TD I3
v1.9.1

N. Malandain, N. Delestre

Table des matières

1 Affectation et schéma conditionnel

1.1 Affectation

1.1.1 Échanger

Ecrire une procédure, échangerDeuxEntiers, qui permet d'échanger les valeurs de deux entiers.

1.1.2 Permutation circulaire

Écrire une procédure qui permet d'affectuer une permutation circulaire des valeurs entières de trois variables x, y, z (c'est-à-dire la valeur de y dans x, la valeur de z dans y et la valeur de x dans z.

1.1.3 Décomposition d'une somme d'argent

Écrire une fonction, decomposerSomme, qui à partir d'une somme ronde d'argent donnée, donne le nombre minimal de billets de 5 et 10 € et le nombre de pièces de 2 et 1 € qui la compose.

1.1.4 Parité

Écrire une fonction booléenne, estPair, qui à partir d'un nombre entier strictement positif retourne VRAI si ce nombre est pair et FAUX sinon.

1.1.5 Date valide

Écrire une fonction qui pour un numéro de jour, de mois et d'année donnés, détermine s'il s'agit ou non d'une date, après JC, valide (d'après le calendrier grégorien).
Rappel [1] :

> "Depuis l'instauration du calendrier grégorien, sont bissextiles :
>
> 1. les années divisibles par 4 mais non divisibles par 100
> 2. les années divisibles par 400
>
> Ainsi, l'an 2004 était bissextile suivant la première règle. L'an 1900 n'était pas bissextile, car divisible par 100, ce qui va à l'encontre de la première règle, et non divisible par 400, ce qui va à l'encontre de la seconde. L'an 2000 était bissextile car divisible par 400."

1.2 Schéma conditionnel

1.2.1 Tri de trois entiers

Écrire une procédure, trierTroisEntiers, qui prend en entrée trois paramètres a, b et c contenant chacun un entier et qui les retourne triés par ordre croissant : a contient la valeur minimum, et c contient la valeur maximum.

1. D'après Wikipedia

1.2.2 Nombre de jours de congés

Dans une entreprise, le calcul des jours de congés payés s'effectue de la manière suivante : si une personne est entrée dans l'entreprise depuis moins d'un an, elle a droit à deux jours de congés par mois de présence (au minimum 1 mois), sinon à 28 jours au moins. Si cette personne est un cadre et si elle est âgée d'au moins 35 ans et si son ancienneté est supérieure à 3 ans, il lui est accordé 2 jours supplémentaires. Si elle est cadre et si elle est âgée d'au moins 45 ans et si son ancienneté est supérieure à 5 ans, il lui est accordé 4 jours supplémentaires, en plus des 2 accordés pour plus de 35 ans.

Écrire une fonction, nbJoursDeConges, qui calcule le nombre de jours de congés à partir de l'âge exprimé en année, l'ancienneté exprimée en mois et l'appartenance (booléenne) au collège cadre d'une personne.

2 Schéma itératif

2.1 La multiplication

Écrire une fonction, multiplication, qui effectue la multiplication de deux entiers positifs (notés x et y) donnés en utilisant uniquement l'addition entière.

2.2 Calcul de factorielle n

Écrire une fonction, factorielle, qui calcule pour un entier positif donné n la valeur de n !.

2.3 Partie entière inférieure de la racine carrée d'un nombre

Écrire une fonction, racineEntiere, qui retourne la partie entière de la racine carrée d'un entier positif donné n (sans utiliser racineCarree).

2.4 Multiplication égyptienne

Les égyptiens de l'antiquité savaient :
— additionner deux entiers strictement positifs,
— soustraire 1 à un entier strictement positif,
— multiplier par 1 et 2 tout entier strictement positif,
— diviser par 2 un entier strictement positif pair.

Voici un exemple qui multiplie 14 par 13 en utilisant uniquement ces opérations :

14×13	= 14 + **14** $\times$ **(13 - 1)**	= 14 + **14** $\times$ **12**
	= 14 + **(14** $\times$ **2)** $\times$ **(12 / 2)**	= 14 + **28** $\times$ **6**
	= 14 + **(28** $\times$ **2)** $\times$ **(6 / 2)**	= 14 + **56** $\times$ **3**
	= 14 + 56 + **56** $\times$ **(3 - 1)**	= 70 + **56** $\times$ **2**
	= 70 + **(56** $\times$ **2)** $\times$ **(2 / 2)**	= 70 + **112** $\times$ **1**
	= 70 + 112	= 182

Donner le corps de la fonction suivante :

fonction multiplicationEgyptienne (a,b : **Naturel) : Naturel**

2.5 Intégration par la méthode des trapèzes

Écrire une fonction, integrale, qui retourne la valeur de l'intégrale d'une fonction f(x) réelle continue sur l'intervalle réel $[a, b]$. L'intégration consiste à découper cet intervalle, en n sous-

intervalles de longueur Δ. L'intégrale d'un sous-intervalle [x, x + Δ] est approximée au trapèze de bases f(x),f(x + Δ) et de hauteur Δ. a, b et n vous sont donnés.

Remarque : la communication de f entre l'appelant et la fonction appelée, est réalisée de manière implicite (opération transparente pour vous). Cette remarque est valide pour tous les exercices numériques de ce type.

3 Schéma itératif (suite)

3.1 Développement limité

Lorsque x est proche de 0, $sinh(x)$ peut être approximé à l'aide de la formule suivante :

$$\sum_{i=0}^{n} \frac{x^{2i+1}}{(2i+1)!}$$

Écrire la fonction sinh en n'utilisant aucune autre fonction (pas d'analyse descendante) :

fonction sinh (x : **Reel**, n : **Naturel) : Reel**
  Déclaration i : **Naturel**
    numerateur,denominateur : **Reel**
    resultat : **Reel**
debut
  ...
fin

3.2 Nombres premiers

Écrire une fonction booléenne, estPremier, qui à partir d'un entier strictement positif donné, retourne le résultat VRAI ou FAUX selon que le nombre est premier ou non. Pour mémoire, voici la liste des nombres premiers inférieurs à 100 : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

3.3 Recherche du zéro d'une fonction par dichotomie

Écrire une fonction, zeroFonction, qui calcule le zéro d'une fonction réelle f(x) sur l'intervalle réel [a, b], avec une précision epsilon. La fonction f, monotone et continue, ne s'annule qu'une seule et unique fois sur]a, b[. Pour trouver ce zéro, la recherche procède par dichotomie, c'est-à-dire divise l'intervalle de recherche par deux à chaque étape. Soit m le milieu de [a, b]. Si f(m) et f(a) sont de même signe, le zéro recherché est dans l'intervalle [m, b], sinon il est dans l'intervalle [a, m]. a, b et epsilon vous sont donnés.

4 Analyse descendante

4.1 Nombre de chiffres pairs dans un nombre

On se propose de calculer le nombre de chiffres pairs d'un nombre entier positif donné. On suit pour cela l'analyse descendante présentée par la figure 1, tel que :

nbChiffresPairs résoud le problème demandé. Par exemple pour le nombre 821, on obtient 2.

nbChiffres permet d'obtenir le nombre de chiffres d'un nombre. Par exemple pour le nombre 821, on obtient 3.

iemeChiffre permet d'obtenir le ième chiffre d'un nombre. Par exemple pour 821, le premier chiffre est 1, le second 2 et le troisième est 8 (on ne traite pas les erreurs).

estPair permet de savoir si un nombre est pair.

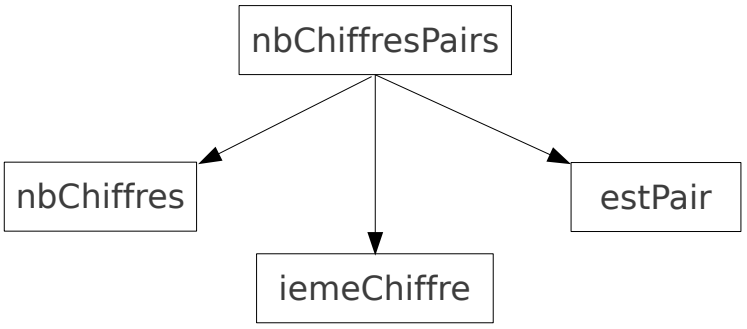

FIGURE 1 – Analyse descendante

1. Reprenez le schéma donné et complétez le (tel que nous l'avons vu en cours).
2. Donnez la signature des fonctions ou procédures des opérations données par l'analyse descendante.
3. Donnez le corps de la fonction ou procédure `nbChiffresPairs`.

4.2 Majuscule

La fonction $majuscule$ permet de calculer à partir d'une chaîne de caractères ch une chaîne de caractères ch' tel que tous les caractères minuscules, et uniquement eux, de ch soient transformés en majuscule dans ch'. La signature de cette est fonction est :

— **fonction** majuscule (uneChaine : **Chaine de caracteres) : Chaine de caracteres**

Ainsi $majuscule(" abc, ?ABC")$ donne la valeur $" ABC, ?ABC"$.

L'objectif de cet exercice est de donner l'algorithme de cette fonction en considérant que nous avons les trois fonctions suivantes :

— **fonction** longueur (uneChaine : **Chaine de caracteres) : Naturel**

— **fonction** iemeCaractere (uneChaine : **Chaine de caracteres**, position : **Naturel) : Caractere**

— **fonction** caractereEnChaine (unCaractere : **Caractere) : Chaine de caracteres**

4.2.1 Analyse

Pour calculer la version majuscule d'une chaîne de caractères ch, on a besoin de savoir calculer la majuscule d'un caractère c de ch lorsque c représente une lettre minuscule. Nous n'avons aucun a priori concernant la table de codage de ces caractères, si ce n'est que :

— le caractère 'a' précède le caractère 'b', qui précède le caractère 'c', etc.

— le caractère 'A' précède le caractère 'B', qui précède le caractère 'C', etc.

Proposez une analyse descendante de ce problème à l'aide du formalisme vu en cours.

4.2.2 Conception préliminaire

Déterminez la signature des fonctions ou procédures correspondant aux opérations de votre analyse descendante.

4.2.3 Conception détaillée

Donnez le corps de chacune de ces fonctions ou procédures.

5 TP Pascal : des multiplications

L'objectif de ce TP est de comparer les algorithmes vus dans les exercices 2.1 et 2.4.

5.1 Une première version du programme

L'objectif de ce premier programme est d'afficher le temps mis pour calculer la multiplication de deux nombreux positifs saisis par l'utilisateur. Le résultat attendu est du type :

```
$ ./multiplication
Calcul de a*b :
  a = 12
  b = 10000000
 Multiplication classique, a*b = 120000000 en 28 ms
 Multiplication egyptienne, a*b = 120000000 en 0 ms
```

Afin de tester la multiplication avec des grands nombres, vous utiliserez le type de données `QWord` pour un ordinateur 64 bits et `LongWord` pour un ordinateur 32 bits [2]. De plus, les ordinateurs d'aujourd'hui sont si rapides, que vous vous arrangerez pour que les deux algorithmes itèrent sur le plus grand des deux nombres à multiplier.

Vous pouvez calculer le temps mis pour l'exécution d'une procédure ou d'une fonction en l'encadrant par deux appels à la fonction `dateTimeToTimeStamp()` avec comme paramètre effectif l'appel à la fonction `time()` (disponibles dans l'unité `sysutils`). La valeur retournée par `dateTimeToTimeStamp` est de type `TTimeStamp`, qui est un enregistrement dont le champ `time` possède l'heure correspondante en ms. La soustraction de ces deux valeurs vous permettra d'avoir une idée quant au temps mis par cette fonction ou procédure.

Vous comparerez le temps mis par ces deux algorithmes en augmentant régulièrement la taille des nombres saisis (vous pouvez aller jusqu'à des valeurs de plusieurs milliards).

5.2 Une deuxième version du programme

Nous voulons maintenant pouvoir tracer des courbes montrant la complexité de ces deux algorithmes. Ces graphiques seront tracés grâce au logiciel Libreoffice calc à partir des calculs effectués par votre programme pascal.

Le programme va multiplier successivement un nombre a de plus en plus grand par un autre nombre constant (par exemple 1). Au départ a aura pour valeur 1000 et sera après chaque itération multiplié par 10 jusqu'à être supérieur à un nombre saisi par l'utilisateur. Pour chaque multiplication vous afficherez la valeur de a, le temps pour la multiplication classique et le temps pour la multiplication égyptienne (valeurs séparées par une virgule).

Voici un exemple d'exécution du programme :

2. L'instruction *for* ne fonctionne pas avec des entiers sur 64 bits. Il faut dans ce cas utiliser un *while*

```
$ ./multiplications
Borne max (>1000 et multiple de 10):10000000000
,Multiplication classique,Multiplication égyptienne
1000,0,0
10000,0,0
100000,0,0
1000000,6,0
10000000,31,0
100000000,199,0
1000000000,1983,0
```

À l'aide d'un copier/coller vous importerez dans LibreOffice calc ces données afin de créer le diagramme demandé (diagramme de type dispersion avec ligne et point).

6 Tableaux

Dans tous les exercices qui vont suivre, le tableau d'entiers t est défini comme étant de type **Tableau[**1..MAX**] d'Entier**.

6.1 Plus petit élément d'un tableau d'entiers

Écrire une fonction, minTableau, qui à partir d'un tableau d'entiers t non trié de n éléments significatifs retourne le plus petit élément du tableau.

6.2 Indice du plus petit élément d'un tableau d'entiers

Écrire une fonction, indiceMin, qui retourne l'indice du plus petit élément d'un tableau d'entiers t non trié de n éléments significatifs.

6.3 Nombre d'occurrences d'un élément

Écrire une fonction, nbOccurences, qui indique le nombre de fois où un élément apparaît dans un tableau d'entiers t non trié de n éléments.

6.4 Élément le plus fréquent d'un tableau

Soit un tableau d'entiers t non trié de n éléments significatifs. Écrire une procédure, determinerElementPlusFrequent, qui calcule l'élément qui apparaît le plus souvent dans le tableau t, ainsi que son nombre d'occurrences. Si plusieurs éléments différents répondent au problème, votre algorithme doit en fournir un, quel qu'il soit. Vous ne devez utiliser aucun autre tableau que celui sur lequel vous travaillez.

6.5 Recherche dichotomique

Écrire une fonction, rechercheDichotomique, qui détermine par dichotomie le plus petit indice d'un élément, (dont on est sûr de l'existence) dans un tableau d'entiers t trié dans l'ordre croissant de n éléments significatifs. Il peut y avoir des doubles (ou plus) dans le tableau.

6.6 Tris

6.6.1 Tri par insertion

Nous avons vu en cours que l'algorithme du tri par insertion est :

procédure triParInsertion **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** nb :**Naturel)**

Déclaration i,j : **Naturel**
temp : **Entier**

debut
pour i ←2 **à** nb **faire**
j ← obtenirIndiceDInsertion(t,i,t[i])
temp ← t[i]
decaler(t,j,i)
t[j] ← temp
finpour
fin

Donnez l'algorithme de la fonction obtenirIndiceDInsertion tout d'abord de manière séquentielle, puis de manière dichotomique. Démontrez que la complexité de ce dernier est-il en $O(log_2(n))$.

6.6.2 Tri shaker

La tri shaker est une amélioration du tri à bulles où les itérations permettant de savoir si le tableau et trié (et qui inverse deux éléments successifs $t[i]$ et $t[i+1]$ lorsque $t[i] > t[i+1]$) se font successivement de gauche à droite puis de droite à gauche.

Donnez l'algorithme du tri shaker.

7 TP Pascal : Matrice

Soit l'unité *Matrice* qui propose un type `TMatrice` et cinq sous-programmes qui permettent de manipuler des variables de type `TMatrice` :

function matriceZero(hauteur,largeur : Integer) : TMatrice permet d'obtenir une matrice composée de 0

function largeur(m : TMatrice) : Integer permet d'obtenir la largeur d'une matrice

function hauteur(m : TMatrice) : Integer permet d'obtenir la hauteur d'une matrice

function obtenir(m : TMatrice ; jHauteur, iLargeur : Integer) : Real ; permet d'obtenir un élément d'une matrice

procedure fixer(var m : TMatrice ; jHauteur, iLargeur : Integer ; val : Real) ; qui permet de fixer la valeur d'un des éléments de la matrice

L'objectif de ce TP est développer une unité *libMatrice*, qui permettra :

— de transformer une matrice en chaîne de caractères (fonction `matriceEnChaine`);
— d'additionner deux matrices (fonction `additionnerDeuxMatrices`);
— de multiplier deux matrices (fonction `multiplierDeuxMatrices`);
— de calculer la transposer d'une matrice (fonction `transposer`).

Pour les fonctions d'addition et de multiplication, les erreurs sont gérées via une variable globale `erreur` de type `TErreur`. Cette variable possédera la valeur `pas_d_erreur` lorsque tout va bien et la valeur `erreur_taille` lorsque la taille des matrices utilisées pour ces deux fonctions sont incompatibles. Dans ce dernier ces fonctions retourneront la valeur (0).

Le fichier `TP-Matrices-Sources.zip` disponible sur moodle, contient :

— l'unité `Matrice.pas`,
— l'unité `libMatrice.pas`,
— l'exécutable `test.pas`.

Complétez l'unité `libMatrice.pas` et utilisez le programme `test.pas` pour tester votre code. Voici un exemple d'exécution du programme test.

```
$ ./test
m1=
1.00 4.00
2.00 5.00
3.00 6.00

m2=
2.00 5.00
3.00 6.00
4.00 7.00

m3=
3.00 5.00 7.00
4.00 6.00 8.00

m1 + m2 =
 3.00  9.00
 5.00 11.00
 7.00 13.00

m1 * m3 =
19.00 29.00 39.00
26.00 40.00 54.00
33.00 51.00 69.00

transpose de m1 =
1.00 2.00 3.00
4.00 5.00 6.00
```

8 Récursivité

8.1 Calcul de C(n,p)

Écrire une fonction `cnp`, qui en fonction des entiers positifs n et p, retourne le nombre de combinaisons de p éléments parmi n.

8.2 Multiplication égyptienne

Soit la multiplication égyptienne définie dans l'exercice 2.4. On se propose cette fois d'écrire cet algorithme de manière récursive.

1. Déterminer le ou les cas d'arrêt (particuliers). Déterminer le ou les cas généraux.
2. Donner le corps de la fonction suivante en utilisant un algorithme récursif :

   fonction multiplicationEgyptienne **(**a,b : **Naturel) : Naturel**

8.3 Des cercles

Supposons que la procédure suivante permette de dessiner un cercle sur un graphique (variable de type `Graphique`) :

— **procédure** cercle **(E/S** g : Graphique,**E** xCentre,yCentre,rayon : **Reel)**

8.3.1 Compréhension

Soit l'algorithme suivant [3] :

procédure cercles (**E/S** g : Graphique,**E** x,y,r : **Reel**, n : **Naturel)**

Déclaration rTemp : **Reel**

debut

si n>0 **alors**

rTemp ← r/(1+racineCarree(2))

cercles(g,x,y+rTemp*racineCarree(2),rTemp,n-1)

cercles(g,x,y-rTemp*racineCarree(2),rTemp,n-1)

cercle(g,x,y,r)

cercles(g,x+rTemp*racineCarree(2),y,rTemp,n-1)

cercles(g,x-rTemp*racineCarree(2),y,rTemp,n-1)

finsi

fin

L'instruction `cercles(g,1.0,1.0,1.0,3)` permet d'obtenir le graphique de la figure 2.

Numérotez les cercles (numéro à mettre au centre du cercle) suivant leur ordre d'apparition sur le graphique.

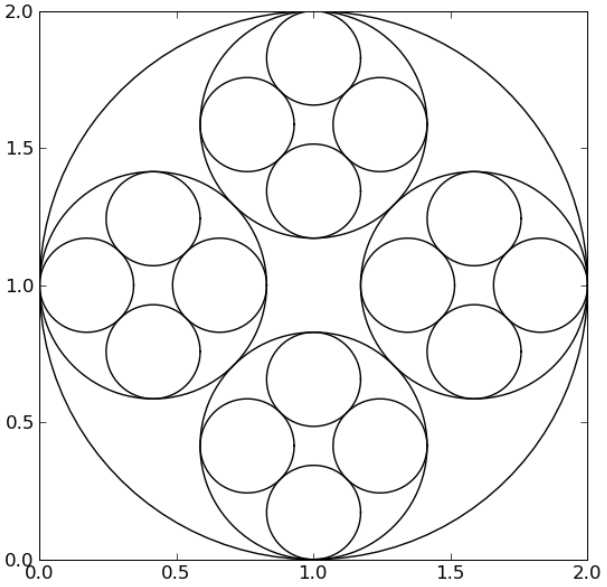

FIGURE 2 – Résultat d'un autre algorithme pour `cercles`

8.3.2 Construction

Proposez un autre algorithme de la procédure `cercles` qui, pour la même instruction `cercles(g,1.0,1.0,1.0,3)`, affiche les cercles dans l'ordre proposé par la figure 3.

3. Pour comprendre les formules mathématiques de cet algorithme, il faut considérer le carré définit par les 4 centres des cercles, de rayon r', intérieurs au cercle courant, de rayon r. Son côté est de $2r'$. Les centres sont donc a une distance de $r'\sqrt{2}$ du centre du cercle courant et donc $r = r'\sqrt{2} + r'$

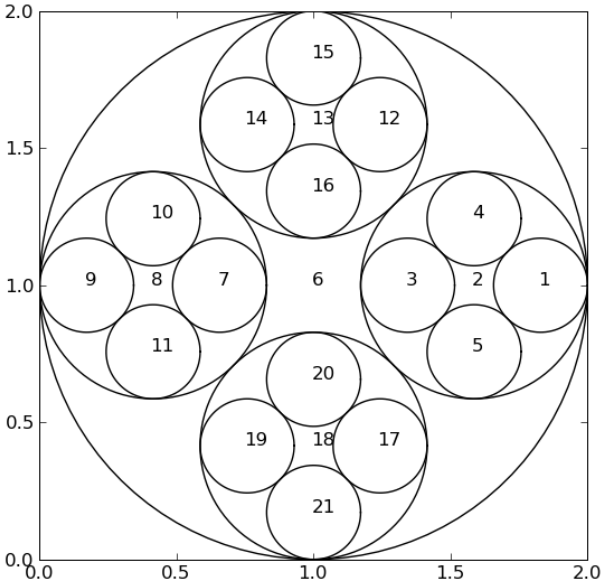

FIGURE 3 – Résultat d'un autre algorithme pour `cercles`

8.4 Recherche d'un élément dans un tableau

Écrire une fonction récursive, estPresent, qui retourne VRAI si un élément donné est un des éléments d'un tableau d'entiers t et FAUX sinon. Etudier les cas où t n'est pas un tableau trié et où t est un tableau trié.

9 TP Pascal : Benchmark de tri

L'objectif de ce TP est d'implanter et de comparer les performances des tris suivants :
— tri par sélection ;
— tri par insertions avec recherche d'insertion séquentielle ;
— tri par insertions avec recherche d'insertion dichotomique ;

9.1 Résultats attendus

Deux programmes sont utilisés dans ce TP : `testTri` et `benchmark`.

9.1.1 `testTri`

Le programme `testTri` a pour objectif de vérifier le bon fonctionnement des tris. Voici un exemple de résultats attendus à son exécution :

```
Verification des tris :
Nb elements :10
  Tableau a trier : 5 5 7 8 6 8 5 8 4 6
  Resultat Tri par selection : 4 5 5 5 6 6 7 8 8 8
  Resultat Tri par insertion (recherche d'insertion sequentielle) : 4 5 5 5 6 6 7 8 8 8
  Resultat Tri par insertion (recherche d'insertion dichotomique) : 4 5 5 5 6 6 7 8 8 8
```

9.1.2 `benchmark`

Le programme `benchmark` a pour objectif de comparer les performances des tris. Voici un exemple de résultats attendus à son exécution (les XX représentent des temps) :

```
Nb elements :10000
Tri par selection : XX ms
Tri par insertion (recherche d'insertion sequentielle) : XX ms
Tri par insertion (recherche d'insertion dichotomique) : XX ms
```

9.2 Travail à réaliser

Comme les tris sont utilisés par deux programmes, nous les avons répartis dans des unités pascal (une unité par tri).

De même le type `TEntiers` représentant un tableau d'entiers (avec des sous-programmes permettant d'en remplir aléatoirement un, d'en afficher un, ou encore de demander à l'utilisateur combien d'entiers il faut utiliser) est utilisé dans toutes les unités et les deux programmes, il est donc déclaré dans une unité.

Les programmes donnés compilent et s'exécutent, mais ne donnent pas les bons résultats. Pour réaliser ce TP, suivez les étapes suivantes :

1. Téléchargez et décompressez l'archive `TP-Tris1-Sources.zip` disponible sur moodle.
2. Complétez l'unité `triselection` et compilez les deux programmes.
3. Complétez l'unité `triinsertionsequentiel` et compilez les deux programmes.
4. Complétez l'unité `triinsertiondichotomique` et compilez les deux programmes.

Comparez les valeurs obtenues avec 1000, 10000, 20000 et 30000 éléments à trier.

10 TP Pascal : Benchmark de tri (suite)

L'objectif de ce TP est de compléter le TP précédent, en implantant et comparant les performances des tris suivants :

— tri à bulles
— tri par sélection ;
— tri par insertions avec recherche d'insertion séquentielle ;
— tri par insertions avec recherche d'insertion dichotomique ;
— tri rapide
— tri par fusion

10.1 Résultats attendus

Deux programmes sont utilisés dans ce TP : `testTri` et `benchmark`.

10.1.1 `testTri`

Le programme `testTri` a pour objectif de vérifier le bon fonctionnement des tris. Voici un exemple de résultats attendus à son exécution :

```
Verification des tris :
Nb elements :10
  Tableau a trier : 5 5 7 8 6 8 5 8 4 6
  Resultat Tri rapide : 4 5 5 5 6 6 7 8 8 8
  Resultat Tri par fusion : 4 5 5 5 6 6 7 8 8 8
  Resultat Tri par selection : 4 5 5 5 6 6 7 8 8 8
```

```
Resultat Tri par insertion (recherche d'insertion sequentielle) : 4 5 5 5 6 6 7 8 8 8
Resultat Tri par insertion (recherche d'insertion dichotomique) : 4 5 5 5 6 6 7 8 8 8
Resultat Tri a bulles : 4 5 5 5 6 6 7 8 8 8
```

10.1.2 `benchmark`

Le programme `benchmark` a pour objectif de comparer les performances des tris. Voici un exemple de résultats attendus à son exécution (les XX représentent des temps) :

```
Nb elements :10000
Tri rapide : XX ms
Tri par fusion : XX ms
Tri par selection : XX ms
Tri par insertion (recherche d'insertion sequentielle) : XX ms
Tri par insertion (recherche d'insertion dichotomique) : XX ms
Tri a bulles : XX ms
```

10.2 Travail à réaliser

Les programmes donnés compilent et s'exécutent, mais ne donnent pas les bons résultats. Pour réaliser ce TP, suivez les étapes suivantes :

1. Téléchargez et décompressez l'archive `TP-Tris2-Sources.zip` disponible sur moodle.
2. Remplacez les unités `triselection`, `triinsertionsequentiel` et `triinsertiondichotomoqie` par celles que vous avez développées au dernier TP.
3. Complétez l'unité `trirapide` et compilez les deux programmes.
4. Complétez l'unité `trifusion` et compilez les deux programmes.

   Comparez les valeurs obtenues avec 1000, 10000, 20000 et 30000 éléments à trier.

11 Liste chaînée

Pour rappel, le type ListeChaineeDEntiers est défini de la façon suivante :

Type ListeChaineeDEntiers **=** ^ NoeudDEntier
Type NoeudDEntier **= Structure**
entier : **Entier**
listeSuivante : ListeChaineeDEntiers
finstructure

Et nous l'utilisons à l'aide des fonctions et procédures suivantes :

— **fonction** listeVide **() :** ListeChaineeDEntiers
— **fonction** estVide **(**uneListe : ListeChaineeDEntiers**) : Booleen**
— **procédure** ajouter **(E/S** uneListe : ListeChaineeDEntiers,**E** element : Entier**)**
— **fonction** obtenirEntier **(**uneListe : ListeChaineeDEntiers**) :** Entier
⌊**précondition(s)** non(estVide(uneListe))
— **fonction** obtenirListeSuivante **(**uneListe : ListeChaineeDEntiers**) :** ListeChaineeDEntiers
⌊**précondition(s)** non(estVide(uneListe))
— **procédure** fixerListeSuivante **(E/S** uneListe : ListeChaineeDEntiers,**E** nelleSuite : ListeChaineeDEntiers**)**
⌊**précondition(s)** non(estVide(uneListe))
— **procédure** supprimerTete **(E/S** l :ListeChaineeDEntiers**)**
⌊**précondition(s)** non estVide(l)

— **procédure** supprimer **(E/S** uneListe : ListeChaineeDEntiers**)**

1. Écrire une procédure itérative, afficher, qui permet d'afficher les éléments d'une liste chaînée.
2. Écrire une procédure récursive, afficher, qui permet d'afficher les éléments d'une liste chaînée.
3. Écrire une fonction booléenne récursive, estPresent, qui permet de savoir si un entier est présent dans une liste chaînée.
4. Écrire une procédure récursive, concatener, qui concatène deux listes chaînées.
5. Écrire une procédure itérative, inverser, qui permet d'inverser une liste chaînée.
6. Écrire une procédure récursive, inverser, qui permet d'inverser une liste chaînée.

12 TP Pascal : Liste chaînée

L'objectif de ce TP est de développer une librairie (unité `LibListeChaineeDEntiers`) sur une liste chaînée d'entiers (unité `ListeChaineeDEntiers`). Cette librairie permettra de :
— savoir si deux listes chaînées sont égales ;
— copier une liste chaînée d'entiers ;
— savoir si un entier est présent dans une liste chaînée d'entiers ;
— savoir les entiers d'une liste sont présents en ordre croissant ;
— concatener deux liste chaînées d'entiers.

Afin de vérifier que la librairie fonctionne, un programme de tests unitaires est proposé (programme `testLibListeChaineeDEntiers`).

12.1 Résultats attendus

L'exécution du programme de tests unitaires devra valider tous les tests :

```
$ ./testLibListeChaineDEntiers
Tests unitaires de la librairie LibListeChaineeDEntiers
  sontEgales(l,l) : OK
  sontEgales(l1,l2) (avec l1 et l2 qui possedent les meme elements) : OK
  sontEgales(l1,l2) (avec l1 et l2 qui ne possedent pas les meme elements) : OK
  copier(l) : OK
  estPresent(l,e) (avec e qui est reellement present au debut): OK
  estPresent(l,e) (avec e qui est reellement present a la fin): OK
  estPresent(l,e) (avec e qui est reellement present, cas general): OK
  estPresent(l,e) (avec e qui n'est pas present): OK
  estEnOrdreCroissant(l) (avec la liste (1 2 3)): OK
  estEnOrdreCroissant(l) (avec la liste (1 2 3 2 1)): OK
  concatener(l1,l2) : OK
```

12.2 Travail à réaliser

1. Téléchargez et décompressez l'archive `TP-Liste-Sources.zip` disponible sur moodle.
2. Complétez l'unité `LibListeChaineeDEntiers`. À chaque fois que vous avez codé une fonction et que la compilation de l'unité réussie, recompilez et lancez le programme de tests unitaires.

13 TP Pascal : Tableaux dynamiques

L'objectif de ce TP est de développer une unité pascal qui propose un type `TTableauDynamique DEntiers` qui permet d'utiliser une liste chaînée d'entiers comme un tableau, à savoir qu'il sera possible d'accéder aux ième éléments du tableau dynamique. Outre ce type, cette unité proposera les fonctions et procédures suivantes :

— création (fonction `créer`) d'un tableau vide (ne contenant aucun entier) ;
— obtention du nombre d'entiers (fonction `longueur`) ;
— insertion (procédure `inserer`) d'un entier à une position donnée ;
— ajout (procédure `ajouter`) d'un entier à la fin du tableau ;
— suppression (procédure `supprimer`) d'un entier à une position donnée ;
— obtention (fonction `iemeEntier`) de l'entier à une position donnée ;
— suppression (procédure `vider`) de tous les éléments du tableau dynamique.

13.1 Résultats attendus

Après vous avoir demandé combien au maximum d'entiers vous vouliez stocker (n), l'exécution du programme `testTableauDynamiqueDEntiers` vous affichera un tableau dynamique suite à :

1. l'ajout de $n/2$ entiers aléatoires ;
2. l'insertion aléatoire de $n/2$ entiers aléatoires ;
3. la suppression aléatoire de $n/2$ entiers ;
4. la suppression de tous les éléments restant.

Voici un exemple d'exécution de ce programme :

```
$ ./testTableauDynamiqueDEntiers
Combien d'entiers :20
Ajout de 54 : 54
Ajout de 59 : 54 59
Ajout de 71 : 54 59 71
Ajout de 84 : 54 59 71 84
Ajout de 60 : 54 59 71 84 60
Ajout de 85 : 54 59 71 84 60 85
Ajout de 54 : 54 59 71 84 60 85 54
Ajout de 84 : 54 59 71 84 60 85 54 84
Ajout de 42 : 54 59 71 84 60 85 54 84 42
Ajout de 62 : 54 59 71 84 60 85 54 84 42 62
Inserer de 43 a pos=8 : 54 59 71 84 60 85 54 43 84 42 62
Inserer de 5 a pos=8 : 54 59 71 84 60 85 54 5 43 84 42 62
Inserer de 38 a pos=1 : 38 54 59 71 84 60 85 54 5 43 84 42 62
Inserer de 81 a pos=4 : 38 54 59 81 71 84 60 85 54 5 43 84 42 62
Inserer de 56 a pos=6 : 38 54 59 81 71 56 84 60 85 54 5 43 84 42 62
Inserer de 83 a pos=8 : 38 54 59 81 71 56 84 83 60 85 54 5 43 84 42 62
Inserer de 8 a pos=14 : 38 54 59 81 71 56 84 83 60 85 54 5 43 8 84 42 62
Inserer de 36 a pos=9 : 38 54 59 81 71 56 84 83 36 60 85 54 5 43 8 84 42 62
Inserer de 77 a pos=18 : 38 54 59 81 71 56 84 83 36 60 85 54 5 43 8 84 42 77 62
Inserer de 87 a pos=12 : 38 54 59 81 71 56 84 83 36 60 85 87 54 5 43 8 84 42 77 62
Suppression pos=7 : 38 54 59 81 71 56 83 36 60 85 87 54 5 43 8 84 42 77 62
Suppression pos=1 : 54 59 81 71 56 83 36 60 85 87 54 5 43 8 84 42 77 62
Suppression pos=14 : 54 59 81 71 56 83 36 60 85 87 54 5 43 84 42 77 62
Suppression pos=17 : 54 59 81 71 56 83 36 60 85 87 54 5 43 84 42 77
Suppression pos=15 : 54 59 81 71 56 83 36 60 85 87 54 5 43 84 77
Suppression pos=2 : 54 81 71 56 83 36 60 85 87 54 5 43 84 77
Suppression pos=2 : 54 71 56 83 36 60 85 87 54 5 43 84 77
Suppression pos=2 : 54 56 83 36 60 85 87 54 5 43 84 77
Suppression pos=4 : 54 56 83 60 85 87 54 5 43 84 77
Suppression pos=10 : 54 56 83 60 85 87 54 5 43 77
Tableau vide :
```

13.2 Travail à réaliser

1. Téléchargez et décompressez l'archive `TP-TableauDynamique.zip` disponible sur moodle.
2. Complétez le fichier `TableauDynamiqueDEntiers.pas`.
   Privilégier l'ordre suivant : iemeEntier, inserer, ajouter

14 TP Pascal : Fichiers

L'objectif de ce TP est de compléter le mini gestionnaire de contacts vu en cours afin qu'en plus de l'ajout et de l'affichage d'un contact, le programme puisse :

— supprimer un contact ;

— n'afficher que les contacts correspondant à un nom ;

— que l'affichage des contacts se fasse en ordre croissant sur les noms.

Comme vu en cours, le programme devra séparer l'interface homme machine et la logique métier. L'interface homme machine sera toujours en mode texte et les actions de l'utilisateur seront toujours spécifiés dans la ligne de commande. Voici un exemple d'aide qui sera affiché lorsque le nombre d'arguments ne sera pas valide :

```
$ ./contactsTxt
contacts nom_fichier : permet d'afficher l'ensemble des contacts
contacts nom_fichier nom : permet d'afficher un contact
contacts nom_fichier nom prenom : permet de supprimer un contact
contacts nom_fichier nom prenom telephone : permet d'ajouter un contact
```

14.1 Résultats attendus

Voici un exemple d'utilisation du programme une fois qu'il sera terminé :

```
$ ./contactsTxt insa.fic
$ ./contactsTxt insa.fic "Delestre" "Nicolas" "02 32 95 98 76"
$ ./contactsTxt insa.fic
Nom : Delestre
Prenom : Nicolas
Telephone : 02 32 95 98 76
$ ./contactsTxt insa.fic "Malandain" "Nicolas" "02 32 95 98 83"
$ ./contactsTxt insa.fic
Nom : Delestre
Prenom : Nicolas
Telephone : 02 32 95 98 76
Nom : Malandain
Prenom : Nicolas
Telephone : 02 32 95 98 83
$ ./contactsTxt insa.fic "Delporte" "Julien" ""
$ ./contactsTxt insa.fic
Nom : Delestre
Prenom : Nicolas
Telephone : 02 32 95 98 76
Nom : Delporte
Prenom : Julien
Telephone :
Nom : Malandain
Prenom : Nicolas
Telephone : 02 32 95 98 83
$ ./contactsTxt insa.fic "Malandain"
Nom : Malandain
Prenom : Nicolas
Telephone : 02 32 95 98 83
$ ./contactsTxt insa.fic "Delestre" "Nicolas"
$ ./contactsTxt insa.fic
Nom : Delporte
Prenom : Julien
Telephone :
Nom : Malandain
Prenom : Nicolas
Telephone : 02 32 95 98 83
```

14.2 Travail à réaliser

1. Téléchargez et décompressez l'archive `TP-Contact-Sources.zip` disponible sur moodle.

2. Compilez et exécutez le programme de façon à vérifier que l'affichage des contacts et l'ajout d'un contact fonctionne bien.
3. Complétez la procédure `supprimerContact` qui permet de supprimer un contact à partir de son nom et de son prénom. Compilez et testez votre programme pour la suppression d'un contact.
4. Complétez la procédure `parcourirContactsAvecComparaison` qui permet de *traiter* tous les contacts c tels que la fonction `comparer` retourne 0 si elle est exécutée avec comme argument $reference$ et c. Compilez et testez votre programme pour l'affichage des contacts d'un certain nom.
5. Modifiez la procédure `ajouterContact` de façon à ce que l'ajout se fasse dans l'ordre croissant des noms des contacts. Compilez et testez votre programme avec un nouveau carnet de contacts.