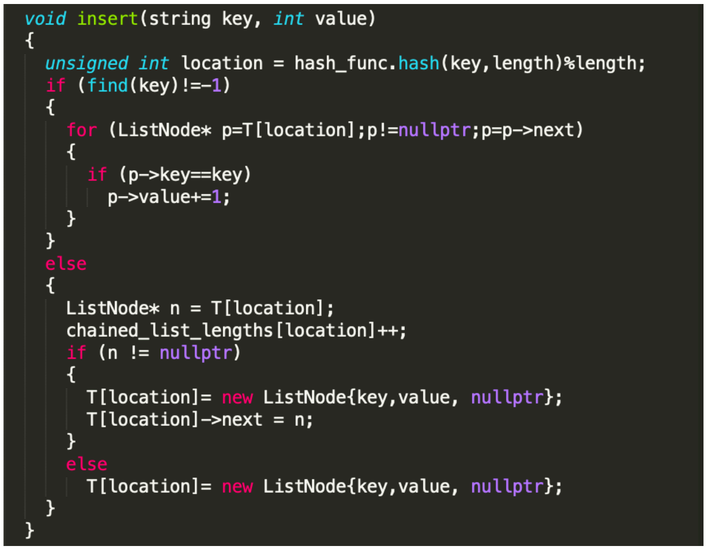
HOMEWORK 5

YANG TANG ID: 53979886

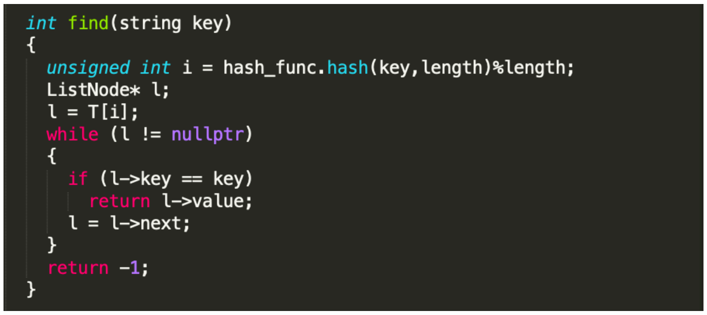
1) Code Implementation & Time complexity Analysis



```
void insert(string key, int value)
{
  unsigned int location = hash_func.hash(key,length)%length;
  if (find(key)!=-1)
  {
    for (ListNode* p=T[location];p!=nullptr;p=p->next)
    {
      if (p->key==key)
        p->value+=1;
    }
  }
  else
  {
    ListNode* n = T[location];
    chained_list_lengths[location]++;
    if (n != nullptr)
    {
      T[location]= new ListNode{key,value, nullptr};
      T[location]->next = n;
    }
    else
      T[location]= new ListNode{key,value, nullptr};
  }
}
```

Typical case Time Complexity: O(1)

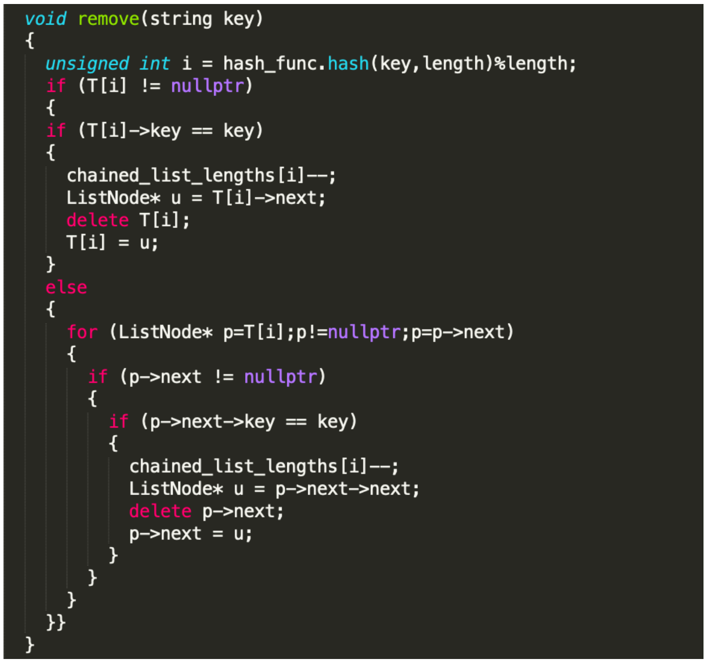
Assuming the hash function produces a uniform distribution, since the maximum chain length is around 10, which is very small, so we can regard the time complexity as O(1).



```
int find(string key)
{
  unsigned int i = hash_func.hash(key,length)%length;
  ListNode* l;
  l = T[i];
  while (l != nullptr)
  {
    if (l->key == key)
      return l->value;
    l = l->next;
  }
  return -1;
}
```

Typical case Time Complexity: O(1)

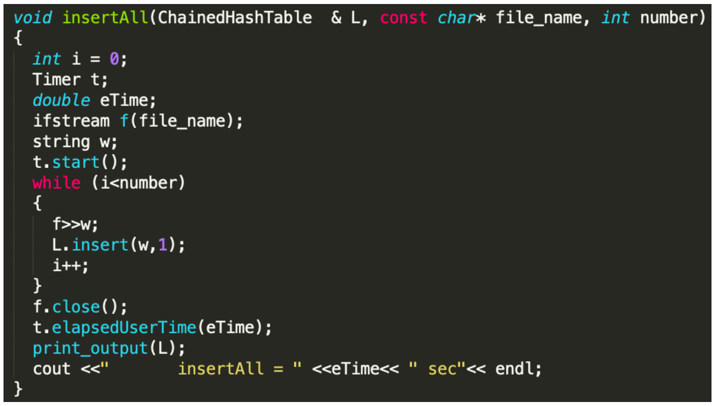
Assuming the hash function produces a uniform distribution, since the maximum chain length is around 10, which is very small, so we can regard the time complexity as O(1).



```
void remove(string key)
{
  unsigned int i = hash_func.hash(key,length)%length;
  if (T[i] != nullptr)
  {
  if (T[i]->key == key)
  {
    chained_list_lengths[i]--;
    ListNode* u = T[i]->next;
    delete T[i];
    T[i] = u;
  }
  else
  {
    for (ListNode* p=T[i];p!=nullptr;p=p->next)
    {
      if (p->next != nullptr)
      {
        if (p->next->key == key)
        {
          chained_list_lengths[i]--;
          ListNode* u = p->next->next;
          delete p->next;
          p->next = u;
        }
      }
    }
  }}
}
```

Typical case Time Complexity: O(1)

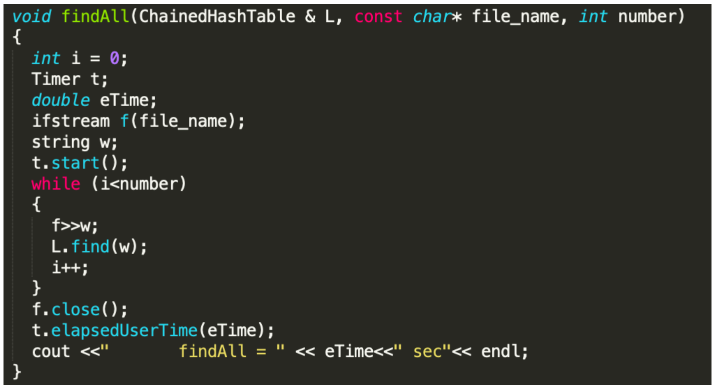
Assuming the hash function produces a uniform distribution, since the maximum chain length is around 10, which is very small, so we can regard the time complexity as O(1).



```
void insertAll(ChainedHashTable  & L, const char* file_name, int number)
{
  int i = 0;
  Timer t;
  double eTime;
  ifstream f(file_name);
  string w;
  t.start();
  while (i<number)
  {
    f>>w;
    L.insert(w,1);
    i++;
  }
  f.close();
  t.elapsedUserTime(eTime);
  print_output(L);
  cout <<"       insertAll = " <<eTime<< " sec"<< endl;
}
```

Typical case Time Complexity: O(N)

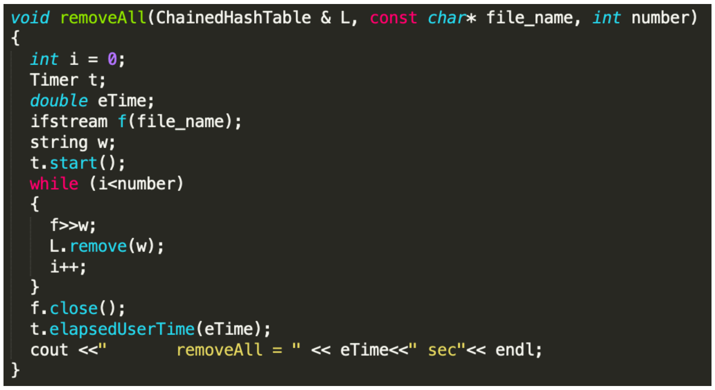
There is a while loop which execute insert() for N times.



```
void findAll(ChainedHashTable & L, const char* file_name, int number)
{
  int i = 0;
  Timer t;
  double eTime;
  ifstream f(file_name);
  string w;
  t.start();
  while (i<number)
  {
    f>>w;
    L.find(w);
    i++;
  }
  f.close();
  t.elapsedUserTime(eTime);
  cout <<"       findAll = " << eTime<<" sec"<< endl;
}
```

Typical case Time Complexity: O(N)

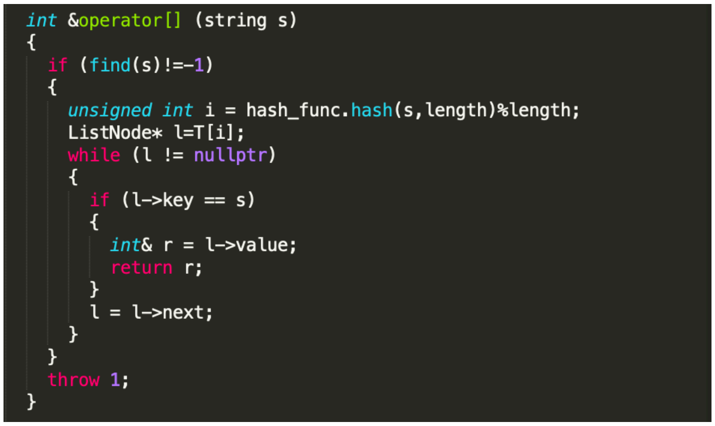
There is a while loop which execute find() for N times.



```
void removeAll(ChainedHashTable & L, const char* file_name, int number)
{
  int i = 0;
  Timer t;
  double eTime;
  ifstream f(file_name);
  string w;
  t.start();
  while (i<number)
  {
    f>>w;
    L.remove(w);
    i++;
  }
  f.close();
  t.elapsedUserTime(eTime);
  cout <<"       removeAll = " << eTime<<" sec"<< endl;
}
```

Typical case Time Complexity: O(N)

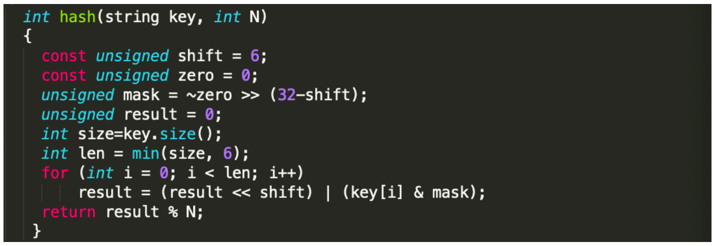
There is a while loop which execute remove() for N times.



```
int &operator[] (string s)
{
  if (find(s)!=-1)
  {
    unsigned int i = hash_func.hash(s,length)%length;
    ListNode* l=T[i];
    while (l != nullptr)
    {
      if (l->key == s)
      {
        int& r = l->value;
        return r;
      }
      l = l->next;
    }
  }
  throw 1;
}
```

Typical case Time Complexity: O(1)

Assuming the hash function produces a uniform distribution, since the maximum chain length is around 10, which is very small, so we can regard the time complexity of the while loop as O(1).

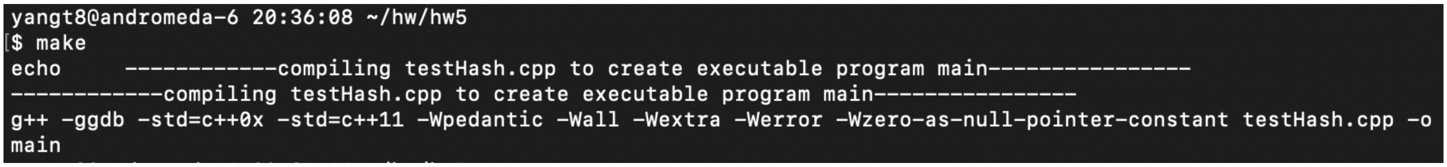


```
int hash(string key, int N)
{
  const unsigned shift = 6;
  const unsigned zero = 0;
  unsigned mask = ~zero >> (32-shift);
  unsigned result = 0;
  int size=key.size();
  int len = min(size, 6);
  for (int i = 0; i < len; i++)
      result = (result << shift) | (key[i] & mask);
  return result % N;
}
```

Typical case Time Complexity: O(1)

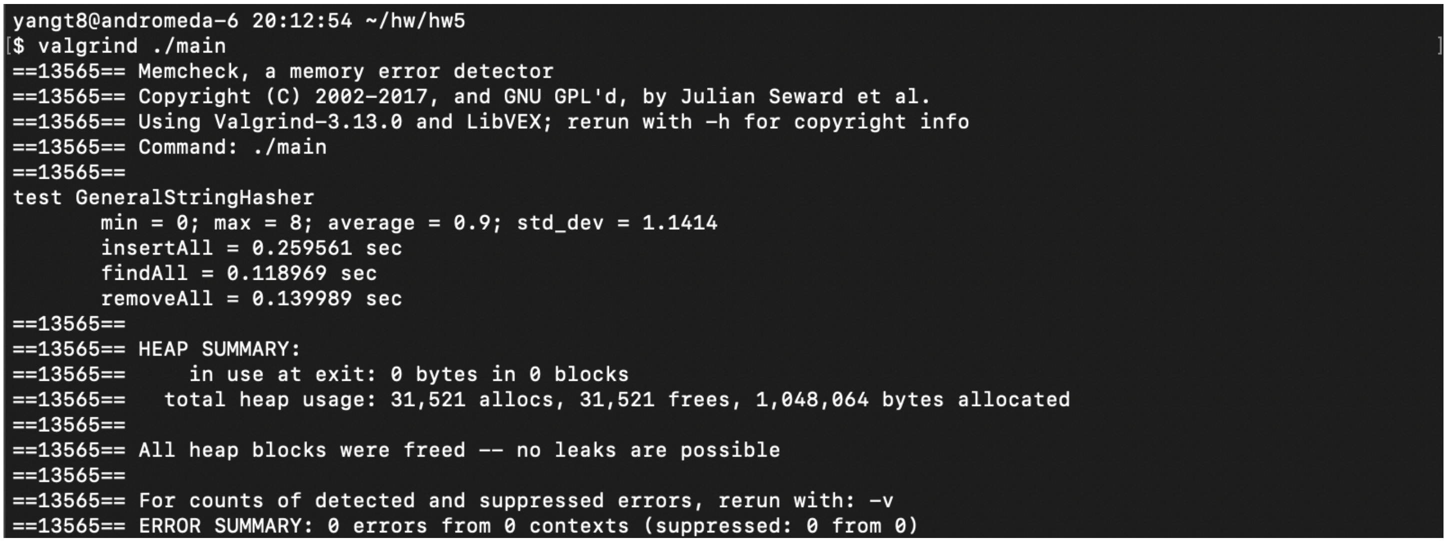
Because the for loop at most executes 6 times, so we can regard the time complexity of it as O(1)

2) Proof of compilation (test by the first 4500 words from random.txt)



```
yangt8@andromeda-6 20:36:08 ~/hw/hw5
[$ make
echo     ------------compiling testHash.cpp to create executable program main----------------
------------compiling testHash.cpp to create executable program main----------------
g++ -ggdb -std=c++0x -std=c++11 -Wpedantic -Wall -Wextra -Werror -Wzero-as-null-pointer-constant testHash.cpp -o
main
```

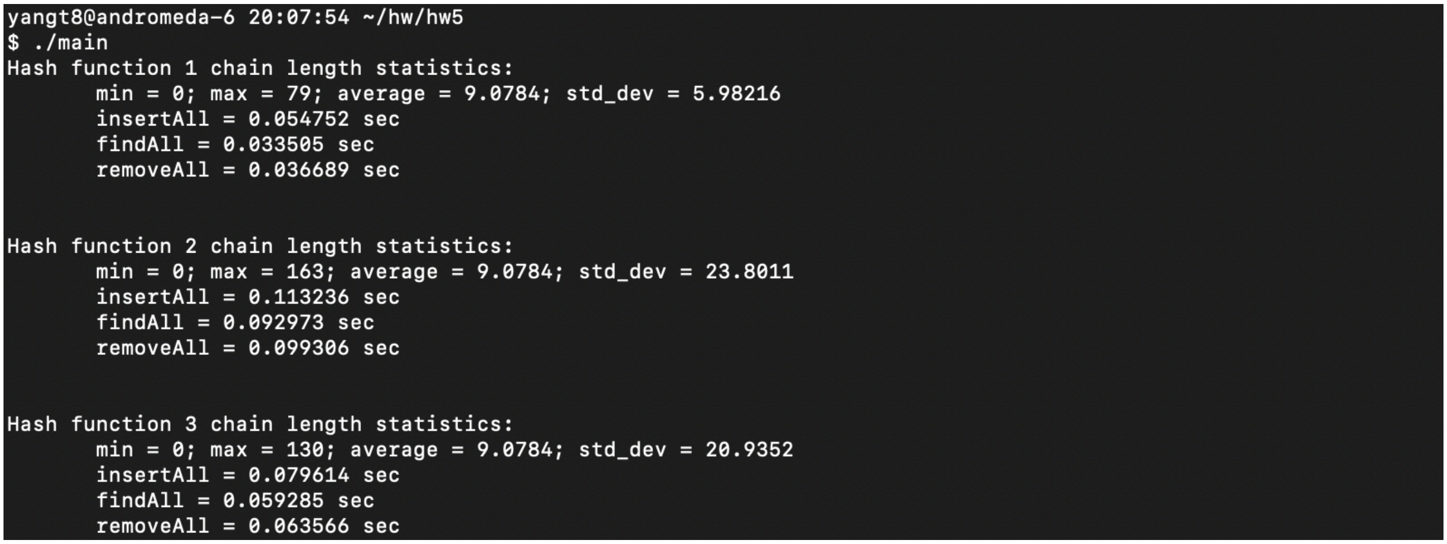
3) Proof of execution under valgrind with no memory leaks (test by the first 4500 words from random.txt) all the words from random.txt



```
yangt8@andromeda-6 20:12:54 ~/hw/hw5
[$ valgrind ./main
==13565== Memcheck, a memory error detector
==13565== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13565== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13565== Command: ./main
==13565==
test GeneralStringHasher
       min = 0; max = 8; average = 0.9; std_dev = 1.1414
       insertAll = 0.259561 sec
       findAll = 0.118969 sec
       removeAll = 0.139989 sec
==13565==
==13565== HEAP SUMMARY:
==13565==     in use at exit: 0 bytes in 0 blocks
==13565==   total heap usage: 31,521 allocs, 31,521 frees, 1,048,064 bytes allocated
==13565==
==13565== All heap blocks were freed -- no leaks are possible
==13565==
==13565== For counts of detected and suppressed errors, rerun with: -v
==13565== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4) Testing output

- 3 console outputs (test by all the words from random.txt)



```
yangt8@andromeda-6 20:07:54 ~/hw/hw5
$ ./main
Hash function 1 chain length statistics:
       min = 0; max = 79; average = 9.0784; std_dev = 5.98216
       insertAll = 0.054752 sec
       findAll = 0.033505 sec
       removeAll = 0.036689 sec


Hash function 2 chain length statistics:
       min = 0; max = 163; average = 9.0784; std_dev = 23.8011
       insertAll = 0.113236 sec
       findAll = 0.092973 sec
       removeAll = 0.099306 sec


Hash function 3 chain length statistics:
       min = 0; max = 130; average = 9.0784; std_dev = 20.9352
       insertAll = 0.079614 sec
       findAll = 0.059285 sec
       removeAll = 0.063566 sec
```

- 2 tables

	random.txt									
	N(number of inputs)									
	4500	9000	13500	18000	22500	27000	31500	36000	40500	45000
insertAll T(N)	0.004694	0.008306	0.012401	0.016282	0.024923	0.027224	0.034621	0.039753	0.046068	0.051625
findAll T(N)	0.002333	0.00489	0.007636	0.010554	0.013704	0.017028	0.020004	0.024507	0.027207	0.033064
remove All T(N)	0.00263	0.005527	0.008573	0.011845	0.015344	0.019023	0.022885	0.026996	0.031535	0.035103

	words.txt									
	N(number of inputs)									
	4500	9000	13500	18000	22500	27000	31500	36000	40500	45000
insertAll T(N)	0.004116	0.008285	0.012351	0.018083	0.022592	0.029206	0.033131	0.037838	0.043744	0.045624
findAll T(N)	0.002447	0.004947	0.00777	0.009539	0.012482	0.016461	0.019674	0.02336	0.026411	0.030047
remove All T(N)	0.002752	0.005567	0.008711	0.011769	0.015076	0.017484	0.021958	0.025612	0.029439	0.033321