

Recursive Change Maker

Stephen Gerkin

November 16, 2019

CIST 2362 CRN 22282

Program Description

Create an algorithm to demonstrate recursive concepts. Demonstrate the algorithm with a driver program.

Programming Strategy:

A class that can determine what denominations to return to a customer will be created. It will receive the amount owed for a product and the amount tendered by the customer. An algorithm will then be used to recursively add denominations to a list of what we are to return to the customer. Each successive call to the algorithm will include either a lowered amount due for making change or a list of denominations we use not included. Further information on how this algorithm works can be found at the end of the **ChangeMaker** class documentation for the *makeChange()* function (or internally within the code).

A **Demonstration** class will be created to allow the user to enter amounts and then display the denominations to return based on the amount owed and the amount tendered. Additionally, input functions will themselves be recursive when bad input is received from the user.

Class Documentation

ChangeMaker

ChangeMaker is a utility class for making change given an amount owed and an amount tendered by a customer.

The amounts given should be integer values of the lowest possible currency denomination (ie \$10.00 becomes 1000). The class uses a map of denomination values and the names associated with them to determine how to make change for a customer. Given that a customer purchases an item that costs \$15.59, and tenders \$20.00 to us, we should return \$4.41 in change. This class will take that information and return a map of the bills and coins to return to the customer (4 \$1 bills, 1 quarter, 1 dime, 1 nickel, 1 penny). The map returned is in ascending order of the name of the denomination. As such, this class provides a **getChangeStr()** method that will return a string representation of what we are to return with each denomination on a new line. Take care to first make change before calling this function or an exception will be thrown. An exception will also be thrown if the amount owed is greater than the amount tendered.

Member Variables

Data Type	Name	Description
<code>std::map<int, std::string></code>	denominationValueNameMap	Holds a map of integer values for a denomination as the key and the string representation of the denomination as the value.
<code>std::map<std::string, int></code>	changeMap	For holding onto the quantity of a given denomination we need to return.

Member Class

`NoChangePossible` (*inherits `std::runtime_error`*)

Class exception for handling invalid amounts.

This exception can result from attempting to make change when a customer still owes us money (ie the item to buy costs \$20.00 but they only give us \$10.00). It is also thrown if a call to **getChangeStr()** occurs before making any change using **makeChange()**.

Constructors and Destructors

`ChangeMaker::ChangeMaker ()`

Default constructor.

Initializes the map we are to use for our denominations.

`ChangeMaker::~ChangeMaker() [default]`

Member Function Documentation

`std::string ChangeMaker::getChangeStr ()`

Creates a string representation of our change to be made.

Each denomination used is added to a new line of the string with the quantity we are to use followed by the name of the denomination (ie "1 nickel").

Returns

A string containing the number of denominations to give as change.

Exceptions

<code>NoChangePossible</code>	if <code>makeChange()</code> function has not been called yet.
--------------------------------------	---

`std::vector< int > ChangeMaker::getDescOrderedDenominationList () [private]`

Creates a list (vector) of our denomination values.

It constructs the vector from our denominationValueNameMap. The list is in descending order of value to work with our algorithm for making change.

Returns

A vector containing our denomination values in descending order.

`void ChangeMaker::initChangeMap () [private]`

Initializes our changeMap object.

Each key is the name value of a denomination in the till. Each value is initialized as 0 to start off.

`std::map< std::string, int > ChangeMaker::makeChange (int amountOwed, int amountTendered)`

Creates a map of denominations and the quantity to return to a customer.

Each key represents the name of a denomination (ie "nickel") and each value is the number of that denomination to return.

Parameters

<i>amountOwed</i>	The amount the customer owes us.
<i>amountTendered</i>	The amount the customer gave to us.

Returns

A map of denominations and their quantity to give for change.

Exceptions

NoChangePossible	If the amount owed is more than the amount tendered.
-------------------------	--

`std::vector< int > ChangeMaker::makeChange (int changeAmt) [private]`

Internal starting point for makeChange algorithm.

Creates the lists we need to store information about the change we are to make and then fills each list by calling the recursive function of the same name.

Parameters

<i>changeAmt</i>	The amount of change we are to return.
------------------	--

Returns

A list of denominations (value) that is to be returned.

`void ChangeMaker::makeChange (int changeAmt, const std::vector< int > & denominationList,
std::vector< int > & changeList) [private]`

Recursive function to determine the amount of change to return.

Given an amount of change, a list of denominations, and a list to place each denomination used into, this function recurs until there is no more change to be given. Each pass through the function subtracts the highest denomination in the list (if possible) from the change amount, then calls itself again with the amount reduced by the value of the denomination used. If a denomination cannot be used, the function is called again with the same change amount but with the first denomination in the list absent.

Parameters

<i>changeAmt</i>	The amount of change to return.
<i>denominationList</i>	A list of denomination (values) to use.
<i>changeList</i>	The list of denominations we are to pull and return to the customer.

Demonstration

For demonstrating the **ChangeMaker** class.

Constant Attributes

Data Type	Name	Description
std::string	QUIT	To signal the user's desire to end the program.

Member Variables

Data Type	Name	Description
ChangeMaker	changemaker	The ChangeMaker object we will use for demonstration.

Member Class

QuitException (inherits std::exception)

For signaling to program that the user wishes to end the program.

Constructors and Destructors

Demonstration::Demonstration() [default]

Demonstration::~~Demonstration() [default]

Member Function Documentation

int Demonstration::getCurrency (const std::string & what) [private]

Gets a currency value from the user.

If invalid entry, will recur.

void Demonstration::getMoreMoney (int owed, int tendered) [private]

Gets more money from the user if the amount owed is more than tendered.

Parameters

<i>owed</i>	The current amount owed.
-------------	--------------------------

<i>tendered</i>	The current amount tendered to us.
-----------------	------------------------------------

`std::string Demonstration::getStringInput (const std::string & inputMsg) [private]`

Gets string input from a user and sets display formatting.

Parameters

<i>inputMsg</i>	Information about what value we want to receive from the user.
-----------------	--

Returns

The user input as a string.

`void Demonstration::makeChange (int owed, int tendered) [private]`

Attempts to use the **ChangeMaker::makeChange()** function.

If it fails, this is because the customer has not given enough money and this function will call the function to get more money.

Parameters

<i>owed</i>	The amount owed.
<i>tendered</i>	The amount tendered.

`void Demonstration::start ()`

Starts the demonstration.

Loops the **makeChange()** function to demonstrate until the user indicates the desire to quit.

Input and Output Demonstration

Welcome message with program description

```
Welcome to the ChangeMaker program demonstration.  
This program will demonstrate a recursive algorithm that determines  
how to make change for a purchase.  
You will be asked for an amount owed for an item and then  
an amount tendered by the customer.  
The program will then determine how to make change for these amounts  
and display the result.  
For example, here is the display you will receive if you enter the  
amount owed is $15.59 and the customer gives you a $20 bill.  
  
Here is how you will make change for that:  
1 penny  
1 nickel  
1 dime  
1 quarter  
4 $1 bill  
  
If you enter too many decimal values (0.001), the value you entered  
will be rounded as appropriate.  
  
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>>
```

Using the example given for input

```
Enter the amount the customer owes:
Enter 'quit' at any time to exit.

>> 15.59

Enter the amount the customer tendered:
Enter 'quit' at any time to exit.

>> 20.00

Here is how you will make change for that:
1 penny
1 nickel
1 dime
1 quarter
4 $1 bill

Enter the amount the customer owes:
Enter 'quit' at any time to exit.

>>
```

Using value with too much precision rounds down correctly

```
Enter the amount the customer owes:
Enter 'quit' at any time to exit.

>> 20

Enter the amount the customer tendered:
Enter 'quit' at any time to exit.

>> 20.001

There is no change to be made.
```

Using value with too much precision rounds up correctly

```
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>> 20  
  
Enter the amount the customer tendered:  
Enter 'quit' at any time to exit.  
  
>> 20.005  
  
Here is how you will make change for that:  
1 penny
```

Entering a numeric value

```
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>> asdf  
  
That was not a valid value.  
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
|>
```

Entering a negative value

```
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>> -20  
  
That was not a valid value.  
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
|>
```

Trying to make change but customer still owes

```
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>> 20  
  
Enter the amount the customer tendered:  
Enter 'quit' at any time to exit.  
  
>> 15  
  
The customer still owes money!  
Enter additional amount the customer tenders:  
Enter 'quit' at any time to exit.  
  
>> 4.49  
  
The customer still owes money!  
Enter additional amount the customer tenders:  
Enter 'quit' at any time to exit.  
  
>> 20  
  
Here is how you will make change for that:  
4 penny  
2 dime  
1 quarter  
4 $1 bill  
1 $5 bill  
1 $10 bill
```

Quit entered

```
Enter the amount the customer owes:  
Enter 'quit' at any time to exit.  
  
>> quit  
  
Process finished with exit code 0
```