5) Sökträd och söktabeller

Snabba upp sökningar

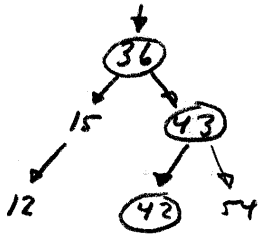
(Ex) Sök efter nyckeln 42 i mängden
36, 15, 43, 12, 42, 54.

a) Linjär sökning i lista

Lista → 36, 15, 43, 12, 42, 54

Hittar nyckeln efter 5 jämförelser

b) Binärt sökträd



Hittar nyckeln efter
3 jämförelser

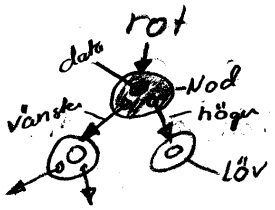
c) Söktabell (hashtabell)

hashtab [2] → 42 → 12
[3] → 43
[4] → 54
[5] → 15
[6] → 36

Hittar nyckeln efter
2 jämförelser

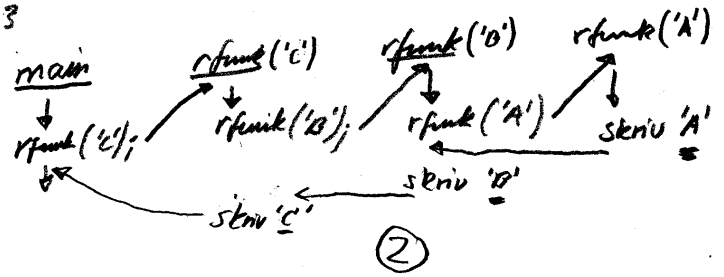
Binärt träd – är tomt eller består av en nod med data och ett vänsterträd och ett högerträd

Ett binärt träd är en rekursiv datastruktur dvs. den upprepar sig själv.

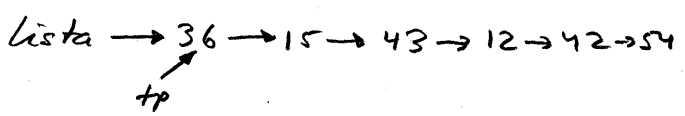


(Ex) En rekursiv funktion anropar sig själv

```
void rfunk(char ch)
{
    if (ch > 'A')
    {  rfunk(ch-1); // Rekursivt anrop
    }
    putchar(ch);
}
void main()
{
   rfunk('C');
}
```



(Ex) Skriv en linjär sökfunktion för en lista



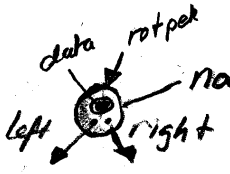
a) iterativ

```
Linktyp *isok(Linktyp *lista, int key)
{
    Linktyp *tp = lista;
    while (tp != NULL && key != tp->data)
    {
        tp = tp->next;
    }
    return tp;
}
```

b) rekursiv – en lista är antingen tom eller består av en första länk och en återstående lista

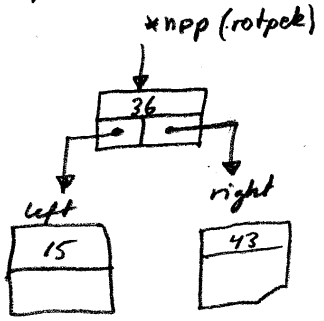
```
Linktyp *rsok(Linktyp *lista, int key)
{
    Linktyp *tp = lista;
    if (tp == NULL)
    {
        return NULL;
    }
    else if (key == tp->data)
    {
        return tp;
    }
    else
    {
        return rsok(tp->next, key); //Rekursivt anrop
    }
}
```

(Ex) Sätt in talen 36, 15, 43, 12, 42, 53 i ett binärt sökträd och sök efter inläst nyckel i trädet

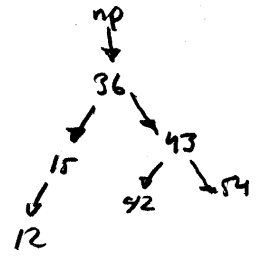



```
typedef struct nod
{
    int data;
    struct nod *left, *right;
} nodtyp;

void intotree (nodtyp **npp, int tal)
{
   if (*npp == NULL)
   {
      *npp = malloc (sizeof(nodtyp));
      (*npp)->data = tal;
      (*npp)->left = (*npp)->right = NULL;
   }
   else if (tal < (*npp)->data)
   {
      intotree (&(*npp)->left, tal);
   }
   else
   {
      intotree (&(*npp)->right, tal);
   }
}
```



```
void showtree(nodtyp *np)
{
    if (np != NULL)
    {
        showtree(np->left);
        printf("%d ", np->data);
        showtree(np->right);
    }
}
```



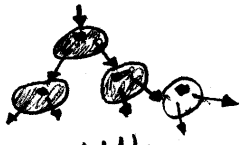
Skriver ut
12 15 36 42 43 54

```
nodtyp *search_tree(nodtyp *np, int key)
{
    if (np == NULL)
    {
        return NULL;
    }
    else if (np->data == key)
    {
        return np;
    }
    else if (np->data > key)
    {
        return searchtree(np->right, key);
    }
    else
    {
        return searchtree(np->left, key);
    }
}
```

```
void main()
{
   int i, v[6]={36,15,43,12,42,54}; nyckel;
   nodtyp *rotpek = NULL; //OBS!

   for(i=0; i<6; i++)
   {   intotree(&rotpek, v[i]);
   }
   showtree(rotpek);
   printf("Ge nyckel : ");
   scanf("%d", &nyckel);
   if (seachtree(rotpek, nyckel)
   {  printf("Nyckel finns! ");
   }
   else
   {  printf("Nyckel finns ej! ");
}
```

(Ex) Funktion som räknar noderna i ett binärt träd



```
int noder(nodtyp *np)
{ if (np == NULL) return 0;
  else return 1 + noder(np->left) + noder(np->right);
}
```

Söktabell (hashtabell, hash bedyds finhackad)

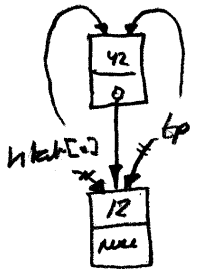
(Ex) Sätt in heltalen 36, 15, 43, 12, 42, 54 i en hashtabell med hashfunktionen tal % 10 och kollisionshantering med stackar. Sök sedan efter inläst nyckel.

htab[0] NULL
htab[1] NULL
htab[2] → 42 → 12
htab[3] → 43
htab[4] → 54
htab[5] → 15
htab[6] → 36
htab[7] NULL
htab[8] NULL
htab[9] NULL

```
typedef
struct link
{  int data;
   struct link *next;
} linktyp;

linktyp *htab[10];
```

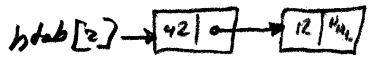
```
void intolinkhash(linktyp *htab[], int tal)
{
   int i;
   linktyp *tp;
   i = tal % 10;
   tp = malloc(sizeof(linktyp));
   tp->data = tal;
   tp->next = htab[i];
   htab[i] = tp;
}
```



```
linktyp *searchlinkhash(linktyp *htab[], int key)
{
    int i;
    linktyp *tp;

    i = key % 10;
    tp = htab[i];
    while( tp != NULL && tp->data != key)
    {
        tp = tp->next;
    }
    return tp;
}

void main()
{
    int i, v[6] = {36, 15, 43, 12, 42, 54}, nyckel;
    linktyp *hashtab[10];
    for(i=0; i<10; i++)
    {
        hashtab[i] = NULL;
    }
    for(i=0; i<6; i++)
    {
        intolinkhash(hashtab, v[i]);
    }
```



```
    printf("Ge nyckel: ");
    scanf("%d", &nyckel);
    if (searchlinkhash(hashtab, nyckel) == NULL)
    {  printf("Nyckeln finns ej! ");
    }
    else
    {
       printf("Nyckel finns!");
    }
}
```

(Ex) Samma som ovan men kollisionshantering med öppen adressering med hoppfunktionen hopp=1 först och sedan hopp += 2.

htab [0]
[1] → 42
[2] → 12
[3] → 43
[4] → 54
[5] → 15
[6] → 36
[7]
[8]
[9]

(42 kollar först [2] upptaget. Med hopp=1 kollar [3] upptaget. Med hopp=3 kollar [6] upptaget. Med hopp=5 kollar [1] ledigt och 42 instoppas)

```
void main()
{
    int i, V[6] = {36, 15, 43, 12, 42, 54};
    int nyckel, *htab[10];

    for(i=0; i<10; i++)
    {
        htab[i] = NULL;
    }

    for(i=0; i<6; i++)
    {
        intoopenhash(htab, V[i]);
    }
    printf("Ge nyckel: ");
    scanf("%d", &nyckel);

    if (searchopenhash(htab, nyckel) == NULL)
    {
        printf("Nyckeln finns ej!");
    }
    else
    {
        printf("Nyckel finns!");
    }
}
```

Hemuppgift: Skriv en rekursiv funktion som räknar antalet Löv i ett binärt träd. Löv är noder som saknar vänster- och höger träd dvs. sådana noder som har left och right NULL. Skriv också ett huvudprogram som läser in ett antal heltal och stoppar in dem i ett binärt träd och räknar antalet Löv i trädet