**Templates**

The following example is used to demonstrate templated classes. This is a Dictionary class. There is a term and definition. The example in main() will use the Dictionary to keep track of the number of students who have a particular grade.

You may run this program in repl.it by clicking the Form button, or download the files by selecting the three dots next to the Files menu <https://replit.com/@KimMurphy/Dictionary#main.cpp>

**Templated Class**

1. Observe the Dictionary.h file. Find the class template above the class. How many template parameters are there? **3**
2. Observe the private variables. What data type is used for the terms? What is used for the definitions? **The data type T, which is set to char, is used for Terms and D, which is set to int, for definition**
3. Now let’s look at the constructor. Notice the scope resolution operator in constructor implementation. Why does the Dictionary need to have the templated parameters? **Because a dictionary is just a key value lookup table, and these could be anything from int and string to bool and int to double to float.**
4. What is the variable size set to and why? **It is set to MAX\_SIZE because that is the initialized size of the arrays and thus can only be that big.**
5. Observe the Add method. There are two parameters for term and definition. What is the data type of those parameters? **The term has data type T and definition data type D**
6. Observe the variables for the two arrays. What position of the array are the parameters being put into the array. **The parameters are being put into the furthest/last open position as long as it is not above the MAX\_SIZE**
7. Observe the IndexOfTerm() function. Note the return type. Why do you suppose it is not one of the typed variables? **Index values are always integers because it is the position in a list. You cannot of a “Think” position of a list, only a 4th position or 8th position.**
8. Why do you suppose it is iterating to size, instead of MAX\_SIZE? **We only need to go through the filled spots in the dictionary, it would be wasteful to go through all the unfilled positions, which would be going all the way to MAX\_SIZE**
9. Look at the GetDefinition() method. How is the function calculating the position of the definition? **It is calling the IndexOfTerm function to get the index of the key value, or the term in this case, and going to the same equivalent position in the definition array**
10. Your turn, complete the code for the UpdateDefinition() function. Make the function search for the term, and change the definition to the parameter newDefinition.

**Main() method**

1. Navigate to the main() method. Find where the Dictionary is being defined. What types of variables are going to be used for the term and definition in this example? **Char for the term and int for the definition, the max size is 5**
2. What does the 5 represent in the class\_grades variable? **The MAX\_SIZE**
3. Note the For Loop. What is the current grade set to? **The current student in the iterations equivalent array index value.**
4. Look at the If statement. What is the condition to determine that the grade is not in the Dictionary? **If the value of the IndexOfTerm is less than 0, because it will return -1 if the term does not exist**
5. What definition is getting set for that grade? **The current\_grade value found in the grades array**
6. Follow the comments to complete the else so that if it finds the grade in the dictionary, it will add one to it.
7. Run the program and make sure that there are at least 32 grades total. **There are.**

Dictionary.h:

#include <iostream>

template <class T, class D, int MAX\_SIZE>

class Dictionary;

template <class T, class D, int MAX\_SIZE>

std::ostream &operator<<(std::ostream &, const Dictionary<T, D, MAX\_SIZE> &d);

template <class T, class D, int MAX\_SIZE>

class Dictionary {

private:

T terms[MAX\_SIZE];

D definitions[MAX\_SIZE];

int size;

public:

Dictionary();

void Add(T term, D definition);

int IndexOfTerm(T term) const;

D GetDefinition(T term) const;

void UpdateDefinition(T term, D definition);

friend std::ostream &operator<< <>(std::ostream &,

const Dictionary<T, D, MAX\_SIZE> &d);

};

/\*\*

\* Creates an empty dictionary

\*/

template <class T, class D, int MAX\_SIZE>

Dictionary<T, D, MAX\_SIZE>::Dictionary() {

size = 0;

}

/\*\*

\* Adds a term and definition to the dictionary

\* @param term: The term to be added

\* @param definition: The definition to be paired with the term

\*/

template <class T, class D, int MAX\_SIZE>

void Dictionary<T, D, MAX\_SIZE>::Add(T term, D definition) {

if (size < MAX\_SIZE) {

terms[size] = term;

definitions[size] = definition;

size++;

}

}

/\*\*

\* Searches the dictionary for the term

\* @param term: The term that the dictionary is searching for

\* @return the position of the term or -1 if it is not found

\*/

template <class T, class D, int MAX\_SIZE>

int Dictionary<T, D, MAX\_SIZE>::IndexOfTerm(T term) const {

for (int i = 0; i < size; i++) {

if (term == terms[i]) {

return i;

}

}

return -1;

}

/\*\*

\* Finds the definition to match the given term

\* @param term: The term that the dictionary is searching for

\* @return the corresponding definition

\*/

template <class T, class D, int MAX\_SIZE>

D Dictionary<T, D, MAX\_SIZE>::GetDefinition(T term) const {

int pos = IndexOfTerm(term);

if (pos >= 0) {

return definitions[pos];

}

throw std::out\_of\_range("Term not found");

}

/\*\*

\* Updates the term with a new definition

\* @param term: The term that the dictionary is searching for

\*/

template <class T, class D, int MAX\_SIZE>

void Dictionary<T, D, MAX\_SIZE>::UpdateDefinition(T term, D newDefinition) {

int pos = IndexOfTerm(term);

if (pos >= 0) {

definitions[pos] = newDefinition;

return;

}

throw std::out\_of\_range("Term not found");

}

/\*\*

\* Displays the term and definition pairs

\*/

template <class T, class D, int MAX\_SIZE>

std::ostream &operator<<(std::ostream &out,

const Dictionary<T, D, MAX\_SIZE> &d) {

for (int i = 0; i < d.size; i++) {

out << d.terms[i] << "\t" << d.definitions[i] << std::endl;

}

return out;

}

Main.cpp

#include "Dictionary.h"

#include <iostream>

/\*

\* @file main.cpp

\* @author Kim Murphy

\* @brief Use the Templated Dictionary to determine the number of each of the

\* grades in the class.

\* @date 1/2/2023

\*/

using namespace std;

int main() {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Use the Templated Dictionary to determine the number of each of the grades in

the class.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Number of students

const int NUM\_STUDENTS = 32;

// Student grades

char grades[NUM\_STUDENTS] = {'A', 'A', 'B', 'A', 'C', 'A', 'C', 'A',

'B', 'A', 'D', 'F', 'A', 'A', 'B', 'C',

'B', 'A', 'C', 'A', 'A', 'B', 'B', 'B',

'A', 'B', 'D', 'D', 'A', 'A', 'A', 'A'};

// Define the dictionary

Dictionary<char, int, 5> class\_grades;

// Iterate through each grade

for (int i = 0; i < NUM\_STUDENTS; i++) {

char current\_grade = grades[i];

if (class\_grades.IndexOfTerm(grades[i]) < 0) {

class\_grades.Add(current\_grade, 1);

} else {

// Update the else statement

// find the definition of the given grade in the class\_grades dictionary

// Add one to the definition

// update the class\_grades with the new definition of the grade

class\_grades.UpdateDefinition(current\_grade, class\_grades.GetDefinition(current\_grade) + 1);

}

}

cout << class\_grades;

}