|  |
| --- |
| **Российский университет транспорта (МИИТ) Институт транспортной техники и систем управления Кафедра «Управление и защита информации»** |
| **Отчет**  **по практическому заданию по теме «Структуры данных»**  **по дисциплине «Системы управления базами данных»** |
| Выполнили:  Студенты группы ТКИ-441  Чекан Ф.С. Комаричев Г.Ю.  Проверил:  Доцент кафедры УиЗи, к.т.н., с.н.с.  Васильева М.А. |
| Москва 2023 |

Оглавление

[Задание 3](#_bookmark0)

1. [UML диаграмма 4](#_bookmark1)
2. [Текст программы на языке C++ 4](#_bookmark2)
   1. [Код файла HashMap.h 4](#_bookmark3)
   2. [Код файла HashMap.cpp 6](#_bookmark4)
   3. [Код файлa main.cpp 8](#_bookmark5)
   4. [Код файлa HashMapTest.cpp 9](#_bookmark6)
3. [Результат работы программы 10](#_bookmark7)

[Заключение 12](#_bookmark8)

# Задание

Разработать структуру данных на языке программирования С++ в ООП парадигме. В нашем случае структура данных – это словарь (map), основные операции которого будут добавление элемента, обновления значения по ключу, получение значения по ключу и удаление пары (ключ, значения). Структура способна обрабатывать любые типы данных (template).

# UML диаграмма

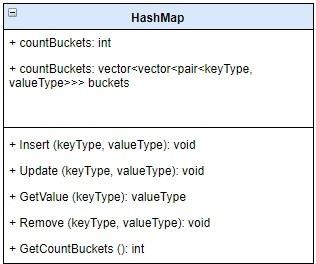


Рисунок 1 – UML диаграмма классов HashMap

# Текст программы на языке C++

# Код файла HashMap.h

#include <vector> #include <iostream>

using namespace std;

/\*\*

\* @brief Класс HashMap.

* @tparam keyType тип данных ключа.
* @tparam valueType тип данных значения.

\*/

template <typename keyType, typename valueType> class HashMap {

public:

/\*\*

* @brief Конструктор по умолчанию.

\*/

HashMap();

/\*\*

* @brief Метод вставки пары (Ключ, значение).
* @param key Ключ.
* @param value значение.

\*/

void Insert(const keyType &key, const valueType &value);

/\*\*

* @brief Метод обновление значения по ключу.
* @param key Ключ.
* @param value Новое значение.

\*/

void Update(const keyType &key, const valueType &value);

/\*\*

* @brief Метод получения значения по ключу.
* @param key Ключ.
* @return �-начение.

\*/

valueType GetValue(const keyType &key);

/\*\*

* @brief Метод удаления пары (ключ, значения) по ключу.
* @param key Ключ.

\*/

void Remove(const keyType &key);

/\*\*

* @brief Метод получения размера HashMap.
* @return размер HashMap.

\*/

int GetCountBuckets();

/\*\*

* @brief размер HashMap.

\*/

int countBuckets;

/\*\*

* @brief Массив пар (ключ, значение).

\*/

vector<vector<pair<keyType, valueType>>> buckets;

};

# Код файла HashMap.cpp

#include <vector> #include "HashMap.h"

template <typename keyType, typename valueType> HashMap<keyType, valueType>:: HashMap() {

countBuckets = 32; buckets.resize(countBuckets);

}

template <typename keyType, typename valueType>

void HashMap<keyType, valueType>::Insert(const keyType &key, const valueType &value) { int hashIndex = hash<keyType>{}(key) % countBuckets;

for (auto &pair: buckets[hashIndex]) { if (pair.first == key) {

throw out\_of\_range("Pair already exists, choose update to set new value.

\n");

}

}

buckets[hashIndex].push\_back(make\_pair(key, value));

}

template <typename keyType, typename valueType>

valueType HashMap<keyType, valueType>::GetValue(const keyType &key) { int hashIndex = hash<keyType>{}(key) % countBuckets;

if (buckets[hashIndex].empty())

{

throw out\_of\_range("Pair is not found");

}

for (auto &pair : buckets[hashIndex]) { if (pair.first == key) {

return pair.second;

}

}

}

template <typename keyType, typename valueType>

void HashMap<keyType, valueType>::Update(const keyType &key, const valueType &value) {

int hashIndex = hash<keyType>{}(key) % countBuckets; if (buckets[hashIndex].empty())

{

throw out\_of\_range("Pair is not found");

}

for (auto &pair: buckets[hashIndex]) { if (pair.first == key) {

pair.second = value; return;

}

}

}

template <typename keyType, typename valueType>

void HashMap<keyType, valueType>::Remove(const keyType &key) { int hashIndex = hash<keyType>{}(key) % countBuckets;

for (auto it = buckets[hashIndex].begin(); it != buckets[hashIndex].end(); ++it){ if (it->first == key) {

buckets[hashIndex].erase(it); return;

}

}

}

template<typename keyType, typename valueType>

int HashMap<keyType, valueType>::GetCountBuckets() { return this->countBuckets;

}

# Код файлa main.cpp

#include <iostream> #include <vector> #include "HashMap.h" #include "HashMap.cpp"

using namespace std; int main() {

HashMap<string, int> car;

car.Insert("Kia", 123);

car.Insert("BMW", 420);

car.Insert("Tesla", 1020);

cout << "Value of Kia: " << car.GetValue("Kia") << endl; cout << "Value of BMW: " << car.GetValue("BMW") << endl; cout << "Value of Tesla: " << car.GetValue("Tesla") << endl;

car.Insert("Ford", 100);

cout << "Value of Ford: " << car.GetValue("Ford") << endl; cout << "Remove value of BMW" << endl;

car.Remove("BMW");

cout << "Remove successful" << endl;

cout << "Updating value of Kia: " << endl; car.Update("Kia", 130);

cout << "Update successful" << endl;

cout << "Value of Kia after update: " << car.GetValue("Kia") << endl; cout << "Value of BMW after remove" << car.GetValue("BMW") << endl;

return 0;

}

# Код файлa HashMapTest.cpp

#include <gtest/gtest.h>

#include "../Task2/Common/HashMap.h" #include "../Task2/Common/HashMap.cpp"

TEST(HashMapTests, InsertTest) { HashMap<int, int> test; test.Insert(1, 2);

EXPECT\_EQ(test.GetValue(1), 2);

}

TEST(HashMapTests, GetValueTest) { HashMap<int, int> test; test.Insert(1, 2);

int testValue = test.GetValue(1); EXPECT\_EQ(testValue, 2);

}

TEST(HashMapTests, UpdateTest) { HashMap<int, int> test; test.Insert(1, 2);

test.Update(1, 5);

EXPECT\_EQ(test.GetValue(1), 5);

}

TEST(HashMapTests, RemoveTest) { HashMap<int, int> test; test.Insert(1, 2);

test.Insert(3, 4); test.Remove(1);

EXPECT\_THROW(test.GetValue(1), out\_of\_range);

}

TEST(HashMapTests, CreateTest) { HashMap<int, int> test; EXPECT\_EQ(test.GetCountBuckets(), 32);

}

TEST(HashMapTests, CopyConstructorTest) { HashMap<int, int> test; test.Insert(1, 2);

HashMap<int, int> copy = HashMap<int, int>(test);

EXPECT\_EQ(copy.GetValue(1), 2);

}

TEST(HashMapTests, OperatorTest) { HashMap<int, int> test; test.Insert(1, 2);

HashMap<int, int> test2 = test;

EXPECT\_EQ(test2.GetValue(1), 2);

}

# Результат работы программы

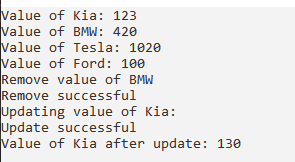


Рисунок 2 – Результат отладки программы



Рисунок 3 – Результат выполнения тестов для HashMap.h

# Заключение

В результате выполнения практического задания была разработана структура данных - словарь (map). Для обеспечения обработки различных типов данных была использована техника шаблонов, что позволило использовать данную структуру с любым типом данных.