HASHING

Öğrenci Adı: Melih Yelman
Öğrenci Numarası: 21011702
Ders Eğitmeni: Do.. Dr. Mehmet Amaç Güvensan
Video Linki: https://youtu.be/w6PoGraTcts

1- Problemin Çözümü:

Bu çalışmada, bir C programında tanımlanan ve kullanılan değişkenlerin sembol tablosu üzerinden hata kontrolü yapılmıştır. İlk olarak, programdaki değişken sayısı tahmin edilerek ve bu sayı kullanılarak sembol sembol tablosunun boyutu, iki katından büyük ilk asal sayı olacak şekilde belirlenmiştir. Ardından her değişen ismi Horner Kuralı ile sayısal bir değere dönüştürülmüş, bu değerler açık adresleme ve çift hashleme yöntemleri kullanılarak sembol tablosuna eklenmiştir.

Geliştirilen algoritma iki temel işlevden oluşmaktadır: Insert fonksiyonu yeni tanımlanan değişkenleri tabloya yerleştirirken daha önce tanımlanıp tanımlanmadığını kontrol etmektedir. Lookup fonksiyonu ise program içinde kullanılan bir değişkenin sembol tablosunda varlığını sorgulamakta ve tanımlanmamış ya da birden fazla kez tanımlanmış durumlarda hata mesajları üretmektedir. Böylece, programda değişkenlere ilişkin doğrulama ve kullanım hataları saptanarak uyarılabilir hale getirilmiştir.

2- Karşılaşılan Sorunlar:

Programın sembol tablosunu oluşturma sürecinde özellikle dosyadan okunan değişkenlerin sayısını tahmin etmek ve bu değişkenlerin kod içinde doğru şekilde kullanıldığını kontrol etme konularında zorluklarla karşılaşılmıştır. C dosyasında tanımlı değişkenleri önceden sayarak sembol tablosunun boyutunu belirlemede kritik bir rol oynamaktadır. Ancak, kod içerisindeki birden fazla aynı türde değişken veya farklı türde ama aynı isimde değişken kullanımı bu tahmini zorlaştırmıştır.

Benzer şekilde, hem değişken tanımlarını hem de kullanım yöntemlerini inceleyerek hataları tespit etmekte de zorlukla karşılaşılmıştır. Dosyadan okunan her satırdaki boşluklar, değişkenlerin birden fazla kez tanımlanması veya hiç tanımlanmadan kullanılma durumlarının saptanması, işlemi gerçekleştiren fonksiyonu karmaşık hale getirmiştir.

3- Karmaşıklık Analizi:

Bu uygulamada karmaşıklık analizini kaba hatlarıyla değerlendirmek için, işlem adımlarını temel methodlara ayırmak mümkündür.

- **Değişken Sayısı Tahmini**

  Dosya içeriği satır satır okunmakta ve her satırda değişken tanımlamaları aranarak sayılmaktadır. Bu işlem, dosyadaki toplam satır sayısı N ve her satırın ortalama uzunluğu M olarak kabul edildiğinde O(N * M) zaman alabilir. Lineer zamanda, satır araması ve token kullanımı söz konusudur.

- **Hash Tabloda Insert ve Lookup İşlemleri**

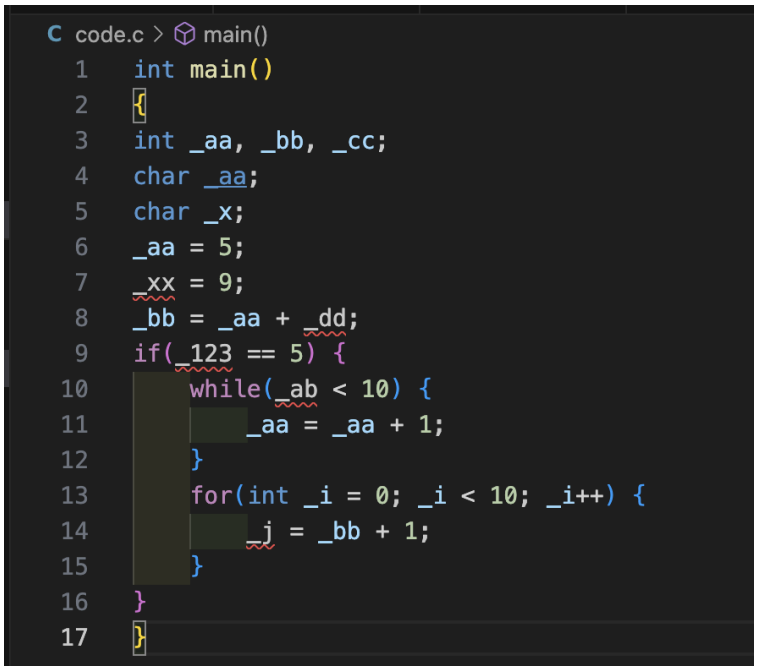
  Açık adresleme ve çift hashleme yöntemleri ortalama durumda O(1) karmaşıklığa sahiptir. N adet değişken için ekleme işlemleri yaklaşık O(N) zaman alırken, kullanım kontrolü için yapılan lookup işlemleri de yine toplam M kullanılma sayısı için O(M) olacaktır.

- **Tablo Ekleme ve Kontrol Methodu**

  Dosya ikinci kez okunarak değişken kullanımları için incelenir ve tabloya eklenir. Bu da benzer biçimde O(N * M) ile ifade edilebilir

4- Ekran Görüntüleri

Örnek 1:



```
❯ ./a.out NORMAL code.c
_aa variable is already declared.
_xx variable is not declared.
_dd variable is not declared.
_123 variable is not declared.
_ab variable is not declared.
_j variable is not declared.
❯
❯
❯
❯ ./a.out DEBUG code.c
Declared variable count (initial estimate): 6
Symbol table size m = 13
DEBUG: _aa (int) initial=10, placed=10
DEBUG: _bb (int) initial=9, placed=9
DEBUG: _cc (int) initial=8, placed=8
_aa variable is already declared.
DEBUG: _x (char) initial=8, placed=4
DEBUG: _i (int) initial=6, placed=6
_xx variable is not declared.
_dd variable is not declared.
_123 variable is not declared.
_ab variable is not declared.
_j variable is not declared.

Final Symbol Table:
[0]: EMPTY
[1]: EMPTY
[2]: EMPTY
[3]: EMPTY
[4]: _x (char)
[5]: EMPTY
[6]: _i (int)
[7]: EMPTY
[8]: _cc (int)
[9]: _bb (int)
[10]: _aa (int)
[11]: EMPTY
[12]: EMPTY
```

Örnek 2:

```
C code2.c > main()
1  int main()
2  {
3  int _aa, _bb, _cc;
4  char _aa;
5  char _x;
6  _aa = 5;
7  _xx = 9;
8  _bb = _aa + _dd;
9  }
```

```
❯ ./a.out NORMAL code2.c
_aa variable is already declared.
_xx variable is not declared.
_dd variable is not declared.
❯
❯
❯
❯ ./a.out DEBUG code2.c
Declared variable count (initial estimate): 5
Symbol table size m = 11
DEBUG: _aa (int) initial=3, placed=3
DEBUG: _bb (int) initial=8, placed=8
DEBUG: _cc (int) initial=2, placed=2
_aa variable is already declared.
DEBUG: _x (char) initial=5, placed=5
_xx variable is not declared.
_dd variable is not declared.

Final Symbol Table:
[0]: EMPTY
[1]: EMPTY
[2]: _cc (int)
[3]: _aa (int)
[4]: EMPTY
[5]: _x (char)
[6]: EMPTY
[7]: EMPTY
[8]: _bb (int)
[9]: EMPTY
[10]: EMPTY
```

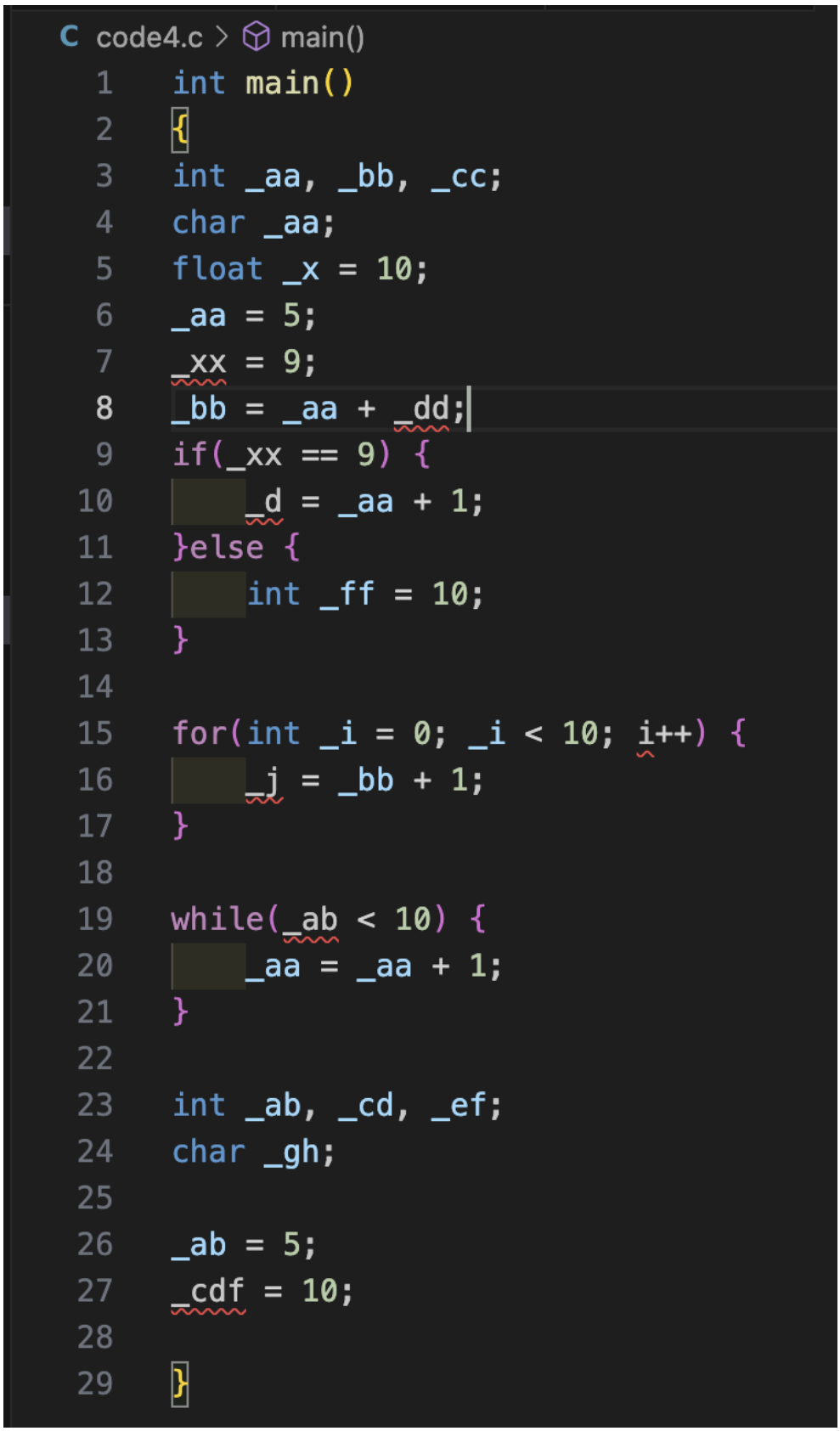
Örnek 3:

code3.c > main()

```c
int main()
{
int _aa, _bb, _cc;
char _aa;
float _x = 10;
_aa = 5;
_xx = 9;
_bb = _aa + _dd;
if(_xx == 9) {
    _d = _aa + 1;
}else {
    int _ff = 10;
}

for(int _i = 0; _i < 10; i++) {
    _j = _bb + 1;
}

while(_ab < 10) {
    _aa = _aa + 1;
}

}
```

```
❯ ./a.out NORMAL code3.c
_aa variable is already declared.
_xx variable is not declared.
_dd variable is not declared.
_xx variable is not declared.
_d variable is not declared.
_j variable is not declared.
_ab variable is not declared.
❯
❯
❯
❯
❯ ./a.out DEBUG code3.c
Declared variable count (initial estimate): 7
Symbol table size m = 17
DEBUG: _aa (int) initial=2, placed=2
DEBUG: _bb (int) initial=6, placed=6
DEBUG: _cc (int) initial=10, placed=10
_aa variable is already declared.
DEBUG: _x (float) initial=14, placed=14
DEBUG: _ff (int) initial=5, placed=5
DEBUG: _i (int) initial=16, placed=16
_xx variable is not declared.
_dd variable is not declared.
_xx variable is not declared.
_d variable is not declared.
_j variable is not declared.
_ab variable is not declared.

Final Symbol Table:
[0]: EMPTY
[1]: EMPTY
[2]: _aa (int)
[3]: EMPTY
[4]: EMPTY
[5]: _ff (int)
[6]: _bb (int)
[7]: EMPTY
[8]: EMPTY
[9]: EMPTY
[10]: _cc (int)
[11]: EMPTY
[12]: EMPTY
[13]: EMPTY
[14]: _x (float)
[15]: EMPTY
[16]: _i (int)
```

Örnek 4:


code4.c > main()
int main()
{
int _aa, _bb, _cc;
char _aa;
float _x = 10;
_aa = 5;
_xx = 9;
_bb = _aa + _dd;
if(_xx == 9) {
_d = _aa + 1;
}else {
int _ff = 10;
}
for(int _i = 0; _i < 10; i++) {
_j = _bb + 1;
}
while(_ab < 10) {
_aa = _aa + 1;
}
int _ab, _cd, _ef;
char _gh;
_ab = 5;
_cdf = 10;
}

```
❯ ./a.out NORMAL code4.c
_aa variable is already declared.
_xx variable is not declared.
_dd variable is not declared.
_xx variable is not declared.
_d variable is not declared.
_j variable is not declared.
_cdf variable is not declared.
❯
❯
❯
❯ ./a.out DEBUG code4.c
Declared variable count (initial estimate): 11
Symbol table size m = 23
DEBUG: _aa (int) initial=19, placed=19
DEBUG: _bb (int) initial=11, placed=11
DEBUG: _cc (int) initial=3, placed=3
_aa variable is already declared.
DEBUG: _x (float) initial=1, placed=1
DEBUG: _ff (int) initial=2, placed=2
DEBUG: _i (int) initial=9, placed=9
DEBUG: _ab (int) initial=20, placed=20
DEBUG: _cd (int) initial=4, placed=4
DEBUG: _ef (int) initial=11, placed=0
DEBUG: _gh (char) initial=18, placed=18
_xx variable is not declared.
_dd variable is not declared.
_xx variable is not declared.
_d variable is not declared.
_j variable is not declared.
_cdf variable is not declared.

Final Symbol Table:
[0]: _ef (int)
[1]: _x (float)
[2]: _ff (int)
[3]: _cc (int)
[4]: _cd (int)
[5]: EMPTY
[6]: EMPTY
[7]: EMPTY
[8]: EMPTY
[9]: _i (int)
[10]: EMPTY
[11]: _bb (int)
[12]: EMPTY
[13]: EMPTY
[14]: EMPTY
[15]: EMPTY
[16]: EMPTY
[17]: EMPTY
[18]: _gh (char)
[19]: _aa (int)
[20]: _ab (int)
[21]: EMPTY
[22]: EMPTY
```