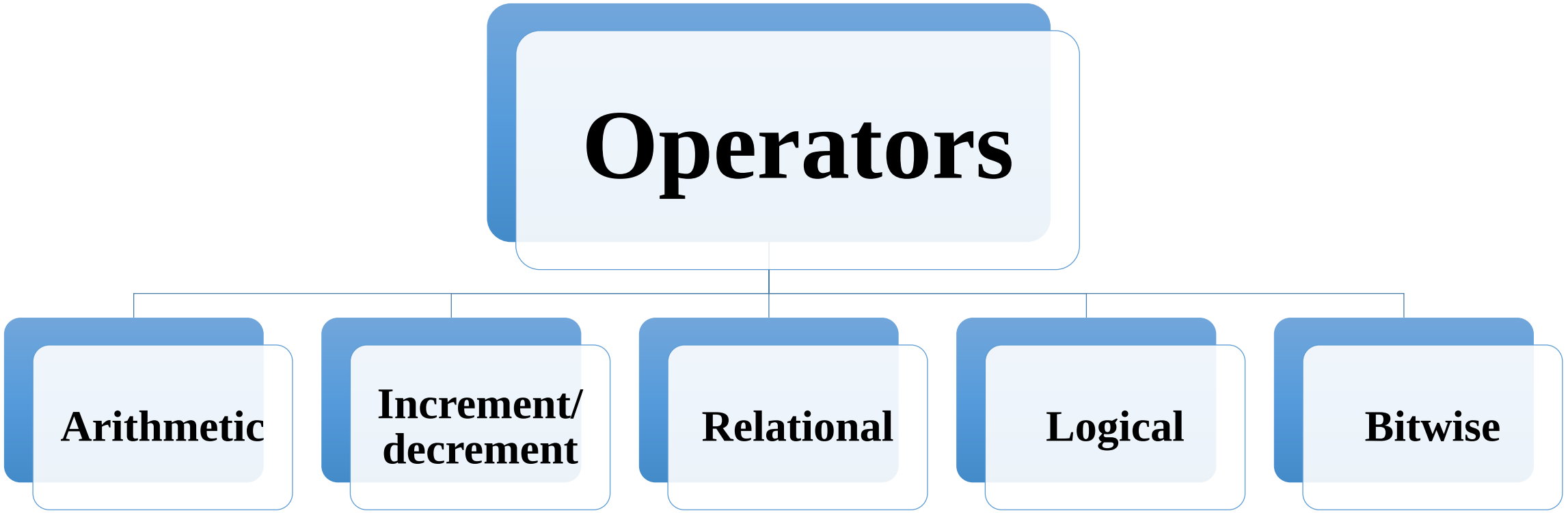
E1123 Computer Programming (a)

(Fall 2020)

Operators and Control Structures

INSTRUCTOR

DR / AYMAN SOLIMAN

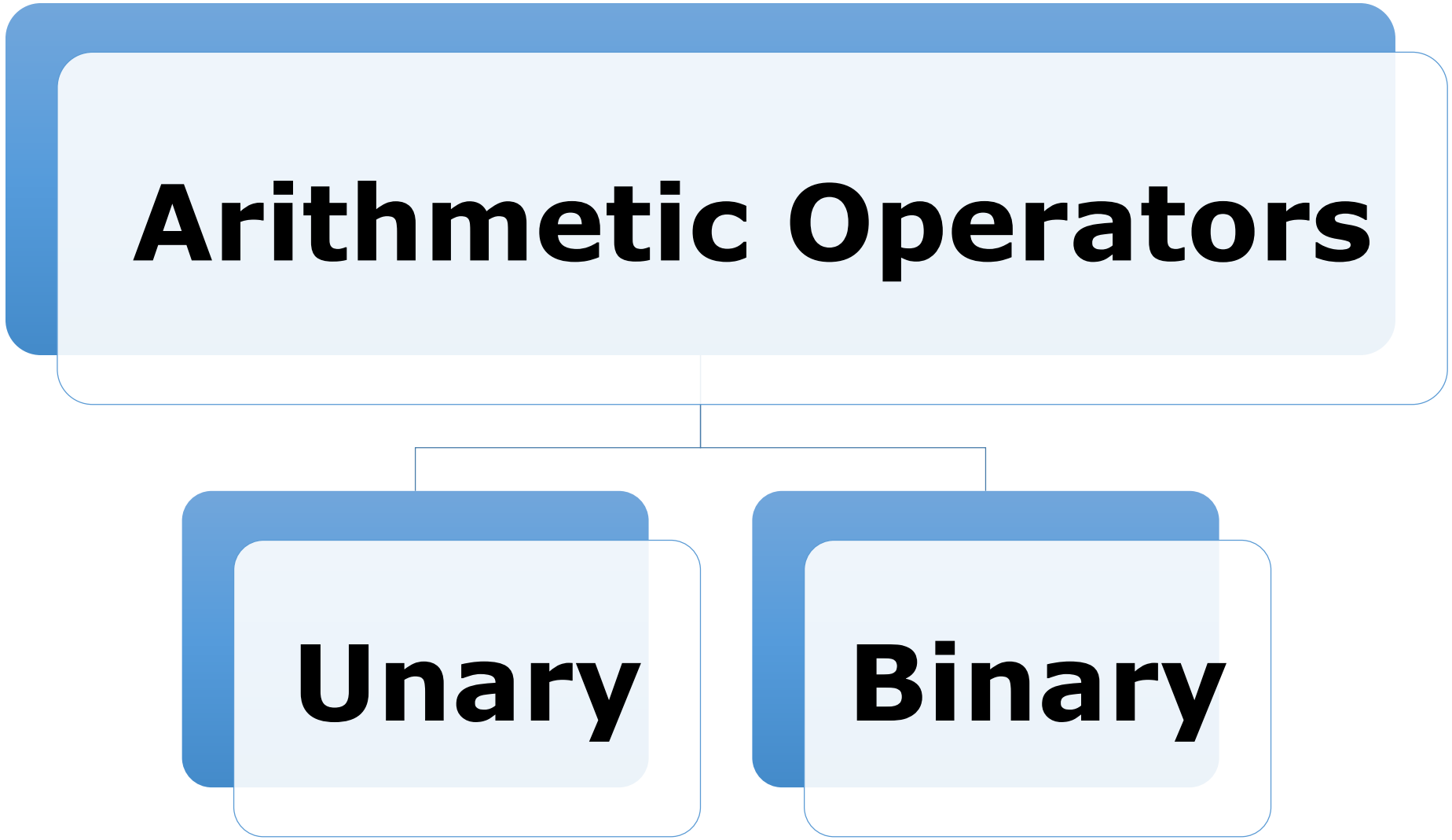

Operators
Arithmetic
Increment/ decrement
Relational
Logical
Bitwise

➢ Operators

An **operation** is a mathematical calculation involving zero or more input values (called **operands**) that produces an output value and in mathematics, operators such as + , - , * , / , …etc.

➢ Precedence and Associativity

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right


Arithmetic Operators
Unary
Binary

Arithmetic Operators

Unary

Operator	Symbol	Form	Operation
Unary plus	+	+x	Value of x
Unary minus	-	-x	Negation of x

Binary

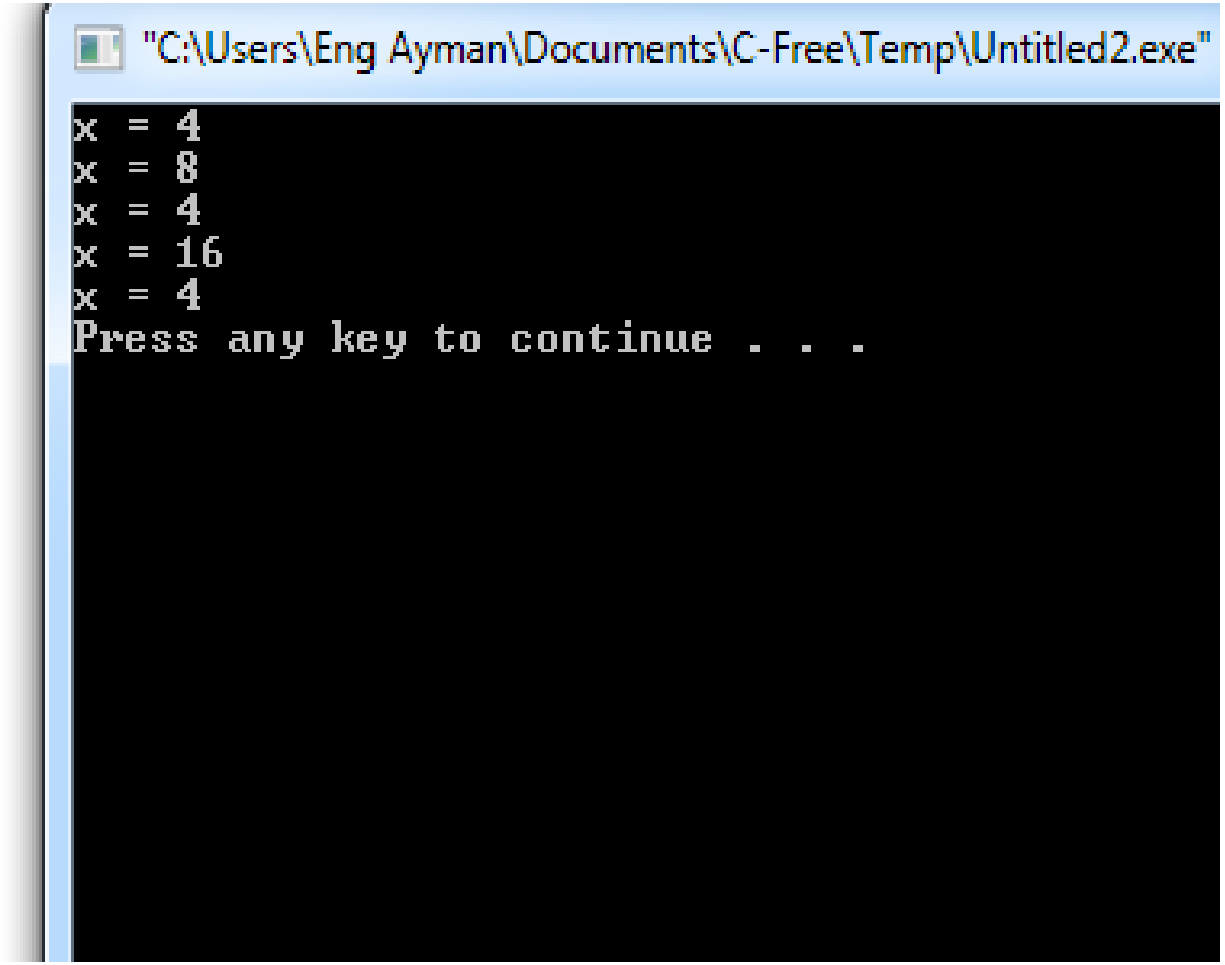
Operator	Symbol	Form	Operation
Addition	+	x + y	x plus y
Subtraction	-	x - y	x minus y
Multiplication	*	x * y	x multiplied by y
Division	/	x / y	x divided by y
Modulus (Remainder)	%	x % y	The remainder of x divided by y

➢ Arithmetic assignment operators

Operator	Symbol	Form	Operation
Assignment	=	x = y	Assign value y to x
Addition assignment	+=	x += y	Add y to x (x = x + y)
Subtraction assignment	-=	x -= y	Subtract y from x (x = x - y)
Multiplication assignment	*=	x *= y	Multiply x by y (x = x * y)
Division assignment	/=	x /= y	Divide x by y (x = x / y)
Modulus assignment	%=	x %= y	Put the remainder of x / y in x (x = x % y)

➢ Example

```
1  #include <iostream.h>
2  int main()
3  {
4      double x=10.5,y=4;
5      x=y;
6      cout<<"x = "<< x <<endl;
7      x += y;
8      cout<<"x = "<< x <<endl;
9      x -= y;
10     cout<<"x = "<< x <<endl;
11     x *= y;
12     cout<<"x = "<< x <<endl;
13     x /= y;
14     cout<<"x = "<< x <<endl;
15     return 0;
16 }
```

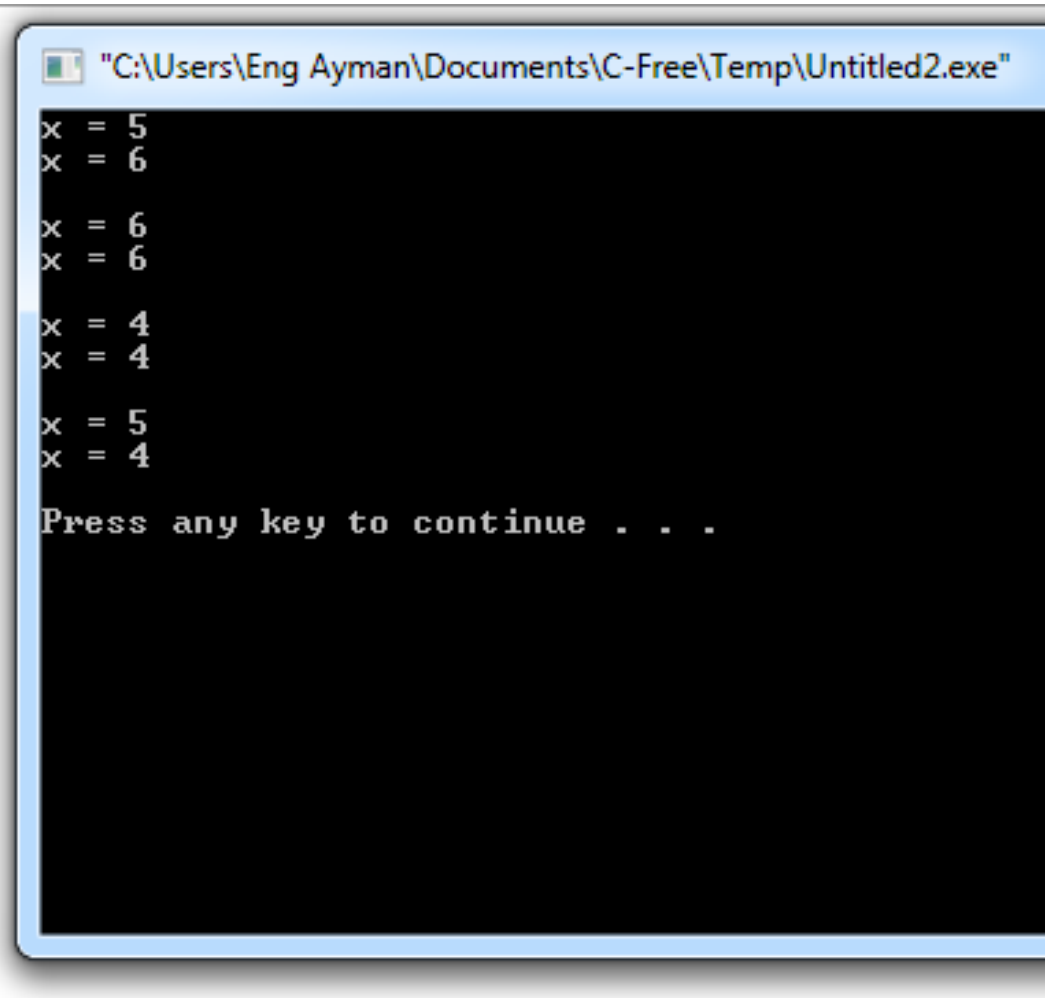


➢ Increment/decrement operators

Operator	Symbol	Form	Operation
Prefix increment (pre-increment)	++	++x	Increment x, then evaluate x
Prefix decrement (pre-decrement)	– –	– –x	Decrement x, then evaluate x
Postfix increment (post-increment)	++	x++	Evaluate x, then increment x
Postfix decrement (post-decrement)	– –	x– –	Evaluate x, then decrement x

➢ Example

```
1  #include <iostream.h>
2  int main()
3  {
4      double x=5;
5      cout<<"x = "<< x++ <<endl;
6      cout<<"x = "<< x <<endl;
7      cout<<endl;
8      x=5;
9      cout<<"x = "<< ++x <<endl;
10     cout<<"x = "<< x <<endl;
11     cout<<endl;
12     x=5;
13     cout<<"x = "<< --x <<endl;
14     cout<<"x = "<< x <<endl;
15     cout<<endl;
16     x=5;
17     cout<<"x = "<< x-- <<endl;
18     cout<<"x = "<< x <<endl;
19     cout<<endl;
20     return 0;
21 }
```



➢ Relational Operators (Comparisons)

Operator	Symbol	Form	Operation
Greater than	>	x > y	true if x is greater than y, false otherwise
Less than	<	x < y	true if x is less than y, false otherwise
Greater than or equals	>=	x >= y	true if x is greater than or equal to y, false otherwise
Less than or equals	<=	x <= y	true if x is less than or equal to y, false otherwise
Equality	==	x == y	true if x equals y, false otherwise
Inequality	!=	x != y	true if x does not equal y, false otherwise

➢ Logical operators

Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR	||	x || y	true if either x or y are true, false otherwise

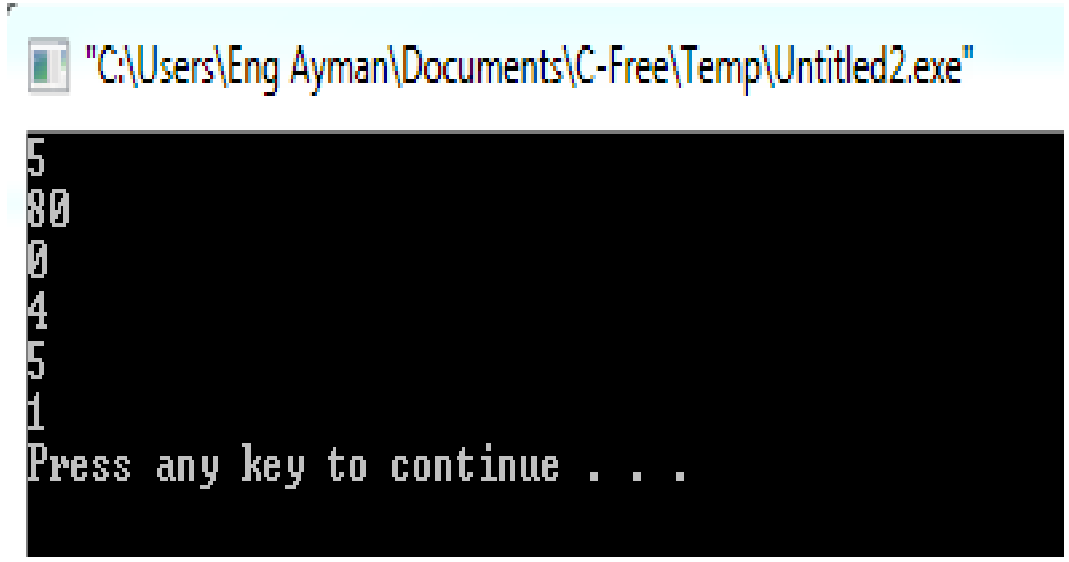
➢ Bitwise Operators

Using bitwise operators, it is possible to write functions that allow us to compact 8 Booleans into a single byte-sized variable, enabling significant memory savings at the expense of more complex code.

Operator	Symbol	Form	Operation
left shift	<<	x << y	all bits in x shifted left y bits
right shift	>>	x >> y	all bits in x shifted right y bits
bitwise NOT	~	~x	all bits in x flipped
bitwise AND	&	x & y	each bit in x AND each bit in y
bitwise OR	|	x | y	each bit in x OR each bit in y
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y

➢ Example

```
#include <iostream.h>
int main()
{
int x = 5, y = 4, z;
cout << x << '\n';    //x= 00000000 00000000 00000000 00000101 = 5
z = x << y;
cout << z << '\n';    //0101 << 4 = 01010000 = 80
z = x >> y;
cout << z << '\n';    //0101 >> 4 = 0000        = zero
z = x & y;
cout << z << '\n';    //0101 & 0100 = 0100     =4
z = x | y;
cout << z << '\n';    //0101 | 0100= 0101      =5
z = x ^ y;
cout << z << '\n';    //0101^0100= 0001        =1
return 0;
}
```

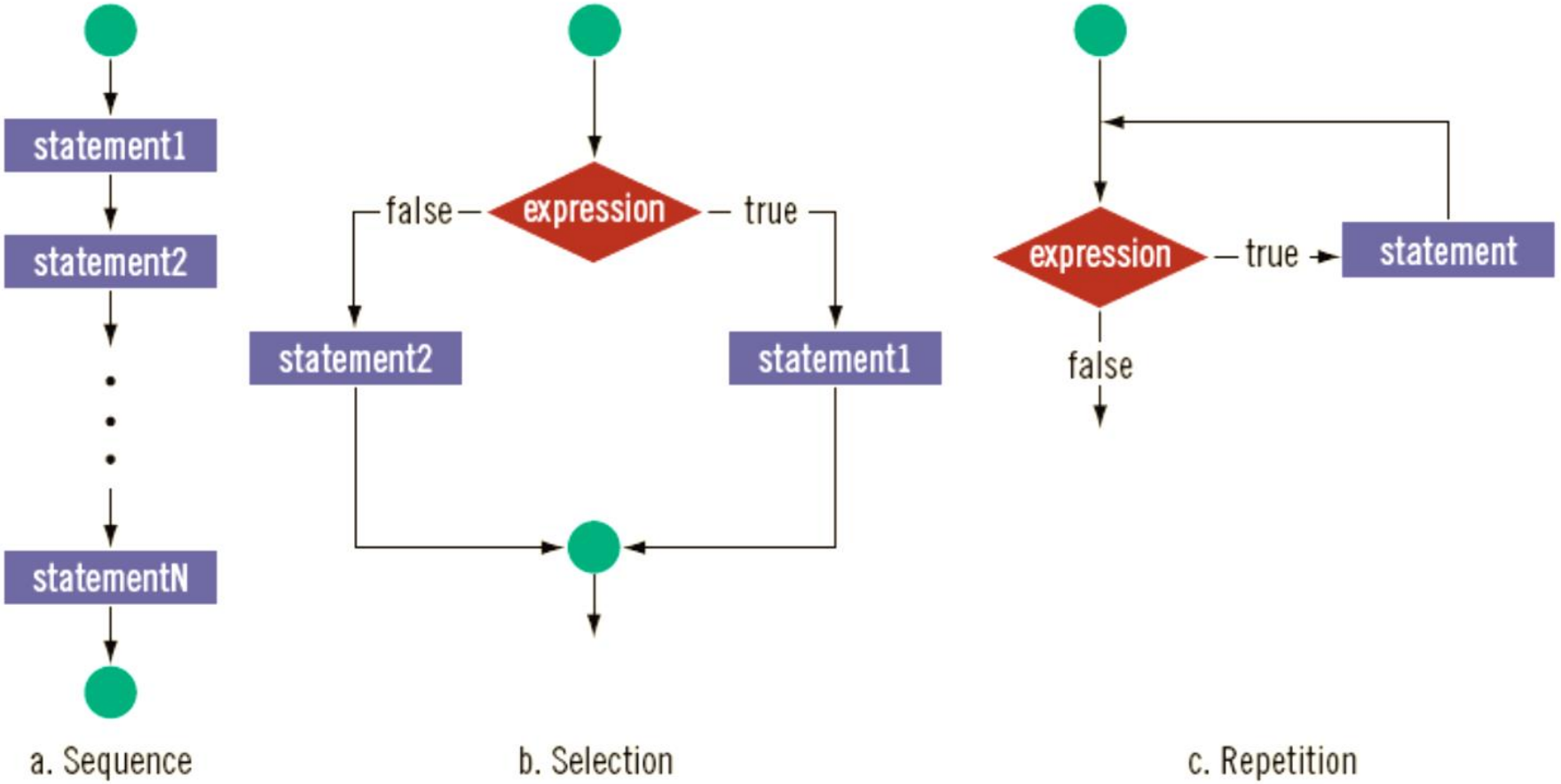


➢ Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) – looping

- Some statements are executed only if certain conditions are met
- A condition is met if it evaluates to true

➢ Control Structures (cont.)



➢ Relational Operators and Simple Data Types

You can use the relational operators with all three simple data types:

In the following example, the expressions use both integers and real numbers:

8 < 15 evaluates to →true

6 != 6 evaluates to →false

2.5 > 5.8 evaluates to →false

5.9 <= 7.5 evaluates to →true

➢ Comparing Characters

ASCII Value	Char	ASCII Value	Char	ASCII Value	Char	ASCII Value	Ch
32	' '	61	=	81	Q	105	i
33	!	62	>	82	R	106	j
34	"	65	A	83	S	107	k
42	*	66	B	84	T	108	l
43	+	67	C	85	U	109	m
45	-	68	D	86	V	110	n
47	/	69	E	87	W	111	o
48	0	70	F	88	X	112	p
49	1	71	G	89	Y	113	q
50	2	72	H	90	Z	114	r
51	3	73	I	97	a	115	s
52	4	74	J	98	b	116	t
53	5	75	K	99	c	117	u
54	6	76	L	100	d	118	v
55	7	77	M	101	e	119	w
56	8	78	N	102	f	120	x
57	9	79	O	103	g	121	y
60	<	80	P	104	h	122	z

Expression	Value of Expression	Explanation
`' ' < 'a'`	`true`	The ASCII value of `' '` is 32, and the ASCII value of `'a'` is 97. Because 32 < 97 is `true`, it follows that `' ' < 'a'` is `true`.
`'R' > 'T'`	`false`	The ASCII value of `'R'` is 82, and the ASCII value of `'T'` is 84. Because 82 > 84 is `false`, it follows that `'R' > 'T'` is `false`.
`'+' < '*'`	`false`	The ASCII value of `'+'` is 43, and the ASCII value of `'*'` is 42. Because 43 < 42 is `false`, it follows that `'+' < '*'` is `false`.
`'6' <= '>'`	`true`	The ASCII value of `'6'` is 54, and the ASCII value of `'>'` is 62. Because 54 <= 62 is `true`, it follows that `'6' <= '>'` is `true`.

➢ Relational Operators and the string Type

- Relational operators can be applied to strings
- Strings are compared character by character, starting with the first character
- Comparison continues until either a mismatch is found, or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
  - The shorter string is less than the larger string

➢ Example

Suppose we have the following declarations:

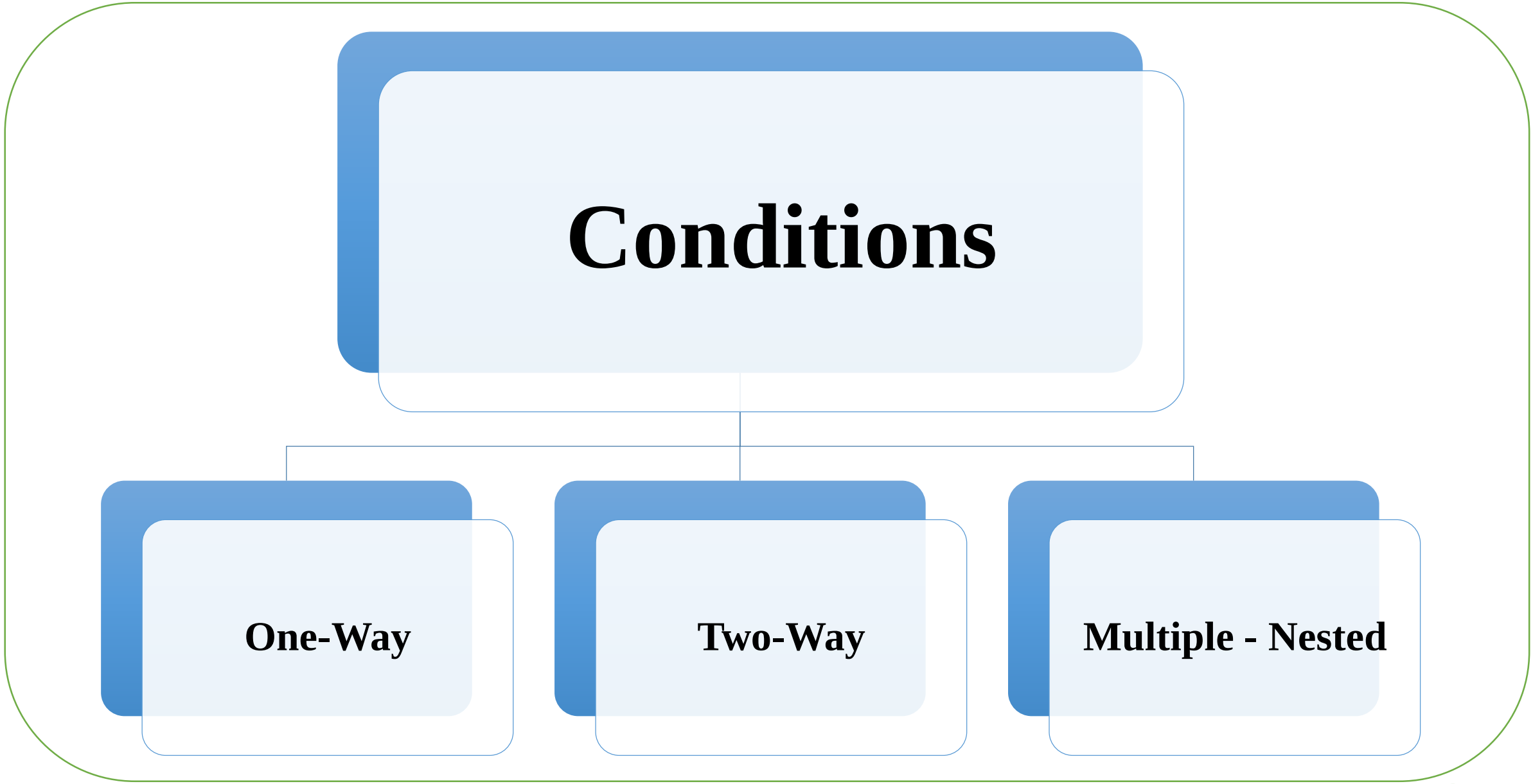
```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str4 = "Big";
```

Expression	Value	Explanation
`str1 < str2`	`true`	`str1 = "Hello"` and `str2 = "Hi"`. The first characters of `str1` and `str2` are the same, but the second character `'e'` of `str1` is less than the second character `'i'` of `str2`. Therefore, `str1 < str2` is `true`.
`str1 > "Hen"`	`false`	`str1` = `"Hello"`. The first two characters of `str1` and `"Hen"` are the same, but the third character `'l'` of `str1` is less than the third character `'n'` of `"Hen"`. Therefore, `str1 > "Hen"` is `false`.
`str3 < "An"`	`true`	`str3 = "Air"`. The first characters of `str3` and `"An"` are the same, but the second character `'i'` of `"Air"` is less than the second character `'n'` of `"An"`. Therefore, `str3 < "An"` is `true`.

➢ Example

Expression	Value	Explanation
`str4 >= "Billy"`	`false`	`str4 = "Bill"`. It has four characters and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"`, and `"Billy"` is the larger string. Therefore, `str4 >= "Billy"` is `false`.
`str5 <= "Bigger"`	`true`	`str5 = "Big"`. It has three characters and `"Bigger"` has six characters. Therefore, `str5` is the shorter string. All three characters of `str5` are the same as the corresponding first three characters of `"Bigger"`, and `"Bigger"` is the larger string. Therefore, `str5 <= "Bigger"` is `true`.

Expression	Value	Explanation
`str1 == "hello"`	`false`	`str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the ASCII value of `'H'` is 72, and the ASCII value of `'h'` is 104. Therefore, `str1 == "hello"` is `false`.
`str3 <= str4`	`true`	`str3 = "Air"` and `str4 = "Bill"`. The first character `'A'` of `str3` is less than the first character `'B'` of `str4`. Therefore, `str3 <= str4` is `true`.
`str2 > str4`	`true`	`str2 = "Hi"` and `str4 = "Bill"`. The first character `'H'` of `str2` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is `true`.


Conditions
One-Way
Two-Way
Multiple - Nested


Thank
you