Info 4A

Langage C et C++, L2
Examen : première session, année 2018-2019

Modalités
Documents autorisés : trois feuilles A4 recto-verso manuscrites ou imprimées. Ordinateurs et smartphones interdits.

Partie 1 : scalaires, bitwises

A1 (10%)
Donnez les affichages réalisés par l'exécution de la fonction suivante.

```
void bw()
{
    printf("%x\n", 0xF0 | 0xBA);
    printf("%x\n", 0x0A << 0x0A);
    printf("%c\n", 'F' - 'A' + 'b');
    printf("%x\n", (1<<5) | (0x1000>>3));
}
```

A2 (5%)
On dit (dans le cadre de cet exercice) que deux mots binaires p et q sont *indépendants* si et seulement si il n'existe aucun entier i tel que p et q ont tous les deux un bit à 1 en position i. Par exemple, 00110001 et 11001000 sont indépendants, alors que 11110000 et 00010000 ne l sont pas, car en position 4 (numérotées sans perte de généralité de gauche à droite en partant de 0), il y a un bit à 1 dans les deux mots binaires.
Réalisez une fonction int `sb(int x, int y)` qui retourne un Booléen (représenté par un entier) ayant la valeur vrai si et seulement si les représentations binaires de x et y sont indépendantes au sens expliqué ci dessus. Vous ne devez utiliser ni boucles (while, for...) ni récursivité.

Partie 2 : Tableaux, chaines

B1 (5%)
La fonction suivante accepte en paramètres un caractère `c` et un entier `base`, dont la valeur est supposée comprise entre 2 et 10, qui désigne une base dans laquelle un entier peut être représenté à l'aide de chiffres compris entre 0 et base-1.

```
int conv(char c, int base)
{
    int chiffre = c-'0';
    if((chiffre>=base) || (chiffre<0))
    {
        return -1;
    }
    else
    {
        return chiffre;
    }
}
```

Donnez (dans l'ordre) les valeurs retournées par les appels suivants :
`conv('2',3)` `conv('3',3)` `conv('a',3)` `conv('0',3)`

B2 (10%)
Réalisez une fonction `int trans(const char* s, int b)`, dans laquelle s désigne une chaîne de caractères et b est un entier supposé avoir une valeur comprise entre 2 et 10, et qui a le comportement suivant :
Si s désigne une représentation valide d'un entier en base b, alors la valeur retournée est cet entier, sinon la valeur retournée est -1.
Par exemple, `trans("1021",3)` retourne 34, qui est l'entier dont la représentation en base 3 est 1021. Mais `trans("1321",3)` retourne -1 car `1321` n'est pas une représentation valide en base 3.

La fonction `trans` *doit* appeler la fonction `conv` introduite dans la question précédente.

Partie 3 : Structures, variables dynamiques

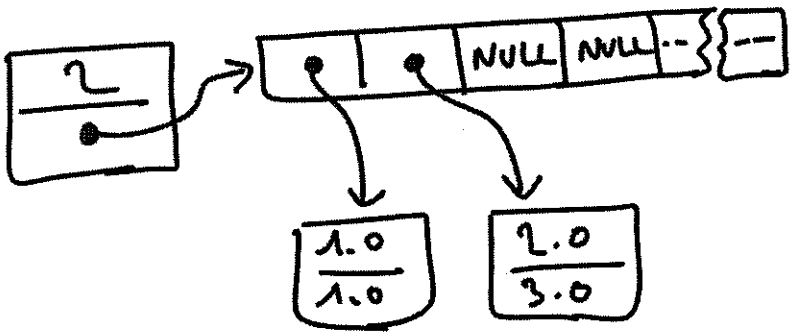
On définit la constante `MAX` et les structures `point` et `panel` de la manière suivante :

```
#define MAX 100

typedef struct
{
    double x;
    double y;
}point;

typedef struct
{
    int n;
    point** tab;
}panel;
```

La structure `point` permet de représenter des points ayant deux coordonnées de type `double`. La structure `panel` permet de représenter une liste de points appelée *panel de points*. Son champ `tab` désigne un tableau comportant `MAX` cellules contenant chacune soit `NULL`, soit l'adresse d'une instance de point. Son champ `n` indique le nombre de points dans le panel (dont les adresses sont dans le tableau désigné par `tab`, aux indices 0 à n-1). Voici un exemple de représentation d'un panel de 2 points.



C1 (10%)
Réalisez une fonction `panel* createPanel()` qui crée dans le tas un panel vide de capacité maximale `MAX` et qui retourne l'adresse de l'instance de la structure `panel` représentant le panel créé.

C2 (10%)
Réalisez une fonction `void delPanel(panel* pan)` qui détruit le panel désigné par pan et libère *toute* la mémoire occupée dans le tas par ce panel.

C3 (10%)
Réalisez une fonction `void addPoint(panel* pan, double x, double y)` qui ajoute au panel désigné par `pan` un nouveau point de coordonnées (x,y). On *suppose* que le nombre de points déjà dans le panel est strictement inférieur à `MAX`.

Partie 4 : Classes, gestion de la mémoire (C++)

La classe suivante représente une association (un couple) de deux chaînes de caractères.

```
class Coupling
{
private:
    string a;
    string b;

public:
    Coupling(string a, string b)
    {
        this->a = a;
        this->b = b;
    }

    Coupling()
    {
        this->a = "";
        this->b = "";
    }

    void print()
    {
        cout << "(" << a << "," << b << ")" << endl;
    }
};
```

La classe suivante représente un dictionnaire, c'est à dire une liste d'associations représentées à l'aide de la classe `Coupling`.

```
class Dico
{
private:
    Coupling* tab;
    int capa;
    int n;

public:
    Dico(int capa)
    {
        this->capa = capa;
        this->n = 0;
        this->tab = new Coupling[capa];
    }

    ~Dico()
    {
        delete[] tab;
    }
}
```

La fonction suivante crée un nouveau tableau de taille `size` (supposée supérieur ou égal à n), copie dans ce tableau les n premiers éléments du tableau désigné par `tab`, et retourne l'adresse du tableau créé.

```
Coupling* xdup(Coupling* tab, int n, int size)
{
    Coupling* newTab = new Coupling[size];
    for(int i=0; i<n; i++)
    {
        newTab[i] = tab[i];
    }
    return newTab;
}
```

D1 (10%)
Réalisez une méthode `void add(string a, string b)` de la classe `Dico` permettant d'ajouter une nouvelle association de deux chaînes (des copies de celles désignées par les paramètres a et b) au dictionnaire courant. Si le nombre n d'associations déjà présentes dans le dictionnaire est égal à la capacité `capa` de ce dictionnaire, alors la capacité du dictionnaire doit être augmentée de 10. Cette augmentation de capacité suppose de créer un nouveau tableau de stockage plus grand que l'ancien et d'y recopier le contenu de l'ancien tableau dans le nouveau. A cette fin, vous *devez* utiliser la fonction `xdup` introduite précédemment.

D2 (10%)
Redéfinissez l'opérateur d'affectation `Dico& operator=(const Dico& m)` de la classe `Dico` de manière à respecter les bonnes pratiques de programmation en C++ (copie en profondeur du modèle, pas de fuite ni d'accident mémoire). Vous pouvez utiliser la fonction `xdup` introduite précédemment.

Partie 5 : Structures de données chaînées (C++)

La classe suivante permet de représenter des arbres pouvant avoir des noeuds unaires, des noeuds binaires et des feuilles. Chaque noeud et feuille a une étiquette, qui est une chaîne de caractères.

```
class Etree
{
private:
    string label;
    Etree* f1;
    Etree* f2;

public:
    Etree(string label)
    {
        this->label = label;
        f1 = NULL;
        f2 = NULL;
    }

    Etree(string label, Etree* f)
    {
        this->label = label;
        f1 = f;
        f2 = NULL;
    }

    Etree(string label, Etree* g, Etree* d)
    {
        this->label = label;
        f1 = g;
        f2 = d;
    }

    ~Etree()
    {
        if(f1!=NULL) delete f1;
        if(f2!=NULL) delete f2;
    }
```

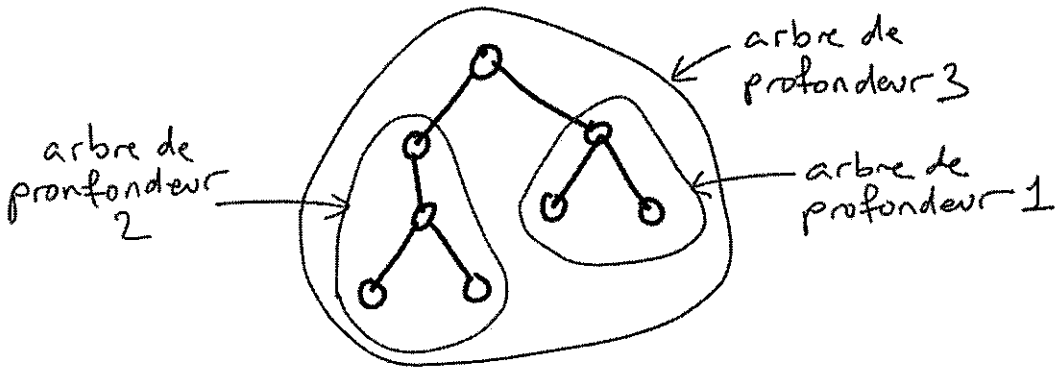
```
    void print()
    {
        if(f1==NULL)
        {
            cout << "<" << label << ">";
        }
        else if(f2==NULL)
        {
            cout << label << "->" << "["; f1->print(); cout << "]";
        }
        else
        {
            cout << label << "->" << "("; f1->print(); cout << ";";
                  f2->print(); cout << ")";
        }
    }
}
```

E1 (6%)
Donnez l'affichage produit par l'exécution des lignes suivantes.

```
Etree* a1 = new Etree("aaa");
Etree* a2 = new Etree("bbb");
Etree* a3 = new Etree("ccc");
Etree* a4 = new Etree("eee",a1);
Etree* a5 = new Etree("fff",a2,a3);
Etree* a = new Etree("ggg",a4,a5);
a->print();
```

E2 (7%)
La profondeur d'un arbre est définie de la manière suivante :
- La profondeur d'une feuille est 0.
- La profondeur d'un arbre qui n'est pas une feuille est 1 plus le maximum des profondeurs des fils de de la racine de cet arbre.

Voici la représentation d'un arbre de profondeur 3 dont la racine a un fils de profondeur 2 et un autre de profondeur 1.



Réalisez une méthode `int depth()` qui retourne la profondeur de l'arbre courant.

E3 (7%)
On dit qu'un arbre est binaire si et seulement si une des deux conditions suivantes est réalisée :
- L'arbre est une feuille.
- La racine de l'arbre a exactement deux fils et chacun de ces fils est un arbre binaire.

En d'autres termes, un arbre binaire est un arbre qui ne contient que des feuilles et/ou des noeuds binaires.

Réalisez une méthode `int binary()` qui retourne 1 si l'arbre courant est binaire, et 0 sinon.

Langage C et C++, L2
Examen : première session, année 2018-2019

Numéro anonymat :

A1 (10%)

A2 (5%)

```
int sb(int x, int y)
{



}
```

B1 (5%)

B2 (10%)

```
int trans(const char* s, int b)
{
    int n = (int)strlen(s);
    int r = 0;
    for(int i=0; i<n; i++)
    {
        int chiffre =
        if(chiffre<0)
        {
            return -1;
        }


    }
    return r;
}
```

C1 (10%)

```
panel* createPanel()
{









}
```

C2 (10%)

```
void delPanel(panel* pan)
{




}
```

C3 (10%)

```
void addPoint(panel* pan, double x, double y)
{




}
```

D1 (10%)

```
void Dico::add(string a, string b)
{









}
```

D2 (10%)

```
Dico& Dico::operator=(const Dico& m)
{



}
```

E1 (6%)

E2 (7%)

```
int Etree::depth()
{



}
```

E3 (7%)

```
int Etree::binary()
{



}
```