

C programming language pt.2

Error types:

- | | |
|-------------|-------------|
| 1. Syntax | 4. Logical |
| 2. Run-time | 5. Semantic |
| 3. Linker | |

Syntax -> compilation time

Run-time -> execution time

Linker -> while executable file is being created.

Logical -> logical thinking of the developer.

Semantic -> when compiler does not understand statements.

Literals:

Integer literals:

- Integer literals can be a decimal, octal or hexadecimal constant.
- Prefix: 0x or 0X for hexadecimal, 0 for octal and nothing for decimal
- Suffix: is a combination of U and L for unsigned and long
- Suffix can be uppercase or lowercase and can be in any order.

Floating-point literals:

- Floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
- You can represent floating-point literal either in decimal form or exponential form.
- While representing decimal form, you must include integer part, decimal point and fractional part.
- While representing exponential form, you must include the integer part and exponent part.
- The signed exponent is introduced by e or E.

String literals:

- You can break a long line into multiple lines using string literals and separating them using white spaces.

Operators:

a) Relational

Table



Operator	Description
<	Less than operator. Returns true if the left-hand operand is less than the right-hand operand, false otherwise.
>	Greater than operator. Returns true if the left-hand operand is greater than the right-hand operand, false otherwise.
<=	Less than or equal to operator. Returns true if the left-hand operand is less than or equal to the right-hand operand, false otherwise.
>=	Greater than or equal to operator. Returns true if the left-hand operand is greater than or equal to the right-hand operand, false otherwise.
==	Equality operator. Returns true if the operands are equal, false otherwise.
!=	Inequality operator. Returns true if the operands are not equal, false otherwise.

b) Arithmetic

Table



Operator	Description
++	Unary increment operator. Increments its operand by 1.
--	Unary decrement operator. Decrements its operand by 1.
*	Binary multiplication operator. Multiplies two operands.
/	Binary division operator. Divides two operands.
%	Binary remainder operator. Returns the remainder of dividing two operands.
+	Binary addition operator. Adds two operands.
-	Binary subtraction operator. Subtracts one operand from another.

Prefix has priority over assignment.

Assignment has priority over postfix.

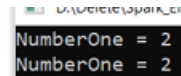
In pre-increment, value is first incremented and then used in expression.

In post-increment, value is first used in expression then incremented.

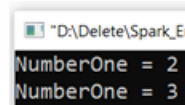
If we assign the post-incremented value to the same variable, then the value of that variable will not get incremented.

The pre-increment work normally in this case.

```
unsigned short NumberOne = 1;
NumberOne++; /* Post-increment -> NumberOne = 2 */
printf("NumberOne = %i\n", NumberOne); /* NumberOne = 2 */
NumberOne = NumberOne++; /* NumberOne = 2 */
printf("NumberOne = %i\n", NumberOne); /* NumberOne = 2 */
```



```
++NumberOne; /* Pre-increment -> NumberOne = 2 */
printf("NumberOne = %i\n", NumberOne); /* NumberOne = 2 */
NumberOne = ++NumberOne; /* NumberOne = 3 */
printf("NumberOne = %i\n", NumberOne); /* NumberOne = 3 */
```



c) Logical

Table

Operator	Description
!	Unary logical negation operator. Returns the opposite of its operand.
&	Binary logical AND operator. Returns true if both operands are true, false otherwise.
	Binary logical OR operator. Returns true if either operand is true, false otherwise.
^	Binary logical exclusive OR operator. Returns true if exactly one operand is true, false otherwise.
&&	Conditional logical AND operator. Returns true if both operands are true, false otherwise. If the left-hand operand is false, the right-hand operand is not evaluated.
	Conditional logical OR operator. Returns true if either operand is true, false otherwise. If the left-hand operand is true, the right-hand operand is not evaluated.

Bits and Bytes:

Bit is a single binary digit. It can have value (1/0), (true/false), (yes/no) or (on/off)

Byte is a group of 8 bits.

Byte can take any value between 0 and 255.

Byte can represent a character (letter, digit, punctuation symbol) using ASCII coding.

Bit is the smallest piece of information a computer can store.

A byte is the smallest amount of data that computers can read from or write to memory.

Computer memory is recognized as bytes and files as well.

Nibble is like a byte but has 4 bits.

Nibble can take any value between 0 and 15.

Word size depends on the architecture.

2 bytes at least can be combined to create a 16-bit word.

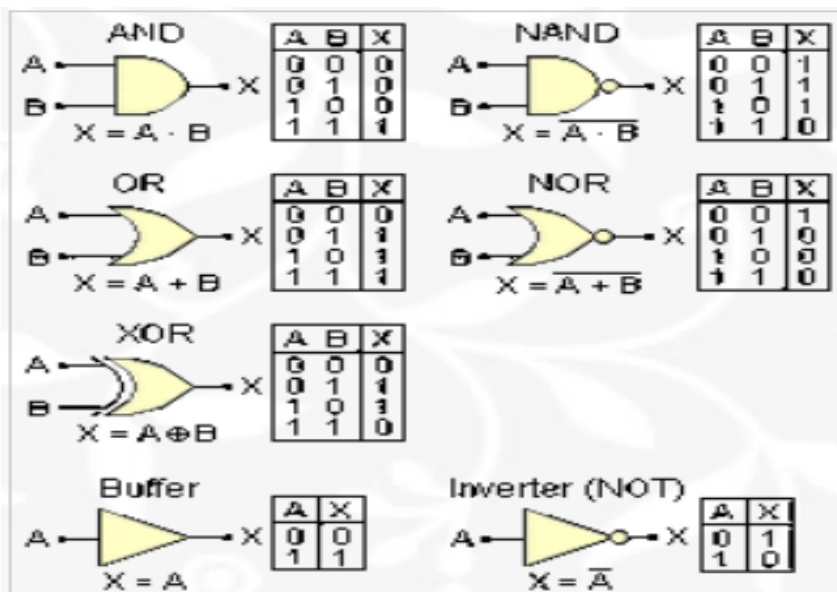
Dword can take any value between 0 and 65,535.

4 bytes can be combined to create a 32-bit word (double word) **DWORD**.

Lword can take any value between 0 and 4,294,967,296.

8 bytes can be combined to create a 64-bit word (long word) **LWORD**.

Sword can take any value between 0 and 1.84467441E19.



d) Bitwise

Table



Operator	Description
~	Unary bitwise complement operator. Flips the bits of its operand.
<<	Binary left-shift operator. Shifts the bits of its left-hand operand to the left by a number of positions specified by its right-hand operand.
>>	Binary right-shift operator. Shifts the bits of its left-hand operand to the right by a number of positions specified by its right-hand operand.
&	Binary AND operator. Returns a value where each bit is set to 1 only if the corresponding bits of both operands are 1.
	Binary OR operator. Returns a value where each bit is set to 1 if the corresponding bits of either operand are 1.
^	Binary XOR operator. Returns a value where each bit is set to 1 only if the corresponding bits of one operand are 1 and the other operand are 0.

Bit wise operators are useful in bit manipulation.

Bitwise AND is used as a bit mask.

Bit mask is used to mask out unwanted bits in a byte.

Bitwise OR is used as combine bit field.

Bitwise XOR is used to toggle any bit.

There are two types of shifting:

arithmetic considers signal.

logical doesn't consider signal.

Integer promotion:

char, short or enum are automatically promoted to int or unsigned int when operation is performed on them.

They first converted to int or unsigned int and then arithmetic is done on them.

If an int can represent all values of the original type, the value is converted to an int. Otherwise, it is converted to unsigned int.

To clear any bit in memory:

```
variable = variable & ~(1<< bit_position);
```

To set any bit in memory:

```
variable = variable | (1 << bit_position );
```

To set any bit in memory:

```
variable = variable ^ (1 << bit_position );
```

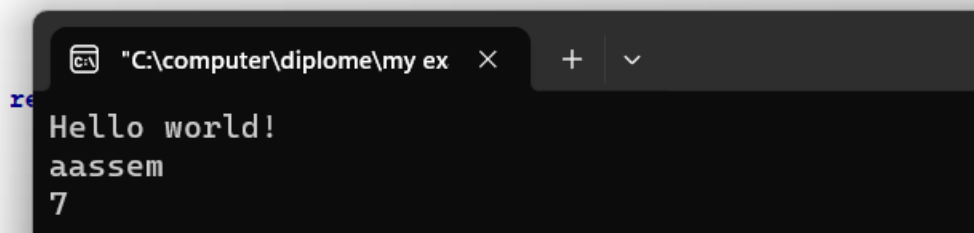
e)Assignment

Table



Operator	Description
<	Less than operator. Returns true if the left-hand operand is less than the right-hand operand, false otherwise.
>	Greater than operator. Returns true if the left-hand operand is greater than the right-hand operand, false otherwise.
<=	Less than or equal to operator. Returns true if the left-hand operand is less than or equal to the right-hand operand, false otherwise.
>=	Greater than or equal to operator. Returns true if the left-hand operand is greater than or equal to the right-hand operand, false otherwise.
==	Equality operator. Returns true if the operands are equal, false otherwise.
!=	Inequality operator. Returns true if the operands are not equal, false otherwise.

```
int x = printf("aassem\n"); // assigns number of characters
printf("%i\n", x);
```



- f) Conditional
g) Others

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators' associativity and precedence:

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

With some operators, the result is compiler dependent.

For example: the addition of two functions.

Decision making:

If statement:

```
if (/*condition_1*/) {  
    // to do  
}  
else if (/*condition_2*/) {  
    // to do  
}  
else {  
    // to do  
}
```