YIĞIT
STACK

Doç. Dr. Aybars UĞUR

Giriş

- Eleman ekleme çıkarmaların en üstten (top) yapıldığı veri yapısına **Yığıt (Stack)** adı verilir.
- Bir eleman ekleneceğinde yığıtın en üstüne konulur. Bir eleman çıkarılacağı zaman yığıtın en üstündeki eleman çıkarılır. Bu eleman da yığıttaki elemanlar içindeki en son eklenen elemandır.
- Bu nedenle yığıtlara **LIFO** (*Last In First Out* : **Son giren ilk çıkar**) listesi de denilir.

Yığıt İşlemleri ve Tanımları

(Tanım) **Boş yığıt (empty stack) :** Elemanı olmayan yığıt.

push (yığıta eleman ekleme) : "push(s,i)", s yığıtının en üstüne i değerini eleman olarak ekler.

pop (yığıttan eleman çıkarma) : "i = pop(s)", s yığıtının en üstündeki elemanı çıkartır ve değerini i değişkenine atar.

empty (yığıtın boş olup olmadığını belirleyen işlem) : empty(s), yığıt boş ise TRUE değerini, değilse FALSE değerini döndürür.

stacktop (yığıttan çıkarılmaksızın en üstteki elemanın değerini döndüren işlem) Denk işlem : (peek)

```
i = pop(s);
push(s,i);
```

(Tanım) **Underflow :** Boş yığıt üzerinden eleman çıkarılmaya veya yığıtın en üstündeki elemanın değeri belirlenmeye çalışıldığında oluşan geçersiz durum. (Çözümü : pop(s) veya stacktop(s) yapılmadan, empty(s) kontrolü yapılarak, boş bir yığıt olup olmadığı belirlenir. Boş ise bu işlemlerin yapılması engellenir.)

Yığıt Kullanımı - I

Bir yığıt ve üzerindeki işlemlerin tanımlandığını düşünelim. İç içe parantezler içeren bir ifadede parantezlerin geçerli olması için :

1. Açılan ve kapanan toplam parantez sayısı eşit olmalıdır. Aç “(“ ve kapa “)” parantezlerin eşitliğine bakılır.
2. Kapanan her parantezden önce bir parantez açılmış olmalıdır. Her “)” için bir “(“ olup olmadığına bakılır.

“((A+B)” ve “A+B(“ 1. şarta uymaz.

“)A+B(-C” ve “(A+B))-(C+D” 2. şarta uymaz.

Problemin çözümü için her açılan parantezde bir geçerlilik alanı açılır ve kapanan parantezde de kapanır. İfadenin herhangi bir noktasındaki Nesting Depth (parantez derinliği) o ana kadar açılmış fakat kapanmamış parantezlerin sayısıdır. İfadenin herhangi bir noktasındaki parantez sayısı = “(“ sayısı – “)” sayısı olarak belirtilebilir. Parantezleri geçerli bir ifadede şu şartlar olmalıdır.

Yığıt Kullanımı - II

1. İfadenin sonunda parantez sayısı 0 olmalıdır. İfadede ya hiç parantez yoktur veya açılan parantezlerin sayısı ile kapanan parantezlerin sayısı eşittir.
2. İfadenin hiçbir noktasında parantez sayısı negatif olmamalıdır. Bu, parantez açılmadan bir parantezin kapanmadığını garantiler.

Aşağıdaki ifade iki şarta da uyar :

7-((x*((x+y)/(j-3))+y)/(4-2.5))

00122234444334444322211222 2 10

Aşağıdaki ifade 1. şarta uymaz :

((A+B)

122221 (İfade sonunda 1 parantez arttı. 0 değil)

Aşağıdaki ifade 2. şarta uymaz :

(A + B)) - (C + D

1 1 1 1 0-1-1 0 0 0 0 (Arada negatif oldu)

Yığıt Kullanımı - III

Problem biraz değiştirildiğinde :

parantezler (parentheses) : “(“, “)”

köşeli parantezler (brackets) : “[“, “]”

küme parantezleri (braces) : “{“, “}”

içerebilen ifadelerin parantez doğruluğu kontrolünün yapılması istendiğinde parantez sayılarının yanında tiplerinin de tutulması gerekecektir. Bu nedenle yukarıdaki yöntemin kullanılması uygun olmaz.

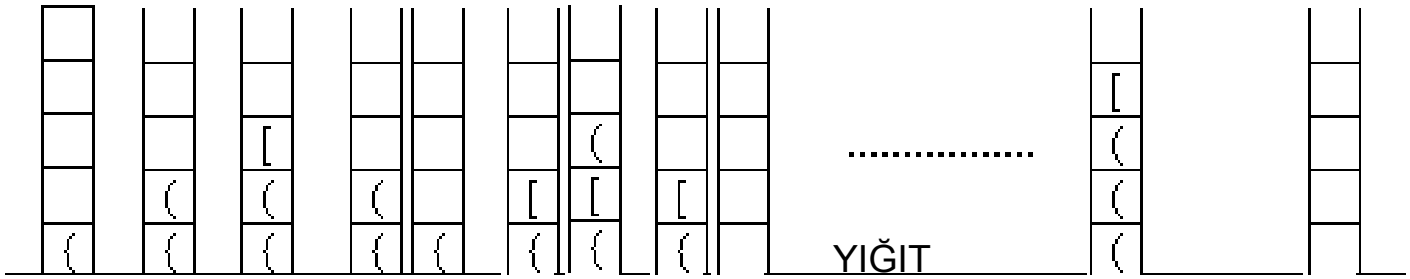
void main() { printf(“Merhaba”};);

yanlış bir ifadedir.

Yığıt Kullanımı - IV

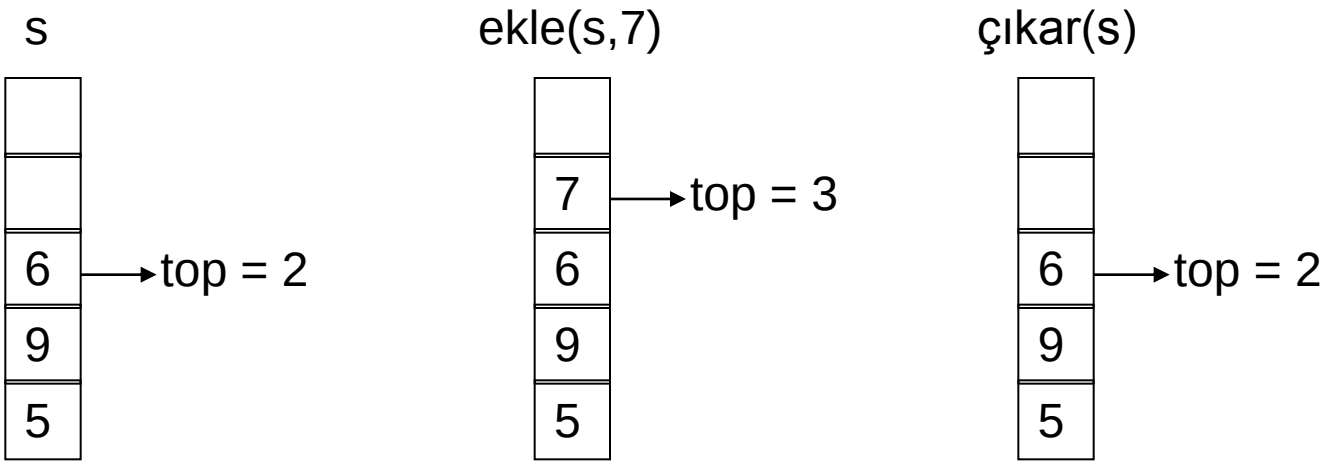
Karşılaşılan parantezleri tutmak üzere yığıt kullanılabilir (Şekil 5.1). Bir parantezle karşılaşıldığında yığıta eklenir. İlgili parantezlerin karşılığı ile karşılaşıldığında ise yığıta bakılır. Yığıt boş değilse yığıttan bir eleman çıkarılarak doğru karşılık olup olmadığı kontrol edilir. Doğruysa işlem sürdürülür. Değilse ifade geçersizdir. Yığıt sonuna ulaşıldığında yığıt boş olmalıdır. Aksi halde açılmış ama kapanmamış parantez olabilir. İfadelerin parantez geçerliliğinin belirlenmesinde kullanılan yığıt :

```
{x+(y-[a+b])*c-[(d+e)]}/(h-(j-(k-[l-n]))).
e  e  e   çç   ee   ççç e  e  e  e   ççç
```



Yığıt Üzerinde Ekleme ve Çıkarma İşlemleri

Yığıtın dizi kullanılarak gerçekleştirimi, s yığıtı üzerinde ekleme ve çıkarma işlemleri :



top = 2'nin anlamı yığıtta 3 eleman vardır ve yığıtın en üstündeki eleman [2]'dir. Boş yığıtta top = -1'dir.

Yığıt Tasarımı ve C# Dilinde Gerçekleştirimi

C# Dilinde Karakter Yığıtı Sınıfı Oluşturulması

yığıtın elemanlarını tutan dizi

```
class StackChar
{
    private int maxSize;
    private char[] stackArray;
    private int top;

    public StackChar(int max)
    {
        maxSize = max;
        stackArray = new char[maxSize];
        top = -1;
    }

    public void push(char j)
    { stackArray[++top] = j; }

    public char pop()
    { return stackArray[top--]; }

    public bool isEmpty()
    { return top == -1; }
}
```

Tasarlanan Karakter Yığıtının Kullanımı

```
static void Main(string[] args)
{
  StackChar y = new StackChar(100);
  string str = "Merhaba";
  for(int i=0; i<str.Length; ++i) y.push(str[i]);
  while(!y.isEmpty()) Console.WriteLine(y.pop());
}
```

Java'ya dönüştürmek için sonraki Java kodunu inceleyerek 6 farkı bulunuz!

- Main -> main
- string -> String
- Console.WriteLine -> System.out.println();
- str[i] -> str.charAt(i)
- bool -> boolean
- ? …..

Java ile Yığıt Tasarımı

JAVA'da Karakter Yığıtı

```
import java.io.*;

class StackChar
{
  private int maxSize;
  private char[] stackArray;
  private int top;

  public StackChar(int max)
  {
    maxSize = max;
    stackArray = new char[maxSize];
    top = -1;
  }

  public void push(char j)
  { stackArray[++top] = j; }

  public char pop()
  { return stackArray[top--]; }

  public boolean isEmpty()
  { return top==-1; }

}
```

Tasarlanan Karakter Yığıtının Kullanımı

```
class Reverse
{
  public static void main(String args[])
  {
    StackChar y = new StackChar(100);
    String str = "Merhaba";
    for(int i=0; i<str.length(); ++i) y.push(str.charAt(i));
    while(!y.isEmpty()) System.out.println(y.pop());
  }
}
```

Generic (Şablon) Tipte Yığıt Sınıfı

Koleksiyonlar içindeki (Hazır) Generic Yığıt Sınıfı

```
using System.Collections.Generic;
…

// string'ler, oluşturulan s yığıtına yerleştirilerek ters sırada listelenmektedir:
string[] str = { "Bilgisayar", "Dolap", "Masa", "Sandalye",
"Sıra" };

Stack<string> s = new Stack<string>();

for(int i=0; i<str.Length;++i)
  s.Push(str[i]);

while(s.Count>0) Console.WriteLine(s.Pop());
// string temp = s.Pop(); da yapılabilir ekrana yazdırmak yerine!
```

Ekran Çıktısı :
Sıra
Sandalye
Masa
Dolap
Bilgisayar

Koleksiyonlar içindeki (Hazır) **Generic Olmayan** Yığıt Sınıfı

```
using System.Collections;

string[] str = { "Bilgisayar", "Dolap", "Masa", "Sandalye", "Sıra" };

Stack s = new Stack();

for(int i=0; i<str.Length;++i)
  s.Push(str[i]);

string temp;

while(s.Count>0) temp = s.Pop(); // Hata!
```

cannot convert object to string -> Tüm elemanları **object** tipindedir.

Console.WriteLine'da hata vermez?

Yığıtı Klonlama

```
string[] str = { "Bilgisayar", "Dolap", "Masa", "Sandalye",
"Sıra" };

Stack s = new Stack();

for(int i=0; i<str.Length;++i)
  s.Push(str[i]);

Stack klon = (Stack)s.Clone();

while(s.Count>0) Console.WriteLine(s.Pop()); // s yığıtı bitti

Console.WriteLine(klon.Pop());
```

Hazır Stack Sınıfı Metotları

- http://msdn.microsoft.com/en-us/library/1f1xe273.aspx

- Peek, CopyTo ve ToArray metotlarını deneyebilirsiniz.

Java'da hazır Stack (yığıt) sınıfı

String'ler, oluşturulan s yığıtına yerleştirilerek ters sırada listelenmektedir:

```
import java.util.*;

public class StackTest
{
  public static void main(String args[])
  {
    String str[] = { "Bilgisayar", "Dolap", "Masa",
                     "Sandalye", "Sıra" };

    Stack s = new Stack();

    for(int i=0; i<str.length;++i)
      s.push(str[i]);

    while(!s.empty()) System.out.println(s.pop());
  }
}
```

C# ile Generic Yığıt Sınıfı Oluşturulması

- Generic'ler, veri işleme algoritmalarının, Veri Tipine özel kodu tekrarlamaksızın yeniden kullanılabilmesini sağlarlar.
- C++'teki Template'lere benzerler, ancak farklılıkları vardır.
- Generic'ler, tip güvenli (type-safe) sınıflar yazmayı sağlarlar.
- Sınıflarınızı < **Tip** > parantezleri içerisinde gösterdiğiniz, istediğiniz **Tip** ile kullanabilirsiniz.

C# ile Generic Yığıt Sınıfı Oluşturulması

An Introduction to C# Generics'ten

http://msdn.microsoft.com/en-us/library/ms379564%28v=vs.80%29.aspx

```
public class MyStack<T>
{
  readonly int m_Size;
  int m_StackPointer = 0;
  T[] m_Items;

  public MyStack()
      : this(100)
  { }

  public MyStack(int size)
  {
      m_Size = size;
      m_Items = new T[m_Size];
  }
```

```
public void Push(T item) {
  if (m_StackPointer >= m_Size)
    throw new StackOverflowException();
  m_Items[m_StackPointer] = item;
  m_StackPointer++;
}

public T Pop() {
  m_StackPointer--;
  if (m_StackPointer >= 0)
  { return m_Items[m_StackPointer]; }
  else {
    m_StackPointer = 0;
    throw new InvalidOperationException("Cannot
pop an empty stack");
   }
} // Pop method
} // class
```

Tasarlanan Generic Yığıt Sınıfının Kullanımı

```
// string Yığıtı olarak kullanımı
 MyStack<string> ms1 = new MyStack<string>();
 ms1.Push("Masa");
 ms1.Push("Sandalye");
 ms1.Push("Tahta");
 string s = ms1.Pop();
 Console.WriteLine(s);

// int Yığıtı olarak kullanımı
 MyStack<int> ms2 = new MyStack<int>();
 ms2.Push(5);
 ms2.Push(6);
 Console.WriteLine(ms2.Pop() + ms2.Pop());
 // int sayi = ms2.Pop(); // Hata verir!
```

Ekran Çıktısı :
Tahta
11

Tasarlanan Generic Yığıt Sınıfının Kullanımı

```
public class Ogrenci {
  public string ad, soyad;
  public Ogrenci(string adarg, string soyadarg)
  {
    ad = adarg; soyad = soyadarg;
  }
}
…
// Ogrenci Yığıtı olarak kullanımı
 MyStack<Ogrenci> msOgr = new MyStack<Ogrenci>();
 Ogrenci ogr1 = new Ogrenci("Ali","Yılmaz");
 msOgr.Push(ogr1);
 msOgr.Push(new Ogrenci("Veli","Çetin"));
 Console.WriteLine(msOgr.Pop().soyad);
 Console.WriteLine(msOgr.Pop().soyad);
```

Ekran Çıktısı :
Çetin
Yılmaz

INFIX, POSTFIX, PREFIX

Bu kısımda bilgisayar alanındaki önemli konulardan biri olan infix, postfix ve prefix kavramları üzerinde durulacak ve bu kavramlarda yığıt kullanımı gösterilecektir.

A+B

operator (işlemci) : +

operands (işlenenler) : A, B

infix gösterim : A+B

prefix gösterim : +AB (benzeri bir gösterim add(A,B) fonksiyonu)

postfix gösterim : AB+

in, pre ve post, operator'ün operand'lara göre yerine karşılık gelir. Infix gösterimde işlemci (+), işlenenlerin (A,B) arasında yer alır. Prefix gösterimde işaretçi, işlenenlerden önce gelir, postfix gösterimde de sonra gelir.

Infix'ten Postfix'e Çevirme

A+B*C infix ifadesini postfix'e çevirelim.

Bunun için işlem önceliğine bakmak gerekir. Çarpmanın toplamaya önceliği olduğu için, A+(B*C) şeklinde düşünülebilir. Önce çarpma kısmı postfix'e çevrilecek sonra da sonucu.

A+(B*C) anlaşılırlığı artırmak için parantez kullandık.

A+(BC*) çarpım çevrildi.

A(BC*)+ toplam çevrildi.

ABC*+ postfix form.

İşlem önceliği (büyükten küçüğe)

- Üs alma
- Çarpma/Bölme
- Toplama/Çıkarma

Parantezsiz ve aynı önceliğe sahip işlemcilerde işlemler soldan sağa doğru yapılır (üs alma hariç). Üs almada sağdan sola doğrudur. A-B+C 'de öncelik (A-B)+C şeklindedir. A^B^C'de ise A^(B^C) şeklindedir. Parantezler default öncelikleri belirtmek için konulmuştur.

Çevrim Örnekleri

Infix	Postfix	Prefix
A+B-C	AB+C-	-+ABC
(A+B)*(C-D)	AB+CD-*	*+AB-CD
A^B*C-D+E/F/(G+H)	AB^C*D-EF/GH+/+	+-*ÂBCD//EF+GH
((A+B)*C-(D-E))^(F+G)	AB+C*DE—FG+^	^-*+ABC-DE+FG
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

Dikkat edilecek olunursa, postfix ile prefix ifadeler birbirinin ayna görüntüsü değildir.

Postfix Formun Özellikleri

Postfix formda parantez kullanımına gerek yoktur. İki infix ifadeyi düşünün : A+(B*C) ve (A+B)*C. İlk ifadede parantez gereksizdir. İkincide ilk ifade ile karıştırılmaması için gereklidir. Postfix forma çevirmek bu karışıklığı önler. (A+B)*(C+D)'yi infix formda parantezsiz ifade etmek için çarpım işlemi yapılırsa işlem sayısı çoğalır.

Infix formdan postfix forma çevrilen bir ifadede operand'ların (sayı veya sembol) bağlı olduğu operator'leri (+,-,*,/) görmek zorlaşır (3 4 5 * + ifadesinin sonucunun 23'e, 3 4 + 5 * ifadesinin sonucunun 35'e karşılık geldiğini bulmak zor gibi görünür). Fakat parantez kullanmadan tek anlama gelen bir hale dönüşür. İşlemleri, hesaplamaları yapmak kolaylaşır.

Postfix İfadenin Sonucunun Hesaplanması

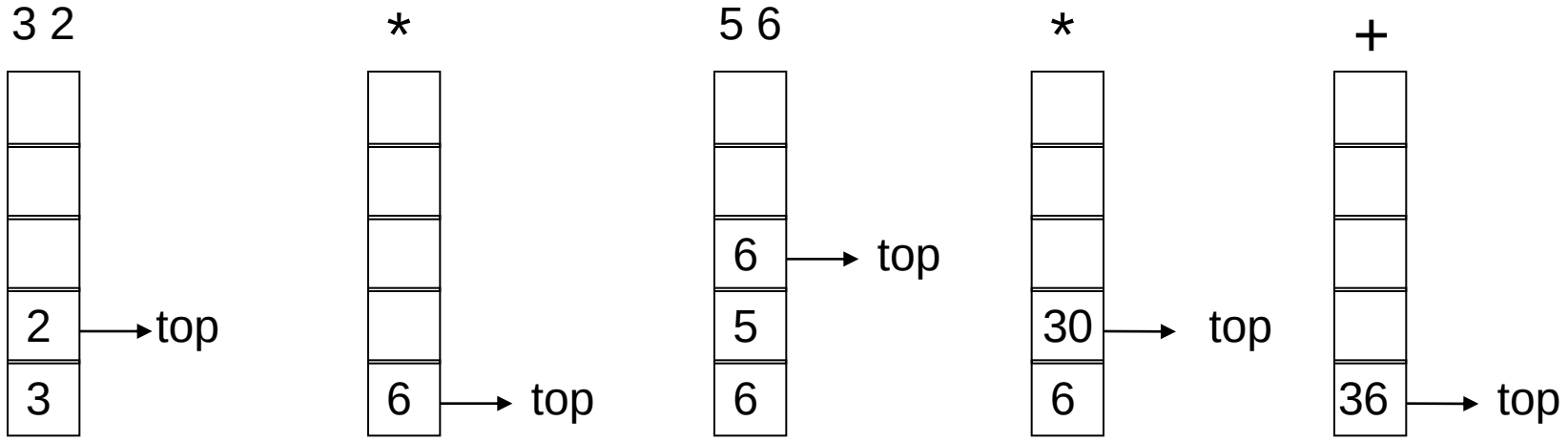
postfix ifadenin sonucunun hesaplanması (ve bunu gerçekleştiren algoritma):

Bir postfix string'inde her operator kendinden önce gelen iki operand üzerinde işlem yapacaktır. Bu operandlar daha önceki operator'lerin sonuçları da olabilir. Bir anda ifadeden bir operand okunarak yığıta yerleştirilir. Bir operator'e ulaşıldığında, yığıtın en üstündeki iki eleman bu operator'ün operand'ları olacaktır. İki eleman yığıttan çıkarılır işlem yapılır ve sonuç tekrar yığıta konulur. Artık bir sonraki operator'ün operand'ı olmaya hazırdır.

Birçok derleyici 3*2+5*6 gibi bir infix ifadenin değerini hesaplayacağı zaman postfix forma dönüştürdükten (belirsizliği ortadan kaldırdıktan sonra) sonucu hesaplar : "3 2 * 5 6 * +"

Bir postfix ifadenin “3 2 * 5 6 * +” sonucunun hesaplanması

Bir operatöre ulaşıldığında,
son iki eleman yığıttan çıkarılır, işlem yapılır, sonuç yığıta yerleştirilir:



Postfix ifadelerin sonucunu hesaplayan algoritma

Algoritma : (Bir postfix ifadenin sonucunu hesaplar)

```
opndstk = the empty stack;
// scan the input string reading one element at a time into symb
while (not end of input) {
    symb = next input character;
    if (symb is an operand)
        push(opndstk,symb);
    else {
        // symb is an operator
        opnd2 = pop(opndstk);
        opnd1 = pop(opndstk);
        value = result of applying symb (case *,/,+,-) to opnd1 and opnd2
        push(opndstk,value);
    } // end else
} // end while
return(pop(opndstk));
```

Infix ifadeleri Postfix'e çeviren algoritma üzerinde düşününüz!

Fonksiyon ve Metot Çağrımları - I

Yığıtlar, listelerin ters sırada yazdırılması, palindrom (okunduğunda ve tersten okunduğunda aynı sonucu veren karakter dizisi) benzeri yapıların bulunması, bir ifadedeki parantez gibi sembollerin geçerliliğinin test edilmesi, ifadelerin sonuçlarının hesaplanıp değerlerinin elde edilmesi, infix ifadelerin postfix ifadeye dönüştürülmesi gibi amaçlarla kullanılabildiği gibi, programlama dili derleyicileri (compiler) fonksiyon çağrımlarında da yığıtlardan yararlanırlar.

Bir fonksiyon çağrıldığında, çağıran fonksiyonun yerel değişkenleri sistem tarafından kaydedilmelidir; aksi halde çağrılan fonksiyon çağıran fonksiyonun değişkenlerini ve değerlerini ortadan kaldıracaktır. Ayrıca çağıran fonksiyonda kalınan nokta (geri dönüş adresi), çağrılan fonksiyonun bitmesinden sonra geri dönmek üzere tutulmalıdır.

Fonksiyon ve Metot Çağrımları - II

Soyut olarak bakıldığında, yeni bir fonksiyonun çağrılması ile çağıran fonksiyonun değişkenleri ve geri dönülecek adres bir kağıda kaydedilir ve daha önce çağrılan fonksiyonlara ilişkin bilgilerin tutulduğu kağıtların üzerine konulur. Bundan sonra denetim çağrılan fonksiyona geçer. Kendi değişkenlerine yer açarak onlar üzerinde işlemler yapılmasını sağlar. Fonksiyondan geri dönüleceği zaman en üstteki kağıda bakılıp değişkenler üzerinde ilgili değişiklikler yapılarak geri dönüş adresine atlanır. Derleyici bu işlemleri kağıt yerine yığıt kullanarak gerçekleştirir.