```
template <typename PairTKey, typename PairTValue>
class KeyValuePair {
public:
    PairTKey key;
    PairTValue value;

    KeyValuePair() :key(), value() {}
    KeyValuePair(const PairTKey key, const PairTValue value)
        :key(key), value(value) {}

    PairTValue& operator[](PairTKey key) { return value; }
};


template <typename TKey, typename TValue>
class Dictionary {
public:

    Dictionary()
    {
        _pairs = new KeyValuePair<TKey, TValue>[_capacity];
    }

    TValue& item(const TKey key);

    TValue& operator[](const TKey key)
    {
        for (size_t i = 0; i < _count; i++)
        {
            if (_pairs[i].key == key) return _pairs[i][key];
        }

        // Key tapılmasa yenisini əlavə edin.
    }



    void add(const KeyValuePair<TKey, TValue> pair);

    void add(const TKey key, const TValue value) {
```

```
        //  check capacity

        _pairs[_count++] = { key, value };


        // Alternative
        // _pairs[_count++] = KeyValuePair<TKey, TValue>{ key, value };
    }

    size_t count() const;
    size_t capacity() const;
    TKey* keys();
    TValue* values();
    KeyValuePair<TKey, TValue>* items();

    void clear();
    bool containsKey(const TKey key) const;
    bool containsValue(const TValue value) const;
    bool remove(const TKey key);

private:
    KeyValuePair<TKey, TValue>* _pairs = nullptr;
    size_t _count = 0;
    size_t _capacity = 5;
};


int main() {

    Dictionary<int, string> dict;

    dict.add(1, "Apple");
    // dict[1] = "Apple";

    cout << dict[1] << endl;

    dict[1] = "Banana";
    cout << dict[1] << endl;
}
```