 1 print(i) } output 1,2,3</pre>
<b>semicolon</b>	<pre>;</pre>	ends statement	<pre>let a = 1;</pre>
<b>return</b>	<pre>return</pre>	return the value or end the function	<pre>func abc() { return 0; }</pre>
<b>print</b>	<pre>print()</pre>	prints the output	<pre>print("Hello World")</pre>
<b>exit</b>	<pre>exit()</pre>	terminates program execution	<pre>exit();</pre>

## Conditions

<u>Action</u>	<u>Command</u>	<u>Description</u>	<u>Example</u>
<b>1. if</b> <b>2. else if or elif</b> <b>3. else</b>	<pre>if (condition) { } else if (condition) { } else { }</pre>	if then some condition and then the body of if and so on for else if and else	<pre>// only if if (i == 0) { } // if-else if (i == 0) { } else { } // if - else if - else if (i == 0) { } else (if i == 1) { } else { }</pre>

## Operators

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

<b>Not Equal Comparison</b>	<code>!= OR !==</code>		<code>0 != 1 -&gt; true</code>
<b>Greater Than OR Greater Than Equal To</b>	<code>&lt; OR &lt;=</code>		<code>10 &gt;= 10 -&gt; true</code>
<b>Less Than OR Less Than Equal To</b>	<code>&gt; OR &gt;=</code>		<code>10 &lt;= 15 -&gt; true</code>
<b>Power</b>	1. <code>**</code> 2. <code>pow(base, exponent)</code>		<code>let a = pow(2,4) // 16</code>
<b>Square Root</b>	<code>sqrt(num)</code>		<code>let a = sqrt(4) -&gt; 2</code>
<b>Modulus</b>	<code>% or %=</code>		<code>let isEven = a%2</code>
<b>Not</b>	<code>!</code>		<code>if(!notCompleted)</code>
<b>OOP</b>			
<b>Action</b>	<b>Command</b>	<b>Description</b>	<b>Example</b>
<b>Class</b>	<code>class name {}</code>	the name of the class	<code>class CompilerUBIT {}</code>
<b>Object</b>	<code>ClassName()</code>	the object of the class	<code>let object = ClassName()</code>
<b>Constructor</b>	<code>init() { }</code>		
<b>Destructor</b>	<code>deinit() { }</code>		
<b>Public Method (Default)</b>	<code>public func name() {}</code>	by default if not specified method will be public	<code>public func compile() {}</code>
<b>Protected</b>	<code>protected func name() {}</code>	protected function can be called at public but can not be override	<code>protected func compile() {}</code>
<b>Private</b>	<code>private func name() {}</code>	neither override nor called on public only accessed within class methods	<code>private func compile() {}</code>
<b>inheritance</b>	<code>class A: B {}</code>	Multiple Inheritance not allowed	<code>class Compiler: Construction {}</code>
<b>super</b>	<code>super.method() super.init()</code>	when child class function is override and but dev wants to run the child func also he calls <code>super.functionName()</code> or for constructor he calls <code>super.init()</code>	<pre> 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   } } </pre>
<b>Polymorphism</b>	<code>func abc(a: Int, b: Int){}</code>	functions can be of same name but parameters can be different	<pre> let a = A() a.abc(1,2) a.abc(1.1, 2.2) </pre>

<b>abstract class</b>	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() { } }
<b>this</b>	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 }
<b>Datatype</b>			
<u><b>Action</b></u>	<u><b>Command</b></u>	<u><b>Description</b></u>	<u><b>Example</b></u>
<b>int</b>	0, 10, -5	integer data type	let a = 1
<b>float</b>	0.1 , 9.99	float numbers data type	let a = 1.5
<b>char</b>	"a" or 'a'	single character	let a = 'b'
<b>str</b>	"hello world"	two or more characters	let a = "hello world"
<b>bool</b>	true or false	boolean conditions	let isCompleted = true
<b>null</b>	null	Null Data type	let a = null
<b>Functions</b>			
<u><b>Action</b></u>	<u><b>Command</b></u>	<u><b>Description</b></u>	<u><b>Example</b></u>
<b>replace</b>	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)
<b>find</b>	find(input for find, from string/array)	Find the given word from given string/array	let a = "hello world" find("hello", a)
<b>len</b>	len(string/array)	returns the length of given string/array	let a = "hello" print(len(a)) // 5
<b>Special Characters</b>			
<u><b>Action</b></u>	<u><b>Command</b></u>	<u><b>Description</b></u>	<u><b>Example</b></u>
<b>&amp;double;</b>	"&double;"	Add Double Quote	let newline = "a&double;b" return "a"b"
<b>&amp;single;</b>	"&single;"	Add Single Quote	let newline = "a&single;b" return "a"b"
<b>\n</b>	"\n"	Goes to next line	let newline = "line1\nline2" return "line1 line2"
<b>&amp;nbsp;</b>	"&nbsp;"	Add Space	let newline = "a&nbsp;b" return "a b"
<b>&amp;back;</b>	"&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;

## Continue

<loop\_body> -> {...<continue>...}

<continue> -> return false;

## 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>}



