API - TP1 (4h)

1 Les nombres aléatoires

Reprendre l'exercice sur les nombres aléatoires (1.10).

a. Ecrire un programme C qui :

- affiche un entier aléatoire appartenant à [0, RANDOM_MAX]
- affiche un entier aléatoire appartenant à [0, 4]
- affiche un entier aléatoire appartenant à [7, 11]
- affiche un réel aléatoire appartenant à [0, 4]
- affiche un réel aléatoire appartenant à [7, 11]

b. Vérifier que deux exécutions du programme donnent bien des nombres différents.

c. Ajouter au code précédent deux fonctions :

aleaEntier(↓ bInf : entier ; ↓ bSup : entier) : entier qui renvoie un entier dans [bInf, bSup]
aleaReel(↓ bInf : réel ; ↓ bSup : réel) : réel qui renvoie un réel dans [bInf, bSup]

Ajouter au programme précédent du code qui :

- affiche un entier aléatoire appartenant à [a, b]
- affiche un réel aléatoire appartenant à [c, d]

avec a, b, c et d des bornes saisies par l'utilisateur et en utilisant les deux fonctions précédentes.

2 Transport de marchandise

Une marchandise peut être envoyée par avion (coût : 6 euros/kg), par bateau (coût : 5 euros/m3) ou par train (coût : 4 euros/kg + 2 euros/ m3).

a. Écrire un module qui calcule le prix à payer pour un colis donné dont on connait le poids et le volume selon le mode de transport. Pour le type de transport on utilisera des constantes comme suit :

```
Constante
    AVION       = 1
    BATEAU      = 2
```

TRAIN = 4

Le paramétrage du Module est :

D : poids du colis (réel), volume du colis (réel) et le type de transport à savoir une des constantes ci-dessus.
R : le prix à payer en fonction des tarifs indiqués dans l'énoncé
D/R : vide

b. Une entreprise de transport basée à Marseille dessert :

- Brest par avion ou bateau (trajet1),
- Paris par avion ou train (trajet 2),
- Nice par train ou bateau (trajet3).

Écrire un algorithme qui calcule, pour un colis donné, sur un trajet donné, le mode de transport le plus avantageux. Voici une proposition d'algorithme :

```
Algorithme Transport

Constante
        AVION       = 1
        BATEAU      = 2
        TRAIN       = 4

Fonction Cout (↓poids, ↓volume : réel ; ↓typeEnvoi : entier) : réel
Début
        … à compléter selon la question a.
Fin

Variable
        poids, volume : réel
        destination : entier
        prixAvion, prixBateau, prixTrain : réel
Début
        Afficher (« Quel est le poids de votre colis ? »)
        Saisir (poids)
        Afficher (« Quel est le volume de votre colis ? »)
        Saisir (volume)
        Afficher (« Quelle est la destination de votre colis parmi les choix suivants ? »)
        ALaLigne
        Afficher (« 1. BREST »)
        ALaLigne
        Afficher (« 2. PARIS »)
        Afficher (« 3. NICE »)
        ALaLigne
        Afficher (« Votre choix (1, 2 ou 3) : »)
        Saisir (destination)
        Selon (destination)
                // BREST (avion / bateau)
```

```
            1 :     prixAvion ←Cout (poids, volume, AVION)
                    prixBateau ←Cout (poids, volume, BATEAU)
                    Si (prixAvion < prixBateau) Alors
                            Afficher (« L'avion est le plus économique : », prixAvion)
                    Sinon
                            Afficher (« Le bateau est le plus économique : », prixBateau)
                    FSi

            // PARIS (avion / train)
            2 :     prixAvion ←Cout (poids, volume, AVION)
                    prixTrain ←Cout (poids, volume, TRAIN)
                    Si (prixAvion < prixTrain) Alors
                            Afficher (« L'avion est le plus économique : », prixAvion)
                    Sinon
                            Afficher (« Le train est le plus économique : », prixTrain)
                    FSi

            // NICE (train / bateau)
            3 :     prixTrain ←Cout (poids, volume, TRAIN)
                    prixBateau ←Cout (poids, volume, BATEAU)
                    Si (prixTrain < prixBateau) Alors
                            Afficher (« Le train est le plus économique : », prixTrain)
                    Sinon
                            Afficher (« Le bateau est le plus économique : », prixBateau)
                    FSi
        FSelon
Fin
```

Traduire cet algorithme en C, le taper et le tester.

3 Le monnayeur

Reprendre l'exercice sur le monnayeur (2.10). Un distributeur de confiseries propose divers produits. Il accepte les pièces de vingt, dix, cinq et un centimes. Écrire un algorithme qui calcule la somme à rendre sous la forme de pièces de 20, 10, 5 et 1 centimes.
Exemple :

coût friandise : 4 pièces de 20 introduites : 0
pièces de 20 présentes : 7 pièces de 10 introduites : 2 (eh oui !)
pièces de 10 présentes : 0 pièces de 5 introduites : 0
pièces de 5 présentes : 2 pièces de 1 introduites : 0
pièces de 1 présentes : 121
Résultat : on rend 0 pièce de 20, 1 pièce de 10, 1 pièce de 5 et 1 pièce de 1.
S'il n'y avait pas eu de pièces de un, on n'aurait pas pu rendre la monnaie.

a. Ecrire un module rendreMonnaie qui calcule un rendu en fonction d'une somme, d'une valeur faciale de pièce et d'un stock. Le stock est mis à jour et le rendu possible indiqué.
   D : valeur faciale de la pièce à considérer (entier)
   D/R :

- somme à rendre comme donnée (entier), cette somme est modifiée en fonction du rendu possible
- stock des pièces indiquées comme donnée (entier), le stock est modifié en fonction du rendu

R : rendu à partir du stock en nombre de pièces (entier)

Exemple : considérons une valeur faciale de 10ct et une somme à rendre de 92 cts. Avec un stock de 5 pièces, le rendu serait de 5 et le stock serait modifié à 0. Avec un stock de 12 pièces, le rendu serait de 9 et le stock serait modifié à 3.

Testez votre module.

b. Voici une proposition d'algorithme :

```
Algorithme Monnayeur

Procedure rendreMonnaie (↓valeurFaciale, ↕sommeARendre, ↕stock, ↑rendu : entier)
Début
        … à compléter selon la question a.
Fin

Variable
        cout : entier
        p20, p10, p5, p1 : entier
        i20, i10, i5, i1 : entier
        s20, s10, s5, s1 : entier
        r20, r10, r5, r1 : entier
        rendu : entier

Début
        Afficher (« coût friandise : »)
        Saisir (cout)
        Afficher (« pièces de 20 présentes : »)
        Saisir (p20)
        Afficher (« pièces de 10 présentes : »)
        Saisir (p10)
        Afficher (« pièces de 5 présentes : »)
        Saisir (p5)
        Afficher (« pièces de 1 présentes : »)
        Saisir (p1)
        Afficher (« pièces de 20 introduites : »)
        Saisir (i20)
        Afficher (« pièces de 10 introduites : »)
        Saisir (i10)
        Afficher (« pièces de 5 introduites : »)
        Saisir (i5)
        Afficher (« pièces de 1 introduites : »)
        Saisir (i1)
```

```
        s20 ← p20 + i20
        s10 ← p10 + i10
        s5 ← p5 + i5
        s1 ← p1 + i1
        rendu ← (i20*20 + i10*10 + i5*5 + i1*1) - cout

        // on supposera que le rendu est positif ou nul …

        rendreMonnaie (20, rendu, s20, r20)

        rendreMonnaie (10, rendu, s10, r10)

        rendreMonnaie (5, rendu, s5, r5)

        rendreMonnaie (1, rendu, s1, r1)

        Si (rendu > 0) Alors
                // il reste à rendre quelque chose le rendu est donc impossible
                Afficher (« Le rendu est impossible
        Sinon
                // Le rendu est vide, il est donc possible de rendre la monnaie
                // avec le stock de pièce du monnayeur
                Afficher (« On rend »,          r20, « pièce(s) de 20 »,
                                                r10, « pièce(s) de 10 »,
                                                r5, « pièce(s) de 5 »,
                                                r1, « pièce(s) de 1 »).
                // On met à jour le monnayeur
                p20 ←s20
                p10 ←s10
                p5 ←s5
                p1 ←s1
        FSi
Fin
```

Traduire cet algorithme en C, le taper et le tester au moins avec l'exemple du cours.

4 Nombre …

Traduire et taper en C l'algorithme suivant :

```
Algo CreationNombre

fonction creerEntier(↓ a : entier ; ↓b : entier) : entier
Variables
        resu, m, u : entier
debut
        m ←   b div 10
        u ←  b mod 10
        resu ← m*1000 + a*10 + u
```

```
        return resu
fin

Variables
        n1, n2, r : entier

debut
        afficher (« Donnez deux nombres compris entre 10 et 99 : »)
        saisir(n1, n2)
        r ← creerEntier(n1,n2)
        afficher(r)
fin
```

Que fait cet algorithme ?

5 Miroir …

Ecrire et traduire en C un algorithme qui « retourne » un nombre n saisi compris entre 0 et 999.
Exemples :

- 178 donnera 871
- 32 (032) donnera 230
- 100 donnera 1 (001)

S'inspirer de l'exercice sur l'extraction des chiffres de la centaine, de la dizaine et de l'unité vu en EI en réalisant une procédure dont l'en-tête est :

Procédure extraction (↓ n, ↑ centaine, ↑ dizaine, ↑ unité : entier)

En utilisant la procédure précédente, réaliser le retournement à partir d'une fonction et d'une procédure dont les en-tête sont :

Fonction miroir (↓ n : entier) : entier

Procedure miroir (↕ n : entier)

6 Polynôme du second degré à l'aide de fractions

On propose d'utiliser les structures suivantes :

```
Type
        TFraction =     Structure
                                num, den : entier
                        FinStructure
        TPolynomeDegre2 =  Structure
                                        a, b, c : TFraction
                                FinStructure
```

Reprendre le travail fait en EI et implémenter les modules pour gérer les fractions de type TFraction.

Ecrire un module qui construit un polynôme de type TPolynomeDegre2 à partir de 3 fractions de type TFraction.

Ecrire un module qui affiche un polynôme de type TPolynomeDegre2.

Ecrire un module qui dérive un polynôme de type TPolynomeDegre2. Faire 2 versions :

- Une fonction qui prend en entrée un TPolynomeDegre2 et retourne un TPolynomeDegre2.
- Une procédure qui prend en donnée / résultat un TPolynomeDegre2.

Ecrie et traduire en langage C un algorithme qui teste tout cela.