UTS IF2110 2020

Ujian Tengah Semester IF2110 Algoritma dan Struktur Data, Semester I 2020/2021

Petunjuk

- Waktu: 120 menit (13:00–15:00)
- Boleh membuka diktat Notasi Algoritmik
- Untuk menuliskan karakter spesial, dapat digunakan *shortcut* ⊞ ; (di Windows), ^ ⌘ Space (di Mac), atau silakan salin & tempel dari sini: → ← ≠ ≤ ≥

Bagian I. ADT Matriks (Notasi Algoritmik, bobot: 25%)

Diberikan definisi dan spesifikasi ADT Matriks sebagai berikut:

```
{ ADT Matriks }
type indeks: integer { indeks baris dan kolom }
constant BrsMin: indeks = 0
constant BrsMax: indeks = 99
constant KolMin: indeks = 0
constant KolMax: indeks = 99
type el_type: integer
type MATRIKS:
      < Mem: matrix[BrsMin..BrsMax,KolMin..KolMax] of el_type,
        NBrsEff: integer > 0, { banyaknya/ukuran baris yg terdefinisi }
        NKolEff: integer > 0  { banyaknya/ukuran kolom yg terdefinisi } >
{ Memori matriks yang dipakai selalu di "ujung kiri atas" }

{************ DEFINISI PROTOTIPE PRIMITIF ************}
{*** Konstruktor membentuk Matriks ***}
procedure MakeMATRIKS (input NB, NK: integer, output M: MATRIKS)
{ Membentuk sebuah Matriks "kosong" berukuran NB×NK di "ujung kiri atas"
  memori.
  I.S. NB dan NK adalah valid untuk memori matriks yang dibuat.
  F.S. Matriks M terdefinisi dengan ukuran NB×NK. }

{*** Selektor Matriks ***}
{*** Selektor Get ***}
function GetNBrsEff (M: MATRIKS) → integer
{ Mengirimkan banyaknya baris efektif M }
function GetNKolEff (M: MATRIKS) → integer
{ Mengirimkan banyaknya kolom efektif M }
function GetFirstIdxBrs (M: MATRIKS) → indeks
{ Mengirimkan indeks baris terkecil M }
function GetFirstIdxKol (M: MATRIKS) → indeks
{ Mengirimkan indeks kolom terkecil M }
function GetLastIdxBrs (M: MATRIKS) → indeks
{ Mengirimkan indeks baris terbesar M }
function GetLastIdxKol (M: MATRIKS) → indeks
{ Mengirimkan indeks kolom terbesar M }
function GetElmtDiagonal (M: MATRIKS, i: indeks) → el_type
{ Mengirimkan elemen M(i,i) }
{*** Selektor Set: Operasi mengubah nilai elemen matriks ***}
function GetElmt (M: MATRIKS; i, j: integer) → el_type
{ Mengirimkan elemen matriks M[i,j] }
procedure SetElmt (input/output M: MATRIKS; input i, j: integer; input X: el_type)
{ I.S. M sudah terdefinisi; F.S. M[i,j] bernilai X }
```

Dengan menggunakan ADT di atas, **buatlah implementasi prosedur dan fungsi di bawah ini**. Spesifikasi prosedur/fungsi yang sudah didefinisikan pada ADT tidak perlu ditulis ulang. Tidak diperkenankan membuat fungsi/prosedur tambahan lainnya.

```
function IsSegitigaBawah (M: MATRIKS) → boolean
{ Mengirimkan true jika M adalah matriks segitiga bawah,dengan M_ij = 0
  untuk semua i < j.
  Prekondisi: M adalah matriks bujur sangkar (berukuran n × n) }

procedure DetriMatriks (input M: MATRIKS, output Det: integer)
{ Menghitung nilai determinan matriks segitiga M, dimana determinant
  dihitung berdasarkan perkalian semua elemen diagonalnya.
  I.S. Matriks M terdefinisi;
  F.S. Det berisi nilai determinan matriks segitiga bawah M atau 0 jika
       bukan matriks segitiga bawah. }
```

Contoh:

Untuk matriks M1 dan M2 berikut:

```
M1:   1   0   0   0      M2:   7  12   0   0
      3  15   0   0            2  13   8   4
     -7   9   8   0           16   3   0   0
     -4   2  11   6            3  10   1   5
```

Maka:

- `IsSegitigaBawah(M1) = true`
- `IsSegitigaBawah(M2) = false`
- `DetriMatriks(M1,Det) ⇒ Det=720`
- `DetriMatriks(M2,Det) ⇒ Det=0`

Bagian II. Mesin Kata dan ADT Stack (Notasi Algoritmik, bobot: 25%)

Sebuah **mesin token (modifikasi dari mesin kata model 1)** digunakan untuk membaca sebuah pita karakter yang berisi data transaksi di sebuah ATM dalam satu hari, yang masing-masing direpresentasikan dalam sebuah token dan diakhiri dengan '`.`'. Sebuah token menggambarkan sebuah transaksi di ATM. Ada dua jenis transaksi yaitu: **`tarik`** dan **`setor`**. Beberapa contoh isi pita karakter adalah sebagai berikut:

```
tarik-2500000    tarik-500000   setor-250000  tarik-50000  .
```

```
   .
```

```
  setor-1000000.
```

Pita karakter diasumsikan selalu dalam format yang benar (tidak ada bentuk-bentuk "kata" lain yang yang tidak valid). Pita bisa merupakan pita kosong hanya terdiri atas '`.`' (mungkin didahului sejumlah spasi/blank). Hasil akuisisi "kata" dalam mesin ini adalah sebuah token **CToken** yang terdiri dari dua komponen ⟨`opr`,`val`⟩, dengan *op* berisi operan yang dibaca (bernilai "`tarik`" atau "`setor`") dan *val* berisi nilai uang yang ditarik atau disetor dari/ke ATM.

Contoh: token-token yang dihasilkan dari pita pertama berturut-turut adalah ⟨"`tarik`",`2500000`⟩, ⟨"`tarik`",`500000`⟩, ⟨"`setor`",`250000`⟩, ⟨"`tarik`",`50000`⟩.

Berikut adalah definisi Mesin Token (hasil modifikasi **Mesin Kata model 1**):

```
{ ***** MesinToken ***** }
use MESINKARAKTER
{ Primitif Mesin karakter adalah: START dan ADV.
  State Mesin karakter adalah:  CC (karakter hasil akuisisi) dan
                               EOP (true jika CC = MARK) }
{ ***** Konstanta ***** }
constant  MARK: character = '.'
constant BLANK: character = ' '

{ ***** Definisi State Mesin ***** }
type Token = < opr: string,  { operator yang diakuisisi, bisa "tarik"
                               atau "setor" }
               val: integer  { nilai uang yang ditarik atau disetor } >
```

```
CToken: Token
EndToken: boolean

{ ***** Primitif-Primitif Mesin Integer ***** }
procedure STARTTOKEN
{ I.S.: CToken sembarang }
{ F.S.: Jika CC = MARK, maka EndToken = true; }
{       atau EndToken = false, CToken adalah token yang sudah diakuisisi;
        CC karakter pertama sesudah karakter terakhir yang diakuisisi. }
procedure ADVTOKEN
{ I.S.: EndToken = false; CC adalah karakter sesudah karakter terakhir
        token yang sudah diakuisisi. }
{ F.S.: Jika CC = MARK, maka EndToken = true; }
{       atau EndToken = false, CToken adalah token yang sudah diakuisisi;
        CC karakter pertama sesudah karakter terakhir yang diakuisisi. }
procedure IgnoreBlank
{ I.S.: CC sembarang }
{ F.S.: CC <> BLANK atau CC = MARK. }
procedure SalinToken
{ Mengakuisisi token. Hasil akuisisi (token) disimpan ke dalam CToken. }
{ I.S.: CC adalah karakter pertama dari token. }
{ F.S.: CToken berisi token yang sudah diakuisisi; CC = BLANK
        atau CC = MARK;
        CC adalah karakter sesudah karakter terakhir yang diakuisisi;
        CToken bisa berupa <"tarik",nilai integer> atau
        <"setor",nilai integer>.
        Proses akuisisi untuk integer menggunakan fungsi KarakterToInteger. }
function KarakterToInteger (C: character) → integer
{ Mengirimkan integer [0..9] yang merupakan konversi dari character C yang
  merupakan salah satu dari karakter ['0 '..'9']. }
```

Diketahui sebuah **ADT Stack** yang direpresentasikan dengan **array statik** sebagai berikut:

```
{ *** ADT Stack *** }
{ *** Konstanta *** }
constant Nil: integer = 0
constant MaxEl: integer = 100
{ *** Definisi elemen dan address *** }
type infotype: integer
type address: integer   { indeks tabel }
{ *** Definisi Type Queue *** }
type Stack: <  T: array [1..MaxEl] of infotype, { tabel penyimpan elemen }
            TOP: address { alamat top stack } >
{ Definisi Stack kosong: Top = Nil }
{ *** Predikat Pemeriksaan Kondisi Stack *** }
function IsEmpty (S: Stack) → boolean
{ Mengirim true jika S kosong }
function IsFull (S: Stack) → boolean
{ Mengirim true jika tabel penampung elemen S sudah penuh yaitu mengandung
  MaxEl elemen }
{ *** Konstruktor *** }
procedure CreateEmpty (output S: Stack)
{ I.S. Sembarang
  F.S. Sebuah S kosong terbentuk dengan TOP = Nil }
{ *** Operator-Operator Dasar Stack *** }
procedure Push (input/output S: Stack, input X: infotype)
{ Proses: Menambahkan  X pada S dengan aturan LIFO }
  I.S. S mungkin kosong, tabel penampung elemen S TIDAK penuh.
  F.S. X menjadi TOP yang baru, TOP "maju". }
procedure Pop (input/output S: Stack, output X: infotype)
{ Proses: Menghapus elemen pada S dengan aturan LIFO
  I.S. S tidak kosong
  F.S. X = nilai elemen TOP pada I.S., TOP "mundur". }
```

1. Buatlah realisasi prosedur **`STARTTOKEN`** dan **`SalinToken`** pada modul mesin token sesuai dengan spesifikasi yang diberikan (spesifikasi tidak perlu ditulis ulang). Tidak boleh menambahkan

fungsi/prosedur baru.

2. Lengkapi program utama berikut ini, yang membaca sebuah pita karakter dengan menggunakan mesin token untuk mendapatkan nilai total setoran dikurangi total penarikan pada hari itu. Gunakan *stack of integer* untuk membantuk melakukan penghitungan. Spesifikasi program tidak perlu ditulis kembali.

```
program TransaksiATM
{  Input: Pita karakter (yang dibaca menggunakan mesin token).
  Proses: Baca token dari pita karakter.
          Selama Stack tidak penuh:
            Jika yang dibaca <"setor", nilai-integer>, push nilai
            integer tersebut ke dalam Stack
            Jika yang dibaca <"tarik", nilai-integer>, push nilai
            integer dikali (-1)
            ke dalam Stack.
          Jika Stack penuh, tampilkan pesan "Stack penuh"
          Jika Stack tidak penuh dan pita karakter selesai dibaca:
            pop dan jumlahkan seluruh nilai yang ada pada Stack, lalu
            tampilkan hasilnya
  Output: - pesan "tidak ada transaksi" jika pita karakter kosong atau
            hanya berisi blank, ATAU
          - pesan "Stack penuh" jika Stack penuh, ATAU
          - hasil penjumlahan seluruh elemen Stack, yang menggambarkan
            total setoran dikurangi total penarikan }
Use ...

KAMUS
  ...
ALGORITMA
  ...
```

Bagian III. ADT Queue (Notasi Algoritmik, bobot: 25%)

Diberikan definisi ADT Queue yang direpresentasikan dengan array statik menggunakan alternatif-1 sebagai berikut.

```
{ Modul ADT Queue – Alternatif I }
{ *** Deklarasi Queue yang diimplementasi dengan array *** }
{ *** HEAD dan TAIL adalah alamat elemen pertama dan terakhir  *** }
{ *** Queue mampu menampung MaxEl buah elemen *** }
{ *** Konstanta *** }
constant Nil: integer = -1
constant MaxEl: integer = 100 { maksimum banyaknya elemen queue }
{ *** Definisi elemen dan address *** }
type infotype: integer
type address : integer {indeks tabel}

{ *** Definisi Type Queue *** }
type Queue: <      T: array [0..MaxEl-1] of infotype
                              { array penyimpan elemen Queue }
               HEAD: address, { alamat penghapusan}
               TAIL: address, { alamat penambahan} >
{ Definisi Queue kosong: Head = Nil; TAIL = Nil.}
{ Definisi akses dengan Selektor:
  Head(Q), Tail(Q), MaxEl(Q), InfoHead(Q), InfoTail(Q)
  dengan Q adalah Queue }
{ *** Predikat Pemeriksaan Kondisi Queue *** }
function IsEmpty (Q: Queue) → boolean
{ Mengirim true jika Q kosong }
function IsFull (Q: Queue) → boolean
{ Mengirim true jika array penampung elemen Q sudah penuh yaitu mengandung
  MaxEl elemen }
function NBElmt (Q: Queue) → integer
{ Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong. }
{ *** Konstruktor dan Destruktor *** }
```

```
procedure CreateEmpty (output Q: Queue, input Max: integer > 0)
{ I.S. Max terdefinisi }
{ F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb:       }
{      Jika alokasi berhasil, tabel memori dialokasi berukuran Max }
{      atau: jika alokasi gagal, Q kosong dg MaxEl=0               }
{ Proses: Melakukan alokasi memori dan membuat sebuah Q kosong     }
procedure DeAlokasi (input/output Q: Queue)
{ Proses: Mengembalikan memori Q }
{ I.S. Q pernah dialokasi }
{ F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 }
{ *** Operator-Operator Dasar Queue *** }
procedure Enqueue (input/output Q: Queue, input X: infotype)
{ Proses: Menambahkan  X pada Q dengan aturan FIFO }
{ I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh }
{ F.S. X menjadi TAIL yang baru}
procedure Dequeue (input/output Q: Queue, output X: infotype)
{ Proses: Menghapus elemen pertama pada Q dengan aturan FIFO }
{ I.S. Q tidak kosong }
{ F.S. X = nilai elemen HEAD pada I.S.,
       Jika Queue masih isi: HEAD diset tetap = 0, elemen-elemen setelah
       HEAD yang lama digeser ke "kiri", TAIL = TAIL - 1;
       Jika Queue menjadi kosong, HEAD=TAIL=Nil }
{ *** Operator Tambahan *** }
procedure Split (input Q: Queue, input ratio: real, output Q1, Q2: Queue)
{ I.S. Q tidak kosong, dan 0 ≤ X  ≤ 1 }
{ F.S. Q1 dan Q2 akan berisi elemen-elemen dari Q secara bersilangan
       (elemen pertama Q ke Q1, elemen kedua Q ke Q2, elemen ketiga Q ke
       Q1, dan seterusnya), dengan NBElmt(Q1) = ratio × NBElmt(Q).
       Jika hasilnya bukan bilangan bulat, maka dilakukan pembualatan ke
       atas.
       Contoh: NBElmt(Q)=5 dan ratio = 0,25, maka NBElmt(Q1)=0,25×5=1,25≈2.
       Sehingga Q1 akan memiliki 2 elemen (dibulatkan ke atas), dan Q2 akan
       memiliki 3 elemen. }
```

Dengan spesifikasi ADT Queue di atas, realisasikan prosedur-prosedur berikut (spesifikasi tidak perlu ditulis ulang). Gunakan semaksimal mungkin primitif dan selektor yang ada. Tidak diperkenankan untuk membuat fungsi/prosedur baru.

1. Prosedur `Dequeue`.
2. Prosedur `Split`.

Bagian IV. ADT Set dan Hash (Notasi Algoritmik, bobot: 25%)

Cpl. Gionno (dibaca: Kopral Jono) ingin berhenti dari tentara untuk membuat suatu sistem *e-commerce*. Untuk itu ia membutuhkan ADT Bag yang merepresentasikan keranjang belanja. ADT Bag mirip dengan ADT Set, bedanya untuk setiap item yang disimpan dalam sebuah Bag terdapat pencatat jumlah. Satu unit item dapat ditambahkan dan dikurangi dari Bag dengan operasi `Add` dan `Remove`. Sebagai ilustrasi, untuk sebuah Bag B yang semula kosong, urutan pemanggilan operasi `Add(B,telur); Add(B,susu); Add(B,susu); Add(B,telur); Remove(B,susu); Add(B,roti); Remove(B,roti);` menghasilkan isi B: `[telur⇒2, susu⇒1]`. Selain itu terdapat fungsi `Get` yang mengembalikan jumlah kemunculan sebuah item dalam Bag. Pada ilustrasi tadi, `Get(B,telur)` mengembalikan `2` sedangkan `Get(B,roti)` mengembalikan `0`.

Agar penggunaan memori efisien, Gionno tidak mau menyimpan item dalam Bag sebagai `string`, melainkan sebagai `integer` yang merepresentasikan ID barang. Ia menggunakan ADT Map untuk menyimpan pemetaan ID barang ke *tuple* `Item` berisi nama dan harga satuan, misal `[54⇒ ⟨"telur",3500⟩ , 72⇒ ⟨"susu",12000⟩ , 63⇒ ⟨"roti",16000⟩ ]`. Nama dan harga sebuah `item` dapat diakses dengan `item.name` dan `item.price`. Bantulah Gionno membangun sistem *e-commerce* ini dengan:

1. Membuat **definisi ADT Bag**. Gunakan *array* kontigu untuk menyimpan elemen-elemen Bag dan `NbElmt` untuk mencatat jumlah jenis item yang ada di dalam Bag. Tuliskan prototipe untuk konstruktor dan operasi-operasi `Add`, `Remove`, dan `Get` lengkap dengan spesifikasinya (prekondisi/I.S.,F.S.)

2. Membuat **implementasi operasi `Add` dan `Remove`**. Asumsikan item disimpan dalam *array* sesuai urutan penambahan item ke Bag. (Tidak *sorted*, tidak pakai *hash table*.)

3. Membuat **prosedur `CetakInvoice(input B:Bag, input M:Map)`** yang mencetak jumlah, nama, dan harga setiap item di dalam Bag B, serta total harga yang harus dibayar. Untuk memudahkan iterasi, Gionno sudah membuat fungsi `ItemIDs(B: Bag) → array of integer` yang mengembalikan *array* berisi ID semua item yang ada di B. ADT Map dengan key bertipe `integer` dan value bertipe `Item` juga sudah dibuat oleh Gionno dengan sebagian spesifikasi sebagai berikut—Anda tinggal menggunakannya:

```
type Item: < name: string, price: integer  >
type keytype: integer
type infotype: Item
type Map: ... { terdefinisi }

{** aksesor **}
function Get(M: Map, k: keytype) → infotype
{ Mengembalikan tuple Item yang berkorespondensi dengan key k dalam M
  atau Nil jika key k tidak ada dalam M }
```

Contoh hasil pemanggilan `CetakInvoice` pada Bag B `[54⇒2,72⇒1]` dan Map M `[54⇒⟨"telur",3500⟩ , 72⇒ ⟨"susu",12000⟩ , 63⇒ ⟨"roti",16000⟩ ]`:

```
2 telur 7000
1 susu 12000
TOTAL 19000
```

Tidak memperkenankan membuat/mengasumsikan adanya fungsi/prosedur selain yang disebutkan pada soal.