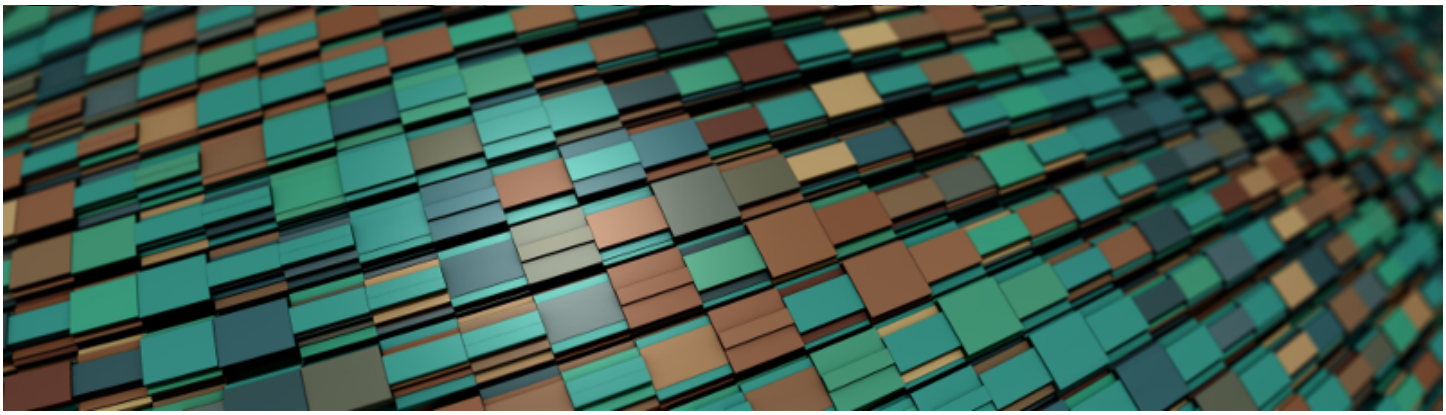
Lab 7. A Pretty-printing / Automatic Style Program



In this lab, we will develop a small utility program that can fix indentation in C or C++ source code files. It will have some limitations, but it will be able to cover a significant subset of valid C++ programs. Specifically, given a file with messed up indentation style:

```
                int main(){
            // Hi, I'm a program!
int x = 1;
    for(int i = 0; i < 10; i++) {
cout << i;
          cout << endl;
 }
    }
```

It will output a well-formatted program:

```
int main(){
    // Hi, I'm a program!
    int x = 1;
    for(int i = 0; i < 10; i++) {
        cout << i;
```

```
            cout << endl;
        }
    }
```

Task A. Removing indentation

Before we make a program that indents source code files, let's make a program that unindents them.

Start by writing a function

```
string removeLeadingSpaces(string line);
```

that takes one line of code as input and returns its copy *without leading spaces and tabs*:

```
removeLeadingSpaces("       int x = 1;  ") == "int x = 1;  "
```

Make use of the function `isspace` defined in `<cctype>` to check if a character is a whitespace. Your function should probably iterate over the input string, skip all spaces, and after it finds the first non-space character, start accumulating the characters into a new string, which will be returned.

Write a program `unindent.cpp` that reads input from `cin` and prints out each input line with leading spaces removed.

Example:

To test, create a badly indented file, for instance, called `bad-code.cpp` :

```
                int main(){
            // Hi, I'm a program!
int x = 1;
    for(int i = 0; i < 10; i++) {
cout << i;
          cout << endl;
 }
    }
```

Since our `unindent` program reads its input from `cin`, the badly indented text can be fed into it using standard input redirection:

```
$ ./unindent < bad-code.cpp
int main(){
// Hi, I'm a program!
int x = 1;
for(int i = 0; i < 10; i++) {
cout << i;
cout << endl;
}
}
```

Task B. Counting blocks opened and closed by `{` and `}`, and adding real indentation

To count open blocks, we need to count how many curly braces get opened and closed on each line, so we need to count `{` and `}`

Write a function

```
int countChar(string line, char c);
```

that scans the `line` and returns the number of occurrences of the character `c`.

Write a new program `indent.cpp` that enhances the program from the previous task. As it reads the input line by line, it should also count the number of open and closed `{` `}` in it, and keep track of **how many blocks is currently open at the beginning of each line**.

In the listing below, the number of open blocks is shown on the left:

```
0   int main(){
1   // Hi, I'm a program!
1   int x = 1;
```

```
1   for(int i = 0; i < 10; i++) {
2   cout << i;
2   cout << endl;
2   }
1   }
```

Then instead of printing the number of open blocks, add that number of tabs `'\t'` at the beginning of each line, and you will get:

```
int main(){
    // Hi, I'm a program!
    int x = 1;
    for(int i = 0; i < 10; i++) {
        cout << i;
        cout << endl;
        } // <-- closing for loop
    }     // <-- closing main
```

Notice that the closing curly braces are indented one level further than what they should be.

To fix that, when indenting the line, check its very first character. If it is a closing curly brace `}`, its indentation level should be reduced by one:

```
int main(){
    // Hi, I'm a program!
    int x = 1;
    for(int i = 0; i < 10; i++) {
        cout << i;
        cout << endl;
    }
}
```

Shortcomings of our program:

- No support for loops and if statements that don't have curly braces. For example,

```
if (c == 'A')
    s = s + c;
```

will be incorrectly indented as

```
if (c == 'A')
s = s + c;
```

- No support for `//` and `/* */`. Commented out curly braces should not affect indentation
- Symbols `{`, `}` inside `string` and `char` literals are misinterpreted as blocks

```
if (true) {
    s = "\{\{";
    t = "ABC";
}
```

will be incorrectly indented as

```
if (true) {
    s = "\{\{";
            t = "ABC";
        }
```

How to submit your programs

Each program should be submitted through Gradescope

Write separate programs for each part of the assignment.
Submit only the source code (.cpp) files, not the compiled executables.
Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: your name
Course: CSCI-136
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```