Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 25 tháng 3 2025, 2:39 PM
Kết thúc lúc	Thứ Ba, 25 tháng 3 2025, 3:39 PM
Thời gian thực hiện	1 giờ
Điểm	6,00/6,00
Điểm	10.00 trên 10.00 (100 %)

```
Câu hỏi 1
Đúng
Đạt điểm 1,00 trên 1,00
```

Implement the **put** method in template class **XHashMap** representing the **Hash Table**. The Hash Table is implemented with **Open Hashing** for handling collision, using a <u>Singly linked list</u> to store keys with the same index. The description of the method is given in the code.

```
int hashFunction(int key, int capacity) {
  return key % capacity;
}
template<class K, class V>
class XHashMap {
public:
  class Entry {
  public:
    K key;
     V value;
    Entry* next;
     Entry(K key, V value, Entry* next = 0) {
       this->key = key;
       this->value = value;
       this->next = next;
    }
  };
private:
  Entry** table; // hash table
  int capacity; // size for the hash table
  int count;
public:
  // Constructor
  XHashMap() {
    this->capacity = 10;
    this->count = 0;
    table = new Entry*[capacity];
    //reset table to 0
    for (int i = 0; i < \text{capacity}; i++) {
       table[i] = 0;
    }
  }
  ~XHashMap(){ // Destructor
    //Remove all entries in the current map
    for(int idx=0; idx < this->capacity; idx++){
       Entry* entry = this->table[idx];
       while (entry != 0){
         Entry * next = entry->next;
         delete entry;
         entry = next;
       }
    }
     //Remove table
     delete []table;
  // YOUR ANSWER
  // print table method (for testing)
  void printTable() const {
     cout << "-----The hash table is shown below-----\n";
    for (int i = 0; i < capacity; i++) {
       Entry* entry = table[i];
```

```
cout << "Index " << i << ": ";
    while (entry != 0) {
        cout << "(" << entry->key << ", " << entry->value << ") ";
        entry = entry->next;
    }
    cout << endl;
}
cout << "-----\n";
}
</pre>
```

For example:

Test	Result
map.put(6, 100);	The hash table is shown below
<pre>map.printTable();</pre>	Index 0:
	Index 1:
	Index 2:
	Index 3:
	Index 4:
	Index 5:
	Index 6: (6, 100)
	Index 7:
	Index 8:
	Index 9:
map.put(0, 10);	The hash table is shown below
<pre>map.put(10, 100); // adding not replacing</pre>	Index 0: (10, 100) (0, 10)
<pre>map.printTable();</pre>	Index 1:
	Index 2:
	Index 3:
	Index 4:
	Index 5:
	Index 6:
	Index 7:
	Index 8:
	Index 9:
map.put(1, 15);	The hash table is shown below
$\mbox{\tt map.put(1, 200);}$ // updating new value for existing index	Index 0:
<pre>map.printTable();</pre>	Index 1: (1, 200)
	Index 2:
	Index 3:
	Index 4:
	Index 5:
	Index 6:
	Index 7:
	Index 8:
	Index 9:

Answer: (penalty regime: 0, 0, 0, 0, 0, 100 %)

```
1 void put(int key, int value) {
2
     // Compute the index using the hash function
3
        int index = hashFunction(key, capacity);
4
5
        // Check if the key exists in the linked list at the index
6
        Entry* current = table[index];
        while (current != 0) {
7
8
            // If key is found, update its value
9,
            if (current->key == key) {
10
                current->value = value;
11
                return;
12
13
            current = current->next;
14
```

```
15
        // If key doesn't exist, create and insert a new entry
16
        Entry* newEntry = new Entry(key, value, table[index]);
17
18
        table[index] = newEntry;
19
20
        // Increment the count and ensure load factor
21
        count++;
22
        ensureLoadFactor(count);
23
24
    }
25
26
    void ensureLoadFactor(int current_size) {
        // Calculate the maximum allowed size based on the load factor (0.75 * capacity)
27
28
        int maxAllowedSize = 0.75 * capacity;
29
30
        // If the current size exceeds or equals the max allowed size, trigger rehashing
        if (current_size >= maxAllowedSize) {
31
32
            // Calculate the new capacity (1.5 times the old capacity)
            int newCapacity = capacity * 1.5;
33
34
35
            // Call the rehash function with the new capacity
36
            rehash(newCapacity);
37
        }
38
39
40 •
    void rehash(int newCapacity) {
41
      // Store the current table and capacity
        Entry** oldTable = table;
42
43
        int oldCapacity = capacity;
44
45
        // Create a new table with the new capacity and update the capacity
46
        capacity = newCapacity;
        table = new Entry*[newCapacity];
47
48
49
        // Initialize the new table with nullptr values
50
        for (int i = 0; i < newCapacity; i++) {</pre>
51
            table[i] = 0;
52
53
        // Reset count as we'll be reinserting all entries
54
55
        count = 0;
56
57
        // For each index in the old table
58
        for (int i = 0; i < oldCapacity; i++) {</pre>
59
            Entry* current = oldTable[i];
60
61
            // Traverse the linked list at that index
62 🔻
            while (current != 0) {
```

	Test	Expected	Got	
/	map.put(6, 100);	The hash table is	The hash table is	~
	<pre>map.printTable();</pre>	shown below	shown below	
		Index 0:	Index 0:	
		Index 1:	Index 1:	
		Index 2:	Index 2:	
		Index 3:	Index 3:	
		Index 4:	Index 4:	
		Index 5:	Index 5:	
		Index 6: (6, 100)	Index 6: (6, 100)	
		Index 7:	Index 7:	
		Index 8:	Index 8:	
		Index 9:	Index 9:	

	Test	Expected	Got	
/	map.put(0, 10);	The hash table is	The hash table is	Τ,
	map.put(10, 100); // adding not	shown below	shown below	
	replacing	Index 0: (10, 100) (0, 10)	Index 0: (10, 100) (0, 10)	
	map.printTable();	Index 1:	Index 1:	
		Index 2:	Index 2:	
		Index 3:	Index 3:	
		Index 4:	Index 4:	
		Index 5:	Index 5:	
		Index 6:	Index 6:	
		Index 7:	Index 7:	
		Index 8:	Index 8:	
		Index 9:	Index 9:	
_	map.put(1, 15);	The hash table is	The hash table is	Τ,
		shown below	shown below	
	map.put(1, 200); // updating new value			
	for existing index	Index 0:	Index 0:	
	<pre>map.printTable();</pre>	Index 1: (1, 200)	Index 1: (1, 200)	
		Index 2:	Index 2:	
		Index 3:	Index 3:	
		Index 4:	Index 4:	
		Index 5:	Index 5:	
		Index 6:	Index 6:	
		Index 7:	Index 7:	
		Index 8:	Index 8:	
		Index 9:	Index 9:	
/	map.put(101, 500);	The hash table is	The hash table is	Τ,
	map.put(101, 100);	shown below	shown below	
	map.put(101, 300);	Index 0:	Index 0:	
	<pre>map.printTable();</pre>	Index 1: (101, 300)	Index 1: (101, 300)	
		Index 2:	Index 2:	
		Index 3:	Index 3:	
		Index 4:	Index 4:	
		Index 5:	Index 5:	
		Index 6:	Index 6:	
		Index 7:	Index 7:	
		Index 8:	Index 8:	
		Index 9:	Index 9:	
	map.put(874, 912);	The hash table is	The hash table is	Τ,
-		shown below	shown below	
	map.put(557, 367);			
	map.put(738, 612);	Index 0:	Index 0:	
	map.put(986, 477);	Index 1:	Index 1:	
	map.put(424, 315);	Index 2:	Index 2:	
	<pre>map.printTable();</pre>	Index 3:	Index 3:	
		Index 4: (424, 315) (874, 912)	Index 4: (424, 315) (874, 912)	
		Index 5:	Index 5:	
		Index 6: (986, 477)	Index 6: (986, 477)	
		Index 7: (557, 367)	Index 7: (557, 367)	
		Index 8: (738, 612)	Index 8: (738, 612)	
		Index 8: (738, 612) Index 9:	Index 8: (738, 612) Index 9:	

	Test	Expected	Got	
/	map.put(112, 545);	The hash table is	The hash table is	\
	map.put(790, 999);	shown below	shown below	
	map.put(350, 432);	Index 0: (240, 890)	Index 0: (240, 890)	
	map.put(678, 805);	Index 1:	Index 1:	
	map.put(478, 389); map.put(432, 217);	Index 1:	Index 1:	
	map.put(883, 374);	Index 3: (678, 805)	Index 3: (678, 805)	
	map.put(112, 596);	Index 4:	Index 4:	
	map.put(926, 314);	Index 5: (350, 432)	Index 5: (350, 432)	
	map.put(240, 890);	Index 6:	Index 6:	
	map.put(432, 410);	Index 7: (112, 596)	Index 7: (112, 596)	
	<pre>map.printTable();</pre>	Index 8:	Index 8:	
		Index 9:	Index 9:	
		Index 10: (790, 999)	Index 10: (790, 999)	
		Index 11: (926, 314)	Index 11: (926, 314)	
		Index 12: (432, 410)	Index 12: (432, 410)	
		Index 13: (883, 374)	Index 13: (883, 374)	
		Index 14:	Index 14:	
/	map.put(510, 789);	The hash table is	The hash table is	Τ,
	map.put(734, 645);	shown below	shown below	
	map.put(510, 132);	Index 0: (510, 732) (210, 473)	Index 0: (510, 732) (210, 473)	
	map.put(341, 981);	(480, 280)	(480, 280)	
		Index 1: (556, 990)	Index 1: (556, 990)	
	map.put(210, 473);		, , ,	
	map.put(653, 550);	Index 2:	Index 2:	
	map.put(480, 280);	Index 3:	Index 3:	
	map.put(556, 990);	Index 4:	Index 4:	
	map.put(808, 359);	Index 5:	Index 5:	
	map.put(510, 732);	Index 6:	Index 6:	
	<pre>map.printTable();</pre>	Index 7:	Index 7:	
		Index 8: (653, 550)	Index 8: (653, 550)	
		Index 9:	Index 9:	
		Index 10:	Index 10:	
		Index 11: (341, 981)	Index 11: (341, 981)	
		Index 12:	Index 12:	
		Index 13: (808, 359)	Index 13: (808, 359)	
		Index 14: (734, 645)	Index 14: (734, 645)	
/	map.put(601, 238);	The hash table is	The hash table is	١,
	map.put(148, 870);	shown below	shown below	
	map.put(481, 711);	Index 0:	Index 0:	
	map.put(753, 922);	Index 1: (601, 238) (481, 711)	Index 1: (601, 238) (481, 711)	
	map.put(642, 583);	Index 2:	Index 2:	
	map.put(191, 774);	Index 3: (753, 922)	Index 2: Index 3: (753, 922)	
	map.put(191, 774); map.put(854, 417);	Index 4: (529, 908)	Index 4: (529, 908)	
	map.put(770, 101);	Index 5: (980, 346) (770, 101)	Index 5: (980, 346) (770, 101)	
	map.put(980, 346);	Index 6:	Index 6:	
	map.put(529, 908);	Index 7:	Index 7:	
	<pre>map.printTable();</pre>	Index 8:	Index 8:	
		Index 9:	Index 9:	
		Index 10:	Index 10:	
		Index 11: (191, 774)	Index 11: (191, 774)	
		Index 12: (642, 583)	Index 12: (642, 583)	
		Index 13: (148, 870)	Index 13: (148, 870)	
		Index 14: (854, 417)	Index 14: (854, 417)	

	Test	Expected	Got	
<u> </u>	map.put(143, 541);	The hash table is	The hash table is	\
	map.put(143, 222);	shown below	shown below	
	map.put(143, 735);	Index 0:	Index 0:	
	map.put(143, 900);	Index 1:	Index 1:	
	map.put(143, 587);	Index 2:	Index 2:	
	map.put(143, 101);	Index 3: (143, 541)	Index 3: (143, 541)	
	map.put(143, 348);	Index 4:	Index 4:	
	map.put(143, 641);	Index 5:	Index 5:	
	map.put(143, 924);	Index 6:	Index 6:	
	map.put(143, 541);	Index 7:	Index 7:	
	<pre>map.printTable();</pre>	Index 8:	Index 8:	
		Index 9:	Index 9:	
~	map.put(132, 664);	The hash table is	The hash table is	+
	map.put(312, 305);	shown below	shown below	
	map.put(232, 743);	Index 0:	Index 0:	
	map.put(322, 101);	Index 1:	Index 1:	
	map.put(452, 651);	Index 2: (452, 651) (542, 836)	Index 2: (452, 651) (542, 836)	
	map.put(542, 836);	Index 3:	Index 3:	
	map.put(672, 129);	Index 4:	Index 4:	
	map.put(762, 432);	Index 5:	Index 5:	
	map.put(892, 923);	Index 6:	Index 6:	
	map.put(982, 489);	Index 7: (982, 489) (892, 923)	Index 7: (982, 489) (892, 923)	
	<pre>map.printTable();</pre>	(232, 743) (322, 101)	(232, 743) (322, 101)	
		Index 8:	Index 8:	
		Index 9:	Index 9:	
		Index 10:	Index 10:	
		Index 11:	Index 11:	
		Index 12: (762, 432) (132, 664)	Index 12: (762, 432) (132, 664)	
		(312, 305) (672, 129)	(312, 305) (672, 129)	
		Index 13:	Index 13:	
		Index 14:	Index 14:	

Đúng
Marks for this submission: 1,00/1,00.

```
Câu hỏi 2
Đúng
Đạt điểm 1,00 trên 1,00
```

Implement the **get** method in template class **XHashMap** representing the **Hash Table**. The Hash Table is implemented with **Open Hashing** for handling collision, using a <u>Singly linked list</u> to store keys with the same index. The description of the method is given in the code.

```
int hashFunction(int key, int capacity) {
  return key % capacity;
}
template<class K, class V>
class XHashMap {
public:
  class Entry {
  public:
    K key;
     V value;
    Entry* next;
    Entry(K key, V value, Entry* next = 0) {
       this->key = key;
       this->value = value;
       this->next = next;
    }
  };
private:
  Entry** table; // hash table
  int capacity; // size for the hash table
  int count;
public:
  // Constructor
  XHashMap() {
    this->capacity = 10;
    this->count = 0;
    table = new Entry*[capacity];
    //reset table to 0
    for (int i = 0; i < \text{capacity}; i++) {
       table[i] = 0;
    }
  }
  ~XHashMap(){ // Destructor
    //Remove all entries in the current map
    for(int idx=0; idx < this->capacity; idx++){
       Entry* entry = this->table[idx];
       while (entry != 0){
         Entry * next = entry->next;
         delete entry;
         entry = next;
       }
    }
    //Remove table
    delete []table;
  }
  // put method
  void put(int key, int value); //Already implemented
  // YOUR ANSWER
};
```

For example:

Test	Result
<pre>vector<int> keys = {20};</int></pre>	Value for key 20: 95
for(int key: keys){	
try{	
<pre>cout << "Value for key " << key <<": " << map.get(key) << endl;</pre>	
}	
<pre>catch (std::out_of_range& e){</pre>	
<pre>cout << e.what() << endl;</pre>	
}	
}	

Answer: (penalty regime: 0, 0, 0, 0, 0, 100 %)

```
1 • V& get(K key) {
                                                                                       // Calculate the index using the hash function
         2
         3
                                                                                       int index = hashFunction(key, capacity);
         4
         5
                                                                                       // Access and traverse the linked list at that index
                                                                                      Entry* current = table[index];
         6
         7 •
                                                                                       while (current != 0) {
                                                                                                                               // If the current entry's key matches, return its value % \left( 1\right) =\left( 1\right) \left( 1
         8
         9
                                                                                                                               if (current->key == key) {
10
                                                                                                                                                                          return current->value;
11
12
                                                                                                                               current = current->next;
13
14
15
                                                                                       // If not found, throw std::out_of_range exception
16
                                                                                      throw std::out_of_range("Key not found");
17 }
```

	Test	Expected	Got	
~	<pre>vector<int> keys = {20}; for(int key: keys){ try{ cout << "Value for key " << key <<": " << map.get(key) << endl; } catch (std::out_of_range& e){ cout << e.what() << endl; } }</int></pre>	Value for key 20: 95	Value for key 20: 95	~
~	<pre>vector<int> keys = {52, 35}; for (int key : keys) { try { cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (const std::out_of_range& e) { cout << "Key " << key << " not found: " << e.what() << endl; } }</int></pre>	Value for key 52: 70 Value for key 35: 99	Value for key 52: 70 Value for key 35: 99	~
~	<pre>vector<int> keys = {20, 1, 57}; for(int key: keys){ try{ cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (std::out_of_range& e){ cout << e.what() << endl; } }</int></pre>	Value for key 20: 95 Value for key 1: Key not found Value for key 57: 80	Value for key 20: 95 Value for key 1: Key not found Value for key 57: 80	~
~	<pre>vector<int> keys = {87, 65, 43, 92, 12}; for (int key : keys) { try { cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (std::out_of_range& e) { cout << e.what() << endl; } }</int></pre>	Value for key 87: 77 Value for key 65: Key not found Value for key 43: 15 Value for key 92: 32 Value for key 12: Key not found	Value for key 87: 77 Value for key 65: Key not found Value for key 43: 15 Value for key 92: 32 Value for key 12: Key not found	~
~	<pre>vector<int> keys = {53, 91, 79, 4, 99}; for (int key : keys) { try { cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (std::out_of_range& e) { cout << e.what() << endl; } }</int></pre>	Value for key 53: 26 Value for key 91: Key not found Value for key 79: Key not found Value for key 4: 74 Value for key 99: Key not found	Value for key 53: 26 Value for key 91: Key not found Value for key 79: Key not found Value for key 4: 74 Value for key 99: Key not found	~

https://lms.hcmut.edu.vn/mod/quiz/review.php? attempt = 5500163&cmid = 522397

	Test	Expected	Got	
,	<pre>vector<int> keys = {77, 33, 56, 20, 60, 70, 15, 48, 68, 11}; for (int key : keys) { try { cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (std::out_of_range& e) { cout << e.what() << endl; } }</int></pre>	Value for key 77: Key not found Value for key 33: 96 Value for key 56: 29 Value for key 20: 95 Value for key 60: Key not found Value for key 70: 43 Value for key 15: Key not found Value for key 48: 31 Value for key 68: 57 Value for key 11: 72	Value for key 77: Key not found Value for key 33: 96 Value for key 56: 29 Value for key 60: Key not found Value for key 70: 43 Value for key 15: Key not found Value for key 48: 31 Value for key 68: 57 Value for key 11: 72	~
	<pre>vector<int> keys = {62, 89, 1, 96, 81, 40, 54, 9, 82, 18}; for (int key : keys) { try { cout << "Value for key " << key << ": " << map.get(key) << endl; } catch (std::out_of_range& e) { cout << e.what() << endl; } }</int></pre>	Value for key 62: Key not found Value for key 89: Key not found Value for key 1: Key not found Value for key 96: Key not found Value for key 81: Key not found Value for key 40: 28 Value for key 54: Key not found Value for key 54: Key not found Value for key 9: 81 Value for key 82: 65 Value for key 18: 54	Value for key 62: Key not found Value for key 89: Key not found Value for key 1: Key not found Value for key 96: Key not found Value for key 81: Key not found Value for key 40: 28 Value for key 54: Key not found Value for key 9: 81 Value for key 9: 81 Value for key 82: 65 Value for key 18: 54	~

Test	Expected	Got	
vector <int> keys = {94, 71, 26, 25, 84, 97, 17, 100, 31, 42};</int>	Value for key 94:	Value for key	~
<pre>for (int key : keys) {</pre>	Key not found	94: Key not	
try {	Value for key 71:	found	
cout << "Value for key " << key << ": " << map.get(key) <<	90	Value for key	
endl;	Value for key 26:	71: 90	
<pre>} catch (std::out_of_range& e) {</pre>	79	Value for key	
<pre>cout << e.what() << endl;</pre>	Value for key 25:	26: 79	
}	11	Value for key	
}	Value for key 84:	25: 11	
	Key not found	Value for key	
	Value for key 97:	84: Key not	
	88	found	
	Value for key 17:	Value for key	
	23	97: 88	
	Value for key 100:	Value for key	
	Key not found	17: 23	
	Value for key 31:	Value for key	
	45	100: Key not	
	Value for key 42:	found	
	Key not found	Value for key	
		31: 45	
		Value for key	
		42: Key not	
		found	

	Test	Expected	Got	
/	vector <int> keys = {86, 2, 22, 75, 78, 67, 29, 88, 45, 63, 69, 7, 95,</int>	Value for key 86:	Value for key	~
	66, 34, 52, 13, 58, 46, 49};	Key not found	86: Key not	
	for (int key : keys) {	Value for key 2:	found	
	try {	Key not found	Value for key	
	cout << "Value for key " << key << ": " << map.get(key) <<	Value for key 22:	2: Key not	
	endl;	89	found	
	} catch (std::out_of_range& e) {	Value for key 75:	Value for key	
	<pre>cout << e.what() << endl;</pre>	Key not found	22: 89	
	}	Value for key 78:	Value for key	
	}	19	75: Key not	
		Value for key 67:	found	
		Key not found	Value for key	
		Value for key 29:	78: 19	
		48	Value for key	
		Value for key 88:	67: Key not	
		68	found	
		Value for key 45:	Value for key	
		Key not found	29: 48	
		Value for key 63:	Value for key	
		14	88: 68	
		Value for key 69:	Value for key	
		55	45: Key not	
		Value for key 7:	found	
		Key not found	Value for key	
		Value for key 95:	63: 14	
		Key not found	Value for key	
		Value for key 66:	69: 55	
		85	Value for key	
		Value for key 34:	7: Key not	
		86	found	
		Value for key 52:	Value for key	
		70	95: Key not	
		Value for key 13:	found	
		84	Value for key	
		Value for key 58:	66: 85	
		Key not found	Value for key	
		Value for key 46:	34: 86	
		73	Value for key	
		Value for key 49:	52: 70	
		93	Value for key	
			13: 84	
			Value for key	
			58: Key not	
			found	
			Value for key	
			46: 73	
			Value for key	
			49: 93	

```
Expected
                                                                                              Got
vector<int> keys = {85, 32, 50, 83, 21, 600, 3, 73, 93, 74, 345, 57,
                                                                         Value for key 85:
                                                                                              Value for key
10, 24, 200, 44, 18, 72, 80, 105, 51, 889, 26, 123, 59, 0, 101, 57,
                                                                         Key not found
                                                                                              85: Key not
35, 110};
                                                                         Value for key 32:
                                                                                              found
for (int key : keys) {
                                                                         Key not found
                                                                                              Value for key
   try {
                                                                         Value for key 50:
                                                                                              32: Key not
        cout << "Value for key " << key << ": " << map.get(key) <</pre>
                                                                                              found
                                                                         21
                                                                         Value for key 83:
                                                                                              Value for key
endl:
   } catch (std::out_of_range& e) {
                                                                         87
                                                                                              50: 21
        cout << e.what() << endl;</pre>
                                                                         Value for key 21:
                                                                                              Value for key
                                                                         94
                                                                                              83: 87
    }
}
                                                                         Value for key 600:
                                                                                              Value for key
                                                                         Key not found
                                                                                              21: 94
                                                                         Value for key 3:
                                                                                              Value for key
                                                                         51
                                                                                              600: Key not
                                                                         Value for key 73:
                                                                                              found
                                                                                              Value for key
                                                                         30
                                                                         Value for key 93:
                                                                                              3: 51
                                                                                              Value for key
                                                                         Value for key 74:
                                                                                              73: 30
                                                                         Key not found
                                                                                              Value for key
                                                                                              93: 53
                                                                         Value for key 345:
                                                                         Kev not found
                                                                                              Value for kev
                                                                         Value for key 57:
                                                                                              74: Key not
                                                                                              found
                                                                         Value for key 10:
                                                                                              Value for key
                                                                         Key not found
                                                                                              345: Key not
                                                                         Value for key 24:
                                                                                              found
                                                                         Key not found
                                                                                              Value for key
                                                                         Value for key 200:
                                                                                              57: 80
                                                                         Key not found
                                                                                              Value for key
                                                                         Value for key 44:
                                                                                              10: Key not
                                                                         16
                                                                                              found
                                                                         Value for key 18:
                                                                                              Value for key
                                                                                              24: Key not
                                                                         54
                                                                         Value for key 72:
                                                                                              found
                                                                         Key not found
                                                                                              Value for key
                                                                         Value for key 80:
                                                                                              200: Key not
                                                                         Key not found
                                                                                              found
                                                                         Value for key 105:
                                                                                              Value for key
                                                                         Key not found
                                                                                              44: 16
                                                                         Value for key 51:
                                                                                              Value for key
                                                                         Key not found
                                                                                              18: 54
                                                                         Value for key 889:
                                                                                              Value for key
                                                                         Key not found
                                                                                              72: Key not
                                                                         Value for key 26:
                                                                                              found
                                                                                              Value for key
                                                                         Value for key 123:
                                                                                              80: Key not
                                                                         Kev not found
                                                                                              found
                                                                         Value for key 59:
                                                                                              Value for key
                                                                         60
                                                                                              105: Key not
                                                                         Value for key 0:
                                                                                              found
                                                                         Key not found
                                                                                              Value for key
                                                                         Value for key 101:
                                                                                              51: Key not
                                                                         Key not found
                                                                                              found
                                                                         Value for key 57:
                                                                                              Value for key
                                                                                              889: Key not
                                                                         Value for key 35:
                                                                                              found
                                                                                              Value for key
                                                                         Value for key 110:
                                                                                              26: 79
                                                                         Key not found
                                                                                              Value for key
                                                                                              123: Key not
                                                                                              found
                                                                                              Value for key
                                                                                              59: 60
                                                                                              Value for key
                                                                                              0: Key not
                                                                                              found
                                                                                              Value for key
                                                                                              101: Key not
                                                                                              found
                                                                                              Value for key
```

Test	Expected	Got	
		57: 80	
		Value for key	
		35: 99	
		Value for key	
		110: Key not	
		found	



Marks for this submission: 1,00/1,00.

```
Câu hải 3
Đúng
Đạt điểm 1,00 trên 1,00
```

Implement the **remove** method in template class **XHashMap** representing the **Hash Table**. The Hash Table is implemented with **Open Hashing** for handling collision, using a <u>Singly linked list</u> to store keys with the same index. The description of the method is given in the code.

```
int hashFunction(int key, int capacity) {
  return key % capacity;
}
template<class K, class V>
class XHashMap {
public:
  class Entry {
  public:
     K key;
     V value;
    Entry* next;
     Entry(K key, V value, Entry* next = 0) {
       this->key = key;
       this->value = value;
       this->next = next;
     }
  };
private:
  Entry** table; // hash table
  int capacity; // size for the hash table
  int count;
public:
  // Constructor
  XHashMap() {
     this->capacity = 10;
     this->count = 0;
     table = new Entry*[capacity];
     //reset table to 0
    for (int i = 0; i < \text{capacity}; i++) {
       table[i] = 0;
    }
  }
  ~XHashMap(){ // Destructor
     //Remove all entries in the current map
     for(int idx=0; idx < this->capacity; idx++){
       Entry* entry = this->table[idx];
       while (entry != 0){
         Entry * next = entry->next;
         delete entry;
          entry = next;
       }
     }
     //Remove table
     delete []table;
  }
  // put method
  void put(int key, int value); //Already implemented
  V& get (int key); //Already implemented
  // YOUR ANSWER
};
```

For example:

Test	Result
<pre>vector<int> keys = {68}; // Update the keys vector to hold the new key for (int key : keys) { try {</int></pre>	Remove for key = 68, value = 57
<pre>cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; }</pre>	
<pre>catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message }</pre>	
}	
<pre>vector<int> keys = {92, 51, 34}; // Include all keys in the vector for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl;</int></pre>	Remove for key = 92, value = 32 Remove for key = 51, value = Key not found Remove for key = 34, value = 86
<pre>} catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } }</pre>	
<pre>vector<int> keys = {83, 4, 77, 28, 56}; // Update the keys vector with the new keys for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } }</int></pre>	Remove for key = 83, value = 87 Remove for key = 4, value = 74 Remove for key = 77, value = Key not found Remove for key = 28, value = Key not found Remove for key = 56, value = 29

Answer: (penalty regime: 0, 0, 0, 0, 0, 100 %)

```
1 v V remove(K key) {
 2
        // Calculate the index using the hash function
        int index = hashFunction(key, capacity);
3
4
        // Start at the head of the linked list for this index \,
5
6
        Entry* current = table[index];
        Entry* prev = nullptr;
7
8
        // Traverse the linked list to find the key
9
        while (current != nullptr) {
10
11
             // If key is found
            if (current->key == key) {
12
13
                 \ensuremath{//} Store the value to return
14
                V value = current->value;
15
16
                // If this is the first node in the list
                if (prev == nullptr) {
17
18
                     // Update the head of the list
                     table[index] = current->next;
19
20
                }
                else {
21
22
                     // Link previous node to the next node
23
                     prev->next = current->next;
24
25
26
                // Decrement the count
27
                count--;
28
29
                 // Delete the entry and return its value
30
                 delete current;
31
                return value;
32
            }
33
             // Move to next node
34
```

```
Hash: Xem lại lần làm thử | BK-LMS
35
               prev = current;
36
               current = current->next;
37
38
          // If key is not found, throw an out_of_range exception
throw std::out_of_range("Key not found");
39
40
41 }
```

	Test	Expected	Got	
~	<pre>vector<int> keys = {68}; // Update the keys vector to hold the new key for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } } }</int></pre>	Remove for key = 68, value = 57	Remove for key = 68, value = 57	~
~	<pre>vector<int> keys = {92, 51, 34}; // Include all keys in the vector for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } }</int></pre>	Remove for key = 92, value = 32 Remove for key = 51, value = Key not found Remove for key = 34, value = 86	Remove for key = 92, value = 32 Remove for key = 51, value = Key not found Remove for key = 34, value = 86	~
~	<pre>vector<int> keys = {83, 4, 77, 28, 56}; // Update the keys vector with the new keys for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } }</int></pre>	Remove for key = 83, value = 87 Remove for key = 4, value = 74 Remove for key = 77, value = Key not found Remove for key = 28, value = Key not found Remove for key = 56, value = 29	Remove for key = 83, value = 87 Remove for key = 4, value = 74 Remove for key = 77, value = Key not found Remove for key = 28, value = Key not found Remove for key = 56, value = 29	~

Test	Expected	Got	
<pre>vector<int> keys = {65, 21, 100, 47, 59, 14, 99, 76, 22, 70};</int></pre>	Remove for key = 65, value = Key not found Remove for key = 21, value = 94 Remove for key = 100, value = Key not found Remove for key = 47, value = Key not found Remove for key = 59, value = 60 Remove for key = 14, value = 23 Remove for key = 99, value = Key not found Remove for key = 76, value = 43 Remove for key = 22, value = 89 Remove for key = 70, value = 43	Remove for key = 65, value = Key not found Remove for key = 21, value = 94 Remove for key = 100, value = Key not found Remove for key = 47, value = Key not found Remove for key = 59, value = 60 Remove for key = 14, value = 23 Remove for key = 99, value = Key not found Remove for key = 22, value = 43 Remove for key = 22, value = 89 Remove for key = 70,	
<pre>vector<int> keys = {11, 41, 87, 9, 39, 7, 25, 46, 40, 16}; // Updated keys vector for (int key : keys) { try { cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception cout << e.what() << endl; // Print the exception message } }</int></pre>	Remove for key = 11, value = 72 Remove for key = 41, value = 62 Remove for key = 87, value = 77 Remove for key = 9, value = 81 Remove for key = 39, value = 91 Remove for key = 7, value = Key not found Remove for key = 25, value = 11 Remove for key = 46, value = 73 Remove for key = 40, value = 28 Remove for key = 16, value = Key not found	Remove for key = 11, value = 72 Remove for key = 41, value = 62 Remove for key = 87, value = 77 Remove for key = 9, value = 81 Remove for key = 39, value = 91 Remove for key = 7, value = Key not found Remove for key = 25, value = 11 Remove for key = 46, value = 73 Remove for key = 40, value = 28 Remove for key = 16, value = Key not	~

Test		Expected	Got	
<pre>// Updated keys vo for (int key : key try { cout << "! << map.remove(key } catch (std::oo out_of_range except</pre>	<pre>gemove for key = " << key << ", value = ") << endl; ut_of_range& e) { // Catch the</pre>	Remove for key = 85, value = Key not found Remove for key = 73, value = 30 Remove for key = 80, value = Key not found Remove for key = 15, value = Key not found Remove for key = 66, value = 85 Remove for key = 88, value = 68 Remove for key = 60, value = Key not found Remove for key = 53, value = 26 Remove for key = 90, value = Key not found Remove for key = 97, value = 80	Remove for key = 85, value = Key not found Remove for key = 73, value = 30 Remove for key = 80, value = Key not found Remove for key = 15, value = Key not found Remove for key = 66, value = 85 Remove for key = 88, value = 68 Remove for key = 60, value = Key not found Remove for key = 60, value = Key not found Remove for key = 53, value = 26 Remove for key = 90, value = Key not found Remove for key = 90, value = Key not found Remove for key = 57, value = 80	
<pre>// Updated keys vo for (int key : key try { cout << "!</pre>	<pre>general system (system) s</pre>	Remove for key = 26, value = 79 Remove for key = 1, value = Key not found Remove for key = 5, value = 37 Remove for key = 94, value = Key not found Remove for key = 42, value = Key not found Remove for key = 43, value = 15 Remove for key = 97, value = 88 Remove for key = 20, value = 95 Remove for key = 64, value = 49 Remove for key = 54, value = Key not found	Remove for key = 26, value = 79 Remove for key = 1, value = Key not found Remove for key = 5, value = 37 Remove for key = 94, value = Key not found Remove for key = 42, value = Key not found Remove for key = 43, value = 15 Remove for key = 97, value = 88 Remove for key = 20, value = 95 Remove for key = 64, value = 49 Remove for key = 54, value = Key not	

	Test	Expected	Got	
/	<pre>vector<int> keys = {37, 93, 18, 62, 86, 46, 30, 35, 8, 69, 96, 24, 81, 13, 52, 84, 45, 23, 2, 31}; // Updated keys vector for (int key : keys) { try {</int></pre>	Remove for key = 37, value = Key not found Remove for key = 93, value = 53 Remove for key = 18,	Remove for key = 37, value = Key not found Remove for key = 93, value = 53	~
	<pre>cout << "Remove for key = " << key << ", value = " << map.remove(key) << endl; } catch (std::out_of_range& e) { // Catch the out_of_range exception</pre>	<pre>value = 54 Remove for key = 62, value = Key not found Remove for key = 86, value = Key not found</pre>	Remove for key = 18, value = 54 Remove for key = 62, value = Key not found	
	<pre>cout << e.what() << endl; // Print the exception message } </pre>	Remove for key = 46, value = 73 Remove for key = 30, value = 52	Remove for key = 86, value = Key not found Remove for key = 46,	
		Remove for key = 35, value = 99 Remove for key = 8, value = 66 Remove for key = 69,	<pre>value = 73 Remove for key = 30, value = 52 Remove for key = 35, value = 99</pre>	
		value = 55 Remove for key = 96, value = Key not found Remove for key = 24,	Remove for key = 8, value = 66 Remove for key = 69, value = 55	
		value = Key not found Remove for key = 81, value = Key not found Remove for key = 13,	Remove for key = 96, value = Key not found Remove for key = 24,	
		value = 84 Remove for key = 52, value = 70 Remove for key = 84, value = Key not found	<pre>value = Key not found Remove for key = 81, value = Key not found</pre>	
		Remove for key = 45, value = Key not found Remove for key = 23, value = Key not found	Remove for key = 13, value = 84 Remove for key = 52, value = 70	
		Remove for key = 2, value = Key not found Remove for key = 31, value = 45	Remove for key = 84, value = Key not found Remove for key = 45,	
			<pre>value = Key not found Remove for key = 23, value = Key not</pre>	
			found Remove for key = 2, value = Key not found	
			Remove for key = 31, value = 45	

	Test	Expected	Got	
,	vector <int> keys = {12, 61, 82, 10, 17, 49, 33, 74, 75, 86,</int>	Remove for key = 12,	Remove for key = 12,	\
	57, 78, 19, 50, 27, 67, 38, 100, 15,	value = Key not found	value = Key not	
	45}; // Updated keys vector	Remove for key = 61,	found	
	for (int key : keys) {	value = 58	Remove for key = 61,	
	try {	Remove for key = 82,	value = 58	
	cout << "Remove for key = " << key << ", value = "	value = 65	Remove for key = 82,	
	<pre><< map.remove(key) << endl;</pre>	Remove for key = 10,	value = 65	
	}	value = Key not found	Remove for key = 10,	
	<pre>catch (std::out_of_range& e) { // Catch the</pre>	Remove for key = 17,	value = Key not	
	out_of_range exception	value = 23	found	
	<pre>cout << e.what() << endl; // Print the exception</pre>	Remove for key = 49,	Remove for key = 17,	
	message	value = 93	value = 23	
	}	Remove for key = 33,	Remove for key = 49,	
	}	value = 96	value = 93	
		Remove for key = 74,	Remove for key = 33,	
		value = Key not found	value = 96	
		Remove for key = 75,	Remove for key = 74,	
		value = Key not found	value = Key not	
		Remove for key = 86,	found	
		value = Key not found	Remove for key = 75,	
		Remove for key = 57,	value = Key not	
		value = 80	found	
		Remove for key = 78,	Remove for key = 86,	
		value = 19	value = Key not	
		Remove for key = 19,	found	
		value = Key not found	Remove for key = 57,	
		Remove for key = 50,	value = 80	
		value = 21	Remove for key = 78,	
		Remove for key = 27,	value = 19	
		value = Key not found	Remove for key = 19,	
		Remove for key = 67,	value = Key not	
		value = Key not found	found	
		Remove for key = 38,	Remove for key = 50,	
		value = Key not found	value = 21	
		Remove for key = 100,	Remove for key = 27,	
		value = Key not found	value = Key not	
		Remove for key = 15,	found	
		value = Key not found	Remove for key = 67,	
		Remove for key = 45,	value = Key not	
		value = Key not found	found	
			Remove for key = 38,	
			value = Key not	
			found	
			Remove for key =	
			100, value = Key not	
			found	
			Remove for key = 15,	
			value = Key not	
			found	
			Remove for key = 45,	
			value = Key not	
			found	

	Test	Expected	Got	
~	vector <int> keys = {540, 89, 200, 99, 111, 73, 120, 23, 62, 101,</int>	Remove for key = 540, value = Key not found	Remove for key = 540, value = Key not	~
	92, 133, 57, 999, 80, 109, 66, 124, 66,	Remove for key = 89,	found	
	321, 106, 95, 128, 63, 118, 102, 55, 87,	value = Key not found Remove for key = 200,	Remove for key = 89, value = Key not	
	1}; // Updated keys vector	value = Key not found	found	
	<pre>for (int key : keys) { try {</pre>	Remove for key = 99, value = Key not found	Remove for key = 200, value = Key not	
	cout << "Remove for key = " << key << ", value = "	Remove for key = 111,	found	
	<pre><< map.remove(key) << endl; }</pre>	value = Key not found Remove for key = 73,	Remove for key = 99, value = Key not	
	catch (std::out_of_range& e) { // Catch the	value = 30	found	
	out_of_range exception	Remove for key = 120, value = Key not found	Remove for key =	
	<pre>cout << e.what() << endl; // Print the exception message</pre>	Remove for key = 23,	111, value = Key not found	
	}	value = Key not found	Remove for key = 73,	
	}	Remove for key = 62, value = Key not found	value = 30 Remove for key =	
		Remove for key = 101,	120, value = Key not	
		value = Key not found	found	
		Remove for key = 92, value = 32	Remove for key = 23, value = Key not	
		Remove for key = 133,	found	
		<pre>value = Key not found Remove for key = 57,</pre>	Remove for key = 62, value = Key not	
		value = 80	found	
		Remove for key = 999, value = Key not found	Remove for key = 101, value = Key not	
		Remove for key = 80,	found	
		value = Key not found	Remove for key = 92,	
		Remove for key = 109, value = Key not found	value = 32 Remove for key =	
		Remove for key = 66,	133, value = Key not	
		value = 85 Remove for key = 124,	found Remove for key = 57,	
		value = Key not found	value = 80	
		Remove for key = 66, value = Key not found	Remove for key = 999, value = Key not	
		Remove for key = 137,	found	
		value = Key not found	Remove for key = 80,	
		Remove for key = 321, value = Key not found	value = Key not found	
		Remove for key = 106,	Remove for key =	
		<pre>value = Key not found Remove for key = 95,</pre>	109, value = Key not found	
		value = Key not found	Remove for key = 66,	
		Remove for key = 128, value = Key not found	value = 85 Remove for key =	
		Remove for key = 63,	124, value = Key not	
		value = 14	found	
		Remove for key = 118, value = Key not found	Remove for key = 66, value = Key not	
		Remove for key = 102,	found	
		value = Key not found Remove for key = 55,	Remove for key = 137, value = Key not	
		value = Key not found	found	
		Remove for key = 87, value = 77	Remove for key = 321, value = Key not	
		Remove for key = 1,	found	
		value = Key not found	Remove for key =	
			106, value = Key not found	
			Remove for key = 95,	
			value = Key not found	
			Remove for key =	
			128, value = Key not found	
			Remove for key = 63,	
			value = 14	
			Remove for key =	

Test	Expected	Got
		118, value = Key not
		found
		Remove for key =
		102, value = Key not
		found
		Remove for key = 55,
		value = Key not
		found
		Remove for key = 87,
		value = 77
		Remove for key = 1,
		value = Key not
		found



Dúng

Marks for this submission: 1,00/1,00.

```
Câu hởi 4
Đúng
Đạt điểm 1,00 trên 1,00
```

Implement three following hashing function:

```
long int midSquare(long int seed);
long int moduloDivision(long int seed, long int mod);
long int digitExtraction(long int seed, int* extractDigits, int size);
```

Note that:

In midSquare function: we eliminate 2 last digits and get the 4 next digits.

In digitExtraction: extractDigits is a sorted array from smallest to largest index of digit in seed (index starts from 0). The array has size **size**.

For example:

Test	Result
<pre>int a[]={1,2,5}; cout << digitExtraction(122443,a,3);</pre>	223
<pre>cout <<midsquare(9452);< pre=""></midsquare(9452);<></pre>	3403

Answer: (penalty regime: 0, 0, 0 %)

```
long int midSquare(long int seed)
 1
2 ,
        seed = seed * seed;
3
4
        seed /= 100;
5
        return seed % 10000;
6
 7
    long int moduloDivision(long int seed, long int mod)
8 ▼ {
9
        return seed % mod;
10
    long int digitExtraction(long int seed,int* extractDigits,int size)
11
12 ▼ {
13
       int tmp[1000];
        for(int i = 0;i < 1000; i++) tmp[i] = -1;</pre>
14
15
        int i = 0;
16
        while(seed > 0){
17
            tmp[i] = seed % 10;
            seed /= 10;
18
19
            i++;
20
21
         long int result = 0;
22
        long int n = 1;
23
         int j = 0;
        while(size){
24
25
            result = result * 10 + tmp[i - extractDigits[j] - 1];
26
            i += 1;
27
            n *= 10;
28
            size --;
29
30
31
        return result;
32
```

	Test	Expected	Got	
~	<pre>int a[]={1,2,5}; cout << digitExtraction(122443,a,3);</pre>	223	223	~
~	<pre>cout <<midsquare(9452);< pre=""></midsquare(9452);<></pre>	3403	3403	~



Đúng) Marks for this submission: 1,00/1,00.

```
Câu hồi 5
Đúng
Đạt điểm 1,00 trên 1,00
```

Implement function

```
int foldShift(long long key, int addressSize);
int rotation(long long key, int addressSize);
```

to hashing key using Fold shift or Rotation algorithm.

Review:

The **folding method** for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size). These pieces are then added together to give the resulting hash value.

The **rotation** method rotates the last digit to the front, and apply foldShift.

For example:

Test				Result
cout	<<	rotation(600101,	2);	26

Answer: (penalty regime: 0 %)

```
#include<math.h>
 1
    #include<string.h>
 3 .
    long int to_int(string s) {
 4
         long int base=1;
        long int res=0;
5
        for (int i=s.size()-1; i>=0; i--) {
 6
7
             res += (s[i]-48)*base;
 8
             base*=10;
9
10
        return res;
11
12
    int foldShift(long long key, int addressSize)
13 ,
14
         string s="";
15
         string num=to_string(key);
16
         long int sum=0;
         for (int i=0; i<int(num.size()); ) {</pre>
17
18
             s="";
19
             for (int j=0; j<addressSize&&i+j<int(num.size()); j++) {</pre>
20
                 s+= num[i+j];
21
            }
22
             i=i+addressSize;
23
             sum+=to_int(s);
24
25
         long int mod = pow(10, addressSize);
26
         return sum % mod;
27
    }
28
29
    int rotation(long long key, int addressSize)
30 ▼ {
31
        string num=to_string(key);
32
         string s1 = num.substr(0, num.size()-1);
33
        num=num[num.size()-1]+s1;
34
        long int n = to_int(num);
35
        return foldShift(n, addressSize);
36
         return 0;
```

	Test	Expected	Got	
~	cout << rotation(600101, 2);	26	26	~



Đúng
Marks for this submission: 1,00/1,00.

1.

```
Câu hỏi 6
Đúng
Đạt điểm 1,00 trên 1,00
```

There are n people, each person has a number between 1 and 100000 (1 \le n \le 100000). Given a number target. Two people can be matched as a **perfect pair** if the sum of numbers they have is equal to target. A person can be matched no more than 1 time.

Request: Implement function:

```
int pairMatching(vector<int>& nums, int target);
```

Where nums is the list of numbers of n people, target is the given number. This function returns the number of **perfect pairs** can be found from the list.

Example:

The list of numbers is {1, 3, 5, 3, 7} and target = 6. Therefore, the number of **perfect pairs** can be found from the list is 2 (pair (1, 5) and pair (3, 3)).

Note:

In this exercise, the libraries iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm has been included and namespace std are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result	
<pre>vector<int>items{1, 3, 5, 3, 7}; int target = 6; cout << pairMatching(items, target);</int></pre>	2	
<pre>int target = 6; vector<int>items{4,4,2,1,2}; cout << pairMatching(items, target);</int></pre>	2	

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

```
int pairMatching(vector<int>& nums, int target) {
        map<int, int> m;
 2
 3
 4
        // Đếm số lần xuất hiện của từng số
 5
        for(int i : nums)
            m[i]++;
 6
7
8
        int count = 0;
9
        for(auto it = m.begin(); it != m.end(); ++it) {
10
11
            int num = it->first;
12
            int freq = it->second;
13
14
            // Trường hợp num + num == target
15
            if (num * 2 == target) {
16
                count += freq / 2;
                m[num] = 0; // Đánh dấu là đã sử dụng hết số này
17
18
19
            // Trường hợp tìm thấy cặp (num, target - num)
20
            else if (m.find(target - num) != m.end() && m[target - num] > 0) {
21
                int pair_count = min(freq, m[target - num]);
22
                count += pair_count;
                m[num] = 0;
23
24
                m[target - num] = 0; // Đánh dấu cả hai số là đã sử dụng
            }
25
26
        return count;
27
28
```

	Test	Expected	Got	
~	<pre>vector<int>items{1, 3, 5, 3, 7}; int target = 6; cout << pairMatching(items, target);</int></pre>	2	2	~



Marks for this submission: 1,00/1,00.



1.