L1 – Algorithmique 2 – Période 3

Exercices[1]

1 Récursivité

1.1 Fonction mystère

Détailler l'exécution du sous-programme suivant, pour une valeur de paramètre `n = 4`. Que calcule-t-il ?

```
int c (int n)
{
    if (n == 1) return 1 ;
    else return c(n - 1) + 2 * (n - 1) + 1;
}
```

1.2 Récursivité indirecte

Écrire des fonctions `pair` et `impair` qui déterminent si un nombre **n** est pair / impair, sans utiliser les opérateurs de la division euclidienne mais le principe de récursivité indirecte.

1.3 PGCD

Écrire une fonction récursive qui retourne le plus grand commun diviseur de deux entiers.

1.4 Puissance

On rappelle ci-dessous une écriture récursive de la fonction puissance :

```
float puissance (float x, int n)
{
    if (n == 0) return 1;
    else return x * puissance(x,n-1);
}
```

1. Modifier cette fonction afin de traiter le cas où **n** est négatif.
2. L'algorithme d'*exponentiation rapide* est basé sur la propriété suivante :

$$x^n = x^{\lfloor n/2 \rfloor^2} \times x^{n \bmod 2}$$

   Modifier la fonction puissance en utilisant cette propriété. Expliquer pourquoi ceci rend son exécution plus efficace dans le cas général en détaillant une exécution avec une valeur de **n** appropriée.

1.5 Sous-programmes récursifs simples [TP]

1. Écrire une procédure récursive qui affiche **n** fois le message `bonjour`.
2. Écrire une fonction récursive qui calcule la somme des **n** premiers carrés.

[1] Légende : ⓐ Application directe du cours. [TP] Exercice à tester sur machine. Dans les exemples d'exécution, les valeurs en gras indiquent des lectures au clavier.

1.6 Nombres premiers [TP]

1. Écrire une fonction entière récursive `diviseur` qui détermine le plus petit entier strictement supérieur à `d`, qui divise un entier `n`.
   → On suppose $1 \leq$ `d` $<$ `n`. `n` et `d` sont dans cet ordre les deux paramètres du sous-programme.
   → Ex : `diviseur(10,3)` vaut 5 ; `diviseur(13,1)` vaut 13 ; `diviseur(8,2)` vaut 4 ; `diviseur(15,1)` vaut 3.
2. En déduire une fonction booléenne `est_premier` qui détermine si un nombre strictement positif est premier.
3. Écrire une procédure récursive `affiche_premiers` qui affiche tous les nombres premiers inférieurs à un entier `n`.

1.7 Recherche d'un zéro d'une fonction

La *méthode de dichotomie* est une technique mathématique permettant de rechercher la valeur approchée d'un zéro d'une fonction, c'est-à-dire un nombre x tel que $f(x) \approx 0$.

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction définie et continue sur $[a, b]$, et telle que $f(a)$ et $f(b)$ sont de signes différents (c'est-à-dire $f(a) \times f(b) < 0$). La méthode de dichotomie consiste à poursuivre la recherche du zéro dans le demi-intervalle préservant cette propriété, et ainsi de suite jusqu'à ce que l'intervalle en question soit suffisamment petit.

Écrire une fonction récursive déterminant par la méthode de dichotomie une approximation à ϵ près (avec $0 < \epsilon < 0.1$) d'une valeur $x_0 \in [a, b]$ telle que $f(x_0) \approx 0$, c'est-à-dire une valeur x telle que $|x - x_0| \leq \epsilon$. On pourra e.g. considérer $f(x) = 4 \times x \times (4 \times x \times (3 \times x - 1) - 8) - 1$.

1.8 Tableaux et chaines de caractères

1. Écrire une procédure récursive de saisie d'un tableau de `n` entiers.
2. Écrire une fonction récursive qui détermine si une valeur appartient à un tableau d'entiers.
3. Sur la même base, écrire une fonction récursive qui recherche la position d'une valeur dans un tableau d'entiers. La fonction retournera -1 si la valeur est absente.
4. Écrire une fonction récursive qui détermine si une lettre (caractère) apparait ou non dans un mot (chaine de caractères) à partir d'une certaine position

1.9 Chaines de caractères [TP]

Écrire les fonctions récursives effectuant les traitements suivants :

1. Extraire une sous-chaine d'une chaine de caractères, étant donnés un indice de départ et une longueur.
2. Tester si une chaine est un palindrome.
3. Compter le nombre de d'occurrences d'un caractère dans une chaine.

2 Enregistrements

2.1 Ensembles de points

On désire représenter des points dans le plan. Chaque point est codé sous la forme d'un enregistrement contenant une coordonnée `x` et une coordonnée `y`.

1. Définir un type structuré point permettant de représenter un point.

2. Écrire une fonction déterminant si deux points sont confondus.
3. Écrire une fonction qui calcule la distance euclidienne entre deux points.
4. Écrire une fonction qui calcule le milieu d'un segment constitué de deux points.

On désire maintenant représenter des ensembles de points. Pour cela on stockera des points dans un tableau.

5. Définir un type enspoints représentant un ensemble de points sous forme d'un enregistrement composé d'un tableau et de sa taille effective.
6. Écrire une procédure de saisie d'un ensemble de points.
7. Écrire une fonction calculant le centre de gravité d'un ensemble de points.
8. Écrire une fonction plusproche qui retourne, parmi un ensemble de points passé en paramètre, le point le plus proche de l'origine (0,0).
9. Écrire une fonction distmin qui calcule la plus petite distance entre deux points d'un ensemble de points (non nécessairement consécutifs).
10. Écrire une fonction qui détermine si les points d'un ensemble (tableau) sont ordonnés selon leur abscisse.

2.2 Fractions [TP]

Cet exercice consiste à manipuler des nombres fractionnaires sans passer par des valeurs décimales. Tout au long de l'exercice, il faut compléter le programme principal de manière à tester et utiliser les différents sous-programmes.

1. Définir un type structuré fraction permettant de représenter une fraction.
2. Écrire une procédure `saisie` qui permette de saisir une fraction au clavier (sous forme de deux entiers). On ne vérifiera pas ici que le dénominateur est valide (différent de 0).
3. Écrire une procédure `affichage` qui affiche une fraction (par exemple : `2/5`).
4. Écrire une fonction `mult` qui multiplie deux fractions (sans simplifier).
5. Écrire une fonction `add` qui somme deux fractions (sans simplifier).
6. Écrire les fonctions `opp` et `inv`, qui renvoient respectivement l'opposé et l'inverse de la fraction passée en paramètre.
7. Écrire les fonctions `soustr` et `div`, qui réalisent respectivement la soustraction et la division de deux fractions.
8. Écrire une fonction `pgcd` qui calcule le plus grand commun diviseur de deux entiers.
9. En utilisant la fonction `pgcd`, écrire une fonction `simpl` qui simplifie la fonction passée en paramètre en la rendant irréductible, et en imposant un dénominateur positif.
10. Modifier `mult`, `add`, `soustr` et `div` afin qu'ils renvoient une fraction simplifiée.

2.3 Gestion de prêt d'une bibliothèque [TP]

Dans une petite bibliothèque d'école, un inscrit est identifié par un numéro de lecteur, un nom et un prénom.

1. Définir un type structuré `lecteur` permettant de représenter une personne inscrite.
2. Écrire un sous-programme permettant d'effectuer la `saisie` des informations sur un lecteur.
3. Écrire un sous-programme qui `affiche` les informations propres à un lecteur.
4. Définir un type `enslecteurs` comme un enregistrement composé d'un tableau de lecteurs et du nombre effectif de lecteurs. Une variable de type enslecteurs représentera un ensemble de lecteurs.
5. Écrire un sous-programme qui `initialise` ou vide un ensemble de lecteurs.
6. Écrire un sous-programme qui `ajoute` un nouveau lecteur à un ensemble de lecteurs.
7. Écrire un sous-programme `saisietous` qui permet la saisie d'un ensemble de lecteurs.
8. Écrire un sous-programme `affichetous` qui affiche tous les lecteurs d'un ensemble de lecteurs.
9. Compléter le programme principal afin d'exécuter une saisie et un affichage d'un ensemble de lecteurs.
10. Écrire un sous-programme `rechercheNom` qui, à partir d'un ensemble de lecteurs et d'un numéro, retourne le nom du lecteur identifié par ce numéro.

Un livre est identifié par un numéro ISBN, un titre et un nom d'auteur. Un variable de type bibliothèque contient un ensemble de livres et un ensemble de lecteurs, et permet la gestion des emprunts.

11. Définir les types `livre`, `enslivres`, et `biblio` permettant de représenter respectivement un livre, un ensemble de livres, et une bibliothèque. On doit savoir, pour chaque livre, s'il est emprunté ou pas, et qui l'a emprunté (en dernier).
12. Écrire un sous-programme qui affiche tous les emprunts en cours d'une bibliothèque : noms des livres empruntés et des emprunteurs.

3 Pointeurs

3.1 Étude de programmes @

1. Suivre le déroulement du programme A à l'aide d'une représentation schématique de l'état des variables, afin de déterminer quelles sont les valeurs successives prises par les variables entières, et sur quelles variables pointent les variables pointeurs.
2. Représenter schématiquement l'état des variables au cours de l'exécution du programme B. Donner l'affichage produit par l'exécution. Compléter le programme de manière à libérer explicitement la mémoire allouée dynamiquement.
3. Déterminer l'affichage produit par l'exécution du programme C.

```
// Programme A

int main ()
{
  int * p1;
  int * p2 ;
  int i = 1 ;
  p1 = &i ;
  *p1 = 2 ;
  p2 = new int ;
  *p2 = 3 ;
  p1 = p2 ;
   *p1 = i+2 ;
  delete p1 ;
  p1 = new int ;
  *p1 = 5 ;
  delete p1 ;
  return 0 ;
}
```

```
// Programme B

#include <iostream>

int main ()
{
  int * x;
  int * y;
  int * z ;
  x = new int ;
  y = new int ;
  z = y ;
  *y = 3 ;
  *x = *y ;
  *z = *x + *y ;
   *x = 2**y ;
  std::cout << *x << ","
    << *y << "," << *z ;
  // (compléter)
  return 0 ;
}
```

```
// Programme C

#include <iostream>

void elle (char & c)
{
   c = 'l' ;
}

void aime (char * c)
{
   *c = 'm';
}

int main ()
{
   char x ;
   char * y;
   char * z;
   elle(x) ;
   std::cout << x << std::endl ;
   y = new char ;
   aime(y) ;
   std::cout << *y << std::endl ;
   z = y ;
   y = &x ;
   std::cout << x << *y
     << *z << std::endl ;
   aime(y) ;
   std::cout << x << *y
     << *z << std::endl ;
   return 0 ;
}
```

3.2 Tableaux alloués dynamiquement [TP]

On dispose d'un ensemble de données sur des films (titre, année, réalisateur, durée, langue, et éventuellement d'autres informations) et l'on souhaite pouvoir construire des sélections de films selon différents critères sans dupliquer l'ensemble des informations. Un ensemble de films sera représenté par une structure composée d'un champ de type `film*` (permettant de déclarer un tableau de films alloué dynamiquement) et d'un entier (nombre de films). Une sélection sera une structure similaire, avec comme premier champ l'adresse d'un tableau de pointeurs vers des films à la place d'un tableau de films.

1. Définir les types `film`, `ensemble` et `selection`.
2. Écrire les sous-programmes appropriés permettant d'importer depuis un fichier texte les informations relatives à un ensemble de films. Le fichier devra comporter le nombre de films sur la première ligne, puis chaque information sur une ligne séparée. Par exemple :

```
3
Sueurs froides
1958
Alfred Hitchcock
128
anglais
Taxi Driver
1976
Martin Scorsese
110
anglais
Parasite
2019
Bong Joon-ho
132
coréen
```

3. Écrire un sous-programme qui construit la sélection comprenant l'intégralité des films d'un ensemble de films.

4. Écrire un sous-programme qui demande à l'utilisateur un réalisateur (ou la chaîne "X" pour tout réalisateur), une année minimale et maximale, une durée minimale et maximale, et qui construit la sélection de films en conséquence, à partir d'un ensemble de films.

5. Écrire un sous-programme qui affiche une sélection, avec un film par ligne. Par exemple :

```
Sueurs froides (Alfred Hitchcock, 1958)
Parasite (Bong Joon-ho, 2019)
```

6. Écrire un sous-programme permettant d'enregistrer une sélection dans un fichier texte, selon la syntaxe présentée question 2.

7. Écrire une fonction calculant le nombre de films en commun, étant données deux sélections de films.

8. Écrire un sous-programme permettant de trier une sélection par ordre chronologique des films.