# Semester Project Part 0: A Tokenizer class

# Data Structures and Analysis of Algorithms, akk5

## Objectives

- To strengthen student's knowledge of C++ programming
- To give student experience reading and parsing strings of commands
- To give student experience in writing classes

## Instructions

Parsing text can be a frustrating part of any programming assignment. Parsing a line of text involves subdividing the line into individual units of text and using those units to determine the meaning of the line. Each unit of text is sometimes referred to as a token. For this assignment we will write a tokenizer class.

The purpose of this class is to subdivide a given line of text into individual tokens. The tokenizer class should be able to divide the string into words, integers, doubles, and lines of text. It should also be able to set the string to be tokenized, clear the string being tokenized, restore the string being tokenized, and determine if the string being tokenized is empty.

You will be using this class for the remaining programs. For this program, you need to design and write the Tokenizer class and a main program to test its features.

## Guidance

Tokening text can be a frustrating part of any programming assignment. Although C++ supports multiple approaches to handling this challenge, I suggest the following approach.

First, convert the string into a *stringstream*. The *stringstream* is probably new to most of you, but if you are comfortable working with streams it is easy enough to understand.

Stingstream is accessed by including sstream ( *#include <sstream>* ); since *stringstream* is in the std namespace, make certain you place a using clause in your code as well (using *std::stringstream*;).

You can convert a string to a *stringstream* as part of declaring the variable; as an example, the line of code below creates a *stringstream* labeled ss containing the contents of the string variable str:

```
stringstream ss(str);
```

At this point, any function, method, or operator that works with an *istream* will work with the *stringstream ss*. This means that >> and getline both work with a stringstream. We will use both to retrieve the requested tokens from the stringstream.

When using the >> operation to tokenize the strings we will need to set the *ios::failbit* to detect when >> fails because of type mismatch or an unexpected end of line. To do this, use the following line of code:

```
ss.exceptions(ios::failbit);
```

Enabling the *ios::failbit* causes >> to throw an exception when it fails to extract information from the *istream* in question. Using this method will require you to place your entire input processing code block into a *try {} catch() {}* block. You will want to consider a code structure like

```
try
{
        ss >> x;
        if (x == “y”) {}
} catch (ios_base::failure)
{
        ss.clear();
}
```

## Grading Breakdown

| **Point Breakdown** | |
|---|---|
| ***Structure*** | 12 pts |
| The program has a header comment with the required information. | 3 pts |
| The overall readability of the program. | 3 pts |
| Program uses separate files for main and class definitions | 3 pts |
| Program includes meaningful comments | 3 pts |
| ***Syntax*** | 16 pts |
| Implements Class Tokenizer correctly | |
| ***Behavior*** | 72 pts |
| Program tests each of the required features of the class | |
| • Creating a default instance of the class | 8 pts |
| • Creating an instance of the class using a string | 8 pts |
| • Clearing the string the class is using | 8 pts |
| • Setting the string the class is using | 8 pts |
| • Reading a word from the string | 8 pts |
| • Reading an integer from the string | 8 pts |
| • Reading a double from the string | 8 pts |
| • Reading the remaining string | 8 pts |
| • Resetting the string to its original value | 8 pts |
| ***Total Possible Points*** | **100pts** |
| | |
| Penalties | |
| Program does NOT compile | -100 |
| Late up to 72 hrs | -10 per day |
| Late more than 72 hrs | -100 |

## Header Comment

At the top of each program, type in the following comment:

/*

Student Name: <student name>

Student NetID: <student NetID>

Compiler Used: <Visual Studio, GCC, etc.>

Program Description:

<Write a short description of the program.>

*/

Example:

/*

Student Name: John Smith

Student NetID: jjjs123

Compiler Used: Eclipse using MinGW

Program Description:

This program prints lots and lots of strings!!

*/

## Assignment Information

Due Date: 8/29/2021

Files Expected:

1. Main.cpp – File containing function main
2. Tokenizer.h - File containing the Tokenizer class definitions.
3. Tokenizer.cpp - File containing the code for the Tokenizer methods.