Exercice 1 – Variables dynamiques

a) Suivre le déroulement du programme A en précisant quelles sont les valeurs successives de i, *p1, *p2, et sur quelles variables pointent p1 et p2.

b) 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.

c) Déterminer l'affichage produit par l'exécution du programme C.

Programme A

```
int main ()
{
    int *p1, *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, *y, *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, *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 ;
}
```

Exercice 2 – Listes chainées d'entiers

Définir une structure de données permettant de manipuler des listes d'entiers, puis écrire les sous-programmes suivants :

a) Initialiser une liste ;
b) Déterminer si une liste est vide ;
c) Afficher les éléments d'une liste (2 versions : itératif et récursif) ;
d) Déterminer la longueur d'une liste ;
e) Ajouter un entier *n* en tête de liste ;
f) Ajouter un entier *n* en fin de liste ;
g) Ajouter un entier *n* dans une liste L après l'élément d'adresse *adr* donnée, ou en fin de liste si *adr* n'est pas dans L ;
h) Tester si un entier *n* appartient à une liste ;
i) Retourner l'adresse de l'élément contenant la première occurrence d'un entier *n* ;
j) Ajouter dans une liste L un entier *n* après la première occurrence d'un entier *x* donné, ou en fin de liste si *x* n'est pas dans L ;
k) Supprimer le premier élément d'une liste ;
l) Supprimer la première occurrence d'un entier *n* dans une liste ;
m) Vider une liste ;
n) Compter le nombre d'occurrences d'un entier *n* dans une liste L ;
o) Vérifier si une liste d'entiers est triée dans l'ordre croissant ;
p) Vérifier si une liste d'entiers est strictement monotone (croissante ou décroissante) ;
q) Afficher les éléments de rang impair d'une liste (le premier, le troisième, le cinquième, ...) ;
r) Vérifier si une liste est sans doublons ;
s) Fusionner deux listes triées dans l'ordre croissant, en créant une troisième liste triée dans l'ordre croissant et contenant les mêmes valeurs que les listes initiales.

Exercice 3 – Crible d'Eratosthène

Écrire un programme permettant d'afficher la liste des nombres premiers inférieurs ou égaux à un entier *n* donné. La méthode devra consister à construire une liste chainée d'entiers éligibles (ordonnés du plus petit au plus grand), puis de supprimer, pour chaque élément de la liste, tous leurs multiples stricts.

Exercice 4 – Polynômes

On veut écrire des fonctions et procédures permettant de manipuler des polynômes. Un polynôme p est une somme de n monômes : $p = m_1 + \ldots + m_n$; un monôme est de la forme $a.x^b$ (un coefficient a et une puissance b)

a) Définir les structures de données **monome** et **polynome**, telles qu'un polynôme soit représenté comme une liste chainée de monômes.

b) Écrire une procédure **insererEnTete (polynome &p, monome m)** qui insère en tête le monôme m dans le polynôme p.

c) Écrire une fonction **degre** qui retourne le degré d'un polynôme passé en paramètre (la plus grande puissance des monômes).

d) Écrire une fonction réelle **valeurEn (polynome p, float x)** qui calcule la valeur d'un polynôme en fonction d'une valeur x passée en paramètre. On utilisera une fonction puissance définie par ailleurs.

e) Écrire une fonction **derive (polynome p)** qui calcule et retourne le polynôme dérivé de p.

f) Écrire une procédure **ajoutMonome (polynome &p, monome m)** qui ajoute le monôme m au polynôme p.

g) Écrire une fonction **somme (polynome p1, polynome p2)** qui calcule et retourne la somme de deux polynômes.

h) Écrire une fonction **multMonome (polynome p, monome m)** qui calcule et retourne le produit du polynôme p par le monôme m (attention au cas où le coefficient de m est nul).

i) [Difficile] Écrire une fonction **produit (polynome p1, polynome p2)** qui calcule et retourne le produit de deux polynômes p1 et p2. Il est possible de définir des sous-programmes annexes pour réaliser ce traitement.

Exercice 5 – File d'attente

Une file est une structure de données où les éléments sont systématiquement ajoutés en queue et supprimés en tête.

a) Définir les structures permettant de manipuler des variables de type file d'attente de personnes. Une personne est représentée par un nom et un prénom.

b) Écrire des primitives relatives aux files : initialise, est_vide, affiche, ajoute, enlève.

c) Écrire une fonction qui teste si deux files contiennent les mêmes personnes, dans le même ordre.

d) Finaliser le programme de manière à pouvoir manipuler successivement deux files d'attente (ajouter et enlever des personnes, afficher la liste des personnes, …), puis de tester si les deux files sont identiques.

Exercice 6 – Arbre binaire de caractères

Un arbre binaire de caractères est une structure de données où chaque élément appelé nœud est composé d'un caractère et de deux références (gauche, droite) vers d'autres nœuds appelés descendants ou fils. Aucun cycle ne doit apparaitre dans les listes de références successives.

a) Écrire une fonction qui teste si un arbre est une feuille (ie. un nœud n'ayant aucun descendant).

b) Écrire des procédures permettant d'afficher les caractères de l'arbre, selon un parcours respectivement infixé, préfixé, postfixé.

c) Écrire une fonction qui retourne la taille de l'arbre (ie. le nombre de nœuds présents dans l'arbre).

d) Écrire une fonction qui recherche si un caractère est présent dans un arbre.

e) Écrire deux fonctions min et max retournant respectivement le premier et dernier caractère d'un arbre, suivant l'ordre alphabétique.

f) Écrire une fonction qui teste si l'arbre est ordonné (ie. si les caractères de l'arbre sont placés, selon un parcours infixé, en ordre croissant).

g) Écrire une fonction qui retourne la hauteur de l'arbre.

h) Écrire une fonction qui teste si un arbre binaire est entier, c'est-à-dire qu'aucun nœud ne possède qu'un seul descendant non vide.

i) Écrire une fonction qui teste si un arbre binaire est parfait, c'est-à-dire que toutes les feuilles sont au même niveau.

j) Écrire une procédure qui affiche le mot de l'arbre, chaine formée par les feuilles avec un sens de lecture de gauche à droite.

k) Écrire une fonction qui teste si deux arbres sont égaux.