**// Code dùng tham khảo (bài làm của sinh viên)**

void CreateEmptyHashtable(HASHTABLE &ht)

{

cin >> ht.M;

ht.table = new NODE[ht.M];

for (int i = 0; i < ht.M; i++)

{

ht.table[i].flag = EMPTY;

ht.table[i].key = 0;

}

ht.n = 0;

}

void Insert(HASHTABLE &ht, int key)

{

if (float(ht.n+1)/ht.M > LOAD)

{

return;

}

int i = 0;

while (i< ht.M)

{

int index = HF\_LinearProbing(ht, key, i++);

if (ht.table[index].flag == EMPTY || ht.table[index].flag == DELETED)

{

ht.table[index].flag = OCCUPIED;

ht.table[index].key = key;

ht.n++;

return;

}

else if (ht.table[index].flag == OCCUPIED && ht.table[index].key == key)

{

return;

}

}

}

void Delete(HASHTABLE &ht, int key)

{

int i=0;

while (i< ht.M)

{

int index = HF\_LinearProbing(ht, key, i);

if (ht.table[index].flag == EMPTY || ht.table[index].flag == DELETED)

{

return;

}

else if (ht.table[index].flag == OCCUPIED && ht.table[index].key == key)

{

ht.table[index].flag = DELETED;

ht.table[index].key =0;

ht.n--;

return;

}

i++;

}

}

void Traverse(HASHTABLE &ht)

{

for (int i=0; i<ht.M; i++)

{

cout << i << " --> ";

if (ht.table[i].flag == OCCUPIED) cout << ht.table[i].key << endl;

else if (ht.table[i].flag == DELETED)

{

cout << "DELETED" << endl;

}

else cout << "EMPTY" << endl;

}

}

void CreateHashTable(HASHTABLE &ht)

{

CreateEmptyHashtable(ht);

int num;

cin >> num;

for (int i=0; i < num; i++)

{

int a,b;

cin >> a >>b;

if (a==1) Insert(ht, b);

if (a==0) Delete(ht, b);

}

}