Introduction

Project Bombo is a C++ Terminal-based Development tool which assists with creating consistent Code which is particularly good for large scale development as it creates a consistent format that code follows:

Format

The format followed is standardised to make it easier for Bombo to read through code and remember certain things with the code, but also to make Code more efficient to read as it uses a consistent language across the source code.

Firstly, each project starts with three basic files, the main.cpp being the initial entry point of the program, a util.h a header file to store all standard library includes, namespace usage and generic functionality that can be helpful anywhere and finally a util.cpp: which contains the functionality of functions defined in the util.h header file. A firm rule with the util files is that variables, classes, structs cannot be defined here rather only return functions can be defined. The files furthermore cannot include any header files of the source code and this is to avoid circular dependency as util.h is the only file that must be included everywhere within the source code. Another important pillar of the format is .cpp files, main.cpp is the only .cpp file that can include other files as every other .cpp file shall only include its parent header file. Here's an example of the format expected:

- Example of main.cpp:

```cpp
#include "util.h"

int main()
{

   return 0;
}
```

- Example of util.h:

```cpp
#ifndef UTIL_H
#define UTIL_H

#include <iostream>

using namespace std;

// functions

double getRandomDouble(double min, double max);

#endif
```

- Example of util.cpp:

```
#include "util.h"

// functions

double getRandomDouble(double min, double max)
{
   // functionality
}
```

- Example of a user-created .h file:

```
#ifndef SOMETHING_H
#define SOMETHING_H

#include "util.h"

// structs

// classes

// variables

// functions

#endif
```

- And a complimentary .cpp file:

```
#include "something.h"

// variables

// functions
```

Functionality:

The functionality will work around a personified bot, Bombo stores information about your project in a .PROJECT plain-text file. Bombo navigates the files by jumping to the necessary markers seen as // classes, // structs, // variables, // functions. Bombo adds specific functionality specified by jumping to the necessary marker in a specified file, moving down to spaces and then placing the function. Comments above functions can also optionally be added but Bombo ensures everything has exactly 1 space in between each added object. Bombo also has some basic error detection and error prevention measures as Bombo will be able to detect composition and inheritance errors e.g. being defined before a parent class or a composed object. Bombo has no knowledge of user implemented functionality and simply ignores it when correction errors, user errors are up to the user to fix on their own.
Classes also follow a strict guideline, every variable created in a class by default is private but provided with it is a get function which returns the value of that variable in the class e.g.

string getName();. Functions also follow a specific naming convention such that the first word is lowercase with every other following word having a capitalised first letter. Users don't have to follow this naming convention but this is what Bombo follows.

Compiling:

Bombo generates a makefile for the project which it maintains as other files and imports are added. This makefile however can only be run by the user manually by running ‘make’ in a terminal. Bombo should also include the ability for ‘make clean’ to be run to clean out the objects and executable. Source code is placed in a src folder, objects are to be placed in a obj folder and the executable is to be placed in the same src folder as the source code(Bombo is located outside and upon running creates these folders at the start of a new project).