

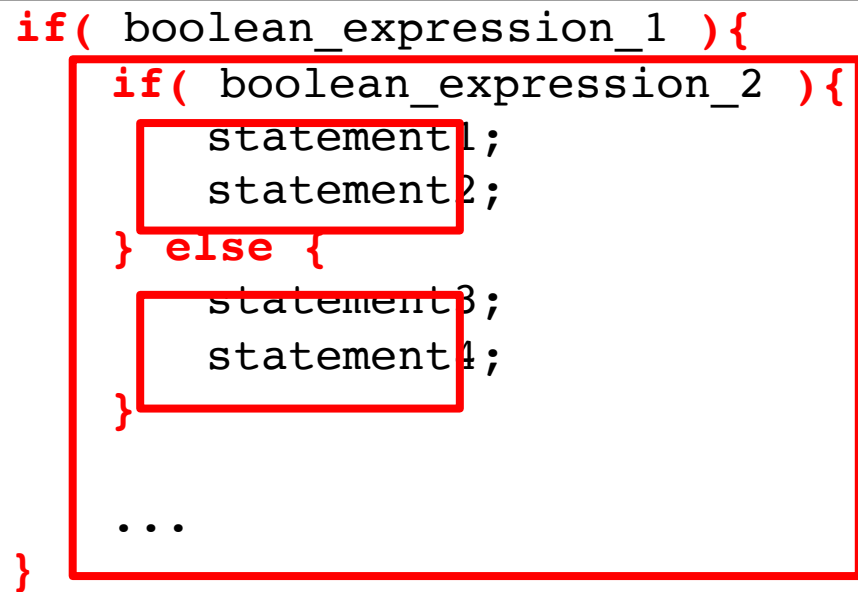
CSC111 – Jan 27th

Admin

- Jupyter Lab maintenance
 - Friday Jan 28th 8am - 10am
- Assignment submissions
 - Use filename exactly as specified.
 - **Assignment1.c (2)** is not acceptable
 - **assignment1.c** was specified
- UVIC emails – please use when contacting me.
 - <https://webmail.uvic.ca>

Review: A conditional statements can be inside a conditional statement

```
if( boolean_expression_1 ){  
    if( boolean_expression_2 ){  
        statement1;  
        statement2;  
    } else {  
        statement3;  
        statement4;  
    }  
    ...  
}
```

The diagram illustrates nested conditional statements. A large red box encloses the entire code block, from the opening 'if(' to the closing '}'. Inside this box, a smaller red box highlights the inner conditional statement, starting with 'if(boolean_expression_2 {' and ending with '}'. Within this inner box, two more red boxes highlight specific statements: one for 'statement1;' and 'statement2;', and another for 'statement3;' and 'statement4;'. This visual representation demonstrates how a conditional statement can be nested within another conditional statement.

Example

```
void commute(int rain_forecast, int wind_forecast){  
  
    if(rain_forecast > 10){  
        if(wind_forecast > 20){  
            printf("Take the bus\n");  
        } else {  
            printf("Ride your bike\n");  
        }  
  
        printf("Remember your raincoat\n");  
  
    }else{  
        printf("Ride your bike. It is nice!\n");  
    }  
}
```

Demo

Short circuit evaluation

Logical Operator	Example Boolean expression	Result of Boolean expression
	<code>x < y y < z</code>	<p>The <code>x<y</code> evaluates first -></p> <p>If it is True, the rest of the expression is not evaluated</p> <p>If it is False, the <code>y<z</code> is evaluated, then finally the <code> </code> expression is evaluated</p>
&&	<code>x < y && y < z</code>	<p>The <code>x<y</code> evaluates first -></p> <p>If it is False, the rest of the expression is not evaluated</p> <p>If it is True, the <code>y<z</code> is evaluated, then finally the <code>&&</code> expression is evaluated</p>

Demo

Function Return Values

Defining a function with a return value

General form:

Any valid C type:
int, double,
float...

```
type fn_name(type param_name);
```

Function prototype

Concrete example:

```
int add1(int num);
```

Must match

```
type fn_name(int param_name) {  
    C statement 1;  
    C statement 2;  
    return x;  
}
```

```
int add1(int num) {  
    int return_value;  
    return_value = num + 1;  
    return return_value;  
}
```

Function definition

types must match

Documentation expands...

```
/* Purpose: calculates num + 1
 * Parameters: int num – a number
 * Returns: int – the value of num + 1
 */
int add1 (int num) {
    int return_value;
    return_value = num + 1;
    return return_value;
}
```

- Every function you write SHOULD have a purpose comment
- If the function takes parameter(s), you MUST list them and the purpose must describe how the parameter(s) is(are) used.
- **If the function returns a value**, you must state the **type of the return** and a **description** of what is returned

```
#include <stdio.h>
```

```
int add1(int num);
```

← Function PROTOTYPE

```
int main ( ) {  
    int x = 12;  
    int result;  
    result = add1(x);  
    result = add1(11);  
    return 0;  
}
```

} Function CALLs
passing expected **argument**
storing the returned result

```
/* Purpose: calculates num + 1  
 * Parameters: int num – a number  
 * Returns: int – the value of num + 1  
 */
```

} Documentation

```
int add1 (int num) {  
    int return_value;  
    return_value = num + 1;  
    return return_value;  
}
```

} Function DEFINITION

Demo