1. **Program Statement**

This program uses copy constructors in two classes when one class is inherited to be able to test instance of a class. This program also includes the use of templates for any data types passed in in main. At the end, the program outputs all the tests information to an output file.

1. **Requirements**
   1. **Assumptions**
      1. User understands how the code works.
   2. **Specifications**
      1. Copy constructors in each class
      2. Make a template data type to be used for data to be stored in classes
      3. Getters and setters in each class
      4. 2 classes
         1. Class A
            1. Copy constructor
            2. Private variable valuea
            3. Public functions getValues and setValues
         2. Class B, which inherits class A
            1. Copy constructor
            2. Private variable valueb
            3. Public function getValues and setValues
      5. Test an instance of class B in main with data types int, float, char, string, and also struct separately.
      6. Struct Data that holds day, month, and year of data type int
      7. Print all the results to screen and to an output file called *out.txt*
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Two integer data types for the instance testing | Call the functions and constructors of the instance | Results to screen and output file |
| Two float data types for the instance testing | Call the functions and constructors of the instance | Results to screen and output file |
| Two char data types for the instance testing | Call the functions and constructors of the instance | Results to screen and output file |
| Two string data types for the instance testing | Call the functions and constructors of the instance | Results to screen and output file |

1. **Test Strategy**
   1. **Valid Data**
   2. **Invalid Data**
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Pass in values of int data type |  |  |  |  |
| Valid | 2 | Pass in values of float data types |  |  |  |  |
| Valid | 3 | Pass in values of char data types |  |  |  |  |
| Valid | 4 | Pass in values of string data types |  |  |  |  |
| Valid | 5 | Pass in values of struct data types |  |  |  |  |

1. **Initial Algorithm**
   1. In main
      1. Include template <class T> right before main
      2. Create instance of class B and perform test using int, float, char, and string data types along with struct.
         1. Test it by passing in valuea = 1.34 and valueb = 3.14
         2. Test it by passing in valuea = 1 and valueb = 3
         3. Test it by passing in valuea = ‘a’ and valueb = ‘c’
         4. Test it by passing in valuea = “good” and valueb = “morning”
         5. Test it by passing in valuea = {27, 10, 2015} and valueb = {2, 11, 2015}
      3. Print all the outputs to the screen as well as an output file called *out.txt*
   2. In class A
      1. Include template <class T> right before this class
      2. Make a private variable valuea with data type T
      3. Make a default constructor with no parameters and leave it empty
      4. In *getValue()* function with data type T
         1. Return valuea
      5. In *setValues()* function with data type void
         1. Set valuea to equal what is passed in
      6. Copy constructor with a constant pass by reference data type of class A
         1. Set valuea to equal the valuea of passed in class
   3. In class B that inherits class A
      1. Include template <class T> right before this class
      2. Make a private variable valueb with data type T
      3. Make a default constructor with no parameters and leave it empty
      4. In *getValueb()* function with data type T
         1. Return valueb
      5. In *setValueb()* function with data type void
         1. Set valueb to equal what is passed in
      6. Copy constructor with a constant pass by reference data type of class B
         1. Set valueb to equal the valueb of passed in class
2. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Pass in values of int data type | Valuea = 1  Valueb = 3 | Valuea = 1  Valueb = 3 |  |  |
| Valid | 2 | Pass in values of float data types | Valuea = 1.34  Valueb = 3.14 | Valuea = 1.34  Valueb = 3.14 |  |  |
| Valid | 3 | Pass in values of char data types | Valuea = ‘a’  Valueb = ‘c’ | Valuea = ‘a’  Valueb = ‘c’ |  |  |
| Valid | 4 | Pass in values of string data types | Valuea = “good”  Valueb = “morning” | Valuea = “good”  Valueb = “morning” |  |  |
| Valid | 5 | Pass in values of struct data types | Valuea = {27, 10, 2015}  Valueb = 2, 11, 2015} | Valuea, Day = 27  Valuea Month = 10  Valuea Year = 2015  Valueb Day = 2  Valueb month = 11  Valueb year = 2015 |  |  |

1. **Code**

Main()

// CIS200\_Lab\_9.2.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <fstream>

#include "B.h"

struct Date

{

int day;

int month;

int year;

};

using namespace std;

int main()

{

ofstream fout;

fout.open("out.txt");

B <float>objectFloat;

objectFloat.setValuea(1.34);

objectFloat.setValueb(3.14);

cout << "Float value for B: " << objectFloat.getValueb() << endl;

cout << "Flaot value for A: " << objectFloat.getValuea() << endl << endl;

fout << "Float value for B: " << objectFloat.getValueb() << endl;

fout << "Flaot value for A: " << objectFloat.getValuea() << endl << endl;

B <int>objectInt;

objectInt.setValuea(1);

objectInt.setValueb(3);

cout << "Int value for B: " << objectInt.getValueb() << endl;

cout << "Int value for A: " << objectInt.getValuea() << endl << endl;

fout << "Int value for B: " << objectInt.getValueb() << endl;

fout << "Int value for A: " << objectInt.getValuea() << endl << endl;

B <char>objectChar;

objectChar.setValuea('a');

objectChar.setValueb('c');

cout << "Char value for B: " << objectChar.getValueb() << endl;

cout << "Char value for A: " << objectChar.getValuea() << endl;

fout << "Char value for B: " << objectChar.getValueb() << endl;

fout << "Char value for A: " << objectChar.getValuea() << endl;

B <string>objectString;

objectString.setValuea("good");

objectString.setValueb("morning");

cout << "String value for B: " << objectString.getValueb() << endl;

cout << "String value for A: " << objectString.getValuea() << endl << endl;

fout << "String value for B: " << objectString.getValueb() << endl;

fout << "String value for A: " << objectString.getValuea() << endl << endl;

Date DateA = {27,10,2015};

Date DateB = { 2,11,2015 };

B <Date>objectDate;

objectDate.setValuea(DateA);

objectDate.setValueb(DateB);

cout << "Date value for B: DAY = " << objectDate.getValueb().day << endl;

cout << "Date value for B: MONTH = " << objectDate.getValueb().month << endl;

cout << "Date value for B: YEAR = " << objectDate.getValueb().year << endl;

cout << "Date value for A: DAY = " << objectDate.getValuea().day << endl;

cout << "Date value for A: MONTH = " << objectDate.getValuea().month << endl;

cout << "Date value for A: YEAR = " << objectDate.getValuea().year << endl;

fout << "Date value for B: DAY = " << objectDate.getValueb().day << endl;

fout << "Date value for B: MONTH = " << objectDate.getValueb().month << endl;

fout << "Date value for B: YEAR = " << objectDate.getValueb().year << endl;

fout << "Date value for A: DAY = " << objectDate.getValuea().day << endl;

fout << "Date value for A: MONTH = " << objectDate.getValuea().month << endl;

fout << "Date value for A: YEAR = " << objectDate.getValuea().year << endl;

fout.close();

return 0;

}

A.h

#pragma once

#ifndef A\_H

#define A\_H

using namespace std;

template <class T>

class A

{

T valuea;

public:

A() {};

T getValuea() const { return valuea; }

void setValuea(T x) { valuea = x; }

//copy constructor

A(const A &A2)

{

valuea = A2.valuea;

}

};

#endif // !A\_H

B.h

#pragma once

#ifndef B\_H

#define B\_H

#include "A.h"

template <class T>

class B : public A <T>

{

T valueb;

public:

T getValueb() const { return valueb; }

void setValueb(T x) { valueb = x; }

B() {};

//copy constructor

B(const B &B2) :A(B2)

{

valueb = B2.valueb;

}

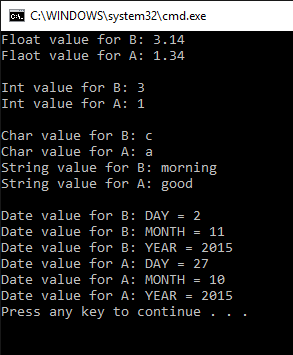
};

#endif // !B\_H

1. **Updated Algorithm**
   1. In main
      1. Make a struct with variables of day, month and year inside to be used for testing a data type
      2. Include template <class T> right before main
      3. Create instance of class B and perform test using int, float, char, and string data types along with struct.
         1. Test it by passing in valuea = 1.34 and valueb = 3.14
         2. Test it by passing in valuea = 1 and valueb = 3
         3. Test it by passing in valuea = ‘a’ and valueb = ‘c’
         4. Test it by passing in valuea = “good” and valueb = “morning”
         5. Test it by passing in valuea = {27, 10, 2015} and valueb = {2, 11, 2015}
      4. Print all the outputs to the screen as well as an output file called *out.txt*
      5. Open the output file in the beginning and close it at the end
   2. In class A
      1. Include template <class T> right before this class
      2. Make a private variable valuea with data type T
      3. Make a default constructor with no parameters and leave it empty
      4. In *getValue()* function with data type T
         1. Return valuea
      5. In *setValues()* function with data type void
         1. Set valuea to equal what is passed in
      6. Copy constructor with a constant pass by reference data type of class A
         1. Set valuea to equal the valuea of passed in class
   3. In class B that inherits class A
      1. Include template <class T> right before this class
      2. Make a private variable valueb with data type T
      3. Make a default constructor with no parameters and leave it empty
      4. In *getValueb()* function with data type T
         1. Return valueb
      5. In *setValueb()* function with data type void
         1. Set valueb to equal what is passed in
      6. Copy constructor with a constant pass by reference data type of class B
         1. Set valueb to equal the valueb of passed in class
2. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | Pass in values of int data type | Valuea = 1  Valueb = 3 | Valuea = 1  Valueb = 3 | Valuea = 1  Valueb = 3 | Pass |
| Valid | 2 | Pass in values of float data types | Valuea = 1.34  Valueb = 3.14 | Valuea = 1.34  Valueb = 3.14 | Valuea = 1.34  Valueb = 3.14 | Pass |
| Valid | 3 | Pass in values of char data types | Valuea = ‘a’  Valueb = ‘c’ | Valuea = ‘a’  Valueb = ‘c’ | Valuea = ‘a’  Valueb = ‘c’ | Pass |
| Valid | 4 | Pass in values of string data types | Valuea = “good”  Valueb = “morning” | Valuea = “good”  Valueb = “morning” | Valuea = “good”  Valueb = “morning” | Pass |
| Valid | 5 | Pass in values of struct data types | Valuea = {27, 10, 2015}  Valueb = 2, 11, 2015} | Valuea = {27, 10, 2015}  Valueb = 2, 11, 2015} | Valuea = {27, 10, 2015}  Valueb = 2, 11, 2015} | Pass |

1. **Screenshots**



1. **Status**

Program works perfectly with assumptions in mind