2025

COMPILER CONSTRUCTION

CSSE 501

MISS MARYAM FEROZE

Group “ ”

Rameen Khan	>	B22110106067
Kinza Akhter	>	B22110106035
Amna Fazal	>	B22110106015

KHATOON SCRIPT

LANGUAGE > JAVA

KEYWORDS

Access Modifiers

Private	>	chokidarAunty
Protected	>	fomoBeti
Public	>	blogBaji

Class, Object, and Interfaces

Class	>	channel
Interface	>	kittyParty
Extends	>	ammaOf
Implements	>	kitty

Control Flow

If	>	agar
Else	>	magar
ElseIf	>	agarMagar

Switch	>	dekhoBaji
Case	>	yaTo
Default	>	warna

For	>	apHoAo
Do - While	>	Ao - Ghumo
Break	>	rukBehn
Continue	>	chaloYar
Return	>	yeLo

Exception Handling

Try	>	pakdam
Catch	>	pakdai
Finally	>	yeToHoga
Throw	>	panchait
Throws	>	zadaPanchait

Data Types and Type Declarations

Int	>	digitDidi
Float	>	pointyDidi
Boolean	>	hanKNa
Void	>	khali
Char	>	alphaDidi
String	>	betaDidi
true	>	han
false	>	na

Modifiers

Static	>	chillKro // harJagah

Object and Class Keywords

This	>	yeh
Super	>	maharani
New	>	nayi

Basic Structure

```
public class ClassName {
    public static void main(String[] args) {
        // Code goes here
    }
}


blogBaji channel ClassName {
     blogBaji chillKro khali main( betaDidi[] args) {
          // Code
     }
}
```

Data Types

```
int c = 1000;
digitDidi c = 1000;

float e = 10.5f;
pointyDidi e = 10.5f

boolean h = true;
hanKNa h = true;

// Reference Data Types
String name = "Java";
alphaDidi name = “Java”;
```

Variables

```
final int y = 20;    // Constant variable
pakka digitDidi y = 20;
```

Control Flow

```
// If-else
if (condition) {
    // Code
} else if (condition) {
    // Code
} else {
    // Code
}

Agar (condition) {
    // Code
} agarMagar (condition) {
    // Code
} magar{
    // Code }



// Switch-case
switch (variable) {
    case value1:
        // Code
        break;
    case value2:
        // Code
        break;
    default:
        // Code
}dekhoBaji (variable) {
    yaTo value1:
        // Code
       rukBehn;
    yaTo value2:
        // Code
        rukBehn;
    warna:
        // Code
}
```

Loops

```
// For loop
for (int i = 0; i < 10; i++) {
    // Code
}
apHoAo (digitDidi i = 0; i < 10; i++) {
    // Code
}

// While loop
while (condition) {
    // Code
}
aoGhum (condition) {
    // Code
}

// Enhanced For loop
for (int item : array) {
    // Code
}

apHoAo(digitDidi item : array) {
    // Code
}
```

Methods

```
// Method Declaration
public returnType methodName(parameters) {
    // Code
    return value;
}

blogBaji digitDidi methodName(parameters) {
    //Code
    yeLo value; }
```

Classes and Objects

```
// Class
class MyClass {
    int x; // Field
    void myMethod() { // Method
        System.out.println("Hello!");
    }
}
channel MyClass {
    digitDidi x;            // Field
    khali myMethod() {      // Method
        System.out.println("Hello!");
    }
}

// Object
MyClass obj = new MyClass();
obj.myMethod();
// Object
MyClass obj = nayi MyClass();
obj.myMethod();
```

Arrays

```
// Declaration
int[] numbers = new int[5];
int[] numbers = {1, 2, 3, 4, 5};

// Declaration
digitDidi[] numbers = nayi digitDidi[5];
digitDidi[] numbers = {1, 2, 3, 4, 5};
```

Inheritance

```
class Parent {
    void display() {
        System.out.println("Parent class");
    }
}

channel Parent {
    khali display() {
        System.out.println("Parent class");
    }
}

class Child extends Parent {
    @Override
    void display() {
        System.out.println("Child class");
    }
}



channel Parent ammiOf Child  {
    khali display() {
        System.out.println("Child class");
    }
}
```

Interfaces

```
interface MyInterface {
    void method1();
}
```

```
kittyParty MyInterface {
    khali method1();
}
```

```
class MyClass implements MyInterface {
    @Override
    public void method1() {
        System.out.println("Implemented method");
    }
}
```

```
channel MyInterface kitty MyClass  {
    blogBaji khali method1() {
        System.out.println("Implemented method");
    }
}
```

Exception Handling

```
try {
    // Code that may throw an exception
} catch (ExceptionType e) {
    // Handle exception
} finally {
    // Code always executed
}
```

```
pakdam{
    // Code that may throw an exception
} pakdai(ExceptionType e) {
   // Handle exception
} yehToHoga{
   :) Code always executed }
```

Input/Output

```
import java.util.Scanner;

Scanner sc = new Scanner(System.in);
int num = sc.nextInt();
System.out.println("You entered: " + num);
```

```
leAoBehn java.util.Scanner;

Scanner sc = nayi Scanner(System.in);
digitDidi num = sc.nextInt();
System.out.println("You entered: " + num);
```

Packages

```
// Defining a package
package mypackage;

// Importing a package
import java.util.Scanner;
```

```
// Defining a package
Bazar mypackage;

// Importing a package
leAoBehn java.util.Scanner;
```

CODE

Objective:
Java program that takes an input number and prints its multiplication table according to the conditions If the number is less than 5, the table prints up to 15, but only for even multiples (using continue).

If the number is 5 or greater, it prints the table up to 10. If the number is greater than or equal to 10, it stops the loop using the break statement once the table exceeds 10.

//Code

```
import java.util.Scanner

public class DynamicMultiplicationTable {

    // Method to print the multiplication table
    public static void printTable(int number) {
        // Determine the range based on the input number
 int limit; // Declare the variable
if (number < 5) {
    limit = 15; // If number is less than 5, set limit to 15
} else {
    limit = 10; // Otherwise, set limit to 10
}



        for (int i = 1; i <= limit; i++) {
            // If number is greater than 10, stop the loop using break
            if (number >= 10 && i > 10) {
                break;
            }

            // If number is less than 5, continue after printing even numbers
only

            if (number < 5 && i % 2 == 0) {
                continue;
            }

            // Print the table
            System.out.println(number + " x " + i + " = " + (number * i));
        }
    }

    public static void main(String[] args) {
        // Create a scanner to read input
        Scanner scanner = new Scanner(System.in);

        // Ask the user for a number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Print the multiplication table based on the input
        printTable(number);
        // Close the scanner
        scanner.close();
    }
}
```

```
leAoBehn java.util.Scanner;

blogBaji channel DynamicMultiplicationTable {

    // Method to print the multiplication table
    blogBaji chillKro Khali printTable(digitDidi number) {
        // Determine the range based on the input number
        digitDidi limit;   //Declare the variable
       agar  (number < 5) {
           limit = 15;  //If number is less than 5, set limit to 15
       } magar {
           limit = 10;   //Otherwise, set limit to 10
      }



        apHoAo (digitDidi i = 1; i <= limit; i++) {
            //If number is greater than 10, stop the loop using break
            agar (number >= 10 && i > 10){
               rukBehn;
            }

            // If number is less than 5, continue after printing even
numbers only
            agar (number < 5 && i % 2 == 0) {
                chaloYar;
            }

            // Print the table
            System.out.println(number + " x " + i + " = " + (number *
i));
        }
    }
    blogBaji chillKro Khali main(betaDidi[] args) {
        // Create a scanner to read input
        Scanner scanner = nayi Scanner(System.in);

        //Ask the user for a number
        System.out.print("Enter a number: ");
        digitDidi number = scanner.nextInt();

        // Print the multiplication table based on the input
        printTable(number);

        // Close the scanner
        scanner.close();     }}
```

Traditional Java Code:

```
public class Person {
     private String name;
     private int age;
     private String address;
     private int postalCode;
     private char gender;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() { return name; }
    public int getAge() { return age; }
}
```

Updated Java Code

```
public class Person {
    private {
        String name;
        int age;
        String address;
        int postalCode;
        char gender;
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() { return name; }
    public int getAge() { return age; }
}
```

Traditional Java Code:

```
class Animal {  // Parent class
    public void eat() {
        System.out.println("This animal eats food.");
    }
}

class Dog extends Animal {  // Child class extends Parent
    public void bark() {
        System.out.println("The dog barks.");
    }
}
```

Updated Java Code

```
class Animal {  // Parent class
    public void eat() {
        System.out.println("This animal eats food.");
    }
}

class Animal extendsTo Dog {  // Child class extends Parent
```

```
    public void bark() {
        System.out.println("The dog barks.");
    }
}
```

Comments

- Single line
  `//              >          :)`

- Multi line
  `/* ... */       >          <3  <3`

LAB 2

Punctuators

```
Qoutes “”       >    ^ ^
Eg: ^string^

// Keeping other punctuators the same
, : ; ( ) [ ] { } . ?
```

Operators

```
//All same
```

Identifiers

Rules:

- Must start with an alphabet
- Must Contain a digit
- Can end with either a digit or alphabet
- Only ~ symbol can be used in between

eg: my~variable2

Regular Expression

[A] > All Alphabets both upper and lower case

[D] > All Digits[~] > ~ symbol

Regex:

`^[a-zA-Z][[a-zA-Z0-9~]*\d[a-zA-Z0-9~]*[a-zA-Z0-9]$|[a-zA-Z0-9~]*[0-9]$]+`

Constants

>Integer

Rules:

- Only Digits
- Can Start with -

Regular Expression

[D] > All Digits

[-] > - symbol

[/\] > Null

R: $(\text{“-”} + /\backslash) + D^{*}$

>Float

Rules:

- Must contain a floating point
- A digit must follow a floating point
- Can Start with - symbol

Regular Expression

[D]	>	All Digits
[-]	>	- symbol
[/\]	>	Null

R: $(\text{“-”} + /\backslash) + D^{*} + (\ . \ ) + D^{+}$

>String

Regular Expression

[A]	>	Digits , Alphabets and Symbols without \
[E]	>	Alphabets that can be with and without \ \n \t \b
[S]	>	Digits , Alphabets and Symbols occurring with \ \’ \” \\

R: $(\backslash(S + E) + E + A)^{*}$

CFGS

<Script>

```
->   <is_public><ScriptOptions>

<ScriptOptions>

      ->  <Interface> <HasInterface> <Main Class> <hasMultiClass>

          | <Main Class> <hasMultiClass>


<HasInterface> -> Null | <isPublic> <Interface> <HasInteface>

<HasMultiClass> -> Null | <Class> <HasMultiClass>
```

<Interface>

```
-> interface <id> <extends> { <IntefaceBody> }

     <isPublic> -> Null | Public

     <extends> -> Null | extends <id>

     <InterfaceBody> -> Null | <InterfaceOptions> <InterfaceBody>

     <InterfaceOptions> -> <InterfaceFeilds> | <InterfaceMethods>

                         | <Interface>


     <InterfaceFeilds> -> <DT> <IFeildsCases>

     <IFeildsCases> -> <id> = <const> ; | [] <id> = {<arrayList>}

     <arrayList> -> <const><arrayListAddOn>
```

<arrayListAddOn> -> Null | , <const><arralyList>

<InterfaceMethods> -> <returnType> <id> (<params>) ;
| Static <returnType> <id> (<params>) {<MST><ReturnST>}

<isStatic> -> Null | Static

<returnType> -> Void | <id> | <DT> <isArray>

<isArray> -> Null | []

<DT> -> int | char | boolean | float | String

<MainClass>

-> Class <id> <classOptions> {<MainMethod> <ClassBody>}

<classOptions> -> <extends> <Implements>

<extends> -> Null | extends <id>

<Implements> -> Null | implements <id> <implementsOptions>

<ImplementsOptions> -> Null | , <id> <implementsOptions>

<MainMethod> -> public static void main (String args[]) {<MST>}

<ClassBody> -> Null
| <AccessModifier> <classBodyOptions> <ClassBody>

<classBodyOptions> -> <Constructors> | <Class>
| <isStatic> <classBodyBlock> | { <MultiDec> }

<classBodyBlock> -> Void ID <method> | DataType <classBodyTypes> | final <Dec>;

<classBodyTypes> -> [] <id> <arrayInit>;| ID <classBodyDec>

<classBodyDec> -> <Method> | <Init>;

<Method>-> (<params>) {<MST> <ReturnSt>}

<ReturnSt> -> return <returnValue> ;

<returnValue> -> Null | KuchNahi | <const>

| (<expression>) | <TS> <id> <returnCases> | new <newOptions>

<TS> -> Null | this. | super.

<**newOptions>** -> <id> (<arguments>) | <DT> [] {<arrayList}

<returnCases> -> (<arguments>) | [<arrayArguments>] <dotCase><Options> | Null

<Options> -> <Assign> | <inc_dec>

<Constructor> -> <id> (<params>) { <TS()> <MST> }

<TS()> -> Null | this (<arguments>) ; | super (<arguments>) ;

<Class>

-> class <id> <classOptions> { <classBody> }

<SST>

-> <IDStatement>; | <TSIDStatement>; | <If_else> | <Do_While>

| <Switch> | <Dec> ; | <Try_Catch> | <For_Loop> | <Throw>;

<MST> -> Null | <SST> <MST>

<IDStatement> -> <id> <IDList>

<IDLIST> -> <dotCases> <Options>

| <id> = new <id> (<arguments>)

<Options> -> <Assign> | <inc_dec>

<TSIDStatement> -> this. <id> <TSIDList> | super . <id> <TSIDList>

<TSIDList> -> <dotCases> <Options> |

<Assign> -> <AssignOP> <Value>

<Value> -> <id><IDList2> | <const> | (<expression>)

<IDList2> -> Null | <Assign> | <dotCases> <Options> |

<AssignOp> -> = | <compoundAssign>

<arrayArguments> -> <int_const> | <inc_dec> <TS> <id>

                | <TS> <id> <subCase>

               <subCase> -> Null | <inc_dec>

<dotCases> -> Null | . <id> <dotCases> | [arrayArguments] <dotCases>

               | (<arguments>) <dotSubcases>

    <dotSubcases> -> Null | .<id> <dotCases>

<Dec> -> <DT> <DecType>

    <DecType> -> [] <id> <arrayInit> | ID <Init>

    <arrayInit> -> Null |= <ArrayType>

    <ArrayType> -> { <arrayList> } | new DT [<int_const>]

    <Init> -> Null | ,id <Init> | = <expression>

<If_Else> -> if (<expression>) { <MST> } <elseOptions>

    <elseOptions> -> Null | <elseIf> <elseOptions> | <else>

    <elseIf> -> elseIf (<expression>) { <MST> }

    <else> -> else { <MST> }

<Switch> -> Switch (<expression>) { <SwitchBody> }

    <SwitchBody -> <Cases> <SwitchBody> | <default>

    <Cases> -> case <expression> : <MST> break ;

```
     <default -> default : <MST>


<Do_While> -> Do { <MST> } while ( <expression> )



<Try_Catch> -> Try { <MST> } <catch> <tryCatchOptions>

     <catch> -> catch ( <ExceptionID> ID ) { <MST> }

     <tryCatchOptions> -> Null | <catch> <tryCatchOptions>

                        | finally { <MST> }


<Throw> -> throw new <ExceptionID> ( <arguments> )


<For_Loop> -> for ( <F1> ; <F2> ; <F3> ) { <MST> }

     <F1> -> <Dec> | <TS> <id> <DotCases> <Assign>

     <F2> -> Null | <expression>

     <F3> -> <inc_dec> <TS> <id> <DotCases>

          | <TS> <id> <DotCases> <inc_dec>




<params> -> Null | <DT> <id> <ParamList>

     <ParamList> -> ... | <paramType><addParam>

     <paramType> ->  Null | []

     <addParam> -> , <DT> <id> <paramType> <addParam> | Null


<arguments> -> Null | <expression> <addArgs>
```

<addArgs> -> , <expression> <addArgs> | Null

<expression>

<OR> -> <And> <OR'>

<OR'> -> Null | <||> <And> <OR'>

<And> -> <RO2> <And'>

<And'> -> Null | <&&> <RO2> <And'>

<RO2> -> <RO1> <RO2'>

<RO2'> -> Null | <rop_2> <RO1> <RO2'>

<RO1> -> <PM> <RO1'>

<RO1'> -> Null | <rop_1> <PM> <RO1'>

<PM> -> <MDM> <PM'>

<PM'> -> Null | <pm_op> <MDM> <PM'>

<MDM> -> <variable> <MDM'>

<MDM'> -> Null | <mdm_op> <variable> <MDM'>

<variable> -> <const> | (<expression>) | !<variable> | <TS> <id> <ref>

| <inc_dec> <TS> <id> <ref2>

<ref2> -> (<arguments>) | [<arrayArguments>] | Null

<ref> -> Null | <dotCases> <inc_dec> | <ref2>

LL1

<Script>

<NT>	First	Follow	cond1	con2
<Script>	Interface,Class, Public	$		
<hasInterface>	Null, Interface, Public	public,class		
<hasMultiClass>	Null, Class	$		

<Interface>

<NT>	First	Follow	cond1	cond2
<interface>	Public, interface	Public, class		
<is_public>	Null , Public	Interface , Class		
<extends>	Null, Extends	{ , id , implements		
<InterfaceOptions>	Public,void, int, char,float,boolean,String ,interface,Static	} , Datatype, void, id, Public, interface, static		
<InterfaceBody>	Null, Public, void, int, char,float, boolean,String, interface, Static , id	}		
<InterfaceFeilds>	Int, char, boolean, float, String	} , Datatype, void, id, Public, interface, static		
<IFeildsCases>	Id , [	} , Datatype, void, id, Public, interface, static		

<arrayList>	Null | ,	}		
<InterfaceMethods>	Void, int, char, boolean, Float, String, static	} , Datatype, void, id, Public, interface, static		
<isStatic>	Static, Null	Void, id, Datatype, {		
<returnType>	Int, char, boolean, float, String, void ,id	Id		
<isArray>	Null , [	id		

<MainClass>

<NT>	**First**	**Follow**	**C ond1**	**con2**
<MainClass>	Public, Class	Null ,Class		
<classOptions>	Extends , Implements, {	{		
<extends>	Null	{ , Implements		
	extends	{		
<implements>	Null	{		
	Implements			
<ImplementsOptions>	Null , **,**	{		
<MainMethod>	Public	DataType, AccessModifier, Void, { , final , id, class, }		
<classBody>	Null ,	}		
	AccessModifier, DataType, void, final, id, class, {			
<classBodyOptions>	id	AccessModfier, Void, Datatype,{ , final , id,class, }		
	class			
	Void, dataType, { , final,static			
<classBodyBlock>	Void , Datatype	Void , AccessModifier, DataType, { , final, id, class, },interface		
	final , {, Datatype			
	{			

<Feilds>	DataType, { , final	Void , AccessModifier, DataType, { , final, id, class, }		
<is_Final>	Null	dataType , {		
	Final			
<declarationOption>	dataType	Void , AccessModifier, DataType, { , final, id, class,		
	{			
<MultiDec>	Null	}		
	Datatype			
<Methods>	Void, DataType	Void , AccessModifier, DataType, { , final, id, class, }		
<returnStatement>	return	}		
<returnValue>	Null	;		
	NullKeyword			
	const			
	(			
	This, super,id			
	new			
<TS>	Null	id		
	this			
<newOptions>	Id	;		
	DataType			

<returnCases>	(	;		
	[			
	Null			
<Options>	Inc_dec	;		
	= , compoundAssign			
<Cosnstructor>	id	AccessModifier, Void, DataType, id, final, interface, class, }		

<TS()>	Null,	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw		
	this			
	Super			
<AccessModifier>	Null, AccessModifier	void , DataType, final , id , class, {		

<MST>

<NT>	First	Follow	cond1	con2
<MST>	id, if, for , do, Switch, Datatype, this, Super, try, throw	Return, }, break		
	Null			
<SST>	id	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
	This , super			
	If			
	do			
	Switch			
	DataType			
	try			
	For			
	throw			
<Dec>	DataType	;		
<decType>	[	;		
	id			
<arrayType>	{	;		
	new			
<arrayInit>	Null	;		
	{, new			
<init>	Null	;		

	,			
	=			
<DotCase>	Null	=, compoundAssign, inc_dec		
	.			
	[			
	(			
<dotSubCases>	Null	=, compoundAssign, inc_dec		
	.			
<If_Else>	if	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<elseOptions>	Null	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
	elseIf			
	else			
<elseIf>	elseIf	Else, elseIf, Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<else>	else	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<Switch>	Switch	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<SwitchBody>	Case	}		
	default			
<Cases>	Case	Case, default		
<default>	default	}		
<doWhile>	do	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<Try_Catch>	try	Null, id, if, for , do, Switch, Datatype,		

		this, Super, try, throw,Return, }, break		
<Catch>	Catch	Null,Catch, Finally		
<tryCatchOptions>	Null	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
	catch			
	finally			
<throw>	throw	;		
<For_Loop>	for	Null, id, if, for , do, Switch, Datatype, this, Super, try, throw,Return, }, break		
<F1>	DataType	;		
	This, super , id			
<F3>	Inc_Dec	)		
	this, super, id			
<F2>	Null,	;		
	Const, (, ! , ind_dec, this, super, id			
<params>	Null	)		
	DataType			
<paramList>	.	)		
	[, ,			
<paramType>	Null	), ,		
	[			
<addParams>	,	)		
	Null			
<arguments>	Null	)		
	const, (, ! , Inc_Dec, this, Super, id			
<addArguments>	,	)		

	Null			
<IDStatement>	Id	;		
<idList>	=, compoundAssign, inc_dec, [, (,	;		
	[			
	id			
<idList2>	(	;		
	=, compoundAssign			
	(			
	[			
	. , (, [			
<TSIDStatemment>	This ,	;		
	super			
<TSIDList>	= , compoundAssign, ind_dec, . , (,[	;		
	[			
<Assign>	=, compoundAssign	;		
<AssignOp>	=	Id, const, (, [, = , compoundAssign, Null		
	compoundAssign			
<Value>	Id,	;		
	const			
	(			
<arrayArguments>	Int_const,	]		
	inc_dec			
	This, super, id			
<subCase>	Null	]		
	Inc_dec			
<expression>	const , (, !, inc_dec, this, Super	, ,), : , ;		
<OR’>	Null	, ,), : , ;		
	||			
<AND>	const , (, !, inc_dec, this, Super	), : , ; ,||, ,		

<AND’>	Null	), : , ; ,|| , ,		
	&&			
<RO2>	const , (, !, inc_dec, this, Super	), &&, : , ; ,|| , ,		
<RO2’>	Null	). &&, : , ; ,|| , ,		
	rop_2			
<RO1>	const , (, !, inc_dec, this, Super	rop_2,), &&, : , ; ,|| , ,		
<RO1’>	Null, rop_1	rop_2,). &&, : , ; ,|| , ,		
<PM>	const , (, !, inc_dec, this, Super	Rop_1, rop_2,), &&, : , ; ,|| , ,		
<PM’>	Null , pm_op	Rop_1, rop_2,), &&, : , ; ,|| , ,		
<MDM>	const , (, !, inc_dec, this, Super	Pm_op, Rop_1, rop_2,), &&, : , ; ,|| , ,		
<MDM’>	Null	Pm_op, Rop_1, rop_2,), &&, : , ; ,|| , ,		
	mdm_op			
<Variable>	const	Mdm_op, Pm_op, Rop_1, rop_2,), &&, : , ; ,|| , ,		
	(			
	!			
	this, Super id			
	inc_dec			
<ref2>	null	Mdm_op,pm_op,rop_1,rop_2, ,&&,||,,,:,;,)		
	(			
	[			
<ref>	null	Mdm_op,pm_op,rop_1,rop_2, ,&&,||,,,:,;,)		
	(,[,.,incdec			
	Null , [,(			