```
//
//  main.cpp
//  assignment1
//
//  Created by Lucie Chevreuil on 9/6/23.
//

#include <iostream>
#include <vector>
using namespace std;

// ################ QUESTION 1
#####################################################################
class Money {
public:
    int dollars;
    int cents;

    Money(int d = 0, int c = 0) : dollars(d), cents(c) {}

    void Set(int d, int c) {
        dollars = d;
        cents = c;
    }

    int TotalCents() const {
        return dollars * 100 + cents;
    }


    static Money CalculateChange(const Money& itemPrice, const Money&
payment) {
        int totalPaidCents = payment.TotalCents();
        int totalPriceCents = itemPrice.TotalCents();
        Money change;

        if (totalPaidCents < totalPriceCents) {
            std::cout << "Error: Insufficient payment." << std::endl;
            change.Set(0, 0);
        } else {
            int changeCents = totalPaidCents - totalPriceCents;
            change.dollars = changeCents / 100;
            change.cents = changeCents % 100;
        }

        return change;
    }
};

// ################ QUESTION 2
```

```
#################################################################
class Date {
public:
    int month;
    int day;
    int year;

    // Constructor to initialize a Date object
    Date(int m, int d, int y) : month(m), day(d), year(y) {}

    // Helper method to check if a year is a leap year
    bool isLeapYear(int year) {
        return ((year % 4 == 0 && year % 100 != 0) || (year % 400 ==
0));
    }

    // Helper method to get the number of days in a month
    int daysInMonth(int month, int year) {
        int days[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
        if (month == 2 && isLeapYear(year)) {
            return 29;
        }
        return days[month];
    }

    // Method to advance the date by one day
    void advanceOneDay() {
        day++;
        if (day > daysInMonth(month, year)) {
            day = 1;
            month++;
            if (month > 12) {
                month = 1;
                year++;
            }
        }
    }
};

// ################ QUESTION 3
#################################################################
template <class ItemType>
class Bag {
public:
    std::vector<ItemType> items;
    int itemCount;

    // Default constructor
    Bag() {
```

```
        itemCount = 0;
    }

    // Method to add an item to the bag
    void add(const ItemType& item) {
        items.push_back(item);
        itemCount++;
    }

    // Method to compute the intersection of two bags
    Bag<ItemType> intersection(const Bag<ItemType>& otherBag) const {
        Bag<ItemType> intersectionBag;

        for (const ItemType& item : items) {
            if (otherBag.contains(item)) {
                intersectionBag.add(item);
            }
        }

        return intersectionBag;
    }

    // Method to check if an item is in the bag
    bool contains(const ItemType& item) const {
        for (const ItemType& bagItem : items) {
            if (bagItem == item) {
                return true;
            }
        }
        return false;
    }
};



int main() {
    cout << "Hello, World!" << endl;
    cout << "This is Assignment #1!" << endl;

    // ################ QUESTION 1
##################################################################
    Money itemPrice(25, 50); // Item price is $25.50
    Money payment(30, 75);  // Paid $30.75
    Money change = Money::CalculateChange(itemPrice, payment);
    cout << "Change to be given: $" << change.dollars << "." <<
change.cents << endl;

    // ################ QUESTION 2
##################################################################
    Date myBirthday(2, 2, 1997); // Birthday is Feb 2 1997
```

```
    myBirthday.advanceOneDay();
    cout << "New date: " << myBirthday.month << "/" << myBirthday.day
<< "/" << myBirthday.year << endl;

    // ################ QUESTION 3
################################################################
    Bag<string> bag1;
    Bag<string> bag2;

    bag1.items.push_back("a");
    bag1.items.push_back("b");
    bag1.items.push_back("c");

    bag2.items.push_back("b");
    bag2.items.push_back("b");
    bag2.items.push_back("d");
    bag2.items.push_back("e");

      Bag<string> intersectionResult = bag1.intersection(bag2);

      // Print the elements in the intersection bag
      for (const string& item : intersectionResult.items) {
          cout << item << " ";
      }
      cout << endl;

    return 0;
}

// ################ QUESTION 1
################################################################
// #1. The question asks for specifications for a function that
calculates the change when paying for an item with dollars and cents,
using a "money" class with dollars and cents as arguments. It should
include a statement of purpose, preconditions, postconditions, and a
function prototype.

// Statement of Purpose: Calculate the change to be given when
purchasing an item
// Preconditions: Non negative price and non negative payment
// Postconditions: Non negative change calculated
// Function Prototype: See Above

// ################ QUESTION 2
################################################################
// #2. The question involves defining specifications for a method
within a class called "Date" that advances a given date by one day. It
asks for a statement of purpose, preconditions, description of
arguments, and return value for this method. Additionally, it requests
a C++ implementation of this method, along with the design and
```

specification of any other necessary methods, with comments for future maintenance.

```
// Statement of Purpose: Increment a given date by one day,
considering the month, day, and year representation as integers.
// Preconditions: Non negative date and is also a valid date (no dates
larger than 28/30/31 or months > 12)
// Postconditions: Valid date is incremented by 1 day
// Function Prototype: See Above

// ################ QUESTION 3
#####################################################################
// #3. The question requests the design and specification of a method
named "intersection" for the Abstract Data Type (ADT) bag. This method
is intended to return a new bag containing elements that are common
between the bag on which the method is called and another bag provided
as an argument. The method should not modify the original bags and
should account for potential duplicate elements in the intersection.
Use a temporary bag and a loop to find intersecting elements.
// Specifications: See Above
```