DOR'S CODING SCHOOL

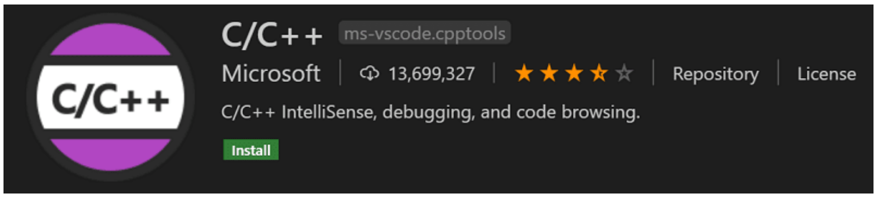
Week 1 Practice Problems

1. DEBUG

Each programming language has its own syntactical rules, which include the combination of both words and punctuation.

✏ **There are a few ways you can avoid syntax errors in the C programming language:**

1. Use a text editor or integrated development environment (IDE) that includes features such as syntax highlighting and automatic indentation, which can help you spot syntax errors more easily.

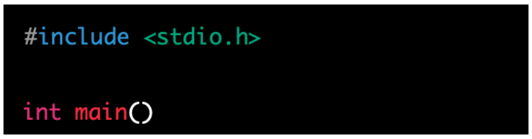


C/C++ IntelliSense, debugging, and code browsing.

2. Carefully read and re-read your code before attempting to compile it. Pay close attention to things like matching curly braces, proper use of punctuation, and spelling.
3. Test your code frequently as you write it. This will allow you to catch syntax errors earlier in the development process, when they are easier to fix.
4. Ask for help if you get stuck. Other programmers, or online resources such as forums or documentation, can often help you identify and fix syntax errors in your code.

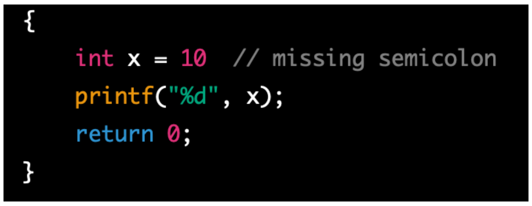
✏ **Examples of a syntactical error**

1. Missing a semicolon at the end of a statement:



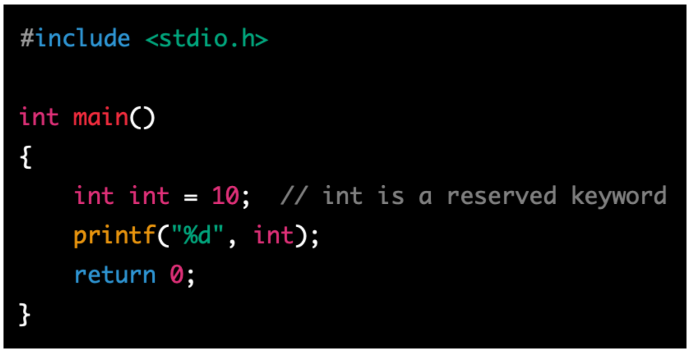
```
#include <stdio.h>

int main()
```



```c
{
    int x = 10  // missing semicolon
    printf("%d", x);
    return 0;
}
```

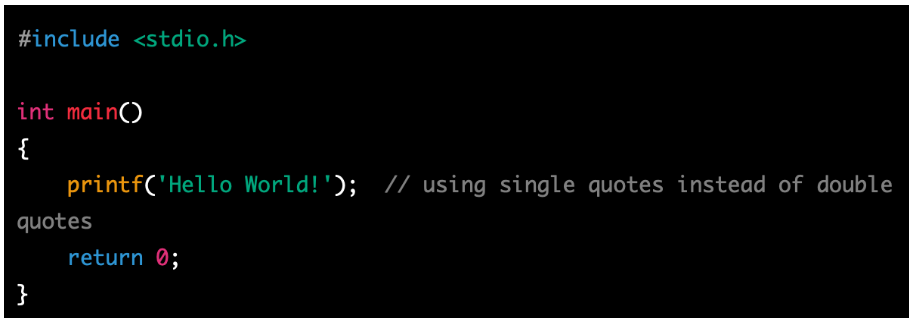
2. Using a reserved keyword as a variable name:



```c
#include <stdio.h>

int main()
{
    int int = 10;  // int is a reserved keyword
    printf("%d", int);
    return 0;
}
```

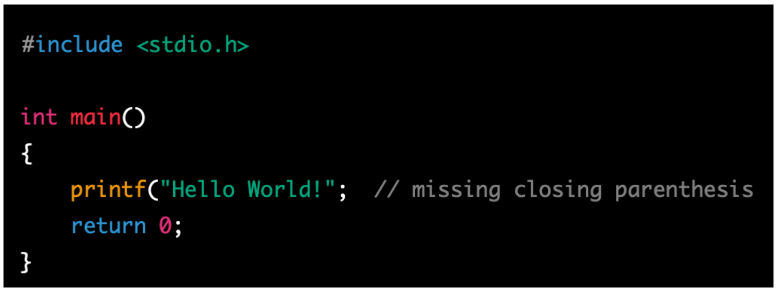
3. Using a single quote instead of a double quote to enclose a string:



```c
#include <stdio.h>

int main()
{
    printf('Hello World!');  // using single quotes instead of double
quotes
    return 0;
}
```

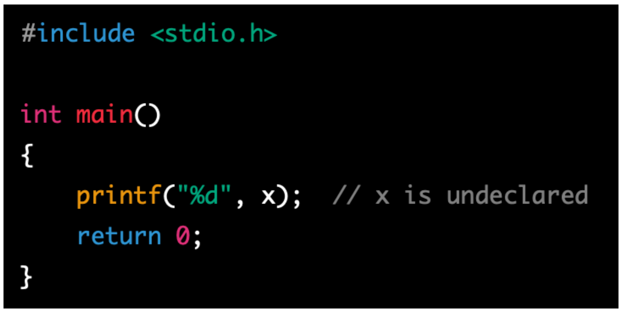
4. Omitting a closing parenthesis:



```c
#include <stdio.h>

int main()
{
    printf("Hello World!";  // missing closing parenthesis
    return 0;
}
```

5. Using an undeclared variable:



```c
#include <stdio.h>

int main()
{
    printf("%d", x);  // x is undeclared
    return 0;
}
```

2. HALF

Your function should use the input parameters, bill, tax, and tip, to calculate the final amount. However, since these values are percentages, you'll have to do some work to convert these to more appropriate formats to use for your calculation.

The final amount due, should add the tax to the bill before calculating the tip. Finally, you will return exactly half of the full amount, including the bill amount, the tax and the tip.

✏ Understanding the problem

The parameters 'bill' and 'tax' are float numbers, which means they have decimal places. The parameter 'tip' is an int number, which means it is an integer.

We need to:

1. Calculate tax amount
2. Calculate bill + tax amount
3. Calculate tip amount
4. Calculate total amount
5. Calculate and return half of the total amount

For example, let's suppose we have bill = 12.50, tax = 8.875 and tip = 20. What we would see in the steps above is:

1. Calculate tax amount

```
// Calculate tax amount
float taxAmount = 12.50 * (8.875 / 100);
```

2. Calculate bill + tax amount

```
// Calculate bill + tax amount
float billAmount = 12.50 + 1.11;
```

3. Calculate tip amount

```
// Calculate tip amount
float tipAmount = 13.61 * ((float)20 / 100);
```

4. Calculate total amount

```
// Calculate bill + tax amount
float billAmount = 12.50 + 1.11;
```

5. Calculate and return half of the total amount

```
// Calculate and return half of the total amount
return 16.33 / 2;

// Calculate and return half of the total amount
return 8.165;
```

3. PRIME

Prime numbers are defined as whole numbers greater than 1, whose only factors are 1 and itself.

So 3 is prime because its only factors are 1 and 3, while 4 is composite and not prime, because it is the product of 2 × 2.

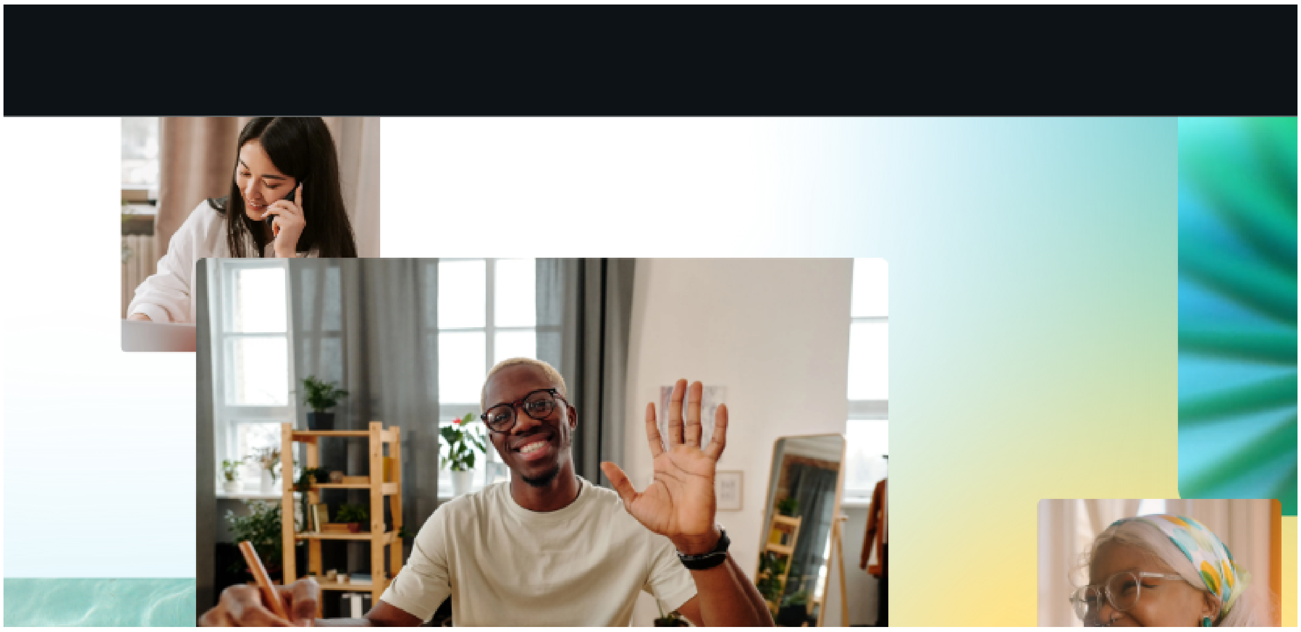
In this lab you will write an algorithm to generate all prime numbers in a range specified by the user.

✏ **Understanding the problem**

The prime() function takes an integer number as input and returns a boolean value indicating whether the given number is prime or not.

First, we check for two special cases where the number is less than or equal to 1 or 3. If the number is less than or equal to 1, it is not considered a prime number, so we return false. If the number is equal to 2 or 3, it is considered a prime number, so we return true.

Next, we have a for loop that iterates through the range from 2 to the square root of the number. For each iteration, we check if the number is divisible by the current loop variable (i) with no remainder. If the number is divisible, it is not a prime number, so we return false. If the for loop completes without finding any divisors, it means that the number is prime, so we return true.



Congratulations!

Now you're more than prepared to do the Lab 1 and then the Problem Set!

Good Luck