CST 238 Spring 2020, Dr. Gross, CSUMB

Project 1: Automat

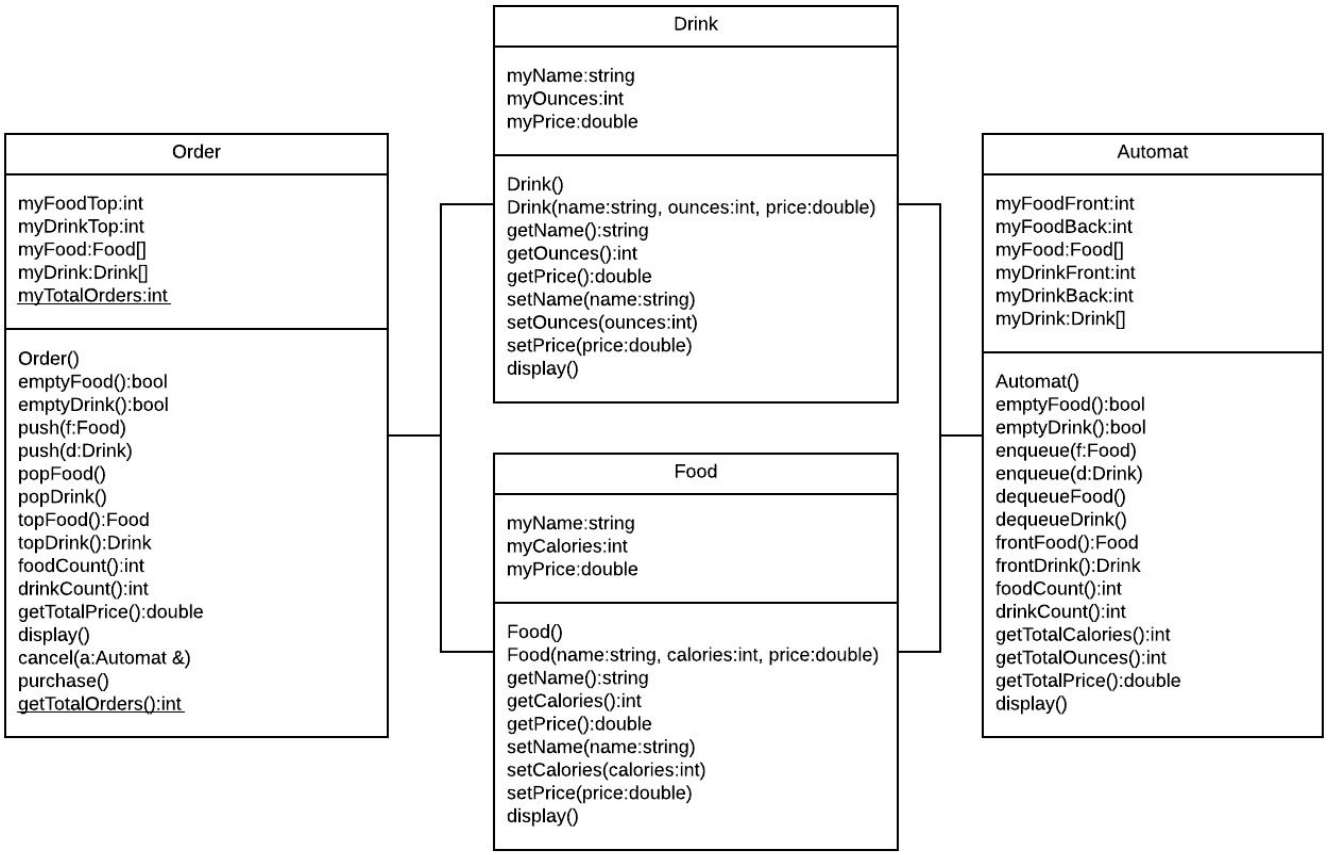
Due: Tuesday, Mar 3, 11:55pm. How to turn in:

- Write a programs and submit it on iLearn
- If you use repl.it, the file will be named main.cpp; you will need to rename it
- The names are in the assignment; incorrect names will be graded as 0

This is the first project for the course. Unlike a homework assignment, you will have two weeks to work on the project. However, it is significantly larger than most homework assignments. You will need 500-1,000 lines of code to complete this assignment.

You are going to build an automat, kind of a vending machine on steroids. The automat will hold food items and drinks. A person can add food items or drinks to an order, and then either purchase or cancel the whole order.

Class Diagram for System



Classes, Methods, and Attributes

Food

This is a fairly simple class to store information about a food item, including a name (e.g., "Let's Potato Chips"), a number of calories, and a price. In addition to constructors, methods to get and set those attributes must be created, along with a function to easily display the content.

Attributes

Name	Data Type	Example
`myName`	string	"Let's Potato Chips"
`myCalories`	int	500
`myPrice`	double	1.5

Methods

Name	Return	Parameter	Description
`Food`		`N/A`	Default constructor

`Food`		`name:string`
		`calories:int`
		`price:double`
			Parameterized constructor, should take parameters and assign value to attributes

`getName()`	`string`	`N/A`	Return the value of myName
`getCalories()`	`int`	`N/A`	Return value of myCalories
`getPrice()`	`double`	`N/A`	Return the value of myPrice
`setName()`	`void`	`name:string`	Set the value of myName
`setCalories()`	`void`	`calories:int`	Set value of myCalories
`setPrice()`	`void`	`price:doubt`	Set the value of myPrice
`display()`	`void`	`N/A`	Print out information in pattern "name: x calories, $y" e.g.: `Let's Potato Chips: 300 calories, $1.5`

Drink

This is a fairly simple class to store information about a drink, including a name (e.g., "Coffee"), a number of ounces, and a price. In addition to constructors, methods to get and set those attributes must be created, along with a function to easily display the content.

Attributes

Name	Data Type	Example
`myName`	string	"Coffee"
`myOunces`	int	12
`myPrice`	double	1.5

Methods

Name	Return	Parameter	Description
`Drink`		`N/A`	Default constructor
`Drink`		`name:string ounces:int price:double`	Parameterized constructor, should take parameters and assign value to attributes
`getName()`	`string`	`N/A`	Return the value of myName
`getOunces()`	`int`	`N/A`	Return value of myOunces
`getPrice()`	`double`	`N/A`	Return the value of myPrice
`setName()`	`void`	`name:string`	Set the value of myName
`setOunces()`	`void`	`ounces:int`	Set value of myOunces
`setPrice()`	`void`	`price:doubt`	Set the value of myPrice
`display()`	`void`	`N/A`	Print out information in pattern "name: x ounces, $y" e.g.: `Coffee: 12 ounces, $1.5`

Automat

This is a form of vending machine that holds two queues: one for food items and one for drinks. Orders (described below) can see the frontmost Food and the frontmost Drink, and can add either by calling the appropriate front and dequeue methods. If an order is canceled, the items must be returned to the Automat by enqueuing them using the appropriate method.

You should also have a const int variable called AUTOMAT_MAX set to 10 to control the array bounds. This means that the Automat can only contain at most 9 Food items and at most 9 Drink items at a time. Attempts to add more should result in an error message (e.g., "`Automat food full; new food not added`"), but the program should continue to run properly.

Attributes

Name	Data Type	Example
`myFoodFront`	int	0
`myFoodBack`	int	0
`myDrinkFront`	int	0
`myDrinkBack`	int	0
`myFood`	Food[]	array of Food objects
`myDrink`	Drink	array of Drink objects

Methods

Name	Return	Parameter	Description
`Automat`	`N/A`	`N/A`	Default constructor
`emptyFood()`	`bool`	`N/A`	Return true if there are no Food items, false otherwise
`emptyDrink()`	`bool`	`N/A`	Return true if there are no Drink items, false otherwise
`enqueue()`	`void`	`f:Food`	Add a Food item to the back of the Food queue
`enqueue()`	`void`	`d:Drink`	Add a Drink item to the back of the Drink queue
`dequeueFood()`	`void`	`N/A`	Move myFoodFront forward to remove the frontmost Food
`dequeueDrink()`	`void`	`N/A`	Move myDrinkFront forward to remove the frontmost Drink
`frontFood()`	`Food`	`N/A`	Get the frontmost Food item
`frontDrink()`	`Drink`	`N/A`	Get the frontmost Drink item
`foodCount()`	`int`	`N/A`	Get the count of Food items currently in the Automat
`drinkCount()`	`int`	`N/A`	Get the count of Drink items currently in the Automat
`getTotalCalories()`	`int`	`N/A`	Return the sum of the calories of all Food items stored
`getTotalOunces()`	`int`	`N/A`	Return the sum of the ounces of all Drink items stored
`getTotalPrice()`	`double`	`N/A`	Return the sum of the prices of all Food and Drink items
`display`	`void`	`N/A`	Print out the full Automat state; see examples below

Order

An Order allows a user to add Food and/or Drink items, each to a separate stack. The user can then choose to cancel the Order, which will result in all items being returned to the Automat, or purchase the order, which should result in the stacks being emptied. NOTE: Adding items to an Order and then canceling the order WILL change the sequencing of items in the Automat. This is fine.

You should also have a const int variable called ORDER_MAX set to 5 to control the array bounds. This means that the Order can only contain at most 5 Food items and at most 5 Drink items at a time. Attempts to add more should result in an error message (e.g., "`Order food full; new food not added`"), but the program should continue to run properly.

Attributes

Name	Data Type	Example
`myFoodTop`	int	0
`myDrinkTop`	int	0

`myFood`	Food[]	array of Food objects
`myDrink`	Drink	array of Drink objects
`static myTotalOrders`	int	0

Methods

Name	Return	Parameter	Description
`Order`	`N/A`	`N/A`	Default constructor, should also increment myTotalOrders by 1
`emptyFood()`	`bool`	`N/A`	Return true if there are no Food items, false otherwise
`emptyDrink()`	`bool`	`N/A`	Return true if there are no Drink items, false otherwise
`push()`	`void`	`f:Food`	Add a Food item to the top of the Food stack
`push()`	`void`	`d:Drink`	Add a Drink item to the top of the Drink stack
`popFood()`	`void`	`N/A`	Move myFoodTop down to remove the topmost Food
`popDrink()`	`void`	`N/A`	Move myDrinkTop down to remove the topmost Drink
`topFood()`	`Food`	`N/A`	Get the topmost Food item
`topDrink()`	`Drink`	`N/A`	Get the topmost Drink item
`foodCount()`	`int`	`N/A`	Get the count of Food items currently in the Order
`drinkCount()`	`int`	`N/A`	Get the count of Drink items currently in the Order
`getTotalPrice()`	`double`	`N/A`	Return the sum of the prices of all Food and Drink items
`display`	`void`	`N/A`	Print out the full Automat state; see examples below
`cancel()`	`void`	`Automat &`	Should pop each item from each stack, in order, and enqueue into the Automat. Note that the Automat MUST be passed by reference.
`purchase()`	`void`	`N/A`	Should reset each stack
`getTotalOrders()`	`int`	`N/A`	A STATIC method to return myTotalOrders

Examples

Imagine you have the following test code:

```
Automat a;
a.enqueue(Food("Let's Potato Chips", 300, 1.50));
a.enqueue(Food("Mini Donuts", 500, 1));
a.enqueue(Drink("Coffee", 6, 5.50));
a.enqueue(Drink("Nacho Drink", 22, 2.50));
cout << "Starting Automat\n";
a.display();
```

```
Order o;
o.push(a.frontFood());
a.dequeueFood();
o.push(a.frontDrink());
a.dequeueDrink();
cout << "Ordered one food, one drink\n";
o.display();
a.display();

cout << "Canceled order\n";
o.cancel(a);
o.display();
a.display();

cout << "Ordered one food, one drink, purchased\n";
o.push(a.frontFood());
a.dequeueFood();
o.push(a.frontDrink());
a.dequeueDrink();
o.purchase();
o.display();
a.display();
```

This should result in the following output:

```
Starting Automat
Automat:
  Food:
      Let's Potato Chips: 300 calories, $1.5
      Mini Donuts: 500 calories, $1
      2 food items - Total Calories: 800
  Drinks:
      Coffee: 6 ounces, $5.5
      Nacho Drink: 22 ounces, $2.5
      2 drinks - Total Ounces: 28

Ordered one food, one drink
Order:
  Food (1 items):
      Let's Potato Chips: 300 calories, $1.5
  Drink (1 items):
      Coffee: 6 ounces, $5.5
  Order total: $7

Automat:
  Food:
      Mini Donuts: 500 calories, $1
      1 food items - Total Calories: 500
```

```
  Drinks:
      Nacho Drink: 22 ounces, $2.5
      1 drinks - Total Ounces: 22

Canceled order
Order:
  Food (0 items):
  Drink (0 items):
  Order total: $0

Automat:
  Food:
      Mini Donuts: 500 calories, $1
      Let's Potato Chips: 300 calories, $1.5
      2 food items - Total Calories: 800
  Drinks:
      Nacho Drink: 22 ounces, $2.5
      Coffee: 6 ounces, $5.5
      2 drinks - Total Ounces: 28

Ordered one food, one drink, purchased
Order:
  Food (0 items):
  Drink (0 items):
  Order total: $0

Automat:
  Food:
      Let's Potato Chips: 300 calories, $1.5
      1 food items - Total Calories: 300
  Drinks:
      Coffee: 6 ounces, $5.5
      1 drinks - Total Ounces: 6
```

Final Notes:

- **IF YOU MISSPELL, MISCAPITALIZE, OR OTHERWISE MISNAME ANYTHING, IT CAN COST YOU POINTS, UP TO RECEIVING ZERO CREDIT FOR THE ASSIGNMENT**
- To simplify the assignment, you should NOT use the following
    - **These rules are only for THIS PROJECT**
    - **Do NOT inline any functions**
    - **Do NOT use const parameters**
    - **Do NOT use const member functions**
    - **Do NOT use static except where specifically noted**
    - **Do NOT use pass-by-reference except where specifically noted**
- This program should be uploaded to iLearn as project-1.cpp