Vending a High Score

Due Tuesday, April 11 at 8 a.m.

Revision 0

CSE 1325 - Spring 2023 - Homework #9 - 1

Assignment Overview

There are two kinds of people in the world - those who think there are two kinds of people in the world, and those who do not. :)

For those who do, there are two kinds of people in the world - those who prefer to buy from a human being, and those who love vending machines.

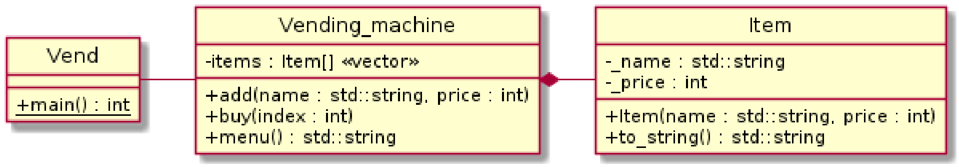
For those who love vending machines, let's code one up in C++!

IMPORTANT: I divided this assignment into full credit, bonus 1, and bonus 2 sections to give you extra points, but *it is important for you to do all 3 portions* because you will need to do them all on the exam. Each bonus is worth 15 points this week.

IMPORTANT: Code and data resources are provided to you at cse1325-prof/P09/baseline which you may freely use and adapt without attribution.

Full Credit

Consider the following class diagram. (NOTE: I prefix field names with underscores, but you may use or omit those as you please. Also, while main is always a funtion in C++, we show it in the diagram as a method because UML offers no way to represent functions.)



In your git-managed **cse1325/P09/full_credit** directory, using C++ only, write class `Item` as files item.h and item.cpp, class `Vending_machine` as files vending_machine.h and vending_machine.cpp, and file vend.cpp (containing NO class, only our main *function*). Also include a Makefile such that `make` will build all executables specified below, and `make clean` will remove all executables and .o files. (See cse1325_prof/P09/baseline/Makefile for an example that *may* work unmodified on your system for all levels of this assignment.)

Class `Item` models one product sold by the vending machine.

- The constructor initializes the `name` and `price` fields. If price is negative, throw a standard runtime error with an appropriate message.
- The `to_string()` method returns the name and price of the item as a single string. For example, if the name is "Oreos" and the price is 189, return something like "Oreos ($1.89)". (We'll delete this Java-ish approach in Bonus 1 to overload the << operator instead, like a good C++ program!)

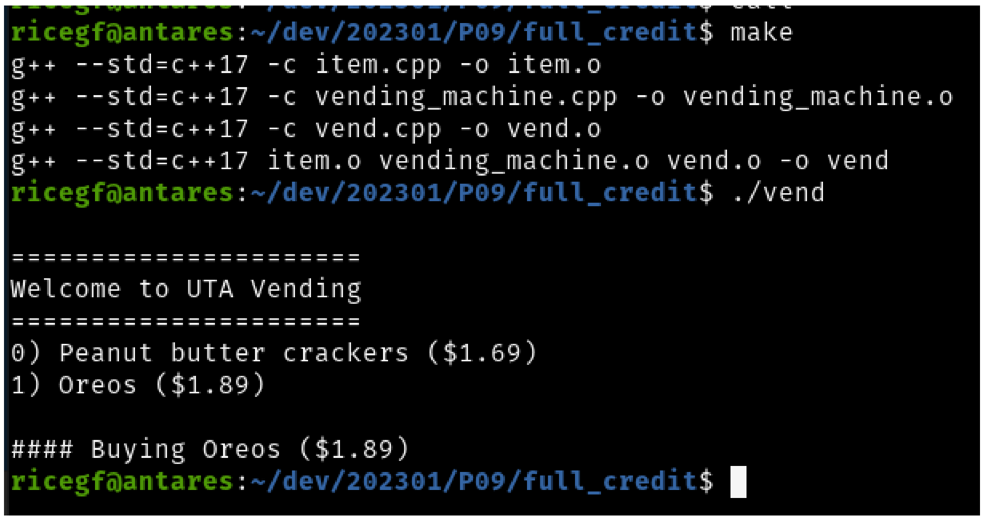
Class `Vending_machine` models the device that stores items and dispenses one on command.

- Initially, we'll allow the vector to construct an empty instance of itself, thus we need no explicit `Vending_machine` constructor. (In Bonus 2, we'll provide an input file stream object and construct a full vending machine from a text file!)
- The `add` method uses its parameters to construct an Item object and add it to the `items` field.
- The `buy` method simply prints "#### Buying " and the item that was purchased, which is the item object in the `items` vector at the index specified by the parameter.
- The `menu` method returns a std::string listing all items in the vending machine with their index.

File vend.cpp contains our main function, in which you should

- Instance a `Vending_machine`.
- Add two `Item` objects to it.
- Print the menu to the console.
- Buy one of the items (the index may be hard-coded).

Add all files to git, commit, and push to GitHub.

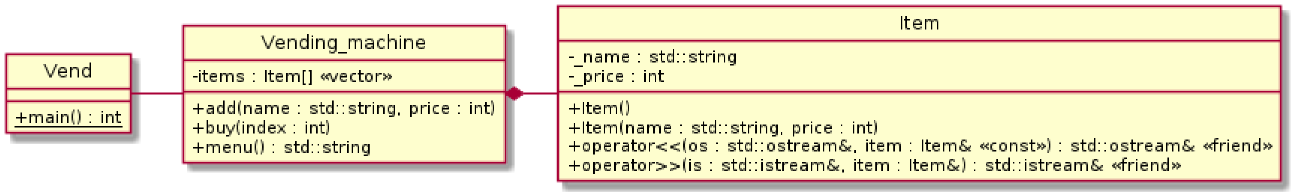


```
ricegf@antares:~/dev/202301/P09/full_credit$ make
g++ --std=c++17 -c item.cpp -o item.o
g++ --std=c++17 -c vending_machine.cpp -o vending_machine.o
g++ --std=c++17 -c vend.cpp -o vend.o
g++ --std=c++17 item.o vending_machine.o vend.o -o vend
ricegf@antares:~/dev/202301/P09/full_credit$ ./vend

======================
Welcome to UTA Vending
======================
0) Peanut butter crackers ($1.69)
1) Oreos ($1.89)

#### Buying Oreos ($1.89)
ricegf@antares:~/dev/202301/P09/full_credit$
```

Bonus 1 - Operator Overloading

In this section, we will overload the << operator for Item and Vending_machine, and the >> operator for Item. (Yes, you could also overload >> for Vending_machine, but instead we'll use our stream-based constructor in Bonus 2 to fill our Vending_machine from a text file. Hang tight!)

Duplicate your full_credit directory to the bonus1 directory. We will make the following changes to our code.


Vend
+main() : int
Vending_machine
-items : Item[] «vector»
+add(name : std::string, price : int)
+buy(index : int)
+menu() : std::string
Item
-_name : std::string
-_price : int
+Item()
+Item(name : std::string, price : int)
+operator<<(os : std::ostream&, item : Item& «const») : std::ostream& «friend»
+operator>>(is : std::istream&, item : Item&) : std::istream& «friend»

Note that we **deleted** the following methods:

- `Item::to_string()` was replaced by the `operator<<(std::ostream&, Item&)` *friend function* with equivalent functionality.
- `Vending_machine::menu()` was replaced by the `operator<<(std::ostream&, Vending_machine&)` *friend function* with equivalent functionality.

We also **added** the following constructor and friend function:

- `Item()` will construct a default Item object (to be overwritten by `operator>>`, so the field values don't matter - just use "" and 0).
- `operator>>(std::istream&, Item&)` will be needed for Bonus 2 to read Item objects from the text file. Stream in the `name` as a full line of text, and then the `price` as a full line of text converted to an int (consider >> followed by `is.ignore()`, or a `std::getline` followed by `stoi`).

Hint: For `operator<<` the `item` parameter is a *const reference* - it does not change as it is output. But for `operator>>` the `item` is a *reference* (NOT *const*) - it changes as it is input. The `std::ostream&` and `std::istream&` objects are just a *reference* because they are changed during output or input.

Hint: Be careful to return the stream in your `operator<<` and `operator>>` implementations, otherwise you'll get the dreaded segfault!

Hint: Friend functions are NOT a member of the class. They can see the class' private fields ONLY because they are a *friend* of the class. So to access (for example) the name of the item in the `operator<<` friend function, you must use `item._name` rather than just `_name`. Also in your .cpp file, don't write (for example) `Item::operator<<` - it is NOT an Item method, it's just a friend function. So just write `operator<<` instead.

Since we're using overloaded streaming operators now, you'll need to change how they are used. For example, if you had code like this for full_credit:

```
Item item{"Cookies", 195};
std::cout << item.to_string() << std::endl;
```

this would now become simply

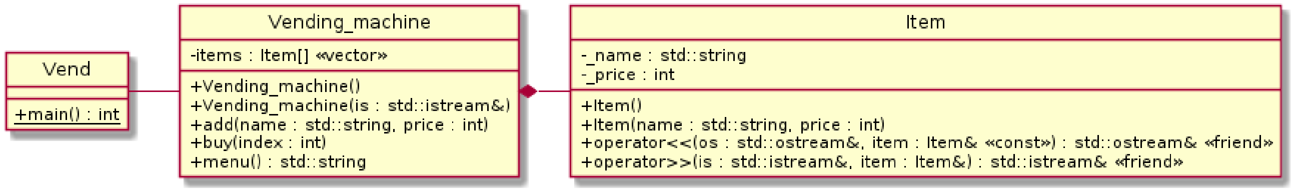
```
Item item{"Cookies", 195};
std::cout << item << std::endl;
```

Output should look the same as in the full credit section. But we know the code looks nicer, don't we?

Add, commit, and push all files.

Bonus 2

In **cse1325/P9/bonus2**, baseline your bonus1 code.


Vend
+main() : int
Vending_machine
-items : Item[] «vector»
+Vending_machine()
+Vending_machine(is : std::istream&)
+add(name : std::string, price : int)
+buy(index : int)
+menu() : std::string
Item
-_name : std::string
-_price : int
+Item()
+Item(name : std::string, price : int)
+operator<<(os : std::ostream&, item : Item& «const») : std::ostream& «friend»
+operator>>(is : std::istream&, item : Item&) : std::istream& «friend»

No changes should be needed to your Item class - we set this up in Bonus 1.

Add a default constructor to `Vending_machine`, then a second constructor that accepts a `std::istream` (standard input stream) object.

We'll actually pass in a `std::ifstream` (standard input *file* stream) object, but our constructor could actually read *any* input stream - from a network connection, from a std::string (wrapped in an input *string* stream), even `std::cin`!

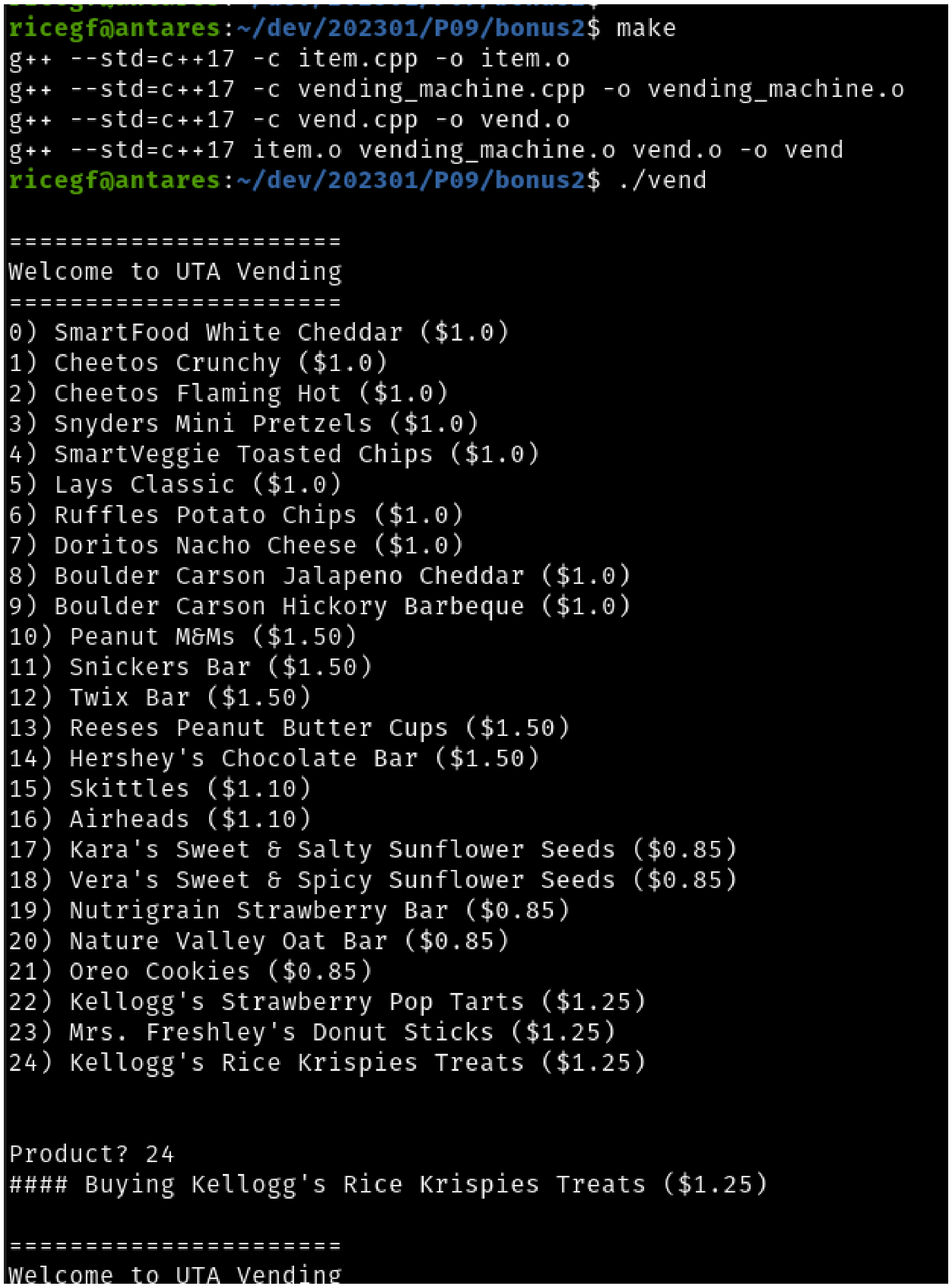
In `Vending_machine(std::istream& is)` constructor instances a default Item object, then streams `Item` objects from `is` until end of file. Expect to write the declaration plus only *three* lines of code - it's that simple!

Finally, rework your `main()` function in file vend.cpp to instance a vending machine using the file "products.txt" (one is provided for you at cse1325-prof/P09/baseline/products.txt).

Then loop until end of file:

- Display the vending machine menu a
- Accept an item number (the index) from the user, and break if it is negative
- Buy the item at that index.

Add, commit, and push all files.



```
ricegf@antares:~/dev/202301/P09/bonus2$ make
g++ --std=c++17 -c item.cpp -o item.o
g++ --std=c++17 -c vending_machine.cpp -o vending_machine.o
g++ --std=c++17 -c vend.cpp -o vend.o
g++ --std=c++17 item.o vending_machine.o vend.o -o vend
ricegf@antares:~/dev/202301/P09/bonus2$ ./vend

======================
Welcome to UTA Vending
======================
0) SmartFood White Cheddar ($1.0)
1) Cheetos Crunchy ($1.0)
2) Cheetos Flaming Hot ($1.0)
3) Snyders Mini Pretzels ($1.0)
4) SmartVeggie Toasted Chips ($1.0)
5) Lays Classic ($1.0)
6) Ruffles Potato Chips ($1.0)
7) Doritos Nacho Cheese ($1.0)
8) Boulder Carson Jalapeno Cheddar ($1.0)
9) Boulder Carson Hickory Barbeque ($1.0)
10) Peanut M&Ms ($1.50)
11) Snickers Bar ($1.50)
12) Twix Bar ($1.50)
13) Reeses Peanut Butter Cups ($1.50)
14) Hershey's Chocolate Bar ($1.50)
15) Skittles ($1.10)
16) Airheads ($1.10)
17) Kara's Sweet & Salty Sunflower Seeds ($0.85)
18) Vera's Sweet & Spicy Sunflower Seeds ($0.85)
19) Nutrigrain Strawberry Bar ($0.85)
20) Nature Valley Oat Bar ($0.85)
21) Oreo Cookies ($0.85)
22) Kellogg's Strawberry Pop Tarts ($1.25)
23) Mrs. Freshley's Donut Sticks ($1.25)
24) Kellogg's Rice Krispies Treats ($1.25)


Product? 24
#### Buying Kellogg's Rice Krispies Treats ($1.25)

======================
Welcome to UTA Vending
```