Apstraktni tip podataka `List`

`elementtype` . . . bilo koji tip.
`List` . . . podatak tipa `List` je konačni niz (ne nužno različitih) podataka tipa `elementtype`.
`position` . . . podatak ovog tipa služi za identificiranje elementa u listi, dakle za zadavanje pozicije u listi. Smatramo da su u listi $(a_1, a_2, \ldots, a_n)$ definirane pozicije koje odgovaraju prvom, drugom, ..., n-tom elementu, a također i pozicija na kraju liste (neposredno iza n-tog elementa).
`position LiEnd(List L)` . . . funkcija koja vraća poziciju na kraju liste `L`.
`position LiMakeNull(List *Lp)` . . . funkcija pretvara listu `*Lp` u praznu listu, i vraća poziciju `LiEnd(*Lp)`.
`void LiInsert(elementtype x, position p, List *Lp)` . . . funkcija ubacuje podatak `x` na poziciju `p` u listu `*Lp`. Ako je `*Lp` oblika $(a_1, a_2, \ldots, a_n)$, tada `*Lp` postaje $(a_1, a_2, \ldots, a_{p-1}, x, a_p, \ldots, a_n)$. Ako je `p==LiEnd(*Lp)` tada `*Lp` postaje $(a_1, a_2, \ldots, a_n, x)$. Ako u `*Lp` ne postoji pozicija `p`, rezultat je nedefiniran.
`void LiDelete(position p, List *Lp)` . . . funkcija izbacuje element na poziciji `p` iz liste `*Lp`. Ako je lista `*Lp` oblika $(a_1, a_2, \ldots, a_n)$ tada `*Lp` postaje $(a_1, a_2, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$. Rezultat nije definiran ako `*Lp` nema pozicije `p` ili ako je `p==LiEnd(*Lp)`.
`position LiFirst(List L)` . . . funkcija vraća prvu poziciju u listi `L`. Ako je `L` prazna, vraća se `LiEnd(L)`.
`position LiNext(position p, List L), position LiPrevious(position p, List L)` . . . funkcije koje vraćaju poziciju iza odnosno ispred `p` u listi `L`. Ako je `p` zadnja pozicija u `L`, tada je `LiNext(p,L)==LiEnd(L)`. `LiNext()` je nedefinirana za `p==LiEnd(L)`. `LiPrevious()` je nedefinirana za `p==LiFirst(L)`. Obje funkcije su nedefinirane ako `L` nema pozicije `p`.
`elementtype LiRetrieve(position p, List L)` . . . funkcija vraća element na poziciji `p` u listi `L`. Rezultat je nedefiniran ako je `p==LiEnd(L)` ili ako `L` nema pozicije `p`.

Apstraktni tip podataka `Stack`

`elementtype` . . . bilo koji tip.
`Stack` . . . podatak tipa `Stack` je konačan niz podataka tipa `elementtype`.
`void StMakeNull(Stack *Sp)` . . . funkcija pretvara stog `*Sp` u prazni stog.
`int StEmpty(Stack S)` . . . funkcija koja vraća 1 ako je `S` prazan stog. Inače vraća 0.
`void StPush(elementtype x, Stack *Sp)` . . . funkcija ubacuje element `x` na vrh stoga `*Sp`.
`void StPop(Stack *Sp)` . . . funkcija izbacuje element s vrha stoga `*Sp`.
`elementtype StTop(Stack S)` . . . funkcija vraća element koji je na vrhu stoga `S` (stog ostaje nepromijenjen).

Apstraktni tip podataka `Queue`

`elementtype` . . . bilo koji tip.
`Queue` . . . podatak tipa `Queue` je konačan niz podataka tipa `elementtype`.
`void QuMakeNull(Queue *Qp)` . . . funkcija pretvara red `*Qp` u prazan red.
`int QuEmpty(Queue Q)` . . . funkcija vraća 1 ako je `Q` prazan red, inače 0.
`void QuEnqueue(elementtype x, Queue *Qp)` . . . funkcija ubacuje element `x` na začelje reda `*Qp`.
`void QuDequeue(Queue *Qp)` . . . funkcija izbacuje element sa čela reda `*Qp`.
`elementtype QuFront(Queue Q)` . . . funkcija vraća element na čelu reda `Q` (red ostaje nepromjenjen).

Apstraktni tip podataka `Tree`

`node` . . . bilo koji tip (imena čvorova). U skupu `node` uočavamo jedan poseban element `LAMBDA` (koji služi kao ime nepostojećeg čvora).
`labeltype` . . . bilo koji tip (oznake čvorova).
`Tree` . . . podatak tipa `Tree` je (uređeno) stablo čiji čvorovi su podaci tipa `node` (međusobno različiti i različiti od `LAMBDA`). Svakom čvoru je kao oznaka pridružen podatak tipa `labeltype`.
`node TrMakeRoot(labeltype l, Tree *Tp)` . . . funkcija pretvara stablo `*Tp` u stablo koje se sastoji samo od korijena s oznakom `l`. Vraća ime korijena.
`node TrInsertChild(labeltype l, node i, Tree *Tp)` . . . funkcija u stablo `*Tp` ubacuje novi čvor s oznakom `l`, tako da on bude prvo po redu dijete čvora `i`. Funkcija vraća ime novog čvora. Nije definirana ako `i` ne pripada `*Tp`.
`node TrInsertSibling(labeltype l, node i, Tree *Tp)` . . . funkcija u stablo `*Tp` ubacuje novi čvor s oznakom `l`, tako da on bude idući po redu brat čvora `i`. Funkcija vraća ime novog čvora. Nije definirana ako je `i` korijen ili ako `i` ne pripada `*Tp`.
`void TrDelete(node i, Tree *Tp)` . . . funkcija izbacuje list `i` iz stabla `*Tp`. Nije definirana ako je `i` korijen, ili ako `i` ne pripada `*Tp` ili ako `i` ima djece.
`node TrRoot(Tree T)` . . . funkcija vraća ime korijena stabla `T`.
`node TrFirstChild(node i, Tree T)` . . . funkcija vraća prvo po redu dijete čvora `i` u stablu `T`. Ako je `i` list, vraća `LAMBDA`. Nije definirana ako `i` ne pripada `T`.
`node TrnextSibling(node i, Tree T)` . . . funkcija vraća idućeg po redu brata čvora `i` u stablu `T`. Ako je `i` zadnji brat, tada vraća `LAMBDA`. Nije definirana ako `i` ne pripada `T`.
`node TrParent(node i, Tree T)` . . . funkcija vraća roditelja čvora `i` u stablu `T`. Ako je `i` korijen, tada vraća `LAMBDA`. Nije definirana ako `i` ne pripada `T`.
`labeltype TrLabel(node i, Tree T)` . . . funkcija vraća oznaku čvora `i` u stablu `T`. Nije definirana ako `i` ne pripada `T`.
`void TrChangeLabel(labeltype l, node i, Tree *Tp)` . . . funkcija mijenja oznaku čvora `i` u stablu `*Tp`, tako da ona postane `l`. Nedefinirana ako `i` ne pripada `*Tp`.

Apstraktni tip podataka `BinaryTree`

`node` . . . bilo koji tip (imena čvorova). U skupu `node` uočavamo jedan poseban element `LAMBDA` (koji služi kao ime nepostojećeg čvora).
`labeltype` . . . bilo koji tip (oznake čvorova).
`BinaryTree` . . . podatak tipa `BinaryTree` je binarno stablo čiji čvorovi su podaci tipa `node` (medusobno različiti i različiti od `LAMBDA`). Svakom čvoru je kao oznaka pridružen podatak tipa `labeltype`.
`void BiMakeNull(BinaryTree *Tp)` . . . funkcija pretvara binarno stablo `*Tp` u prazno binarno stablo.
`int BiEmpty(BinaryTree T)` . . . funkcija vraća 1 ako je `T` prazno binarno stablo, inače vraća 0.
`void BiCreate(labeltype l, BinaryTree TL, BinaryTree TR, BinaryTree *Tp)` . . . funkcija stvara novo binarno stablo `*Tp`, kojem je lijevo podstablo `TL`, a desno podstablo `TR` (`TL` i `TR` moraju biti disjunktni). Korijen od `*Tp` dobiva oznaku `l`.
`void BiLeftSubtree(BinaryTree T, BinaryTree *TLp), void BiRightSubtree(BinaryTree T, BinaryTree *TRp)` . . . funkcija preko `*TLp` odnosno `*TRp` vraća lijevo odnosno desno podstablo binarnog stabla `T`. Nije definirana ako je `T` prazno.
`node BiInsertLeftChild(labeltype l, node i, BinaryTree *Tp), node BiInsertRightChild(labeltype l, node i, BinaryTree *Tp)` . . . funkcija u binarno stablo `*Tp` ubacuje novi čvor s oznakom `l`, tako da on bude lijevo odnosno desno dijete čvora `i`. Funkcija vraća novi čvor. Nije definirana ako `i` ne pripada `T` ili ako `i` već ima to dijete.
`void BiDelete(node i, BinaryTree *Tp)` . . . funkcija izbacuje list `i` iz binarnog stabla `*Tp`. Nije definirana ako `i` ne pripada `*Tp` ili ako `i` ima djece.
`node BiRoot(BinaryTree T)` . . . funkcija vraća korijen binarnog stabla `T`. Ako je `T` prazno, vraća `LAMBDA`.
`node BiLeftChild(node i, BinaryTree T), node BiRightChild(node i, BinaryTree T)` . . . funkcija vraća lijevo odnosno desno dijete čvora `i` u binarnom stablu `T`. Ako `i` nema dotično dijete, vraća `LAMBDA`. Nije definirana ako `i` ne pripada `T`.
`node BiParent(node i, BinaryTree T)` . . . funkcija vraća roditelja čvora `i` u binarnom stablu `T`. Ako je `i` korijen, vraća `LAMBDA`. Nije definirana ako `i` ne pripada `T`.
`labeltype BiLabel(node i, BinaryTree T)` . . . funkcija vraća oznaku čvora `i` u binarnom stablu `T`. Nije definirana ako `i` ne pripada `T`.
`void BiChangeLabel(labeltype l, node i, BinaryTree *Tp)` . . . funkcija mijenja oznaku čvora `i` u stablu `*Tp`, tako da ta oznaka postane `l`. Nije definirana ako `i` ne pripada `*Tp`.

Apstraktni tip podataka `Set`

`elementtype` . . . bilo koji tip s totalnim uređajem `<=`.
`Set` . . . podatak tipa `Set` je konačan skup čiji elementi su (medusobno različiti) podaci tipa `elementtype`.
`void SeMakeNull(Set *Ap)` . . . funkcija pretvara skup `*Ap` u prazan skup.
`void SeInsert(elementtype x, Set *Ap)` . . . funkcija ubacuje element `x` u skup `*Ap`.
`void SeDelete(elementtype x, Set *Ap)` . . . funkcija izbacuje element `x` iz skupa `*Ap`.
`int SeMember(elementtype x, Set A)` . . . funkcija vraća 1 ako je `x` element od `A`, odnosno 0 ako nije.
`elementtype SeMin(Set A), elementtype SeMax(Set A)` . . . funkcija vraća najmanji odnosno najveći element skupa `A`, u smislu uredaja `<=`. Nije definirana ako je `A` prazan skup.
`int SeSubset(Set A, Set B)` . . . funkcija vraća 1 ako je `A` podskup od `B`, inače vraća 0.
`void SeUnion(Set A, Set B, Set *Cp)` . . . funkcija pretvara skup `*Cp` u uniju skupova `A` i `B`.
`void SeIntersection(Set A, Set B, Set *Cp)` . . . funkcija pretvara skup `*Cp` u presjek skupova `A` i `B`.
`void SeDifference(Set A, Set B, Set *Cp)` . . . funkcija pretvara skup `*Cp` u razliku skupova `A` i `B`.

Apstraktni tip podataka `Dictionary`

`elementtype` . . . bilo koji tip s totalnim uredajem `<=`.
`Dictionary` . . . podatak tipa `Dictionary` je konačan skup čiji elementi su (međusobno različiti) podaci tipa `elementtype`.
`void DiMakeNull(Dictionary *Ap)` . . . funkcija pretvara riječnik `*Ap` u prazan riječnik.
`void DiInsert(elementtype x, Dictionary *Ap)` . . . funkcija ubacuje element `x` u riječnik `*Ap`.
`void DiDelete(elementtype x, Dictionary *Ap)` . . . funkcija izbacuje element `x` iz riječnika `*Ap`.
`int DiMember(elementtype x, Dictionary A)` . . . funkcija vraća 1 ako je `x` element od `A`, odnosno 0 ako nije.

Apstraktni tip podataka `PriorityQueue`

`elementtype` . . . bilo koji tip s totalnim uredajem `<=`.
`PriorityQueue` . . . podatak ovog tipa je konačan skup čiji elementi su (međusobno različiti) podaci tipa `elementtype`.
`void PrMakeNull(PriorityQueue *Ap)` . . . funkcija pretvara skup `*Ap` u prazan skup.
`int PrEmpty(PriorityQueue A)` . . . funkcija vraća 1 ako je `A` prazan skup, inače vraća 0.
`void PrInsert(elementtype x, PriorityQueue *Ap)` . . . funkcija ubacuje element `x` u skup `*Ap`.
`elementtype PrDeleteMin(PriorityQueue *Ap)` . . . funkcija iz skupa `*Ap` izbacuje najmanji element i vraća taj izbačeni element. Nije definirana ako je `*Ap` prazan skup.

Apstraktni tip podataka `Mapping`

`domain` . . . bilo koji tip (domena).
`range` . . . bilo koji tip (kodomena).
`Mapping` . . . podatak tipa `Mapping` je preslikavanje čiju domenu čine podaci tipa `domain`, a kodomenu podaci tipa `range`.
`void MaMakeNull(Mapping *Mp)` . . . funkcija pretvara preslikavanje `*Mp` u nul-preslikavanje, tj. takvo koje nije nigdje definirano.
`void MaAssign(Mapping *Mp, domain d, range r)` . . . funkcija definira `*Mp(d)` tako da bude `*Mp(d)` jednako `r`, bez obzira da li je `*Mp(d)` prije bilo definirano ili nije.
`void MaDeassign(Mapping *Mp, domain d)` . . . funkcija uzrokuje da `*Mp(d)` postane nedefinirano, bez obzira da li je `*Mp(d)` bilo prije definirano ili nije.
`int MaCompute(Mapping M, domain d, range *rp)` . . . ako je `M(d)` definirano, tada funkcija vraća 1 i pridružuje varijabli `*rp` vrijednost `M(d)`, inače funkcija vraća 0.

Apstraktni tip podataka `Relation`

`domain1` . . . bilo koji tip (prva domena).
`domain2` . . . bilo koji tip (druga domena).
`set1` . . . podatak tipa `set1` je konačan skup podataka tipa `domain1`.
`set2` . . . podatak tipa `set2` je konačan skup podataka tipa `domain2`.
`Relation` . . . podatak tipa `Relation` je binarna relacija čiju prvu domenu čine podaci tipa `domain1`, a drugu domenu podaci tipa `domain2`.
`void ReMakeNull(Relation *Rp)` . . . funkcija pretvara relaciju `*Rp` u nul-relaciju, tj. takvu u kojoj ni jedan podatak nije ni s jednim u relaciji.
`void ReRelate(Relation *Rp, domain1 d1, domain2 d2)` . . . funkcija postavlja da je `d1 *Rp d2`, bez obzira da li je to već bilo postavljeno ili nije.
`void ReUnrelate(Relation *Rp, domain1 d1, domain2 d2)` . . . funkcija postavlja da je `d1 *R̸p d2`, bez obzira da li je to već bilo postavljeno ili nije.
`void ReCompute2(Relation R, domain1 d1, set2 *S2p)` . . . za zadani `d1` funkcija skupu `*S2p` pridružuje kao vrijednost $\{$`d2` | `d1 R d2`$\}$.
`void ReCompute1(Relation R, set1 *S1p, domain2 d2)` . . . za zadani `d2` funkcija skupu `*S1p` pridružuje kao vrijednost $\{$`d1` | `d1 R d2`$\}$.