Introduction

Project Bombo is a C++ Terminal-Based Development Tool which assists with creating consistent Code formatting and facilitating Cohesion among teams particularly for large scale development.

Format

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

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:

Examples

- 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

int getRandomInt(int min, int max);

#endif
```

- Example of util.cpp:

```
#include "util.h"

// functions

int getRandomInt(int min, int max)
{
  // TODO: Implement function
}
```

- Example of a user created .h file:

```
#ifndef NEW_FILE_H
#define NEW_FILE_H

#include "util.h"

// structs

struct new_struct
{
  int new_struct_parameter;
};

// classes

class new_class
{
  private:
    int new_class_parameter;
    void private_new_class_function();
  protected:
  public:
    int getnew_class_parameter();
    void new_class_function();
};

// variables

extern int new_variable;

// functions

int new_function(string name);

#endif
```

As seen above, Bombo uses markers (// functions, // variables, // classes, // structs) to organise and know where to place the objects in the code. These markers help with organisation and ensuring consistency. All parameters in classes are by default private and can be accessed by a function created by Bombo to access the variable.
Below is the .cpp file created alongside this file

- Example of a user created .cpp file:

```cpp
#include "new_file.h"

// variables

int new_variable;

// functions

int new_class::getnew_class_parameter()
{
  return new_class_parameter;
}
void new_class::new_class_function()
{
// TODO: Implement function
}
void new_class::private_new_class_function()
{
// TODO: Implement function
}



int new_function(string name)
{
  // TODO: Implement function
}
```

Bombo wont implement functionality for you but will assist with the blueprint of the C++ code.

Here's the list of commands used to get to this stage to showcase how Bombo functions:

```
% ./Bombo
Bombo > createProject()
Enter a project name: new_project
directory /new_project created.
directory /new_project/src created.
directory /new_project/obj created.
file /new_project/src/main.cpp created.
```

```
file /new_project/src/util.h created.
file /new_project/src/util.cpp created.
file /new_project/Makefile created.
file /new_project/Bombo.log created.
file /new_project/.PROJECT created.
Bombo > createFunction()
Enter a filename: util
Enter a return type: int
Enter a function name: getRandomInt
Enter parameter type (or type 'done' to finish): int
Enter parameter name: min
Enter parameter type (or type 'done' to finish): int
Enter parameter name: max
Enter parameter type (or type 'done' to finish): done
Function getRandomInt created successfully.
Bombo > createFile()
Enter a filename: new_file
file /new_project/src/new_file.h created.
file /new_project/src/new_file.cpp created.
Bombo > createFunction()
Enter a filename: new_file
Enter a return type: int
Enter a function name: new_function
Enter parameter type (or type 'done' to finish): string
Enter parameter name: name
Enter parameter type (or type 'done' to finish): done
Function new_function created successfully.
Bombo > createFunction()
Enter a filename: util
Enter a return type: double
Enter a function name: getRandomDouble
Enter parameter type (or type 'done' to finish): double
Enter parameter name: min
Enter parameter type (or type 'done' to finish): double
Enter parameter name: max
Enter parameter type (or type 'done' to finish): done
Function getRandomDouble created successfully.
Bombo > createVariable()
Enter a filename: new_file
Enter a variable type: int
Enter a variable name: new_variable
Variable new_variable created successfully.
Bombo > createStruct()
Enter a filename: new_file
Enter a struct name: new_struct
Enter parameter type (or type 'done' to finish): int
Enter parameter name: new_struct_parameter
Enter parameter type (or type 'done' to finish): done
Struct new_struct created successfully.
Bombo > createClass()
Enter a filename: new_file
Enter a class name: new_class
Enter parameter type (or type 'done' to finish): int
Enter parameter name: new_class_parameter
Enter parameter type (or type 'done' to finish): done
Enter a function name(or type 'done' to finish): new_class_function
Enter a return type: void
Enter for (public, private or protected): public
Does this function need variables?(yes or no): no
Enter a function name(or type 'done' to finish): private_new_class_function
Enter a return type: void
Enter for (public, private or protected): private
Does this function need variables?(yes or no): no
Enter a function name(or type 'done' to finish): done
```

```
Class new_class created successfully.
Bombo >
```

- Example of Bombo.log:

```
[2025-01-22 13:55:57]Project new_project created.
[2025-01-22 13:58:28]Function getRandomInt created in util.
[2025-01-22 14:00:49]new_file file created.
[2025-01-22 14:05:11]Project loaded.
[2025-01-22 14:05:48]Function new_function created in new_file.
[2025-01-22 14:06:08]Function getRandomDouble created in util.
[2025-01-22 14:08:40]Project loaded.
[2025-01-22 14:09:13]Variable new_variable created in new_file.
[2025-01-22 14:09:46]Struct new_struct created in new_file.
[2025-01-22 14:10:50]Class new_class created in new_file.
[2025-01-22 14:42:28]Project loaded.
[2025-01-22 14:42:43]User User1 created.
[2025-01-22 14:42:57]Project loaded by User1.
```

Every Change is Logged in a plain-text file called Bombo.log, and when users are enabled, each action states which user did which action like this example:

```
[2025-01-22 14:42:57]Project loaded by User1.
[2025-01-22 14:49:42]new_file2 file created by User1.
[2025-01-22 14:50:08]Function newFunction2 created in new_file2 by User1.
```

Functionality

(explanation from the old documentation with some removed segments)
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. 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();.

- Example of the .PROJECT plain-text metadata file

```
ProjectName: new_project
SourceDir: src/
ObjectDir: obj/
MainFile: src/main.cpp
UtilHeader: src/util.h
UtilSource: src/util.cpp
OtherHeader: src/new_file.h
OtherSource: src/new_file.cpp
MakeFile: Makefile
Functions: getRandomInt new_function getRandomDouble
Classes: new_class
Included Libraries:
UsersEnabled: false
Users:
BackupsEnabled: false
BackupDestination:
```

As you can see above, The metadata file stores all of the important information including some more recent additions like User support, support for Backups and Libraries.

User Functionality

Users can first be created by Enabling Users on a Pre-existing project or Logging into a user on a project which already has Users enabled, heres an example starting from a project with users disabled:

```
 % ./Bombo
Bombo > loadProject()
Enter a project name: new_project
Meta data found.
Project loaded.
Hi whoever you are :|
Bombo > enableUsers()
Users successfully enabled.
Encrypted: new_project/src/main.cpp
Encrypted: new_project/src/util.h
Encrypted: new_project/src/util.cpp
Encrypted: new_project/src/new_file.h
Encrypted: new_project/src/new_file.cpp
Enter a user name: User1
Key: Hq!7Gu$0Tq#5Je*9
file /new_project/.tempUser1Key created.
Enter a project name: new_project
Meta data found.
Enter your user name: User1
Enter your Key: Hq!7Gu$0Tq#5Je*9
Decrypted: new_project/src/main.cpp
Decrypted: new_project/src/util.h
Decrypted: new_project/src/util.cpp
Decrypted: new_project/src/new_file.h
Decrypted: new_project/src/new_file.cpp
Project loaded.
こんにちは User1 :>
Bombo >
```

For security purposes you are initially loaded out of the project once credentials is given, when no one is logged in, each file is still stored in plain text but is encrypted to prevent unauthorised access. Heres what is seen by someone attempting to look at the files without Logging in:

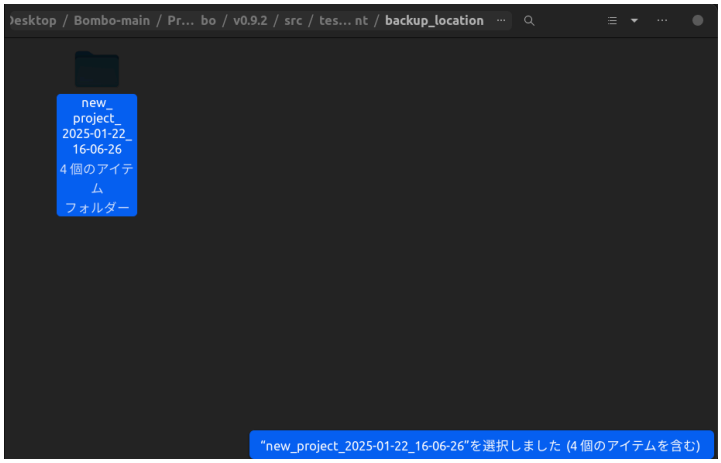
Or in some text editors:

Backup Functionality

We have also implemented backups that works by first enabling backups and giving Bombo a file address to send the backups at:

```
Bombo > enableBackups()
Backups successfully enabled.
Enter a Destination Directory(Absolute Path):
/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/backup_location
Destination Directory Successfully set.
Bombo > createBackup()
Backup created successfully:
"/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/backup_location/new_project_2025-01-22_16-06-26"
Bombo >
```

As you can see, the backup is dated and sends the backup to the specified path and will look something like this in the path:



Compile Functionality

Bombo is able to use and maintain the Makefiles and so is able to compile and clean the project as you see fit by calling the appropriate function:

- Example of Compilation:

```
Bombo > compile()
make: ディレクトリ '/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/new_project' に入ります
g++ -c src/main.cpp -o obj/main.o
g++ -c src/util.cpp -o obj/util.o
src/util.cpp: In function ‘double getRandomDouble(double, double)’:
src/util.cpp:8:1: warning: no return statement in function returning non-void [-Wreturn-type]
   8 | }
     | ^
src/util.cpp: In function ‘int getRandomInt(int, int)’:
src/util.cpp:14:1: warning: no return statement in function returning non-void [-Wreturn-type]
  14 | }
     | ^
g++ -c src/new_file.cpp -o obj/new_file.o
src/new_file.cpp: In function ‘int new_function(std::string)’:
src/new_file.cpp:26:1: warning: no return statement in function returning non-void [-Wreturn-type]
  26 | }
     | ^
g++ -c src/new_file2.cpp -o obj/new_file2.o
src/new_file2.cpp: In function ‘int newFunction2(int)’:
src/new_file2.cpp:10:1: warning: no return statement in function returning non-void [-Wreturn-type]
  10 | }
     | ^
g++ -o new_project.exe obj/main.o obj/util.o obj/new_file.o obj/new_file2.o
make: ディレクトリ '/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/new_project' から出ます

Bombo >
```

- Example of Cleaning:

```
Bombo > clean()
make: ディレクトリ '/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/new_project' に入ります
rm -f new_project.exe obj/*.o
make: ディレクトリ '/home/surai/Desktop/Bombo-main/Project_Bombo/v0.9.2/src/test_enviroment/new_project' から出ます

Bombo >
```

It is able to display all errors and warnings provided by the Compilation client

Current Progress and Future (from 22nd January 2025)

This is all the current user callable functions and their level of implementation:
Appear to be fully functional:

```
// ==== Generic Operations ====
void createProject();
void compile();
void clean();
void createFile();
void createFunction();
void createClass();
void createVariable();
void createStruct();
void addLibrary();
void loadProject();
```

```
// ==== User Related Operations ====
void enableUsers();
void createUser();
```

```
// ==== Bombo Personality Functions ====
void greeting();
void joke();
```

```
// ==== Backup Related Operations ====
void enableBackups();
void changeDestinationDirectory();
void createBackup();
bool BackupsEnabled();
```

Incomplete Functions:

```
//  ==== Delete and Remove Operations ====
void deleteFile(); // TODO
void deleteFunction(); // TODO
void deleteClass(); // TODO
void deleteStruct(); // TODO
void deleteUser(); // TODO
void removeLibrary(); // TODO
```

And lastly Future additions:

```
// enablePermissions() (adds an Admin account which has certain controls which normal users do not possess)
// mergeProjects() (able to grab multiple versions of a project and merge them into one)
// logComment() (adds a comment into the log file)
```