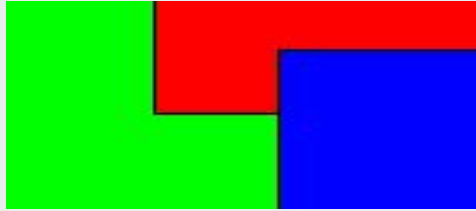


Sprint #1



3Blockz



3Blockz Development
Team F

Cody Crawford, Tyler Loewen, Troy Paul
February 28, 2018



Project Introduction

The 3Blockz Development team set out to create a series of code generation tools to be implemented as a plug in for Code Blocks.

The tools set out to be designed were:

- Empty Hash Generator
- Equality Overload Function Generator
- Getter/Setter Member Function Generators
- Class Builder Generator (using the builder pattern)



Software Description

Empty Hash Generator:

- Generates a hash for the specified object of the defined class type.
- For this sprint the function was planned to return zero.
- Future iterations will expand this tool to generate an appropriate hash based on object's member values.



Software Description

```
struct MyHash
{
    std::size_t operator()(S const& s) const noexcept
    {
        std::size_t h1 = std::hash<std::string>{}(s.first_name);
        std::size_t h2 = std::hash<std::string>{}(s.last_name);
        return h1 ^ (h2 << 1); // or use boost::hash_combine (see Discussion)
    }
};
```



Software Description

Equality Operator Overload Generator:

- The team set out to make an equality function generator
- The plugin would take in the inputs for a specified object and generator an overloaded equality operator for that object



Software Description

```
bool operator== (const Car &c1, const Car &c2)
{
    return (c1.m_make== c2.m_make &&
            c1.m_model== c2.m_model);
}
```



Software Description

Getter/Setter:

- This aspect of the plugin was to generate member functions for the specified object.
- The user would simply input the object desired and the necessary variable names needed and the code would generate the expected member variables



Software Description

Getter:

```
public:
    string getAddress()
    {
        return address;
    }

    long int getInsurancelumber()
    {
        return insurancelumber;
    }
```




Software Description

Setter:

```
public:
    void setInsuranceNumber(unsigned long int insurance)
    {
        if (insurance >= 100000000 && insurance <= 999999999)
            insuranceNumber = insurance; // a correct value is set
        else
            // display only error message and do not set incorrect value
            cout << "Incorrect Insurance number" << endl;
    }
```



Software Description

Builder:

- This tool generates a code for a new builder class
- Builder pattern allows the user to build new instances of objects easier than using a polluted constructor.
- Will be able to call individual functions to set different parameters which either may be optional or required once the user is done they may call the build function which will validate all of the parameters and build the object instance.
- This will use call cascading: where every setter member function will return a reference to itself

Software Description



Builder:

```
class Product{
public:
    // use this class to construct Product
    class Builder;

private:
    // variables in need of initialization to make valid object
    const int i;
    const float f;
    const char c;

    // Only one simple constructor - rest is handled by Builder
    Product( const int i, const float f, const char c ) : i(i), f(f), c(c){}

public:
    // Product specific functionality
    void print();
    void doSomething();
    void doSomethingElse();
};
```

```
int main(){
    // simple usage
    Product p1 = Product::Builder().setI(2).setF(0.5f).setC('x').build();
    p1.print(); // test p1

    // advanced usage
    Product::Builder b;
    b.setProductP();
    Product p2 = b.build(); // get Product P object
    b.setC('!'); // customize Product P
    Product p3 = b.build();
    p2.print(); // test p2
    p3.print(); // test p3
}
```



Sprint #1 Plan







The first sprint was supposed to be setting up the backbone of the plugin we had certain goals we wanted to achieve:

- ❑ Create a functioning menu button on the CodeBlocks system
- ❑ Create the UI to be called
- ❑ Have that menu connect to UI for each button
- ❑ Have the UI call the appropriate areas of the generator
- ❑ Create the Hash/Equality generator
- ❑ Connect the generator to the UI



Work Completed

Goals:

-  Create a functioning menu button on the CodeBlocks system
-  Create the UI to be called
-  Have that menu connect to UI for each button
-  Have the UI call the appropriate areas of the generator
-  Create the Hash/Equality generator
-  Connect the generator to the UI



3 Things Done Well

1. CI works exceptionally using all the files desired
2. Learned a lot of new patterns and syntax that we were unfamiliar with before examples: Menu system, wxSmith,
3. Learned all the 101 ways not to build a plugin



3 Things to Improve

1. Scheduling and prioritizing work to be completed.
2. More communication and collaboration between teammates to ensure more productivity.
3. Adding flexibility to some code used in the learning process



But How?

The development team has addressed these issues and come up with solutions to them:

1. GitLab issues board will be used more to schedule tasks better
2. The team will meet at more consistent times to work together on the project. Slack will also be used to ensure teammates can communicate with each other.
3. The code will be refactored to be more flexible to make future iterations easier to work with.



Bibliography

- <http://en.cppreference.com/w/cpp/utility/hash>
- <http://www.learncpp.com/cpp-tutorial/96-overloading-the-comparison-operators/>
- <https://www.tutorialcup.com/cplusplus/getters-setters.htm>
- https://en.wikipedia.org/wiki/Builder_pattern