ФГАОУВО «Южно-Уральский государственный университет (НИУ)»

Институт естественных и точных наук

Кафедра «Прикладная математика и программирование»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

по дисциплине «Объектно-ориентированное программирование»

Автор работы

студент группы ЕТ-211

_____ М.В.Савонин

_____2022 г.

Работа зачтена с оценкой

_____

_____А.К.Демидов

_____2022 г.

Челябинск, 2022

1 Постановка задачи

I. Определить класс-шаблон с использованием динамического распределения памяти согласно варианту и необходимые конструкторы и операции, включая конструктор копий, операция присваивания и если указано операцию индексации. При выходе за границу, переполнении и т.п. вызвать исключительную ситуацию (определить собственные классы) для информирования программы, вызвавшей метод.

12. класс ассоциативный массив, в котором можно осуществлять поиск значений заданного типа по ключу другого заданного типа, для которого определена проверка на равенство.
операция доступа по ключу val=obj[key], добавление, удаление значения, соответствующее ключу.

При определении друзей класса-шаблона использовать следующий пример
II. Реализовать main с тестами
(создание объектов и выполнение действий с ними, в т.ч. действие, приводящее к возникновению исключительной ситуации, которую необходимо перехватить)

2 Описание интерфейса классов

```
   struct maperror { // базовый класс для ошибок
      virtual ~maperror() {} // деструктор
      virtual const char *what() const=0; // сообщение для
печати
   };

   struct mapempty: maperror {
      const char *what() const {return "массив пуст";} //
сообщение для печати
   };

   struct mapnotfound: maperror {
      const char *what() const {return "Элемент не найден";}
// сообщение для печати
   };

   struct maphave: maperror {
      const char *what() const {return "Ключ уже
используется";} // сообщение для печати
   };
```

```
template <typename K, typename V>
class Map {
   K *key; // указатель на ключи в массиве
   V *value; // указатель на значения в массиве
   int col; // текущее количество
public:
   Map(): col(0), key(new K[0]), value(new V[0]) {} //
конструктор
   Map(const Map<K, V> &);// конструктор копий
   ~Map() throw() {delete []key; delete []value;}//
деструктор
   Map<K, V> &operator=(const Map<K, V> &);// операция
присваивания
   V &operator[](K); // поиск по ключу
   V operator[](K) const; // поиск по ключу
   Map<K, V> &del(K); // удаление по ключу
   Map<K, V> &add(K, V); // добавление по ключу
   bool operator==(Map<K, V>&);  // сравнение массивов
};
```

3 Описание тестов для проверки классов

```
int main()
{
   Map<char, int> obj;
   cout<<"Тест 1. Добавление\n";
   try {
      for (char i = 'a'; i < 'a'+10; i++) {
         obj.add(i, int(i-'a'));
      }
      obj.add('a', int('a'));
   }
   catch (maperror &e) {
      cout<<e.what();
   }

   Map<char, int> obj2;
   cout<<"\nТест 2. Поиск по ключу\n";
   try {
      for (char i = 'a'; i < 'a'+11; i++) {
         cout << obj[i];
         obj2.add(i, obj[i]);
      }
   }
   catch (maperror &e) {
      cout<<e.what();
```

```
    }

    cout<<"\nТест 3. Сравнение массивов\n";
    try {
       cout << (obj == obj2) << endl;
       obj2['a'] = 10;
       cout << (obj == obj2) << endl;
       obj2['a'] = 0;
       obj2.add('k', 10);
       cout << (obj == obj2) << endl;
       obj2.del('k');
       obj2.del('a');
       cout << (obj == obj2) << endl;
    }
    catch (maperror &e) {
       cout<<e.what();
    }

    cout<<"\nТест 4. Удаление по ключу\n";
    try {
       for (char i = 'a'; i < 'a'+11; i++) {
          obj.del(i);
       }
    }
    catch (maperror &e) {
       cout<<e.what();
    }

    return 0;
}
```

Полученные результаты

Тест 1. Добавление
Ключ уже используется
Тест 2. Поиск по ключу
0123456789Элемент не найден
Тест 3. Сравнение массивов
1
0
0
0

Тест 4. Удаление по ключу
массив пуст

4 Листинг реализации класса

```
   template <typename K, typename V>
   Map<K, V>::Map(const Map <K, V> &m): key(new K[m.col]),
value(new V[m.col]), col(m.col) {
      for (int i=0; i<col; i++)
      {
         key[i] = m.key[i];
         value[i] = m.value[i];
      }
   }

   template <typename K, typename V>
   Map<K, V> &Map<K, V>::operator=(const Map<K, V> &m)
   {
      delete []key;
      delete []value;
      col = m.col;
      key = new K[col];
      value = new V[col];
      for (int i=0; i<col; i++)
      {
         key[i] = m.key[i];
         value[i] = m.value[i];
      }
      return *this;
   }

   template <typename K, typename V>
   V &Map<K, V>::operator[](K k)
   {
      for(int i = 0; i < col; i++)
      {
         if(key[i] == k) return value[i];
      }
      throw mapnotfound();
   }

   template <typename K, typename V>
   V Map<K, V>::operator[](K k) const
   {
      for(int i = 0; i < col; i++)
      {
         if(key[i] == k) return value[i];
      }
      throw mapnotfound();
   }
```

```
template <typename K, typename V>
Map<K, V> &Map<K, V>::del(K k)
{
   if(col == 0) throw mapempty();
   for(int i = 0; i < col; i++)
   {
      if(key[i] == k)
      {
         for(int j = i; j < col-1; j++)
         {
            key[j] = key[j+1];
            value[j] = value[j+1];
         }
         col--;
         return *this;
      }
   }
   throw mapnotfound();
}

template <typename K, typename V>
Map<K, V> &Map<K, V>::add(K k, V v)
{
   K *keycop = new K[col];
   V *valuecop = new V[col];
   for(int i = 0; i < col; i++)
   {
      keycop[i] = key[i];
      valuecop[i] = value[i];
      if(key[i] == k) throw maphave();
   }
   delete []key;
   delete []value;
   key = new K[col+1];
   value = new V[col+1];
   for(int i = 0; i < col; i++)
   {
      key[i] = keycop[i];
      value[i] = valuecop[i];
   }
   key[col] = k;
   value[col] = v;
   col++;
   delete []keycop;
   delete []valuecop;
   return *this;
```

```
}

template <typename K, typename V>
bool Map<K, V>::operator==(Map<K, V>& m)
{
   if(m.col != col) return false;
   try {
      for(int i = 0; i < col; i++)
      {
         if(m[key[i]] != value[i]) return false;
      }
   }
   catch (maperror &e) {
      return false;
   }
   return true;
}
```