File: MyLinkedList.h

```
#ifndef _MY_LINKED_LIST_H_
#define _MY_LINKED_LIST_H_
struct Node{
	int key; //khoa
	int value; //gia tri theo khoa
	Node* next;
};
typedef Node* NodePtr;
//khoi tao danh sach
void List_init(Node*& head);
//kiem tra danh sach rong
bool List_isEmpty(Node* head);
//giai phong danh sach
void List_free(Node*& head);
//chen phan tu moi vao danh sach
void List_insert(Node*& head, int key, int value);
//xoa mot phan tu
void List_remove(Node*& head, int key);
Node* List_search(Node* head, int key);
void List_traverse(Node* head);
#endif
```

File: MyLinkedList.cpp

```
/*
Title: Bang bam ket noi truc tiep
Author: Nhan V.T
Last update: 2022-12-16
*/

#include "MyLinkedList.h"
#include <stdio.h>

void List_init(Node*& head)
{
	head = NULL;
}
bool List_isEmpty(Node* head)
{
	return head == NULL;
}
Node* createNode(int key, int value)
{
	Node* p = new Node;
	p->key = key;
	p->value = value;
	p->next = NULL;
	return p;
}
void List_insert(Node*& head, int key, int value)
{
	Node* p = createNode(key, value);
	//chon pp them dau (add head)
	if (List_isEmpty(head))
	{
		head = p;
	}
```

```
        else
        {
                p->next = head;
                head = p;
        }
}
void removeHead(Node*& head)
{
        if (List_isEmpty(head)) return;
        else
        {
                Node* tmp = head;
                head = head->next;
                delete tmp;
        }
}
void removeAfter(Node*& head, Node* q)
{
        if (List_isEmpty(head) || q == NULL || q->next == NULL) return;
        else
        {
                Node* p = q->next;
                q->next = p->next;
                delete p;
        }
}
void List_remove(Node*& head, int key)
{
        if (List_isEmpty(head)) return;
        else
        {
                Node* p = head, *q = NULL; // q la node truoc p
                while (p != NULL && p->key != key)
                {
                        q = p;
                        p = p->next;
                }
                if (p == NULL)
                {
                        printf("Khoa %d khong ton tai!\n", key);
                        return;
                }
                else
                {
                        if (q == NULL)
                                removeHead(head); //p la head
                        else
                                removeAfter(head, q);//xoa p
                }
        }
}
void List_free(Node*& head)
{
        Node* tmp;
        while (head != NULL)
        {
                tmp = head;
                head = tmp->next;
```

```
			delete tmp;
		}
		head = NULL;
}
Node* List_search(Node* head, int key)
{
		Node* p = head;
		while (p != NULL)
		{
			if (p->key == key)
				return p;
			p = p->next;
		}
		return NULL;
}
void List_traverse(Node* head)
{
		Node* p = head;
		while (p != NULL)
		{
			printf("<%4d,%4d> ", p->key,p->value);
			p = p->next;
		}
		printf("\n");
}
```

File: HashTable_Chaning.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include "MyLinkedList.h"
#define DEFAULT_LOAD_FACTOR 0.7

struct HashTable{
		NodePtr* bucket; //su dung mang dong
		int size; //kich thuoc bang bam (so bucket)
		int count; //so luong phan tu tren bang bam
};
//ham bam
int hash(int key, int maxSize)
{
		return key% maxSize;
}
// khoi tao
void HT_init(HashTable& ht, int capacity)
{
		ht.size = capacity;
		ht.count = 0;
		ht.bucket = new NodePtr[ht.size];
		for (int i = 0; i < ht.size; i++)
		{
			List_init(ht.bucket[i]);
		}
}
//huy bang bam
void HT_free(HashTable& ht)

{
		for (int i = 0; i < ht.size; i++)
```

```
	{
		List_free(ht.bucket[i]);
	}
	ht.size = 0;
	ht.count = 0;
	delete[] ht.bucket;
}

//tim kiem
Node* HT_get(HashTable ht, int key)
{
	int h = hash(key, ht.size);

	return List_search(ht.bucket[h], key);
}
//xoa phan tu
void HT_remove(HashTable &ht, int key)
{
	int h = hash(key, ht.size);
	List_remove(ht.bucket[h], key);
}

//bam lai
void rehashing_HT(HashTable& ht_old, int capacity_new);
//chen phan tu moi vao bang bam
void HT_push(HashTable& ht, int key, int value)
{
	int h = hash(key, ht.size);
	if (List_search(ht.bucket[h], key) != NULL)
	{
		printf("Trung khoa %d\n", key);
		return;
	}
	else
	{
		//them khoa moi vao (them dau)
		List_insert(ht.bucket[h], key, value);
		ht.count++;

		//tinh lai tai trong tren bang bam
		double loadFactor = ht.count *1.0 / ht.size;
		//printf("--> load factor %.3f\n", loadFactor);
		if (loadFactor > DEFAULT_LOAD_FACTOR)
		{
			//HT_traverse(ht);
			rehashing_HT(ht, ht.size * 2); //kich thuoc moi gap doi cu
		}
	}

}
void rehashing_HT(HashTable& ht, int capacity_new)
{
	HashTable ht_new;
	HT_init(ht_new, capacity_new);
	//chuyen du lieu cua bang bam cu sang bang moi
	//duyet qua cac bucket
	for (int i = 0; i < ht.size; i++)
	{
```

```c
			//duyet cac phan tu tren bucket
			for (Node* p = ht.bucket[i]; p != NULL; p = p->next)
			{
				int key = p->key;
				int value = p->value;
				//them du lieu vao ht moi
				int h = hash(key, ht_new.size);
				HT_push(ht_new, key, value);
			}
		}
		HashTable ht_old = ht;
		ht = ht_new;
		HT_free(ht_old);
}

//duyet
void HT_traverse(HashTable ht)
{
		//duyet qua cac bucket
		for (int i = 0; i < ht.size; i++)
		{
			printf("bucket[%d]: ", i);
			List_traverse(ht.bucket[i]);
		}
		double loadFactor = 0;
		if (ht.count != 0)
			loadFactor = ht.count*1.0 / ht.size;
		printf("He so tai hien tai: [%.2f]% \n", loadFactor);
}
```

```c
void menu()
{
		printf("\n----Menu---");
		printf("\n0.Thoat");
		printf("\n1.Tai du lieu mac dinh");
		printf("\n2.Them mot phan tu");
		printf("\n3.Xoa mot phan tu");
		printf("\n4.Tra cuu phan tu theo key:");
}

void main()
{
		//khai bao bang bam: loai dia chi moi, dung pp do tuyen tinh, kich thuoc ban dau = 5
		HashTable ht;
		int tableSize = 5;

		//khoi tao bang bam
		HT_init(ht, tableSize);

		int choose = 0;
		do
		{
			system("cls");
			HT_traverse(ht);
			menu();
			printf("\nChon: ");
			scanf_s("%d", &choose);

			switch (choose)
		{
		case 0:
		{
				break;
		}
		case 1:
		{
				int a[] = { 30, 21, 4, 13, 15, 18, 22, 28, 24, 45 };
				int n = sizeof(a) / sizeof(a[0]);

				for (int i = 0; i < n; i++)
				{
					HT_push(ht, a[i], a[i]);
				}

				break;
		}
		case 2:
		{
				int key;
				printf("\nNhap key: ");
				scanf_s("%d", &key);
				HT_push(ht, key, key); //lay value trung key
				break;
		}
		case 3:
		{
				int key;
				printf("\nNhap key: ");
				scanf_s("%d", &key);
				HT_remove(ht, key);
				break;
		}
		case 4:
		{
				int key;
				printf("\nNhap key: ");
				scanf_s("%d", &key);
				Node* p = HT_get(ht, key);
				if (p != NULL)
				{
					printf("Tim thay %d", key);
				}
				else
				{
					printf("Khoa %d khong ton tai", key);
				}
				break;
		}

		default:
			break;
		}

		system("pause");

	} while (choose != 0);

	//huy bang bam
	HT_free(ht);
}
```