

## MODUL PRAKTIKUM IFS2101 – Algorithma dan Struktur Data

### Binary Search Tree ADT

Minggu	:	11
Tujuan	:	<ul style="list-style-type: none"> <li>• Mampu membedakan karakteristik struktur data non-linier dengan struktur data linier.</li> <li>• Mampu mendefinisikan, menggambar skema diagram, dan menjelaskan teknik implementasi pohon umum (<i>generic tree</i>)</li> <li>• Mampu menggambarkan skema diagram untuk implementasi pohon umum yang efisien dengan struktur linked list.</li> <li>• Mampu menjelaskan varian-varian struktur data pohon.</li> <li>• Mampu mendefinisikan dan menggambarkan skema diagram pohon biner</li> <li>• Mampu mendefinisikan dan menggambarkan skema diagram pohon pencarian biner (<i>binary search tree</i>).</li> <li>• Mampu memprogram Binary Search Tree ADT untuk aksi <i>insert</i>, <i>update</i>, <i>search</i>, dan <i>delete</i>.</li> </ul>
Setoran	:	<b>Kertas Jawaban diserahkan kepada TA Program disubmit via moodle (lihat poin 3 Bagian Pemrograman)</b>
Waktu penyetoran	:	<b>24 November 2015 pukul 17:00 WIB</b>

### Petunjuk Praktikum

1. Anda dapat mengerjakan praktikum ini secara berkelompok dengan maksimum 3 mahasiswa per kelompok.

### Referensi

1. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithm*, 2<sup>nd</sup> eds., MIT Press 2001.
2. M.A. Weiss, *Data Structures and Algorithm Analysis in C*, 2<sup>nd</sup> Eds., Addison-Wesley, 1997.

### Ulasan

1. Jelaskan perbedaan karakteristik struktur data non-linier dari struktur data linier.
2. Definisikanlah pohon umum secara lengkap.
3. Jelaskan teknik implementasi pohon umum yang efisien dimana informasi mengenai jumlah anak setiap node tidak diketahui sebelumnya.
4. Jelaskan varian-varian struktur data tree.
5. Definisikanlah pohon pencarian biner. Deklarasi fungsi pencarian yang ada fail antarmuka `binary_search_tree.h` pada praktikum ini diberikan sebagai berikut:  

```
Position Find(ElementType X, BinarySearchTree T);
```

Jelaskanlah fungsi tersebut menurut hal berikut: tipe kembalian dan argumen (parameter) fungsi.
6. Deklarasi fungsi penghapusan yang ada fail antarmuka `binary_search_tree.h` pada praktikum ini diberikan sebagai berikut:  

```
BinarySearchTree Delete(ElementType X, BinarySearchTree T);
```

Jelaskanlah fungsi tersebut menurut hal berikut: tipe kembalian dan argumen (parameter) fungsi.

### **Pendalaman**

1. Gambarkanlah skema pohon umum untuk data kunci indeks (*index keys data*) yang berakar pada index\_key 12.

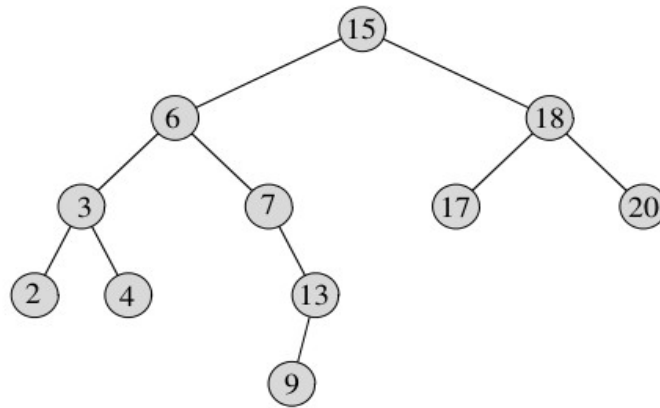
index_key	child_1	child_2	child_3	child_4
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-
4	6	9	-	-
5	8	-	-	-
6	-	-	-	-
7	-	-	-	-
8	15	-	-	-
9	-	-	-	-
10	-	-	-	-
11	16	18	3	-
12	11	13	17	-
13	4	1	-	-
14	-	-	-	-
15	-	-	-	-
16	-	-	-	-
17	7	5	10	14
18	2	-	-	-

2. Mengacu pada soal no. 1. Gambarkanlah skema diagram untuk implementasi pohon umum tersebut dengan efisien (yang memungkinkan jumlah node berapa pun untuk setiap simpul).
3. Diambil dari [1] *exercise 10.4-1*, halaman 216. Gambarkanlah skema diagram pohon biner yang berakar pada index 6 untuk data berikut:

index	key	left	right
1	12	7	3
2	15	8	NIL
3	4	10	NIL
4	10	5	9
5	2	NIL	NIL
6	18	1	4
7	7	NIL	NIL
8	14	6	2
9	21	NIL	NIL
10	5	NIL	NIL

4. Jika diberikan himpunan data berikut:  $D = \{1, 2, 3, \dots, 20\}$ . Gambarkanlah skema diagram dari pohon pencarian biner (*binary search tree*) untuk menyimpan data tersebut.

Soal non 5 s/d 9 mengacu ke struktur pohon pencarian biner (*Binary Search Tree*) yang ditunjukkan oleh gambar di bawah ini. Gambar tersebut diambil dari [1] , Figure 12.2, halaman 257.



5. Gambarkanlah pohon baru yang terbentuk akibat penghapusan simpul dengan elemen (nilai kunci) = 4.
6. Gambarkanlah pohon baru yang terbentuk akibat penghapusan simpul dengan elemen (nilai kunci) = 7.
7. Gambarkan pula pohon baru yang terbentuk akibat penghapusan *root* (akar).
8. Tunjukkanlah dengan perhitungan dan dengan gambar waktu terbaik dan waktu terburuk dari operasi dasar (*insert*, *search*, *delete*) pada Binary Search Tree.
9. Jumlah kombinasi dari Binary Search Tree yang mungkin terbentuk untuk elemen sebesar  $n$  adalah faktorial  $n$  ( $n!$ ). Gambarkanlah semua kombinasi Binary Search Tree yang mungkin terbentuk untuk himpunan elemen berikut:  $S = \{1, 3, 5\}$ .

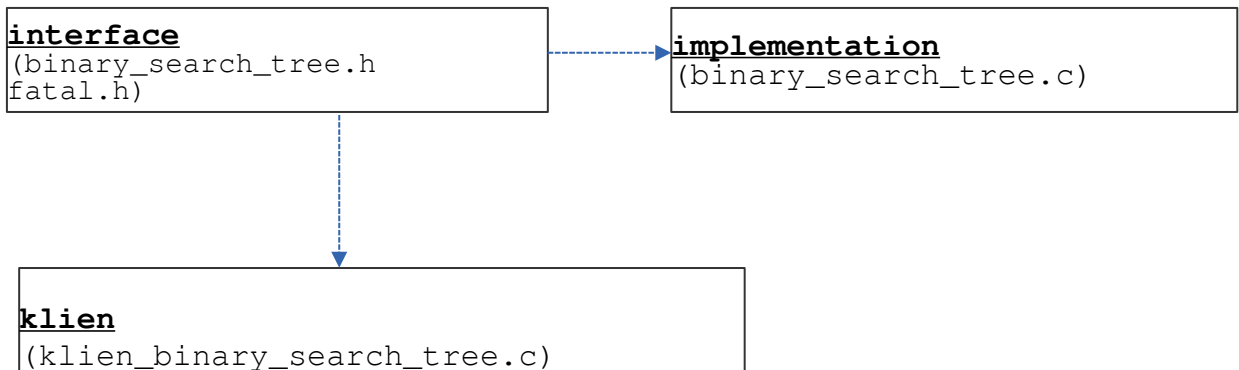
## Pemrograman:

Pada praktikum ini anda akan menggunakan bahasa pemrograman C untuk membuat **Binary Search Tree ADT**, yakni sebuah pohon biner dengan karakteristik dimana elemen yang lebih kecil diletakkan pada node sebelah kiri dan elemen yang lebih besar diletakkan pada node sebelah kanan. Asumsi yang kita gunakan adalah elemen pohon unik (tidak ada elemen yang sama). Pada dasarnya anda juga dapat membuat beberapa elemen dengan nilai yang sama dan menyesuaikan kode program.

Dengan pemahaman dan keahlian pemrograman yang anda miliki, diharapkan anda dapat membuat ADT untuk varian-varian *tree* yang lain.

### 1. Pemasukan elemen pada Binary Search Tree ADT

Komponen penyusun dari Binary Search Tree ADT ditunjukkan oleh gambar di bawah ini.

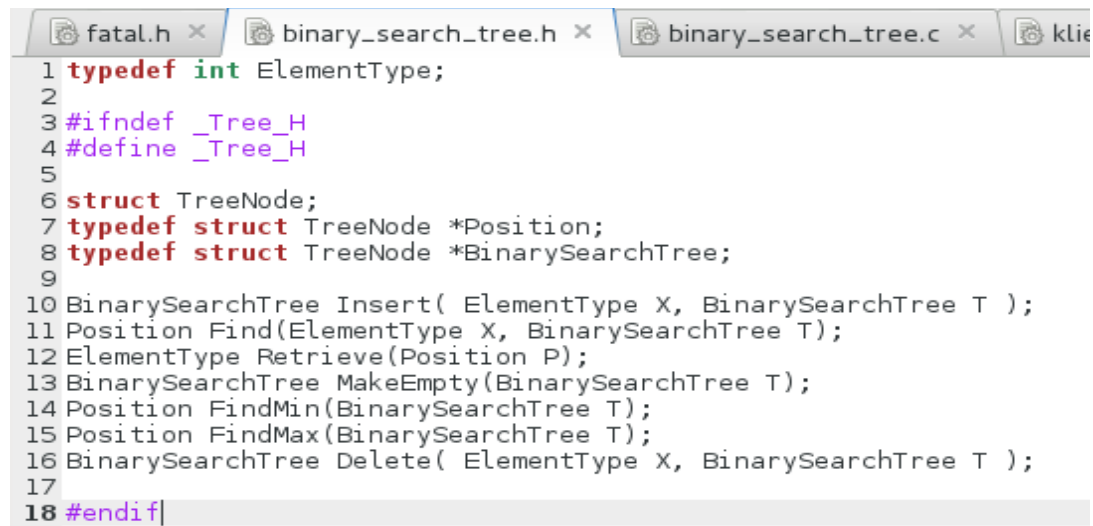


- α) Edit fail berikut dan simpan sebagai `fatal.h`. Fail ini merupakan antarmuka (*interface*) dari Binary Search Tree ADT yang berisi deklarasi dan definisi fungsi yang mencetak pesan error ke monitor.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define Error(Str)      FatalError( Str )
5 #define FatalError(Str) fprintf( stderr, "%s\n", Str ), exit( 1 )
  
```

- β) Selanjutnya, edit fail pada halaman selanjutnya dan simpan sebagai `binary_search_tree.h`.



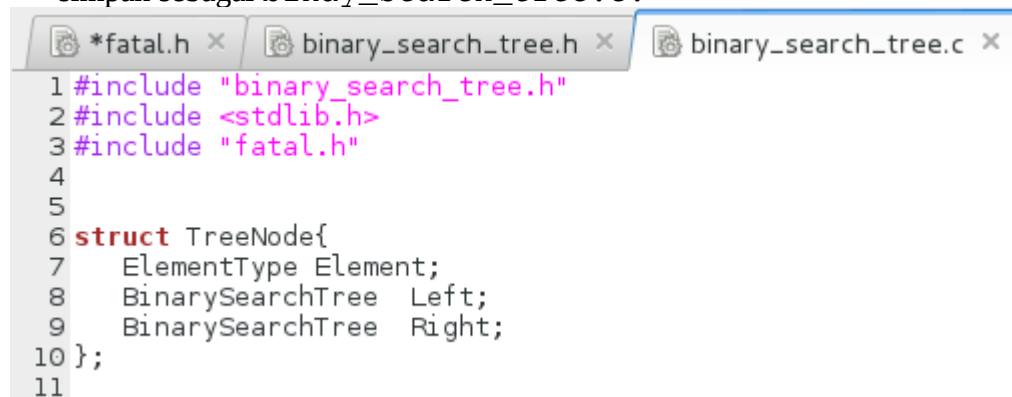
```

1 typedef int ElementType;
2
3 #ifndef _Tree_H
4 #define _Tree_H
5
6 struct TreeNode;
7 typedef struct TreeNode *Position;
8 typedef struct TreeNode *BinarySearchTree;
9
10 BinarySearchTree Insert( ElementType X, BinarySearchTree T );
11 Position Find( ElementType X, BinarySearchTree T );
12 ElementType Retrieve( Position P );
13 BinarySearchTree MakeEmpty( BinarySearchTree T );
14 Position FindMin( BinarySearchTree T );
15 Position FindMax( BinarySearchTree T );
16 BinarySearchTree Delete( ElementType X, BinarySearchTree T );
17
18 #endif

```

Fail ini merupakan antarmuka yang berisi deklarasi operasi-operasi (fungsi-fungsi) legal yang dapat dilakukan pada Binary Search Tree ADT. Simpul pada tree direpresentasikan oleh sebuah struktur: struct TreeNode. TreeNode akan didefinisikan pada fail implementasi. Root dari pohon pencarian biner direpresentasikan oleh tipe data BinarySearchTree yang dideklarasikan sebagai pointer ke struct TreeNode. Fungsi yang menjadi perhatian kita saat ini ini adalah fungsi Insert yang tertulis pada baris ke-10. Fungsi ini menerima 2 parameter, yakni elemen yang akan dimasukkan (ElementType X) dan root (BinarySearchTree T). Tipe kembalian adalah BinarySearchTree.

- χ) Untuk implementasi, pertama sekali anda harus mendefinisikan struktur struct TreeNode yang merupakan representasi simpul pada tree. Edit fail berikut dan simpan sebagai binay\_search\_tree.c.



```

1 #include "binary_search_tree.h"
2 #include <stdlib.h>
3 #include "fatal.h"
4
5
6 struct TreeNode{
7     ElementType Element;
8     BinarySearchTree Left;
9     BinarySearchTree Right;
10 };
11

```

- δ) Sebelum menulis fungsi Insert, ada baiknya kita implementasi dulu fungsi MakeEmpty (lihat antarmuka binary\_search\_tree.h, baris ke-13). Fungsi ini bertujuan untuk mengosongkan tree. Tambahkan fungsi MakeEmpty ke dalam fail implementasi: binay\_search\_tree.c.

```

38 BinarySearchTree MakeEmpty(BinarySearchTree T){
39     if(T != NULL){
40         MakeEmpty( T->Left );
41         MakeEmpty( T->Right );
42         free(T);
43     }
44     return NULL;
45 }

```

Anda dapat melihat bahwa pengosongan pohon dilakukan dengan mengosongkan root T dan child nya, yakni T->Left dan T->Right. Fungsi tersebut mengembalikan NULL pointer ke klien.

- ε) Selanjutnya, anda akan mengimplementasikan fungsi Insert. Tambahkan fungsi Insert yang ada di bawah ini ke fail implementasi `binay_search_tree.c`.

```

21 BinarySearchTree Insert(ElementType X, BinarySearchTree T){
22     if(T == NULL){
23         T = malloc(sizeof(struct TreeNode ) );
24         if( T == NULL )
25             FatalError( "Out of space!!!" );
26         else{
27             T->Element = X;
28             T->Left = T->Right = NULL;
29         }
30     }
31     else if( X < T->Element ){
32         T->Left = Insert(X, T->Left);
33     }
34     else if( X > T->Element ){
35         T->Right = Insert(X, T->Right);
36     }
37     //printf("C imp: T->Element = %d \n", T->Element);
38 |
39     return T;
40 }

```

- φ) Untuk mengambil elemen dari struktur data, anda menggunakan fungsi Retrieve (lihat antarmuka `binary_search_tree.h`, baris ke-12). Tambahkan definisi fungsi tersebut ke fail implementasi: `binay_search_tree.c`.

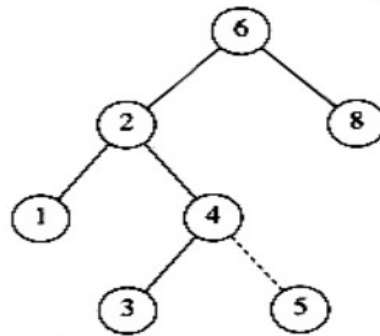
```

43 ElementType Retrieve( Position P ){
44     return P->Element;
45 }
46 |

```

Fungsi Retrieve terlihat sederhana. Fungsi tersebut menerima parameter P, yakni satu simpul pada Binary Search Tree ADT dan mengembalikan elemen simpul tersebut. P bertipe `Position`, yakni sebuah tipe data yang didefinisikan sebagai pointer ke `struct TreeNode` (lihat antarmuka `binary_search_tree.h`, baris ke-7).

- γ) Terakhir sekali, anda akan membuat program klien untuk mengakses struktur data. Elemen yang akan kita insert ditunjukkan oleh skema diagram dari *binary search tree* berikut.



Program main untuk membuat struktur pohon pencarian biner tersebut sebelumnya dapat anda lihat pada kode di bawah ini. Edit dan simpan sebagai: klien\_binary\_search\_tree.c.

```

1 #include "binary_search_tree.h"
2 #include <stdio.h>
3
4 int main( ){
5     BinarySearchTree T;
6
7     T = MakeEmpty(NULL); //fungsi untuk mengosongkan binary search tree
8
9     T = Insert(6,T);
10    T = Insert(2,T);
11    T = Insert(8,T);
12    T = Insert(1,T);
13    T = Insert(4,T);
14    T = Insert(5,T);
15
16    printf("\nElemen root : T->Element = %d\n", Retrieve(T));
17
18    return 0;
19 }
20 |
  
```

η) Kompilasi dan jalankan program sebagai berikut:

```

root@kodi1:/home/c_alg_sd/mg11_sesi3# gcc -Wall klien_binary_search_tree.c binary_search_tree.c
root@kodi1:/home/c_alg_sd/mg11_sesi3# ./a.out

Elemen root : T->Element = 6
  
```

## 2. Pencarian elemen dan penghapusan pada Binary Search Tree ADT.

- α) Edit fail `binary_search_tree.c` yang telah anda buat sebelumnya dan tambahkan kode untuk fungsi `FindMin`, `FindMax` dan `Delete` di bawah ini. `FindMin` merupakan fungsi untuk mengambil simpul dengan elemen terkecil, `FindMax` mengembalikan simpul dengan elemen terbesar dan `Delete` untuk menghapus simpul.

```

58 Position FindMin( BinarySearchTree T ){
59     if( T == NULL )
60         return NULL;
61     else
62         if( T->Left == NULL )
63             return T;
64         else
65             return FindMin( T->Left );
66 }
67 Position FindMax( BinarySearchTree T ){
68     if( T != NULL )
69         while( T->Right != NULL )
70             T = T->Right;
71
72     return T;
73 }
74
75 BinarySearchTree Delete( ElementType X, BinarySearchTree T ){
76     Position TmpCell;
77
78     if( T == NULL )
79         Error( "Element not found" );
80     else if( X < T->Element ) /* Go left */
81         T->Left = Delete( X, T->Left );
82     else if( X > T->Element ) /* Go right */
83         T->Right = Delete( X, T->Right );
84     else /* Found element to be deleted */
85         if( T->Left && T->Right ){ /* Two children */
86             /* Replace with smallest in right subtree */
87             TmpCell = FindMin( T->Right );
88             T->Element = TmpCell->Element;
89             T->Right = Delete( T->Element, T->Right );
90         } else { /* One or zero children */
91             TmpCell = T;
92             if( T->Left == NULL ) /* Also handles 0 children */
93                 T = T->Right;
94             else if( T->Right == NULL )
95                 T = T->Left;
96             free( TmpCell );
97         }
98     return T;
99 }

```

- β) Selanjutnya, modifikasi program klien untuk mencari elemen dengan nilai 3, mencari elemen minimum dan mencari elemen maksimum seperti ditunjukkan oleh program pada halaman selanjutnya.

`klien_binary_search_tree.c`.



```

1 #include "binary_search_tree.h"
2 #include <stdio.h>
3
4 int main( ){
5     BinarySearchTree T=NULL;
6     Position P = NULL;
7
8     T = MakeEmpty(NULL); //fungsi untuk mengosongkan binary search tree
9
10    //pembuatan BinarySearchTree
11    T = Insert(6,T);
12    T = Insert(2,T);
13    T = Insert(8,T);
14    T = Insert(1,T);
15    T = Insert(4,T);
16    T = Insert(3,T);
17    printf("\nElemen root : T->Element = %d\n", Retrieve(T));
18
19    //mencari dan mencetak elemen 3
20    P = Find(3,T);
21    printf("\nElemen dengan nilai kunci %d ditemukan\n", Retrieve(P));
22
23    //mencari elemen minimum
24    P = FindMin(T);
25    printf("\nElemen (nilai kunci) minimum = %d\n", Retrieve(P));
26
27    //mencari elemen maksimum
28    P = FindMax(T);
29    printf("\nElemen (nilai kunci) maksimum = %d\n", Retrieve(P));
30
31    return 0;
32 }

```

χ) Kompilasi dan jalankan program sebagai berikut:

```

root@kodi1:/home/c_alg_sd/mq12_sesi3# gcc -Wall binary_search_tree.c klien_binary_search_tree.c -o klien_bst
root@kodi1:/home/c_alg_sd/mq12_sesi3# ./klien_bst

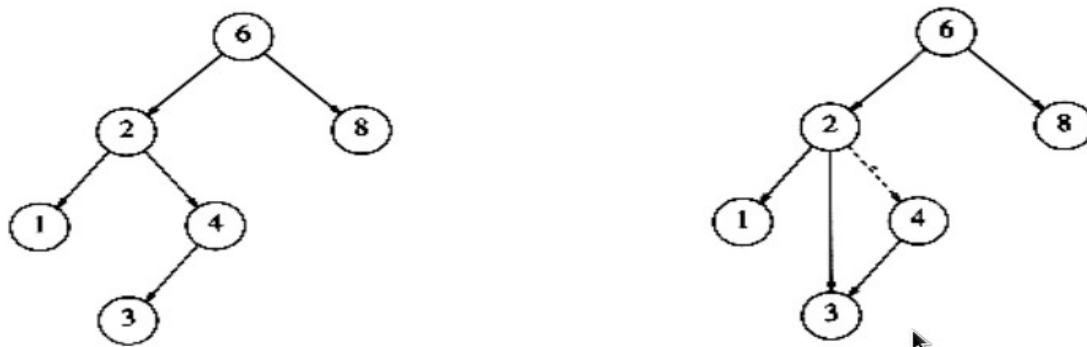
Elemen root : T->Element = 6

Elemen dengan nilai kunci 3 ditemukan

Elemen (nilai kunci) minimum = 1
Elemen (nilai kunci) maksimum = 8

```

δ) Operasi terakhir adalah penghapusan. Contoh yang diberikan di bawah ini adalah penghapusan elemen (nilai kunci) yang berada pada simpul dengan 2 anak.



ε) Edit fail klien\_binary\_search\_tree.c dan tambahkan kode berikut:

```

31 //menghapus simpul dengan nilai elemen 2
32 T = Delete(2, T);
33
34 if((P = Find(2,T)) == NULL)
35     printf("\nSimpul dengan elemen 2 tidak ditemukan\n");
36 else
37     printf("Simpul dengan elemen %d ditemukan\n", Retrieve(P));

```

φ) Kompilasi dan jalankan program sebagai berikut:

```

root@kodi:/home/c_alg_sd/mgl2_sesi3# gcc -Wall binary_search_tree.c klien_binary_search_tree.c -o klien_bst
root@kodi:/home/c_alg_sd/mgl2_sesi3# ./klien_bst

Elemen root : T->Element = 6

Elemen dengan nilai kunci 3 ditemukan

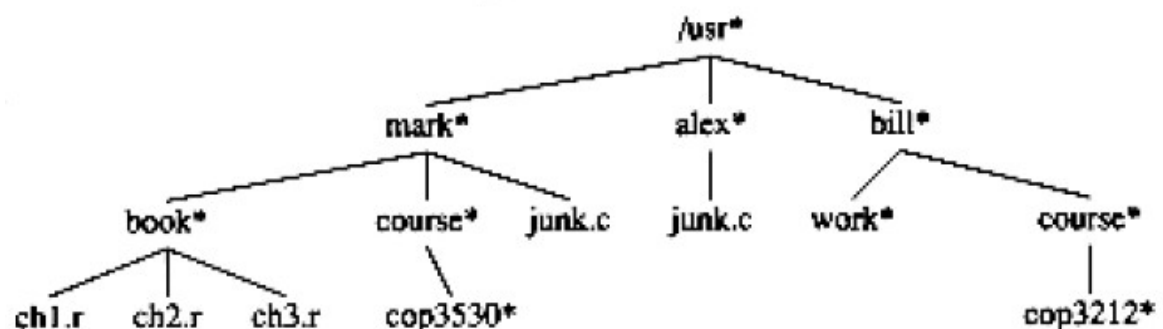
Elemen (nilai kunci) minimum = 1
Elemen (nilai kunci) maksimum = 8

Simpul dengan elemen 2 tidak ditemukan

```

### 3. Tugas Pemrograman:

- Modifikasi fungsi `FindMin` sehingga fungsi tersebut menjadi non-rekursif.
- Modifikasi fungsi `FindMax` sehingga menjadi fungsi rekursif.
- Modifikasi fungsi `Find` sehingga menjadi fungsi non-rekursif.
- Tambahkan fungsi `GetDepth` untuk menghitung kedalaman pohon pencarian biner
- Buatlah program untuk mengimplementasikan pohon umum (*generic tree*) yang efisien untuk menerapkan struktur direktori berikut yang diambil dari literature [2] bab 4, subbab 4.1.2.



### **Setoran**

1. **Lembar jawaban dikumpulkan kepada TA .**  
Lembar jawaban berisi jawaban terhadap soal-soal pada bagian Ulasan dan Pendalaman.
2. Kode program untuk Tugas Pemrograman disetor melalui link setoran moodle.  
Tambahkan komentar untuk menulis anggota kelompok pada bagian paling atas kode program.
3. Kode program c untuk BinarySearchTree ADT:
  - Antarmuka : `binary_search_tree.h` dan `fatal.h`
  - implementasi : `binary_search_tree.c`
  - program klien : `klien_binary_search_tree.c`