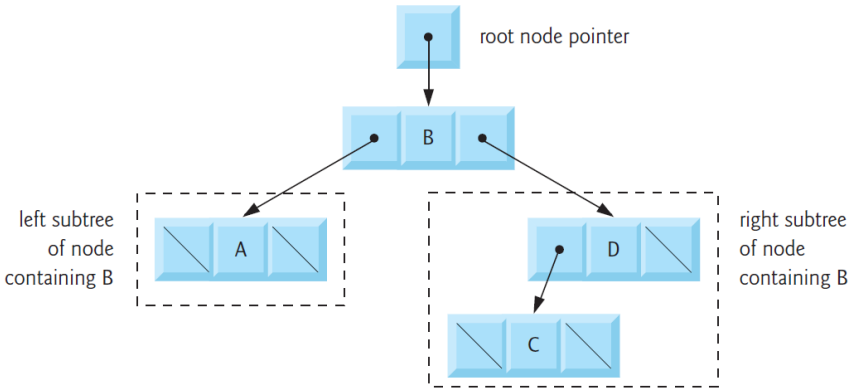
NOT-1: Bu bölüm "Paul Deitel, Harvey Deitel - 9nth_editionC How to Program-Pearson (2022)" kitabı 9ncu baskıdan anlatılmıştır. Bu bölüm 8nci baskıya göre daha açıklamalı verildiğinden 9ncu baskı tercih edilmiştir.

NOT-2: Bu bölündeki programları kaydederken C++ compiler kullanmayın, hata verebilir. Bunun yerine yazdığınız programı C compiler olarak kaydedin ve çalıştırın.

Ağaç Yapıları (Trees)

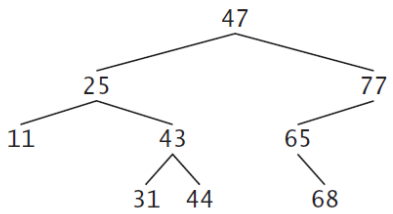
Şimdiye kadar doğrusal veri yapılarını (bağlantılı listeler, yığınlar ve kuyruklar) gördük. Bir ağaç, özel özelliklere sahip doğrusal olmayan, iki boyutlu bir veri yapısıdır. Ağaç düğümleri iki veya daha fazla bağlantı içerir. Bu bölümde, aşağıdaki diyagramda olduğu gibi, düğümler iki bağlantı içeren ikili ağaçlar (binary tree) ile açıklanmaktadır:



Her düğümdeki bağlantıların hiçbiri, biri veya her ikisi de NULL olabilir. Kök düğüm, ağaçtaki ilk düğümdür. Kök düğümdeki her bağlantı bir çocuğu ifade eder. Sol çocuk, sol alt ağaçtaki ilk düğümdür ve sağ çocuk, sağ alt ağaçtaki ilk düğümdür. Belirli bir düğümün çocuklarına kardeşler denir. Çocuğu olmayan bir düğüm yaprak düğümdür. Bir yaprak düğümün bağlantılarını NULL olarak ayarlamamak çalışma zamanı hatalarına yol açabilir. Bilgisayar bilimciler tipik olarak ağaçları kök düğümü en üstte olacak şekilde çizerler; bu, doğadaki ağaçların tam tersidir.

İkili Arama Ağacı (binary search tree)

Bu bölümde, herhangi bir sol alt ağaçtaki değerlerin üst düğümdeki değerlerden küçük olması ve herhangi bir sağ alt ağaçtaki değerlerin üst düğümdeki değerlerden büyük olması özelliğine sahip, farklı değerler içeren bir ikili arama ağaç yapısı işlenecektir. Aşağıdaki şekil, dokuz değere sahip bir ikili arama ağacını göstermektedir:



Bir veri kümesi için ikili arama ağacının şekli, değerlerini girdiğiniz sıraya göre değişebilir.

İkili Arama Ağacı (Binary Search Tree) Uygulaması

Şekil 12.4'te verilen program bir ikili arama ağacı oluşturur ve düğümlerini çapraz takip eder, yani ağaçtaki her düğümü ziyaret ederek düğüm değerlerini gösterir. Ağacın düğümlerine üç şekilde geçilebilir - sıralı, ön sıralı ve son sıralı (inOrder, preOrder ve postOrder). Program 10 rasgele sayı üretir ve yinelenen değerlerin atılması dışında her birini ağaca ekler.

9–13 arasındaki satırlar, ağacın düğümlerini temsil etmek için kullanacağımız struct treeNode'u tanımlar. Yine, kodu daha okunaklı hale getirmek için typedef'leri (satır 15–16) kullanıyoruz.

```
1  // fig12_04.c
2  // Creating and traversing a binary tree
3  // preorder, inorder, and postorder
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  // self-referential structure
9  struct treeNode {
10    struct treeNode *leftPtr; // pointer to left subtree
11    int data; // node value
12    struct treeNode *rightPtr; // pointer to right subtree
13 };
14
15 typedef struct treeNode TreeNode; // synonym for struct treeNode
16 typedef TreeNode *TreeNodePtr; // synonym for TreeNode *
17
18 // prototypes
19 void insertNode(TreeNodePtr *treePtr, int value);
20 void inOrder(TreeNodePtr treePtr);
21 void preOrder(TreeNodePtr treePtr);
22 void postOrder(TreeNodePtr treePtr);
23
24 int main(void) {
25    TreeNodePtr rootPtr = NULL; // tree initially empty
26
```

insertNode Fonksiyonu

Şekil 12.4'te ikili arama ağacı oluşturan ve onu çapraz gezinen fonksiyonlar yinelemelidir (recursive). insertNode fonksiyonu (satır 51–75), ağacın kök düğümüne işaretçinin adresini ve eklenecek bir tamsayıyı bağımsız değişken olarak alır. İkili arama ağacındaki (Binary Search Tree) her yeni düğüm, başlangıçta bir yaprak düğüm olarak eklenir. Yeni bir düğüm ekleme adımları aşağıdaki gibidir:

1. *treePtr NULL ise (satır 52), 53. satır malloc'u çağırarak yeni bir yaprak düğüm oluşturur ve ayrılan belleği *treePtr'ye atar. Satır 56 (*treePtr)->data'ya depolanacak tamsayıyı atar. 57–58 arasındaki satırlar (*treePtr)->leftPtr ve (*treePtr)->rightPtr'ye NULL atar. Ardından kontrol çağrı yapana geri döner – bu, ya ana çağrı ya da insertNode'a bir önceki çağrıdır.

2. *treePtr NULL değilse ve eklenecek değer (*treePtr)->data'dan küçükse, 66. satır, yeni düğümü treePtr tarafından işaret edilen düğüm öğesinin sol alt ağacına eklemek için (*treePtr)->leftPtr adresiyle yinelemeli olarak (recursively) insertNode'u çağırır.

3. Eklenecek değer (*treePtr)-> data 'dan büyükse, satır 69, treePtr tarafından işaret edilen düğümün sağ alt ağacına eklemek için (*treePtr)->rightPtr adresiyle yinelemeli olarak (recursively) insertNode'u çağırır.

Özyinelemeli adımlar, insertNode bir NULL işaretçisi bulana kadar devam eder, ardından Adım 1, yeni düğümü yaprak düğüm olarak ekler.

NOT: Burada Derlemeyi C'de yaptığımız için i değişkeninin for döngüsü dışında int i; olarak tanımlanması gerekiyor. Kitaptaki programda 31nci satırda bu düzeltmeyi yapalım.

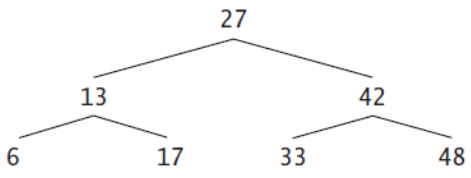
```
27      srand(time(NULL));
28      puts("The numbers being placed in the tree are:");
29
30      // insert random values between 0 and 14 in the tree
31      for (int i = 1; i <= 10; ++i) {
32         int item = rand() % 15;
33         printf("%3d", item);
34         insertNode(&rootPtr, item);
35      }
36
37      // traverse the tree preOrder
38      puts("\n\nThe preOrder traversal is:");
39      preOrder(rootPtr);
40
41      // traverse the tree inOrder
42      puts("\n\nThe inOrder traversal is:");
43      inOrder(rootPtr);
44
45      // traverse the tree postOrder
46      puts("\n\nThe postOrder traversal is:");
47      postOrder(rootPtr);
48   }
49
50   // insert node into tree
51   void insertNode(TreeNodePtr *treePtr, int value) {
52      if (*treePtr == NULL) { // if tree is empty
53         *treePtr = malloc(sizeof(TreeNode));
54
55         if (*treePtr != NULL) { // if memory was allocated, then assign data
56            (*treePtr)->data = value;
57            (*treePtr)->leftPtr = NULL;
58            (*treePtr)->rightPtr = NULL;
59         }
60         else {
61            printf("%d not inserted. No memory available.\n", value);
62         }
63      }
64      else { // tree is not empty
65         if (value < (*treePtr)->data) { // value goes in left subtree
66            insertNode(&((*treePtr)->leftPtr), value);
67         }
68         else if (value > (*treePtr)->data) { // value goes in right subtree
69            insertNode(&((*treePtr)->rightPtr), value);
70         }
71         else { // duplicate data value ignored
72            printf("%s", "dup");
73         }
74      }
75   }
76
```

```
77  // begin inorder traversal of tree
78  void inOrder(TreeNodePtr treePtr) {
79     // if tree is not empty, then traverse
80     if (treePtr != NULL) {
81        inOrder(treePtr->leftPtr);
82        printf("%3d", treePtr->data);
83        inOrder(treePtr->rightPtr);
84     }
85  }
86
87  // begin preorder traversal of tree
88  void preOrder(TreeNodePtr treePtr) {
89     // if tree is not empty, then traverse
90     if (treePtr != NULL) {
91        printf("%3d", treePtr->data);
92        preOrder(treePtr->leftPtr);
93        preOrder(treePtr->rightPtr);
94     }
95  }
96
97  // begin postorder traversal of tree
98  void postOrder(TreeNodePtr treePtr) {
99     // if tree is not empty, then traverse
100    if (treePtr != NULL) {
101       postOrder(treePtr->leftPtr);
102       postOrder(treePtr->rightPtr);
103       printf("%3d", treePtr->data);
104    }
105 }
```

```
The numbers being placed in the tree are:
  6  7  4 12  7dup  2  2dup  5  7dup 11

The preOrder traversal is:
  6  4  2  5  7 12 11

The inOrder traversal is:
  2  4  5  6  7 11 12

The postOrder traversal is:
  2  5  4 11 12  7  6
```

Çapraz Geçişler: inOrder, preOrder ve postOrder fonksiyonları

inOrder (78–85. satırlar), preOrder (88–95. satırlar) ve postOrder (98–105. satırlar) fonksiyonlarının her biri, bir ağacın kök düğümüne birer işaretçi alır ve ağacı çapraz takip eder. Aşağıda verilen geçişleri açıklayalım ve aşağıdaki ağaç için her birinin uygulanmasının sonucunu gösterelim:



inOrder (Sıralı) Çapraz Geçiş

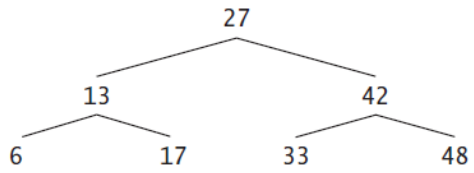
Bir inOrder geçişi için adımlar şunlardır:

1. Sol alt ağacı sırayla çapraz taranır (satır 81).

2. Düğümdeki değer işlenir (satır 82).

3. Sırayla sağ alt ağacı çapraz taranır (satır 83).

Bu geçiş, sol alt ağaçtaki değerleri işledikten sonra her bir düğümün değerini işler. Önceki ağacın inOrder geçişi şöyledir:



6 13 17 27 33 42 48

Bir ikili arama ağacının sıralı geçişi, düğümleri artan düzende işler. Bir ikili arama ağacı oluşturmak aslında verileri sıralar. Dolayısıyla, bu işleme ikili ağaç sıralaması denir.

preOrder (Ön Sıralı) Çapraz Geçiş

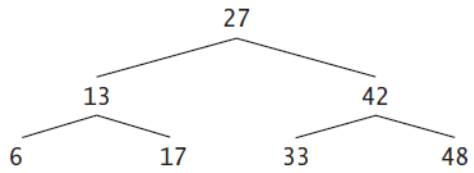
preOrder çapraz geçişi için adımlar şunlardır:

1. Düğümdeki değeri işleyin (satır 91).

2. Sol alt ağaç ön sıralı (preOrder) (satır 92) çapraz taranır.

3. Sağ alt ağaç ön sıralı (preOrder) (satır 93) çapraz taranır.

Bu geçiş, düğüm ziyaret edildiğinde her düğümün değerini işler. Değeri işledikten sonra, bir ön sıralı (preOrder) geçişi sol alt ağacın değerlerini, ardından sağ alt ağacın değerlerini işler. Önceki ağacın ön sıralı (preOrder) geçişi şöyledir:



27 13 6 17 42 33 48

Son sıralı (postOrder) çaprazlama geçiş

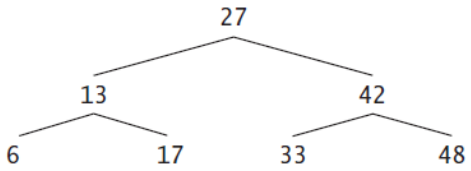
Bir postOrder geçişi için adımlar şunlardır:

1. Sol alt ağaç son sıralı (postOrder) (satır 101) çapraz taranır.

2. Sağ alt ağaç son sıralı (postOrder) (satır 102) çapraz taranır.

3. Düğümdeki değer işlenir (satır 103).

Bu geçişte, önce sol sonra sağ alt ağaçlarda düğümün çocuklarının değerleri işlenir daha sonra her düğümün değeri işlenir. Önceki ağacın postOrder geçişi şöyledir:



6 17 13 33 48 42 27

Tekrarları Eleme

İkili arama ağaçları, tekrarlanan verileri elemeyi kolaylaştırır. Ağacı oluşturmak için değerler eklediğinizde, yinelenen bir değer, orijinal değerin yaptığı gibi her karşılaştırmada aynı "sola git" veya "sağa git" kararlarını takip edecektir. Böylece, yinelenen, sonunda ağaçta aynı değeri içeren bir düğümle karşılaştırılacaktır. Bu noktada, yinelenen değer göz ardı edilir.