計算機工学期末

ハッシュにチェイン実装

HashRecordに次のアドレスを追加する

```
typedef struct
{
  char key[64];
  char value[64];
  int state;
} HashRecord
```

これを

```
typedef struct __node
{
  char key[64];
  char value[64];
  int state;
  struct __node  *next;
} HashRecord;
```

こう

挿入時に被った際にリスト最後尾に追加する

```
int HashAdd(Hash *hash, char *key, char *value)
{
  int code = HashCode(hash, key);
  while (hash->table[code].state == FULL) {
    code = (code + 1) % hash->size;
    if (code == HashCode(hash, key))return FALSE;
  }
  strcpy(hash->table[code].key, key);
  strcpy(hash->table[code].value, value);
  hash->table[code].state = FULL;
  return TRUE;
}
```

これを

```
int HashAdd(Hash *hash, char *key, char *value)
{
  int code = HashCode(hash, key);
  if(hash->table[code].state == FULL) {
    HashRecord *lhash = &hash->table[code];
    while(lhash->next != NULL){
      lhash = lhash->next;
    }
    HashRecord *nhash  = malloc(sizeof(HashRecord));
    lhash->next = nhash;
    strcpy(nhash->key, key);
    strcpy(nhash->value, value);
    return TRUE;
  }
  strcpy(hash->table[code].key, key);
  strcpy(hash->table[code].value, value);
  hash->table[code].state = FULL;
  return TRUE;
}
```

こう
これで各HashRecordにリストが連なっていく

HashDumpでリストも表示できるようにする

デバッグするためであって必要はないと思う

```
void HashDump(Hash * hash)
{
  int i, h;

  for (i = 0; i < hash->size; i++)
  {
    printf("%4d: ", i);

    switch (hash->table[i].state)
    {
      case FULL:
        h = HashCode(hash, hash->table[i].key);
        printf("(%s[%d], %s)", hash->table[i].key, h, hash->table[i].value);
        break;
      case EMPTY:
        printf("empty");
        break;
      case DELETED:
        printf("deleted");
        break;
      default:
        printf("unknown");
        break;
    }
    printf("\n");
  }
}
```

これを

```
void HashDump(Hash * hash)
{
  int i, h;

  for (i = 0; i < hash->size; i++)
  {
    printf("%4d: ", i);

    switch (hash->table[i].state)
    {
      case FULL:
        h = HashCode(hash, hash->table[i].key);
        printf("(%s[%d], %s)", hash->table[i].key, h, hash->table[i].value);
        HashRecord *phash = &hash->table[i];
        while(phash->next != NULL){
          phash = phash->next;
          printf("\n%4d: ", i);
          printf("(%s[%d], %s)", phash->key, h, phash->value);
        }
        break;
      case EMPTY:
        printf("empty");
        break;
      case DELETED:
        printf("deleted");
        break;
      default:
        printf("unknown");
        break;
    }

    printf("\n");
  }
}
```

こう
同じハッシュのものを下に連ねて表示できる

```
?> p
p
   0: (A[0], A)
   0: (AA[0], B)
   0: (AAA[0], C)
   1: empty
   2: empty
```

こんな感じ

HashDeleteをリストまで探索して削除できるようにする

```
int HashDelete(Hash *hash, char *key)
{
  int code = HashCode(hash, key);
  while (strcmp(hash->table[code].key, key) != 0) {
    code = (code + 1) % hash->size;
    if (code == HashCode(hash, key))return FALSE;
  }
  hash->table[code].state = DELETED;
  return TRUE;
}
```

これを

```
int HashDelete(Hash *hash, char *key)
{
  int code = HashCode(hash, key);
  if(strcmp(hash->table[code].key, key) != 0){
    HashRecord *lhash = &hash->table[code];
    HashRecord *rhash;
    while(strcmp(lhash->key, key) != 0){
      if(lhash->next == NULL)return FALSE;
      rhash = lhash;
      lhash = lhash->next;
    }
    rhash->next = lhash->next;
    free(lhash);
    return TRUE;
  }
  hash->table[code].state = DELETED;
  return TRUE;
}
```

こう

```
?> p
p
   0: (A[0], A)
   0: (AA[0], B)
   0: (AAA[0], C)
   0: (AAAA[0], D)
   1: empty
   2: empty
?> d
誰を削除しますか?: AAA
d -> AAA
AAAを削除しました.
?> p
p
   0: (A[0], A)
   0: (AA[0], B)
   0: (AAAA[0], D)
   1: empty
   2: empty
?>
```

特に問題なく動いている

HashGetをリストまで探索して出力できるようにする

```
int HashGet(Hash *hash, char *key, char *value)
{
  int code = HashCode(hash, key);
  while (strcmp(hash->table[code].key, key) != 0) {
    code = (code + 1) % hash->size;
    if (code == HashCode(hash, key))return FALSE;
  }
  strcpy(value, hash->table[code].value);
  return TRUE;
}
```

これを

```
int HashGet(Hash *hash, char *key, char *value)
{
  int code = HashCode(hash, key);
  if(strcmp(hash->table[code].key, key) != 0){
        HashRecord *lhash = &hash->table[code];
        while(strcmp(lhash->key, key) != 0){
          if(lhash->next == NULL)return FALSE;
          lhash = lhash->next;
        }
        strcpy(value, lhash->value);
        return TRUE;
      }
  strcpy(value, hash->table[code].value);
  return TRUE;
}
```

こう

```
p
   0: (A[0], A)
   0: (AA[0], B)
   0: (AAA[0], C)
   1: empty
   2: empty
?> g
名前を入力して下さい: AA
g -> AA
AAの血液型はBです.
?> g
名前を入力して下さい: AAA
g -> AAA
AAAの血液型はCです.
?>
```

問題なく動いている。

おきもち

リストにするだけなので特別何かを考えたり工夫することなく実装することが出来ました。
改変した関数も出来るだけ変更を少なくしたので、元々のプログラムと同じくらいの可読性はあると思います。