|  |
| --- |
| creeper1.jpg |
| Assignment 4 |
| CPTN230 |
|  |
| **Paul Fahey** |
| **10/16/2011** |



Contents

[Introduction 3](#_Toc307050845)

[Application Summary 3](#_Toc307050846)

[Object and Variable Functionality 4](#_Toc307050847)

[Pseudo Code 4](#_Toc307050848)

[Assignment 4 Header File 4](#_Toc307050849)

[Assignment 4 Class File 5](#_Toc307050850)

[Assignment 4 Application File 6](#_Toc307050851)

[Conclusions 7](#_Toc307050852)

[Captured Screen Output 7](#_Toc307050853)

[References 10](#_Toc307050854)

# Introduction

The goal of this assignment is to create a class and demonstrate operator overloading with that class. The application will create multiple instances of the class chosen and demonstrate the use of overloaded C++ operators with built in data types of the language. There will also be demonstration of an overloaded constructor as well. A variety of tests will be performed to test the overloaded operators to make sure they are working as intended.

# Application Summary

This program will demonstrate the functionality of a creeper object which came from the sandbox building game minecraft. The user will be greeted to the program, and there will be a target, explosion, awareness, and direction set for a bunch of creeper instances displayed on the screen. Following this a message will be displayed saying if a test passed or not for the overloaded operators used comparing sets of creeper instances with each other.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Object** | **Target** | **Explosion** | **Awareness** | **Direction** |
| Creeper1 | 1 | 50 | 100 | 180 |
| Creeper2 | 2 | 100 | 20 | 90 |
| Creeper3 | 3 | 65 | 75 | 120 |
| Creeper4 | 1 | 50 | 100 | 180 |
| Creeper5 | 3 | 65 | 75 | 120 |
| Creeper6 | 2 | 100 | 20 | 90 |
| Creeper7 | 3 | 65 | 75 | 120 |
| Creeper8 | 1 | 100 | 20 | 90 |
| Creeper9 | 2 | 50 | 20 | 90 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Status** | **Result** | **Comment** |
| Creeper1==Creeper4 | Pass | True | All fields == |
| Creeper2==Creeper8 | Fail | False | Target != Target |
| Creeper3==Creeper1 | Fail | False | All fields != |
| Creeper6!=Creeper7 | Pass | True | All fields != |
| Creeper9!=Creeper2 | Pass | True | Explosion != |
| Creeper3!=Creeper7 | Fail | False | All fields == |
| Creeper4>Creeper1 | Fail | False | All fields == |
| Creeper7>Creeper8 | Fail | False | Explosion<Explosion |
| Creeper3>Creeper1 | Pass | True | Explosion&Awareness> |
| Creeper5>Creeper2 | Fail | False | Explosion&Awareness< |
| Creeper7>Creeper4 | Pass | True | Awareness>Awareness |

# Object and Variable Functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Members of the Creeper Object** | | | | |
| **Members:** | target | explosion | awareness | direction |
| **Data Type:** | integer | integer | integer | integer |
| **Initial Value:** | 1 | 50 | 100 | 180 |
| **Purpose:** | To give a target for the creeper | To give an explosion for the creeper | To give an awareness for the creeper | To give a direction for the creeper |

# Pseudo Code

## Assignment 4 Header File

* A class called creeper is created

**Data Members (Private):**

* A *target* with type integer
* An *explosion* with type integer
* An *awareness* with type integer
* A *direction* with type integer

**Member Functions (Public):**

* A default **creeper** constructor that will initialize all the data members
* A overloaded **creeper** constructor that will initialize all the data members and take in one integer value
* A overloaded **creeper** constructor that will initialize all the data members and take in two integer values
* A overloaded **creeper** constructor that will initialize all the data members and take in three integer values
* A overloaded **creeper** constructor that will initialize all the data members and take in four integer values
* An **operator** function that will overload the *==* operator using the address of a creeper object
* An **operator** function that will overload the *!=* operator using the address of a creeper object
* An **operator** function that will overload the *>* operator using the address of a creeper object
* A **set** function to set the *target* that will not return any value but take in a string value
* A **get** function to get the *target* that will return an integer value but not take in any value
* A **set** function to set the *explosion* that will not return any value but take in a string value
* A **get** function to get the *explosion* that will return an integer value but not take in any value
* A **set** function to set the *awareness* that will not return any value but take in a string value
* A **get** function to get the *awareness* that will return an integer value but not take in any value
* A **set** function to set the *direction* that will not return any value but take in a string value
* A **get** function to get the *direction* that will return an integer value but not take in any value

## Assignment 4 Class File

* Includes the creeper class header file

Default **Creeper** constructor initializes the data members:

* *Target* is set with an integer value of 1
* *Explosion* is set with an integer value of 50
* *Awareness* is set with an integer value of 100
* *direction* initialized with an integer value of 180

Overloaded **Creeper** constructor initializes the data members with a target parameter:

* *Target* is set with an integer value that came from the application file
* *Explosion* is set with an integer value of 100
* *Awareness* is set with an integer value of 20
* *direction* initialized with an integer value of 90

Overloaded **Creeper** constructor initializes the data members with a target and explosion parameter:

* *Target* is set with an integer value that came from the application file
* *Explosion* is set with an integer value that came from the application file
* *Awareness* is set with an integer value of 75
* *direction* initialized with an integer value of 120

Overloaded **Creeper** constructor initializes the data members with a target, explosion and awareness parameter:

* *Target* is set with an integer value that came from the application file
* *Explosion* is set with an integer value that came from the application file
* *Awareness* is set with an integer value that came from the application file
* *direction* initialized with an integer value of 180

Overloaded **Creeper** constructor initializes the data members with a target, explosion, awareness, an direction parameter:

* *Target* is set with an integer value that came from the application file
* *Explosion* is set with an integer value that came from the application file
* *Awareness* is set with an integer value that came from the application file
* *direction* initialized with an integer value that came from the application file
* An **operator** function with a return type of boolean that will overload the *==* operator using the address of a creeper object
* Conditional statements that will check to see if the target, explosion, awareness, and direction are all the same for both objects
* An **operator** function with a return type of boolean that will overload the *!=* operator using the address of a creeper object
* Conditional statements that will check to see if either target, explosion, awareness, and direction are not or all not the same for the objects being compared
* An **operator** function with a return type of boolean that will overload the *>* operator using the address of a creeper object
* Conditional statements that will check to see if the explosion and awareness is greater or one or the other is greater with the target and explosion being non effective in this statement for the objects being compared
* The **set target** function is initialized that takes in a *input* target as a parameter
* The *input* target is given as the value for data member *target pointer*
* The **get** **target** function is initialized that takes in no parameters
* It returns the *target pointer* data member to the application file
* The **set explosion** function is initialized that takes in a *input* explosion as a parameter
* The *input* explosion is given as the value for data member *explosion*
* The **get** **explosion** function is initialized that takes in no parameters
* It returns the *explosion* data member to the application file
* The **set** **awareness** function is initialized that takes in a *input* awareness as a parameter
* The *input* awareness is given as the value for data member *awareness*
* The **get** **awareness** function is initialized that takes in no parameters
* It returns the *awareness* data member to the application file
* The **set** **direction** function is initialized that takes in a *input* direction as a parameter
* The *input* direction is given as the value for data member *direction*
* The **get** **direction** function is initialized that takes in no parameters
* It returns the *direction* data member to the application file

## Assignment 4 Application File

* Includes the creeper class header file
* A prototype function is created that will display the contents of the creeper object using the address of the creeper instance
* The main function begins
* A message that welcomes the user to the program
* Nine creeper objects are made to test the constructors and the results are displayed to the screen
* Tests described in the application summary in a table are executed for the nine objects created with an output of pass or fail to the screen
* A message is displayed that thanks the user for using the program
* The main function has ended
* The **display creeper** function is defined that takes in a **the address of a creeper object**as a parameter of type **creeper**
* **get target** is called and displays the target on the screen
* **get explosion** is called and displays the explosion on the screen
* **get awareness** is called and displays the awareness on the screen
* **get direction** is called and displays the direction on the screen

# Conclusions

This assignment was about demonstrating overloading of the C++ operators. What I thought went well was my original design plan and chosen operators for this assignment. I was able to overload the operators for the most part that I chose and was able to reuse my design in the previous assignments to save time. The issues I encountered were setting up the statements in the overloaded operator functions. It wasn’t giving me the results I wanted for some things and I needed to delete some stuff to make it work right that took some time to fix. As far as things that didn’t work I was able to fix those mistakes so I didn’t have anything that was unresolved. I took me a little while to wrap my head around the concept of operator overloading which stalled the design somewhat, but I was able to figure out the concepts that gave me confusion. There were no features that I had that could have been implemented better then they currently are to make the program better than it was. If there was a second version of this program the only thing I can think of to add to it would just be more data members to implement the object better and other overloaded operators to accompany them.

# Captured Screen Output

Welcome to assignment 4

creeper1:

Target: 1 was chosen

Explosion: 50 feet

Awareness: 100 foot radius

Direction: 180 degrees

creeper2:

Target: 2 was chosen

Explosion: 100 feet

Awareness: 20 foot radius

Direction: 90 degrees

creeper3:

Target: 3 was chosen

Explosion: 65 feet

Awareness: 75 foot radius

Direction: 120 degrees

creeper4:

Target: 1 was chosen

Explosion: 50 feet

Awareness: 100 foot radius

Direction: 180 degrees

creeper5:

Target: 3 was chosen

Explosion: 65 feet

Awareness: 75 foot radius

Direction: 120 degrees

creeper6:

Target: 2 was chosen

Explosion: 100 feet

Awareness: 75 foot radius

Direction: 120 degrees

creeper7:

Target: 3 was chosen

Explosion: 65 feet

Awareness: 75 foot radius

Direction: 120 degrees

creeper8:

Target: 1 was chosen

Explosion: 100 feet

Awareness: 20 foot radius

Direction: 90 degrees

creeper9:

Target: 2 was chosen

Explosion: 50 feet

Awareness: 20 foot radius

Direction: 90 degrees

Test 1 Passed

Test 2 Failed

Test 3 Failed

Test 4 Passed

Test 5 Passed

Test 6 Failed

Test 7 Failed

Test 8 Failed

Test 9 Passed

Test 10 Failed

Test 11 Passed

Thank you for using assignment 4

Press any key to continue . . .

# References

Bettle, H. (n.d.). Assignment 4 for CPTN230 Assignment Description. 6.

Bettle, H. (n.d.). Assignment 4 Sample Files. 3.

Deitel, P., & Deitel, H. (2010). *C++ How To Program.* Upper Saddle River: Pearson Education Inc.

Fahey, P. (2011). Assignment 2 Design Document. 8.