Toets Programmeren, 2YP05
op donderdag 13 november 2008, 09:00-12:00

TU/e Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
(Na de toets gecorrigeerde versie)

PROBLEEM: Sleutels

Lees de probleembeschrijving van de opgave *Sleutels* nauwkeurig door (deze staat op een apart vel). Dit was probleem C bij de voorronde van het Nederlands Kampioenschap Programmeren (NKP) 2001.

AANVULLENDE INFORMATIE

Uw programma dient de **invoer** te lezen uit het tekstbestand met de naam `sleutels.in` (let op: dit moeten kleine letters zijn).

Uw programma dient de **uitvoer** te schrijven naar het nieuw aan te maken tekstbestand met de naam `sleutels.out` (let op: dit moeten kleine letters zijn). De uitvoerregels kunnen heel lang worden.

N.B. Het maakt verder niet zo uit wat uw programma op standaarduitvoer schrijft. Een slotregel als '`Tik <return> ...`' op standaarduitvoer wordt geaccepteerd.

U kunt er op rekenen dat het volgende geldt (uw programma hoeft dit dus niet te controleren):

- Het tekstbestand `sleutels.in` bestaat en bevat invoer die aan de gestelde eisen voldoet.
- Een sleutelset bevat geen duplicaten, d.w.z. alle sleutels erin zijn verschillend.
- Het bestand `sleutels.out` bestaat nog niet.
- Het invoerbestand bevat minder dan 1000 (duizend) sleutelsets.

ANALYSE

Elke sleutelset kan afzonderlijk verwerkt worden. Het is wel nodig om de gehele sleutelset op te slaan alvorens de bijbehorende uitvoer bepaald kan worden. Na verwerking is de sleutelset niet meer nodig.

Laten we een sleutel *slecht* noemen wanneer deze is om te vijlen tot een *andere* sleutel in de set. Per sleutel kan nagegaan worden of deze *slecht* is door hem achtereenvolgens te vergelijken met elke andere sleutel in de set. De sleutelset is *goed* als deze geen *slechte* sleutels bevat. *Slechte* sleutels moeten uitgevoerd worden.

ONTWERP

De beschrijving van een sleutel wordt aangeboden als *Integer*, maar het is eenvoudiger om een sleutel te verwerken als *String*:

```
type
  TSleutel = String; { bestaat uit B cijfers }
```

Het is niet nodig om de hoogte van een enkel staafje te weten (d.w.z. een karakter te vertalen naar een getal). Het is alleen nodig om (overeenkomstige) staafjes van twee steutels te vergelijken in hoogte: als `s1` en `s2` twee variabelen zijn van het type `TSleutel` dan kan dat met `s1[i] < s2[i]`.

Met deze aanpak blijken de invoerwaarden B en H niet meer relevant te zijn. **N.B.** Ze moeten wel ingelezen worden, maar spelen verder geen rol!

Introduceer een **record** `TSleutelSet` om de gegevens van een sleutelset op te slaan:

```
type
  TSleutelIndex = ...

  TSleutelSet = record
    B: ...; { ... }
    H: ...; { ... }
    N: ...; { ... }
    sleutel: array [ TSleutelIndex ] of TSleutel; { ... }
  end;
```

Definieer geschikte en relevante routines, waaronder ten minste één functie en één procedure. Voor de hand liggen:

- procedure `ReadSleutelSet` voor inlezen van een sleutelset uit een gegeven, voor lezen geopend, invoerbestand.
- Boolean functie `IsOmTeVijlen` om van twee gegeven sleutels na te gaan of de eerste is om te vijlen tot de tweede.

- Boolean functie `IsSlechteSleutel` om na te gaan of een gegeven sleutel is om te vijlen tot een *andere* sleutel in een gegeven sleutelset.
- procedure `CheckSleutelSet` om na te gaan of een gegeven sleutelset *goed* is en bijbehorende uitvoer te schrijven naar een gegeven, voor schrijven geopend, uitvoerbestand.

Bovendien valt te denken aan:

- procedure om een sleutelset leeg te initialiseren.
- procedure om een sleutel toe te voegen aan een sleutelset.
- procedure om van een sleutelset de verzameling *slechte* sleutels te bepalen, via een `TSleutelSet` out parameter.
- procedure om een sleutelset te schrijven naar een uitvoerbestand.
- procedure voor verwerken van één sleutelset, inclusief inlezen en uitvoer schrijven.

CODEREN

Als vangnet wordt een monolithische oplossing van dit probleem meegeleverd in een Lazarus project, zie `sleutels0.lpr`. Dit programma bevat geen definities van constantes, records of andere types, of routines, en is verder van (te) weinig commentaar voorzien. Het is niet vereist dit programma te raadplegen of ingredienten ervan over te nemen. U mag uw eigen oplossing bedenken.

Denk aan de **codeerstandaard**.

N.B Alle routines dienen voorzien te zijn van een specificatie op basis van een pre/post/ret contract.

Een stappenplan:

1. Definieer record type(s).
2. Geef routine specificaties, bestaande uit aanhef met naam, parameters en types, en het contract.
3. Definieer routine implementaties.
4. Maak het hoofdprogramma.

NORMERING

Een programma dat geen geschikte en relevante record types of routines definieert, levert *geen* voldoende op. Voor een 6 is een programma met een record type en twee routines vereist. Met vier routines, waaronder een functie en een procedure, is maximaal een 9 te halen. Voor een 10 zijn ten minste zes routines nodig.

Ten overvloede: de record types en routines dienen geschikt te zijn en relevant voor het probleem.

INLEVEREN

- Lever alleen de Pascal programmatekst in. In Lazarus is dat het `.lpr` bestand.
- **Lever altijd wat in**, ook als het onvolledig is of niet goed werkt. Geschikte definities van routines kunnen ook punten opleveren als ze niet tot een werkend geheel samengevoegd zijn.
- Vermeld in uw programma uw **naam**, **idententiteitsnummer** en de **datum**.

C Sleutels

In het stadje Sloterdam woont een sleutelmaker, die sleutels maakt. Iedere sleutel heeft een uniek deel, wat bepaalt of een sleutel wel of niet op een bepaald slot past. Dit deel heeft een bepaalde geheeltallige breedte B en een, eveneens geheeltallige, hoogte H. Dit houdt in dat het bestaat uit B 'staafjes', die ieder maximaal H eenheden hoog kunnen zijn. Bijvoorbeeld:

Deze sleutel heeft breedte 4 en hoogte 3. De 'staafjes' hebben de hoogten 3, 1, 3 en 2.

Nu is het mogelijk om van één sleutel een andere sleutel te maken door er één of meerdere stukjes af te vijlen. Dit wil de sleutelmaker natuurlijk vermijden. Daarom zorgt hij er altijd voor, dat de sleutels uit één set zó van elkaar verschillen, dat dit niet mogelijk is. Hij heeft echter een leerling in dienst die dit idee niet helemaal begrepen heeft. Deze leerling heeft voor de sleutelmaker een groot aantal ontwerpen gemaakt voor sleutelsets, zodat de sleutelmaker aan het ontwerpen niet zoveel tijd kwijt is. Hierbij heeft hij er echter niet op gelet of ze wel aan de eis van de sleutelmaker voldoen. Hij heeft er wel voor gezorgd dat alle sleutels in zo'n set van elkaar verschillen. Voordat de sleutelmaker een sleutelset daadwerkelijk gaat maken, wil hij er natuurlijk wel van verzekerd zijn dat het een goede set is. Gelukkig heeft de leerling zijn ontwerpen in een tekstbestand op de computer opgeslagen, zodat dit makkelijk met een computerprogramma te controleren is.

Probleem

Schrijf een programma dat van een gegeven sleutelset bepaalt of dit een goede set is. Dat wil zeggen een set, waarbij het niet mogelijk is om van een sleutel uit de set een andere sleutel uit dezelfde set te maken, door er één of meerdere delen af te vijlen. Indien de set niet goed is, print dan díe sleutels waarmee de eigenaar de deur van iemand anders zou kunnen openen, indien hij er een deel af zou vijlen.

Invoer

Eerst een regel met daarop een enkele positieve integer: het aantal sets. Daarna per set:

- één regel met daarop 3 integers: de breedte ($2 \leq B \leq 20$) en de maximale hoogte ($2 \leq H \leq 9$) van de sleutels, en het aantal sleutels N in deze set ($2 \leq N \leq 100$)
- N regels met op iedere regel een integer van B cijfers, die een sleutel uit de set representeert. Ieder cijfer geeft de hoogte h van een van de staafjes weer ($0 \leq h \leq H$), waarbij deze staafjes uiteraard in de volgorde staan waarin ze op de sleutel zouden komen.

Uitvoer

Voor iedere sleutelset één regel met daarop het woord '`GOED`', indien de set goed is. Indien de set niet goed is, alle sleutels die door te vijlen omgezet zouden kunnen worden in een (of meerdere) andere sleutel(s) uit dezelfde set, gescheiden door een spatie. De sleutels dienen door hetzelfde getal te worden gerepresenteerd als bij de invoer, en ook in dezelfde volgorde te staan waarin ze bij de invoer voorkomen.

Voorbeeld

invoer	bijbehorende **uitvoer**

`2`
`3 2 3`
`112`
`121`
`211`
`2 3 3`
`22`
`23`
`21`
	`GOED`
	`22 23`