Praktikum StrukturData
Modul PSDA

PSDA-03. ADT LIST BERKAIT LINIER

Representasi Fisik Pointer – Type List dengan First Eksplisit

1. Buatlah ADT List Berkait Linier dengan representasi fisik pointer dalam bahasa C sesuai spesifikasi di bawah ini. Untuk type List, digunakan type list dengan elemen First yang dinyatakan secara eksplisit (lihat kembali diktat "Struktur Data"). Untuk type boolean, gunakan file berisi definisi type boolean secara terpisah (**boolean.h**).
2. Buatlah driver untuk memeriksa apakah semua primitif yang didefinisikan telah berjalan dengan baik.

```
{ ***************** MODUL LIST BERKAIT ***************** }
{ List direpresentasi dengan pointer, First dinyatakan secara eksplisit. }

{ Konstanta }
constant Nil : ... { address tidak terdefinisi }

type infotype : integer
type ElmtList : < Info : infotype, Next : address >
type address : pointer to ElmtList
type List : < First : address >

{ Jika L : List dan P : address (address untuk traversal), maka penulisan : }
{      First(L) menjadi L.First
       Next(P)  menjadi P↑.Next
       Info(P)  menjadi P↑.Info    }

{ Definisikan selektor yang tepat. }

{ PROTOTYPE }
{ ***************** TEST LIST KOSONG ***************** }
function ListEmpty (L : List) → boolean
{ Mengirim true jika list kosong }

{ ***************** PEMBUATAN LIST KOSONG ***************** }
procedure CreateList (output L : List)
{ I.S. sembarang             }
{ F.S. Terbentuk list kosong }

{ ***************** Manajemen Memori ***************** }
function Alokasi (X : infotype) → address
{ Mengirimkan address hasil alokasi sebuah elemen }
{ Jika alokasi berhasil, maka address tidak Nil, dan misalnya menghasilkan P, maka
   Info(P) = X, Next(P) = Nil }
{ Jika alokasi gagal, mengirimkan Nil }
procedure Dealokasi (input/output P : address)
{ I.S. P terdefinisi }
{ F.S. P dikembalikan ke sistem }
{ Melakukan dealokasi/pengembalian address P }

{ ***************** PENCARIAN SEBUAH ELEMEN LIST ***************** }
function Search (L : List, X : infotype) → address
{ Mencari apakah ada elemen list dengan Info(P) = X }
```

```
{ Jika ada, mengirimkan address elemen tersebut }
{ Jika tidak ada, mengirimkan Nil }
function FSearch (L : List, P : address) → boolean
{ Mencari apakah ada elemen list yang beralamat P }
{ Mengirimkan true jika ada, false jika tidak ada }
function SearchPrec (L : List, X : infotype) → address
{ Mengirimkan address elemen sebelum elemen yang nilainya = X }
{ Mencari apakah ada elemen list dengan Info(P) = X }
{ Jika ada, mengirimkan address Prec, dengan Next(Prec) = P dan Info(P) = X }

{ Jika tidak ada, mengirimkan Nil }
{ Jika P adalah elemen pertama, maka mengirimkan Nil }
{ Search dengan spesifikasi seperti ini menghindari traversal ulang jika setelah
  Search akan dilakukan operasi lain }

{ ***************** PRIMITIF BERDASARKAN NILAI ***************** }

{ *** PENAMBAHAN ELEMEN *** }
procedure InsVFirst (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. X ditambahkan sebagai elemen pertama L }
{ Proses : Melakukan alokasi sebuah elemen dan menambahkan elemen pertama dengan
           nilai X jika alokasi berhasil.
             Jika alokasi gagal: I.S.= F.S. }
procedure InsVLast (input/output L : List, input X : infotype)
{ I.S. L mungkin kosong }
{ F.S. X ditambahkan sebagai elemen terakhir L }
{ Proses : Melakukan alokasi sebuah elemen dan menambahkan elemen list di akhir :
           elemen terakhir yang baru bernilai X jika alokasi berhasil.
           Jika alokasi gagal: I.S.= F.S. }

{ *** PENGHAPUSAN ELEMEN *** }
procedure DelVFirst (input/output L : List, input X : infotype)
{ I.S. List L tidak kosong }
{ F.S. Elemen pertama list dihapus : nilai info disimpan pada X }
{      dan alamat elemen pertama di-dealokasi }
procedure DelVLast (input/output L : List, output X : infotype)
{ I.S. list tidak kosong }
{ F.S. Elemen terakhir list dihapus : nilai info disimpan pada X }
{      dan alamat elemen terakhir di-dealokasi }

{ ***************** PRIMITIF BERDASARKAN ALAMAT ***************** }
{ *** PENAMBAHAN ELEMEN BERDASARKAN ALAMAT *** }
procedure InsertFirst (input/output L : List, input P : address)
{ I.S. Sembarang, P sudah dialokasi  }
{ F.S. Menambahkan elemen ber-address P sebagai elemen pertama }
procedure InsertAfter (input/output L : List, input P : address, input Prec :
address)
{ I.S. Prec pastilah elemen list dan bukan elemen terakhir, }
{      P sudah dialokasi  }
{ F.S. Insert P sebagai elemen sesudah elemen beralamat Prec }
procedure InsertLast (input/output L : List, input P : address P)
{ I.S. Sembarang, P sudah dialokasi }
{ F.S. P ditambahkan sebagai elemen terakhir yang baru }

{ *** PENGHAPUSAN SEBUAH ELEMEN *** }
procedure DelFirst (input/output L : List, output P : address)
```

```
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen pertama list sebelum penghapusan }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
{ First element yang baru adalah suksesor elemen pertama yang lama }
procedure DelP (input/output L : List, input X : infotype)
{ I.S. Sembarang }
{ F.S. Jika ada elemen list beraddress P, dengan Info(P) = X }
{ Maka P dihapus dari list dan di-dealokasi }
{ Jika tidak ada elemen list dengan Info(P) = X, maka list tetap }
{ List mungkin menjadi kosong karena penghapusan }
procedure DelLast (input/output L : List, input P : address)
{ I.S. List tidak kosong }
{ F.S. P adalah alamat elemen terakhir list sebelum penghapusan  }
{      Elemen list berkurang satu (mungkin menjadi kosong) }
{ Last element baru adalah predesesor elemen pertama yang lama, jika ada }
procedure DelAfter (input/output L : List, output Pdel : address, input Prec :
address)
{ I.S. List tidak kosong. Prec adalah anggota list L. }
{ F.S. Menghapus Next(Prec) : Pdel adalah alamat elemen list yang dihapus }

{ ***************** PROSES SEMUA ELEMEN LIST ***************** }
procedure PrintInfo (input L : List)
{ I.S. List mungkin kosong }
{ F.S. Jika list tidak kosong, }
{      Semua info yg disimpan pada elemen list diprint }
{      Jika list kosong, hanya menuliskan "list kosong" }
function NbElmt (L : List) → integer
{ Mengirimkan banyaknya elemen list; mengirimkan 0 jika list kosong }

{ *** Prekondisi untuk Max/Min/rata-rata : List tidak kosong *** }
function Max (L : List) → infotype
{ Mengirimkan nilai Info(P) yang maksimum }
function AdrMax (L : List) → address
{ Mengirimkan address P, dengan Info(P) yang bernilai maksimum }
function Min (L : List) → infotype
{ Mengirimkan nilai Info(P) yang minimum }
function AdrMin (L : List) → address
{ Mengirimkan address P, dengan Info(P) yang bernilai minimum }
function Average (L : List) → real
{ Mengirimkan nilai rata-rata Info(P) }

{ ***************** PROSES TERHADAP LIST ***************** }
procedure DelAll (input/output L : List)
{ Delete semua elemen list dan alamat elemen di-dealokasi }
{ I.S. : L terdefinisi, boleh kosong }
{ F.S. : Jika L tidak kosong, semua elemen list di-delete dan didealokasi }

procedure InversList (input/output L : List)
{ I.S. L terdefinisi, boleh kosong }
{ F.S. Elemen list L dibalik : }
{      Elemen terakhir menjadi elemen pertama, dan seterusnya. }
{      Membalik elemen list, tanpa melakukan alokasi/dealokasi. }

function FInversList (L : List) → List
{ Mengirimkan list baru, hasil invers dari L dengan menyalin semua elemn list. }
{ Alokasi mungkin gagal. Jika alokasi gagal, hasilnya list kosong dan semua elemen
  yang terlanjur di-alokasi, harus didealokasi. }
```

```
procedure CopyList (input/output L1 : List, output L2 : List)
{ I.S. L1 terdefinisi, L2 sembarang. F.S. L2 = L1 }
{ L1 dan L2 "menunjuk" kepada list yang sama. }
{ Tidak ada alokasi/dealokasi elemen baru. }

function FCopyList (L : List) →List
{ Mengirimkan list yang merupakan salinan L dengan melakukan alokasi elemen baru. }
{ Jika ada alokasi gagal, hasilnya list kosong dan semua elemen yang terlanjur di-
  alokasi, harus didealokasi. }

procedure CpAlokList (input Lin : List, input/output Lout : List)
{ I.S. Lout sembarang, Lin terdefinisi. }
{ F.S. Jika semua alokasi berhasil, maka Lout berisi hasil copy Lin }
{      Jika ada alokasi yang gagal, maka Lout = Nil dan semua elemen yang terlanjur
       dialokasi, didealokasi }

procedure Konkat (input L1, L2 : List; input/output L3 : List)
{ I.S. L1 dan L2 terdefinisi, boleh kosong. }
{ F.S. L1 dan L2 tetap, L3 adalah hasil konkatenasi L1 dan L2 }
{ Jika semua alokasi berhasil, maka L3 adalah hasil konkatenasi L1 dan L2. }
{ Jika ada alokasi yang gagal, semua elemen yang sudah dialokasi harus didealokasi
  dan L3 = Nil. }
{ Konkatenasi dua buah list : L1 dan L2 menghasilkan L3 yang "baru". }
{ Elemen L3 adalah hasil alokasi elemen yang "baru". }
{ Jika ada alokasi yang gagal, maka L3 harus bernilai Nil dan semua elemen yang
  pernah dialokasi harus didealokasi. }
procedure Konkat1 (input/output L1, L2 : List; output L3 : List)
{ I.S. L1 dan L2 sembarang }
{ F.S. L1 dan L2 kosong, L3 adalah hasil konkatenasi L1 dan L2 }
{ Konkatenasi dua buah list : L1 dan L2  menghasilkan L3 yang baru (dengan elemen
  list L1 dan L2) dan L1 serta L2 menjadi list kosong. }
{ Tidak ada alokasi/dealokasi pada prosedur ini }

procedure PecahList (output L1, L2 : List, input L : List)
{ I.S. L mungkin kosong }
{ F.S. Berdasarkan L, dibentuk dua buah list L1 dan L2 }
{ L tidak berubah : untuk membentuk L1 dan L2 harus alokasi. }
{ L1 berisi separuh elemen L dan L2 berisi sisa elemen L. }
{ Jika elemen L ganjil, maka separuh adalah NbElmt(L) div 2. }
```