Part C - Class Relationships

Templates

Workshop 3

In this workshop, you design and code a class template and test it on two unrelated classes.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- design and code a class template
- store key-value information in a pair of parallel arrays
- describe to your instructor what you have learned in completing this workshop

SPECIFICATIONS

The source files for this workshop include:

- A header file - `KVList.h` - defines a class template for a list of key-value pairs
- A client file - `w3.cpp` - the program that uses your class template
- An input file - `input.txt` - contains the test data for your solution

KVList Template

Design and code a class template named `KVList` that manages lists of key-value pairs. The classes generated by your template contain two parallel arrays of dimension `N` - a key array of type `K` and a value array of type `V`. `K`, `V` and `N` are template parameters. The client application specifies the arguments for these parameters.

Save your template definition in a header file named `KVList.h`.

Your template design includes the following member functions:

- `KVList()` - default constructor - creates an object in a safe empty state
- `size_t size() const` - a query that returns the number of entires in the key-value list
- `const K& key(int i) const` - a query that returns an unmodifiable reference to the key of element `i` in the list; if `i` is out of bounds, returns a reference to the key for the first element
- `const V& value(int i) const` - a query that returns an unmodifiable reference to the value of element `i` in the list; if `i` is out of bounds, returns a reference to the value for the first element
- `KVList& add(const K&, const V&)` - adds a new element to the list if room exists and returns a reference to the current object, does nothing if no room exists
- `int find(const K& k) const` - returns the index of the first element in the list with a key equal to `k` - defaults to 0
- `KVList& replace(int i, const K& k, const V& v)` - replaces element `i` in the list with the key and value received and returns a reference to the current object

Client Program

The client program that uses your template definition accepts and displays data for

1. an inventory of product-price pairs
2. a glossary of acronym-definition pairs

For each list, this client program sets the maximum number of entries to 5.

This program can accept data either from the console or from an input file. The program is initialized to accept input data from the console. To receive data from a file, comment out the **`#define CONSOLE`** macro. The name of the file is specified on the command line. (The source of the input data is determined by the presence or absence of the pre-processor macro **`CONSOLE`**. If this macro has been defined, the client program accepts input from the console. If this macro has not been defined, the client program accepts input from the file named on the command line.)

```
// Workshop 3 - Templates
// w3.cpp
// Chris Szalwinski
// 2018.05.17

#include <fstream>
#include <iostream>
#include <iomanip>
#include <string>
#include "KVList.h"
#define CONSOLE
#ifndef CONSOLE
#define SOURCE input
#else
#define SOURCE std::cin
#endif

// display list of key-value pairs
//
template <typename K, typename V, int N>
void display(const std::string& msg, const KVList<K, V, N>& list, int w) {
    std::cout << msg;
    for (size_t i = 0; i < list.size(); i++)
        std::cout << std::setw(w) << list.key(i)
         << " : " << list.value(i) << std::endl;
}

// prompt for user input
//
void prompt(const char* str) {
   #ifndef CONSOLE
   std::cout << ". ";
   #else
   std::cout << str;
   #endif
}

int main(int argc, char** argv) {
    std::cout << "Command Line : ";
    for (int i = 0; i < argc; i++) {
        std::cout << argv[i] << ' ';
    }
    std::cout << std::endl;

    #ifndef CONSOLE
    if (argc == 1) {
        std::cerr << "\n*** Insufficient number of arguments ***\n";
        std::cerr << "Usage: " << argv[0] << " fileName \n";
        return 1;
    }
    else if (argc != 2) {
        std::cerr << "\n*** Too many arguments ***\n";
        std::cerr << "Usage: " << argv[0] << " fileName \n";
        return 2;
    }
    std::ifstream input(argv[1]);
    if (!input) {
        std::cerr << "*** Failed to open file " << argv[1] << " successfully ***\n";
        return 3;
    }
    #endif

    bool keepreading;
    std::cout << std::fixed << std::setprecision(2);
```

```
        std::cout << "\nInventory\n=========\n";
        KVList <std::string, double, 5> inventory;
        std::string str;
        double price;

        keepreading = true;
        do {
            prompt("Product : ");
            getline(SOURCE, str);
            if (str.compare("quit") == 0) {
                keepreading = false;
            }
            else {
                prompt("Price : ");
                SOURCE >> price;
                SOURCE.ignore();
                inventory.add(str, price);
            }
        } while (keepreading);
        display("\nPrice List\n-----------\n", inventory, 13);

        std::cout << "\nCorrections\n-----------\n";
        keepreading = true;
        do {
           prompt("Product : ");
            getline(SOURCE, str);
           if (str.compare("quit") == 0) {
                keepreading = false;
            }
            else {
                int i = inventory.find(str);
                if (i != -1) {
                    prompt("Price : ");
                    SOURCE >> price;
                    SOURCE.ignore();
                    inventory.replace(i, str, price);
                }
            }
        } while (keepreading);
        display("\nPrice List\n-----------\n", inventory, 13);

        std::cout << "\nGlossary\n========\n";
        KVList <std::string, std::string, 5> glossary;
        std::string key, definition;

        keepreading = true;
        do {
            prompt("Key : ");
            getline(SOURCE, key);
            if (key.compare("quit") == 0) {
                keepreading = false;
            }
            else {
                prompt("Definition : ");
                getline(SOURCE, definition);
                glossary.add(key, definition);
            }
        } while (keepreading);
        display("\nEntries\n-------\n", glossary, 5);
    }
```

For the console input listed below, the client program using your template produces the following output. A copy of the file containing identical input is available here.

```
    Inventory                  Glossary
    =========                  ========
    Product : Pizza            Key : CPU
    Price : 4.49               Definition : central processing unit
    Product : Pierogi          Key : ALU
    Price : 2.56               Definition : arithmetic logic unit
    Product : Potato Chips     Key : quit
    Price : 2.29
    Product : Black Tea        Entries
```

```
Price : 4.49
Product : Green Tea
Price : 3.46
Product : Fruit Tea
Price : 2.29
Product : quit

Price List
----------
       Pizza : 4.49
     Pierogi : 2.56
 Potato Chips : 2.29
   Black Tea : 4.49
   Green Tea : 3.46

Corrections
-----------
Product : Black Tea
Price : 5.29
Product : quit

Price List
----------
       Pizza : 4.49
     Pierogi : 2.56
 Potato Chips : 2.29
   Black Tea : 5.29
   Green Tea : 3.46
```

```
-------
 CPU : central processing unit
 ALU : arithmetic logic unit
```

Note that the input data is only stored for the first N items, which is the size specified in the definitions of **`inventory`** and **`glossary`** (5). All additional input is discarded.

SUBMISSION

Follow your professor's submission instructions.

Unless otherwise stated by your instructor, your submission should include the following components:

1. source code for your **`w3.cpp`** file
2. source code for your **`KVList.h`** file
3. a text file named **`reflect.txt`** that includes:
   - the corrected answers to the latest quiz that you received
   - a description of what you learned in completing this workshop