

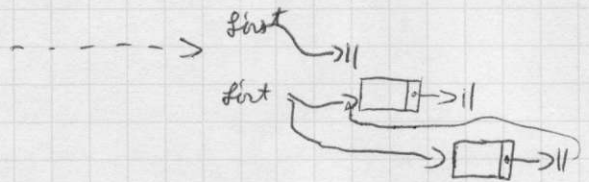
Erzeugen einer ^{linearen} Liste



```

struct list_element {
    void *data;
    struct list_element *fw;
};
list_element *first = NULL;

```



```
list_element *slist_new(void *data) {
```

```
    list_element *new = malloc(sizeof(list_element));
```

```
    if (new == NULL) {
```

```
        fprintf(stderr, "Nicht genug Speicher...\n");
```

```
        exit(1);
```

```
    }
```

```
    new->data = data;
```

```
    new->fw = NULL;
```

```
    return new;
```

```
}
```



```
slist_new
```

```
void slist_append(void *data) {
```

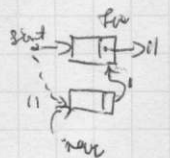
```
    list_element *new =
```

```
        slist_new(data);
```

```
    new->fw = first;
```

```
    first = new;
```

```
}
```



```
void list_all(void) {
```

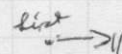
```
    list_element *runner = first;
```

```
    while (runner != NULL) {
```

```
        printf("%s\n", runner->data);
```

```
        runner = runner->fw;
```

```
    }
```



```
void delete_all (void) {
```

```
    list = delete * runner;
```

```
    for (runner = first; runner != NULL; runner = runner->next)
        delete (runner);
```

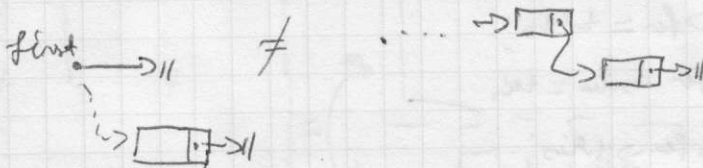
```
void delete_all (list_element * act) {
```

```
    if (act == NULL)
```

```
        return;
```

```
    delete (act);
```

```
    delete_all (act->next);
```



```
void delete_append (void * data) {
```

```
    list_element * new = delete_new (data);
```

```
    list_element * last
```

```
    if (last == NULL) {
```

```
        first = new;
```

```
    } return;
```

```
    last = first;
```

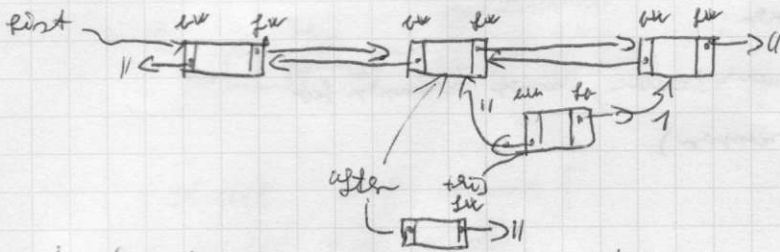
```
    while (last->next != NULL)
```

```
        last = last->next;
```

```
    last->next = new;
```

```
}
```

Duplein edvált lista



```
void list_insert (dlist_element *after,
                 dlist_element *this);
```

```
this->next = after->next;
```

```
this->prev = after;
```

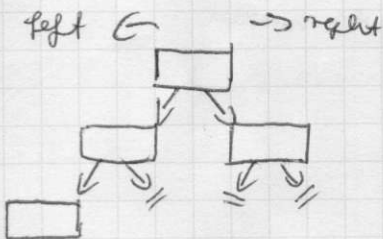
```
if (after->next == NULL)
    after->next = this;
else
    after->next->prev = this;
after->next->next = this;
```

```
after->next->next = this;
```

```
if (after->next != NULL)
```

```
after->next->prev = this;
```

Bináris keresés



```
struct node {
```

```
    void *data;
```

```
    struct node *left;
```

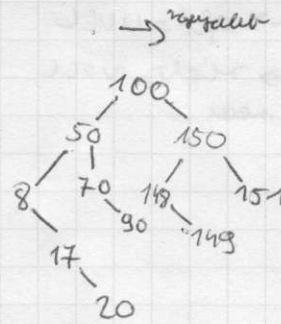
```
    struct node *right;
```

```
int key;
```

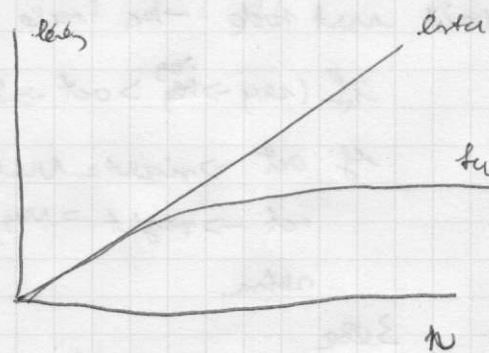
```
}
```


100 ✓
50 ✓
8
17
20
150
148
149
70
159
90

← previous



N	Lista	for
1	1	1
2	2	2
3	3	2
4	4	3
⋮	⋮	⋮
7	7	3
⋮	⋮	⋮
15	15	4



```

typedef struct tnode {
    void *data;
    int key;
    struct tnode *left;
    struct tnode *right;
} tnode;

tnode *root = NULL;
  
```

```

tnode * new_node (int key, void *data) {
    tnode * new = malloc (sizeof (tnode));
    if (new == NULL) {
        fprintf (stderr, "Out of memory");
        exit (1);
    }
    new->key = key;
    new->data = data;
}
  
```

```

    new → left = NULL
    new → right = NULL
    return new
}

```



```

void insert-new-node(int key, void * data) {
    node * new = new-node(key, data);
    if (root == NULL) {
        root = new;
        return;
    }
    insert-node-rec(root, new);
}

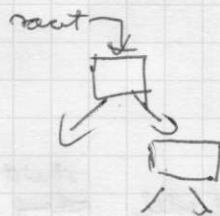
```

~~void~~

```

void insert-node-rec(node * act, node * new) {
    if (new->key > act->key)
        if (act->right == NULL) {
            act->right = new;
            return;
        }
}

```



```

else
    insert-node(act->left, new);
}

```

```

if (new->key < act->key)
    if (act->left == NULL) {
        act->left = new;
        return;
    }
}

```

```

insert-node(act->left, new);
}

```

```

if (new->key == act->key) {
    printf("Error: key already exists\n");
    exit(1);
}
}

```

}

90.9%, negative order

node * node -> loopup (start key) {

return node loopup - rec (root, key)

}

node * node loopup - rec (node * root, int key) {

if (root == NULL)

return NULL;

if (root -> key == key)

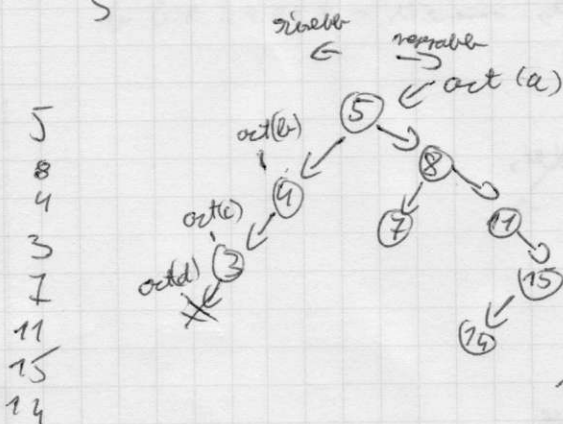
return root

if (~~key > root -> key~~)

return node - loopup - rec (root -> right, key);

else return node loopup - rec (root -> left, key);

}



minimizes search cost
3, 4, 5, 7, 8, 11, 14, 15,

void Print - all (void) {

print - all - rec (root);

}

void Print - all - rec (node * root) {

if (root == NULL)

return;

Print - all - rec (root -> left);

Print f (%d root -> key)

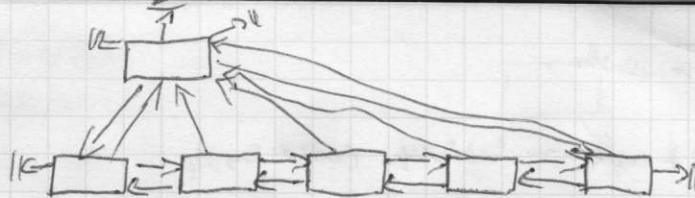
Print all - rec (root -> right);

}

recursion
order
search

1
2
3
4

W



type def struct node {

...
 struct node * first-child;
 struct node * last-child;
 struct node * next-sibling;
 struct node * prev-sibling;
 struct node * parent;

}

Algorithm: rekursiv eldeltett rek

Funkció: rekursív, rekursív

int odd (a, b) {
 ...
 return a+b;
}

c = odd(2, 2) ^{rekursív} ~~rekursív~~

Enter with parameter data

int z = 0
 int n(a, b) {
 z = a+1
 return z+b
}
 n = n(z, 2)

- 1, Kerketer is
- 2,
- 3, diese Schichten f6hlt erkl6rt techn. Sch6pfer oder f6hlt es zu bezeichnen u sich
- 4 neu auf neu wird
- 5 this f6cher is
- 6
- 7

6, teile

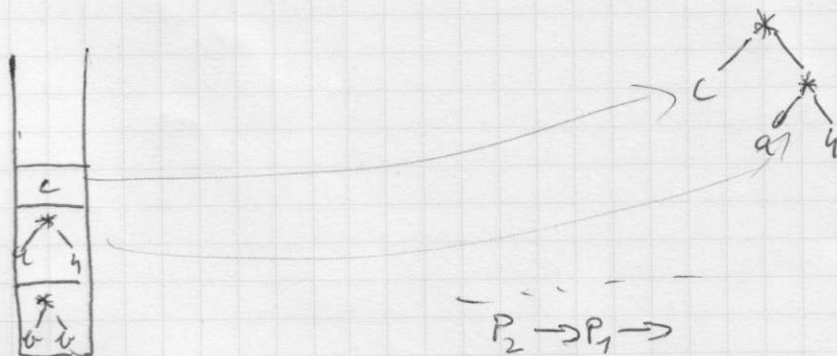
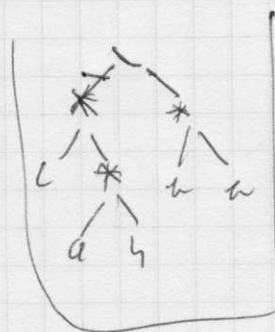
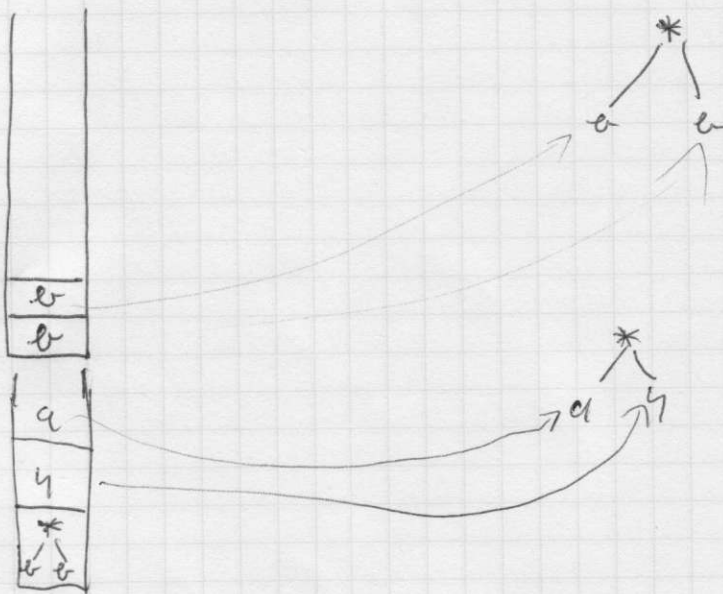
7, f6hlt

8/ Hoch f6hlt neu idell

9/ H6her als ~~idell~~

Prefix or mit ar operator cell' h6r

Postfix b b * h a * c * -



$$P_2 \rightarrow P_1 \rightarrow$$

$$E * P_1$$

$$V_1 * V_2 + V_3$$

$$P_2 \rightarrow P_1 + P_2 \Rightarrow$$

$$E * P_1 + P_2 \rightarrow$$

$$V_1 * V_2 + V_3$$