C++ Templated Bag ADT

Data Structures · C++ Templates · OOP · Abstract Interface Design

My Role: Implemented all three set operations (union, intersection, difference), full template class, abstract interface, and menu-driven client program

C++	Templates / OOP	Advanced Data Structures

PROJECT OVERVIEW

What I Built

A fully templated C++ Bag Abstract Data Type (ADT) built on top of a pure virtual interface class. The Bag stores items of any type using C++ templates and supports five operations through a menu-driven client program: display, union, intersection, difference, and removal. The interface (BagInterface2) defines the contract; the Bag class implements it with an array-based internal structure.

The three set operations:union, intersection, and difference, were the core of the project. Each required reasoning about frequency, containment, and element overlap across two bags of string items.

MY FUNCTIONS

1. bagUnion — Merge All Items

Combines every item from both bags into a new bag, preserving duplicates from each:

```
Bag<ItemType> Bag<ItemType>::bagUnion(const Bag<ItemType>& otherBag) const {
    Bag<ItemType> unionBag;
    // Add all items from this bag
    for (auto item : this->toVector())
        unionBag.add(item);
    // Add all items from the other bag
    for (auto item : otherBag.toVector())
        unionBag.add(item);
    return unionBag;
}
```

2. bagIntersection — Minimum Frequency Overlap

Finds shared elements and adds the minimum frequency of each. Such as if "Pens" appears twice in Bag1 and three times in Bag2, the intersection gets two "Pens":

```
Bag<ItemType> Bag<ItemType>::bagIntersection(const Bag<ItemType>& b2) const {
    Bag<ItemType> bgg;
    for (const auto& item : this->toVector()) {
        if (b2.contains(item) && !bgg.contains(item)) {
            int freqBag1 = this->getFrequencyOf(item);
            int freqBag2 = b2.getFrequencyOf(item);
            int minFreq  = std::min(freqBag1, freqBag2);
            for (int i = 0; i < minFreq; ++i)
                bgg.add(item);  // add min occurrences
```

```
        }
    }
    return bgg;
}
```

3. bagDifference — Symmetric Difference

Finds items that exist in one bag but not the other items unique to Bag1 plus items unique to Bag2:

```
Bag<ItemType> Bag<ItemType>::bagDifference(const Bag<ItemType>& b2) const {
    Bag<ItemType> differenceBag;
    // Items in Bag1 not in Bag2
    for (const auto& element : this->toVector())
        if (!b2.contains(element))
            differenceBag.add(element);
    // Items in Bag2 not in Bag1
    for (const auto& element : b2.toVector())
        if (!this->contains(element))
            differenceBag.add(element);
    return differenceBag;
}
```

4. Abstract Interface — BagInterface2

The interface defines the full contract as pure virtual functions; any class inheriting it must implement all operations:

```
template<class ItemType>
class BagInterface2 {
public:
    virtual int  getCurrentSize() const = 0;
    virtual bool isEmpty()        const = 0;
    virtual bool add(const ItemType& newEntry)    = 0;
    virtual bool remove(const ItemType& anEntry)  = 0;
    virtual void clear()                          = 0;
    virtual int  getFrequencyOf(const ItemType&)  const = 0;
    virtual bool contains(const ItemType&)        const = 0;
    virtual std::vector<ItemType> toVector()      const = 0;
    virtual ~BagInterface2() { }
};
```

5. Menu-Driven Client Program

The main program creates two pre-filled string bags and routes user input through a switch statement to the correct operation. Uses dynamic_cast to access Bag-specific methods through the interface pointer:

```
BagInterface2<string>* Bag1 = new Bag<string>;
Bag<string>* castCool = dynamic_cast<Bag<string>*>(Bag1);

switch (menuChoice) {
    case 1: displayBag(*Bag1);                          break;
    case 2: displayBag(castCool->bagUnion(*castKool));  break;
    case 3: displayBag(castCool->bagIntersection(...)); break;
    case 4: displayBag(castCool->bagDifference(...));   break;
}
```

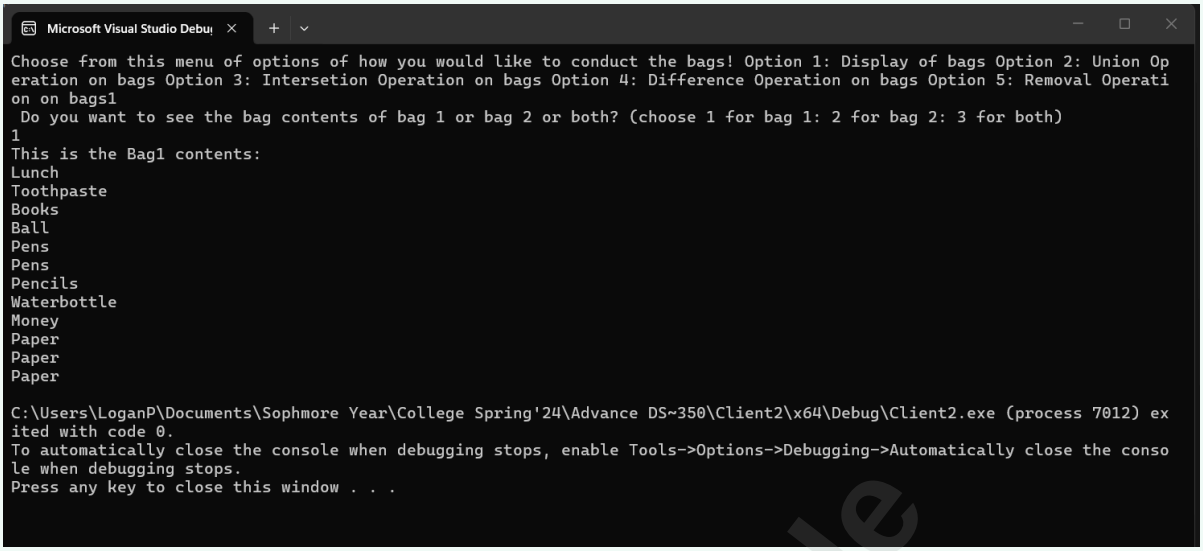
TECHNICAL DETAILS

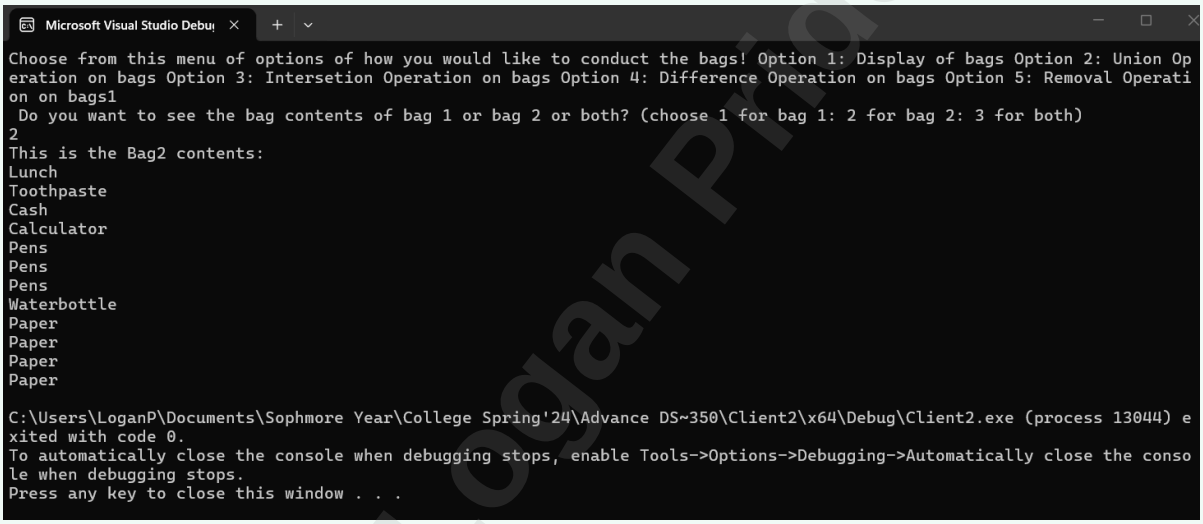
Technical Highlights

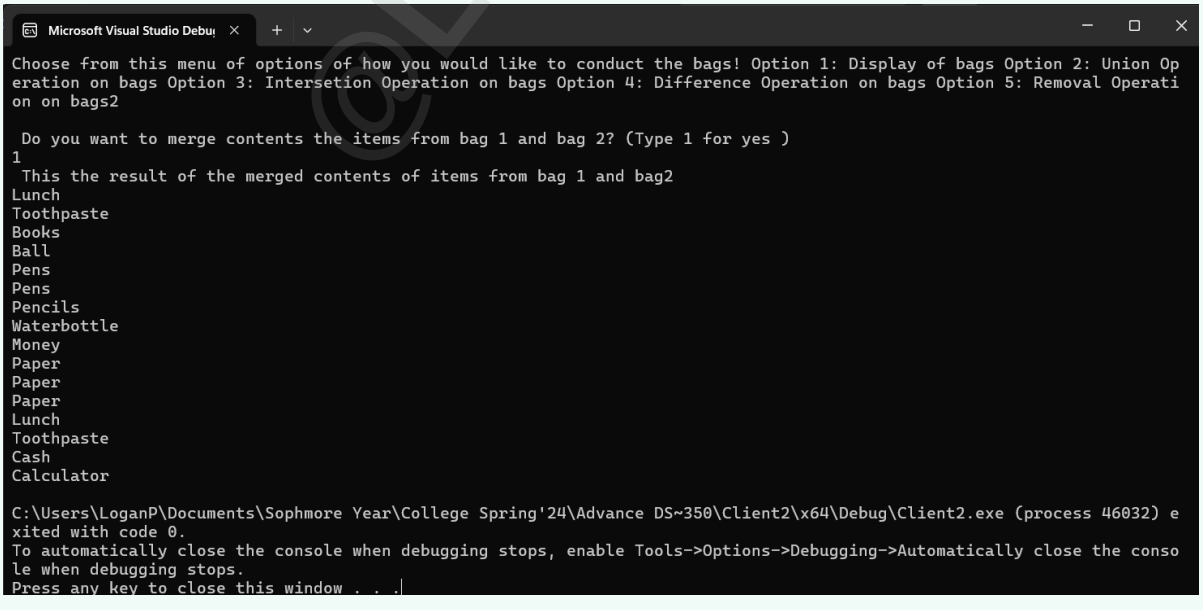
Language	C++ (C++11 or later)
Paradigm	Object-Oriented Programming: inheritance, polymorphism, templates
Interface	Pure virtual abstract base class (BagInterface2)
Implementation	Array-based Bag with fixed capacity (DEFAULT_BAG_SIZE = 16)
Operations	Union, Intersection (min-frequency), Symmetric Difference
Polymorphism	dynamic_cast used to access derived class methods via base pointer
Data Type	Templated works with any ItemType (string, int, etc.)
Key Methods	add, remove, contains, getFrequencyOf, toVector, clear

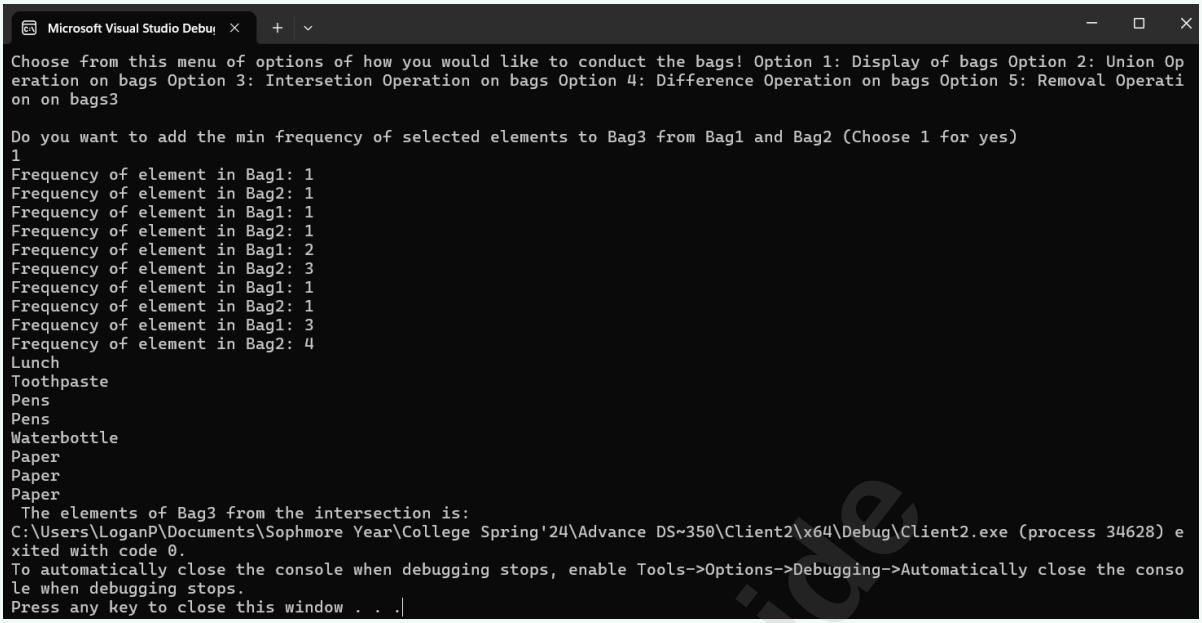
SCREENSHOT

Console Output


Microsoft Visual Studio Debu
Choose from this menu of options of how you would like to conduct the bags! Option 1: Display of bags Option 2: Union Operation on bags Option 3: Intersetion Operation on bags Option 4: Difference Operation on bags Option 5: Removal Operation on bags1
Do you want to see the bag contents of bag 1 or bag 2 or both? (choose 1 for bag 1: 2 for bag 2: 3 for both)
1
This is the Bag1 contents:
Lunch
Toothpaste
Books
Ball
Pens
Pens
Pencils
Waterbottle
Money
Paper
Paper
Paper
C:\Users\LoganP\Documents\Sophmore Year\College Spring'24\Advance DS~350\Client2\x64\Debug\Client2.exe (process 7012) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .


Microsoft Visual Studio Debu
Choose from this menu of options of how you would like to conduct the bags! Option 1: Display of bags Option 2: Union Operation on bags Option 3: Intersetion Operation on bags Option 4: Difference Operation on bags Option 5: Removal Operation on bags1
Do you want to see the bag contents of bag 1 or bag 2 or both? (choose 1 for bag 1: 2 for bag 2: 3 for both)
2
This is the Bag2 contents:
Lunch
Toothpaste
Cash
Calculator
Pens
Pens
Pens
Waterbottle
Paper
Paper
Paper
Paper
C:\Users\LoganP\Documents\Sophmore Year\College Spring'24\Advance DS~350\Client2\x64\Debug\Client2.exe (process 13044) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .


Microsoft Visual Studio Debu
Choose from this menu of options of how you would like to conduct the bags! Option 1: Display of bags Option 2: Union Operation on bags Option 3: Intersetion Operation on bags Option 4: Difference Operation on bags Option 5: Removal Operation on bags2
Do you want to merge contents the items from bag 1 and bag 2? (Type 1 for yes)
1
This the result of the merged contents of items from bag 1 and bag2
Lunch
Toothpaste
Books
Ball
Pens
Pens
Pencils
Waterbottle
Money
Paper
Paper
Paper
Lunch
Toothpaste
Cash
Calculator
C:\Users\LoganP\Documents\Sophmore Year\College Spring'24\Advance DS~350\Client2\x64\Debug\Client2.exe (process 46032) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .



Console output showing menu, bag contents, and set operation results.

TAKEAWAYS

What I Learned

- How to design and implement a C++ abstract interface with pure virtual functions,separating the contract from the implementation.
- C++ template classes writing generic code that works with any data type without rewriting logic.
- Set operation algorithms: union, intersection (with frequency), and symmetric difference all require careful reasoning about containment and duplicates.
- Polymorphism and dynamic_cast using a base interface pointer while still accessing derived-class-specific methods safely.
- Array-based ADT design; managing itemCount, maxItems, and the items array manually without STL containers.