情報科学演習 2_第 2 期課題

６３２１１２０　横溝尚也

提出日：６月１４日（水）

この pdf には課題 1 と課題 2 二つのプログラムの説明と実行結果をまとめている。

1 課題 1

1.1 プログラムの説明

プログラムのソースコードを分割して説明していく。

```
-----------------------------------------------------------------------------------------
#include<iostream>
#include<cstring>
#define HASHSIZE 17
using namespace std;
using ll = long long;

class Member_list{
  Member_list* next = NULL;
  char name[16] ;
  ll age ;

public:
  Member_list* operator()(const char* c) const;
  int hash(const char* key) const;
  Member_list* insert(const char* name, const ll age);
  void operator-= (const char* c);
  const char* fetch_name() const;
  void update_name(const char* name);
  ll fetch_age() const;
  void update_age(const ll age);
  Member_list* fetch_next() const;
  void update_next(Member_list* next);
};



Member_list* Member_list::operator() (const char* c) const{
  const Member_list *p = this;
  while(p != NULL){
    if(strcmp(c, p ->name) == 0){
      return const_cast<Member_list*>(p);
    }
    p = p -> fetch_next();
```

```
    }
  return NULL;
}

int Member_list::hash(const char* key) const{
  int hashval = 0;
  while(*key != '\0'){
    hashval += *key;
    key++;
  }
  return hashval % HASHSIZE;
}

Member_list* Member_list::insert(const char* name, const ll age){
  Member_list *ptr = new Member_list;
  ptr->update_name(name);
  ptr->update_age(age);
  ptr->update_next(this);
  return ptr;
}

void Member_list::operator-= (const char* name){
  Member_list *p = this -> fetch_next();
  Member_list *prev = this;
  while(p != NULL){
    if(strcmp(name, p->fetch_name()) == 0){
      prev -> update_next(p -> fetch_next());
      delete p;
      return ;
    }
    prev = p;
    p = p -> next;
  }
}

const char* Member_list::fetch_name() const{
  return this -> name;
}

void Member_list::update_name(const char* name){
```

```
  strcpy(this->name,name);
}

ll Member_list::fetch_age() const{
  return this -> age;
}

void Member_list::update_age(const ll age){
  this -> age = age;
}

Member_list* Member_list::fetch_next() const {
  return next;
}

void Member_list::update_next(Member_list* next){
  this->next = next;
}
```

ここまでは Member_list クラスの記述である。ここではメンバーの名前と年齢がデータとしてあり、それぞれの関数を宣言している。関数のアルゴリズムは c 言語で記述したものと同じである。

```
class Hashtable{
  Member_list* hashtable[HASHSIZE];
  Member_list* fetch_member(int hashval) const;
  void update_member(int hashval, Member_list* member);

public:
  Hashtable();
  Member_list* operator() (const char* c) const;
  int hash(const char* hashkey) const;
  void insert(const char* name, const ll age);
  void operator-= (const char* c);
};

Member_list* Hashtable::fetch_member(int hashval) const {
  return hashtable[hashval] ;
}

void Hashtable::update_member(int hashval, Member_list* member) {
```

```
  hashtable[hashval] = member;
}

Hashtable::Hashtable(){
  for(int i = 0; i < HASHSIZE; i++){
    hashtable[i] = NULL;
  }
}

Member_list* Hashtable::operator() (const char* c) const{
  return fetch_member(hash(c)) -> operator() (c);
}

int Hashtable::hash(const char* hashkey) const{
  int hashval = 0;
  while(*hashkey != '\0'){
    hashval += *hashkey;
    hashkey++;
    }
  return hashval % HASHSIZE;
}

void Hashtable::insert(const char* name, const ll age) {
  int hashval = hash(name);
  if(fetch_member(hashval) == NULL) {
    hashtable[hashval] = new Member_list;
    hashtable[hashval]->update_name(name);
    hashtable[hashval]->update_age(age);
  } else {
    Member_list* new_member = hashtable[hashval]->insert(name,age);
    if (new_member != NULL) {
      update_member(hashval , new_member);
    }
  }
}

void Hashtable::operator-= (const char* c) {
  int hashval = hash(c);
  if (fetch_member(hashval) != NULL) {
    if(strcmp(fetch_member(hashval)->fetch_name(), c) == 0) {
```

```
      Member_list* to_delete = fetch_member(hashval);
      update_member(hashval,  fetch_member(hashval) ->fetch_next());
      delete to_delete;
    } else {
      fetch_member(hashval)->operator-= (c);
    }
  }
}
```

ここまでで Hashtable クラスの記述である。ここでは Hash_list を一つのオブジェクトとして配列に格納している。この際、hash 関数でハッシュ値を計算してどこに格納するか計算している。

```
int main(){
  char word[1024];
  Member_list* ptr;
  Hashtable table;


  table.insert("takimoto",42);
  table.insert("katsurada" ,122);
  table.insert("matsuzawa" , 35);
  table.insert("ohmura" , 12);
  table-="takimoto";


  while(scanf("%s" , word)!= EOF){
    ptr = table(word);
    if(ptr!= NULL){
       cout<<ptr->fetch_name()<< "" <<ptr -> fetch_age()<<endl;
    }
    else{
      cout << "none" << endl;
    }
  }
  return 0;
}
```

メイン関数では今まで記述した関数を使用して、実際にオブジェクトの探索、挿入、削除を行っている。メイン関数の実行結果は下に記述している。

1.2 実行結果

課題１の出力結果が以下である。

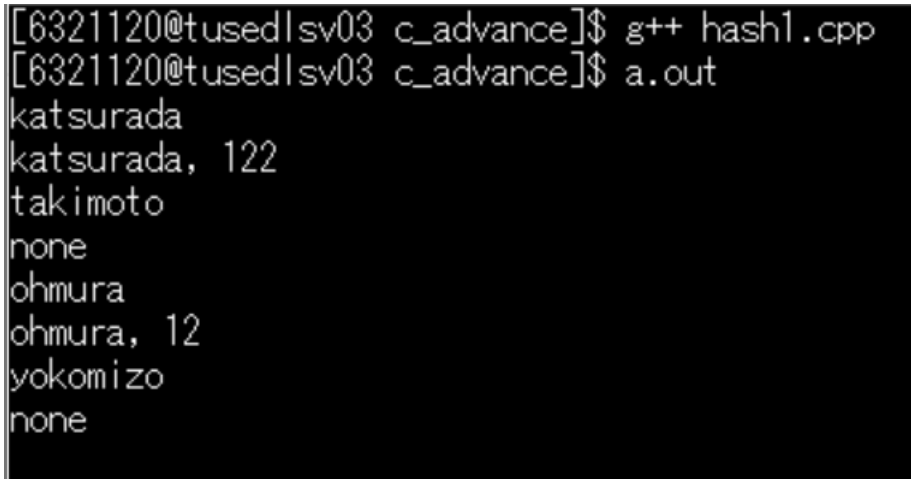


図 1　出力結果

2 課題２

2.1 プログラムの説明

ここでは課題１と違う点のみ記述する。

```
-------------------------------------------------------------------------------------------
#include<iostream>
#include<cstring>
#define HASHSIZE 17
using namespace std;
using ll = long long;


template <typename T>
class Member_list{
  Member_list<T>* next = NULL;
  char name[16] ;
  T age ;

public:
  Member_list<T>* operator()(const char* c) const;
  int hash(const char* key) const;
  Member_list<T>* insert(const char* name, const T age);
  void operator-= (const char* c);
  const char* fetch_name() const;
  void update_name(const char* name);
  T fetch_age() const;
  void update_age(const T age);
  Member_list<T>* fetch_next() const;
  void update_next(Member_list* next);
};


template <typename T>
Member_list<T>* Member_list<T>::operator() (const char* c) const{
  const Member_list<T> *p = this;
  while(p != NULL){
    if(strcmp(c, p ->name) == 0){
      return const_cast<Member_list<T>*>(p);
```

```
    }
    p = p -> fetch_next();
    }
  return NULL;
}

template <typename T>
int Member_list<T>::hash(const char* key) const{
  int hashval = 0;
  while(*key != '\0'){
    hashval += *key;
    key++;
  }
  return hashval % HASHSIZE;
}

template <typename T>
Member_list<T>* Member_list<T>::insert(const char* name, const T age){
  Member_list<T> *ptr = new Member_list<T>;
  ptr->update_name(name);
  ptr->update_age(age);
  ptr->update_next(this);
  return ptr;
}


template<typename T>
void Member_list<T>::operator-= (const char* name){
  Member_list<T> *p = this -> fetch_next();
  Member_list<T> *prev = this;
  while(p != NULL){
    if(strcmp(name, p->fetch_name()) == 0){
      prev -> update_next(p -> fetch_next());
      delete p;
      return ;
    }
    prev = p;
    p = p -> next;
  }
}
```

```
template <typename T>
const char* Member_list<T>::fetch_name() const{
  return this -> name;
}

template <typename T>
void Member_list<T>::update_name(const char* name){
  strcpy(this->name,name);
}

template <typename T>
T Member_list<T>::fetch_age() const{
  return this -> age;
}

template <typename T>
void Member_list<T>::update_age(const T age){
  this -> age = age;
}

template <typename T>
Member_list<T>* Member_list<T>::fetch_next() const {
  return next;
}

template <typename T>
void Member_list<T>::update_next(Member_list<T>* next){
  this->next = next;
}

template <typename T>
class Hashtable{
  Member_list<T>* hashtable[HASHSIZE];
  Member_list<T>* fetch_member(int hashval) const;
  void update_member(int hashval, Member_list<T>* member);

public:
  Hashtable();
  Member_list<T>* operator() (const char* c) const;
```

```
  int hash(const char* hashkey) const;
  void insert(const char* name, const T age);
  void operator-= (const char* c);
};

template <typename T>
Member_list<T>* Hashtable<T>::fetch_member(int hashval) const {
  return hashtable[hashval] ;
}

template <typename T>
void Hashtable<T>::update_member(int hashval, Member_list<T>* member) {
  hashtable[hashval] = member;
}

template <typename T>
Hashtable<T>::Hashtable(){
  for(int i = 0; i < HASHSIZE; i++){
    hashtable[i] = NULL;
  }
}

template <typename T>
Member_list<T>* Hashtable<T>::operator() (const char* c) const{
  return fetch_member(hash(c)) -> operator() (c);
}

template <typename T>
int Hashtable<T>::hash(const char* hashkey) const{
  int hashval = 0;
  while(*hashkey != '\0'){
    hashval += *hashkey;
    hashkey++;
    }
  return hashval % HASHSIZE;
}

template <typename T>
void Hashtable<T>::insert(const char* name, const T age) {
  int hashval = hash(name);
```

```
  if(fetch_member(hashval) == NULL) {
    hashtable[hashval] = new Member_list<T>;
    hashtable[hashval]->update_name(name);
    hashtable[hashval]->update_age(age);
  } else {
    Member_list<T>* new_member = hashtable[hashval]->insert(name,age);
    if (new_member != NULL) {
      update_member(hashval , new_member);
    }
  }
}

template <typename T>
void Hashtable<T>::operator-= (const char* c) {
  int hashval = hash(c);
  if (fetch_member(hashval) != NULL) {
    if(strcmp(fetch_member(hashval)->fetch_name(), c) == 0) {
      Member_list<T>* to_delete = fetch_member(hashval);
      update_member(hashval,  fetch_member(hashval) ->fetch_next());
      delete to_delete;
    } else {
      fetch_member(hashval)->operator-= (c);
    }
  }
}

int main(){
  char word[1024];
  Member_list<float>* ptr; //この２行の型を変える
  Hashtable<float> table;


  table.insert("takimoto",42.2);
  table.insert("katsurada" ,122.3);
  table.insert("matsuzawa" , 35.1);
  table.insert("ohmura" , 12.3);
  table-="takimoto";


  while(scanf("%s" , word)!= EOF){
```

```
    ptr = table(word);
    if(ptr!= NULL){
      cout<<ptr->fetch_name()<< "" <<ptr ->fetch_age()<<endl;
    }
    else{
      cout << "none" << endl;
    }
  }
  return 0;
}
```

---

template ¡typename T¿を記述することで型をパラメータとして一般化している。一般化したい型は年齢 age であるので age を受け取る場所に template ¡typename T¿を記述している。このプログラムによって年齢だけでなく、身長や視力、性別など異なる型でもプログラムを別に作ることなく実現できている。

2.2 出力結果

任意の型で値を収納できていることを表した、int 型、long long 型、double 型の 3 つで main 関数を書き換えたものと、その実行結果を表したものが以下である。ただし、プログラムの説明にもある通り、ソースコードのはじめに long long 型を ll としている。

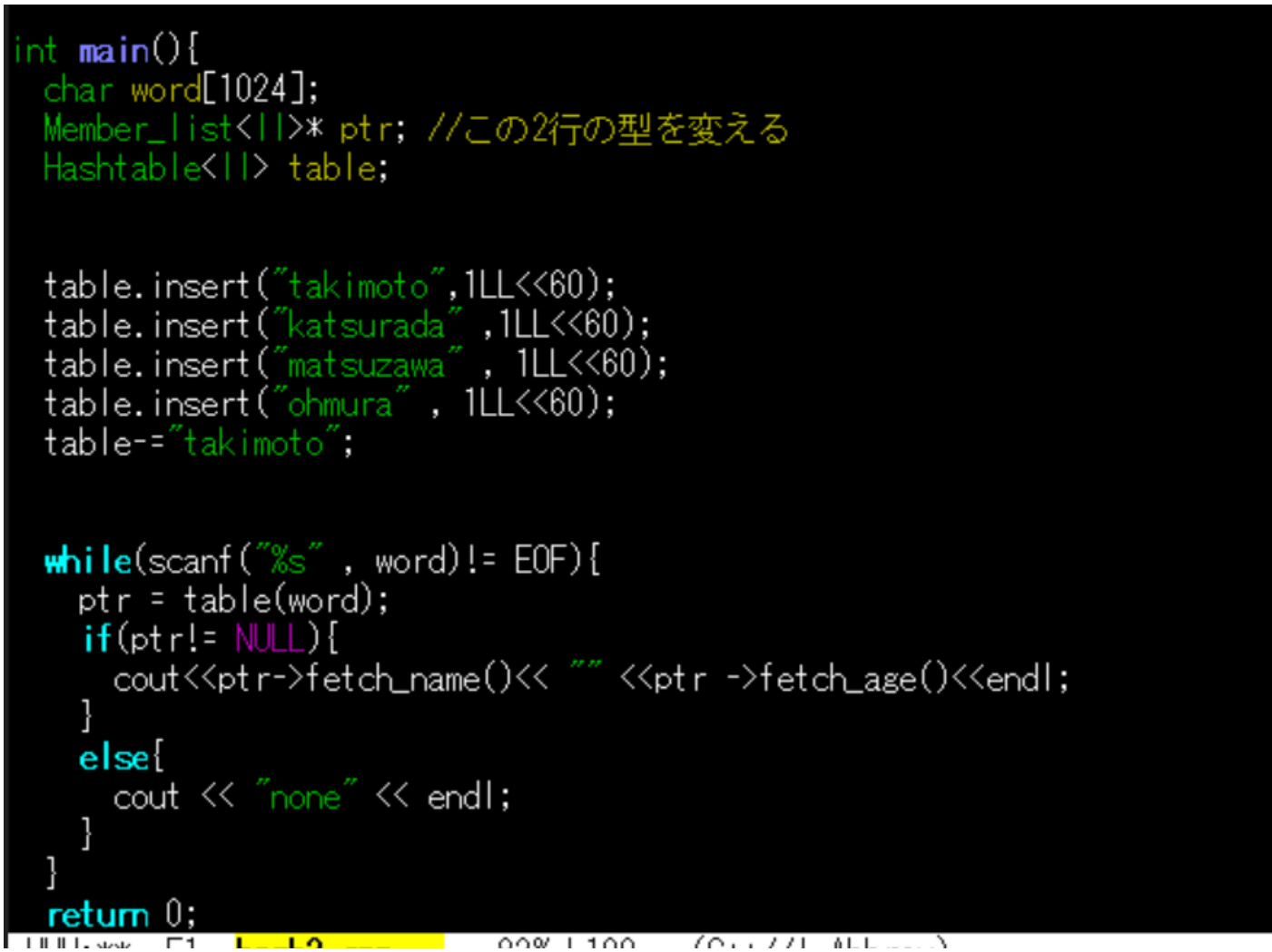
```
int main(){
  char word[1024];
  Member_list<int>* ptr; //この2行の型を変える
  Hashtable<int> table;


  table.insert("takimoto",42);
  table.insert("katsurada" ,122);
  table.insert("matsuzawa" , 35);
  table.insert("ohmura" , 12);
  table-="takimoto";


  while(scanf("%s" , word)!= EOF){
    ptr = table(word);
    if(ptr!= NULL){
      cout<<ptr->fetch_name()<< "" <<ptr ->fetch_age()<<endl;
    }
    else{
      cout << "none" << endl;
    }
  }
  return 0;
}
```

図 2　int 型の main 関数

図 3　int 型の出力結果



```
int main(){
  char word[1024];
  Member_list<ll>* ptr; //この2行の型を変える
  Hashtable<ll> table;


  table.insert("takimoto",1LL<<60);
  table.insert("katsurada" ,1LL<<60);
  table.insert("matsuzawa" , 1LL<<60);
  table.insert("ohmura" , 1LL<<60);
  table-="takimoto";


  while(scanf("%s" , word)!= EOF){
    ptr = table(word);
    if(ptr!= NULL){
      cout<<ptr->fetch_name()<< "" <<ptr ->fetch_age()<<endl;
    }
    else{
      cout << "none" << endl;
    }
  }
  return 0;
```

図 4　long long 型の main 関数内

```
[6321120@tusedlsv03 c_advance]$ emacs hash2.cpp
Display localhost:10.0 unavailable, simulating -nw
[6321120@tusedlsv03 c_advance]$ g++ hash2.cpp
[6321120@tusedlsv03 c_advance]$ a.out
katsurada
katsurada1152921504606846976
takimoto
none
yokomizo
none
ohmura
ohmura1152921504606846976
```

図 5　long long 型の出力結果

```
int main(){
  char word[1024];
  Member_list<float>* ptr; //この2行の型を変える
  Hashtable<float> table;


  table.insert("takimoto",42.2);
  table.insert("katsurada" ,122.3);
  table.insert("matsuzawa" , 35.1);
  table.insert("ohmura" , 12.3);
  table-="takimoto";


  while(scanf("%s" , word)!= EOF){
    ptr = table(word);
    if(ptr!= NULL){
      cout<<ptr->fetch_name()<< "" <<ptr ->fetch_age()<<endl;
    }
    else{
      cout << "none" << endl;
    }
```

図 6　double 型の main 関数内

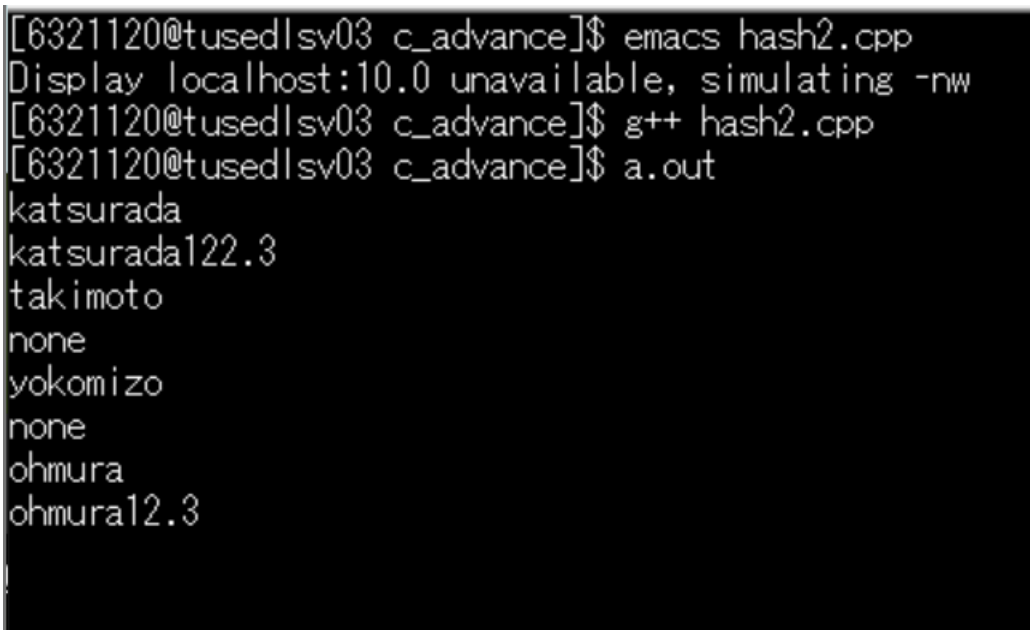


図 7　double 型の出力結果