計算機科学応用演習
C 言語レポート

氏名: 高 健峰
学生番号: 6323041

2025 年 6 月 10 日

目次

1　課題概要

都市データを用いて，後述する実装の項目で指定された関数を記述したプログラムを作成せよ．

1. 都市データファイルからデータを読み込み，リスト構造を用いてデータを作成する関数 add(引数: 都市名，緯度，経度，国名，人口) を作成せよ．ただし，city 構造体のデータは，ハッシュテーブルを用いて探索を効率化すること．また必要に応じて構造体のメンバにポインタ変数を適宜入れること．

2. 作成したハッシュテーブルの分布を調べる関数 dist() を作成せよ．また可能であれば dist() の結果を元に，偏りの少ないハッシュ関数を自作せよ．

3. 標準入力より都市名を入力すると，都市の名前，緯度，経度，国名，人口を表示する関数 print_city(引数: 都市名) を作成せよ．
> 例: print_city(”Noda”);

4. 国名を指定し，その国に存在する都市名を人口順 (多い順) に表示する関数 print_cities(引数: 国名) を作成せよ．
> 例: print_cities(”Japan”);

5. (任意課題)　print_cities() を拡張して，人口順 (0) or 緯度順 (1) or 経度順の指定 (2) と，昇順 (0)or 降順 (1) の指定と表示開始都市名を引数に追加し，指定された並びで都市を表示できるように変更した print_cities2() を作成せよ．
> 例: print_cities2(”Japan”, 0, 1, ”Noda”); この場合は Noda より人口が少ない日本の都市を Noda から順に表示する．

2　アルゴリズム

2.1　データ構造

課題の指示ようり、都市のデータは構造体に保存する。ただし、ハッシュ衝突と解決するため、都市の構造体はデータだけではなく、次の都市の (adress) に指すポインタも保存する。

2.2　ハッシュ関数

今回処理するデータ量が多いなので、ハッシュ関数は「計算速度早い」、「同じの文字列が同じのハッシュ値に生成する」が必要と考える。これより「FNV-1a」ハッシュ関数アルゴリズムを参照して、ハッシュ関数をつくる。
「FNV-1a」アルゴリズムでは初期値 (offset basis) から始め、入力した文字列の各バイトとハッシュ

値を順番に XOR し、その都度特定の素数 (FNV prime) を乗算する。このプロセスにより、入力文字列のわずかな違いが最終的なハッシュ値が大きいな影響を与える、結果としてハッシュ値が広範囲に分散しやすくになる。

例えば、最初のハッシュ値は 11001010, 文字列の第一文字は'C' であるとき
まず XOR を計算する

$$\begin{aligned} &11001010 && (\text{今の}hash\text{値}) \\ &01000011 && (\text{'}C\text{'の値}) \\ &\quad ----------- && \\ =&10001001 && (XOR\text{後の新たな}hash\text{値}) \end{aligned}$$

次にこの結果を設定した素数と乗算して、新たハッシュ値を得る

$$hash = hash * FNV\_prime$$

この新たハッシュ値を使って、次の文字と XOR 計算して、設定した素数と乗算するこのように文字列の最後まで繰り返し計算して、得たハッシュ値がこの文字列のハッシュ値である。

2.3 ハッシュ表とハッシュ表衝突

都市名による検索と国家による検索の要望があるため、都市名と国家によるハッシュテーブル両方が必要である、これより、作った構造体とハッシュテーブルは図１でしめす、ハッシュテーブルの都市構造体の配列であり、index はハッシュ値 % ハッシュテーブルのサイズである

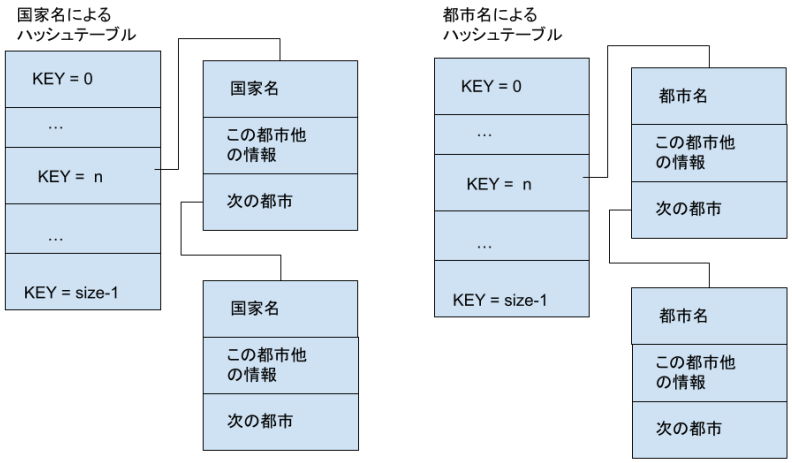

図１　ハッシュテーブルの構造

ここで左側は国家名によるハッシュ値をとる、右側が都市名によるハッシュ値がとる、ハッシュ衝突はどちらもチェンジ法で解決する、衝突があれば、ハッシュテーブルのこの index の先頭に追加する。

2.4 都市を探す

都市の検索は次の順になる

- まず検索したいの都市名を（ハッシュ値 % ハッシュテーブルのサイズ）すなわち、index を計算する。
- 都市による hashtable[index] の先頭の都市の都市名と入力した都市名を比べて。
- 同じならば、検索する都市は該当であり、都市のデータを出力する。
- 違いの場合、同じ都市ま出るまでに、繰り返す次の都市と比べる。
- 最後まで同じの都市名が出ない場合は、検索する都市がない。

2.5 同じ国家の都市を探す

同じ国家の都市を検索する時に、課題４と課題５より、都市の一つのデータを順に並べる必要はあるので、次の三つステップになる

- 国家による hashtable から、同じ国家の都市を抽出する
- 抽出した都市情報を指定するデータを昇順にリスト cityList に保存する
- 指定した順番と開始都市による、cityList の都市データを順番に出力する。

2.6 整列した都市リスト

時間計算量を考慮して、同じ国家の都市は双方向リストに保存する。リストの構造は図２に示す。

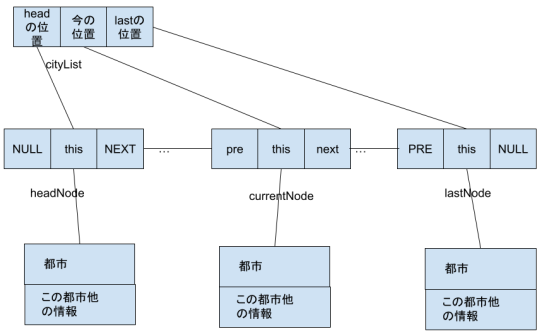

図２　cityList

2.7 データをリストに追加

都市は昇順にリストに追加するので、次の順になる

1. 新たな都市データは newNode となる
2. リストは空であるかとう確認する
   - 空であるとき、新たな都市をに追加して、次の都市を抽出を待ってる。
   - 空でないとき 3 番にいく。
3. リストで保存するデータと新たなデータと比べて
   - List.head.data > new.data の場合で
     new は新たな head になる
   - List.end.data < new.data の場合で
     new は新たな end になる
   - どちらもないの場合で
     List の head からちょうど new.data より大きの Node.data まで、一つずつ探す、この Node.data の左側 (pre) に new を追加する。

2.8 データはしての順番で出力する

指定するように出力するには次のアルゴリズムが使う。

1. 整列したリストから、スタットしたいの都市を探して、StartNode とする.
2. StartNode から
   指定の順番は昇順ならば、右側の Node をすべて出力する
   降順ならば、左側の Node をすべて出力する

3　プログラムの説明

3.1　都市に関する構造体とハッシュ表

```
#define HASHSIZE 101

...

struct city *cityHashtable[HASHSIZE];
...

struct city *countryHashtable[HASHSIZE];

typedef struct city{
    char *cityName; /* 都市名 */
    float lat; /* 緯度 */
    float lng; /* 経度 */
    char *countryName; /* 国名 */
    int pop; /* 人口(population) */
    struct city *nextCity;
    struct city *nextCityinCountry;
}city;
```

上述のコードは都市のデータを保存する構造体とハッシュテーブルを作る。

ここで (#define HASHSIZE) を使って、ハッシュテーブルのサイズが調整やすくなる。

ハッシュテーブルはアルゴリズムの説明ようり、都市名と国家名をキーになる二つハッシュテーブルが使って、すべてのハッシュテーブルは都市構造体の配列である。

```
void print(city *now){
        printf("City Name:%s, lat:%f lng:%f [%s] Population:%d\n",now -> cityName, now
    -> lat, now -> lng,
        now -> countryName, now -> pop);
}
```

上述のコードでは入力した都市構造体の中ですべてデータを出力する関数である。

3.2　ハッシュ関数

```
#define FNVPrime 16777619U
#define FNVOffsetBasis 2166136261U

static int cityTableLoad[HASHSIZE];

```

```
unsigned int hash(char *str){
    int hash = FNVOffsetBasis;
    for(int i = 0; i < strlen(str); i++){
        hash ^= (unsigned int) str[i];
        hash *= FNVPrime;
    }
    return hash;
}
...
```

上述のコードではハッシュ値を計算するハッシュ関数である、アルゴリズムの説明より（FNV-1a）アルゴリズムを使って、入力した文字列各文字列とハッシュ値を XOR 演算して、指定する素数と乗算して、ハッシュ値を返す。
ここで指定する素数 (FNVPrime) と初期ハッシュ値 (FNVOffsetVasis) は (FNV-1a) アルゴリズム 32bit でのおすすめの値である。

3.3 入力データからハッシュテーブルへ

```
void add(char *cityName, float lat, float lng, char *countryName,int pop){
    int cityHashkey = hash(cityName) % HASHSIZE;
    int countryHashkey = hash(countryName) % HASHSIZE;
    city *newcity;
    newcity = malloc(sizeof(struct city));
    newcity -> cityName = malloc(strlen(cityName) + 1);
    strcpy(newcity -> cityName, cityName);
    newcity -> countryName = malloc(strlen(countryName) + 1);
    strcpy(newcity -> countryName, countryName);
    newcity -> lat = lat;
    newcity -> lng = lng;
    newcity -> pop = pop;
    newcity -> nextCity = cityHashtable[cityHashkey];
    cityHashtable[cityHashkey] = newcity;
    cityTableLoad[cityHashkey] ++;
    newcity -> nextCityinCountry = countryHashtable[countryHashkey];
    countryHashtable[countryHashkey] = newcity;
}
```

ここでは入力した都市データからハッシュテーブルに保存するする関数である。入力は順番に、都市の名前、経度、緯度、国家、人口である。

入力からの都市名と国家名をハッシュ関数より、ハッシュキーを計算する。

malloc より、新たな都市構造体のメモリーを確保して、上述ののデータを構造体の各メンバーに代入して、ただし、都市名と国家名は文字列なので strcpy を使って代入する。

代入完了後に、cityHashtable[cityHashkey] の先頭に格納して、次に都市に指すメンバー (nextCity) に元の先頭の都市のアドレスを代入する。また、countryHashtablep[conutryHashkey] についても同じ処理する。ただし、ここで次の都市に指すメンバーは (nextCityinConutry) である

また、ここに各ハッシュテーブルの分布をわかりやすくなるため、都市を追加するとき、このハッシュテーブルの格納情報を記録する整数型配列（cityTableLoad）の同じ index に 1 をたす。

3.4 ハッシュテーブルの格納状況を確認する

```
    void dist(){
    printf("--HASHSIZE: %d--", HASHSIZE);
    float numberOfElement = 0;
    for(int i = 0; i < HASHSIZE; i++){
        printf("Table[%d]:%d\n", i, cityTableLoad[i]);
        numberOfElement += cityTableLoad[i];
    }
    float loadFactor = numberOfElement / HASHSIZE;
    printf("HashLoadFactor:%.2f\n", loadFactor);
    return;
}
```

この関数は課題 2 を指定したハッシュテーブルの分布を調べる関数である。

FOR ループを使って、整数形配列（cityTableLoad）から各 index の値、すなわち、ハッシュテーブルの各 index のチェンジの長さが出力する。

最後にこのハッシュ値が適切かとうかが分析するため、loadFacotr を計算して、各 index の平均負荷率が出力する。

3.5 都市を探す

```
void print_city(char *cityName){
    int cityHashkey = hash(cityName) % HASHSIZE;
    city *now = cityHashtable[cityHashkey];
     while (now != NULL && strcmp(now->cityName, cityName) != 0) {
        now = now->nextCity;
    }
    if (now != NULL) {
        print(now);
    } else {
        printf("No such city named: %s\n", cityName);
    }
    return;
```

```
13 }
```

上述のコードは入力した都市名による、都市のデータが探して出力する関数である。

2 行目で、入力した都市名に対応するハッシュテーブルの index を計算する。

3-11 行目で、アルゴリズムに従って、この index の先頭から順番に探して、入力した都市と同じの都市のデータを出力する.

3.6 整列した CityList の構造と操作関数

3.6.1 CityList の構造

```
1  typedef struct cityList{
2      cities *head;
3      cities *end;
4      cities *current;
5  }cityList;
6
7  typedef struct cities{
8      struct cities *preCityNode;
9      city *this;
10     struct cities *nextCityNode;
11 }cities;
```

アルゴリズムの説明した通り、1-5 行目はリストの首と尾のアドレスと今の Node のアドレスを含む構造体 (cityList) である。

6 − 11 行目は前、次の Node のアドレスと今の指すの都市の構造体のアドレスを含む Node の構造体 (cities) である。

3.6.2 リスト構造体に関する操作関数

```
1  void insertEnd(cities *newNode,cityList *list){
2          newNode -> preCityNode = list -> end;
3          list -> end -> nextCityNode = newNode;
4          list -> end = newNode;
5  }
6
7  void insertHead(cities *newNode,cityList *list){
8      newNode -> nextCityNode = list -> head;
9      list -> head -> preCityNode = newNode;
10     list -> head = newNode;
11     return;
12 }
13
```

```
14 void insertLeft(cities *newNode,cities *currentNode){
15     //modify now node
16     newNode -> preCityNode = currentNode -> preCityNode;
17     newNode -> nextCityNode = currentNode;
18     //modify pre
19     currentNode -> preCityNode -> nextCityNode = newNode;
20     //modify next
21     currentNode -> preCityNode = newNode;
22     return;
23 }
24 void free_cityList(cityList *list) {
25     if (list == NULL) {
26         return;
27     }
28     cities *current = list->head;
29     cities *next;
30     while (current != NULL) {
31         next = current-> nextCityNode;
32         free(current);
33         current = next;
34     }
35     free(list);
36 }
37
38 double getValue(cities* node, int n) {
39     if (node == NULL   node->this == NULL) {
40         return 0.0;
41     }
42
43     switch (n) {
44         case 0:
45             return (double)node->this->pop;
46         case 1:
47             return (double)node->this->lat;
48         case 2:
49             return (double)node->this->lng;
50         default:
51             return (double)node->this->pop;
52     }
53 };
```

上述のコードではリスト構造体を操作する関数である。

1-5 行目は入力したリストの最後に入力した Node を挿入する。

7-12 行目は入力したリストの先頭に入力した Node を挿入する。

14-23 行目は入力した Node の前に、入力した newNode を挿入する。

24-36 行目は入力したリストとその中のすべての Node の確保したメモリを free する関数である、ただし、都市の構造体を確保したメモリが free してない。

38-53 行目では入力した認証子 n により、今の Node を指す都市構造体か対応するメンバーの値が返す関数である。都市のメンバーの値が (int) と (double) があるので、ここですべて double で返す、精度下げるによる間違いがなくなる。この関数は課題指定指定いないが、Node の順番を確認ため役に立つである。

3.6.3 整列挿入関数

```
1  void sortAdd(cities *newCitiNode,cityList *sortedCity,int n){
2      if(sortedCity -> head == NULL){
3          sortedCity -> current = newCitiNode;
4          sortedCity -> head = newCitiNode;
5          sortedCity -> end  = newCitiNode;
6      }
7
8      double new = getValue(newCitiNode, n);
9      double first = getValue(sortedCity -> head, n);
10     double end = getValue(sortedCity -> end, n);
11
12     // new is the smallest case
13     if(new <= first){
14         insertHead(newCitiNode,sortedCity);
15         return;
16     }
17     // new is the biggest case
18     if(new > end){
19         insertEnd(newCitiNode,sortedCity);
20         return;
21     }
22     // new is between case
23     cities *temp = sortedCity -> head;
24     double now = getValue(temp, n);
25     //find the First node bigger than new
26     while (now < new){
27         temp = temp -> nextCityNode;
28         now = getValue(temp, n);
29     }
30     // insert to left
31     if(temp != NULL){
32         //new is smaller than now => add to left
33         insertLeft(newCitiNode, temp);
34         return;
35     }
```

```
36 };
```

上述のコードはアルゴリズムに従って、入力した新たな Node（newNode）を現在のリストに昇順に追加する。

2 − 6 行目はリストいま head がない、すなわち、リストは空であるとき、そのまま新たな Node を挿入し、head と end を更新する。
8 − 10 と 24 行目はコードを読みやすくなるため、新たな Node のデータを new, 頭のデータは head, 最後のデータは end, いまの Node のデータは now とする

次に場合わけて処理する

12-16 行目で new は頭より小さいのとき、insertHead を使って、newNode を先頭に挿入し、新たな先頭になる。

17-21 行目で new は最後より大きいな場合で、insertEnd を使って、newNode の最後に挿入する。

22-35 行目で new は最大と最小ではない場合で、newNode を先頭からの Node 順に比べて、ちょっど大きいな Node が探すのとき、insertLeft を使って、この Node の前に挿入する。

3.6.4 リストを作る関数

```
1  cityList* sortCities(char *countryName, int dataType){
2      int cityHashkey = hash(countryName) % HASHSIZE; //find buscket
3      city *nowCity = countryHashtable[cityHashkey];
4
5      //ini sortcitylist
6      cityList *sortedCity = malloc(sizeof(cityList));
7      sortedCity -> current = NULL;
8      sortedCity -> head = NULL;
9      sortedCity -> end = NULL;
10
11     while (nowCity != NULL)
12     {
13         if(strcmp(nowCity -> countryName, countryName) == 0){  //if nowcity is in this
       country
14             cities *newCitiNode = malloc(sizeof(cities));
15             newCitiNode -> this = nowCity;
16             newCitiNode -> preCityNode = NULL;
17             newCitiNode -> nextCityNode = NULL;
18             // add new citynode to sorted list
19
20             sortAdd(newCitiNode, sortedCity, dataType); //sort by list
21         }
```

```
22          nowCity = nowCity -> nextCityinCountry;
23      }
24
25      return sortedCity;
26  }
```

上述のコードは指定する国家名と並びのデータの指定子を入力して、整列したリストを作って、返す関数である。

2-3 行目は、入力した国家をハッシュ関数よりハッシュ値を計算して、これハッシュ値に対するハッシュテーブルの index の先頭のとしを nowCity とする。

6-9 行目は新たなリストのメモリーを確保して、初期化とする。

11 – 23 行目ではこのリンクリストの中に同じ国家名が確認したら（13 行目)、この都市をひとつ Node として（14 – 17 行目)、リストに順番に追加する（20 行目)。次のハッシュチェンジの次の都市に移動して、先の操作が繰り返し（22 行目)。

25 行目では都市探す完了ならば、そのリストを整列したリストとして返す。

3.7 指定するように出力する

```
1   void print_cities(char *countryName){
2       cityList *sortedCity = sortCities(countryName, 0); //sort by 0 = pop
3       if (sortedCity == NULL   sortedCity->head == NULL) {
4           printf("No cities found for country: %s\n", countryName);
5           if (sortedCity != NULL) {
6               free_cityList(sortedCity);
7           }
8           return;
9       }
10
11      sortedCity -> current = sortedCity -> end;
12      while (sortedCity -> current != NULL){
13          printf("test");
14          print(sortedCity -> current -> this);
15          sortedCity -> current = sortedCity -> current -> preCityNode;
16      }
17
18      free_cityList(sortedCity);
19      return ;
20  }
```

上述のコードは課題 4 の指示より国家名を指定し、その国に存在する都市を人口多い順に表示する関数である。

2 行目で、国家による整列したリストを人口より整列したリストを（sortCities）関数で作る.

3 – 9 行目で、このリストは空である場合で、この国家の都市が見つけないことを出力して、このリスト確保したメモリを free して関数終わり.

11-16 行目で、このリストの最後から先頭までに一つずつ都市の情報を出力する。

17 行目で、このリストを確保したメモリを FREE する。

```
1  void print_cities2(char *countryName, int dataType, int order, char *startCity){
2      cityList *sortedCity = sortCities(countryName, dataType);
3
4      if (sortedCity == NULL   sortedCity->head == NULL) {
5          printf("No cities found for country: %s\n", countryName);
6          if (sortedCity != NULL) {
7              free_cityList(sortedCity);
8          }
9          return;
10     }
11
12     //find startNode
13     cities *startNode = NULL;
14     if(startCity == NULL   strcmp(startCity, "0") == 0){
15         if(order == 0){
16             startNode = sortedCity -> head;
17         }else{
18             startNode = sortedCity -> end;
19         }
20     }else{
21         cities *curNode = sortedCity -> head;
22         while (curNode != NULL){
23             if(strcmp(curNode -> this -> cityName, startCity) == 0){
24                 startNode = curNode;
25                 break;
26             }
27             char *str = curNode-> this -> cityName;
28             curNode = curNode -> nextCityNode;
29         }
30     }
31
32     //print by order
33     if(startNode == NULL){
34         printf("No such city named:%s\n", startCity);
35     }else{
36         cities *current = startNode;
37         if(order == 0){// small to big
```

```
38              while (current != NULL){
39                  print(current -> this);
40                  current = current -> nextCityNode;
41              }
42          }else if(order == 1){// big to small
43              while (current != NULL){
44                  print(current -> this);
45                  current = current -> preCityNode;
46              }
47          }else{
48              printf("order error\n");
49          }
50      }
51      free_cityList(sortedCity);
52      return ;
53  }
```

上述のコードは課題５により入力するように都市の情報が出力する関数である。

アルゴリズムに従って：

2 行目で、国家による整列したリストを入力した指定子より整列したリストを（sortCities）関数で作る.

4-9 行目で、このリストは空である場合で、この国家の都市が見つけないことを出力して、このリスト確保したメモリを free して関数終わり.

13-30 行目で、開始する Node をリストから探す、(14-19 行目）開始する都市指定しない時に、開始 Node は順番より、先頭または最後となる。つぎに（20-30 行目）リストの先頭から、指定する Node までに順番に探して、開始 Node となる, 見つけない場合そのまま Null となる。

33-50 行目で開始 Node から順番に出力する。まず、(33-35 行目) 開始 Node 見つけない場合、見つけないの情報を出力する。(36-47 行目）開始 Node から、順番の指定子より、０ならば、昇順に、１ならば、降順に先頭または最後までに順番に出力する。(47-49 行目) 指定子がそれ以外のときはエーラメッセージを出力する。

51-52 行目で、リストを Free して関数終わり。

3.8 main 関数とプログラムのメニュー

```
1  void sysMenuJp(){
2      printf("============================\n");
3      printf("数字を入力して、プログラムを起動する.\n");
4      printf("[1] ハッシュテーブルの分布を示す\n");
```

```
    printf("[2] 都市の情報を示す\n");
    printf("[3] 国に存在する都市を示す\n");
    printf("[4] 国に存在する都市を指定の形式で示す\n");
    printf("EOFを入力するプログラムを閉じる\n");
    printf("============================\n");
    return;
}
```

上述のコードでは自作のプログラムのメニューである。

```
int main(){
    struct timespec start, end;
    long long elapsed_ns;
    double elapsed_ms;

    char name[1024];
    char country[1024];
    float lat, lng;
    int pop, i, f;
    char *filename = "worldcities.txt";
    FILE *fp;

    for(int i = 0; i < HASHSIZE; i++){
        countryHashtable[i] = NULL;
        cityHashtable[i] = NULL;
        cityTableLoad[i] = 0;
    }
    fp = fopen(filename,"r");
    while((f = fscanf(fp,"%s %f %f %s %d",name,&lat,&lng,country,&pop))!=EOF){
        add(name, lat, lng, country, pop);
    }
    fclose(fp);
    sysMenuJp();
    char buffer[1024];
    int bufferint;
    while(1){
        int scanResult = scanf("%d", &bufferint);
        if(scanResult == EOF){
            printf("EOFを検出する、プログラムを閉じる\n");
            free_all_memory();
            break;
        }else if (scanResult == 1){
            switch (bufferint){
                case 1:
                    dist();
                    break;
                case 2:
                    printf("探したいの都市は？ \n");
                    scanf("%s", buffer);
```

```
40                      clock_gettime(CLOCK_MONOTONIC, &start);
41                      print_city(buffer);
42                      clock_gettime(CLOCK_MONOTONIC, &end);
43                      elapsed_ns = (end.tv_sec - start.tv_sec) * 1000000000LL + (end.
    tv_nsec - start.tv_nsec);
44                      elapsed_ms = elapsed_ns / 1000000.0;
45                      printf("探す時間:  %.3f (ms)\n", elapsed_ms);
46                      break;
47                  case 3:
48                      printf("探したいの国は？ \n");
49                      scanf("%s", buffer);
50                      clock_gettime(CLOCK_MONOTONIC, &start);
51                      print_cities(buffer);
52                      clock_gettime(CLOCK_MONOTONIC, &end);
53                      elapsed_ns = (end.tv_sec - start.tv_sec) * 1000000000LL + (end.
    tv_nsec - start.tv_nsec);
54                      elapsed_ms = elapsed_ns / 1000000.0;
55                      printf("探す時間:  %.3f (ms)\n", elapsed_ms);
56                      break;
57                  case 4:
58                      int dataType,order;
59                      char cityName[1024] = "";
60                      printf("探したい国の名前は？ \n");
61                      scanf("%s", buffer);
62                      printf("並びたい順番を選択してください：\n[0]人口\n[1]緯度\n[2]経度\n");
63                      scanf("%d", &dataType);
64                      while (dataType != 1 && dataType != 2 && dataType != 0)
65                      {
66                          int c;
67                          while ((c = getchar()) != '\n' && c != EOF);
68                          printf("入力が間違います、再度に入力してください\n");
69                          printf("並びたい順番を選択してください：\n[0]人口\n[1]緯度\n[2]経度\n
    ");
70                          scanf("%d", &dataType);
71                      }
72                      printf("並び順番を選択してください：\n[0]昇順\n[1]降順\n");
73                      scanf("%d", &order);
74                      while (order != 1  && order != 0)
75                      {
76                          int c;
77                          while ((c = getchar()) != '\n' && c != EOF);
78                          printf("入力が間違います、再度に入力してください\n");
79                          printf("並びたい順番を選択してください：\n[0]人口\n[1]緯度\n[2]経度\n
    ");
80                          scanf("%d", &order);
81                      }
82                      printf("開始したいの都市ファイルを入力くださ（指定しないのときを0を入力し
    て、すべての都市が表示する）：\n");
83                      scanf("%s", cityName);
```

```
84                      clock_gettime(CLOCK_MONOTONIC, &start);
85                      print_cities2(buffer, dataType, order,cityName);
86                      clock_gettime(CLOCK_MONOTONIC, &end);
87                      elapsed_ns = (end.tv_sec - start.tv_sec) * 1000000000LL + (end.
    tv_nsec - start.tv_nsec);
88                      elapsed_ms = elapsed_ns / 1000000.0;
89                      printf("探す時間:  %.3f (ms)\n", elapsed_ms);
90                      break;
91                  default:
92                      printf("入力が間違います、再度に入力してください。\n");
93                      break;
94                  }
95              }else{
96              printf("入力が間違います、再度に入力してください。\n");
97              int c;
98              while ((c = getchar()) != '\n' && c != EOF);
99          }
100         sysMenuJp();
101     }
102     return 0;
103 }
```

上述のコードでは main 関数である。

6-22 行目で、都市のデータを file から読み、アルゴリズムで記述した二つのハッシュテーブルに保存する。

23-102 行目で、メニューを出力して、入力を続き読み込み、入力によって、様々の関数を呼び出すである。EOF を入力するときすべてのテータを確保したメモリを free してプログラムが終わる。

また、探すアルゴリズムの効率を考察するために毎回探す操作開始直前と直後に L inux の api（clock_gettime()）を使って、操作の時間がＭＳ単位として計算して、出力する。

4　実行結果

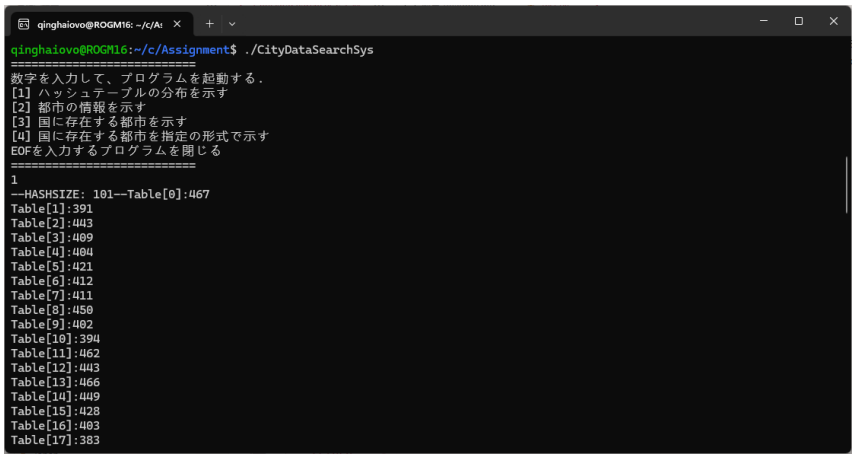

図 3　課題 1

図 4　課題 1 続き

図 3、図 4 は課題 1 の実行結果が示す。

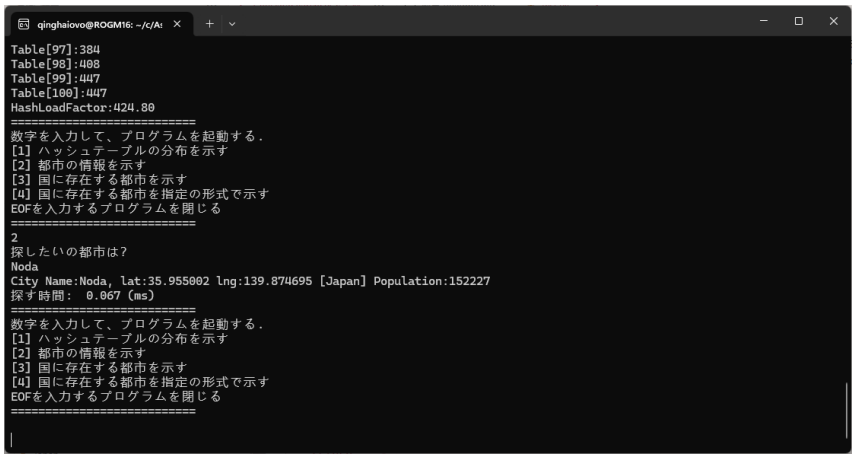

図５　課題２

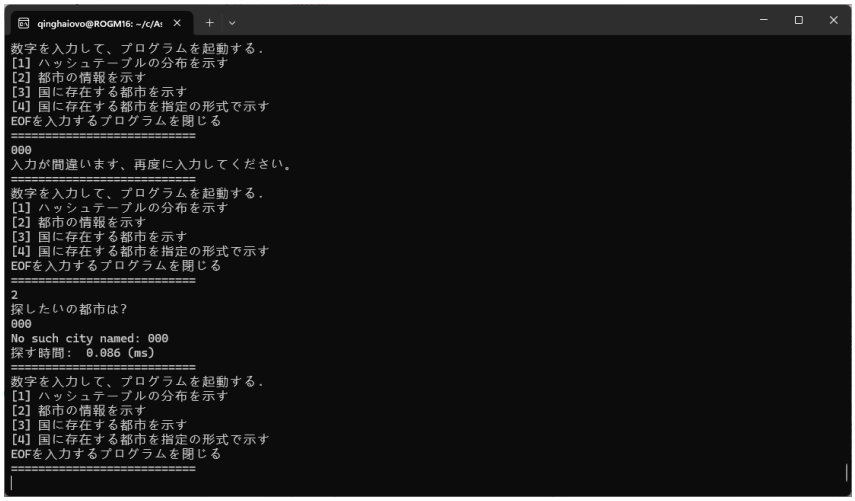

図６　課題２例外処理

図５、６は課題２と例外処理の実行結果が示す。

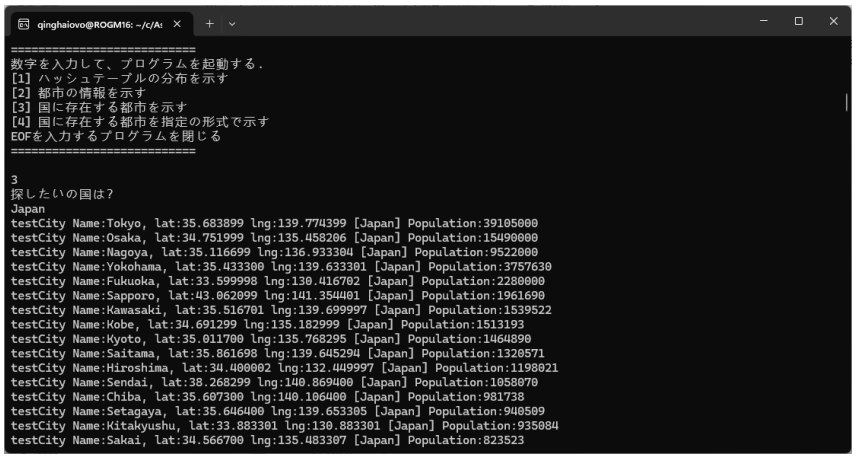

図 7　課題 3

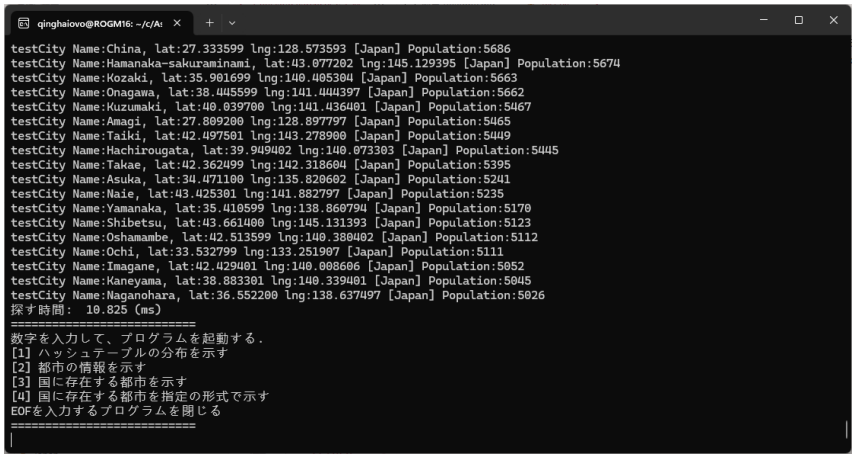

図 8　課題 3 続き

図 9　課題 3 例外処理

図 7 − 9 は課題 3 と例外処理の実行結果が示す。

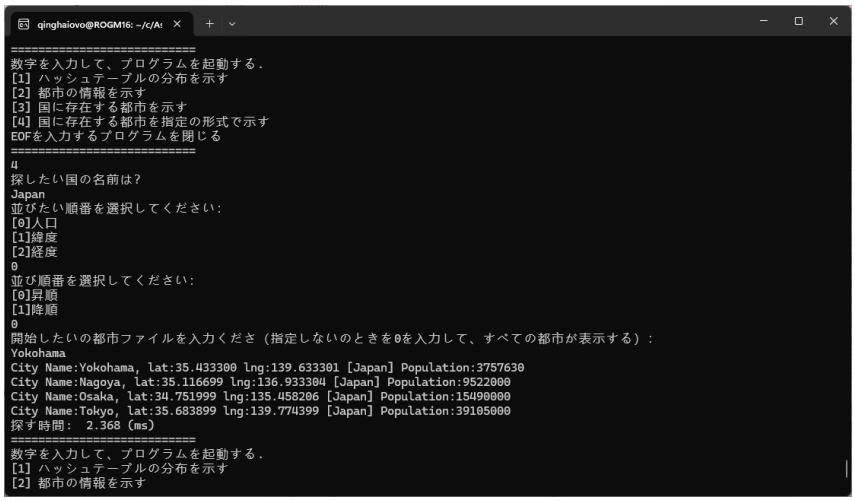

図 10　課題 4 例 1

図 11　課題 4 例 2

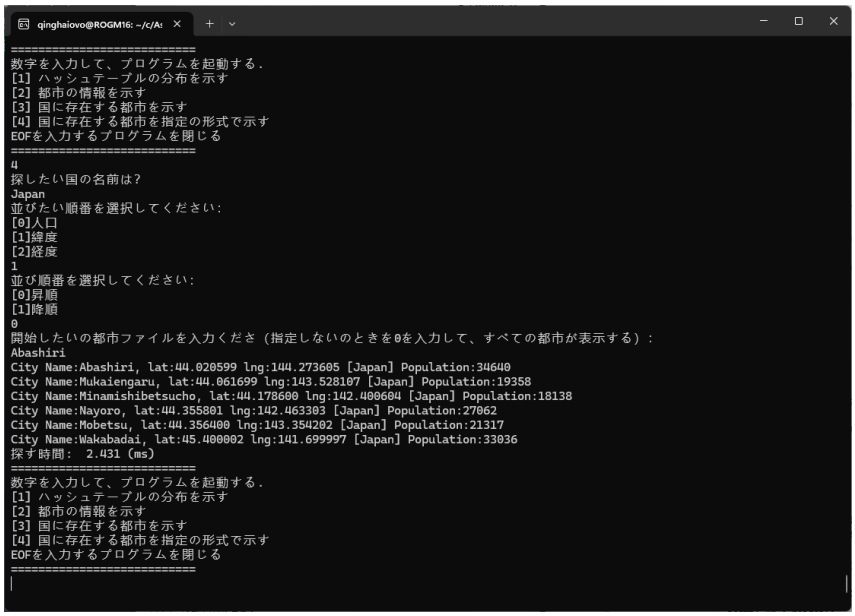

図 12　課題 4 例 3

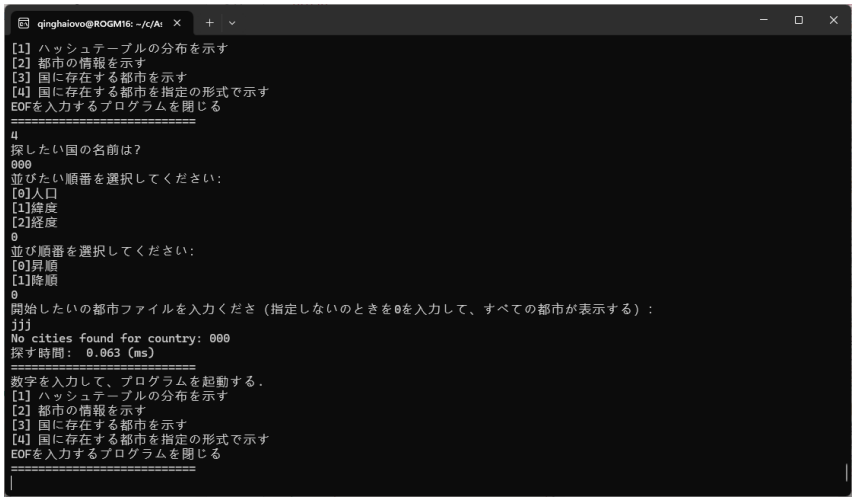

図 13　課題 4 例外処理 1

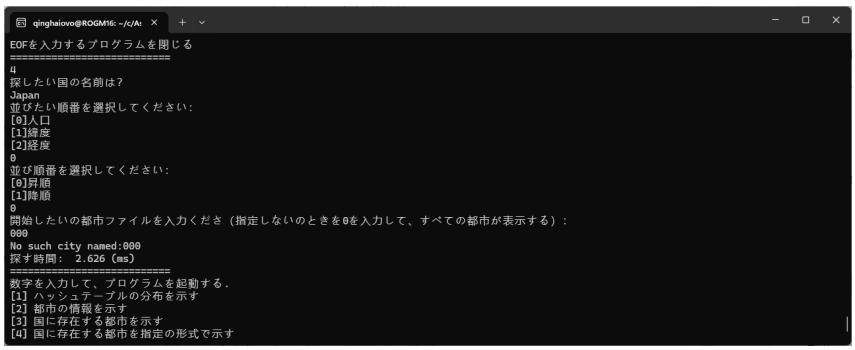

図 14　課題 4 例外処理 2

図 10 から図 14 までは課題 4 の三つの例と例外処理二つの実行結果が示す。

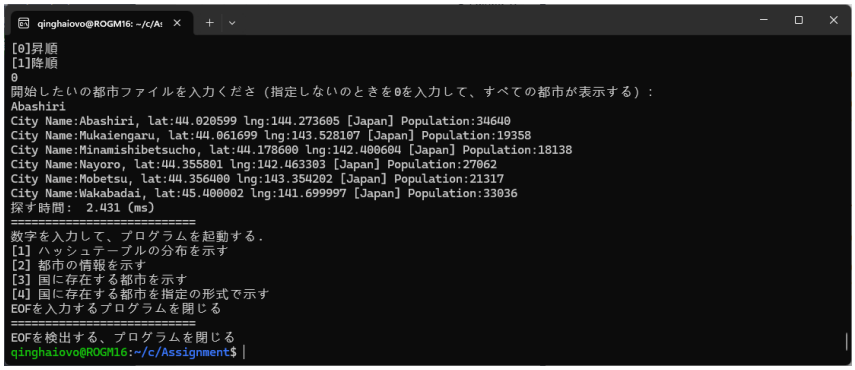

図 15　プログラム終了

図 15 はプログラム終了機能が示す

5　考察

実行結果を示した通り、プログラムが予想通りに動ける。つぎに考察で説明する。

5.1　アルゴリズムに関する考察

１．ハッシュテーブルのサイズが素数となると、、ハッシュ衝突を減らし、データをテーブル全体に均等に分散させる。素数は自身以外に約数を持たないため、ハッシュ値と公倍数があることがすくない、それより偏りが少なくなる。

２．ハッシュ関数の作り方がいくつか考えますが、最初は単に各文字のコードとある素数と乗算して、モジュールする、ただし、この方法では偏りが多いなので、ネットで調べて、今回の FNV-1a アルゴリズムを実装しました。

３．課題４と５はプログラムを構想するときに、最初はリストではなくで、配列をポインタは保存して、整列して出力する。しかし、各国家の都市の数が違い、確認できないのため、両方向リストを作って利用する。またリストの機能に関しては、Ｃ＋＋のプログラムした結構利用したリストの関数をまねして作った。

４．検索にかかるコストはｍｓを使って、出力する時に表示する。元 C 言語 (time.h)lib の clock() 関数を利用したが、精度が低いなので、Linux の clock_gettime を利用になった。

5.2　優れた点と改善できる点

今回のプログラムで優れた点が以下になる

- 両方向リストを使って、計算時間を減らす
- 標準出入力システムで、メニューシステムも作る
- できる限り抽象化したので、プログラムの変更と理解やすくなる
- メモリの管理が厳密に行う

悪い点と改善できる点

- 計算時間が短いですが、ハッシュテーブル二つとリスト一つを使ってので、メモリの使用量が多い
- 繰り返したコードを抽象化すると、プログラム綺麗になれると考える。

5.3　提出したファイル

- $CityDataSearchSys.c$ 計算時間が含むソースコード (Linux また Macos でコンパイラ必要)
- $CityDataSearchSys\_Nodebug.c$ 計算時間が含まないソースコード

- $test.txt$Linux システム Script に記録したプログラムの実行結果