Learn Programming Basics (C Language)

LESSON #006 (Lab)
if, else if, else, ternary, switch case

Purpose

- if, else if and else statement is one of the basics in programming
- learn how to execute codes under specific condition

Objective

- learn to read code
- learn the rules of if, else if and else statement
- learn the different ways to create condition
- learn how condition is being evaluated

if, else if, else

```
if ( condition )                    -    basic structure, syntax
{

}
else if ( condition )
{

}
else
{

}
```

```
if ( condition )
{

}
else if ( condition
)
{
}
else if ( condition
)
{
}
else
{

}
```

if

- the start of **ANY <u>if</u>, <u>else if</u>** OR **<u>else</u>** statement

else if

- always pair up with the **<u>nearest if</u>** OR **<u>nearest else if</u>**
- otherwise compile ERROR

else

- always pair up with the **<u>nearest UNPAIRED if</u>** OR **<u>nearest else if</u>**
- otherwise compile ERROR

else example:

```
if ( condition )
{
   if ( condition )
    {}
    else if ( condition )            /*paired else if with if */
    {}
    else                             /*paired else with else if */
    {}
}
else                                 /*paired else with if */
{}
```

condition

```
if ( condition )
{

}
else if ( condition )
{

}
else
{

}
```

2 kind of condition requirement:

true:

- non-zero value
  - Example: -1, 1, true, 0.000001f

false:

- zero value
  - Example NULL, nullptr, 0, false 0.000000f

example:

```
int i = 1
if (i)
{
	printf(“a”);
	if ( --i )
	{
		printf(“b”);
	}
	else if ( ++i )
	{
		printf(“c”);
	}
	else
	{
		printf(“d”);
	}
}
```

What is he print out?

Answer: “ac”

ternary

```
(condition) ? (execute this if true)  : ( execute this if false );
```

if else form:

```
if ( condition )
{
        printf(“a”);
}
else
{
        printf(“b”);
}
```

ternary:

```
( condition ) ? printf(“a”) : printf(“b”);
```

if else form:

```
if ( condition )
{
        printf(“a”);
}
else if ( condition)
{
        printf(“c”);
}
else
{
        printf(“b”);
}
```

ternary:

```
 (( condition ) ? printf(“a”)  :  ((
condition ) ? printf(“c”) :
printf(“b”) ));
```

features

if one liner:

```
if (condition)
        printf("a");
else if (condition)
        printf("c");
else
        printf("b");
```

- you can write an "if" statement without using the scope { … } if you are just writing one line statement nested ternary is possible

```
(((condition) ? (execute this if true)  : ( execute this if false ) )
? ((condition) ? ((condition) ? (execute this if true)  :
(((condition) ? (execute this if true)  : ( execute this if false ) )
 ? (execute this if true)  : ( execute this if false ) )
 )
 : (((condition) ? (execute this if true)  : ( execute this if false ) )
? (execute this if true)  : ( execute this if false ) )
)
 : (((condition) ? (execute this if true)  : ( execute this if false ) )
 ? ((condition) ? (execute this if true)  : ( execute this if false ) )
  : ( execute this if false ) )
)
```

condition part 2

AND Operator, &&; OR Operator, ||

- AND truth table and OR truth table (Boolean algebra)

A	B	Output AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Output OR
0	0	0
0	1	1
1	0	1
1	1	1

AND Operator(&&); OR Operator(||);

- Play around with this example, and observe

```
int a = 1, b = 0;

if ( a && b )  printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

```
int a = 1, b = 0;

if ( a && ++b ) printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

```
int a = 1, b = 0;

if ( b && ++a ) printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

```
int a = 1, b = 0;

if ( a || b ) printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

```
int a = 1, b = 0;

if ( --a || ++b ) printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

```
int a = 1, b = 0;

if ( b || ++a ) printf(“true”);
else printf(“false”);

printf(“a=%d, b=%d”, a, b);
```

AND Operator(&&); OR Operator(||);

- Explanation
  - the left to right evaluation
  - learning how to read the code

if (a && b);

- if a is true, b is then evaluated to check if true or false
- if a is false, **b is NOT evaluated** and the whole condition returns false

if (a || b);

- if a is true, **b is NOT evaluated** and the whole condition returns true
- if a is false, b is then evaluated to check if true or false

<u>Topic on Short Circuiting</u>

- Similarly to * and / operator that falls under the same precedence level, short circuiting is what it called, **to automatic "parenthesize" the execution order** using the "first come first serve" method
- && - AND operator
- || - OR operator

Short Circuiting Example:

- if (a && b || c) {}
  - auto deduction: (a && b) || c
  - a is being evaluated
    - if a is false, (a && b) = false
      - c is then evaluated
      - if c is true, condition = true
      - else false
    - if a is true, b is then evaluated
      - if b is false, (a && b) = false
        - c is then evaluated
      - if b is true, (a && b) = true, c is NOT evaluated
- if (a || b && c) {}
  - auto deduction: (a || b) && c

<u>**Short Circuiting:**</u>

- Therefore it is <u>**important to add your own parentheses**</u> if <u>**you want the desired results**</u>
- or leave it if you know what you are doing

Example added parentheses:

- if (a && (b || c)) {}
  - auto deduction: a && (b || c)
- if (a || (b && c)) {}
  - auto deduction: a || (b && c)