Fall 2015.

Point Of Sale Application

OOP244 Assignment V5.0 Final Project

Your job for this project is to prepare an application that manages the list of items stored in a store for sale. Your application keeps track of the quantity of items in the store, saved in a file and updates their quantity as they are sold.

The types of items kept in store are Perishable or Non-perishable.

- Perishables: Items that are mostly food and vegetable and have expiration date.
- Non-perishables: Items that are for household use and don't have expiry date.

To prepare the application you need to create several classes that encapsulate the different tasks at hand.

CLASSES TO BE DEVELOPED

The classes required by your application are:

Date	A class that manages date and time.
PosIO	A class that enforces iostream read and write functionality for the derived classes. An instance of any class derived from "PosIO" can read from or write to the console, or be saved to or retrieved from a text file. Using this class the list of items can be saved to a file and retrieved later, and individual item specifications can be displayed on screen (in detail or as a bill item) or read from keyboard.
Item	A class derived from PosIO, containing general information about an item in the store, like the name, Stock Keeping Unit (SKU) number, price, etc.
Perishable	A class holding information for a Perishable item derived from the "Item" class that implements the requirements of the "PosIO" (i.e. implements the pure virtual methods of the PosIO class).
NonPerishable	A class derived from the "Item" class that implements the requirements of the "PosIO" class.
PosSys	The class that manages Perishable and Non-Perishable items in a file. This class manages the listing, adding and updating the data file as the items are bought or sold in the store.

PROJECT DEVELOPMENT PROCESS

Your development work on this project has five milestones and therefore is divided into five deliverables. Shortly before the due date of each deliverable a tester program and a script will be provided to you. Use this tester program to test your solution and use the "submit" command for each of the deliverables as you do for your workshop.
Since the design of this project is an ongoing process, you may have to make minor changes to the previous milestones if there is a bug or incorrect design specs, when a new milestone is published.
The approximate schedule for deliverables is as follows

• Date class	Due: Kickoff (KO) + 5 days
• PosIO class	Due: KO + 7 days
• Item class	Due: KO + 13 days
• Perishable and NonPerish classes	Due: KO + 17 days
• PosSys class.	Due: KO + 23 days

FILE STRUCTURE FOR THE PROJECT

Each class will have its own header (.h) file and implementation (.cpp) file. The names of these files should be the class name.

In addition to the header files for each class, create a header file called "POS.h" that defines general values for the project, such as:

```
TAX (0.13)               The tax rate for the goods
MAX_SKU_LEN (7)          The maximum size of an SKU code

MIN_YEAR (2000)          The min year used to validate year input
MAX_YEAR (2030)          The max year used to validate year input

MAX_NO_ITEMS (2000)       The maximum number of records in the data file.
```

Include this header file wherever you use these values.

Enclose all the code developed for this application within the sict namespace.

Make sure all your header files are guarded against multiple inclusions by adding the following commands at the very beginning of your header file:

```
#ifndef SICT_HeaderFileName_H__
#define SICT_HeaderFileName_H__
```

And adding the following command to the very end of your header files:

```
#endif
```

The "HeaderFileName" in the first two commands are replaced with the name of your header file; for example if your header file name is PosSys.h then the commands will be:

```
#ifndef SICT_POSSYS_H__
#define SICT_POSSYS_H__
```

MILESTONE 1: THE DATE CLASS

The Date class encapsulates a single date and time value in the form of five integers: year, month , day, hour and minute. The date value is readable by an istream and printable by an ostream using the following format: YYYY/MM/DD, hh:mm or YYYY/MM/DD if the class it to hold only the date without the time. (if `_dateOnly` is true; see "`bool _dateOnly;`")

Complete the implementation of the Date class under the following specifications:

Member Data (attributes):

`int _year;` Year; a four digit integer between MIN_YEAR and MAX_YEAR, as defined in "POS.h"

`int _mon;` Month of the year, between 1 to 12

`int _day;` Day of the month, note that in a leap year February has 29 days, (see mday() member function)

`int _hour;` A two digit integer between 0 and 23 for the hour the a day.

`int _min;` A two digit integer between 0 and 59 for the minutes passed the hour

`int _readErrorCode;` Error code which identifies the validity of the date and, if erroneous, the part that is erroneous. Define the possible error values defined in the Date header-file as follows:

```
NO_ERROR    0  -- No error - the date is valid
CIN_FAILED  1  -- istream failed on accepting information
YEAR_ERROR  2  -- Year value is invalid
MON_ERROR   3  -- Month value is invalid
DAY_ERROR   4  -- Day value is invalid
HOUR_ERROR  5  -- Hour value is invalid
MIN_ERROR   6  -- Minute value is invalid
```

`bool _dateOnly;` A flag that is true if the object is to only hold the date and not the time.

Private Member functions (private methods):

`int value()const; (this function is already implemented and provided)`

This function returns a unique integer number based on the date-time. You can use this value to compare two dates. If the value() of one date-time is larger than the value of another date-time, then the former date-time (the first one) follows the second.

`void errCode(int errorCode);`

Sets the _readErrorCode member variable to one of the possible values listed above.

`void set(int year, int mon, int day, int hour, int min);` Sets the member variables to the corresponding arguments ant then sets the _readErrorCode to `NO_ERROR`.

Constructors:

No argument constructor: Sets the _dateOnly attribute to false and then sets the date and time to the current system's date and time using the set() function. (see "`void set();`")

Three argument constructor: This constructor sets the _dateOnly attribute to true and then accepts three integer arguments to set the values of _year, _mon and _day and sets _hour and _min to zero. It also sets the _readErrorCode to NO_ERROR.

Five argument constructorSets the _dateOnly attribute to false and then accepts five integer arguments to set the values of _year, _mon, _day, _hour and _min. It also sets the _readErrorCode to NO_ERROR. The last argument of this constructor (int min) should have a default value of "0" so the constructor can be called with four arguments too.

Public member-functions (methods) and operators:

Relational operator overloads:

```
      bool operator==(const Date& D)const;
      bool operator!=(const Date& D)const;
      bool operator<(const Date& D)const;
      bool operator>(const Date& D)const;
      bool operator<=(const Date& D)const;
      bool operator>=(const Date& D)const;
```

These operators return the result of comparing the left operand to the right operand. These operators use the value() member function in their comparison. For example `operator<` returns true if `this->value()` is less than `D.value()`; otherwise returns false.

`int mdays()const;` (this function is already implemented and provided)

This function returns the number of days in the month based on _year and _mon values.

`void set();` (this function is already implemented and provided)

This function sets the date and time to the current date and time of the system.

Accessor or getter member functions (methods):

`int errCode()const;` Returns the _readErrorCode value.

`bool bad()const;` Returns true if _readErrorCode is not equal to zero.

`bool dateOnly()const;` Returns the _dateOnly attribute.

`void dateOnly(bool value);` Sets the _dateOnly attribute to the "value" argument. Also if the "value" is true, then it will set _hour and _min to zero.

IO member-funtions (methods):

`std::istream& read(std::istream& is = std::cin);`

Reads the date in the following format: YYYY/MM/DD (e.g. 2015/03/24) from the console if _date only is true or in the following format: YYYY/MM/DD, hh:mm (e.g. 2015/03/24, 22:15) if _dateonly is false. This function does not prompt the user. If the istream`(istr)` object fails at any point, this function sets `_readErrorCode to CIN_FAILED` and does NOT clear the istream object. If the istream`(istr)` object reads the numbers successfully, this function validates them. It checks that they are in range, in the order of year, month and day (see the general header-file and the mday() function for acceptable ranges for years and days respectively). If any number is not within range, this function sets `_readErrorCode` to the appropriate error code and omits any further validation. Irrespective of the result of the process, this function returns a reference to the istream`(istr)` object.

```
std::ostream& write(std::ostream& ostr = std::cout)const;
```

This function writes the date to the ostream`(ostr)` object in the following format: YYYY/MM/DD, if _dateOnly is true or YYYY/MM/DD, hh:mm if _dateOnly is false. Then it returns a reference to the ostream`(istr)` object.

Non-member IO operator overloads: (Helpers)

After implementing the Date class, overload the operator<< and operator>> to work with cout to print a Date, and cin to read a Date, respectively, from the console.

Use the read and write member functions. DO NOT use friends for these operator overloads.

Include the prototypes for these helper functions in the date header file.

Preliminary task

To kick-start the first milestone download the Visual Studio project, or individual files for milestone 1 from https://github.com/Seneca-OOP244/FP_MS1

MILESTONE 2: THE POSIO INTERFACE V1.0

The PosIO class is provided to enforce inherited classes to implement functions to work with **fstream** and **iostream** objects.

Download / Clone https://github.com/Seneca-OOP244/FP_MS2 and code /add a class called PosIO in PosIO.h file for your milestone 2 implementation:

You do not need the Date class for this milestone.

Pure virtual member functions (methods):

PosIO class, being an interface, only exists at class definition in a header file and has only four pure virtual member functions (methods) with following names:

1- save

Is a constant member function (does not modify the owner) and receives and returns references of std::fstream.

In future milestones children of PosIO will implement this method, when they are to be stored in a file.

2- load

Receives and returns references of std::fstream.

In future milestones children of PosIO will implement this method, when they are to be read from a file.

3- write

Is a constant member function and returns a reference of std::ostream.
write() receives two arguments: the first is a reference of std::ostream and the second is a bool argument called linear.

In future milestones children of PosIO will implement this method when they are to be printed on the screen in two different formats:
Linear: *the class information is to be printed in one line*
Form: *the class information is to be printed in several lines like a form.*

4- read

Returns and receives references of std::istream.

In future milestones children of PosIO will implement this method when their information is to be received from console.

As you already know, these functions only exist as prototypes in the class definition in the header file.

Submission:

Compile and test your PosIO.h with the provided class TestFile (TestFile.cpp, TestFile.h) and PosIOTester.cpp.

TestFile implements all the pure virtual methods of the PosIO to write and read, into and from a file and display the content of that file in linear or Form format.

When program runs for the first time, it will create a file call posfile.txt and asks you to add to its content by typing few lines from console.

Every time you run this program you will add to the content you added before. If everything compiles and works as described, then you can submit this milestone as usual:

```
$ ~professor.name/submit ms2 <ENTER>
```

MILESTONE 3: THE ITEM CLASS

Create a class called Item. The class Item is responsible for encapsulating a general PosIO item.

Although the class Item is a PosIO (inherited from PosIO) it will not implement any of the pure virtual member functions, therefore it remains abstract.

The class Item is implemented under the sict namespace. Code the Item class in the Item.cpp and Item.h files provided in FP_MS3 repository on github:

https://github.com/Seneca-OOP244/FP_MS3

You do not need the Date class for this milestone.

Item Class specs:

Private Member variables:

_sku: Character array, MAX_SKU_LEN + 1 characters long
  This character array holds the SKU (barcode) of the items as a string.
_name: Character pointer
  This character pointer points to a dynamic string that holds the name of the Item
_price: Double
  Holds the Price of the Item
_taxed: Boolean
  This variable will be true if this item is taxed
_quantity: Integer
  Holds the on hand (current) quantity of the item.

Public member functions and constructors

No argument Constructor;

This constructor sets the item to a safe recognizable empty state. All number values are set to zero in this state.

Four argument Constructor;

Item is constructed by passing 4 values to the constructor:
the SKU, the Name, the price and if the Item is taxed or not.
The constructor:

- Copies the SKU into the corresponding member variable up to MAX_SKU_LEN characters.
- Allocates enough memory to hold the name in the _name pointer and then copies the name into the allocated memory pointed to by the member variable _name.
- Sets quantity on hand to zero.
- Sets the rest of the member variables to the corresponding values received by the arguments.
- If value for Item being taxed is not provided, it will set the _taxed flag to the default value “true”

`Copy Constructor;`
`See below:`

Dynamic memory allocation necessities

Implement the copy constructor and the operator= so the item is copied from and assigned to another Item safely and without any memory leak. Also implement a virtual destructor to make sure the memory allocated by _name is freed when Item is destroyed.
In operator=, if an Item is being set to another Item that is in safe empty state, the operation will be ignored.

Accessors

Setters:

Create the following setter functions to set the corresponding member variables:
- **`sku`**
- **`price`**
- **`name`**
- **`taxed`**
- **`quantity`**

All the above setters return void.

Getters:

Create the following getter functions to return the values or addresses of the member variables: (these getter methods do not receive any arguments)
- **`sku`**`, returns constant character pointer`
- **`price`**`, returns double`
- **`name`**`, returns constant character pointer`
- **`taxed`**`, returns boolean`
- **`quantity`**`, returns integer`

Also:
- **`cost`**`, returns double`

Cost returns the cost of the item after tax. If the Item is not taxed the return value of cost() will be the same as price.
- **`isEmpty`** `returns bool`

isEmpty return true if the Item is in a safe empty state.
All the above getters are constant methods, which means they CANNOT modify the owner.

Member Operator overloads:

`Operator==` `: receives a constant character pointer and returns a Boolean.`

This operator will compare the received constant character pointer to the SKU of the Item, if they are the same, it will return true or else, it will return false.

`Operator+= : receives an integer and returns an integer.`

This operator will add the received integer value to the quantity on hand of the Item, returning the sum.

`Operator-= : receives an integer and returns an integer.`

This operator will reduce the quantity on had of the Item by integer value returning the quantity on hang after reduction.

Non-Member operator overload:

`Operator+= :` receives a double reference value as left operand and a constant Item reference as right operand and returns a double value;
This operator multiplies the cost of the Item by the quantity of the Item and then adds that value to the left operand and returns the result.
Essentially this means this operator adds the total cost of the item on hand to the left operand, which is a double reference, and then returns it.

Non-member IO operator overloads:

After implementing the Item class, overload the operator<< and operator>> to work with ostream (cout) to print a Item to, and istream (cin) to read a Item from, the console. Use the write() and read()methods of PosIO class to implement these operator overloads.

Make sure the prototype of the functions are in Item.h.

Submission:

Compile and test your PosIO.h, POS.h, Item.cpp and Item.h with the provided tester program (ItemTester.cpp). Then, if not already on Matrix, copy your files to matrix and issue the usual command to submit milestone 3:

```
$ ~professor.name/submit ms3<ENTER>
```

MILESTONE 4: THE NONPERISHABLE AND PERISHABLE CLASSES

Part one: NonPerishable

Before starting this, please download/clone the files provided from https://github.com/Seneca-OOP244/FP_MS4 then add all the classes you implemented in previous milestones. To begin, look at the code for the ErrorMessage class:

ErrorMessage is already coded and ready to use. It is a very simple class. Essentially it is a container for a string of 80 characters responsible for holding an error message.
ErrorMessage has the following member functions:

ErrorMessage();	A constructor that creates an empty ErrorMessage
void clear();	Clears the error message to an empty string.
bool isClear()const;	Returns true if the ErrorMessage is empty (No Error)
void message(const char* value);	Sets the ErrorMessage to an error message!
const char* message()const;	This accessor returns the error message to be printed.

We use this object to capture the status of the NonPerishable and Perishable objects. If an error happens during console entry, we set this object to the proper message, to be shown later if needed. Using this, we can find out if a NonPerishable or Perishable object is in an erroneous state and take proper action.

NonPerishable Class

Implement the NonPerishable class as a class derived from an Item class.
Essentially, NonPerishable is an Item class that is not abstract.

Private member variables

NonPerishable class has only one private member variable of type ErrorMessage, called **_err**.

Constructor:

No constructors are created for this class.

Public member functions

NonPerishable implements all four pure virtual methods of the class PosIO (the signatures of the functions are identical to those of PosIO).

std::fstream& save(std::fstream& file)const:

Using the operator<< of ostream first writes the character **"N"** and a comma into the **file** argument, then without any formatting or spaces writes all the member variables of Item, comma separated, in following order:

```
          sku, name, price, taxed, quantity
```
and ends them with a new line. Then it will return the file argument out.

Example:

```
N,1234,Candle,1.23,1,38<Newline>
```

std::fstream& load(std::fstream& file)

Using the operator>>, ignore and getline methods of istream, NonPerishable reads all the fields form the file and sets the member variables using the setter methods. When reading the fields, load assumes that the record does not have the "**N,**" at the beginning, so it starts the reading from the sku.

No error detection is done.
At the end the file argument is returned.

Hint: create temporary variables of type double, int and string and read the fields one by one, skipping the commas. After each read, set the member variables using setter methods.

std::ostream& write(std::ostream& ostr, bool linear)const

If the **_err** member variable is not clear (use isClear member function) it simply prints the _err using ostr and returns ostr. If the **_err** member variable is clear (No Error) then depending on the value of linear, write(), prints the Item in different formats:

Linear is true:

Prints the Item values separated by Bar "|" character in following format:

```
1234   |Candle              |   1.23| t |  38|    52.82|
```

SKU:	left justified in MAX_SKU_LEN characters
Name:	left justified 20 characters wide, trimmed to 20 if longer than 20 characters.
Price:	right justified, 2 digits after decimal point 7 chars wide
Taxed:	"t" if it is taxed, space if it is not taxed, 3 chars wide "t" at centre
Quantity:	right justified 4 characters wide
Cost:	total cost, considering quantity and tax; same format as price, 9 chars wide

One Bar and **NO NEW LINE**

Linear is false:

Prints one member variable per line in following layout.
All formats are like linear layout, with no width restriction, except Name, which occupies one line; 80 chars.

```
Name:
Candle
Sku: 1234
Price: 1.23
Price after tax: 1.39
```

```
Quantity: 38
Total Cost: 52.82 <Newline>

OR if not taxed

Sku: 1234
Name: Candle
Price: 1.23
Price after tax: N/A
Quantity: 38
Total Cost: 46.74 <Newline>
```

Afterwards, write returns the ostr argument.

std::istream& read(std::istream& istr):

Receives the values using istream (the istr argument) exactly as the following:

```
Non-Perishable Item Entry:
Sku: 1234<ENTER>
Name:
Candle<ENTER>
Price: 1.23<ENTER>
Taxed: y<Enter>
Quantity: 38<ENTER>
```

if **istr** is in a **fail** state, then the function exits doing nothing other than returning istr. If at any stage istr fails (cannot read), **_err** will be set to the proper error message and the rest of the entry is skipped and nothing is set in the Item (also no error message is displayed).
Here are the possible error messages:

fail at Price Entry:	**Invalid Price Entry**
fail at Taxed Entry:	**Invalid Taxed Entry, (y)es or (n)o**
fail at Quantity Entry:	**Invalid Quantity Entry**

When validating Taxed Entry, if the character entered is not a valid response to be consistent with an istream failure, manually set the istr to failure mode by calling this function:

```
istr.setstate(ios::failbit);
```

Since the rest of the member variables are text, istr cannot fail on them, therefore there are no error messages designated for them. Make sure at the end of the Entry you do not read the last new line or flush the keyboard.

At end, read will return the istr argument.

Part Two: Perishable Class

Perishable Class

Please note that the Perishable and NonPerishable classes are identical in logic. The only difference is that the Perishable class has one extra member variables that have to be received and printed (in addition to the variables in an Item).

Implement the Perishable class to be derived out of an Item class. Essentially, Perishable is an Item class that is not abstract.

Private member variables

Perishable class has two private member variables:

- An ErrorMessage, called **_err**.
- A Date, called _expiry (date only mode)

Constructor:

Create a no-argument default constructor and set the _expiry attribute to date only mode.

Public member functions

Public Accessors (setters and getters)

const Date& expiry()const;

returns a constant reference to _expiry member variable.

void expiry(const Date &value);

Sets the _expiry attribute to the incoming value.

Pure virtual method implementations

Perishable implements all four pure virtual methods of the class PosIO. (the signatures of the functions are identical to those of PosIO).

std::fstream& save(std::fstream& file)const:

Using the operator<< of ostream, this method first writes the character "**P**" and a comma into the **file** argument, then without any formatting or spaces writes all the member variables of the Item, comma separated, in following order:

```
        sku, name, price, taxed, quantity, expiry date
```

and ends them with a new line. Then it will return the file argument out. Example:

```
P,1234,4L Milk,3.99,0,2,2015/12/10<Newline>
```

std::fstream& load(std::fstream& file)

Using the operator>>, ignore and getline methods of istream, Perishable reads all the fields from the file and sets the member variables using the setter methods. When reading the fields, **load** assumes that the record does not have the “**P,**” at the beginning, so it starts the reading from the sku.

No error detection is done.
At the end the file argument is returned.

Hint: create temporary variables of type double, int, string and Date, then read the fields one by one, skipping the commas. After each read set the member variables using setter methods.

std::ostream& write(std::ostream& ostr, bool linear)const:

If the **_err** member variable is not clear (use isClear member function), it simply prints the _err using ostr and returns ostr. If the **_err** member variable is clear (No Error) then depending on the value of linear, write() prints the Item and Perishable member variables in different layouts:

Linear is true:

Prints the Item values separated by Bar “|” character in following format:

```
1234   |4L Milk             |   3.99|  p|   2|     7.98|
```

Sku:	left justified in MAX_SKU_LEN characters
Name:	left justified 20 characters wide
Price:	(not the price) right justified, 2 digits after decimal point 7 chars wide
Taxed:	“ tp” for taxed, “ p” for not taxed (3 chars wide)
Quantity:	right justified 4 characters wide
Cost:	total cost, considering quantity and tax; same format as price, 9 chars wide

NO NEW LINE

Linear is false:

Prints one member variable per line in following format:

```
Name:
4L Milk
Sku: 1234
Price: 3.99
Price after tax: 4.51
Quantity: 2
Total Cost: 9.02
Expiry date: 2015/12/10 <Newline>
```

Or if not taxed:

```
Name:
4L Milk
```

```
Sku: 1234
Price: 3.99
Price after tax: N/A
Quantity: 2
Total Cost: 7.98
Expiry date: 2015/12/10 <Newline>
```

Afterwards, write returns the ostr argument.

std::istream& read(std::istream& istr):

Receives the values using istream (the istr argument) exactly as the following:

```
Perishable Item Entry:
Sku: 1234<ENTER>
Name:
4L Milk<ENTER>
Price: 3.99<ENTER>
Taxed: n<ENTER>
Quantity: 2<ENTER>
Expiry date (YYYY/MM/DD) : 2015/12/10<ENTER>
```

if **istr** is in a **fail** state, then the function exits doing nothing other than returning istr. If at any stage istr fails (cannot read), **_err** will be set to the proper error message and the rest of the entry is skipped and nothing is set in the Item (also no error messages is displayed).
Here are the possible error messages:

fail at Price Entry:	**Invalid Price Entry**
fail at Taxed Entry:	**Invalid Taxed Entry, (y)es or (n)o**
fail at Quantity Entry:	**Invalid Quantity Entry**

When validating Taxed Entry, if the character entered is not a valid response to be consistent with an istream failure, manually set the istr to failure mode by calling this function:

```
istr.setstate(ios::failbit);
```

If Expiry (Date) Entry fails then, depending of the error code stored in the Date class, set the error message in **_err** to:

CIN_FAILED:	**Invalid Date Entry**
YEAR_ERROR:	**Invalid Year in Date Entry**
MON_ERROR:	**Invalid Month in Date Entry**
DAY_ERROR:	**Invalid Day in Date Entry**

Like Taxed to be consistent with an istream failure, manually set the istr to failure mode by calling this function:

```
istr.setstate(ios::failbit);
```

Since the rest of the member variables are text, istr cannot fail on them, therefore there are no error messages designated for them.

Make sure at the end of the Entry you do not read the last new line or flush the keyboard.

At end, read will return the istr argument.

Submission:

Compile and test your Date.cpp, Date.h, PosIO.h, POS.h, Item.cpp, Item.h, Perishable.cpp, Pershable.h, NonPerishable.cpp and NonPerishable.h with the provided tester programs:
01-NPErrHandling.cpp
02-NPDisplayTest.cpp
03-NPSaveLoad.cpp
04-PerErrHandling.cpp
05-PerDateErrHandling.cpp
06-PerDisplayTest.cpp
07-PerSaveLoad.cpp

Then, if not already on Matrix, copy your files to matrix and issue the usual command to submit milestone 4:

```
$ ~professor.name/submit ms4<ENTER>
```

MILESTONE 5: THE POSAPP CLASS

Before starting this, please download/clone the files provided from https://github.com/Seneca-OOP244/FP_MS5 then add all the classes you implemented in previous milestones to the directory.

PosApp Class

PosApp uses the classes created in milestones 1 to 4 to manage the items in a store and also works as a Point Of Sale system to sell the items to customers.

PosApp has the following private member variables (attributes):

Constructors and assignment operator overload

```
PosApp(const char* filename, const char* billfname);
```

PosApp will get instantiated using a file name and a bill file name (billfname). These two values are copied into their corresponding member variables upto 128 characters.

Copying and assignment

PosApp cannot be copied or assigned to another instance of PosApp.

Private member variables (attributes)

```
char _filename[128];
```

A C style string to hold then name of the data file where the items are stored.

```
char _billfname[128];
```

A C style string to hold then name of a temporary file to hold the items the customer buys. This file is created every time a customer is at the cash register and will hold all the items the customer is checking out. When the sale is final, the content of the file is printed a “bill” for the customer and then it is truncated (emptied) for the next sale.

```
Item* _items[MAX_NO_ITEMS];
```

An array of Item pointers that is to hold all the records of the datafile.
Every and each element of this array is set to a null pointer when PosApp is created.

```
int _noOfItems;
```

This integer will hold the number of Items read from the data file.

Private member functions (methods)

For some of the functions below, after the function description there will be a pseudo code to help you with the logic of the function. You are free to either use your own logic or follow the pseudo code’s logic,
OR
Follow the pseudo code and then modify it to a better the logic.

```
int menu();
```

Menu prints the following menu and then asks the user to select one of the options of the menu. User can enter an integer between 1 and 6 to make the selection, if the value is outside of this range the selection will be set to -1.
Before the user's selection is returned, this function goes to new line.
Note that this function does not print any error message.
The keyboard is flushed and emptied after the entry.

```
The OOPs Store
1- List items
2- Add Perishable item
3- Add Non-Perishable item
4- Update item quantity
5- Show Item
6- POS
0- exit program
> _
```

This function is the main user interface of the PosApp and is displayed every time a choice is to be made.

Data management member functions (methods)

```
void loadRecs();
```

Works in two stages:
1- Opening the file: Opens the data file for read using an instance of fstream with the filename kept in "_filename" and the ios::in flag. If the opening fails (the file does not exist) it will clear the fstream, close it and then reopens it in write mode to create an empty file (ios::out) and exit the function. If the file opening does not fail, it goes to stage two.

2- Reading the file: LoadRecs reads all the items kept in the file into the _items pointer array. Each record is read in two steps; first a single character is read to identify the type of the item (that will be either 'P' or 'N') and the next character (',') is ignored. Depending on the first character being 'P' or 'N' a Perishable or NonPershiable object is created dynamically and the address is held in the _items pointer array. The using the "load" method of the item, the rest of the information is loaded into the object.
Step "2" is repeated until end of file is reached.
See the pseudo code for more detail:

```
set readIndex to zero

open the file for reading (use ios::in)
if the file is in fail state, it means there is no file on the disk, then
  clear the failure
  close the file
  open the file for writing (ios::out) to create the file
  close thefile
otherwise (if the file is not in fail state)
  until reading fails loop:
     delete the item pointer at readindex. (not to have memory leak)
     read one character into the Id character
     if Id character is P
       Dynamically create a Perishable item and hold it in item pointer at readIndex
     if Id character is N
       Dynamically create a NFI item and hold it in item pointer at readIndex
     if either P or N is read
        skip the comma in the file
        load the data into the newly created item from the file
                                    (using its load method)
        add one to readindex
  continue the loop
set number of items to readIndex
close the datafile
```

void saveRecs();

Opens the data file for overwriting using an instance of fstream with the filename kept in "_filename" and the ios::out flag. Loops thought the "_items" array "_noOfItems" times and write them into the file using their "save" method, only if their quantity is more than zero. Then it will close the data file and calls loadRecs() to have the recent update of Items in the PosApp.

int searchItems(const char* sku)const;

Loops through the _items array and compares them with the "sku" using the overloaded "operator==". If there is a match, it will return the index of the found item, or "-1" if nothing is found.

void updateQty();

updates quantity of an Item with the same sku as the one received from the argument of the function:
This function will first prompt the user for an sku. Then it will call searchItem to find the item. If the item is not found, it will print "Not found!" and exit the function. If the item is found it will first display the item in non-linear format and then prompts the user for the number of items purchased. The received number is added it to the quantity of the item using the overloaded "operator +=". Then it will save the records (saveRecs()) and print the message "Updated!".
Function should work exactly as follows:

```
>Start
Please enter the SKU: 9999
Not found!

>END
```

```
>Start
Please enter the SKU: 1234
Name:
Milk
Sku: 1234
Price: 3.99
Price after tax: N/A
Quantity: 2
Total Cost: 7.98
Expiry date: 2015/10/04

Please enter the number of purchased items: 5
Updated!

>END
```

```
void addItem(bool isPerishable);
```

Depending of the value of the “isPershable” argument, it will dynamically create a Perishable or NonPerishable Item. Then it will receive the item information from the user using cin. If anything goes wrong, (cin fails) it will clear cin and flush the keyboard, and then display the item using cout to show the error. If the Item is received successfully from the user, it will add it to the “_items” pointer array and add one to “_onOfItems”. Then it will save the records into the file and print “Item Added.”.

```
>Start
Perishable Item Entry:
Sku: abc
name:
abc
price: abc
Invalid Price Entry

>END
```

```
>Start
NonPerishable Item Entry:
Sku: 3456
Name:
Paper Cups
Price: 5.99
Taxed: y
Quantity: 40
Item added.
```

```
>END
```

```
void listItems()const;
```

Loops through the _items pointer array up to "_noOfItems" and prints them in leaner format as follows:

```
>Start
 Row | SKU     | Item Name           | Price |TX |Qty |   Total |
-----|--------|--------------------|-------|---|----|---------|
   1 | 1234   |Milk                |   3.99|  p|   7|    27.93|
   2 | 2345   |Soap                |  23.45| t |   2|    53.00|
   3 | 3456   |Paper Cups          |   5.99| t |  40|   270.75|
-----^--------^--------------------^-------^---^----^---------^
Total Asset: $351.67
>END
```

As you can see, while looping through and printing the items it will collect the sum of "cost" of all the Items and displays it at the end as the Total Asset value.

Point Of Sale member functions (methods)

```
void showBill();
```

Opens the bill file for reading using an instance of fstream with the filename kept in "_billfname" and the ios::in flag. This file has the same format of the "_datafile". Read the items one by one, the same way you did in loadRecs method. But instead of adding the items to the _items array, print them in linear format and immediately delete what you printed. Continue until you hit the end of file. When done, close the file, and reopen it for overwrite and truncating using (ios::out|ios::trunc) to delete its content and close it again. Like this, any bill that is printed will be deleted in the file. (You cannot print a bill twice)

The printout must be exactly in following format:

```
>Start
v--------------------------------------------------------v
| 2015/12/01, 11:30                                      |
| SKU     | Item Name           | Price |TX |Qty |   Total |
|--------|--------------------|-------|---|----|---------|
| 1234   |Milk                |   3.99|  p|   1|     3.99|
| 2345   |Soap                |  23.45| t |   1|    26.50|
| 3456   |Paper Cups          |   5.99| t |   1|     6.77|
| 2345   |Soap                |  23.45| t |   1|    26.50|
^--------^--------------------^-------^---^----^---------^
Total $63.76
```

```
>END
```

Please note that the current time of the system is displayed at the top of the bill and total cost of the bill is printed at the bottom.

```
void addToBill(Item& I);
```

Adds an item to the bill file, (name kept in "_billfname") with quantity of 1.
First copy the quantity of the Item referred to by the argument **`I`**, into a local variable called **qty** for future use. Then set the quantity if **`I`** to 1. Open the bill file for appending using an instance of fstream with the filename kept in "_billfname" and the (ios::out | ios::app)flag. Like this if the file does not exist, it will create it. Then save the Item referred to by **`I`** using the "save" method. Then reduce the **`qty`** value by one, but do go bellow zero. Finally set the quantity of **`I`** to **`qty`** and save the Records (saveRecs()) to update the datafile.

```
void POS();
```

Continuously asks for an sku and if the sku exist in the data file it will add it to the bill file otherwise it will print "Not found!".

The function will stop and print the bill if and empty sku is entered (<ENTER> is hit when asking for sku without entering any data).

```
Pseudo:
while not done
   Ask for sku
   If sku is an empty string
      show bill and the function is done
   Search for sku in the items
   if found
      print the name only
      add it to the bill
   if not found
      print "Not found!"
End while
```

Your function should run with following format:

```
>START
Sku: 1234
v------------------->
| Milk
^------------------->
Sku: 1234
v------------------->
| Milk
^------------------->
Sku: 2345
Not found!
```

```
Sku: 3456
v------------------->
| Paper Cups
^------------------->
Sku: 4567
v------------------->
| Butter
^------------------->
Sku:
v--------------------------------------------------------v
| 2015/12/01, 12:03                                      |
| SKU     | Item Name           | Price  |TX |Qty |   Total |
|--------|--------------------|-------|---|----|---------|
| 1234   |Milk                |   3.99|  p|   1|     3.99|
| 1234   |Milk                |   3.99|  p|   1|     3.99|
| 3456   |Paper Cups          |   5.99| t |   1|     6.77|
| 4567   |Butter              |   4.56| tp|   1|     5.15|
^--------^--------------------^-------^---^----^---------^
Total $19.90

>END
```

Public member function (method)

PosApp has only one public member function:

```
void run();
```

In a continuous loop run will display the menu and wait for user's selection.

The following will happen upon user's selection:
User selects 1:
Call listItems.

User selects 2:
Call addItem.

User selects 3:
Call addItem.

User selects 4:
Call updateQty.

User selects 5:
Prompt the user for receiving sku.
get the sku and searchItems() for it.
if found display it in non-linear format
if no found display "Not found!".

User selects 6:
Call POS.

User selects 0:
Exit program printing "Goodbye!".

Any other value
print: “===Invalid Selection, try again===”

Your function must run as follows, note that only options 5, 0 and other is demonstrated , since the rest are already demonstrated in other methods.

```
>START
The OOPs Store
1- List items
2- Add Perishable item
3- Add Non-Perishable  item
4- Update item quantity
5- Show Item
6- POS
0- exit program
> 5

Please enter the SKU: 1234
v-----------------------------------v
Name:
Milk
Sku: 1234
Price: 3.99
Price after tax: N/A
Quantity: 4
Total Cost: 15.96
Expiry date: 2015/10/04
^-----------------------------------^

The OOPs Store
1- List items
2- Add Perishable item
3- Add Non-Perishable  item
4- Update item quantity
5- Show Item
6- POS
0- exit program
> 5

Please enter the SKU: 2345
Not found!

The OOPs Store
1- List items
2- Add Perishable item
3- Add Non-Perishable  item
4- Update item quantity
5- Show Item
6- POS
0- exit program
> 20
```

```
===Invalid Selection, try again===

The OOPs Store
1- List items
2- Add Perishable item
3- Add Non-Perishable  item
4- Update item quantity
5- Show Item
6- POS
0- exit program
> 0

Goodbye!!
>END
```

You can try the execution of the Final project by issuing the following command on matrix:

```
~fardad.soleimanoo/fp <ENTER>
```

Submission

TBA