Redovi

```
typedef struct {
        tip polje[MAXRED];
        int ulaz, izlaz;
} Red;
```

Skidanje elemenata iz reda realiziranog cirkularnim poljem

```
int skini (tip *element, Red *red) {

 if (red->ulaz == red->izlaz) return 0;

 red->izlaz++;

 red->izlaz %= n;

 *element = red->polje[red->izlaz];

 return 1;

}
```

Dodavanje elemenata u red(poljem)

```
int dodaj (tip element, Red *red) {

 if ((red->ulaz+1) % n == red->izlaz) return 0;

 red->ulaz++;

 red->ulaz %= n;

 red->polje[red->ulaz] = element;

 return 1;

}
```

-Jedan element je uvijek prazan
-Cirkularnost polja ostvarena %-om
-Prazan red ulaz==izlaz
-Puni red (ulaz+1) % MAXRED ==izlaz

Red listom

```
typedef struct at atom;
struct at {
        int element;
        struct at *sljed;
};
typedef struct {
        atom *ulaz, *izlaz;
} Red;
```

Dodavanje elementa u red realiziran listom

```
int DodajURed (int element, Red *red) {

atom *novi;

if (novi = malloc (sizeof (atom))) {

        novi->element = element;

        novi->sljed = NULL;

        if (red->izlaz == NULL) red->izlaz = novi;      // ako je red bio prazan

                else (red->ulaz)->sljed = novi;         // inace, stavi na kraj

        red->ulaz = novi;                               // zapamti zadnjeg

        return 1;

}

return 0;

}
```

Skidanje elementa s reda

```
int SkiniIzReda (int *element, Red *red) {

 atom *stari;

 if (red->izlaz) {              // ako red nije prazan

  *element = red->izlaz->element;           // element koji se skida

  stari = red->izlaz;               // zapamti trenutni izlaz

  red->izlaz = red->izlaz->sljed;     // novi izlaz

  free (stari);                             // oslobodi memoriju skinutog

  if (red->izlaz == NULL) red->ulaz = NULL; // prazan red

 return 1;

 }

 return 0;}
```

Povezane liste

Pretraživanje liste

```
// trazenje elementa liste

// vraca pokazivac na trazeni element ili NULL ako ga ne nadje

atom *trazi (atom *glava, int element) {

 atom *p;

 for (p = glava; p != NULL; p = p->sljed) {

          if (p ->element == element) return p;

 }

 return NULL;

}
```

Dodavanje na početak liste

```
int dodaj (atom **glavap, int element) {

 atom *novi;

 if ((novi = (atom *) malloc(sizeof(atom))) == NULL)

                  return 0;

 novi->element = element;

 if (*glavap == NULL || (*glavap)->element >= element) {

  // Dodavanje na pocetak liste

   novi->sljed = *glavap;

   *glavap = novi;

 }....
```

Dodavanje unutar liste

```
int dodaj (atom **glavap, int element) {

 atom *novi, *p;

 if ((novi = (atom *) malloc(sizeof(atom))) == NULL)

                  return 0;

 novi->element = element;
```

```
// ako element dodajemo unutar liste

 for (p = *glavap; p->sljed &&(p->sljed)->element < element; p = p->sljed);

                    novi->sljed = p->sljed;

                    p->sljed = novi;

...
```

Brisanje elementa s početka liste

```
int brisi (atom **glavap, int elem) {

 atom *p;

 for (; *glavap && (*glavap)->elem != elem; glavap = &((*glavap)->sljed));

 if (*glavap) {

          p = *glavap;

          *glavap = (*glavap)->sljed;

          free (p);

     return 1;

 } else return 0; }
```

Brisanje elementa iz sredine liste

```
int brisi (atom **glavap, int elem) {

 atom *p;

 for (; *glavap && (*glavap)->elem != elem; glavap = &((*glavap)->sljed));

 if (*glavap) {

          p = *glavap;

          *glavap = (*glavap)->sljed;

          free (p);

     return 1;

 } else return 0;

}
```

Stabla

Stupanj stabla- broj podstabala nekog korijena
Korijen- čvoru na 'vrhu' stabla
Stupanj stabla je maksimalni stupanj od svih čvorova tog stabla
Razina (*level*) nekog čvora određuje se iz definicije da je korijen razine 1, a da su razine djece nekog čvora razine *k* jednaki *k*+1
Dubina (*depth*) stabla je jednaka maksimalnoj razini nekog čvora u stablu

Binarno stablo

maksimalni broj čvorova na ***k***-toj razini jednak je $\mathbf{2^{k-1}}$
maksimalni broj čvorova binarnog stabla dubine *k* jednak je 2^k -1 za $k>0$

Binarna stabla poljem:

pravila za potpuno binarno stablo s *n* čvorova, za *i*-ti čvor su:

roditelj(*i*)=$\lfloor i/2 \rfloor$ za $i \neq 1$; kada je *i*=1, čvor *i* je korijen pa nema roditelja
lijevo_dijete(*i*)=2**i* ako je $2*i \leq n$; kad je $2*i>n$ čvor *i* nema lijevog djeteta
desno_dijete(*i*)=2**i*+1 ako je $2*i+1 \leq n$; kad je $2*i+1>n$ čvor *i* nema desnog djeteta

Binarna stabla dinamičkom strukturom:

```
struct cvor{
  tip podatak;
  struct cvor *lijevo_dijete;
  struct cvor *desno_dijete;
/* ako treba: */
  struct cvor *roditelj;
};
```

Dodavanje elemenata u stablo:

```
struct cvor* dodaj(struct cvor* cvor, tip elem) {
  if (cvor == NULL) {
    return(NoviCvor(elem));
  }
  else {
    if (elem <= cvor->podatak)
            cvor->lijevo = dodaj(cvor->lijevo, elem);
    else
       cvor->desno = dodaj(cvor->desno, elem);
  return(cvor); }
}
```

Stvaranje novog čvora:

```
struct cvor* NoviCvor(int elem) {
  struct cvor* novi =
                    (cvor *) malloc(sizeof(struct cvor));
  novi->podatak = elem;
  novi->lijevo = NULL;
  novi->desno = NULL;

  return(novi);
}
```

Pretraživanje stabla:

```
int trazi (struct cvor* cvor, int trazeno) {
  if (cvor == NULL) {
    return 0;
  }
  else {
    if (trazeno == cvor->podatak) return 1;
    else {
      if (trazeno < cvor->podatak)
                    return(trazi(cvor->lijevo, trazeno));
      else return(trazi(cvor->desno, trazeno));
    }
  }
}
```

postoje 3 standardna načina obilaska stabla kojima se osigurava da je svaki čvor bio "posjećen"

inorder: lijevo podstablo → korijen → desno podstablo

preorder: korijen → lijevo podstablo → desno podstablo

postorder: lijevo podstablo → desno podstablo → korijen

Gomila

Gomila je potpuno binarno stablo gdje se čvorovi mogu uspoređivati nekom uređajnom relacijom (npr. <=) i gdje je bilo koji čvor u smislu te relacije veći ili jednak od svoje djece (ako postoje).