# COMP3512 - Assignment 1

Due: Sunday October 15th, 2017 @ 11:59 PM

This is an exercise that you need to do on a computer. You'll need to commit and push your code to your GitLab repo, and submit for automated marking via Slack.

Dessert shop Sugar High is a new shop that specializes in sweets. They want to automate their sales by building a custom checkout system where their customers can buy sweets online. They hired you to build the system for them.

For this exercise, you will need to write C++ classes that make up the checkout system.

## 1. Project Setup

1. Open Assignment1.sln in Visual Studio 2017
2. Add Item.h file to your project. (refer to Lab 1 if you don't know how)
3. Add the following content in the header file.

#pragma once

#include <string>

namespace assignment1

{

class Item

{

public:

Item(const char\* name);

virtual unsigned int GetCost() const = 0;

const std::string& GetName() const;

protected:

virtual ~Item();

private:

// private variables here

friend class ShoppingCart;

};

}

1. Add Item.cpp file to your project.
2. Add the following empty implementations in the cpp file.

#include "Item.h"

namespace assignment1

{

Item::Item(const char\* name)

{

}

Item::~Item()

{

}

const std::string& Item::GetName() const

{

return std::string(); // return name here

}

}

1. Add Candy.h file to your project and the following content in the file.

#pragma once

#include "Item.h"

namespace assignment1

{

class Candy : public Item

{

public:

Candy(const char\* name, float weight);

unsigned int GetCost() const;

private:

// private variables here

};

}

1. Add Candy.cpp file to your project and the following content in the file.

#include "Candy.h"

#include <cmath>

namespace assignment1

{

Candy::Candy(const char\* name, float weight)

: Item(name)

{

}

unsigned int Candy::GetCost() const

{

return 0U; // return cost here

}

}

1. Add Donut.h with the following content.

#pragma once

#include "Item.h"

namespace assignment1

{

class Donut : public Item

{

public:

Donut(const char\* name, unsigned int count);

unsigned int GetCost() const;

private:

// private variables here

};

}

1. Add Donut.cpp with the following content.

#include "Donut.h"

#include <cmath>

namespace assignment1

{

Donut::Donut(const char \* name, unsigned int count)

: Item(name)

{

}

unsigned int Donut::GetCost() const

{

return 0U; // return cost here

}

}

1. Add IceCream.h with the following content.

#pragma once

#include "Item.h"

namespace assignment1

{

class Cone;

class IceCream : public Item

{

public:

IceCream(const char\* name, unsigned int numScoops);

unsigned int GetCost() const;

void AddCone();

private:

// private variables here

Cone\* mCone;

};

}

1. Add IceCream.cpp with the following content.

#include "IceCream.h"

#include "Cone.h"

#include <cmath>

namespace assignment1

{

IceCream::IceCream(const char\* name, unsigned int scoops)

: Item(name)

, mCone(NULL)

{

}

unsigned int IceCream::GetCost() const

{

return 0U; // return cost here

}

void IceCream::AddCone()

{

}

}

1. Add Cone.h with the following content.

#pragma once

namespace assignment1

{

class Cone

{

public:

Cone();

unsigned int GetCost() const;

};

}

1. Add Cone.cpp with the following content.

#include "Cone.h"

namespace assignment1

{

Cone::Cone()

{

}

unsigned int Cone::GetCost() const

{

return 0U; // return cost here

}

}

1. Add Blizzard.h with the following content.

#pragma once

#include "IceCream.h"

namespace assignment1

{

class Blizzard : public IceCream

{

public:

Blizzard(const char\* name, unsigned int numScoops);

unsigned GetCost() const;

};

}

1. Add Blizzard.cpp with the following content.

#include "Blizzard.h"

namespace assignment1

{

Blizzard::Blizzard(const char\* name, unsigned int scoops)

: IceCream(name, scoops)

{

}

unsigned int Blizzard::GetCost() const

{

return 0U; // return cost here

}

}

1. Add ShoppingCart.h file to your project with the following content.

#pragma once

namespace assignment1

{

class Item;

class ShoppingCart

{

public:

ShoppingCart();

bool AddItem(const Item\* item);

bool RemoveItem(unsigned int index);

const Item\* GetItem(unsigned int index) const;

const Item\* operator[](unsigned int index) const;

float GetTotal() const;

private:

ShoppingCart(const ShoppingCart& other) {}

// private variables here

};

}

1. Add ShoppingCart.cpp and the following content with it.

#include "ShoppingCart.h"

#include "Item.h"

namespace assignment1

{

ShoppingCart::ShoppingCart()

{

}

bool ShoppingCart::AddItem(const Item\* item)

{

return false; // return true if success. Else return false

}

bool ShoppingCart::RemoveItem(unsigned int index)

{

return false; // return true if success. Else return false

}

const Item\* ShoppingCart::GetItem(unsigned int index) const

{

return NULL;

}

const Item\* ShoppingCart::operator[](unsigned int index) const

{

return NULL;

}

float ShoppingCart::GetTotal() const

{

return 0.f; // return total cost in dollars

}

}

## 2. Implement Item class

* Item class is an abstract class.
* It has a pure virtual method GetCost, which will be implemented by its derived classes.
* It has a method GetName, which returns the name of the item.
* Note that the destructor for this class is protected. This is to prevent the destruction of the Item object outside of the ShoppingCart class. This means ShoppingCart class will be responsible for deleting any Items that is added to the cart
* Item class gives friendship to ShoppingCart class to give access to its destructor to ShoppingCart.

## 3. Implement Candy class

* Candy inherits from Item abstract class. It has a weight and a price of ¢68/lbs used to calculate its cost.
* It inherits the GetName method from its parent class.
* It provides its own implementation of the GetCost method, inherited from Item.
* The cost should be rounded to the nearest cent.

## 4. Implement Donut class

* Donut inherits from Item abstract class. It has a count of donuts and a price of ¢899 per dozen used to calculate its cost.
* It inherits the GetName method from its parent class.
* It provides its own implementation of the GetCost method, inherited from Item.
* The cost should be rounded to the nearest cent.

## 5. Implement IceCream class

* IceCream inherits from Item abstract class. It has a number of scoops and a price of ¢120 per ounces, where 1 scoop of ice cream is 2 ounces, are used to calculate its cost.
* It inherits the GetName method from its parent class.
* It provides its own implementation of the GetCost method, inherited from Item.
* IceCream can optionally add a Cone with AddCone method. When a Cone is added, its cost (specified below) is added onto the cost of the ice cream. You can only add one Cone. Note that the Cone is a pointer object. You must use this object.
* The cost should be rounded to the nearest cent.

## 6. Implement Cone class

* Instantiates when added by IceCream by calling IceCream::AddCone method
* Each Cone costs ¢10.
* It has GetCost method which returns its cost.
* Note that the Cone class does not inherit from Item abstract class.

## 7. Implement Blizzard class

* Blizzard inherits from IceCream class.It behaves exactly the same as IceCream class except the cost of a topping is added onto the cost of the IceCream.
* The cost of the topping is ¢120.
* The cost should be rounded to the nearest cent.

## 8. Implement ShoppingCart class

* ShoppingCart can add Item objects into its cart using AddItem method. When NULL item is passed, it must be ignored. Return true when an Item is successfully added. Else return false.
* The maximum size of the shopping cart is 10. I.e. You cannot add more than 10 items to the cart.
* GetTotal method returns the total price of all items in the cart in **floating point** number. Please note that this is in dollars not in cents.
* GetItem method returns an Item object given an index position. If invalid index is passed, NULL should be returned.
* RemoveItem removes an Item object at a given index position. After the removal, the order of the Items must be kept and there should be no empty slots in the items list/array. If item was successfully removed, return true, otherwise return false.
* Subscription operator ([]) returns an Item object given an index position. If invalid index is passed, NULL should be returned.

Ex >

ShoppingCart cart;

.

. // Some AddItem operations to add items to the cart

.

Cart[0]; // returns the first Item object added to the cart

## 9. Test it

1. Add main.cpp into your project and test however you want. Below is a sample test you can use.

#include <iostream>

#include "Blizzard.h"

#include "Candy.h"

#include "Donut.h"

#include "IceCream.h"

#include "Item.h"

#include "ShoppingCart.h"

using namespace assignment1;

int main()

{

ShoppingCart cart;

Item\* blizzard = new Blizzard("Smarties Blizzard", 3);

Item\* candy = new Candy("Gummy Bears", 2.5f);

Item\* donut = new Donut("Canadian Maple", 4);

IceCream\* icecream = new IceCream("Vanilla", 2);

icecream->AddCone();

cart.AddItem(blizzard);

cart.AddItem(candy);

cart.AddItem(donut);

cart.AddItem(icecream);

std::cout << cart.GetTotal() << std::endl; // should print 18

}

1. Compile and run it to see the expected output.

## 5. Commit, Push and Ask for a Build

You know the drill :)

BUT, please note that

* build cooldown time for assignments is 30 mins instead of 5 mins.
* you might to have to implement extra methods to your classes