Program implements a dictionary structure with double hashing as a class template. In this file, there are three structures. There is also a global variable primes referring to 18 element array filled with prime numbers. Every two sequent primes p_1, p_2 satisfy the following condition: p_2 is the next prime number after $10p_1$. In this file, there are two constants defined: DELETED and NOT_DELETED, which are aliases to flags that mean whether an element at a given index is deleted. There is also a definition of size_t type as unsigned long int.

1 uninitializedArraySentinel class

Is a class that manages states of indexes of an uninitialized array. It only provides information about the state of an index.Should be used along with an uninitialized array.
memory complexity: $O(n)$
constructor

```
uninitializedArraySentinel(size_t size);
```

size - the size of a described array.

Creates an object with a length of size also containing uninitialized arrays. Describes a completely uninitialized array with the length size.
t ime complexity: $O(1)$

initialized

```
 bool initialized(size_t index);
```

index - array index.
Returns true if given index was already initialized. False otherwise.
time complexity: $O(1)$

initialize

```
 void initialize(size_t index);
```

index - array index.
regardless what was the state of the index, marks the index as initialized.
time complexity: $O(1)$

2 hashmap template

Is a class template that manages the data kept in a dictionary. It defines the basic interface. Template parameters are:
KeyType
ValueType
It provides a way of inserting, deleting elements and checking the state of an index(the state can be initialized, uninitialized deleted, or not deleted).
memory complexity: $O(n)$
constructor

```
template <typename keyType, typename valueType>
hashmap<keyType, valueType>::hashmap(size_t arraySize);
```

arraySize

Creates an object with empty array of arraySize length.
time complexity: $O(1)$

setItem

```
void setItem(size_t index, keyType key, valueType value);
```

index - an index of a key, value pair
key
value
Inserts key, value pair into dictionary.
time complexity: $O(1)$

deleteItem

```
void deleteItem(size_t index);
```

index - index of a pair.
Deletes the pair at index. Data is not overwritten but just marked as deleted and then reused. The pair can be deleted multiple times.
time complexity: $O(1)$

isInitialized

```
bool isInitialized(size_t index);
```

index - index of a pair.

Returns true if a index is initialized. False otherwise.
time complexity: $O(1)$

isDeleted

```
    bool isDeleted(size_t index);
```

index - index of a pair.
Returns true if elements are deleted at the given index. False otherwise.
time complexity: $O(1)$

getKey

```
    keyType getKey(size_t index);
```

index - index of a key.
Returns key at index. If the index wasn't initialized then the return value is undefined.
time complexity: $O(1)$

getValue

```
    valueType getValue(size_t index);
```

index - index of a value.
Returns value at index. If the index wasn't initialized then the return value is undefined.
time complexity: $O(1)$

3 dictionary template

class template that implements dictionary with double hashing. Template parameters are:
KeyType
ValueType
Structure contains two hashmap objects which allow resizing in $O(1)$. The dictionary keeps a 10% fill ratio.
memory complexity: $O(n)$
constructor

```
    dictionary(size_t (*hashF1)(keyType), size_t (*hashF2)(keyType));
```

hashF1 - first hashing function. Is called when the index is guessed the first time.
hashF2 - second hashing function. Is called unless the index returned is valid, when the first guess wasn't correct.
Second function return value can be any because the size of an array is a prime number. Creates an empty dictionary with hashing determined by functions given.
time complexity: $O(1)$

setItem

```
    template <typename keyType, typename valueType>
    void dictionary<keyType, valueType>::setItem(keyType key, valueType
        val);
```

key
value
Using the double hashing method inserts pair at computed index.
It also rewrites 20 elements from the old array to the new one. When all elements are rewritten time taken by executing that method may be shorter.
When the array is filled in 1/10 it resizes.
If a duplicate is inserted, then the element that was inserted first will be read when calling getItem and deleted when calling deleteItem.
average time complexity: $O(1)$

pesymistic time complexity: $O(n)$

getItem

```
template <typename keyType, typename valueType>
valueType dictionary<keyType, valueType>::getItem(keyType key)
```

key
Returns element identified with key. when the key is not in the array throws std::out_of_range.
It also rewrites 20 elements from the old array to the new one. When all elements are rewritten time taken by executing that method may be shorter.
Caution, elements will be rewritten regardless if the specified key is present in an array.
average time complexity: $O(1)$

pesymistic time complexity: $O(n)$

deleteItem

```
    template <typename keyType, typename valueType>
    void dictionary<keyType, valueType>::deleteItem(keyType key);
```

key
Deletes pair identified by key from dictionary. when the key is not in the array throws std::out_of_range.
It also rewrites 20 elements from the old array to the new one. When all elements are rewritten time taken by executing that method may be shorter.
Caution, elements will be rewritten regardless if the specified key is present in an array.
average time complexity: $O(1)$

pesymistic time complexity: $O(n)$