BLM 2425 ALGORİTMA ANALİZİ

ÖDEV 3

Hashing

Öğrenci Adı: Sinem SARAK
Öğrenci Numarası: 22011647
Dersin Eğitmeni: M. Elif KARSLIGİL
Sunum Video Linki: https://youtu.be/D7PjslAXbKQ

İçindekiler

Problemin Çözümü

Çözüm içerisinde değişlenler myvariable ismiyle oluşturulan bir struct veri yapısı kullanılarak tutulmaktadır. Bu veri yapısı değişkenin ismi ve tipini bir arada tutabilmek için oluşturulmuştur. Değişkenlerin tipleri varType olarak adlandırılan bir enum struct veri yapısı kullanılarak tutulur ve kullanılır. Enum kullanılarak, değişken tiplerinin önceden belirlenmesi ve daha düzenli bir şekilde yönetilmesi amaçlanmıştır. Çözümün uygulanması için izlenen adımlar ve bu adımlarda kullanılan fonksiyonlar alt başlıklar halinde ele alınmıştır:

1- Değişkenlerin Sayılması

Oluşturulacak hash tablosunun uzunluğunun belirlenmesi adına ilk önce değişkenler sayılmıştır. Bu işlemi yapmak için kullanılan temel fonksiyon ve değişkenler şunlardır:

scanFile: Dosya içerisindeki her satırı tarar, _ ile başlayan değişkenleri tespit eder ve bu değişkenlerin tanımlanmış mı yoksa kullanılmış mı olduğunu tespit eder. Bu fonksiyon ilk çağrıldığında hash table parametresi NULL gönderilerek sadece değişkenleri sayması sağlanmıştır.

determineType: Satırdaki kelimeler üzerinde işlem yaparak, bir değişkenin tanımlanıp tanımlanmadığının anlaşılmasına yardımcı olur. Eğer cümle int,char vb. tanımlayıcılarla başlıyorsa bu fonksiyon tarafından ilgili satırda tanım yapıldığına karar verilir ve değişken sayısı artırılır.

variableCount: Tanımlanan tüm değişkenlerin sayısını tutan bir değişkendir. scanFile fonksiyonuna, bu değişkeni işaret eden işaretçi (pointer) gönderilerek tanımlanan değişken sayısının sayılması sağlanır.

scanFile fonksiyonu, her satırı okur ve ilgili satırda tanımlama var ise bunu determineType ile algılayarak variableCount'u artırır. Henüz hash table oluşturulmadığından birden fazla tanımlanan değişkenler tespit edilemeyecektir. Bu nedenle bu sayma işlemi sonucu elde edilen değişken sayısı gerçekten hatasız tanımlanmış değişken sayısını vermeyecek ve sayma işleminin hash table oluşturulduktan sonra tekrardan yapılması gerekecektir.

2- Hash Tablosunun Oluşturulması

Tanımlanan değişken sayısının boyutuna göre *determineM* fonksiyonu kullanılarak hash tablosunun boyutu belirlenir:

determineM: Değişken sayısına göre hash tablosunun boyutunu belirler. Bu değer, değişken sayısının iki katından büyük en küçük asal sayıdır. Bu sayıyı bulmak için değişken sayısının iki katının bir fazlasından başlanarak, asal bir bulunana kadar tüm sayılar *isPrime* fonksiyonu kullanılarak kontrol edilir. Elde edilen ilk asal sayı M değeri olarak döndürülür.

Main fonksiyonu içerisinde calloc ile M sayısı boyutunda bir dizi açılır ve bu dizi hash tablosunu oluşturur.

3- Elemanların Tabloya Yerleştirilmesi ve Hata Kontrolü

Hash tablosu oluşturulduktan sonra, tanımlanan her değişken bu tabloya yerleştirilir. Bu işlemi gerçekleştiren temel fonksiyonlar şunlardır:

scanFile: Dosya içerisindeki her satırı tarar, _ ile başlayan değişkenleri tespit eder ve bu değişkenlerin tanımlanmış mı yoksa kullanılmış mı olduğunu tespit eder. Bu fonksiyon ikinci

defa çağrıldığında hash table parametresi NULL değildir ve bu sefer doğru tanımlanan değişkenlerin sayılmasını ve double hashing kullanılarak hash tablosuna kaydedilmesini sağlar.

processHashTable: scanFile fonksiyonu tarafından algılanan değişken kullanımının uygun olup olmadığını kontrol eder. Öncelikle değişkenin tabloda olup olmadığı lookUp fonksiyonu tarafından kontrol edilir ve eğer değişken tabloda varsa index değişkenine -1, yoksa yerleşmesi gereken indis atanır. Sonrasında determineType fonksiyonu kullanılarak yapılan işlemin bir tanımlama mı yoksa kullanım mı olduğu kontrol edilir. İşlemin tanımlama olduğuna karar verilirse index değişkenindeki indise değişken insert fonksiyonu kullanılarak yerleştirilir ve doğru tanımlanmış değişkenleri saymak için sıfırlanarak tekrardan gönderilmiş olan variableCount değeri artırılır. İndex değerinin -1 olması aynı değişkenin birden fazla tanımlandığı anlamına gelir ve kullanıcı bilgilendirilir. İşlem bir tanımlama değilse değişkenin daha önce tanımlanıp tanımlanmadığı index değişkenine bakılarak kontrol edilir. İndex değişkeni -1' den farklı bir değerse değişken tabloda yoktur ve kullanıcı tanımlanmadan değişken kullandığına dair uyarılır.

determineType: Bir satırdaki işlemin, değişken tanımlama mı yoksa kullanımı mı olduğunu satırdaki ilk kelimeyi kullanarak belirler. İşlem tanımlamaysa, değişkenin tipini temsil eden enum değerini, kullanımsa -1 döner.

lookUp: İsmi parametre olarak verilen değişkenin key değerini getKey fonksiyonu ile hesaplayarak ilgili key değerine karşılık hash tablosunda bulunan değeri kontrol eder. Verilen değişken ilgili adreste yoksa ve adres doluysa, boş bir adres veya verilen değişkenin kendisini bulana kadar double hashing kullanarak hash table içerisinde gezer. Boş bir adres bulursa bu indise döner. Değişken tabloda zaten varsa -1 döndürür.

getKey: Verilen string değişkeninin key değerini Horner's Method kullanarak hesaplar ve bu değere döner.

insert: Hash tablosunun, parametre olarak verilen indisine verilen yeni değişkeni ekler.

scanFile fonksiyonu dosyayı tarayarak değişkenleri analiz eder ve tanımlama işlemlerini algılar. Hash tablosu oluşturulmadan önce yalnızca değişkenlerin toplam sayısı belirlenebilir ve bu aşamada birden fazla tanımlama hataları tespit edilemez. Hash tablosu oluşturulduktan sonra processHashTable fonksiyonu, değişkenleri tabloya yerleştirir, doğru tanımlananları sayar ve hatalı durumları raporlar. Bu süreçte, double hashing yöntemi ile çakışmalar çözümlenir, ve kullanıcıya olası hata durumları bildirilir.

Hash tablosu oluşturulduktan sonra, değişkenler tabloya yerleştirilirken scanFile fonksiyonu satırları tarar ve determineType kullanarak her değişkenin tanımlama mı yoksa kullanım mı olduğunu belirler. İlk çağrımda değişkenlerin yalnızca sayısı tespit edilirken, ikinci çağrımda lookUp ile tablodaki varlıkları kontrol edilir ve insert fonksiyonu ile tabloya eklenir. Tanımlama hataları veya tanımlanmamış değişken kullanımı gibi durumlar processHashTable tarafından değerlendirilerek kullanıcıya bildirilir. Dosyanın taranması bittiğinde doğru tanımlanmış tüm değişkenlerin sayısı ve tutuldukları hash tablosu oluşturulmuş ve hatalı kullanımlarla ilgili kullanıcıya bilgi verilmiş olur.

Karşılaşılan Sorunlar

1- Uzun İsimler Kullanıldığında Integer Taşması Sorunu

Horner's Method kullanılarak string ifadelerden key değerleri hesaplanırken, uzun isimli değişkenler için hesaplanan key değerinin integer türünü aşarak negatif değerlere dönüşmesi gibi bir sorunla karşılaşıldı. Bu durum, özellikle isim uzunluğunun ve karakter değerlerinin büyük olduğu değişkenlerde hash tablosunun yanlış çalışmasına yol açtı.

Sorunun temel nedeni, integer veri tipinin sınırlarının aşılmasıydı. İlk etapta bu sorunu çözmek için hesaplama sırasında overflow durumunu tespit etmeye yönelik kontroller eklendi, ancak bu yöntem performansı olumsuz etkiledi. Sonunda, key hesaplamasında kullanılan veri tipinin **long long** olarak değiştirilmesiyle taşma sorunu tamamen çözüldü. Bu değişiklik, long long tipinin sağladığı geniş sınır sayesinde taşma riski minimize edilerek key değerlerinin doğru şekilde hesaplanmasını sağladı.

2- Program Sonunda Değişken Sayısının Yanlış Hesaplanması

Hash tablosu oluşturulmadan önce tanımlanan değişkenlerin sayısını belirlemek için scanFile fonksiyonu kullanıldı ve kullanıcıya değişken sayısı bilgisi verilirken bu hesaplamaının sonucu kullanıldı. Ancak, bu süreçte hem doğru tanımlanmış hem de yanlış tanımlanmış değişkenler birlikte sayıldığından, değişken sayısı olduğundan fazla çıktı. Bu durum, hatasız değişkenlerin gerçek sayısının tespit edilmesini engelledi ve kullanıcıya verilen bilginin hatalı olmasına neden oldu.

Bu sorunu çözmek için scanFile fonksiyonu, hash tablosu oluşturulduktan sonra çağrıldığında doğru tanımlanmış değişkenlerin sayısı yeniden hesaplandı. Bu işlem sırasında, variableCount değişkeni sıfırlanarak yalnızca doğru tanımlanmış değişkenlerin dikkate alınması sağlandı. Böylece kullanıcıya dönen bilgi düzeltilmiş oldu.

Karmaşıklık Analizi

Kod içerisindeki fonksiyonlar incelendiğinde, scanFile fonksiyonu, zaman karmaşıklığı açısından en fazla yükü taşıyan fonksiyon olarak öne çıkmaktadır. Bunun temel nedeni, dosyanın satır satır taranarak her bir satırdaki değişkenlerin tanımlama veya kullanım durumlarının kontrol edilmesi ve gerekirse hash tablosuna kaydedilmesidir. Ayrıca, bu işlem sırasında diğer fonksiyonlar sıkça çağrılarak ek işlemler yapılmaktadır. Aşağıda scanFile ve processHashTable fonksiyonlarına ait sözde kod görülmektedir:

```
Function scanFile(filename, hashTable, variableCount, M)
  Open file with filename
  If file cannot be opened
    Print error and exit

  While there are lines to read from the file
    Read a line into variable line

    If line contains '_'
      Initialize token to the first word in line
      Initialize tempType to an empty string
```

```
        Initialize index

        If token is not NULL
           Copy token to tempType

        While token is not NULL
           If token starts with '_'
              If hashTable is not NULL
                 Call processHashTable(hashTable, token, tempType, variableCount, M)
              Else if determineType(tempType) is not -1
                 Increment variableCount

           Get the next token in line

   Close the file
End Function


Function processHashTable(hashTable, token, tempType, variableCount, M)
   index = lookUp(M, hashTable, token)

   If determineType(tempType) is not -1
      If index is -1
         Print "The variable token is defined twice"
      Else
         Call insert(hashTable, token, index, tempType)
         Increment variableCount
   Else
      If index is not -1
         Print "Usage without definition of: token"
End Function
```

Döngü içerisinde yapılan diğer işlemler basit artırma ve atama işlemleri olduğundan, scanFile fonksiyonundaki en karmaşık işlemler, döngü içerisinde hash tablosuyla ilgili operasyonları içeren kısımlardır. Döngü içerisinde çağırılan processHashTable ve içerisinde kullanılan, hash tablosunda bulunup bulunmadığını kontrol eden, lookUp fonksiyonu ve yeni bir değişkeni hash tablosuna ekleyen insert fonksiyonları kodun en karmaşık adımlarını oluşturur.

İçten dışa incelemek gerekirse, öncelikle lookUp ve insert fonksiyonları ele alınır. insert fonksiyonu sadece iki adet atama işleminden oluşur ve karmaşıklığı O(1) denilebilir. lookUp fonksiyonu ise bir hash arama fonksiyonudur ve en karmaşık kısmı getKey fonksiyonudur. getKey fonksiyonu kelimenin uzunluğu kadar dönen bir döngü oluşturur. getKey için O(değişkenin uzunluğu) denilebilir. Hash tablosu double hash kullanılarak dağıtıldığından ve değişken sayısının iki katından daha fazla uzunluğa sahip olduğundan boş yer veya aranan elemanı bulma O(1) olarak düşünülebilir. Keyin hesaplanması key'e uygun indis değerinin bulunmasının toplam karmaşıklığı olarak lookUp fonksiyonunun karmaşıklığı O(değişken uzunluğu) olarak düşünülebilir. lookUp ve insert fonksiyonlarından en karmaşık olanı processHashTable'ın karmaşıklığını oluşturur ve bu değer O(değişken uzunluğu) olarak bulunur.

processHashTable fonksiyonu scanFile içerisindeki while döngüsünde dosyada yer alan değişken sayısı kadar çağrılır. Bu durumda, scanFile fonksiyonunun genel karmaşıklığı, dosyadaki değişken sayısı (n) ile her bir değişkenin ortalama uzunluğunun (k) çarpımına bağlıdır. Dolayısıyla, scanFile fonksiyonu için toplam zaman karmaşıklığı O(n * k) şeklinde ifade edilebilir.

Ekran Çıktıları

Debug çıktıları normal modun çıktılarını kapsadığından ödev dosyasında verilen örneğe dair çıktı hariç tüm çıktılar debug modunda verilecektir. Ödev dosyası ve iki farklı kod örneği olmak üzere kod üç farklı örnekle test edilmiş ve çıktıları aşağıda verilmiştir.

1- Ödev Dosyasında Verilen Kod Parçası

```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    int _aa, _bb, _cc;
    char _aa;
    char _x;
    _aa = 5;
     _xx = 9;
    _bb = _aa + _dd;
}
```

```
Hello. Welcome to mini variable checker. Please choose a mode:
(1 for Normal, 2 for Debug): 1
Please enter the name of your file: giventest.c
```

Ödev dosyasında verilen kod parçası normal modda çalıştırıldığında çıktı aşağıdaki gibidir:

```
The variable _aa is defined twice
Usage without definition of: _xx
Usage without definition of: _dd
```

Ödev dosyasında verilen kod parçası debug modda çalıştırıldığında çıktı aşağıdaki gibidir:

```
Hello. Welcome to mini variable checker. Please choose a mode:
(1 for Normal, 2 for Debug): 2
Please enter the name of your file: giventest.c
```

```
The variable _aa is defined twice
Usage without definition of: _xx
Usage without definition of: _dd

Number of declared variables: 4
The size of the hashTable: 11

***************************************************************
                        Hash Table
***************************************************************

hashTable[0]:   is empty.
---------------------------------------------------------------------------
hashTable[1]:   is empty.
---------------------------------------------------------------------------
hashTable[2]:   is empty.
---------------------------------------------------------------------------
hashTable[3]:   is empty.
---------------------------------------------------------------------------
hashTable[4]:   is empty.
---------------------------------------------------------------------------
hashTable[5]:   is empty.
---------------------------------------------------------------------------
hashTable[6]:   Name: _cc       Type: int      Calculated key value: 6
---------------------------------------------------------------------------
hashTable[7]:   Name: _bb       Type: int      Calculated key value: 7
---------------------------------------------------------------------------
hashTable[8]:   Name: _aa       Type: int      Calculated key value: 8
---------------------------------------------------------------------------
hashTable[9]:   Name: _x        Type: char     Calculated key value: 7
---------------------------------------------------------------------------
hashTable[10]:  is empty.
---------------------------------------------------------------------------
```

2- Simple

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int _aa;
    double _uzunbirdegisken;
    _b = 20;
    float _c;
    char _d;
    _d = 'x';
    if(_aa==3){
        printf("iste tanimlanmamis bir degisken: %d", _tanimsiz );
    }

}
```

```
Usage without definition of: _b
Usage without definition of: _tanimsiz

Number of declared variables: 4
The size of the hashTable: 11

***************************************************************
                        Hash Table
***************************************************************

hashTable[0]:    is empty.
---------------------------------------------------------------------------
hashTable[1]:   Name: _c         Type: float    Calculated key value: 8
---------------------------------------------------------------------------
hashTable[2]:    is empty.
---------------------------------------------------------------------------
hashTable[3]:    is empty.
---------------------------------------------------------------------------
hashTable[4]:   Name: _uzunbirdegisken   Type: double   Calculated key value: 4
---------------------------------------------------------------------------
hashTable[5]:    is empty.
---------------------------------------------------------------------------
hashTable[6]:    is empty.
---------------------------------------------------------------------------
hashTable[7]:    is empty.
---------------------------------------------------------------------------
hashTable[8]:   Name: _aa        Type: int      Calculated key value: 8
---------------------------------------------------------------------------
hashTable[9]:   Name: _d         Type: char     Calculated key value: 9
---------------------------------------------------------------------------
hashTable[10]:   is empty.
---------------------------------------------------------------------------
```

3- Complicated

```c
#include <stdio.h>

int main()
{
    int _for;
    float _while;
    int _i, _temp1, _temp2;
    for(_i=0;_i < 34;_yanlisdegisken ++){
        _tanimlanmamisdegisken ++;
    }
    if ( _ifdetanimsiz == 10) {
            _temp1 = _i * 2;
            _temp2 =_result+_temp1;
            printf("Temp2: %.2f\n", _temp2 );
        } else if (_i == 20) {
            printf("i just love coding");
        }
    int _sinem;
    char _sinem,_sarak;

}
```

```
Usage without definition of: _yanlisdegisken
Usage without definition of: _tanimlanmamisdegisken
Usage without definition of: _ifdetanimsiz
Usage without definition of: _result
The variable _sinem is defined twice

Number of declared variables: 7
The size of the hashTable: 17

***************************************************************
                        Hash Table
***************************************************************

hashTable[0]:    is empty.
---------------------------------------------------------------------------
hashTable[1]:   Name: _sinem     Type: int      Calculated key value: 1
---------------------------------------------------------------------------
hashTable[2]:    is empty.
---------------------------------------------------------------------------
hashTable[3]:   Name: _temp1     Type: int      Calculated key value: 3
---------------------------------------------------------------------------
hashTable[4]:   Name: _for       Type: int      Calculated key value: 4
---------------------------------------------------------------------------
hashTable[5]:    is empty.
---------------------------------------------------------------------------
hashTable[6]:   Name: _while     Type: float    Calculated key value: 6
---------------------------------------------------------------------------
hashTable[7]:   Name: _i         Type: int      Calculated key value: 7
---------------------------------------------------------------------------
hashTable[8]:   Name: _sarak     Type: char     Calculated key value: 8
---------------------------------------------------------------------------
hashTable[9]:    is empty.
---------------------------------------------------------------------------
hashTable[10]:   is empty.
---------------------------------------------------------------------------
hashTable[11]:  Name: _temp2     Type: int      Calculated key value: 4
---------------------------------------------------------------------------
hashTable[12]:   is empty.
---------------------------------------------------------------------------
hashTable[13]:   is empty.
---------------------------------------------------------------------------
hashTable[14]:   is empty.
---------------------------------------------------------------------------
hashTable[15]:   is empty.
---------------------------------------------------------------------------
hashTable[16]:   is empty.
---------------------------------------------------------------------------
```