Function Overloading

```
char function(char x);
char function(char x, int y);
char function(double x);
int  function(int x);
```

Function overloading is the act of using the same name for a function but with different formal parameters. If you want to use a different return type, then keep in mind that functions that differ only in their return types cannot be overloaded. An example that would not work would be char function(char x) and int function(char x). The compiler would see this as ambiguous.

The formal parameters are what tells the compiler which function to use at the time of invocation. In the picture, there are four overloaded functions. If you had a function invocation with one char argument, it would execute the first function. If you had a function invocation with a double, it would execute the third function, and so forth. The second function has two arguments. It is okay that the first argument is the same as the first function, because the compiler sees that there is an additional argument being passed through executing the second function. The fourth function has a different return type and a different formal parameter. This is okay since the formal parameter is different than any other overloaded function. Here is an example that focuses on function invocation for brevity:

```
int main() {
    char x = ‘x’;
    double y = 9.0;
    int z = 5;
    char returnedFirstFunction = function(x);
    cout << “Character that is returned from 2nd function: ”
         << function(x, z) << endl;
    cout << “Character that is returned from 3rd function: ”
         << function(y) << endl;
    int returnedFourthFunction = function(z);
}
```

Each function has its own, unique task and would return its respective char or int. An example from CSI would be to draw shapes. Each shape has their unique characteristics, but could use the same named function, namely draw, to complete the task. Remember, this makes your program bigger with more code than may be necessary.

If you are able to use template functions, please do so for terse code. See the **Templates** page for further reading. Additionally, polymorphism may be preferred over function overloading. See the **Classes** section for further reading.