Bölüm 12 – Data (Veri) Yapıları

İçerik

12.1 Giriş

Dinamik Data Yapıları

- İşlem sırasında veri yapıları büyür veya küçülür

Erişim Listeleri

- Herhangi bir yere ekleme ve kaldırmaya izin verir

Yığıtlar

- Sadece yığıtın başında ekleme ve kaldırmaya izin verir

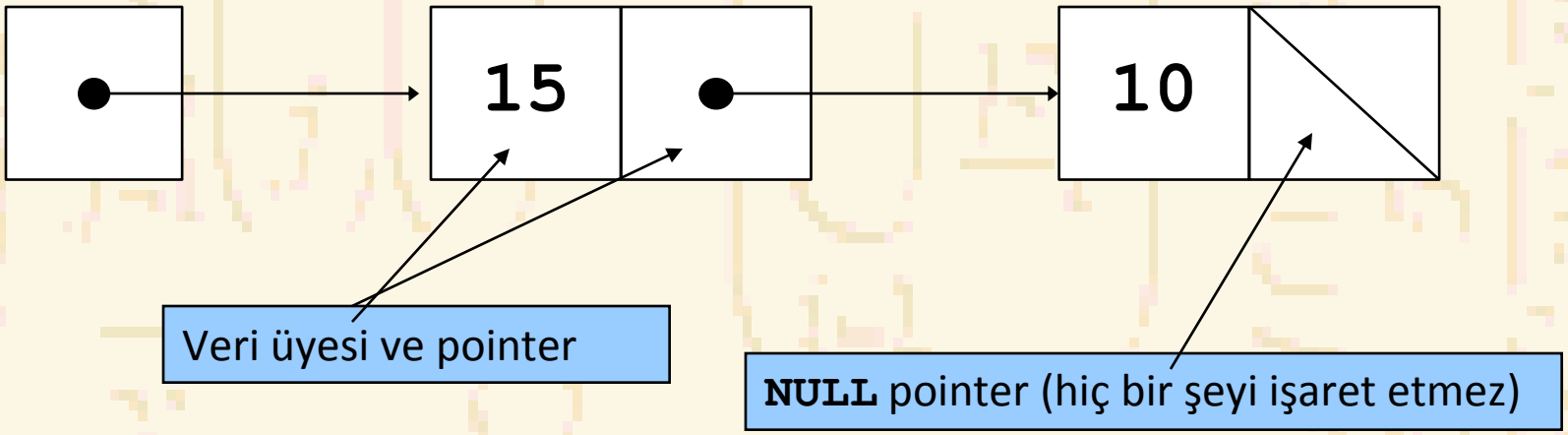
Kuyruklar

- Sona eklemeye ve baştan kaldırmaya izin verir

İkili Ağaçlar

- Verinin yüksek hızda taranma ve sıralanması, tekrarlanmış verilerin etkili eliminasyonu

- Kendine-referans yapıları
  - Aynı tip yapıya pointer içeren yapı
  - Listeler yığınlar kuyruklar ve ağaçlar gibi kullanışlı veri yapıları oluşturmak için birlikte bağlanılabilir
  - Bir **`NULL`** pointer (**`0`**) ile sonlandırılır
- Birlikte bağlanmış iki kendine-referans yapılı nesne diagramı



```
struct nod {
    int veri;
    struct nod *sonrakiPtr;
}
```

- **sonrakiPtr**
  - Nod tipinde bir nesneyi point eder
  - Bir bağlantı olarak refere edilir
    - Bir **nod** u bir başka **nod** a bağlar

- Dinamik Bellek Düzenleme
  - Çalışma sırasında bellek alır veya serbest bırakır
- **`malloc`**
  - Düzenlenecek byte sayısı
    - Nesnenin boyutunu belirlemek için **`sizeof`** kullan
  - **`void *`** tipinde pointer gönderir
    - Bir **`void *`** pointer-ı her hangi bir pointer-a atanabilir
    - Bellekte yer yoksa, **`NULL`** gönderir
  - Örnek

```
yeniPtr = malloc( sizeof( struct nod ) );
```

- **`free`**
  - **`malloc`** ile düzenlenmiş belleği eski haline getirir
  - Bir pointer-ı argüment alır
  - **`free ( yeniPtr );`**

12.4 Erişim Listeleri

- Erişim listesi
  - Nodlar olarak adlandırılan kendine-referans sınıfı nesnelerin lineer birleşimi
  - Pointer erişimleri ile bağlanılır
  - Listenin ilk noduna bir pointer ile erişim sağlanır
  - Diğer nodlara o anki nodun bağlantı pointer-ı aracılığı ile erişim sağlanır
  - Son noddaki bağlantı pointer-ı liste sonunu işaretlemek için null alınır
- Bir dizi yerine erişim listesi kullanınız; eğer
  - Tahmin edilemeyen sayıda veri elemanına sahipseniz
  - Listenin hızlı bir şekilde sıralanması gerekiyorsa

- Erişim listelerinin tipleri:
  - Tekli erişim listesi
    - İlk noda bir pointer ile başlar
    - Bir null pointer ile sona erer
    - Sadece tek yönde hareket eder
  - Dairesel, tekli erişim
    - Son noddaki pointer tekrar ilk nodu işaret eder
  - Çiftli erişim listesi
    - İki "başlama pointer-ı" – ilk eleman ve son eleman
    - Herbir node bir ileri pointer-a ve bir geri pointer-a sahiptir
    - Hem ileri hem geri harekete izin verir
  - Dairesel, çiftli erişim listesi
    - Son nodun ileri pointer-ı ilk nodu ve ilk nodun geri pointer-ı son nodu işaret eder

```c
/* Fig. 12.3: fig12_03.c
   Bir liste üzerinde işlem ve bakım */
#include <stdio.h>
#include <stdlib.h>

struct listeNodu {   /* kendine referans yapısı */
   char veri;
   struct listeNodu *sonrakiPtr;
};

typedef struct listeNodu ListeNodu;
typedef ListeNodu *ListeNoduPtr;

void ekle( ListeNoduPtr *, char );
char sil( ListeNoduPtr *, char );
int bos( ListeNoduPtr );
void yazListe( ListeNoduPtr );
void acıklama( void );

int main()
{
   ListeNoduPtr baslaPtr = NULL;
   int secim;
   char parca;

   acıklama();  /* menüyü göster */
   printf( "? " );
   scanf( "%d", &secim );
```

```
29
30    while ( secim != 3 ) {
31
32       switch ( secim ) {
33          case 1:
34             printf( "Bir karakter gir: " );
35             scanf( "\n%c", &parca );
36             ekle( &baslaPtr, parca );
37             yazListe( baslaPtr );
38             break;
39          case 2:
40             if ( !bos( baslaPtr ) ) {
41                printf( "Silinecek karakteri gir: " );
42                scanf( "\n%c", &parca );
43
44                if ( sil( &baslaPtr, parca ) ) {
45                   printf( "%c silindi.\n", parca );
46                   yazListe( baslaPtr );
47                }
48                else
49                   printf( "%c bulunamadı.\n\n", parca );
50             }
51             else
52                printf( "Liste boş.\n\n" );
53
54             break;
55          default:
56             printf( "Geçersiz seçim.\n\n" );
57             acıklama();
58             break;
59       }
```

```
60
61       printf( "? " );
62       scanf( "%d", &secim );
63    }
64
65    printf( "İşlem sonu.\n" );
66    return 0;
67 }
68
69 /* Açıklamayı yaz */
70 void acıklama( void )
71 {
72    printf( "Seçiminiz:\n"
73            " Listeye bir eleman eklemek için 1.\n"
74            " Listeden bir eleman silmek için 2.\n"
75            " Bitirmek için  3. \n" );
76 }
77
78 /* Sıralanmış listeye yeni bir eleman ekle */
79 void ekle( ListeNoduPtr *sPtr, char deger )
80 {
81    ListeNoduPtr yeniPtr, eskiPtr, suAnkiPtr;
82
83    yeniPtr = malloc( sizeof( ListeNodu ) );
84
85    if ( yeniPtr != NULL ) {
86       yeniPtr->veri= deger;
87       yeniPtr->sonrakiPtr = NULL;
88
89       eskiPtr = NULL;
90       suAnkiPtr = *sPtr;
```

```
91
92        while ( suAnkiPtr != NULL && deger > suAnkiPtr->veri ) {
93           eskiPtr = suAnkiPtr;          /* sonraki noda ...   */
94           suAnkiPtr = suAnkiPtr->sonrakiPtr;  /* ... git */
95        }
96
97        if ( eskiPtr == NULL ) {
98           yeniPtr->sonrakiPtr = *sPtr;
99           *sPtr = yeniPtr;
100       }
101       else {
102          eskiPtr->sonrakiPtr = yeniPtr;
103          yeniPtr->yeniPtr = suAnkiPtr;
104       }
105    }
106    else
107       printf( "%c eklenemedi. Yer yok.\n", deger );
108 }
109
110 /* Liste elemanı silme */
111 char sil( ListeNoduPtr *sPtr, char deger )
112  {
113    ListeNoduPtr eskiPtr, suAnkiPtr, tempPtr;
114
115    if ( deger == ( *sPtr )->veri ) {
116       tempPtr = *sPtr;
117       *sPtr = ( *sPtr )->sonrakiPtr;
118       free( tempPtr );
119       return deger;
120    }
```

```
121    else {
122       eskiPtr = *sPtr;
123       suAnkiPtr = ( *sPtr )->sonrakiPtr;
124
125       while ( suAnkiPtr != NULL && suAnkiPtr->veri != deger ) {
126          eskiPtr = suAnkiPtr;          /* sonraki noda ...   */
127          suAnkiPtr = suAnkiPtr->sonrakiPtr;  /* ... git */
128       }
129
130       if ( suAnkiPtr != NULL ) {
131          tempPtr = suAnkiPtr;
132          eskiPtr->sonrakiPtr = suAnkiPtr->sonrakiPtr;
133          free( tempPtr );
134          return deger;
135       }
136    }
137
138    return '\0';
139 }
140
141 /* Liste boş ise 1 aksi halde 0 gönder */
142 int bos( ListeNoduPtr sPtr )
143 {
144    return sPtr == NULL;
145 }
146
147 /* listeyi yaz */
148 void yazListe( ListeNoduPtr suAnkiPtr )
149 {
150    if ( suAnkiPtr == NULL )
```

12.4 Erişim Listeleri

```
151       printf( "Liste bos.\n\n" );
152    else {
153       printf( "Liste:\n" );
154
155       while ( suAnkiPtr != NULL ) {
156          printf( "%c --> ", suAnkiPtr->veri );
157          suAnkiPtr = suAnkiPtr->sonrakiPtr;
158       }
159
160       printf( "NULL\n\n" );
161    }
162 }
```

12.4 Erişim Listeleri

```
Seçiminiz:
 Listeye bir eleman eklemek için 1.
 Listeden bir eleman silmek için 2.
 Bitirmek için 3.
? 1
Bir karakter gir: B
Liste:
B --> NULL

? 1
Bir karakter gir: A
Liste:
A --> B --> NULL

? 1
Bir karakter gir: C
Liste:
A --> B --> C --> NULL

? 2
Silinecek karakteri gir: D
D bulunamadı.

? 2
Silinecek karakteri gir : B
B silindi.
Liste:
A --> C --> NULL
```

12.5 Yığıtlar

- Yığıt
  - Yeni nod sadece en üste eklenir veya çıkarılır
  - Tabak yığını gibidir
  - Son giren, ilk çıkar (Last-in, first-out (LIFO))
  - Yığıt tabanı **`NULL`** a bir erişim üyesi ile belirlenir
  - Erişimli listenin kısıtlı versiyonu
- **`ilave`**
  - Yığıtın üstüne yeni bir nod ekler
- **`kaldir`**
  - En üsteki nodu kaldırır
  - Kaldırılan değer var ise **`true`** gönderir

```c
/* Fig. 12.8: fig12_08.c
   dinamik yığıt programı */
#include <stdio.h>
#include <stdlib.h>

struct yigitNod {   /* kendine referans yapısı */
   int data;
   struct yigitNod *sonrakiPtr;
};

typedef struct yigitNod YigitNod;
typedef yigitNod  *yigitNodPtr;

void ilave( yigitNodPtr *, int );
int kaldir( yigitNodPtr * );
int bos( yigitNodPtr );
void yazYigit( yigitNodPtr );
void aciklama( void );

int main()
{
   yigitNodPtr yigitPtr = NULL;  /* yığıtın en üstüne pointer */
   int secim, deger;

   aciklama();
   printf( "? " );
   scanf( "%d", &secim );

```

```
29    while ( secim != 3 ) {
30
31       switch ( secim ) {
32          case 1:      /* değeri yığıta ilave et */
33             printf( "Bir tamsayı gir: " );
34             scanf( "%d", &deger);
35             ilave( &yigitPtr, deger );
36             yazYigit( yigitPtr );
37             break;
38          case 2:      /* değeri yığıttan çıkar */
39             if ( !bos( yigitPtr ) )
40                printf( "Atılan değer %d.\n",
41                       kaldir( &yigitPtr ) );
42
43             yazYigit( yigitPtr );
44             break;
45          default:
46             printf( "Yanlış seçim.\n\n" );
47             aciklama();
48             break;
49       }
50
51       printf( "? " );
52       scanf( "%d", &secim );
53    }
54
55    printf( "İşlem sonu.\n" );
56    return 0;
57 }
58
```

```
/* Açıklamaları yaz */
void aciklama( void )
{
   printf( "Seçiminiz:\n"
           " yığıta veri ekleme 1 \n"
           " yığıttan veri çıkarma 2 \n"
           " program sonu 3 \n" );
}

/* Yığıt başına bir nod ekle */
void ilave( yigitNodPtr *topPtr, int bilgi )
{
   yigitNodPtr yeniPtr;

   yeniPtr = malloc( sizeof( yigitNod ) );
   if ( yeniPtr != NULL ) {
      yeniPtr->data = bilgi;
      yeniPtr->yeniPtr = *topPtr;
      *topPtr = yeniPtr;
   }
   else
      printf( "%d eklenemedi. Bellek yetersiz.\n",
              bilgi );
}

```

```
84  /* Yığıt başından veri çıkarma */
85  int kaldir( yigitNodPtr *topPtr )
86  {
87     yigitNodPtr tempPtr;
88     int silDeger;
89
90     tempPtr = *topPtr;
91     silDeger = ( *topPtr )->data;
92     *topPtr = ( *topPtr )->sonrakiPtr;
93     free( tempPtr );
94     return silDeger;
95  }
96
97  /* Yığıtı yaz */
98  void yazYigit( yigitNodPtr suAnkiPtr )
99  {
100    if ( suAnkiPtr == NULL )
101       printf( "Yığıt boş.\n\n" );
102    else {
103       printf( "Yığıt:\n" );
104
105       while ( suAnkiPtr != NULL ) {
106          printf( "%d --> ", suAnkiPtr->data );
107          suAnkiPtr = suAnkiPtr->sonrakiPtr;
108       }
109
110       printf( "NULL\n\n" );
111    }
112 }
113
```

12.5 Yığıtlar

```
114 /* Yığıt boş mu? */
115 int bos( yigitNodPtr topPtr )
116 {
117    return topPtr == NULL;
118 }
```

```
Seçiminiz:
Yığıta veri ekleme 1
Yığıttan veri çıkarma 2
Program sonu 3
? 1
Bir tamsayı gir: 5
Yığıt:
5 --> NULL

? 1
Bir tamsayı gir: 6
Yığıt:
6 --> 5 --> NULL

? 1
Bir tamsayı gir: 4
Yığıt:
4 --> 6 --> 5 --> NULL

? 2
Atılan değer 4.
Yığıt:
6 --> 5 --> NULL
```

12.5 Yığıtlar

```
? 2
Atılan değer 6.
Yığıt:
5 --> NULL

? 2
Atılan değer 5.
Yığıt boş.

? 2
Yığıt boş.

? 4
Yanlış seçim.

Seçiminiz:
Yığıta veri ekleme 1
Yığıttan veri çıkarma 2
Program sonu 3

? 3
Program sonu.
```

12.6 Kuyruklar

- Kuyruk
  - Bir süpermarketdeki kasa kuyruğu gibi
  - İlk giren ilk çıkar (First-in, first-out (FIFO))
  - Nodlar en önden atılır (kuyruk başından)
  - Yeni nod sadece kuyruk sonuna eklenir
- Ekleme ve kaldırma operatörleri
  - kuyrukekle (ekleme) ve kuyrukat(çıkarma)

```c
/* Fig. 12.13: fig12_13.c
   Bir kuyruk üzerinde işlem ve bakım */

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

struct kuyrukNod {    /* kendine referans yapısı */
   char data;
   struct kuyrukNod  *sonrakiPtr;
};

typedef struct kuyrukNod KuyrukNod;
typedef kuyrukNod  *kuyrukNodPtr;

/* fonksiyonlar */
void yazKuyruk( kuyrukNodPtr );
int bos( kuyrukNodPtr );
char kuyrukat( kuyrukNodPtr *, kuyrukNodPtr * );
void kuyrukekle( kuyrukNodPtr *, kuyrukNodPtr *, char );
void aciklama( void );

int main()
{
   kuyrukNodPtr basPtr = NULL, sonPtr = NULL;
   int secim;
   char parca;

   aciklama();
   printf( "? " );
   scanf( "%d", &secim );
```

```
31
32    while ( secim != 3 ) {
33
34       switch( secim ) {
35
36          case 1:
37             printf( "Bir karakter gir: " );
38             scanf( "\n%c", &parca );
39             kuyrukekle( &basPtr, &sonPtr, parca);
40             yazKuyruk( basPtr );
41             break;
42          case 2:
43             if ( !bos( basPtr ) ) {
44                parca = kuyrukat( &basPtr, &sonPtr );
45                printf( "%c atıldı.\n", parca );
46             }
47
48             yazKuyruk( basPtr );
49             break;
50
51          default:
52             printf( "Yanlış secim.\n\n" );
53             aciklama();
54             break;
55       }
56
57       printf( "? " );
58       scanf( "%d", &secim );
59    }
60
61    printf( "İslem sonu.\n" );
62    return 0;
63 }
64
```

```
65 void aciklama( void )
66 {
67    printf ( "Seçiminiz:\n"
68             "   1 -kuyruğa veri ekle \n"
69             "   2 -kuyruktan veri çıkar \n"
70             "   3 -son \n" );
71 }
72
73 void kuyrukekle( kuyrukNodPtr *basPtr, kuyrukNodPtr *sonPtr,
74                  char deger )
75 {
76    kuyrukNodPtr yeniPtr;
77
78    yeniPtr = malloc( sizeof( kuyrukNod ) );
79
80    if ( yeniPtr != NULL ) {
81       yeniPtr->data = deger;
82       yeniPtr->sonrakiPtr = NULL;
83
84       if ( bos( *basPtr ) )
85          *basPtr = yeniPtr;
86       else
87          ( *sonPtr )->sonrakiPtr = yeniPtr;
88
89       *sonPtr = yeniPtr;
90    }
91    else
92       printf( "%c eklenemedi. Bellek yetersiz.\n",
93               deger );
94 }
95
```

```
96 char kuyrukat( kuyrukNodPtr *basPtr, kuyrukNodPtr *sonPtr )
97 {
98    char deger;
99    kuyrukNodPtr tempPtr;
100
101   deger = ( *basPtr )->data;
102   tempPtr = *basPtr;
103   *basPtr = ( *basPtr )->sonrakiPtr;
104
105   if ( *basPtr == NULL )
106      *sonPtr = NULL;
107
108   free( tempPtr );
109   return deger;
110 }
111
112 int bos( kuyrukNodPtr basPtr )
113 {
114   return basPtr == NULL;
115 }
116
117 void yazKuyruk( kuyrukNodPtr suAnkiPtr )
118 {
119   if (suAnkiPtr == NULL )
120      printf( "Kuyruk boş.\n\n" );
121   else {
122      printf( "Kuyruk:\n" );
```

12.6 Kuyruklar

```
123
124       while (suAnkiPtr != NULL ) {
125          printf( "%c --> ", suAnkiPtr ->data );
126                   suAnkiPtr = suAnkiPtr ->sonrakiPtr;
127       }
128
129       printf( "NULL\n\n" );
130    }
131 }
```

```
Seçiminiz:
   1 -kuyruğa veri ekle
   2 -kuyruktan veri çıkar
   3 -son
? 1
Bir karakter gir: A
Kuyruk:
A --> NULL

? 1
Bir karakter gir: B
Kuyruk:
A --> B --> NULL

? 1
Bir karakter gir: C
Kuyruk:
A --> B --> C --> NULL
```

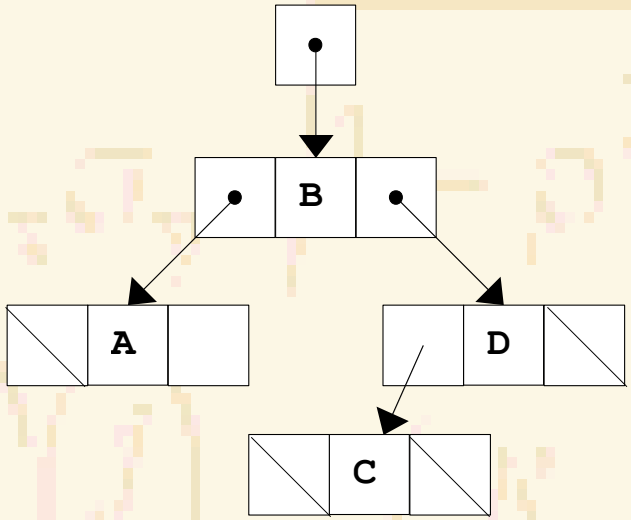
12.6 Kuyruklar

```
? 2
A atıldı.
Kuyruk:
B --> C --> NULL

? 2
B atıldı.
Kuyruk:
C --> NULL

? 2
C atıldı.
Kuyruk boş.

? 2
Kuyruk boş.

? 4
Yanlış seçim.

Seçiminiz:
   1 -kuyruğa veri ekle
   2 -kuyruktan veri çıkar
   3 -son

? 3
İşlem sonu.
```

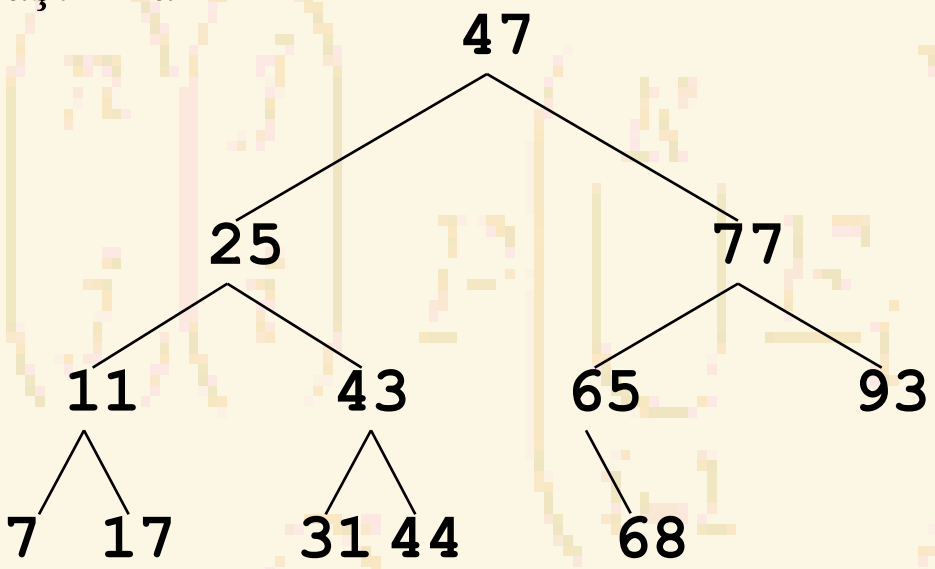
12.7 Ağaçlar

- Ağaç nodları iki veya daha çok erişim içerir
  - Şimdiye kadar olan tüm veri yapıları bir erişim içerir
- İkili ağaç
  - Tüm nodlar iki erişim içerir
    - Hiçbiri, biri veya her ikisi de NULL olabilir
  - Kök nod ağaçtaki ilk noddur.
  - Kök noddaki her bir erişim bir dala gider
  - Dalı olmayan noda yaprak nod denir

- İkili ağaç diagramı



- İkili tarama ağacı
  - Sol alt daldaki değerler köktekinden küçüktür
  - Sağ alt daldaki değerler köktekinden büyüktür
  - Tekrarlı eliminasyonu kolaylaştırır
  - Hızlı tarama – dengeli bir ağaçta, en fazla (log n) karşılaştırma



- Ağaç taraması:
  - Sıralı tarama – nod değerlerini artan sırada yazar
    1. Sıralı tarama ile sol alt dalı tara
    2. Nod değerini işle (yaz)
    3. Sıralı tarama ile sağ alt dalı tara
  - Önsıralı tarama
    1. Nod değerini işle
    2. Önsıralı tarama ile sol alt dalı tara
    3. Önsıralı tarama ile sağ alt dalı tara
  - Sonsıralı tarama
    1. Sonsıralı tarama ile sol alt dalı tara
    2. Sonsıralı the tarama ile sağ alt dalı tara
    3. Nod değerini işle

```c
/* Fig. 12.19: fig12_19.c
   İkili ağaç oluştur ve tara
   önsıralı, sıralı, ve sonsıralı */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct agacNod {
   struct agacNod *solPtr;
   int data;
   struct agacNod *sagPtr;
};

typedef struct agacNod AgacNod;
typedef AgacNod *AgacNodPtr;

void ekleNod( AgacNodPtr *, int );
void sirali( AgacNodPtr );
void onsirali( AgacNodPtr );
void sonsirali( AgacNodPtr );

int main()
{
   int i, parca;
   AgacNodPtr kokPtr = NULL;

   srand( time( NULL ) );

```

```
29    /* Ağaca 1 15 arasında rasgele sayı ekle */
30    printf( "Ağaca yerlestirilen sayılar:\n" );
31
32    for ( i = 1; i <= 10; i++ ) {
33       parca = rand() % 15;
34       printf( "%3d", parca );
35       ekleNod( &kokPtr, parca );
36    }
37
38    /* Önsıralı tarama */
39    printf( "\n\n Önsıralı tarama:\n" );
40    onsirali( kokPtr );
41
42    /* Sıralı tarama */
43    printf( "\n\n Sıralı tarama:\n" );
44    sirali( kokPtr );
45
46    /* Sonsıralı tarama */
47    printf( "\n\n Sonsıralı tarama:\n" );
48    sonsirali( kokPtr );
49
50    return 0;
51 }
52
53 void ekleNod( AgacNodPtr *agacPtr, int deger )
54 {
55    if ( *agacPtr == NULL ) {     /* *agacPtr NULL */
56       *agacPtr = malloc( sizeof( AgacNod ) );
57
58       if ( *agacPtr != NULL ) {
59          ( *agacPtr )->data = deger;
60          ( *agacPtr )->solPtr = NULL;
61          ( *agacPtr )->sagPtr = NULL;
62       }
```

```
63       else
64          printf( "%d eklenemedi. Bellek dolu.\n",
65                  deger );
66    }
67    else
68       if ( deger < ( *agacPtr )->data )
69          ekleNod( &( ( *agacPtr )->solPtr ), deger );
70       else if ( deger > ( *agacPtr )->data )
71          ekleNod( &( ( *agacPtr )->sagPtr ), deger );
72       else
73          printf( "tekrar" );
74 }
75
76 void sirali( AgacNodPtr agacPtr )
77 {
78    if ( agacPtr != NULL ) {
79       sirali( agacPtr->solPtr );
80       printf( "%3d", agacPtr->data );
81       sirali( agacPtr->sagPtr );
82    }
83 }
84
85 void onsirali( AgacNodPtr agacPtr )
86 {
87    if ( agacPtr != NULL ) {
88       printf( "%3d", agacPtr->data );
89       onsirali( agacPtr->solPtr );
90       onsirali( agacPtr->sagPtr );
91    }
92 }
```

12.7 Ağaçlar

```
93
94 void sonsirali( AgacNodPtr agacPtr )
95 {
96    if ( agacPtr != NULL ) {
97       sonsirali( agacPtr->solPtr );
98       sonsirali( agacPtr->sagPtr );
99       printf( "%3d", agacPtr->data );
100   }
101}
```

```
Ağaca yerleştirilen sayılar:
  7   8   0   6 14   1   0tekrar 13   0tekrar   7tekrar

Önsıralı tarama:
  7   0   6   1   8 14 13

Sıralı tarama:
  0   1   6   7   8 13 14

Sonsıralı tarama:
  1   6   0 13 14   8   7
```