Bölüm 8: Yığın ve Kuyruk Uygulamaları

C ile Veri Yapıları ve Algoritmalar

Labirent | Geri izleme | İşleç önceliği | Infix / postfix / prefix | Postfix hesaplama | Parantez denetimi

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Geri izlemeyi kodlamak

Labirentte yığınla yol arayıp çıkmazdan geri dönmeyi izlemek.

En kötü durumu kurmak

Bir algoritma için en kötü veri grubunu tasarlayıp analiz etmek.

Gösterimleri çevirmek

Infix, prefix ve postfix arasında elle ve yığınla çeviri yapmak.

Postfix hesaplamak

`eval` yordamını işlenen/işleç yığınıyla adım adım çalıştırmak.

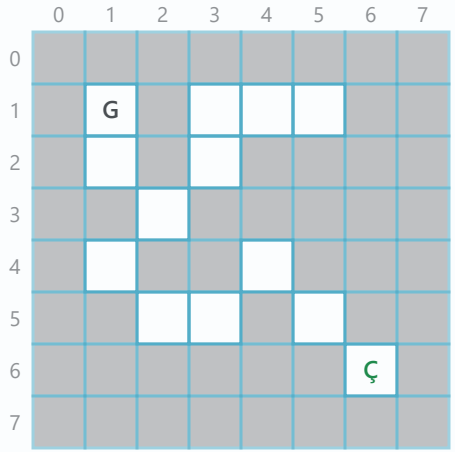
Önceliği kodlamak

isp/icp tablosuyla infix → postfix çeviren yordamı yazmak.

Yığını genellemek

Parantez denetiminde, geri al/yinele gibi problemlerde ve özyinelemesiz gezinmede yığını kullanmak.

Labirent (Maze) Problemi



m×p boyutlu bir matriste **0**'lar açık yolu, **1**'ler duvarı simgeler. Giriş [1][1] (G), çıkış [6][6] (Ç): girişten çıkışa hangi yoldan gidilebileceği bulunmak istenir.

- Yatay, dikey ve **çapraz**: 8 yönde hareket edilebilir.
- Her adımda "matris dışına çıkıyor muyum?" denetiminden kurtulmak için matrisin **çevresi 1 ile çevrilir**.
- Gidilen hücreler `gezildi` matrisinde işaretlenir: aynı yere iki kez girilmez.
- Çıkmaza girilince son yol ayrımına **geri dönülür**: son giren ilk çıkar → yığın.

Sekiz Yön ve `hareket[]` Tablosu

yön	hareket[yön].dikey	hareket[yön].yatay
0	−1	0
1	−1	1
2	0	1
3	1	1
4	1	0
5	1	−1
6	0	−1
7	−1	−1

7 [i−1][j−1]	0 [i−1][j]	1 [i−1][j+1]
6 [i][j−1]	yön [i][j]	2 [i][j+1]
5 [i+1][j−1]	4 [i+1][j]	3 [i+1][j+1]

Yön 0 kuzey; saat yönünde 1…7. Yeni konum: [i + dikey][j + yatay].

Sekizlik **arama tablosu** (look-up table) sekiz ayrı `if` yazmaktan kurtarır: yön bir döngü değişkeni olur.

Labirent İçin Tanımlar

```
#define MAX_STACK_SIZE 100
#define CIKIS_SATIR 6
#define CIKIS_SUTUN 6
typedef struct { short int dikey, yatay; } yonler;
yonler hareket[8] = {{-1, 0}, {-1, 1}, {0, 1}, {1, 1},
                     {1, 0}, {1, -1}, {0, -1}, {-1, -1}};
typedef struct { short int satir, sutun, yon; } element;
element yigit[MAX_STACK_SIZE];
int top = -1;
short int gezildi[8][8];
short int labirent[8][8] = {{1,1,1,1,1,1,1,1},
    {1,0,1,0,0,0,1,1}, {1,0,1,0,1,1,1,1},
    {1,1,0,1,1,1,1,1}, {1,0,1,1,0,1,1,1},
    {1,1,0,0,1,0,1,1}, {1,1,1,1,1,1,0,1},
    {1,1,1,1,1,1,1,1}};
```

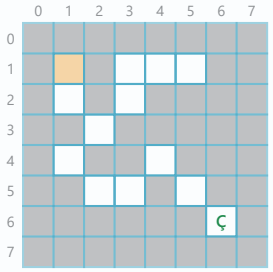
- `element` : yığına konan kayıt, **hangi hücreden** (satir, sutun) ve geri dönülürse **hangi yönden** devam edileceği (yon).
- `gezildi` : ziyaret edilen hücreler (küresel olduğu için başta sıfırdır).
- `labirent` : çevresi 1 ile çevrili 8×8 matris; iç kısım 6×6.
- Yığının en fazla boyutu: açık hücre sayısı kadar (burada 13).

`izbul` Yordamı

```
void izbul(void) {
  short int satir, sutun, y_satir, y_sutun;
  int i, yon, bulundu = FALSE;
  element konum;
  yigit[0].satir = yigit[0].sutun = 1;
  yigit[0].yon = 1;      /* ilk konum yığına */
  gezildi[1][1] = 1;  top = 0;
  while (top > -1 && !bulundu) {
    konum = yigit_al(&top);   /* yeni konum */
    satir = konum.satir;
    sutun = konum.sutun;
    yon = konum.yon;          /* hangi yöne */
    while (yon < 8 && !bulundu) {
      y_satir = satir + hareket[yon].dikey;
      y_sutun = sutun + hareket[yon].yatay;
      if (y_satir == CIKIS_SATIR &&
          y_sutun == CIKIS_SUTUN)
        bulundu = TRUE;       /* çıkış mı? */
      else if (!labirent[y_satir][y_sutun] &&
               !gezildi[y_satir][y_sutun]) {
        gezildi[y_satir][y_sutun] = 1;
        konum.satir = satir;  /* nereden */
        konum.sutun = sutun;
        konum.yon = ++yon;    /* dönüşte yön */
        yigit_ekle(&top, konum);
        satir = y_satir;  sutun = y_sutun;
        yon = 0;
      } else ++yon;           /* başka yön */
    }                         /* geri adım */
  }
```

Sonunda `bulundu` ise yığındaki konumlar (0…top), son konum ve çıkış sırayla yazdırılır; değilse “Cikis yolu bulunamadi!”. (Kodu okunur kılmak için satırlar bölündü; mantık aynen korundu.)

Kod İzleme: Yığınla Geri İzleme



Başlangıç: yığında **(1,1) yön 1**. `yigit` : satir,sutun,yon kayıtları (alttan üste).

Adım 1 / 18

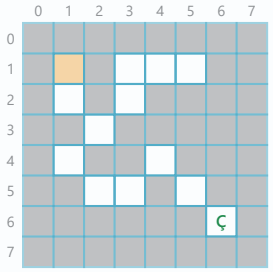
DEĞİŞKENLER

`bulundu = 0` `yigit = 1,1,1`

ÇIKTI

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
12       gezildi[ys][yy] = 1;
13       konum = (element){satir, sutun, ++yon};
14       yigit_ekle(&top, konum);
15       satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



İlk **pop**: (1,1), yön 1'den başlanır (yön 0 zaten duvar).

Adım 2 / 18

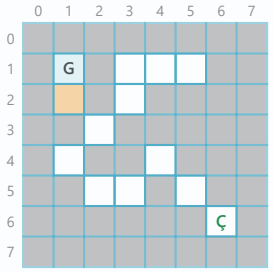
DEĞİŞKENLER

`satir = 1` `sutun = 1` `yon = 1` `yigit = boş`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(1,1)'den yön 4 (G) açık → **(2,1)**'e geç; (1,1, yön 5) yığına itildi.

Adım 3 / 18

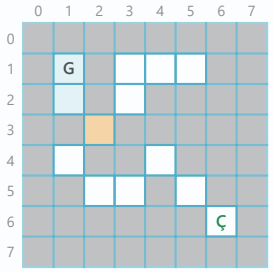
DEĞİŞKENLER

`satir = 2` `sutun = 1` `yon = 0` `yigit = 1,1,5`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(2,1)'den yön 3 (GD) açık → **(3,2)**'e geç; (2,1, yön 4) yığına itildi.

Adım 4 / 18

DEĞİŞKENLER

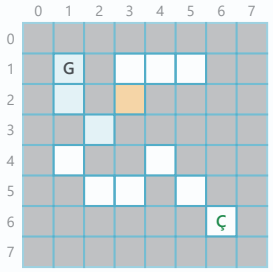
`satir = 3` `sutun = 2` `yon = 0` `yigit = 1,1,5 2,1,4`

ÇIKTI

```
```

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
▶        gezildi[ys][yy] = 1;
▶        konum = (element){satir, sutun, ++yon};
▶        yigit_ekle(&top, konum);
▶        satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



(3,2)'den yön 1 (KD) açık → **(2,3)**'e geç; (3,2, yön 2) yığına itildi.

Adım 5 / 18

DEĞİŞKENLER

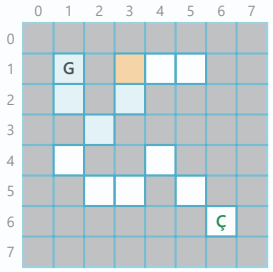
`satir = 2` `sutun = 3` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,2`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(2,3)'den yön 0 (K) açık → **(1,3)**'e geç; (2,3, yön 1) yığına itildi.

Adım 6 / 18

DEĞİŞKENLER

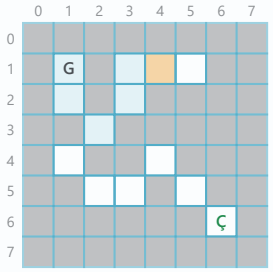
`satir = 1` `sutun = 3` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,2 2,3,1`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(1,3)'den yön 2 (D) açık → **(1,4)**'e geç; (1,3, yön 3) yığına itildi.

Adım 7 / 18

DEĞİŞKENLER

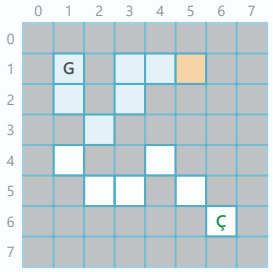
`satir = 1` `sutun = 4` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,2 2,3,1 1,3,3`

ÇIKTI

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
▶        gezildi[ys][yy] = 1;
▶        konum = (element){satir, sutun, ++yon};
▶        yigit_ekle(&top, konum);
▶        satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



(1,4)'den yön 2 (D) açık → **(1,5)**'e geç; (1,4, yön 3) yığına itildi.

Adım 8 / 18

DEĞİŞKENLER

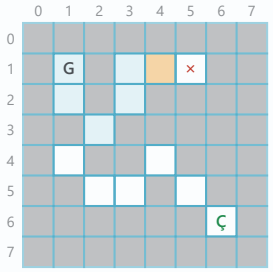
`satir = 1` `sutun = 5` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,2 2,3,1 1,3,3 1,4,3`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



Geri adım / pop: (1,4) konumuna dönülür, yön 3'ten (GD) denemeye devam edilir.

Adım 9 / 18

DEĞİŞKENLER

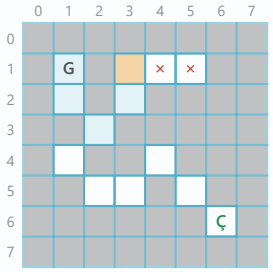
`satir = 1` `sutun = 4` `yon = 3`

`yigit = 1,1,5 2,1,4 3,2,2 2,3,1 1,3,3`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



Geri adım / pop: (1,3) konumuna dönülür, yön 3'ten (GD) denemeye devam edilir.

Adım 10 / 18

DEĞİŞKENLER

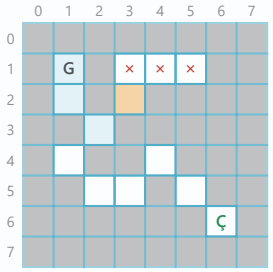
`satir = 1` `sutun = 3` `yon = 3`

`yigit = 1,1,5 2,1,4 3,2,2 2,3,1`

ÇIKTI

```
1 while (top > -1 && !bulundu) {
▶   konum = yigit_al(&top);
▶   satir = konum.satir;  sutun = konum.sutun;
▶   yon = konum.yon;
5   while (yon < 8 && !bulundu) {
6     ys = satir + hareket[yon].dikey;
7     yy = sutun + hareket[yon].yatay;
8     if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9       bulundu = TRUE;
10    else if (!labirent[ys][yy] &&
11             !gezildi[ys][yy]) {
12      gezildi[ys][yy] = 1;
13      konum = (element){satir, sutun, ++yon};
14      yigit_ekle(&top, konum);
15      satir = ys;  sutun = yy;  yon = 0;
16    } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



Geri adım / pop: (2,3) konumuna dönülür, yön 1'den (KD) denemeye devam edilir.

Adım 11 / 18

DEĞİŞKENLER

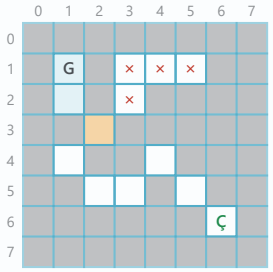
`satir = 2` `sutun = 3` `yon = 1`

`yigit = 1,1,5 2,1,4 3,2,2`

ÇIKTI

```
1 while (top > -1 && !bulundu) {
▶   konum = yigit_al(&top);
▶   satir = konum.satir;  sutun = konum.sutun;
▶   yon = konum.yon;
5   while (yon < 8 && !bulundu) {
6     ys = satir + hareket[yon].dikey;
7     yy = sutun + hareket[yon].yatay;
8     if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9       bulundu = TRUE;
10    else if (!labirent[ys][yy] &&
11             !gezildi[ys][yy]) {
12      gezildi[ys][yy] = 1;
13      konum = (element){satir, sutun, ++yon};
14      yigit_ekle(&top, konum);
15      satir = ys;  sutun = yy;  yon = 0;
16    } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



Geri adım / pop: (3,2) konumuna dönülür, yön 2'den (D) denemeye devam edilir.

Adım 12 / 18

DEĞİŞKENLER

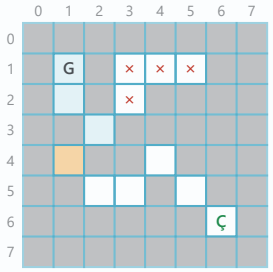
`satir = 3` `sutun = 2` `yon = 2`

`yigit = 1,1,5 2,1,4`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(3,2)'den yön 5 (GB) açık → **(4,1)**'e geç; (3,2, yön 6) yığına itildi.

Adım 13 / 18

DEĞİŞKENLER

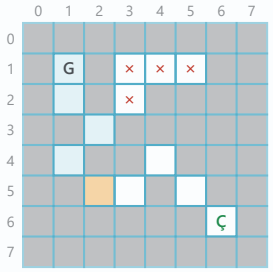
`satir = 4` `sutun = 1` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,6`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(4,1)'den yön 3 (GD) açık → **(5,2)**'e geç; (4,1, yön 4) yığına itildi.

Adım 14 / 18

DEĞİŞKENLER

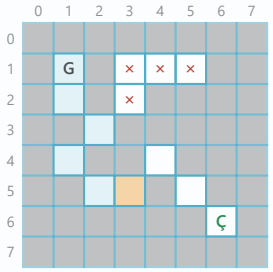
`satir = 5` `sutun = 2` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,6 4,1,4`

ÇIKTI

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
▶        gezildi[ys][yy] = 1;
▶        konum = (element){satir, sutun, ++yon};
▶        yigit_ekle(&top, konum);
▶        satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



(5,2)'den yön 2 (D) açık → **(5,3)**'e geç; (5,2, yön 3) yığına itildi.

Adım 15 / 18

DEĞİŞKENLER

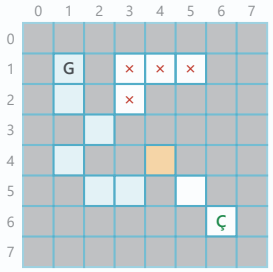
`satir = 5` `sutun = 3` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,6 4,1,4 5,2,3`

ÇIKTI

```
while (top > -1 && !bulundu) {
  konum = yigit_al(&top);
  satir = konum.satir;  sutun = konum.sutun;
  yon = konum.yon;
  while (yon < 8 && !bulundu) {
    ys = satir + hareket[yon].dikey;
    yy = sutun + hareket[yon].yatay;
    if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
      bulundu = TRUE;
    else if (!labirent[ys][yy] &&
             !gezildi[ys][yy]) {
      gezildi[ys][yy] = 1;
      konum = (element){satir, sutun, ++yon};
      yigit_ekle(&top, konum);
      satir = ys;  sutun = yy;  yon = 0;
    } else ++yon;
} }
```

Kod İzleme: Yığınla Geri İzleme



(5,3)'den yön 1 (KD) açık → **(4,4)**'e geç; (5,3, yön 2) yığına itildi.

Adım 16 / 18

DEĞİŞKENLER

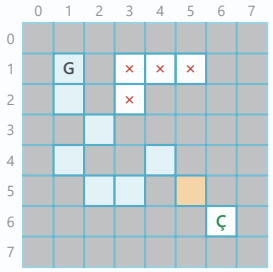
`satir = 4` `sutun = 4` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,6 4,1,4 5,2,3 5,3,2`

ÇIKTI

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
▶        gezildi[ys][yy] = 1;
▶        konum = (element){satir, sutun, ++yon};
▶        yigit_ekle(&top, konum);
▶        satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



(4,4)'den yön 3 (GD) açık → **(5,5)**'e geç; (4,4, yön 4) yığına itildi.

Adım 17 / 18

DEĞİŞKENLER

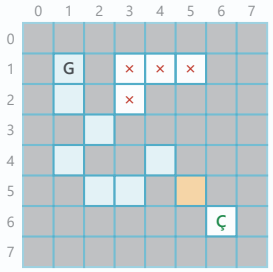
`satir = 5` `sutun = 5` `yon = 0`

`yigit = 1,1,5 2,1,4 3,2,6 4,1,4 5,2,3 5,3,2 4,4,4`

ÇIKTI

```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
9        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
▶        gezildi[ys][yy] = 1;
▶        konum = (element){satir, sutun, ++yon};
▶        yigit_ekle(&top, konum);
▶        satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Kod İzleme: Yığınla Geri İzleme



Yön 3 (GD) **çıkış**! Yol: yığın + (5,5) + (6,6).

Adım 18 / 18

DEĞİŞKENLER

`bulundu = 1`

```
yigi  1,1,5 2,1,4 3,2,6 4,1,4
t =   5,2,3 5,3,2 4,4,4
```

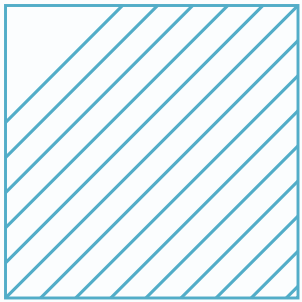
ÇIKTI

```
11 21 32 41 52 53 44 55 66
```

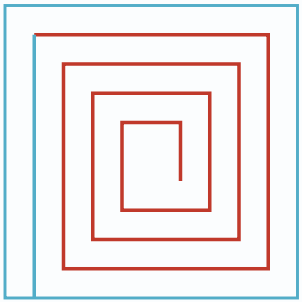
```
1  while (top > -1 && !bulundu) {
2    konum = yigit_al(&top);
3    satir = konum.satir;  sutun = konum.sutun;
4    yon = konum.yon;
5    while (yon < 8 && !bulundu) {
6      ys = satir + hareket[yon].dikey;
7      yy = sutun + hareket[yon].yatay;
8      if (ys == CIKIS_SATIR && yy == CIKIS_SUTUN)
▶        bulundu = TRUE;
10     else if (!labirent[ys][yy] &&
11              !gezildi[ys][yy]) {
12       gezildi[ys][yy] = 1;
13       konum = (element){satir, sutun, ++yon};
14       yigit_ekle(&top, konum);
15       satir = ys;  sutun = yy;  yon = 0;
16     } else ++yon;
17 } }
```

Değerlendirme ve En Kötü Durum

- Sekizlik **arama tablosu** kodu kısalttı ve işi kolaylaştırdı.
- Program labirentte yığın yapısıyla **kaba kuvvet** (brute force) araması yapar.
- En kötü durum analizi için **en kötü veri grubu** bulunmalıdır; yönlerin tarama sırası belirleyicidir.
- **Girdi 1**: ilk denenen yol uzun ama doğru → yalnız ekleme, silme yok.
- **Girdi 2**: ilk denenen yol uzun ve **yanlış** → sonuna kadar gidilir, yığın bir bir boşaltılır; doğru yol dışındaki her yer gezildikten sonra çözüme ulaşılır. En kötü veri grubu budur.



Girdi 1



Girdi 2

HOCADAN "Analizi bile beceremeden bir algoritmaya burun kıvrılmaz!"

LABİRENT

Karmaşıklık ve Kuyrukla Karşılaştırma

Zaman

Her açık hücre en çok bir kez itilir (`gezildi`), her hücrede en çok 8 yön denenir: m×p labirentte **O(m·p)**.

Bellek

Yığın en fazla açık hücre sayısı kadar büyür: **O(m·p)**; `gezildi` da O(m·p).

Yığın (DFS)

Bulunan yol en kısa olmak zorunda değil. Bu labirentte (2,3)→(1,3)→(1,4)→(1,5) çıkmazına girip döndü; bulduğu yol 9 hücre.

Kuyruk (BFS)

Hücreleri uzaklık sırasıyla açar: **en kısa** yolu garanti eder. Burada o da 9 hücrelik aynı yolu bulur; ama genelde DFS'ninki daha uzun olabilir.

HOCADAN "Bir kodu ayrıntılı inceleyebilecek kişi ancak onu yazandır. Bir başkası o kodu tam olarak analiz edemez."

HOCADAN "Öğrenci sayısı 120-130 değil de 30-40 olsa ve yanımda da 4-5 tane asistanım olsa o zaman ödevleri daha ayrıntılı inceleme imkânı bulabilirdim."

İfade Değerlendirme (Evaluation of Expressions)

Mantıksal ve aritmetik ifadelerin derleyiciler tarafından değerlendirilmesinde de yığın kullanılır. Derleyici şu ifadelerin üstesinden nasıl geliyor?

Öncelik

`x = a / b - c + d * e - a * c`

Değer

`((4 / 2) - 2) + (3 * 3) - (4 * 2)` = 1

Parantez

`a / (b - c) + d * (e - a) * c`

- Soldan sağa okurken bir işleç hemen uygulanamaz: sağında daha öncelikli bir işleç olabilir.
- Çözüm iki aşamalı: önce ifade önceliğe göre **postfix**'e çevrilir, sonra postfix tek geçişte **hesaplanır**. İkisi de yığınla yapılır.

HOCADAN "Problem önce araştırılmalı; o soru başkaları tarafından defalarca çözülmüştür. En hızlı yolu bulunmalı. Çözülmemişse, o zaman keşif başlar."

C'de İşleç Önceliği (Özet)

Öncelik	İşleçler	Birleşme
en yüksek	`()` `[]` `→` `.`	soldan sağa
	tekli `-` `!` `++` `--` `*` `&`	sağdan sola
	`*` `/` `%`	soldan sağa
	`+` `-`	soldan sağa
	`<` `≤` `>` `≥`	soldan sağa
	`==` `≠`	soldan sağa
	`&&`	soldan sağa
en düşük	`||`	soldan sağa

- Önce öncelik, eşitlikte **birleşme** yönü karar verir: `a - b - c` = `(a - b) - c`.
- Parantez her şeyi geçersiz kılar.
- Örnek: `a / b - c + d * e - a * c` = `(((a / b) - c) + (d * e)) - (a * c)`.
- Bu bölümün algoritmaları `+ - * / %` ve parantezle sınırlıdır (Horowitz–Sahni, Şekil 3.12).

İFADE GÖSTERİMLERİ

Infix, Prefix, Postfix

- **Infix** (ara ek): işleç işlenenlerinin **arasında**, `a + b`. Hem parantezleri hem öncelikleri dikkate almak gerekir.
- **Prefix** (önek, Polonya gösterimi): işleç **önde**, `+ a b`.
- **Postfix** (sonek, ters Polonya): işleç **sonda**, `a b +`. Prefix ve postfix'te **parantez gerekmez**, öncelik düşünmeden işlem yapılır.

No	Prefix	Infix	Postfix
1	`*34`	`3*4`	`34*`
2	`+2*34`	`2+3*4`	`234*+`
3	`-*+abc/da`	`(a+b)*c-d/a`	`ab+c*da/-`
4	`**/a+-bcd-eac`	`(a/(b-c+d))*(e-a)*c`	`abc-d+/ea-*c*`

Elle Çevirme: Parantezleme, Taşıma, Silme

```
a / b − c + d * e − a * c
```

Başlangıç ifadesi.

Adım 1 / 4

İşlenenlerin **sırası değişmez**; yalnız işleçlerin yeri değişir. Bu gözlem, yığınlı algoritmanın temelidir: işlenen gelir gelmez çıktıya yazılır, işleçler uygun an gelene kadar yığında bekler.

Elle Çevirme: Parantezleme, Taşıma, Silme

```
((((a / b) - c) + (d * e)) - (a * c))
```

1. Öncelik ve birleşmeye göre **tam parantezlenir**.

İşlenenlerin **sırası değişmez**; yalnız işleçlerin yeri değişir. Bu gözlem, yığınlı algoritmanın temelidir: işlenen gelir gelmez çıktıya yazılır, işleçler uygun an gelene kadar yığında bekler.

Elle Çevirme: Parantezleme, Taşıma, Silme

```
((((a b /) c -) (d e *) +) (a c *) -)
```

2. Her işleç kendi parantezinin **sağ** kapanışına taşınır.

Adım 3 / 4

İşlenenlerin **sırası değişmez**; yalnız işleçlerin yeri değişir. Bu gözlem, yığınlı algoritmanın temelidir: işlenen gelir gelmez çıktıya yazılır, işleçler uygun an gelene kadar yığında bekler.

Elle Çevirme: Parantezleme, Taşıma, Silme

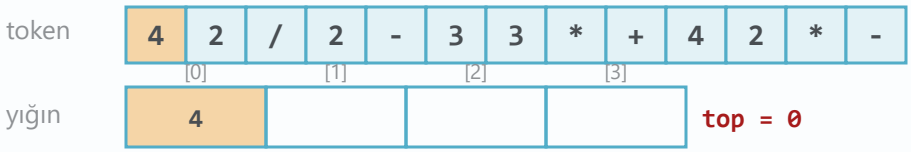
```
a b / c − d e * + a c * −
```

3. Parantezler silinir: **postfix**. (Prefix için işleç **sol** açılışa taşınır: `− + − / a b c * d e * a c` .)

Adım 4 / 4

İşlenenlerin **sırası değişmez**; yalnız işleçlerin yeri değişir. Bu gözlem, yığınlı algoritmanın temelidir: işlenen gelir gelmez çıktıya yazılır, işleçler uygun an gelene kadar yığında bekler.

Postfix → Infix: `42/2-33*+42*-`

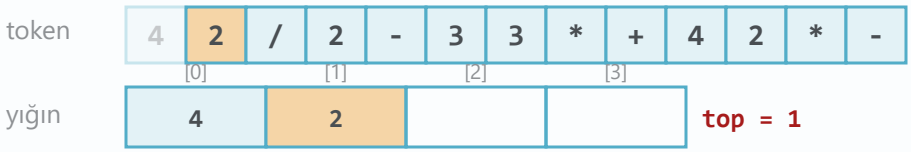


4 işlenen → yığına.

Adım 1 / 13

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

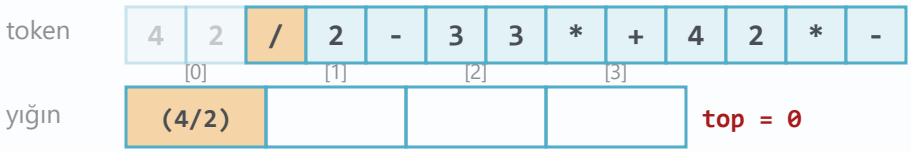
Postfix → Infix: `42/2−33*+42*−`



2 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

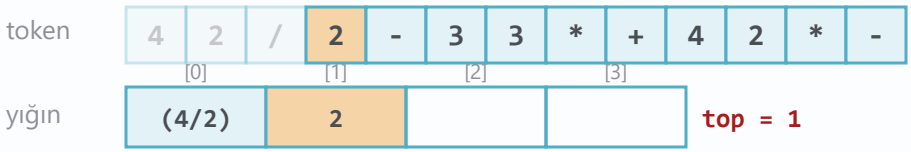
Postfix → Infix: `42/2−33*+42*−`



/ işleç → iki öğe alınır (4, 2), `(4/2)` olarak geri konur.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

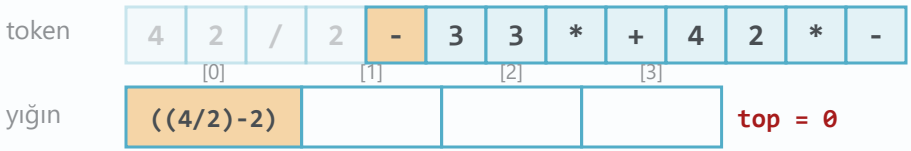
Postfix → Infix: `42/2−33*+42*−`



2 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

Postfix → Infix: `42/2−33*+42*−`

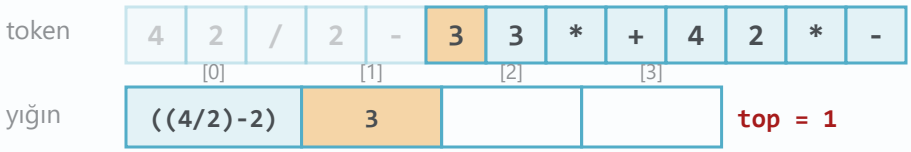


- işleç → iki öğe alınır ((4/2), 2), `((4/2)−2)` olarak geri konur.

Adım 5 / 13

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

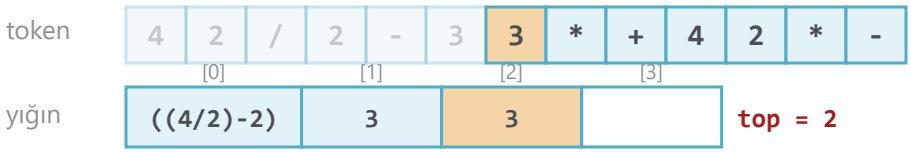
Postfix → Infix: `42/2−33*+42*−`



3 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

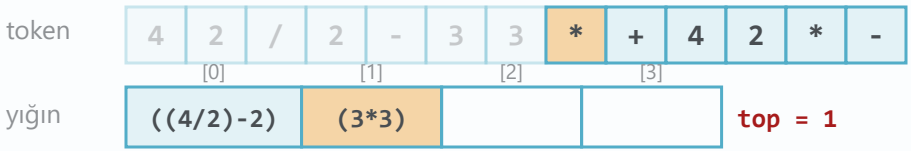
Postfix → Infix: `42/2−33*+42*−`



3 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

Postfix → Infix: `42/2−33*+42*−`

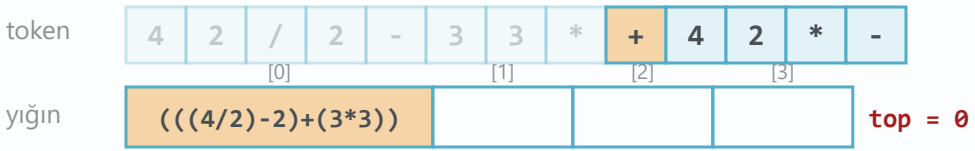


`*` işleç → iki öğe alınır (3, 3), `(3*3)` olarak geri konur.

Adım 8 / 13

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

Postfix → Infix: `42/2−33*+42*−`

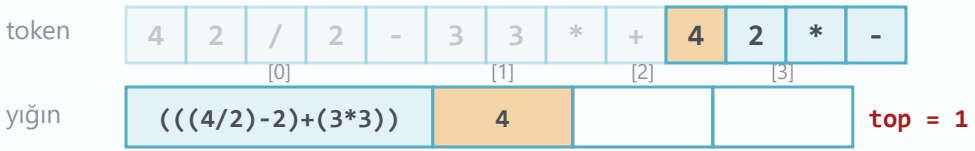


+ işleç → iki öğe alınır (((4/2)-2), (3*3)), `(((4/2)−2)+(3*3))` olarak geri konur.

Adım 9 / 13

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

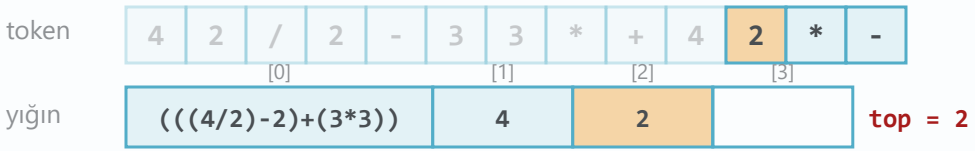
Postfix → Infix: `42/2−33*+42*−`



4 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

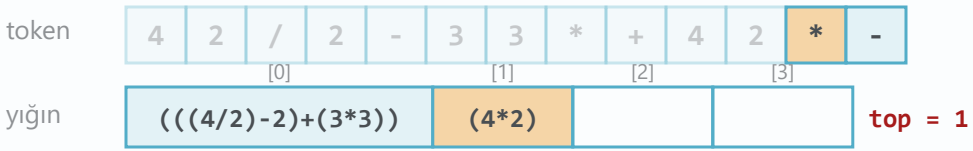
Postfix → Infix: `42/2−33*+42*−`



2 işlenen → yığına.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

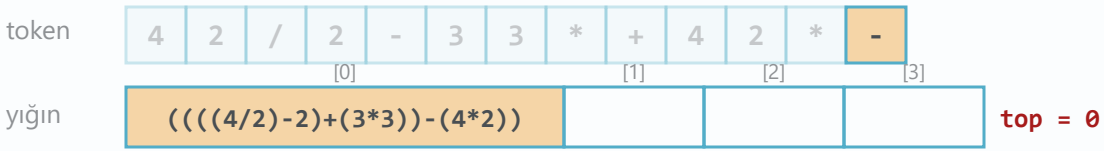
Postfix → Infix: `42/2−33*+42*−`



***** işleç → iki öğe alınır (4, 2), `(4*2)` olarak geri konur.

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

Postfix → Infix: `42/2−33*+42*−`



- işleç → iki öğe alınır ((((4/2)-2)+(3*3)), (4*2)), `((((4/2)−2)+(3*3))−(4*2))` olarak geri konur. Sonuç: `((((4/2)−2)+(3*3))−(4*2))`.

Adım 13 / 13

`*` işlendikten sonra [1] hücresi boş kalmaz; orada `(3*3)` ve `(4*2)` durur (top = 1).

Belirteç: `token_al`

```
typedef enum { sol_paran, sag_paran, arti, eksi,
    carpma, bolme, mod, eos, operand } oncelik;

oncelik token_al(const char *expr, char *sembol, int *n) {
    *sembol = expr[(*n)++];
    switch (*sembol) {
        case '(':  return sol_paran;
        case ')':  return sag_paran;
        case '+':  return arti;
        case '-':  return eksi;
        case '/':  return bolme;
        case '*':  return carpma;
        case '%':  return mod;
        case '\0': return eos;
        default:   return operand;
    }
}
```

- İfade karakter karakter okunur; her karakter bir **belirteç** (token).
- İşlenenler tek basamaklı varsayılır. Rakamı sayıya çevirmek: `sembol - '0'`. '2'nin ASCII kodu 50, '0'ınki 48: 50 − 48 = 2.
- İfade `'\0'` ile biter → `eos`.
- `expr` küresel bir dizi olarak da tutulabilirdi; burada parametre olarak geçirilir.

HOCADAN "typedef enum { … }; ile etiketleme yapmak işimizi kolaylaştırır."

Postfix Hesaplama: `eval` (Küresel Sürüm)

```
float eval(void) {
    char sembol;  oncelik token;
    float op1, op2;
    int n = 0, top = -1;
    token = token_al(expr, &sembol, &n);
    while (token != eos) {
        if (token == operand)
            yigit_ekle(&top, sembol - '0');
        else {
            op2 = yigit_al(&top);   /* önce sağ işlenen */
            op1 = yigit_al(&top);
            yigit_ekle(&top, islem(token, op1, op2));
        }
        token = token_al(expr, &sembol, &n);
    }
    return yigit_al(&top);   /* kalan tek öğe sonuç */
}
```

- İşlenen gelirse yığına konur; işleç gelirse iki işlenen alınır, işlem yapılır, sonuç yığına konur.
- İlk alınan **op2** (sağ işlenen): `6 2 -` için 6 − 2, 2 − 6 değil.
- Sonda yığında tek öğe kalmalı; kalmıyorsa ifade hatalıdır.
- `islem` beş işleci `switch` ile ayırır; `case mod: (int)op1 % (int)op2` : `%` float ile kullanılamaz.

Küresel Değişkenden Parametreye

```
/* kuresel surum: disa bagimli */
char expr[] = "42/2-33*+42*-";
float yigit[MAX_SIZE];
float eval(void);
```

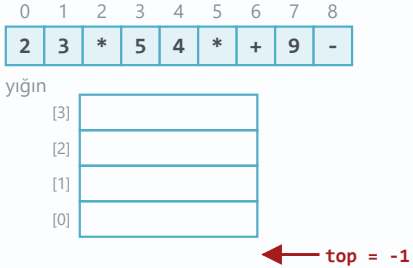
İfade ve yığın küresel: aynı anda iki ifade hesaplanamaz, yordam başka bir programa taşınamaz.

```
/* bileşen: her şey parametre/yerel */
float eval(const char *expr);
/* yığın eval'in içinde yerel dizi */
/* main içinde: */
float a = eval("23*54*+9-");
float b = eval("42/2-33*+42*-");
```

Aynı yordam her yerde, her ifadeyle, yan etkisiz çalışır. Kod izlemelerinde bu sürüm kullanılır.

HOCADAN “Fonksiyonlar component (bileşen) yazar gibi yazılmalı. Global değişken, define gibi kullanımlarla dışa bağımlılık oluşturulmamalı! Böyle component yazılmaz! Yarın bir yerde işe girip ekip hâlinde çalışırken proje yöneticisine şöyle mi denecek: Ben çok has programcıyım. Kendime özgü bir tarzım var. Benim değişkenleri main’in üstüne koy. Ben böyle çalışıyorum. Beni kabul edersen olduğum gibi kabul et, yoksa…”

Kod İzleme: `eval("23*54*+9-")`



İfade **23*54*+9-**; yığın boş.

Adım 1 / 11

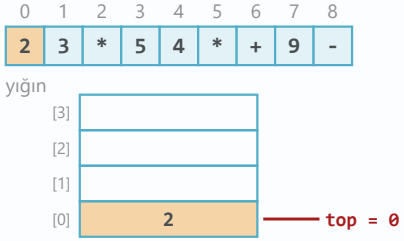
DEĞİŞKENLER

`expr = "23*54*+9-"` `top = -1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'2' işlenen: **'2' − '0' = 2** yığına.

Adım 2 / 11

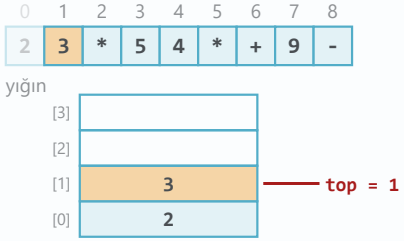
DEĞİŞKENLER

`n = 0` `c = '2'` `top = 0`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'3' işlenen: **'3' − '0' = 3** yığına.

Adım 3 / 11

DEĞİŞKENLER

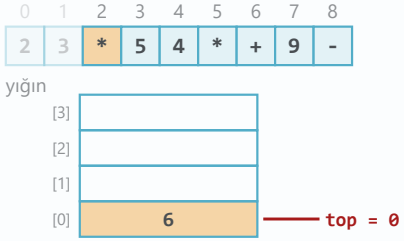
`n = 1` `c = '3'` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: `eval("23*54*+9-")`



'*' işleç: op2 = 3, op1 = 2 → **2 * 3 = 6** yığına.

Adım 4 / 11

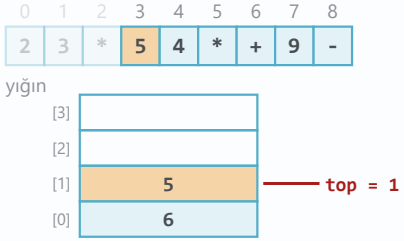
DEĞİŞKENLER

`n = 2` `c = '*'` `op1 = 2` `op2 = 3` `top = 0`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'5' işlenen: **'5' − '0' = 5** yığına.

Adım 5 / 11

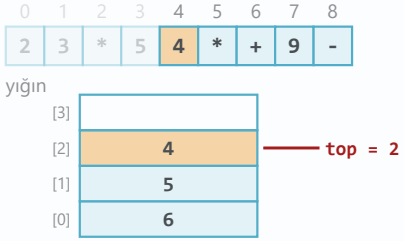
DEĞİŞKENLER

`n = 3` `c = '5'` `top = 1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'4' işlenen: **'4' − '0' = 4** yığına.

Adım 6 / 11

DEĞİŞKENLER

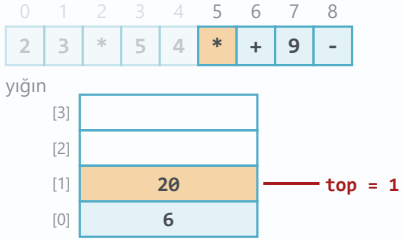
`n = 4` `c = '4'` `top = 2`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: `eval("23*54*+9-")`



'*' işleç: op2 = 4, op1 = 5 → **5 * 4 = 20** yığına.

Adım 7 / 11

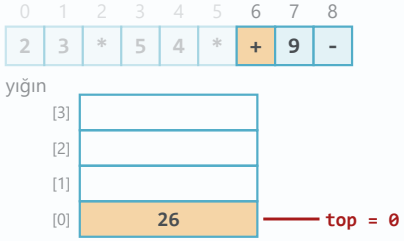
DEĞİŞKENLER

`n = 5` `c = '*'` `op1 = 5` `op2 = 4` `top = 1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'+' işleç: op2 = 20, op1 = 6 → **6 + 20 = 26** yığına.

Adım 8 / 11

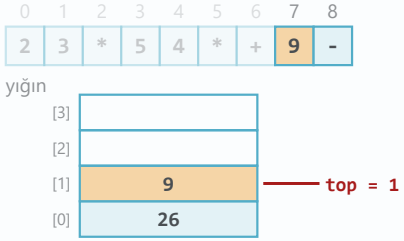
DEĞİŞKENLER

`n = 6` `c = '+'` `op1 = 6` `op2 = 20` `top = 0`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: `eval("23*54*+9-")`



'9' işlenen: **'9' − '0' = 9** yığına.

Adım 9 / 11

DEĞİŞKENLER

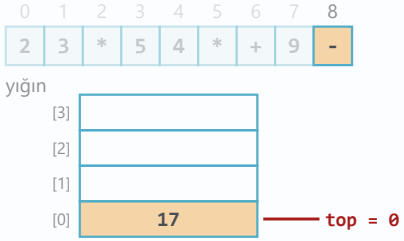
`n = 7` `c = '9'` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: `eval("23*54*+9-")`



'-' işleç: op2 = 9, op1 = 26 → **26 - 9 = 17** yığına.

Adım 10 / 11

DEĞİŞKENLER

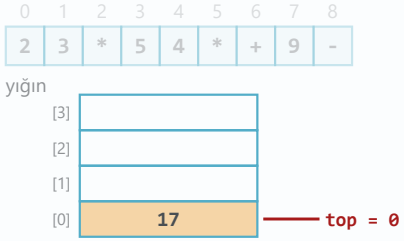
`n = 8` `c = '-'` `op1 = 26` `op2 = 9` `top = 0`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
▶          case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: `eval("23*54*+9-")`



'\0': döngü biter; yığındaki tek öğe **17** döner. Her token bir kez işlendi → **O(n)**.

Adım 11 / 11

DEĞİŞKENLER

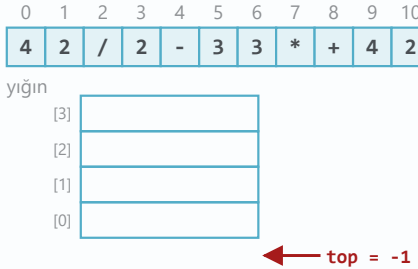
`top = 0` `sonuc = 17`

ÇIKTI

```
17
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
▶    return y[top];
17 }
```

Kod İzleme: Örnek İfade



İfade **42/2-33*+42*-**; yığın boş.

Adım 1 / 15

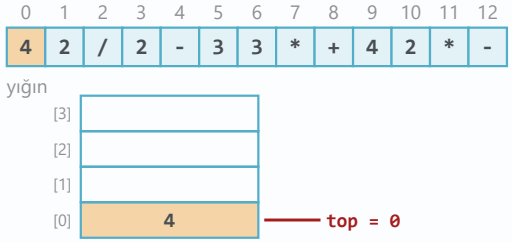
DEĞİŞKENLER

`expr = "42/2-33*+42*-"` `top = -1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'4' işlenen: **'4' − '0' = 4** yığına.

Adım 2 / 15

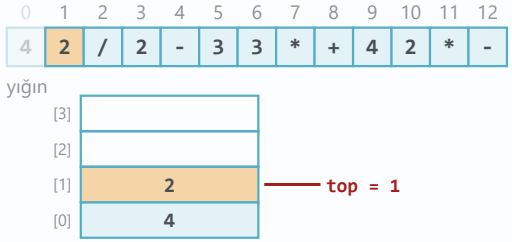
DEĞİŞKENLER

`n = 0` `c = '4'` `top = 0`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'2' işlenen: **'2' − '0' = 2** yığına.

Adım 3 / 15

DEĞİŞKENLER

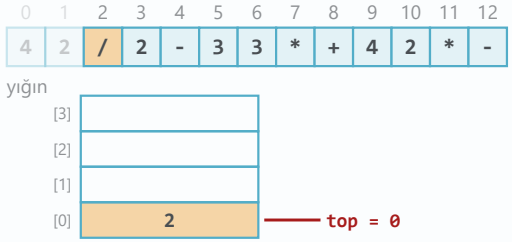
`n = 1` `c = '2'` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'/' işleç: op2 = 2, op1 = 4 → **4 / 2 = 2** yığına.

Adım 4 / 15

DEĞİŞKENLER

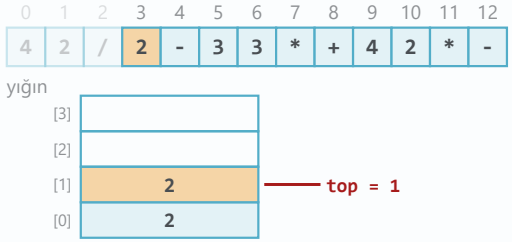
`n = 2` `c = '/'` `op1 = 4` `op2 = 2` `top = 0`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
▶          case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'2' işlenen: **'2' − '0' = 2** yığına.

Adım 5 / 15

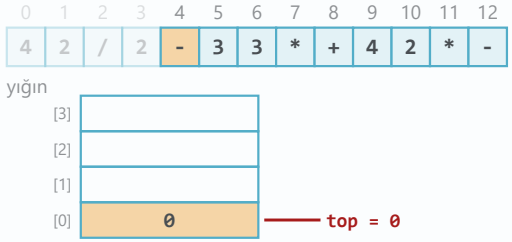
DEĞİŞKENLER

`n = 3` `c = '2'` `top = 1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'-' işleç: op2 = 2, op1 = 2 → **2 - 2 = 0** yığına.

Adım 6 / 15

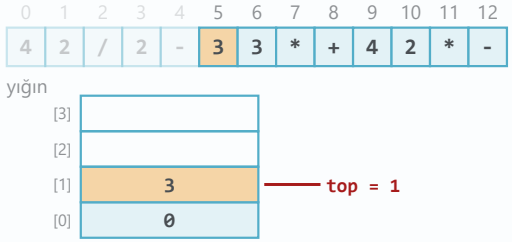
DEĞİŞKENLER

`n = 4` `c = '-'` `op1 = 2` `op2 = 2` `top = 0`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'3' işlenen: **'3' − '0' = 3** yığına.

Adım 7 / 15

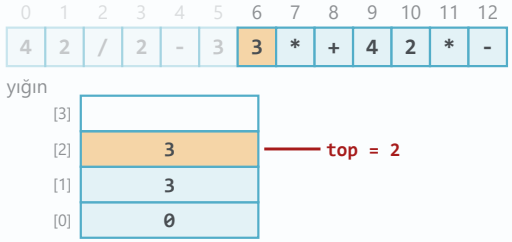
DEĞİŞKENLER

`n = 5` `c = '3'` `top = 1`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'3' işlenen: **'3' − '0' = 3** yığına.

Adım 8 / 15

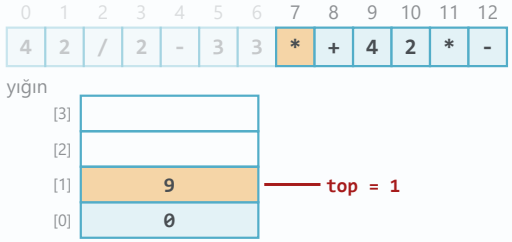
DEĞİŞKENLER

`n = 6` `c = '3'` `top = 2`

ÇIKTI

```
float eval(const char *expr) {
  float y[MAX_SIZE], op1, op2;  int top = -1;
  for (int n = 0; expr[n] != '\0'; n++) {
    char c = expr[n];
    if (isdigit(c)) y[++top] = c - '0';
    else {
      op2 = y[top--];  op1 = y[top--];
      switch (c) {
        case '+': y[++top] = op1 + op2; break;
        case '-': y[++top] = op1 - op2; break;
        case '*': y[++top] = op1 * op2; break;
        case '/': y[++top] = op1 / op2; break;
      }
    }
  }
  return y[top];
}
```

Kod İzleme: Örnek İfade



'*' işleç: op2 = 3, op1 = 3 → **3 * 3 = 9** yığına.

Adım 9 / 15

DEĞİŞKENLER

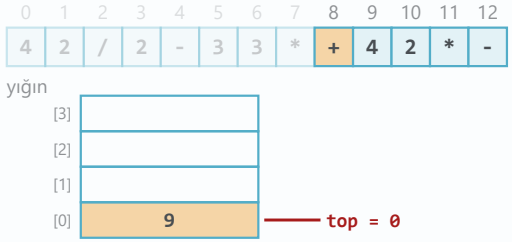
`n = 7` `c = '*'` `op1 = 3` `op2 = 3` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
▶          case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'+' işleç: op2 = 9, op1 = 0 → **0 + 9 = 9** yığına.

Adım 10 / 15

DEĞİŞKENLER

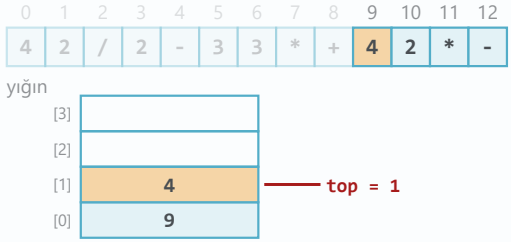
`n = 8` `c = '+'` `op1 = 0` `op2 = 9` `top = 0`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
▶          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'4' işlenen: **'4' − '0' = 4** yığına.

Adım 11 / 15

DEĞİŞKENLER

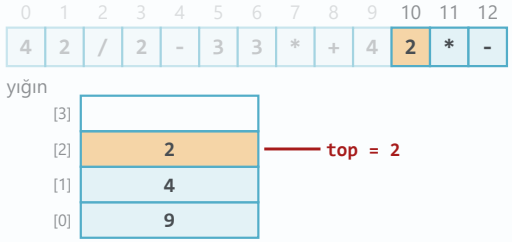
`n = 9` `c = '4'` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'2' işlenen: **'2' − '0' = 2** yığına.

Adım 12 / 15

DEĞİŞKENLER

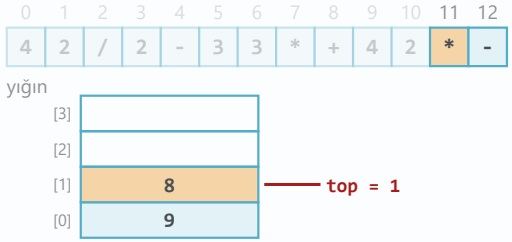
`n = 10` `c = '2'` `top = 2`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
▶      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'*' işleç: op2 = 2, op1 = 4 → **4 * 2 = 8** yığına.

Adım 13 / 15

DEĞİŞKENLER

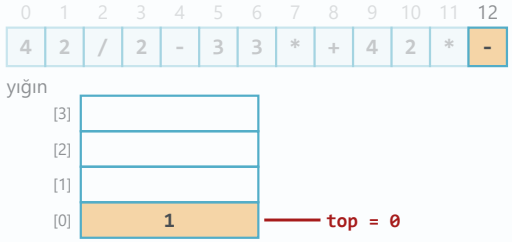
`n = 11` `c = '*'` `op1 = 4` `op2 = 2` `top = 1`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
▶          case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'-' işleç: op2 = 8, op1 = 9 → **9 - 8 = 1** yığına.

Adım 14 / 15

DEĞİŞKENLER

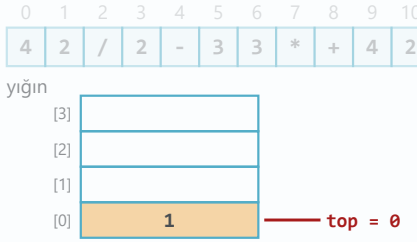
`n = 12` `c = '-'` `op1 = 9` `op2 = 8` `top = 0`

ÇIKTI

```
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
▶        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
▶          case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
16   return y[top];
17 }
```

Kod İzleme: Örnek İfade



'\0': döngü biter; yığındaki tek öğe **1** döner. Her token bir kez işlendi → **O(n)**.

Adım 15 / 15

DEĞİŞKENLER

`top = 0` `sonuc = 1`

ÇIKTI

```
1
```

```
1  float eval(const char *expr) {
2    float y[MAX_SIZE], op1, op2;  int top = -1;
3    for (int n = 0; expr[n] != '\0'; n++) {
4      char c = expr[n];
5      if (isdigit(c)) y[++top] = c - '0';
6      else {
7        op2 = y[top--];  op1 = y[top--];
8        switch (c) {
9          case '+': y[++top] = op1 + op2; break;
10         case '-': y[++top] = op1 - op2; break;
11         case '*': y[++top] = op1 * op2; break;
12         case '/': y[++top] = op1 / op2; break;
13       }
14     }
15   }
▶    return y[top];
17 }
```

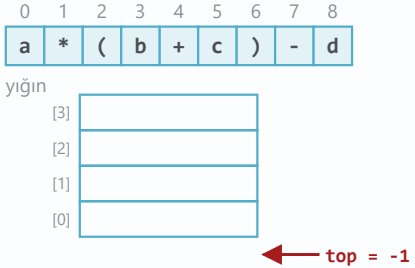
Infix → Postfix: Algoritma

- **İşlenen**: doğrudan çıktıya yazılır.
- **İşleç**: yığındaki işlecin önceliği gelenin önceliğinden **büyük ya da eşit** oldukça o işleç çıktıya alınır; sonra gelen yığına konur (eşitlikte almak = soldan birleşme).
- **(** : her zaman yığına girer; içeride en düşük öncelikli davranır.
- **)**: (görülene kadar yığından çıktıya alınır; (atılır.
- **Sonda**: yığında kalanların hepsi çıktıya alınır.

Token	isp (yığında)	icp (gelen)
`(`	0	20
`* / %`	13	13
`+ –`	12	12
`#` (dip işareti)	–1	—

`(` iki farklı öncelik taşır: gelirken en yüksek (hiçbir şeyi yığından çıkarmadan girer), yığındayken en düşük (üstüne her işleç binebilir). Yığının dibindeki `#` döngüyü durdurur.

Kod İzleme: `a*(b+c)-d` → Postfix



İfade **a*(b+c)-d**. Yığının dibinde **#** (isp = −1) var; şekilde gösterilmiyor.

Adım 1 / 11

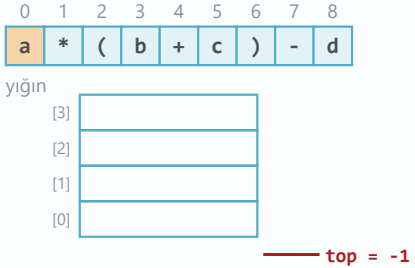
DEĞİŞKENLER

`top = 0` `cikti = ""`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
▶   char y[MAX] = {'#'};  int top = 0, k = 0;
3   for (; *e; e++) {
4     if (isalnum(*e)) cikti[k++] = *e;
5     else if (*e == ')') {
6       while (y[top] != '(')
7         cikti[k++] = y[top--];
8       top--;                /* '(' at */
9     } else {
10      while (isp(y[top]) >= icp(*e))
11        cikti[k++] = y[top--];
12      y[++top] = *e;
13    }
14  }
15  while (top > 0) cikti[k++] = y[top--];
16  cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



a işlenen → çıktıya.

Adım 2 / 11

DEĞİŞKENLER

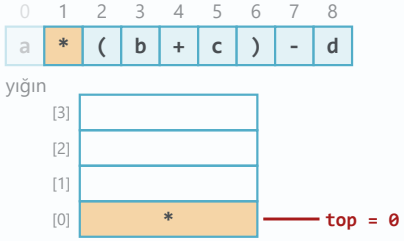
`*e = 'a'` `top = 0` `cikti = "a"`

ÇIKTI

```
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



*: isp(#) = -1 < icp(*) = 13 → * yığına.

Adım 3 / 11

DEĞİŞKENLER

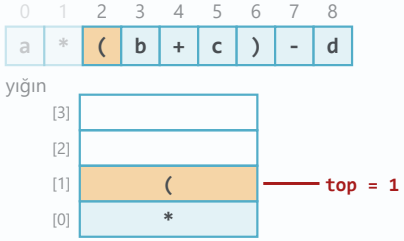
`*e = '*'` `top = 1` `cikti = "a"`

ÇIKTI

```
a
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



(: isp(*) = 13 < icp(() = 20 → **(** yığına.

Adım 4 / 11

DEĞİŞKENLER

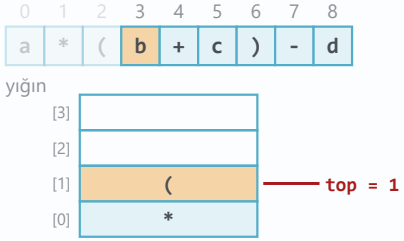
`*e = '('` `top = 2` `cikti = "a"`

ÇIKTI

```
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



b işlenen → çıktıya.

Adım 5 / 11

DEĞİŞKENLER

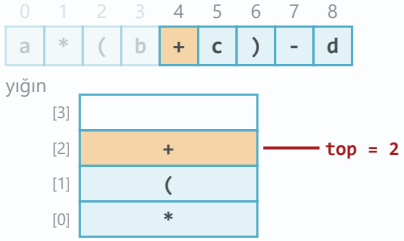
`*e = 'b'` `top = 2` `cikti = "ab"`

ÇIKTI

```
ab
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



+: isp(() = 0 < icp(+) = 12 → + yığına.

Adım 6 / 11

DEĞİŞKENLER

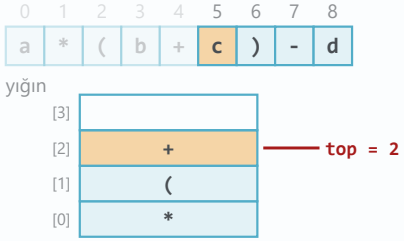
`*e = '+'` `top = 3` `cikti = "ab"`

ÇIKTI

```
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



c işlenen → çıktıya.

Adım 7 / 11

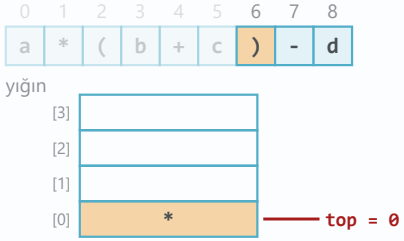
DEĞİŞKENLER

`*e = 'c'` `top = 3` `cikti = "abc"`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



): (görülene kadar al → **+** çıktıya; (atıldı.

Adım 8 / 11

DEĞİŞKENLER

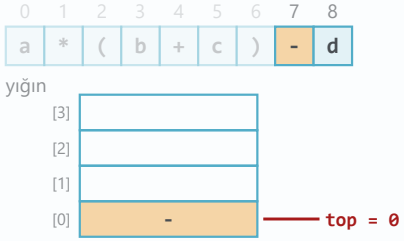
`*e = ')'` `top = 1`

`cikti = "abc+"`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
▶        while (y[top] != '(')
▶          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



-: isp(*) = 13 ≥ icp(-) = 12 → * çıktıya; - yığına.

Adım 9 / 11

DEĞİŞKENLER

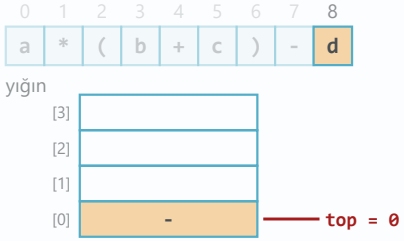
`*e = '-'` `top = 1`

`cikti = "abc+*"`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



d işlenen → çıktıya.

Adım 10 / 11

DEĞİŞKENLER

`*e = 'd'` `top = 1`

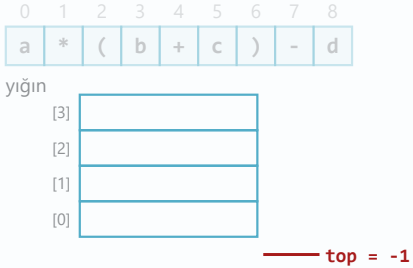
`cikti = "abc+*d"`

ÇIKTI

```
abc+*d
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)-d` → Postfix



Girdi bitti: kalan **-** çıktıya. Sonuç **abc+*d-**. Her token bir kez yığına girip bir kez çıkar → **O(n)**.

Adım 11 / 11

DEĞİŞKENLER

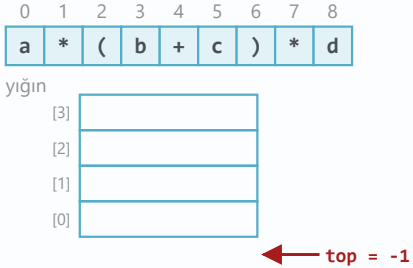
`top = 0` `cikti = "abc+*d-"`

ÇIKTI

```
abc+*d-
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
▶  while (top > 0) cikti[k++] = y[top--];
▶  cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



İfade **a*(b+c)*d**. Yığının dibinde **#** (isp = −1) var; şekilde gösterilmiyor.

Adım 1 / 11

DEĞİŞKENLER

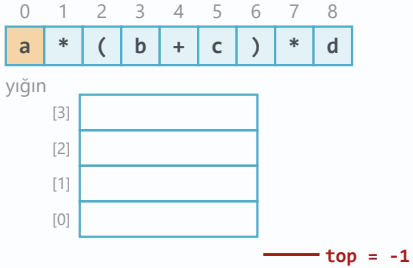
`top = 0` `cikti = ""`

ÇIKTI

```
```

```
1  void postfix(const char *e, char *cikti) {
▶    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



a işlenen → çıktıya.

Adım 2 / 11

DEĞİŞKENLER

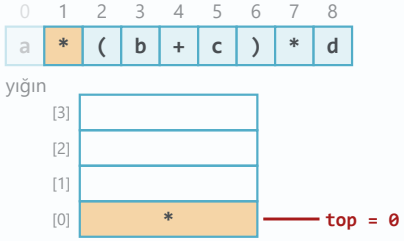
`*e = 'a'` `top = 0` `cikti = "a"`

ÇIKTI

```
a
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



*: isp(#) = -1 < icp(*) = 13 → * yığına.

Adım 3 / 11

DEĞİŞKENLER

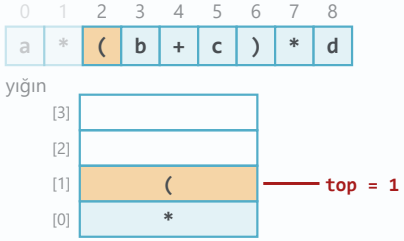
`*e = '*'` `top = 1` `cikti = "a"`

ÇIKTI

```
a
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



(: isp(*) = 13 < icp(() = 20 → **(** yığına.

Adım 4 / 11

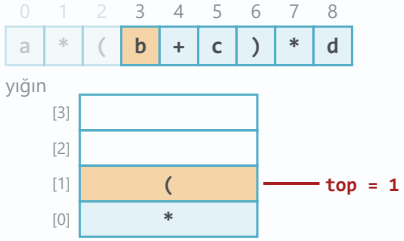
DEĞİŞKENLER

`*e = '('` `top = 2` `cikti = "a"`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



b işlenen → çıktıya.

Adım 5 / 11

DEĞİŞKENLER

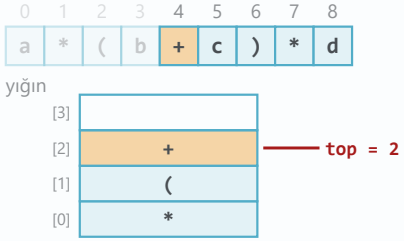
`*e = 'b'` `top = 2` `cikti = "ab"`

ÇIKTI

```
ab
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



+: isp(() = 0 < icp(+) = 12 → + yığına.

Adım 6 / 11

DEĞİŞKENLER

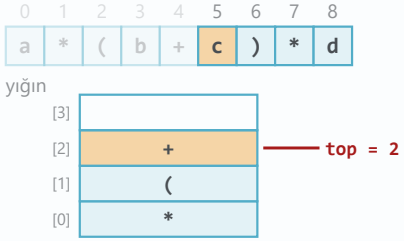
`*e = '+'` `top = 3` `cikti = "ab"`

ÇIKTI

```
ab
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                  /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



c işlenen → çıktıya.

Adım 7 / 11

DEĞİŞKENLER

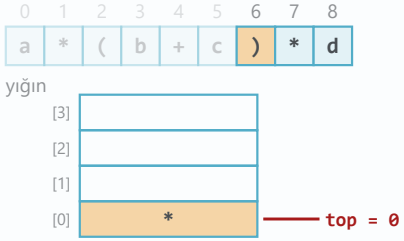
`*e = 'c'` `top = 3` `cikti = "abc"`

ÇIKTI

```
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



): (görülene kadar al → **+** çıktıya; (atıldı.

Adım 8 / 11

DEĞİŞKENLER

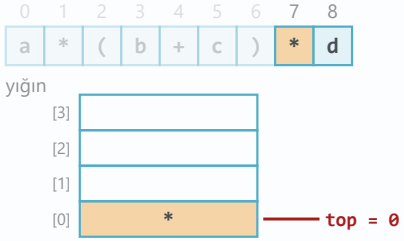
`*e = ')'` `top = 1`

`cikti = "abc+"`

ÇIKTI

```
void postfix(const char *e, char *cikti) {
  char y[MAX] = {'#'};  int top = 0, k = 0;
  for (; *e; e++) {
    if (isalnum(*e)) cikti[k++] = *e;
    else if (*e == ')') {
      while (y[top] != '(')
        cikti[k++] = y[top--];
      top--;                /* '(' at */
    } else {
      while (isp(y[top]) >= icp(*e))
        cikti[k++] = y[top--];
      y[++top] = *e;
    }
  }
  while (top > 0) cikti[k++] = y[top--];
  cikti[k] = '\0';
}
```

Kod İzleme: `a*(b+c)*d` Örneği



*: isp(*) = 13 ≥ icp(*) = 13 → * çıktıya; * yığına.

Adım 9 / 11

DEĞİŞKENLER

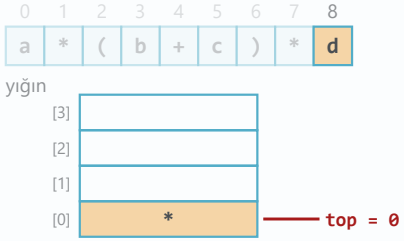
`*e = '*'` `top = 1`

`cikti = "abc+*"`

ÇIKTI

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
▶        while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
▶        y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



d işlenen → çıktıya.

Adım 10 / 11

DEĞİŞKENLER

`*e = 'd'` `top = 1`

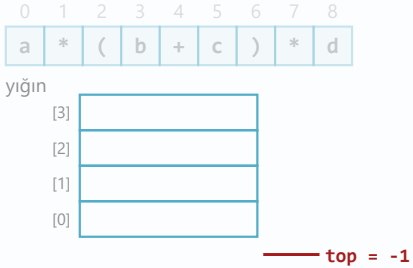
`cikti = "abc+*d"`

ÇIKTI

```
abc+*d
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
▶      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
15   while (top > 0) cikti[k++] = y[top--];
16   cikti[k] = '\0';
17 }
```

Kod İzleme: `a*(b+c)*d` Örneği



Girdi bitti: kalan ***** çıktıya. Sonuç **abc+*d***. Her token bir kez yığına girip bir kez çıkar → **O(n)**.

Adım 11 / 11

DEĞİŞKENLER

`top = 0` `cikti = "abc+*d*"`

ÇIKTI

```
abc+*d*
```

```
1  void postfix(const char *e, char *cikti) {
2    char y[MAX] = {'#'};  int top = 0, k = 0;
3    for (; *e; e++) {
4      if (isalnum(*e)) cikti[k++] = *e;
5      else if (*e == ')') {
6        while (y[top] != '(')
7          cikti[k++] = y[top--];
8        top--;                /* '(' at */
9      } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
▶  while (top > 0) cikti[k++] = y[top--];
▶  cikti[k] = '\0';
17 }
```

İzleme Tablosu: `a+b*c` → `abc*+`

token	[0]	[1]	top	Çıktı
a			−1	a
+	+		0	a
b	+		0	ab
*	+	*	1	ab
c	+	*	1	abc
eos			−1	abc*+

- `*` gelince üstte `+` var: isp(+) = 12 < icp(*) = 13 → `*` **üstüne** biner.
- Sonda önce `*`, sonra `+` çıkar: daha öncelikli olan önce uygulanır.

Sık sorulan soru: “Son işlenen basılmıyor mu?” `postfix` yordamı işlenenleri anında çıktıya yazdığı ve sonda yığını boşalttığı için son işlenen de basılır.

INFİX → POSTFİX

Infix → Prefix: Ters Çevirme, Postfix, Ters Çevirme

```
(a+b)*c-d/a
```

Infix → prefix için bir yol: ifade **ters çevrilir**.

Adım 1 / 5

Doğrudan yol: ifade **sağdan sola** taranır; işlenenler çıktının başına eklenir, işleçler aynı yığın kuralıyla yönetilir. İki yol da O(n).

Infix → Prefix: Ters Çevirme, Postfix, Ters Çevirme

```
a/d-c*)b+a(
```

Ters çevrildi. Şimdi **(ile)** yer değiştirir.

Doğrudan yol: ifade **sağdan sola** taranır; işlenenler çıktının başına eklenir, işleçler aynı yığın kuralıyla yönetilir. İki yol da O(n).

Infix → Prefix: Ters Çevirme, Postfix, Ters Çevirme

```
a/d-c*(b+a)
```

Parantezler düzeltildi. Bu ifade postfix'e çevrilir; ancak eşit öncelikte yığından **alınmaz** (> kullanılır), çünkü ters çevrilmiş ifadede birleşme yönü de ters döner.

Doğrudan yol: ifade **sağdan sola** taranır; işlenenler çıktının başına eklenir, işleçler aynı yığın kuralıyla yönetilir. İki yol da O(n).

Infix → Prefix: Ters Çevirme, Postfix, Ters Çevirme

```
ad/cba+*-
```

Postfix sonucu.

Doğrudan yol: ifade **sağdan sola** taranır; işlenenler çıktının başına eklenir, işleçler aynı yığın kuralıyla yönetilir. İki yol da O(n).

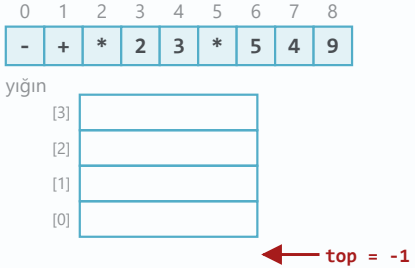
Infix → Prefix: Ters Çevirme, Postfix, Ters Çevirme

```
-*+abc/da
```

Tekrar **ters çevrilir**: prefix. Tablodaki 3. satırla aynı.

Doğrudan yol: ifade **sağdan sola** taranır; işlenenler çıktının başına eklenir, işleçler aynı yığın kuralıyla yönetilir. İki yol da O(n).

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



Prefix **-+*23*549** (infix: 2*3+5*4−9). **Sağdan sola** taranır.

Adım 1 / 11

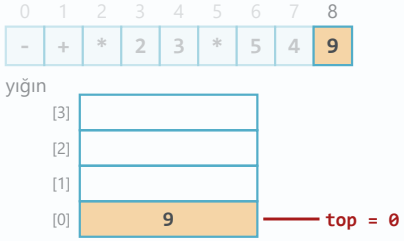
DEĞİŞKENLER

`i = 8` `top = -1`

ÇIKTI

```
float prefix_eval(const char *e) {
  float y[MAX], op1, op2;  int top = -1;
  for (int i = strlen(e) - 1; i >= 0; i--) {
    if (isdigit(e[i])) y[++top] = e[i] - '0';
    else {
      op1 = y[top--];  op2 = y[top--];
      y[++top] = uygula(e[i], op1, op2);
    }
  }
  return y[top];
}
```

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'9' işlenen → yığına.

Adım 2 / 11

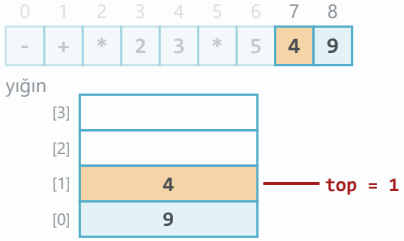
```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
▶      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

DEĞİŞKENLER

`i = 8` `e[i] = '9'` `top = 0`

ÇIKTI

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'4' işlenen → yığına.

Adım 3 / 11

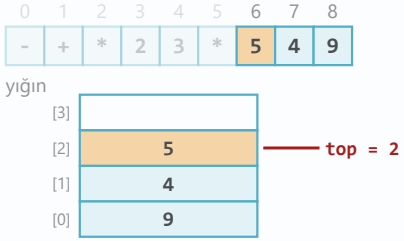
```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
▶      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

DEĞİŞKENLER

`i = 7` `e[i] = '4'` `top = 1`

ÇIKTI

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'5' işlenen → yığına.

Adım 4 / 11

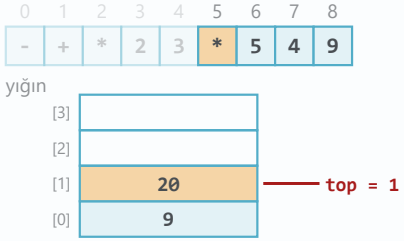
```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
▶      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

DEĞİŞKENLER

`i = 6` `e[i] = '5'` `top = 2`

ÇIKTI

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'*': ilk alınan **op1** = 5 (sol işlenen!), op2 = 4 → **5 * 4 = 20**.

Adım 5 / 11

DEĞİŞKENLER

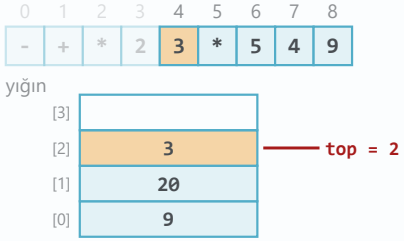
`i = 5` `e[i] = '*'` `op1 = 5` `op2 = 4` `top = 1`

ÇIKTI

```
```

```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
4      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
▶        op1 = y[top--];  op2 = y[top--];
▶        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'3' işlenen → yığına.

Adım 6 / 11

DEĞİŞKENLER

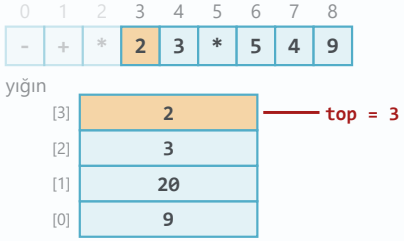
`i = 4` `e[i] = '3'` `top = 2`

ÇIKTI

```
```

```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
▶      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'2' işlenen → yığına.

Adım 7 / 11

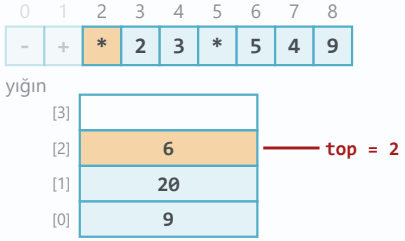
```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
▶      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

DEĞİŞKENLER

`i = 3` `e[i] = '2'` `top = 3`

ÇIKTI

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'*': ilk alınan **op1** = 2 (sol işlenen!), op2 = 3 → **2 * 3 = 6**.

Adım 8 / 11

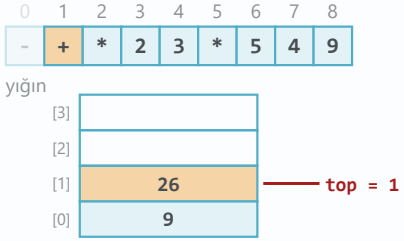
DEĞİŞKENLER

`i = 2` `e[i] = '*'` `op1 = 2` `op2 = 3` `top = 2`

ÇIKTI

```
float prefix_eval(const char *e) {
  float y[MAX], op1, op2;  int top = -1;
  for (int i = strlen(e) - 1; i >= 0; i--) {
    if (isdigit(e[i])) y[++top] = e[i] - '0';
    else {
      op1 = y[top--];  op2 = y[top--];
      y[++top] = uygula(e[i], op1, op2);
    }
  }
  return y[top];
}
```

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'+': ilk alınan **op1** = 6 (sol işlenen!), op2 = 20 → **6 + 20 = 26**.

```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
4      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
▶        op1 = y[top--];  op2 = y[top--];
▶        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

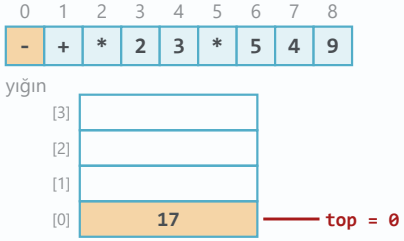
Adım 9 / 11

DEĞİŞKENLER

`i = 1` `e[i] = '+'` `op1 = 6` `op2 = 20` `top = 1`

ÇIKTI

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



'-': ilk alınan **op1** = 26 (sol işlenen!), op2 = 9 → **26 - 9 = 17**.

Adım 10 / 11

DEĞİŞKENLER

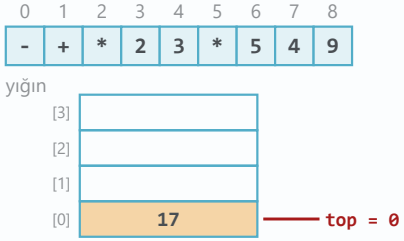
`i = 0` `e[i] = '-'` `op1 = 26` `op2 = 9` `top = 0`

ÇIKTI

```
```

```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
4      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
▶        op1 = y[top--];  op2 = y[top--];
▶        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
10   return y[top];
11 }
```

Kod İzleme: Prefix Hesaplama (`-+*23*549`)



Sonuç **17**: postfix örneğiyle (23*54*+9−) aynı ifade. Postfix'te ilk alınan sağ işlenendi; prefix'te sol işlenen.

Adım 11 / 11

DEĞİŞKENLER

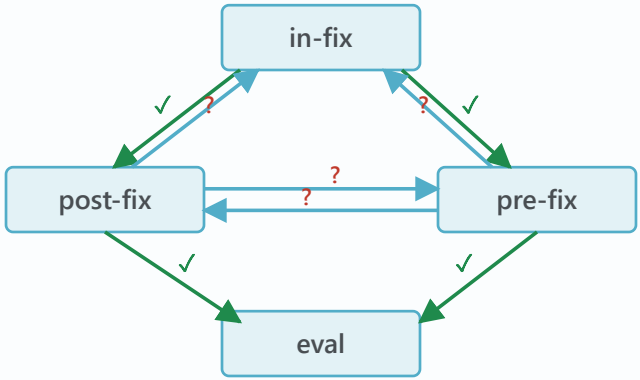
`top = 0` `sonuc = 17`

ÇIKTI

```
17
```

```
1  float prefix_eval(const char *e) {
2    float y[MAX], op1, op2;  int top = -1;
3    for (int i = strlen(e) - 1; i >= 0; i--) {
4      if (isdigit(e[i])) y[++top] = e[i] - '0';
5      else {
6        op1 = y[top--];  op2 = y[top--];
7        y[++top] = uygula(e[i], op1, op2);
8      }
9    }
▶    return y[top];
11 }
```

Çalışma: Diğer Çeviriler

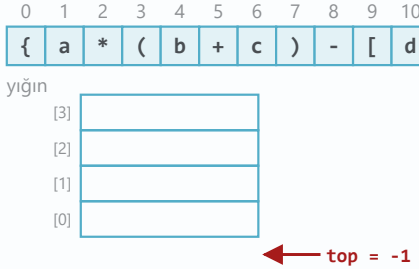


✓: bu bölümde yapıldı · ?: tasarlanacak çeviriler.

ÇALIŞMA: İfade çözümleme ile ilgili diğer çevirme ve hesaplama algoritmalarını tasarlayınız.

- postfix → infix (bu bölümde elle yapıldı; C ile yazın)
- postfix → prefix ve prefix → postfix
- infix → prefix (doğrudan, sağdan sola tarayarak)
- prefix → infix
- infix'i doğrudan hesaplama (iki yığın: işlenen ve işleç)

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



Girdi **{a*(b+c)-[d]}**.

Adım 1 / 15

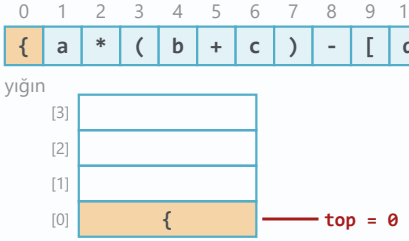
DEĞİŞKENLER

`top = -1`

ÇIKTI

```bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



{ açan → yığına.

Adım 2 / 15

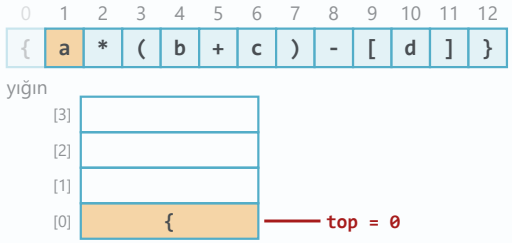
DEĞİŞKENLER

`*e = '{'` `top = 0`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'a' parantez değil, atlanır.

Adım 3 / 15

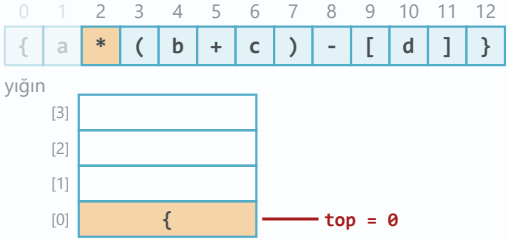
DEĞİŞKENLER

`*e = 'a'` `top = 0`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'*' parantez değil, atlanır.

Adım 4 / 15

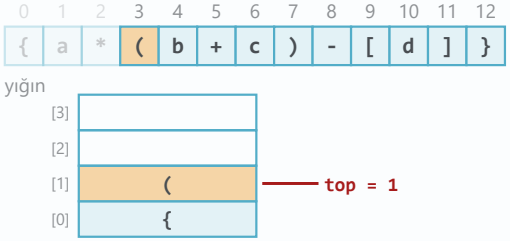
DEĞİŞKENLER

`*e = '*'` `top = 0`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



(açan → yığına.

Adım 5 / 15

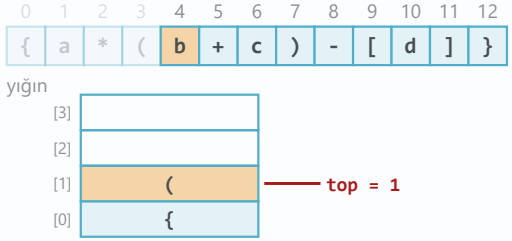
DEĞİŞKENLER

`*e = '('` `top = 1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'b' parantez değil, atlanır.

Adım 6 / 15

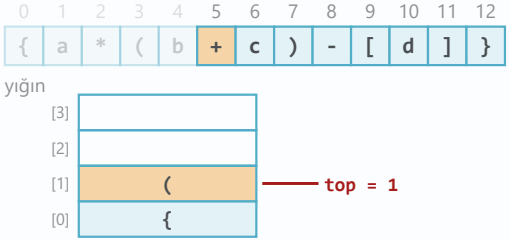
DEĞİŞKENLER

`*e = 'b'` `top = 1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'+' parantez değil, atlanır.

Adım 7 / 15

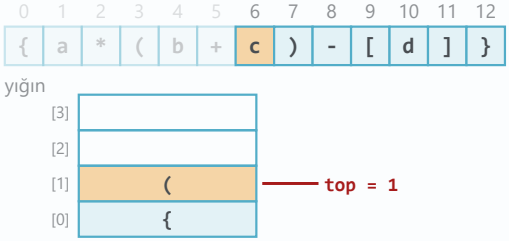
DEĞİŞKENLER

`*e = '+'` `top = 1`

ÇIKTI

```
1  bool dengeli(const char *e) {
2    char y[MAX];  int top = -1;
▶    for (; *e; e++) {
4      if (*e=='(' || *e=='[' || *e=='{')
5        y[++top] = *e;
6      else if (*e==')' || *e==']' || *e=='}') {
7        if (top < 0 || !eslesir(y[top], *e))
8          return false;  /* fazla/yanlış */
9        top--;
10     }
11   }
12   return top == -1;        /* açık kalmadı mı */
13 }
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'c' parantez değil, atlanır.

Adım 8 / 15

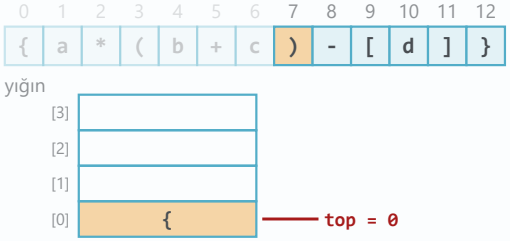
DEĞİŞKENLER

`*e = 'c'` `top = 1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



) kapanan; üstteki (ile eşleşti → pop.

Adım 9 / 15

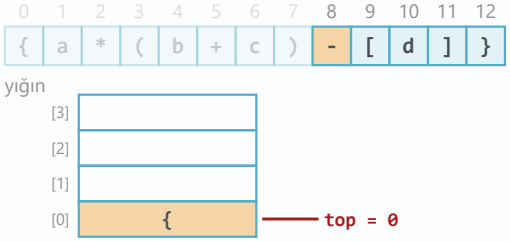
DEĞİŞKENLER

`*e = ')'` `top = 0`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'-' parantez değil, atlanır.

Adım 10 / 15

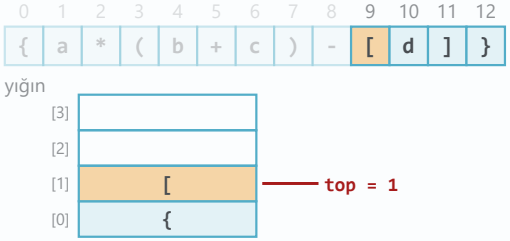
DEĞİŞKENLER

`*e = '-'` `top = 0`

ÇIKTI

```
1  bool dengeli(const char *e) {
2    char y[MAX];  int top = -1;
▶    for (; *e; e++) {
4      if (*e=='(' || *e=='[' || *e=='{')
5        y[++top] = *e;
6      else if (*e==')' || *e==']' || *e=='}') {
7        if (top < 0 || !eslesir(y[top], *e))
8          return false;  /* fazla/yanlış */
9        top--;
10     }
11   }
12   return top == -1;        /* açık kalmadı mı */
13 }
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



[açan → yığına.

Adım 11 / 15

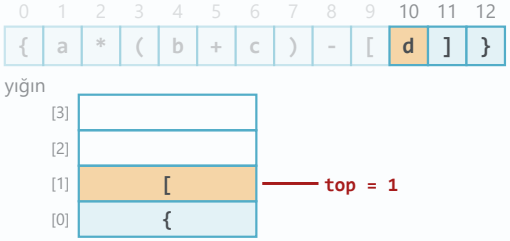
DEĞİŞKENLER

`*e = '['` `top = 1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



'd' parantez değil, atlanır.

Adım 12 / 15

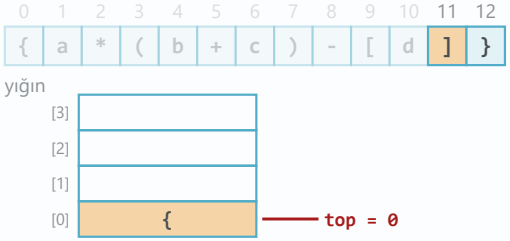
DEĞİŞKENLER

`*e = 'd'` `top = 1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



] kapanan; üstteki [ile eşleşti → pop.

Adım 13 / 15

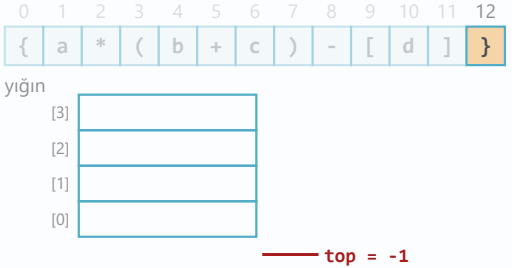
DEĞİŞKENLER

`*e = ']'` `top = 0`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



} kapanan; üstteki { ile eşleşti → pop.

Adım 14 / 15

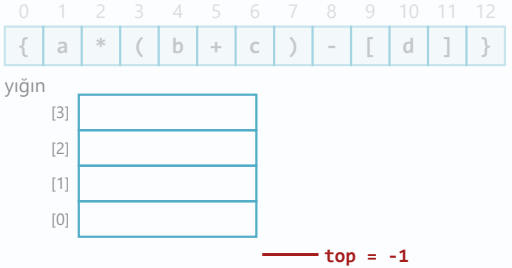
DEĞİŞKENLER

`*e = '}'` `top = -1`

ÇIKTI

```
bool dengeli(const char *e) {
  char y[MAX];  int top = -1;
  for (; *e; e++) {
    if (*e=='(' || *e=='[' || *e=='{')
      y[++top] = *e;
    else if (*e==')' || *e==']' || *e=='}') {
      if (top < 0 || !eslesir(y[top], *e))
        return false;  /* fazla/yanlış */
      top--;
    }
  }
  return top == -1;       /* açık kalmadı mı */
}
```

Kod İzleme: Dengeli Parantez `{a*(b+c)-[d]}`



Girdi bitti ve yığın **boş** → **true**: dengeli.

Adım 15 / 15

DEĞİŞKENLER

`top = -1`

ÇIKTI

```
true
```

```
1  bool dengeli(const char *e) {
2    char y[MAX];  int top = -1;
3    for (; *e; e++) {
4      if (*e=='(' || *e=='[' || *e=='{')
5        y[++top] = *e;
6      else if (*e==')' || *e==']' || *e=='}') {
7        if (top < 0 || !eslesir(y[top], *e))
8          return false;  /* fazla/yanlış */
9        top--;
10     }
11   }
▶  return top == -1;       /* açık kalmadı mı */
13 }
```

DİĞER UYGULAMALAR

Parantez Denetimi: Üç Hata Türü

Girdi	Ne olur?	Hata
`[(a+b]*c)`	`]` gelince üstte `(` var	Yanlış eşleşme
`(a+b))`	İkinci `)` geldiğinde yığın boş	Fazla kapanan
`((a+b)`	Girdi bitince yığında `(` kaldı	Kapanmamış açan

- Yalnız sayaç tutmak tek tür parantez için yeterlidir; **birden çok türde** "en son açılan" bilgisi gerekir → yığın.
- Derleyiciler, HTML/XML denetleyicileri ve metin düzenleyicilerin eşleşen parantez vurgusu bu algoritmayı kullanır.
- Zaman O(n), bellek en kötü O(n) (hepsi açan).

DİĞER UYGULAMALAR

Yığın ve Kuyruğun Başka Uygulamaları

Geri al / yinele

İki yığın: yapılan işlem “geri al” yığınına, geri alınan “yinele” yığınına gider; tarayıcının geri/ileri düğmeleri de böyle çalışır.

Özyinelemesiz gezinme

Kök ortada gezinme, düğüm adreslerini tutan açık bir yığınla döngüye çevrilir: O(n) zaman, O(h) yığın.

Sonraki büyük öğe

Tek geçişte her öğenin sağındaki ilk büyük öğe: azalan yığın, O(n).

Kayan pencere maksimumu

Deque ile her pencerenin en büyüğü O(n)'de bulunur.

BFS ile en kısa yol

Labirentte kuyruk kullanılırsa ilk bulunan yol en kısa yoldur.

Hanoi, n-vezir, sudoku

Özyinelemeli geri izleme; açık yığınla da yazılabilir.

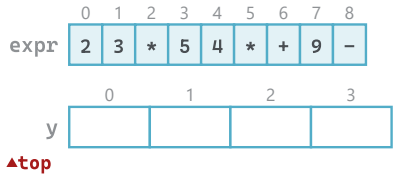
Laboratuvar: Postfix Hesaplama

23*54*+9-

Girdi (postfix ifade, tek basamaklı sayılar) 23*54*+9− Uygula

KOD

```
 1 float eval(const char *expr) {
 2   float y[MAX_SIZE], op1, op2; int top = -1;
 3   for (int n = 0; expr[n] != '\0'; n++) {
 4     char c = expr[n];
 5     if (isdigit(c)) y[++top] = c - '0';
 6     else {
 7       op2 = y[top--];  op1 = y[top--];
 8       switch (c) {
 9       case '+': y[++top] = op1 + op2; break;
10       case '-': y[++top] = op1 - op2; break;
11       case '*': y[++top] = op1 * op2; break;
12       case '/': y[++top] = op1 / op2; break;
13       }
14     }
```



n − c − top −1

ÇIKTI (henüz yok)

`eval("23*54*+9−")` : ifade soldan sağa okunur. İşlenen yığına itilir; işleç gelince üstteki iki değer alınır, sonuç geri itilir.

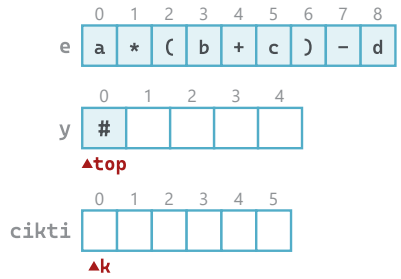
Laboratuvar: Infix → Postfix

a*(b+c)-d

Girdi (infix ifade, ör. a*(b+c)-d) a*(b+c)-d Uygula

KOD

```
 1 void postfix(const char *e, char *cikti) {
 2   char y[MAX] = {'#'};  int top = 0, k = 0;
 3   for (; *e; e++) {
 4     if (isalnum(*e)) cikti[k++] = *e;
 5     else if (*e == ')') {
 6       while (y[top] != '('
 7         cikti[k++] = y[top--];
 8       top--;                /* '(' at */
 9     } else {
10       while (isp(y[top]) >= icp(*e))
11         cikti[k++] = y[top--];
12       y[++top] = *e;
13     }
14   }
```



*e '\0' top 0 k 0

ÇIKTI (henüz yok)

`postfix("a*(b+c)-d", cikti)` : yığının dibinde `'#'` var (isp = −1); işlenenler doğrudan çıktıya yazılır, işleçler önceliklerine göre yığında bekletilir.

ÖZET

Karmaşıklık Özeti

Algoritma	Yapı	Zaman	Ek bellek
Labirent (izbul)	yığın	O(m·p)	O(m·p)
Labirent en kısa yol	kuyruk	O(m·p)	O(m·p)
Postfix / prefix hesaplama	yığın	O(n)	O(n)
Infix → postfix / prefix	yığın	O(n)	O(n)
Parantez denetimi	yığın	O(n)	O(n)
Özyinelemesiz kök ortada	yığın	O(n)	O(h)
Düzey sırası gezinme	kuyruk	O(n)	O(n)

n: ifade uzunluğu ya da düğüm sayısı · m×p: labirent boyutu · h: ağaç yüksekliği.

ÖZET

Sık Yapılan Hatalar

1. Postfix'te `op1` ile `op2`'yi ters almak: `6 2 -` −4 olur.

2. Prefix'te postfix sırasını kullanmak: orada ilk alınan **sol** işlenendir.

3. Infix → postfix'te eşit öncelikte yığından almamak: `a-b-c` yanlış çevrilir.

4. `)` gelince `(`'ı yığında unutmak ya da çıktıya yazmak.

5. Labirentte `gezildi` işaretini unutup sonsuz döngüye girmek.

6. Çevreyi 1 ile çevirmeden dizi dışına taşmak.

ÖZET

Alıştırmalar

1. `a/(b-c)+d*(e-a)*c` ifadesini postfix ve prefix'e çevir.
2. `6 2 3 + - 3 8 2 / + *` postfix ifadesini hesapla.
3. Postfix → infix çeviren C yordamını yaz.
4. `izbul` bulunan yolu labirentte işaretlesin.
5. Labirentte en kısa yolu kuyrukla bul.
6. Özyinelemesiz kök önde ve kök sonda gezinmeyi yaz.

▶ **İpuçları / cevaplar**

ÖZET

Bölüm Özeti

Geri izleme

Labirentte her yol ayrımı yığına; çıkmazda pop ile son ayrıma dönülür.

En kötü durum

Uzun ve yanlış ilk yol: tüm yanlış bölge gezilir.

Gösterimler

Infix öncelik ister; postfix/prefix parantezsiz ve önceliksiz hesaplanır.

Postfix hesaplama

İşlenen itilir; işleçte op2, op1 alınır; sonuç itilir. O(n).

Infix → postfix

isp ≥ icp oldukça alınır; (gelirken yüksek, yığında düşük öncelik.

Genel ilke

"En son açılan ilk kapanır" görülen her yerde yığın; "sırayla" görülen yerde kuyruk.

HOCADAN "Ödev kâğıdını şöyle benim kâğıtla üst üste getirip ışığa tuttuğum zaman... (fark görmememem lazım)"

ÖZET

Kaynaklar ve Devam

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2. bs., Bölüm 3.5 (labirent), 3.6 (ifade değerlendirme).
- R. Sedgewick, *Algorithms in C*, Kısım 1–4, Bölüm 4.3 (postfix), 5 (özyineleme ve yığın).

← Bölüm 7: Kuyruklar

Ders sayfası

Bölüm 9: Liste ADT ve Genelleştirilmiş Listeler

Son

Bölüm 8: Yığın ve Kuyruk Uygulamaları

Sonraki bölüm: Bölüm 9: Liste ADT ve Genelleştirilmiş Listeler · Ders sayfası