İkili Ağaçtan → Çift bağlı Listeye

```
struct node {
 int data;
struct node right ;left ; } ;
typedef struct node *NODE ;
void treetoList (Node root, node &prew,
                    node &head ) {
 if (!root ) return 0 ;
 treetoList (root →left ,prew,head) ;
 root →left =prev ;
 if (prev) { prev→right =root ;
 else head =root ;
 Node right =root →right ;
 head → left =root ;
 root →right =head ;
 prev =root ;
 treetoList (right, prev,head ) }
Node treetoList (Node root)
 { Node .prev =NULL ;
   Node . head=NULL ;
treetoList (root ,prev, heat ) ; }
 return head ; }
```

Scanned by CamScanner

Kuyruktan → İkili Ağaca Aktarma

```
Struct Kuyruk {
 int info;
struct kuyruk *next ; };
typedef struct kuyruk *KUYRUKPTR ;
struct tree {
 int info;
structtree *left;
structtree *right; };
typedef struct tree *TREENODEPTR;
void kuyruktotree (KUYRUKPTR *bas ,
KUYRUKPTR *son, TREENODEPTR *treePtr)
{ Kuyruk currentPtr ;
   int deger ;
   currentPtr = *bas;
  while (currentPtr != NULL)
{ deger = remove(bas, son);
  insertTree (tree ptr , deger) ;
 currentPtr = *bas; } }
```

Scanned by CamScanner

Sayıları Ağaca Yerleş. Prog.

```
Struct nodetype {
int info;
struct nodetype *left;
struct nodetype *right; }
typedef struct nodetyp. *NODEPTR;
main()
NODEPTR p,tree;
NODEPTR p,q;
int number;
scanf("%d",&number);
ptree=maketree(number);
while(scanf("%d",&number)!=EOF){
p=q=ptree;
while(number!=p->info && q!=NULL){
p=q;
if(number<p->info)
  q=p->left; else
  q=p->right;
if (number==p->info)
{ printf("%d" is a duplicate \n",number);
else if (number(p->info)
  setleft (p,number);
else
  setright(p,number); } }
```

Scanned by CamScanner

yığın(stack)tan → Tek bağlı list. aktarma

```
struct node {
   int info;
struct node *next; };
typedef struct stack {
 int top;
 int items[10]; }stack;
 typedef struct node *NODEPTR;
 void insert (NODEPTR p, int x)
{ NODEPTR q;
 if (p = = NULL()) ⊘
{printf("gecersiz eklene"); exit (0); }
q = getnode();
q → info = x;
q → next = p → next;
p → next = q ; }
```

Scanned by CamScanner

İkili Ağaçtan → Yığına (stack) aktarma

```
typedef struct node {
 int data;
struct node *left *right; }node;
void mirror (node *root ) {
 if (root = =NULL )return;
  node *temp;
  node * q[100]={NULL};
  int i, j =0;
while (root)
{ temp=root ->left;
 root->left =root ->right;
 root ->right =temp;
if (root ->left)
q [i++] =root ->left;
if (root ->right)
q [i ++] =root ->right;
root =q [j++ ]; }}
```

Scanned by CamScanner

Ağacın Tüm Düğümlerindeki eleman sayısı toplamı

```
int main();{
int Tara(NODEPTR tree){
int toplam=0; if(tree != NULL){
Tara(tree->left);
Tara(tree->right);
toplam++; } }
return toplam }
int main(){NODEPTR tree;
int eleman;
eleman = Tara(tree);
printf("eleman sayisi = "%d" eleman);
return 0; }
```

İkili Ağacın max elemanını Bulan Prog

```
struct tree{int data; struct tree *right, left;}
*ptree
int maxsayi(ptree *ptree, int max){
if(ptree==null)
 return max;
if(ptree->data>max)
 max = ptree->data;
 maxsayi(ptree->left,max);
 maxsayi(ptree->right,max);
return max; }
```

Scanned by CamScanner

İkili Ağacın Top. Eleman sayısını Bulan Prg.

```
struct tree {int data ; struct tree *right, left; }
*ptree
int toplam ;
int toplamsayi(ptree *ptree) {
toplamsayi(ptree->left);
toplam ++;
toplam sayi(ptree ->right); }
```

İkili ağacın sağ ve sol kollarının sayısını ayrı ayrı bulan prog

```
struct tree {int data; struct tree *right, left; }
*ptree
int sol ; int sag;
int derinlik (ptree *ptree) {
if (ptree = =null)
 return(0);
int right = derinlik (ptree->right);
int left = derinlik (ptree ->left);
 sol = left +1
 sag = right + 1; }
```

Scanned by CamScanner

Matrisin satırını Dizide Sıralama

```
int main(){
int i=0,J=0,k=0,temp,min;
int matris[N][N]={{5,3,2},{4,2,5},{6,1,2}};
int d[N];
printf("Mat. boyutu: %d\n kacıncı satırı sıralamak istiyorsak:",N);
scanf("%d",&k);
for(i=0;i<N;i++)
 d[i]=matris[k-1][i];
for(i=0;i<N-1;i++){
 min=i;
for(J=i+1;J<N;J++)
  if(d[J]<d[min])
  min=J;
  if(min!=i){
temp=d[min];
 d[min]=d[i];
 d[i]=temp; }}
printf("\n\n sıralanmış dizi:");
for(i=0;i<N;i++)
 printf("%d",d[i]);
 printf(" ");
 return 0; }
```

Scanned by CamScanner

Tek bagli Listeyi Tersine Ceviren

```
struct node{int info; struct node *next}*s
void cevir(){
struct node *ptr1,*ptr2, *ptr3 ;
 if (s= =NULL)
  printf ("bos");
 if (s->next= =null)
 printf ("tek elemanli")
ptr1 =s;
ptr2 =ptr1 ->next;
ptr3 =ptr2 ->next;
ptr1 ->next =null;
ptr2 ->next =ptr1;
while(ptr3! =NULL){
ptr1 =ptr2 ; ptr2 =ptr3;
ptr3=ptr3->next;
ptr2->next=ptr1; }
s=ptr2 ; }
```

Scanned by CamScanner

İkili Ağaçtan → Tek bağlı Listeye

```
struct tekbaglı {
 int info;
struct tekbaglı *next; };
 typedef struct tekbağlı *TEKBAĞLIPTR;
void treetoList (AGACPTR trePtr,
                 TEKBAGLIPTR *listPtr )
{if(treePtr != NULL)
 {tekbağlıyaekle(listPtr,treePtr->info);
  treetolist(treePtr->left,listPtr);
  treetolist(treePtr->right,listPtr);}}
```

Scanned by CamScanner

Agactan kuyruga aktarma

```
Struct tree {
 int info;
struct tree *left, *right; };
typedef structtree *TREEPTR;
Struct kuyruk { int info;
Struct kuyruk *bas, *son, *next; };
typedef struct kuyruk *KUYRUKPTR;
int agacelemansayisi (TREEPTR *treePtr){
if (*treePtr ! =NULL)
   return (1 +agacelemansayisi (*treePtr →left)+
          agacelemon sayisi (*treePtr →right);
  else return 1; }
int kuyrugaEkle(KUYRUKPTR *bas, *son, int deger){
Kuyruk newPtr;
newPtr= (KUYRUKPTR)malloc (sizeof(structkuyruk))
newPtr →info = deger;
newPtr → next =NULL;
 if (*bas = =NULL)
    *bas =newPtr);
 else (*son →next)=newPtr;                   *treePtr){
     *son =newPtr; }
int tree ToQueue (KUYRUKPTR *bas *son, *AGACPTR
while (*treePtr ! =NULL) {
KuyrugaEkle(& *bas, & *son, treePtr →info)); }}
```

Scanned by CamScanner

```
int main() {
 AGACPTR *treePtr;
 KUYRUKPTR *kuyrukPtr;
 int N;
 N = agacelemansayisi(& *treePtr);
 for (int i=0; i<N, i++){
tree to Queue (&bas, &son, treePtr->info);
 *bas = *bas->next;
 *treePtr = *treePtr->next;
 }}
```

Scanned by CamScanner