DDC23 Справка

Контейнер данных для использования в разработке приложений, требующих литеральные идентификаторы, типа словарей

Проектная группа	--		
Заказчик	Самостоятельный проект		
Проект	Контейнер данных для использования в разработке приложений, требующих литеральные идентификаторы, типа словарей		
Руководитель проекта	Артур Мангус		
Участники проекта	Самостоятельный проект		
Документ	Справка	Количество страниц	9
Автор документа	Артур Мангус		
Создан	14.01.2023		
Последнее изменение	26.01.2023		
Статус обработки		В обработке	
		Представлен разработчикам и заказчику	
		Одобрен	
	X	Закрыт	

История документа

Версия	Дата	Автор изменения	Описание / замечание
0.1	16.01.2023	Артур Мангус	Старт проекта
0.2	26.01.2023	Артур Мангус	Редактирование документа

Dictionary

dictionary - это сортированный ассоциативный контейнер, который содержит пары «ключ-значение» с уникальными ключами. Операции поиска, удаления и вставки имеют линеарную сложность O(N). ***dictionary*** реализуется в виде *n-арного дерева*.

dictionary удовлетворяет требованиям *Container*, *AssociativeContainer*.

Container - это объект, используемый для хранения других объектов и заботящийся об управлении памятью, используемой объектами, которые он содержит. *AssociativeContainer* - это упорядоченный контейнер, который обеспечивает быстрый поиск объектов на основе ключей.

Параметры шаблона

Типы членов

Member type	Definition
string	Key
mapped_type	T
vector<string>	Key List
map<string, T*>	Key T Map
set<string>	Collector

Запросы

```
bool empty();
bool find( const string& key );
vector<string> keys();
T value( const string& key );
map<string, T> values();
```

Модификаторы

```
void clear();
bool insert( const string& key, const T& value );
bool remove( const string& key );
bool replace( const string& keyOld, const string& keyNew );
bool setValue( const string& key, const T& value );
```

Операторы

```
const T operator[](const string& key ) const;
```

ОПИСАНИЕ

Класс **dictionary** - это шаблонный класс, который предоставляет словарь на основе N-арного дерева. **dictionary** хранит пары (ключ, значение) и обеспечивает быстрый поиск значения, связанного с ключом. **dictionary** обеспечивает быстрый поиск ключей, сравнимый с хэш-таблицей.

Ниже приводится пример **dictionary** с ключами **string** и значениями int:

```
dictionary<int> dict;

dict.insert( "A",  1 );
dict.insert( "ACT",  2 );
dict.insert( "ACTION",  3 );
dict.insert( "BAD",  10 );
dict.insert( "BOARD",  11 );

if( dict.find( "ACTION" ) )
{
    cout << dict.value( "ACTION" );
}

/***********************************
Результат вывода:
>>  3
***********************************/
```

ЗАПРОСЫ

bool empty();
Функция возвращает ***true*** если контейнер содержит ключи, или ***false*** если контейнер пуст.

bool find(*const* string& key);
Функция возвращает true если в контейнере содержится ключ, переданный в аргумент функции, или false если ключ не найден.

vector<string> **keys**();
Функция возвращает контейнер типа vector содержащий все найденные ключи.

```
vector<string> keys = dict.keys();

cout << keys[0];

/***********************************
Результат вывода:
>> "A"
***********************************/
```

T **value**(*const* string& key) *const*;
Функция возвращает данные ассоциированные с ключем, переданным в качестве аргумента, если такой ключ найден в контейнере, или возвращает нулевое значение декларированного

типа. Если декларированный тип является указателем, то функция вернет нулевой указатель. Однако если тип данных является примитивным, то возвращаемый результат будет соответствовать значению, установленному для этого типа по умолчанию, используемым компилятором. Таким образом, валидация значения должна осуществляться после получения результата работы функции. Во избежание непредсказуемых результатов предлагается предварительно использовать функцию **bool find**(*const* string& key).

map<string, T> **values**();
Функция возвращает контейнер типа map содержащий все найденные ключи и ассоциированные с ними данные.

```
    map<string, int> keyDataMap = dict.values();

    for( const auto& keyDataPair : keyDataMap )
    {
        cout << keyDataPair.first << "\t" << keyDataPair.second << endl;
    }

    /************************************
    Результат вывода:
    "A       1"
    "ACT     2"
    "ACTION  3"
    "BAD     10"
    "BOARD   11"
    "BODY    12"
  ************************************/
```

МОДИФИКАТОРЫ

void clear();
Функция очищает контейнер от всех ключей и ассоциированных с ними значений.

bool insert(*const* string& key , *const* T& value);
Функция принимает два аргумента – первый аргумент для ключа, и второй ассоциированное с ключом значение. Если ключ и значение отсутствуют в контейнере, то они будут добавлены в контейнер и функция вернет значение ***true***. Если ключ уже существует в контейнере, функция вернет ***false***, что предупреждает случайную перезапись значения. В случае когда значение для ключа должно быть перезаписано, необходимо использовать функцию **bool setValue**(*const* string& key, *const* T& value).

bool remove(*const* string& key);
Функция принимает в качестве аргумента значение удаляемого ключа. Если ключ найден в контейнере, то и сам ключ, и ассоциированные с ним данные будут удалены из контейнера, и функция вернет ***true.*** Если ключ не найден, функция вернет ***false***.

bool replace(*const* string& keyOld, *const* string& keyNew)
Функция принимает два аргумента – первый аргумент для ключа, который должен быть заменен в контейнеры на новый, представленный вторым аргументом функции. Если ключ найден в контейнере, то этот ключ будет заменен на новый, а ассоциированные с ним данные будут сохранены под новым значением ключа, и функция вернет значение ***true***. Если ключ, который необходимо заменить, не найден, то функция вернет ***false***.

bool setValue(***const*** **string**& key, ***const*** T& value);
Функция принимает два аргумента – первый аргумент для ключа, и второй ассоциированное с ключом значение. Если ключ найден в контейнере, то значение для ключа будет заменено на новое, переданное в аргументе этой функции, и функция вернет ***true***. Если ключ и значение отсутствуют в контейнере, то функция вернет ***false***.