Université de Nice-Sophia Antipolis — Deug MIAS-MI 1
Algorithmique & Programmation — 2003–2004

TP N° 3

Structures de données composées

Le but de cette séance est d'utiliser les structures de données vues en cours et en TD.

1 Recherche d'une chaîne de caractères dans un fichier

La commande `grep` est une des commandes de base des systèmes d'exploitation de la famille Unix. Elle cherche à l'intérieur de fichiers de texte des lignes correspondant à un motif de texte et les affiche. Dans son utilisation la plus basique, ce motif est une simple chaîne de caractères. Le but de cet exercice va être d'implémenter un programme `grep` simple.

a. Sous Emacs, la première entrée du menu `Tools` est `Search File (Grep)...`. En cliquant dessus, vous obtenez une invite dans la zone de saisie de commandes en bas de la fenêtre :

```
Run grep (like this): grep -n -e
```

Complétez cette commande avec la chaîne de caractères `factorielle` suivie du nom de fichier `*.pas`, qui est en fait une liste des fichiers pascal de votre répertoire de travail.

```
Run grep (like this): grep -n -e factorielle *.pas
```

Si vos fichiers du TP numéro 1 sont présents, Emacs vous ouvre un nouveau *buffer* recensant toutes les lignes des fichiers pascal contenant le mot "factorielle". En reproduisant ces opérations mais en saisissant :

```
Run grep (like this): grep -n -e "function factorielle" *.pas
```

Emacs n'affiche que les lignes de la déclaration de la fonction `factorielle`.

b. Les fichiers textes en Pascal ont pour type **`text`**, qui n'est autre que **`file of char`**. Comme vu en cours, l'ouverture d'un fichier en lecture se fait d'abord en associant le fichier à son nom par `assign(f, name)`, puis en déclarant qu'on va l'inspecter par **`reset`**`(f)`. Le caractère dans le tampon est `f^`, et on l'avance d'un caractère par **`get`**`(f)`. `eof(f)` indique si on a atteint la fin du fichier.

Les fichiers textes ont de plus la possibilité d'avancer le tampon jusqu'au début de la prochaine nouvelle ligne avec **`readln`**`(f)`. De plus, `eoln(f)` devient vrai quand on atteint une fin de ligne.

Pour écrire un programme `grep`, vous allez devoir être capables de revenir en arrière dans le texte du fichier, ce qui n'est pas directement possible. Une solution est de stocker le contenu de la ligne dans un tampon ; on définit pour cela :

```
const
   tailleTampon = 8192;  { La taille du tampon d'entrée.
                           C'est aussi la longueur maximale d'une ligne. }
```

Écrivez une procédure `trouveChaine` prenant pour paramètres deux objets de type `string` : le premier est la chaîne de caractères que l'on va chercher, le second est le nom du fichier que l'on va explorer à cette fin :

```
{ Affiche les lignes du fichier contenant la chaîne 's'.
La chaîne 's' ne contient pas de retours à la ligne. }
procedure trouveChaine(s, nomFichier : string);
var
   f      : text;                          { la variable fichier }
   tampon : array[1..tailleTampon] of char;  { le contenu de la ligne lue }
```

c. Les paramètres de la ligne de commande passés à un programme Pascal sont au nombre de `ParamCount`. Leurs valeurs de type chaîne de caractères sont aisément récupérées par l'appel à `ParamStr(i)` où `i` est l'index entre 1 et `ParamCount` du paramètre.

Pour un programme `grep` permettant d'appeler la procédure `trouveChaine` avec pour paramètres les deux paramètres de la ligne de commande dans l'ordre, cela donne :

```
begin
   if (ParamCount <> 2) then
   begin
      writeln('Le programme grep prend deux arguments :');
      writeln('- une chaîne de caractères (au besoin entre guillemets)');
      writeln('- un nom de fichier à explorer')
   end
   else
      trouveChaine(ParamStr(1), ParamStr(2))
end. { grep }
```

Créez un nouveau fichier `test.txt` sous Emacs, écrivez plusieurs lignes de texte dont une contenant la chaîne `tototoro`.

Testez ensuite votre programme (compilé avec l'option `-o tp03-1`):

```
Run grep (like this): ./tp03-3 totoro test.txt
```

La ligne contenant la chaîne `tototoro` doit apparaître.

d. Étendez votre programme pour permettre de chercher la chaîne de caractères dans une liste de fichiers.

```
{ Le programme 'grep' cherche des occurences d'une chaîne
de caractères au sein d'un fichier texte, et affiche
s'il y en a les lignes où la chaîne apparaît. }
program grep;

const
   tailleTampon = 8192;  { La taille du tampon d'entrée.
                           C'est aussi la longueur maximale d'une ligne. }


{ Affiche les lignes du fichier contenant la chaîne 's'.
La chaîne 's' ne contient pas de retours à la ligne. }
procedure trouveChaine(s, nomFichier : string);
var
   f      : text;                          { la variable fichier }
   tampon : array[1..tailleTampon] of char; { le contenu de la ligne lue }
   i      : integer;                       { indice dans 's' }
   d      : integer;                       { point de départ dans 'tampon' }
   j      : integer;                       { indice dans 'tampon' }
   l      : integer;                       { longueur utile dans 'tampon' }
   trouve : boolean;                       { si on a déjà trouvé 's' }
   n      : integer;                       { numéro de la ligne }
begin
   { Ouverture du fichier de texte dans 'f'. }
   assign(f, nomFichier);
   reset(f);

   { Initialisation du tampon. }
   d := 1;
   l := 1;

   { Initialisation de 'trouve' et de 'n'. }
```

```
trouve := false;
n := 1;

{ Parcours du fichier texte. }
while (not eof(f)) do
begin

   { Recopie de la ligne de 'f' dans 'tampon'. }
   while (not eoln(f)) and (l < tailleTampon) do
   begin
      tampon[l] := f^;
      l := l + 1;
      get(f)
   end;

   if (l = tailleTampon) then
      writeln('Erreur : débordement de tampon.');

   { 'l' contient la taille utile du tampon.
   'd' est le point de départ de son exploration. }
   while (d < l) do
   begin
      { On se place au point de départ dans le tampon. }
      j := d;

      { Avance dans 'tampon' jusqu'à atteindre le premier
      caractère de 's'. }
      while (j < l) and (tampon[j] <> s[1]) do
         j := j + 1;

      { Sauvegarde du point de départ pour le prochain tour de boucle. }
      d := j + 1;

      { Avance dans 's' et 'tampon' de manière synchrone. }
      i := 1;
      while (j < l) and (i <= length(s)) and (tampon[j] = s[i]) do
      begin
         i := i + 1;
         j := j + 1
      end;

      { Affiche le contenu de la ligne si nécessaire. }
      if (i = length(s) + 1) then
      begin
         { Affiche le nom du fichier. }
         if (trouve = false) then
         begin
            writeln(nomFichier, ' :');
            trouve := true;
         end;
         write('  ', n, ' : ');
         { Recopie la ligne. }
         for j := 1 to l-1 do
            write(tampon[j]);
         writeln  { ajout du retour à la ligne sur la sortie }
      end
   end;
```

```
        { Prépare la lecture de la ligne suivante. }
        readln(f);
        n := n + 1;
        d := 1;
        l := 1
    end
end; { trouveChaine }

{ Récupération des arguments de la ligne de commande
et appel de trouveChaine. }
var
    i : integer;  { indice dans la liste des arguments }
begin
    if (ParamCount < 2) then
    begin
        writeln('Le programme grep prend au minimum deux arguments :');
        writeln('- une chaîne de caractères (au besoin entre guillemets)');
        writeln('- une liste de noms de fichiers')
    end
    else
        for i := 2 to ParamCount do
            trouveChaine(ParamStr(1), ParamStr(i))
end. { grep }
```

.....

2 Articles

On veut implémenter une notion modélisant un étudiant en tenant compte de trois caractéristiques :

- le prénom,
- le nom,
- l'âge.

e. Déclaration et première utilisation de l'article `etudiant` On se propose d'implémenter cette notion d'`etudiant` avec un article comportant les champs adéquats. On supposera que les noms et les prénoms ne comportent pas plus de 31 caractères et que les âges sont compris entre 1 et 120 ans. Déclarez le type d'un article `etudiant`. Après avoir déclaré un étudiant, vous affecterez ses champs et vous les ferez afficher dans le programme principal.

f. Saisie et affichage d'un `etudiant` Écrire une procédure `SaisirEtudiant` qui prend en argument un `etudiant` et qui affecte les trois champs de cet `etudiant` avec des valeurs saisies au clavier par l'utilisateur. Écrire une procédure `AfficherEtudiant` qui prend en argument un `etudiant` et qui affiche ses trois champs comme dans l'exemple suivant :
`Nom = Knuth; Prénom = Donald; Âge = 66 ans`
Testez vos procédures dans votre programme principal.

g. Groupes d'étudiants On propose de faire des groupes d'étudiants (disons des groupes de 5 étudiants) qui s'entendent bien afin qu'ils travaillent ensemble de façon privilégiée. Déclarez le type `groupe` d'un tableau d'`etudiant` et écrivez deux procédures, l'une pour affecter les `etudiant`s d'un groupe et l'autre pour les afficher. Testez vos procédures dans votre programme principal.

h. Deux prédicats sur les groupes d'étudiants De façon informelle, un prédicat est une fonction qui permet de tester si un article ou une collection d'articles vérifie une propriété donnée. Ici par exemple, on vous demande un prédicat booléen (*i.e.* la fonction répond vrai si la propriété est

vérifiée et faux sinon) permettant à l'utilisateur de tester s'il y a un `etudiant` d'un âge de son choix dans un `groupe`. De même, faites un prédicat permettant à l'utilisateur de tester s'il y a au moins une occurence d'un `etudiant` du nom de son choix dans un groupe. Vous devez tester vos deux prédicats.

```
program exo2;

const
   AgeMin = 1; AgeMax = 120;
   taille = 31;
   nbAmis = 5;


{déclaration du type d'enregistrement pour un étudiant}
type etudiant = record
                   {champ pour le nom de l'étudiant}
                   nom    : string[taille];
                   {champ pour le prénom d'un étudiant}
                   prenom : string[taille];
                   {champ pour l'âge d'un étudiant}
                   age    : AgeMin..AgeMax;
end;

{déclaration du type d'un groupe qui est un tableau de 5 étudiants}
type
   groupe = array[1..nbAmis] of etudiant;


var
   copains : groupe;
   temp    : etudiant;
   cible   : 1..120;
   find    : boolean;


{procédure pour la saisie d'un étudiant}
procedure SaisieEtudiant(var Ami : etudiant);
begin
   Write('Saisie du nom : '); ReadLn(Ami.nom);
   Write('Saisie du prénom : '); ReadLn(Ami.prenom);
   Write('Saisie de l''âge : '); ReadLn(Ami.age)
end; { SaisieEtudiant }


{procédure pour l'affichage d'un étudiant}
procedure AfficheEtudiant(Ami : etudiant);
begin
   Write('Prénom = ');
   Write(Ami.prenom);
   Write('; Nom = ');
   Write(Ami.nom);
   Write('; Âge = ');
   WriteLn(Ami.age)
end; { AfficheEtudiant }

{procédure pour la saisie d'un groupe}
procedure SaisieGroupe(var mesAmis :  groupe);
```

```
var i : 1..nbAmis;
begin
   WriteLn('Saisie d''un groupe :');
   for i:=1 to nbAmis do begin
      Write('Saisie de l''étudiant numéro '); Write(i); WriteLn(' : ');
      SaisieEtudiant(mesAmis[i])
   end
end; { SaisieGroupe }

{procédure pour l'affichage d'un groupe}
procedure AfficheGroupe(mesAmis :  groupe);
var i : 1..nbAmis;
begin
   WriteLn('Affiche d''un groupe :');
   for i:=1 to nbAmis do
      AfficheEtudiant(mesAmis[i])
end; { AfficheGroupe }


{fonction pour tester si il y a un étudiant d'un age donné dans le groupe}
function TestParAge(mesAmis : groupe; ageCible : integer) : boolean;
var i       : 1..nbAmis;
   resultat :  boolean;
begin
   resultat := false;
   for i:= 1 to nbAmis do begin
      if mesAmis[i].age = ageCible then
         resultat := true
   end;
   return(resultat)
end; { TestParAge }

begin
   {tests pour l'exo e}
   temp.nom := 'Truc';
   temp.prenom := 'Bidule';
   temp.age := 19;

   {tests pour l'exo f}
   AfficheEtudiant(temp);
   SaisieEtudiant(temp);
   AfficheEtudiant(temp);

   {test pour l'exo g}
   SaisieGroupe(copains);
   AfficheGroupe(copains);

   {test pour l'exo h}
   Write('Saisir l''âge recherché : '); ReadLn(cible);
   find := TestParAge(copains, cible);
   if find then
      WriteLn('Il y a au moins un étudiant ayant l''âge requis.')
   else
      WriteLn('Il n''y a pas d''étudiant ayant l''âge requis.')
end.
```

.....