COE 251

FUNDAMENTALS OF C PROGRAMMING

Dr. Eliel Keelson

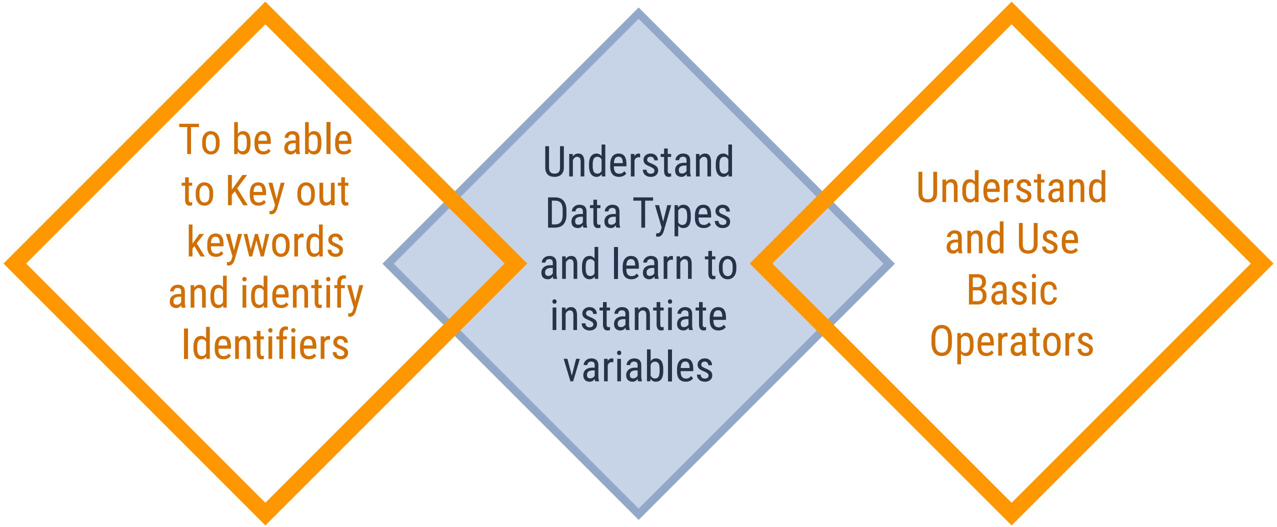
OUTLINE

KEYWORDS & IDENTIFIERS

C DATA TYPES

BASIC OPERATORS

INTENDED LEARNING OUTCOMES (ILOs)



To be able to Key out keywords and identify Identifiers

Understand Data Types and learn to instantiate variables

Understand and Use Basic Operators

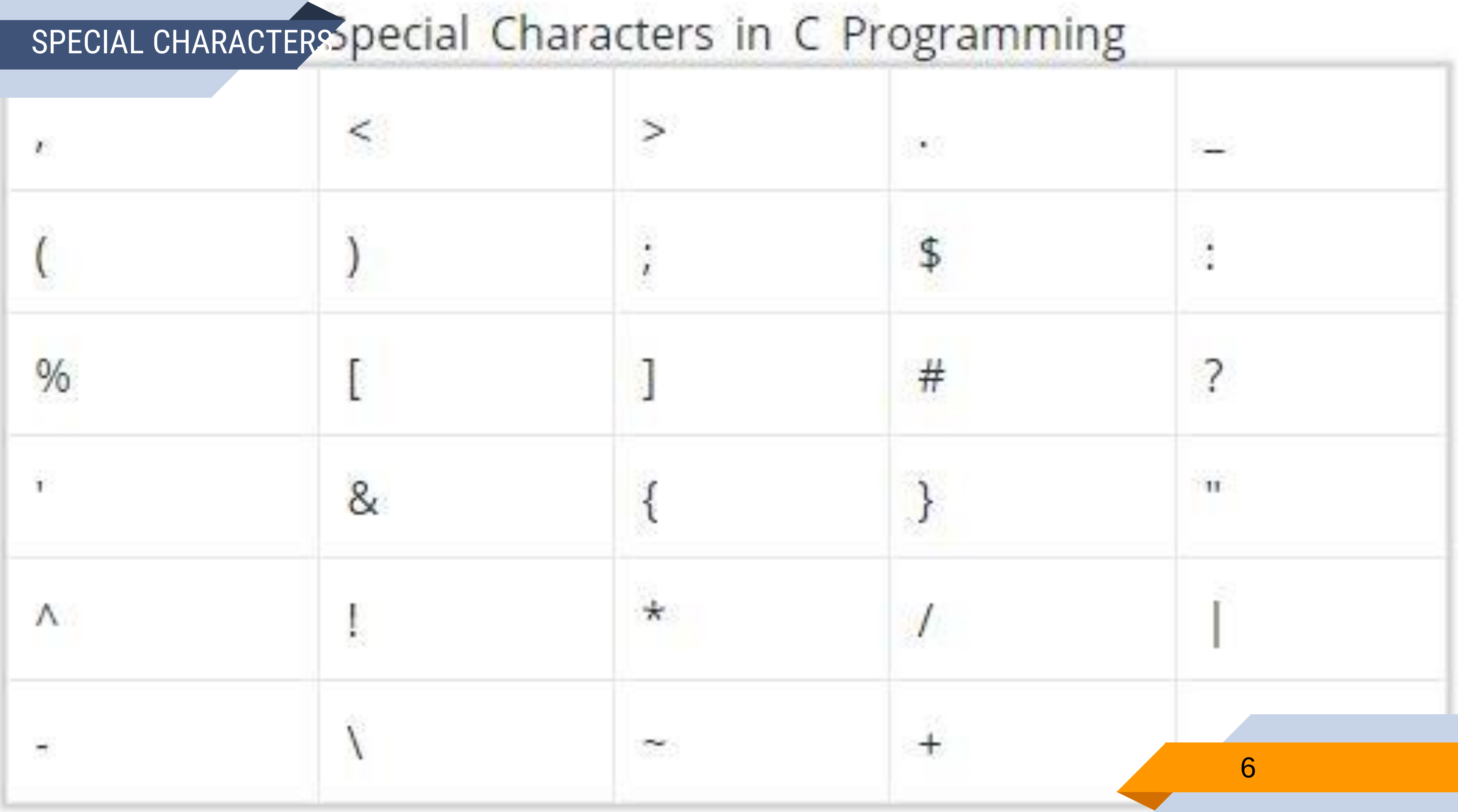
1

KEYWORDS AND IDENTIFIERS

THE CHARACTER SET OF C

- Character set is a set of alphabets, letters and some special characters that are valid in C language.
- C uses the uppercase letters A to Z, lowercase letters a to z, blank space, the digits 0 to 9 and certain special characters as building blocks to form basic program elements.

SPECIAL CHARACTERS



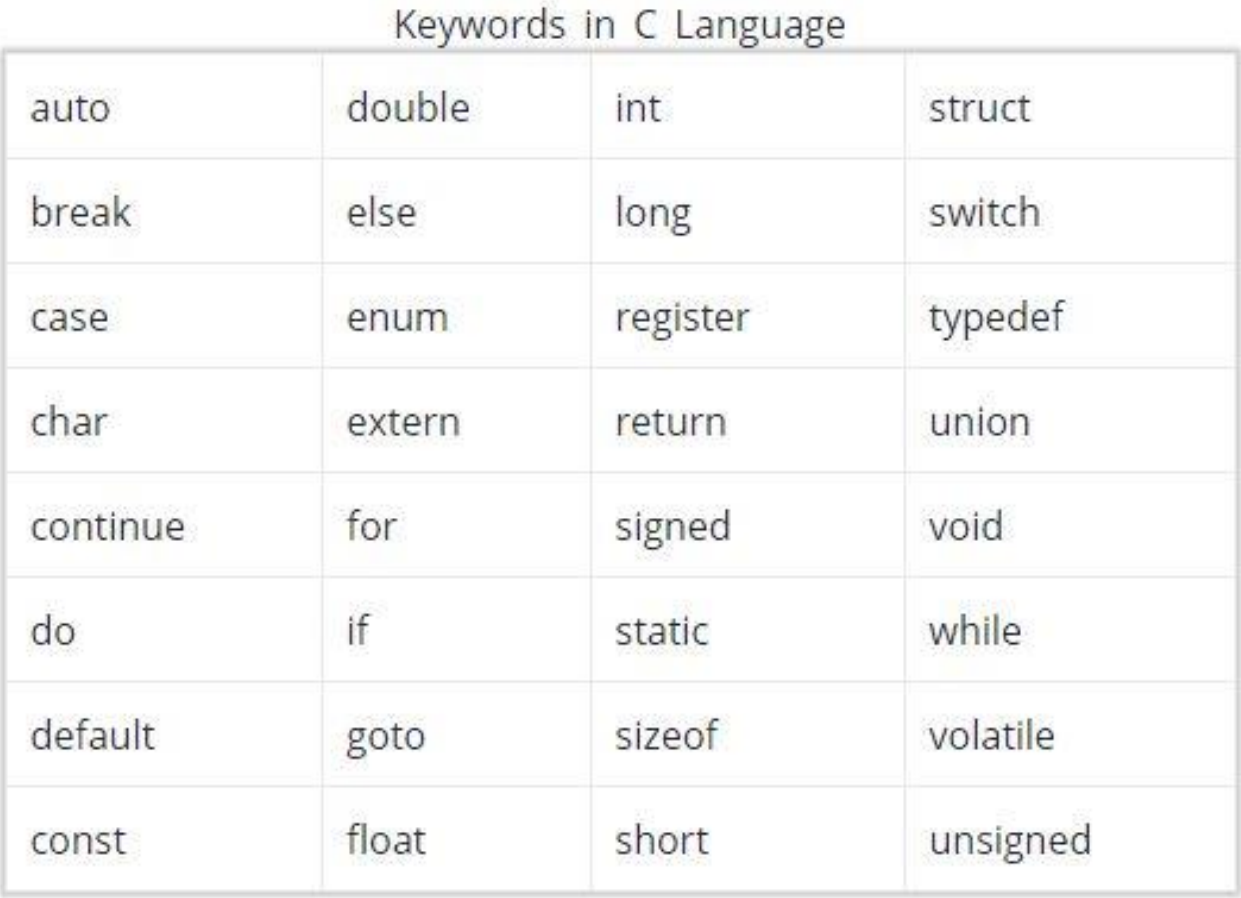
Special Characters in C Programming

,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"
^	!	*	/	|
-	\	~	+	

C KEYWORDS

- Keywords are predefined, reserved words used in programming that have special meanings to the compiler.
- As C is a case sensitive language, all keywords must be written in lowercase.
- The next slide presents a list of all keywords allowed in ANSI (standard) C.

COMMON KEYWORDS



Keywords in C Language

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

IDENTIFIERS

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program.
- For example: **money** = 500;

  age = 24;
- Here money and age are being used as identifiers.

RULES FOR WRITING IDENTIFIERS

- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore.
- An identifier cannot be a Keyword.

GOOD PROGRAMMING PRACTICE

- You can choose any name for an identifier (excluding keywords).
- However, if you give meaningful name to an identifier, it will be easy to understand and work on for you and your fellow programmers.
- For example storing the age of friend using an identifier called age instead afg

VARIABLES

- In programming, a variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).
- A variable can be simply described as a named (storage) space in (the computer's) memory.

VARIABLES

- Variable names are just the symbolic representation of a memory location. For example: playerScore = 95;
- Here, playerScore is a variable which is assigned value: 95.
- The value of a variable can be changed, hence the name '**variable**'.
- Since a variable is an identifier, all the rules governing naming identifiers apply to it.

VARIABLES

- Another important rule that governs the use of variables is that Variables ought to be declared before they are used to store data.
- To declare a variable simply means to state its **Data Type**
- More on Data Types would be covered in the next session

2

DATA TYPES

Types of Data

DATA TYPES

- As already stated Variables need to be declared before they are used.
- Such declarations are done simply by stating the **Data Type** of the Variable.
- These Data Types simply refer to the **type** and **size** of data associated with variables.

DATA TYPES

- In C, there are 4 basic/fundamental categories of data types.
- These data types are represented using these four keywords:

int => **Integers**

char => **Characters**

float => **Floating Points (decimals)**

double => **Floating Points (decimals)**

int - INTEGER DATA TYPES

- Integers are whole numbers that can have both positive and negative values but no decimal values. Example: 0, -5, 10
- In C programming, the keyword **int** is used for declaring integer variables. For example: **int age;**
- By declaring the variable **age** as having a data type of **int** means that **age** would be used to store only integers.
- As such the compiler would reserve just the right amount of memory space for storing an integer.

int - INTEGER DATA TYPES

- After declaring the data type of age, one can then go ahead and use it for storing a value. For example:

```
int age;
age = 24;
```

- Or it could have been written simply as:

```
int age = 24;
```

char - CHARACTER DATA TYPES

- Characters data types are any one thing (key) that can be found on the computer’s keyboard in addition with other special symbols.
- Variables that store one of such characters are declared with the keyword **char** . For example: **char choice = ‘y’;**
- It is important to note that characters in C are always enclosed in single quotation marks i.e. **‘ ’**
- By declaring a variable as **char** the compiler reserves just the right amount of space for holding it in memory

float / double – FLOATING POINT DATA TYPES

- Floating type variables can hold real numbers such as:2.34, -6.8
- You can declare a floating point variable in C by using either **float** or **double** keyword. For example:

```
float accountBalance = 3234.23;

double bookPrice = 23.99;
```

- In C, floating values can be represented in exponential form as well. For example:

```
float normalizationFactor = 22.442e2;
```

DIFFERENCE BETWEEN FLOAT AND DOUBLE

- The storage size of **double** (double precision float data type) is usually twice or more the storage size of a **float** (single precision float data type).

- Also a **float** variables has a precision of 6 digits whereas the precision of **double** is 14 digits.

DECLARING MULTIPLE VARIABLES

- In C, multiple variables of a particular data type can be declared and even assigned in one expression statement. The following are some examples:

```
int age, sum, result;
float weight = 49.36, cwa =73.98;
double balance, interest = 14.3856312;
```

DATA TYPE QUALIFIERS/MODIFIERS

- Data Type Qualifiers alter the meaning of base data types to yield a new data type.
- There are two main categories of Data Type Qualifiers. They are:

1. Size Qualifiers (**short** and **long**)
2. Sign Qualifiers (**signed** and **unsigned**)

SIZE QUALIFIERS/MODIFIERS

- The two size qualifiers; **short** and **long**, when used may have an effect on the storage space reserved for a particular data type.
- **short** hardly has any effect on the size. However, **long** often increases the natural storage space requirement for the data type it is used on. For example: **long int count;** would increase the storage space allocated for the integer variable **count**

SIGN QUALIFIERS/MODIFIERS

- The two size qualifiers; **signed** and **unsigned**, when used may have an effect on the capability of a variable to store both positive and negative values.
- When **signed** is used the variable is capable on storing both positive and negative values. However, when **unsigned** is used the variable can only hold positive values.
- It is important to note that, sign qualifiers can be applied to **int** and **char** types only.

DATA TYPE QUALIFIERS/MODIFIERS

- Having understood that the two size qualifiers are a direct opposite of each other, it would not be wise then to use them at the same time in any declaration.
- The same applies to the two sign modifiers. However, one of each category can be combined in a variable declaration. For example: **long unsigned int age = 25;**
- It is important to note that, it is not mandatory to use any of these qualifiers when declaring a variable's data type.

3

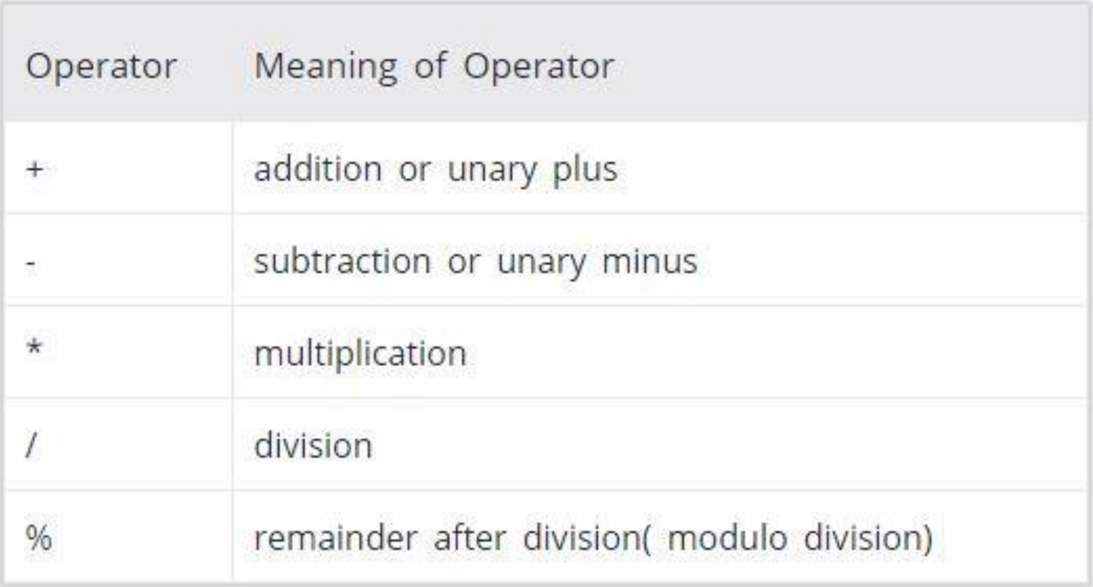
C PROGRAMMING OPERATORS

C PROGRAMMING OPERATORS

- An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition.
- C programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as: Arithmetic Operators, Increment and Decrement Operators, Assignment Operators, Relational Operators, Logical Operators, Conditional Operators, Bitwise Operators, Special Operators

ARITHMETIC OPERATORS

- An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).



Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

ARITHMETIC OPERATORS

- These operators work just as explained in the table.
- It is however important to note that in C that the result of an integer division always results in an integer. For example:

```
int a = 9, b = 4, c;
c = a/b;
```

- The result of **c** would be **2** and not **2.25** just because the values divided are integers.

INCREMENT AND DECREMENT OPERATORS

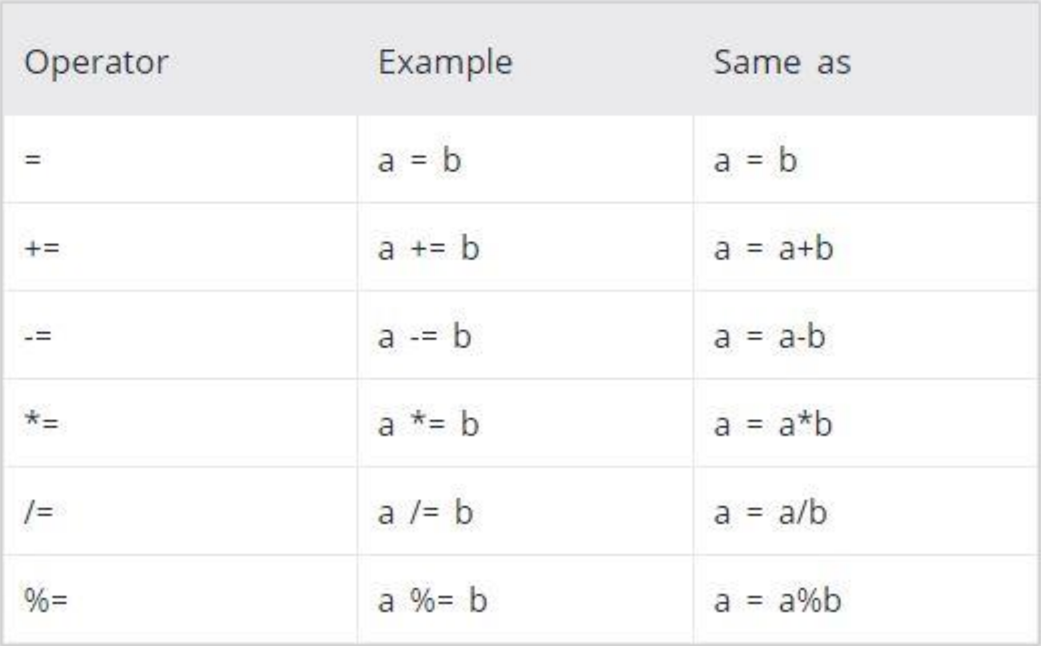
- C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.
- Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1.
- These two operators are unary operators, meaning they only operate on a single operand.

INCREMENT AND DECREMENT OPERATORS

- For example: **int a = 10, b = 20;**
- **++a** would therefore result in **11** and **--b** would result in **19**.
- These are prefix version of these operators.
- Postfix versions of these would look like **a++** and **b--**
- Depending on where and how they are used there **may** be a difference between the two versions.

ASSIGNMENT OPERATORS

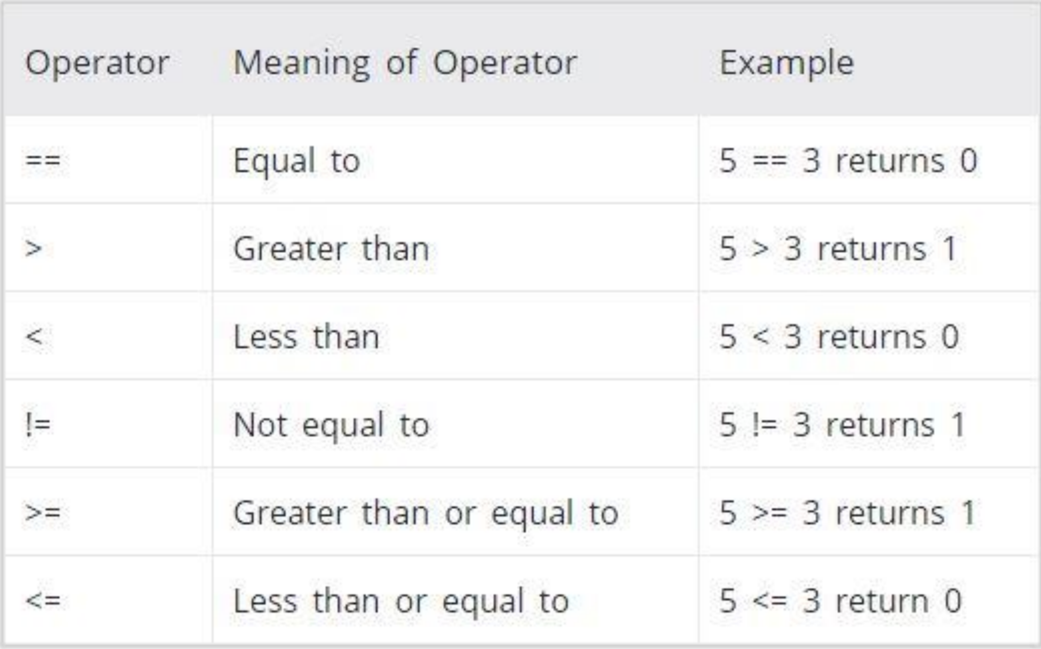
- An assignment operator is used for assigning a value to a variable. The most common assignment operator is **=**



Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

RELATIONAL OPERATORS

- A relational operator checks the relationship between two operands. If the relation is **true**, it returns **1**; if the relation is **false**, it returns value **0**.
- Relational operators are used in decision making and loops.



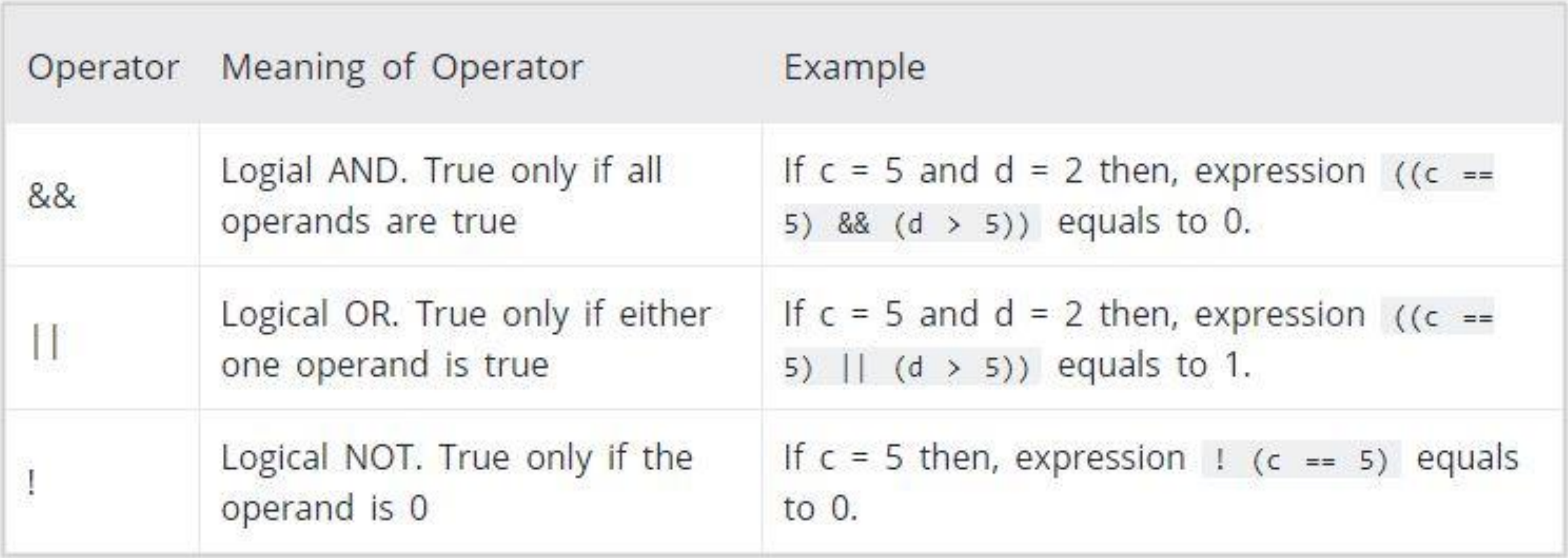
Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

LOGICAL OPERATORS

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.

- Logical operators are commonly used in decision making in C programming.

LOGICAL OPERATORS



Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c == 5) && (d > 5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c == 5) || (d > 5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `! (c == 5)` equals to 0.

LOGICAL OPERATORS

- **(a == b) && (c > 5)** evaluates to 1 because both operands **(a == b)** and **(c > b)** is 1 (true).
- **(a == b) && (c < b)** evaluates to 0 because operand **(c < b)** is 0 (false).
- **(a == b) || (c < b)** evaluates to 1 because **(a = b)** is 1 (true).
- **(a != b) || (c < b)** evaluates to 0 because any one of the operands **(a != b)** or **(c < b)** is 0 (false).

LOGICAL OPERATORS

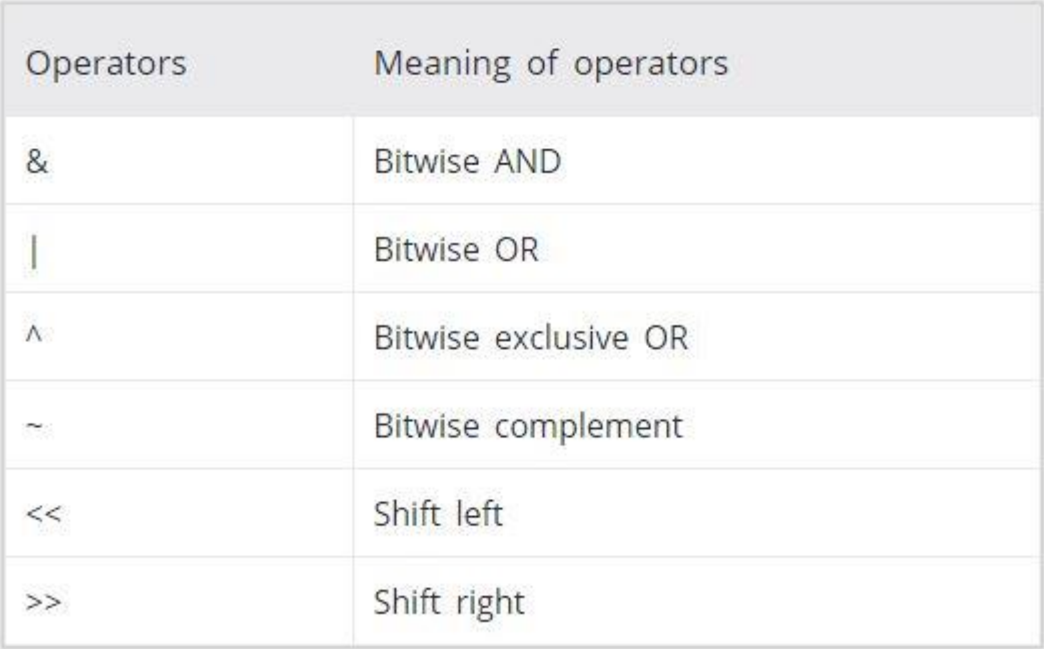
- **!(a != b)** evaluates to 1 because operand **(a != b)** is 0 (false). Hence, **!(a != b)** is 1 (true).

- **!(a == b)** evaluates to 0 because **(a == b)** is 1 (true). Hence, **!(a == b)** is 0 (false).

BITWISE OPERATORS

- During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.
- Bitwise operators are used in C programming to perform bit-level operations.

BITWISE OPERATORS



Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

COMMA OPERATOR

- As demonstrated in previous examples, the Comma operator is used to link related expressions together. For example:

```
int a, c = 5, d;
```

- By now it should be quite obvious that all expression statements in C end with a semi comma (;)

SIZEOF OPERATOR

- The **sizeof** is an unary operator which returns the size of data in bytes of various programming entities (e.g. constant, variables, array, structure etc).
- For example:
- **int a;**
- **sizeof(a);**

SIZEOF OPERATOR

- The **sizeof** is an unary operator which returns the size of data in bytes of various programming entities (e.g. constant, variables, array, structure etc).
- For example:
- **int a;**
- **sizeof(a);**

THANKS!

Any questions?

You can find me at

elielkeelson@gmail.com & ekeelson@knust.edu.gh