Hashing

Öğrenci Adı: Burak Atalay

Öğrenci Numarası: 22011641

Dersin Eğitmeni: M. Elif Karslıgil

Video Linki: https://youtu.be/k19jL1DhQ4U

1- Problemin Çözümü:

Dosya bir stringe okunduktan sonra bu string ‘;’ karakterine göre satırlara ayrılıyor ve satırlar 2 boyutlu bir diziye kaydediliyor. Daha sonrasında kaç adet deklare edilme ihtimali olan değişken olduğu sayılıp bu sayının iki katından sonraki asal sayı m olarak seçiliyor ve map yaratılıyor. Sonrasında her bir satır derleneceği fonksiyona gönderilip derleniyor. Satırın derlemesinin yapıldığı fonksiyonda tokenlama için kendi yazdığım bir fonksiyon kullanıldı. Buna ihtiyaç duyulma sebebi hangi delimiter’a denk gelindiğini bilme ihtiyacıydı. strtok delimiter’ı söylemiyor. Kendi yazdığım fonksiyon delimiter görene kadarki stringi parametre olarak verilen stringe kopyalayıp delimiter’ı return ediyor. İlk tokenı bu fonksiyon yardımıyla aldıktan sonra token char, int veya float ise deklare edilme satırı demektir. Yani ilk token bunlardan biriyse bir if bloğuna değilse bir else bloğuna giriliyor. If bloğuna girdikten sonra satırın sonuna gelene kadar tokenlama yapılıyor. Token alınınca keywordden sonra gelen ilk token hariç sonrakilerde eğer kendinden önce virgül vardı idiyse deklare etme durumu ama virgül yoktu ise ‘=’, ‘+’ gibi delimiterlar var demektir ve bu deklare etme değil kullanım durumudur. Bu, custom fonksiyon yardımıyla edindiğimiz delimiter’a bakarak döngünün sonunda canBeDeclared değişkeni set edilierek yapılıyor. Eğer token valid bir değişken ismi ise ve canBeDeclared 1 ise daha önce deklare edilip edilmediğine bakmak için lookup fonksiyonu çağrılıyor. Fonksiyon bulduysa 1, bulmadıysa 0 döndürüyor. Eğer 1 gelirse daha önce deklare edildiğine dair bir hata mesajı yazdırılıyor. 0 ise insert fonksiyonu çağrılıyor. Eğer canBeDeclared 0 ise kullanım durumu olduğu için lookup fonksiyonu çağrılıyor ve eğer 0 dönerse kullanılan bu değişkenin daha önce deklare edilmediğine dair hata mesajı yazdırılıyor. Eğer valid bir değişken ismi değilse ve deklare etme için bir keyword ise (int, char, float) C derleyicisinin yaptığı gibi bu keyword öncesi ‘;’ veya ‘(‘ delimiter’ı beklendiğine dair bir hata yazdırılıp bir değişkeni set ederek döngüden çıkılıyor ve bu satırı derleme işlemi durduruluyor. Eğer bunlar da değilse ve numara veya diğer keywordler (if, else, for gibi) de değilse değişken ismi invalid diye bir hata mesajı gösteriliyor. Satırın ilk tokenı deklare etme keyword’ü değil iken girilen else bloğunda ise sadece kullanım durumları olarak bir değerlendirme var. Yani görülen delimiter önemli değil. Satır bitene kadar bir döngüde dönülüyor ve her bir token için eğer valid bir değişken ismi ise lookup çağrılıyor ve dönen değer 0 ise kullanılan değişken daha önce deklare edilmedi diye bir hata mesajı döndürülüyor. If bloğunda yapıldığı gibi eğer deklare etme keyword’ü görülürse bu keyword öncesi ‘;’ veya ‘(‘ delimiter’ı beklendiğine dair bir hata yazdırılıp döngüden çıkılıyor. Eğer for, if gibi keyword’ler veya bir numara da değilse C derleyicisinde olduğu gibi bilinmeyen değişken tipi diye bir mesaj yazdırılıyor. Her satırın derlenme işlemi bitince eğer seçilen mode debug ise tablo uzunluğu, deklare edilen değişken sayısı ve tablonun son hali yazdırılıyor. Her hücre için ilk hesaplanan adres değeri ve yapılan probe sayısı da gösteriliyor. Bu son işlemi tekrardan name için h1 ve h2 hesaplanıp O(1) karmaşıklıkta basit bir aritmetik ile probe sayısı hesaplanıyor.
Hash fonksiyonu için de horner methodunda döndürülen key değerinin m ile modunu almak yerine her iterasyonda m ile mod aldım. Bu modun özelliğinden dolayı sonucu değiştirmiyor ama key değerini m’den

2- Karşılaşılan Sorunlar:

Karşılaşılan ilk sorun hash fonksiyonunun nasıl implemente edileceğiydi. Horner metodundan döndürülen key değerinin m ile modunu alabilirdim. Ancak key uzun stringler için çok büyük hale geliyordu ve long long tipi ile tutmak gerekiyordu. Ayrıca büyük sayıların çarpımı aritmetik olarak birazcık daha uzun sürdüğünden fazla değişkenli ve uzun stringler içeren bir input için bu küçük fark büyüyebileceğinden horner’in her adımında m ile mod aldım. Modun özelliğinden dolayı

sonuç değişmeyeceğinden ve key 0-m aralığında tutulacağından dolayı bunu tercih ettim. Ancak keyi bir kez hesaplayıp h1 ve h2'yi birer mod işlemi ile hesaplamak yerine ikisi için ayrı olarak hesaplama durumu oluştu. Ancak dediğim gibi key değerini küçük bir aralıkta tutarak bu küçük sayılarla çarpma yapmak ve her adımda bu küçük sayının modunu almak büyük sayılarla çarpma yapmaya göre çok daha verimli.

İkinci zorluk ise satırı derlerken tokenlama kısmıydı. Dosyayı aldıktan sonra satırlara ayırmak kolaydı ancak satırı derleme kısmında deklare etme satırı iken '=' gördükten sonra virgül görene kadarkileri deklare etme değil de kullanım olarak kontrol etmek üzerine baya mesai harcadım. Çünkü bu kontrolü yapabilmek için ilk önce tokenı aldığım gibi delimiter'a da ihtiyacım vardı. Ancak strtok delimiter'ı döndürmüyor. Bu sebeple custom bir strtok yazdım, bu delimitera kadar olan stringi tokenBuffer stringine kopyalayıp sonra delimiter'ı return ediyor.

Son olarak da ödev için derleyicinin hashingi kullandığı bölümünü yapsak yeterli olur mu yoksa tokenlama kısmı da yapılmalı mı kararsız kaldım. Mesela deklare etme için satır int, float veya char ile başlamalı ve satır ';' ile bitiyor. Eğer bu satırda tekrar bu keywordleri görürsek C derleyicisi hata verip satırın geri kalanını derlemiyor. Böyle bir durumu handle etmek gerekir mi kararsız kaldım ama yaptım. Ancak zor olmadı.

3- Karmaşıklık Analizi:

```c
/*
@brief it uses horner method and calculates the hash.

@param name: key of which this will calculate hash
@param m: size of the table

@return hash value
*/

int hashFunction(char* name, int m){
    int i, L = strlen(name);
    int hash = 0;

    //taking mod in each iter doesnt change the resulted hash value, we could take it at the end but we need to use long long to store hash
    //and also multiyplying big numbers slows the process.
    for(i=0;i<L;i++){
        hash = (hash * R + name[i]) % m;
    }

    return hash;
}
```

Hash hesabının karmaşıklığı stringin uzunluğu kadar yani O(L)

```c
/*
@brief it searches at the map with double hashing looking for name.

@param map: pointer to the map
@param name: key of which we will look if it is in the map

@return 1 if it is found, 0 if not
*/

int lookup(MAP* map, char* name){
    int h1 = hashFunction(name, map->m);
    int h2 = 1 + hashFunction(name, map->m - 3);
    int i = 0, slot = h1;

    while(i < map->m && map->hashMap[slot].name != NULL){ //if there is an empty slot while doing probing, then key cannot be in the map
        if(!strcmp(map->hashMap[slot].name, name)) return 1;
        i++;
        slot = (h1 + i * h2) % map->m; //doing probing
    };

    return 0;
}
```

```c
/*
@brief it inserts to the map with double hashing.

@param map: pointer to the map
@param name: key we will insert to the map
@param type: value

@return
*/

void insert(MAP* map, char* name, char* type){
    int h1 = hashFunction(name, map->m);
    int h2 = 1 + hashFunction(name, map->m - 3);
    int i = 0, slot = h1;

    //it is assumed that lookup function is called before insert function so this loop doesnt check whether there is a variable added before
    while(i < map->m && map->hashMap[slot].name != NULL){
        i++;
        slot = (h1 + i * h2) % map->m;
    };

    map->hashMap[slot].name = (char*) malloc((strlen(name) + 1) * sizeof(char));
    strcpy(map->hashMap[slot].name, name);
    strcpy(map->hashMap[slot].type, type);
    map->n++;
}
```

lookup ve insert fonksiyonlarının karmaşıklıkları aynı. hash hesabı için O(2L) = O(L) ve arama işleminin karmaşıklığı ise ortalama probe sayısı kadar. Bu da ortalamada open addressing için 1 /(1 – load factor) kadar. Load factor'ü 0.5'ten küçük tuttuğumuz için O(1) diyebiliriz. Yani hem lookup hem de insert için O(L) karmaşıklık var. hash kısmını bu şekilde implemente etme sebebimi karşılaşılan zorluklarda belirtmiştim. Big O notasyonunda belirtmediğimiz faktörler bence bu şekilde yapmayı daha mantıklı kılıyor. long long tipinde tutulacak büyük sayıların çarpım maliyeti m'den küçük sayıların çarpım işlemi ve her turda modunu almaktan daha büyük. Bir kez keyi bulup birer mod işlemi yapmak görünürde O(L) ancak bence daha büyük maliyet, özellikle uzun tokenlar için.

```c
/*
@brief it iterates over the line via p until it sees a delimiter and copies the chars to tokenBuffer while iterating.

@param tokenBuffer: a string that will store the current token.
@param p: pointer to the pointer of the current char of the line.
@param DELIMITERS: a string that stores the wanted delimiters.

@return the delimiter
*/

char customStrtok(char* tokenBuffer, char** p, char* DELIMITERS){
    int tokenLen = 0;
    while (**p && !isDelimiter(DELIMITERS, **p)){
        tokenBuffer[tokenLen++] = **p;
        (*p)++;
    }

    tokenBuffer[tokenLen] = '\0';
    char delimiter = **p;
    if(**p) (*p)++;
    return delimiter;
}
```

```c
else{ //if the first token is not a declaration keyword then we enter this block, this is not a declaration line
    while((token[0] != '\0' || *p != '\0') && ctrl){ //iterates until both the line is ended and token gets null char
        if(isValidVariable(token)){ //if it valid variable name then we can check if it is declared before
            if(!lookup(map, token)){ //if it is not declare before we should give error
                printf("Line %d -> ERROR: variable %s was used but was not declared!\n", lineNo, token);
            }
        }
        else if(isDeclarationKeyword(token)){ //if we see a declaration keyword then we should stop compiling this line
            printf("Line %d -> ERROR: expectecd identifier like ';' or '(' before %s!\n", lineNo, token);
            ctrl = 0;
        }
        else if(!isNum(token) && !isAnotherKeyword(token)){ //if token is not a number or another keyword like for, if, while, then this
        //invalid variable name
            printf("Line %d -> ERROR: unknown variable type '%s'\n", lineNo, token);
        }

        skipWhiteSpaces(&p);
        customStrtok(tokenBuffer, &p, DELIMITERS); //getting the token (delimiter is not important for a this line since it is not
        // a declaration line)
        token = tokenBuffer;
    }
}
```

compileLine fonksiyonun if ve else bloğu var iki kolun da karmaşıklığı aynı çünkü satır bitene kadar tokenlama işlemi yapıp şartlar sağlanırsa token için insert ve lookup fonksiyonları çağrılıyor. customStrtok fonksiyonunun yaptığı iş delimiter'a kadar p'yi ilerletip tokenı tutacak bir stringe kopyalamak. Bir dahaki çağrışta kaldığı delimiterın hemen sonrasından devam edeceğinden karmaşıklık toplamda bir satırı bir kez gezmek ve bu satırı bir kez kopyalamak olacak. Bunu her bir satır için yapacağımızdan toplam karmaşıklık kod uzunluğunun iki katı kadar olacak diyebiliriz. Bu da uzunluk K ise O(2* K) = O(K) eder. Satırı gezerken şartları sağlayan her bir değişken için de lookup ve insert fonksiyonları çağrılacak. Bunların karmaşıklığı token uzunluğuna bağlı yani O(L) demiştik. Bu işlemler her satır için olacağından aslında total karmaşıklık lookup ve insertin kaç kere çağrılacağı ile alakalı. Bu da kodda toplamda n değişken varsa O(n * L) karmaşıklık yapar. Yani toplam karmaşıklık O(K + n * L) yapar ancak n * L <= K diyebiliriz çünkü kodda n'den fazla token var ve L tokenın uzunluğunu ifade ediyor. Sonuç olarak toplam karmaşıklığa O(K) diyebiliriz.

4-Ekran Çıktıları:

Input 1:

```
int main()

{

int _aa, _bb, _cc;

char _aa;

char _x;

_aa = 5;

_xx = 9;

_bb = _aa + _dd;

}
```

Output 1:

```
file name: code.txt
enter 1 to run in debug mode...
enter 0 to run in normal mode...
mode: 1

-------------------------------------------------------------------------------------------------------------------------
Line 1 -> variable _aa can be declared...
Line 1 -> variable _bb can be declared...
Line 1 -> variable _cc can be declared...
Line 2 -> ERROR: variable _aa was declared before!
Line 3 -> variable _x can be declared...
Line 5 -> ERROR: variable _xx was used but was not declared!
Line 6 -> ERROR: variable _dd was used but was not declared!

-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
n (declared variables): 4 - m (table size): 11 - load factor: 0.36
-------------------------------------------------------------------------------------------------------------------------
0 -> Empty
1 -> Empty
2 -> Empty
3 -> Empty
4 -> Empty
5 -> Empty
6 -> (_cc, int) - first address (h1): 6 - probe step size (h2): 8 - actual address: 6 - the number of probes: 1
7 -> (_bb, int) - first address (h1): 7 - probe step size (h2): 8 - actual address: 7 - the number of probes: 1
8 -> (_aa, int) - first address (h1): 8 - probe step size (h2): 8 - actual address: 8 - the number of probes: 1
9 -> (_x, char) - first address (h1): 7 - probe step size (h2): 2 - actual address: 9 - the number of probes: 2
10 -> Empty

-------------------------------------------------------------------------------------------------------------------------
```

Input 2:

```
#include<stdio.h>
int main()
{
int _Aa, _BB = _cc + _om, _nn, _mm;float _bc
;
char _aa;
char _x;
float _t_t;
_aa = 5;
_xx = 9;
int _pp, _gg, _aa, _hh, _yy, _1a, _2a, _3a, _y1, aa;
_bb = _aa + _dd;
if(_aa == _b) {
s;
_bb = 5;
}
for(_i=0;_i<10;_i++){
_aa += 1;}
}
```

Output 2:

```
file name: code2.c
enter 1 to run in debug mode...
enter 0 to run in normal mode...
mode: 1

-----------------------------------------------------------------------------------------------------------------------
Line 1 -> variable _Aa can be declared...
Line 1 -> variable _BB can be declared...
Line 1 -> ERROR: variable _cc was used but was not declared!
Line 1 -> ERROR: variable _om was used but was not declared!
Line 1 -> variable _nn can be declared...
Line 1 -> variable _mm can be declared...
Line 2 -> variable _bc can be declared...
Line 3 -> variable _aa can be declared...
Line 4 -> variable _x can be declared...
Line 5 -> variable _t_t can be declared...
Line 7 -> ERROR: variable _xx was used but was not declared!
Line 8 -> variable _pp can be declared...
Line 8 -> variable _gg can be declared...
Line 8 -> ERROR: variable _aa was declared before!
Line 8 -> variable _hh can be declared...
Line 8 -> variable _yy can be declared...
Line 8 -> variable _1a can be declared...
Line 8 -> variable _2a can be declared...
Line 8 -> variable _3a can be declared...
Line 8 -> variable _y1 can be declared...
Line 8 -> ERROR: variable name aa is invalid!
Line 9 -> ERROR: variable _bb was used but was not declared!
Line 9 -> ERROR: variable _dd was used but was not declared!
Line 10 -> ERROR: variable _b was used but was not declared!
Line 11 -> ERROR: unknown variable type 's'
Line 12 -> ERROR: variable _bb was used but was not declared!
Line 14 -> ERROR: variable _i was used but was not declared!
Line 15 -> ERROR: variable _i was used but was not declared!
Line 16 -> ERROR: variable _i was used but was not declared!

-----------------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------------------------
n (declared variables): 16 - m (table size): 37 - load factor: 0.43
-----------------------------------------------------------------------------------------------------------------------
0 -> (_t_t, float) - first address (h1): 0 - probe step size (h2): 11 - actual address: 0 - the number of probes: 1
1 -> Empty
2 -> Empty
3 -> (_yy, int) - first address (h1): 3 - probe step size (h2): 2 - actual address: 3 - the number of probes: 1
4 -> (_1a, int) - first address (h1): 4 - probe step size (h2): 24 - actual address: 4 - the number of probes: 1
5 -> (_y1, int) - first address (h1): 5 - probe step size (h2): 32 - actual address: 5 - the number of probes: 1
6 -> Empty
7 -> Empty
8 -> (_bc, float) - first address (h1): 8 - probe step size (h2): 15 - actual address: 8 - the number of probes: 1
9 -> Empty
10 -> (_3a, int) - first address (h1): 29 - probe step size (h2): 18 - actual address: 10 - the number of probes: 2
11 -> (_pp, int) - first address (h1): 11 - probe step size (h2): 20 - actual address: 11 - the number of probes: 1
12 -> (_aa, char) - first address (h1): 12 - probe step size (h2): 16 - actual address: 12 - the number of probes: 1
13 -> Empty
14 -> (_hh, int) - first address (h1): 14 - probe step size (h2): 2 - actual address: 14 - the number of probes: 1
15 -> Empty
16 -> Empty
17 -> Empty
18 -> Empty
19 -> (_Aa, int) - first address (h1): 19 - probe step size (h2): 10 - actual address: 19 - the number of probes: 1
20 -> Empty
21 -> (_nn, int) - first address (h1): 21 - probe step size (h2): 24 - actual address: 21 - the number of probes: 1
22 -> Empty
23 -> (_gg, int) - first address (h1): 19 - probe step size (h2): 4 - actual address: 23 - the number of probes: 2
24 -> Empty
25 -> Empty
26 -> (_mm, int) - first address (h1): 26 - probe step size (h2): 26 - actual address: 26 - the number of probes: 1
27 -> Empty
28 -> Empty
29 -> (_BB, int) - first address (h1): 19 - probe step size (h2): 10 - actual address: 29 - the number of probes: 2
30 -> Empty
31 -> (_x, char) - first address (h1): 31 - probe step size (h2): 6 - actual address: 31 - the number of probes: 1
32 -> Empty
33 -> Empty
34 -> Empty
35 -> (_2a, int) - first address (h1): 35 - probe step size (h2): 21 - actual address: 35 - the number of probes: 1
36 -> Empty
```

Input 3:

```
int main(){
        int _aa, _bb = _aa, _cc ;
        char _aa;char _x, char _y;float _zz = -43;
        _aa = 5       ;
        _xx
         =
        9
              ;
        float _yy = 7;
        _bb = _aa + _dd;
}
```

Output 3:

```
file name: code3.txt
enter 1 to run in debug mode...
enter 0 to run in normal mode...
mode: 1

-----------------------------------------------------------------------------------------------------------------------
Line 1 -> variable _aa can be declared...
Line 1 -> variable _bb can be declared...
Line 1 -> variable _cc can be declared...
Line 2 -> ERROR: variable _aa was declared before!
Line 3 -> variable _x can be declared...
Line 3 -> ERROR: expected identifier like ';' or '(' before declaration keyword char!
Line 4 -> variable _zz can be declared...
Line 6 -> ERROR: variable _xx was used but was not declared!
Line 7 -> variable _yy can be declared...
Line 8 -> ERROR: variable _dd was used but was not declared!

-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
n (declared variables): 6 - m (table size): 17 - load factor: 0.35
-----------------------------------------------------------------------------------------------------------------------
0 -> Empty
1 -> (_yy, float) - first address (h1): 1 - probe step size (h2): 10 - actual address: 1 - the number of probes: 1
2 -> Empty
3 -> Empty
4 -> Empty
5 -> (_x, char) - first address (h1): 5 - probe step size (h2): 14 - actual address: 5 - the number of probes: 1
6 -> Empty
7 -> Empty
8 -> Empty
9 -> Empty
10 -> Empty
11 -> (_cc, int) - first address (h1): 11 - probe step size (h2): 6 - actual address: 11 - the number of probes: 1
12 -> Empty
13 -> (_bb, int) - first address (h1): 13 - probe step size (h2): 2 - actual address: 13 - the number of probes: 1
14 -> Empty
15 -> (_aa, int) - first address (h1): 15 - probe step size (h2): 12 - actual address: 15 - the number of probes: 1
16 -> (_zz, float) - first address (h1): 16 - probe step size (h2): 14 - actual address: 16 - the number of probes: 1

-----------------------------------------------------------------------------------------------------------------------
```