Examen info4A - session 1 - mai 2022

Université de Bourgogne - UFR Sciences et Techniques - Deuxième année de licence

Documents autorisés : 4 feuilles A4 recto verso, manuscrites et/ou imprimées.

Appareils électroniques programmables et / ou communiquants interdits.

Partie A

A1 (2 points)

Soit la fonction f qui :

- À tout entier x impair associe x.
- À tout entier x pair associe la valeur obtenue en divisant x par 2 jusqu'à obtenir une valeur impaire.

Par exemple,

- $f(5) = 5$
- $f(36) = 9$ (En divisant 36 par 2 on obtient 18, qui est pair. On divise donc à nouveau 18 par 2 et on obtient 9.)

Formellement :

- Pour tout entier positif impair x, $f(x) = x$.
- Pour tout entier positif pair x, $f(x) = f(x/2)$.

Définissez une fonction...

```
int f(int x);
```

...qui retourne la valeur de $f(x)$. Cette fonction peut s'écrire très simplement de manière itérative (avec une boucle), mais une version récursive sera acceptée si elle est correcte.

Partie B

B1 (1 point)

Soit la fonction suivante :

```
uint8_t decLeft(uint8_t data, uint8_t b0)
{
      return (data << 1) | b0;
}
```

Donnez en binaire la valeur de la variable `x` à l'issue de l'exécution des lignes suivantes :

```
uint8_t x = 0b00110011;
x = decLeft(x,0);
x = decLeft(x,1);
```

B2 (2 points)

Complétez la définition de la fonction...

```
uint16_t g()
{
    uint16_t out = 0;
    for(int i=0; i<16; i++)
    {

        if(rand()%3 == 0)
        {

        }
    }
    return out;
}
```

...qui produit un mot binaire aléatoire dans lequel chaque bit à une probabilité 1/3 d'avoir la valeur 1 et une probabilité 2/3 d'avoir la valeur 0.

Partie C

C1 (1 point)

Soit la fonction suivante :

```
void h(int* p, int* q)
{
    if(*p > *q) *p = *q;
    else *q = *p;
}
```

Donnez l'affichage produit par l'exécution des lignes suivantes :

```
int t[] = {1,3,5,6,7,8,9,13};
int r = 4,
h(&r,t+2); h(&r,t+3); h(t,t+1);
for(int i=0; i<8; i++) printf("%d ",t[i]);
```

C2 (2 points)

Définissez la fonction...

```
void sums(const int* tab, int n, int* s1, int *s2);
```

...qui place dans la variable pointée par `s1` la somme des `n` premières valeurs du tableau désigné par le paramètre `tab` et qui place dans la variable pointée par `s2` la somme des carrés des `n` premières valeurs de ce tableau.

Par exemple, si le tableau contient les valeurs 1, 2, 3 et que `n` vaut 3, alors à l'issue de l'exécution de la fonction, la variable pointée par `s1` contiendra 6 et la variable pointée par `s2` contiendra 14.

On suppose que, lors de l'appel de cette fonction, les arguments `s1` et `s2` pointent des variables de type `int`, dont les valeurs initiales sont sans importance.

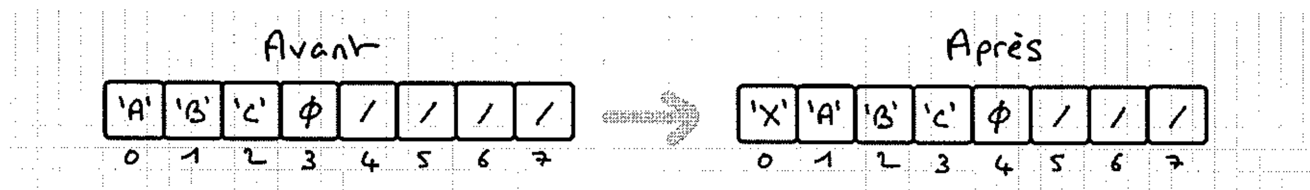
Partie D

D1 (0,5 point)

Le tableau de caractères `s` est initialisé de la manière suivante :

```
char s[10] = "ABC";
```

On souhaite insérer le caractère `'x'` au début de la chaîne contenue dans ce tableau. Voici une figure représentant le contenu du tableau avant et après cette insertion. Les valeurs situées dans les cases marquées d'un trait incliné n'ont pas d'importance.



Complétez les lignes de codes suivantes de manière à ce que leur exécution ait pour effet cette insertion.

```
s[   ] = s [   ];
s[   ] = s [   ];
s[   ] = s [   ];
s[   ] = s [   ];
s[   ] =       ;
```

D2 (1,5 points)

Définissez la fonction...

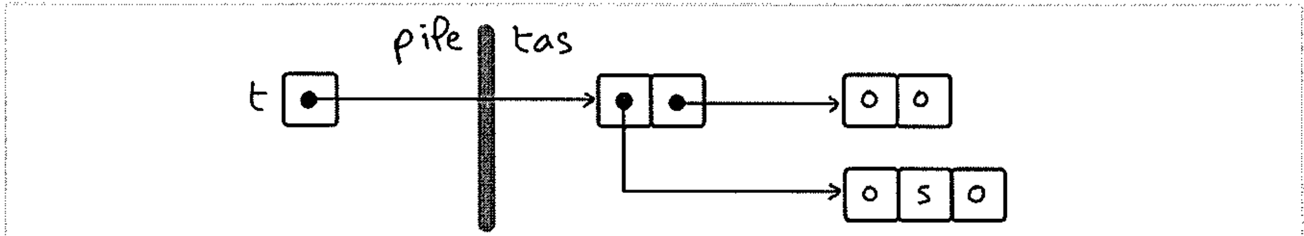
```
void insert(char* str, char c);
```

...qui insère le caractère `c` au début de la chaîne située dans le tableau désigné par `str`. On suppose que le tableau est suffisamment grand pour contenir la nouvelle chaîne, incluant son marqueur de fin.

Partie E

E1 (1,5 points)

Donnez les lignes de code permettant d'obtenir la configuration suivante en mémoire :



la variable t est de type `int**`.

E2 (0,5 point)

Donnez les lignes de code permettant de libérer les blocs mémoire réservés dans le tas par l'exécution des lignes de code de la question précédente.

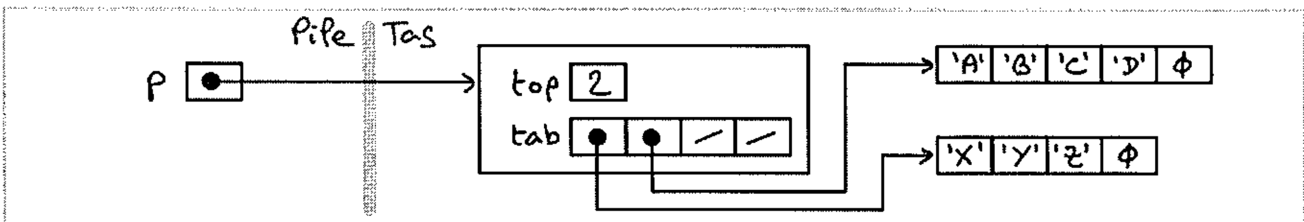
Partie F

On définit la structure suivante, permettant de représenter une pile de chaines de caractères.

```
#define SIZE 4

typedef struct
{
    int top;
    char* tab[SIZE];
}pileStr;
```

Voici un exemple d'instance complète de cette structure, avec sa ramification dans le tas, représentant une pile contenant 2 chaînes :



Les traits inclinés représentent des valeurs indéfinies et/ou sans importance. Une pile vide est représentée par un unique bloc mémoire, situé dans le tas, contenant une instance de `pileStr` dont le champ `top` vaut 0 et dont les cellules du tableau situé dans le champ `tab` sont indéfinies et / ou sans importance.

La fonction...

```
pileStr* creeVide();
```

...crée une pile vide dans le tas. On suppose que cette fonction est correctement définie.

F1 (1,5 point)

Définissez la fonction...

```
void empile(pileStr* p, const char* str);
```

...qui empile dans la pile désignée par `p` une **copie** de la chaîne désignée par `str`.

Avant de répondre, examinez bien la figure ci-dessus et dessinez au brouillon la configuration en mémoire après l'empilement d'une troisième chaîne. Vous ne serez évalué que sur le code de votre solution, mais cette réflexion préalable vous évitera de faire certaines erreurs.

F2 (1,5 point)

On définit la fonction `depile` suivante :

```
char* depile(pileStr* p)
{
    p->top--;
    return p->tab[p->top];
}
```

Définissez une fonction `testPileStr` n'acceptant aucun paramètre et ne retournant rien, qui réalise les actions suivantes :

1. Création d'une pile vide dans le tas. L'adresse de cette pile doit être placée dans une variable `p`.
2. Empilage des chaînes `"Tim"` et `"Tom"` dans cette pile.
3. Dépilage et affichage de la chaîne située au sommet de la pile. La fonction `depile` doit être utilisée.
4. Dépilage et affichage de la chaîne située au sommet de la pile. La fonction `depile` doit être utilisée.

La fonction `testPileStr` ne doit provoquer aucune fuite mémoire ni accident mémoire.

Partie G

Attention : changement de langage. Les questions suivantes concernent le langage C++.

On considère la classe suivante, permettant de représenter une collection de notes (valeurs de type `int`) et de les afficher sous la forme d'un histogramme.

```
class HistoNotes
{
private:
    vector<int> data;
public:
    HistoNotes(int borne);
    void addNote(int note);
    int nbOcc(int note);
    void affiche();
    int nbNotes();
    int borneSup();
};
```

La valeur `data[i]` représente le nombre d'occurrences de la note `i` dans la collection courante. Cette valeur est accessible via la méthode `nbOcc`. Les notes sont ajoutées à la collection à l'aide de la méthode `addNote`.

Voici les définitions du constructeur (qui initialise le vecteur `data` avec des 0) et des méthodes `borneSup` et `nbOcc`.

```
HistoNotes::HistoNotes(int borne) : data(borne+1,0) {}

int HistoNotes::borneSup() {return (int)data.size()-1;}

int HistoNotes::nbOcc(int note){return data[note];}
```

Voici un exemple d'utilisation permettant de bien comprendre le rôle de chaque méthode.

On commence par créer une instance de la classe `HistoNotes` :

```
HistoNotes h(10);
```

La valeur 10 passée au constructeur est la note la plus haute pouvant être enregistrée. La note la plus basse est toujours 0. Donc notre histogramme pourra contenir des notes comprises entre 0 (inclus) et 10 (inclus).

Ensuite, on ajoute des notes, qui doivent être des entiers :

```
h.addNote(5); h.addNote(6); h.addNote(3);
h.addNote(5); h.addNote(7); h.addNote(6);
h.addNote(5);
```

On peut alors savoir le nombre d'occurrences d'une note à l'aide de la méthode `nbOcc`. Par exemple, l'exécution des lignes suivantes...

```
cout << h.nbOcc(5) << endl;
cout << h.nbOcc(6) << endl;
```

...produit l'affichage...

```
3
2
```

...car la note 5 a été ajoutée 3 fois à la collection (avec la méthode `add`) et la note 6 a été ajoutée deux fois.

La méthode `affiche` permet d'afficher la répartition des notes sous la forme d'un histogramme. Dans notre exemple, l'exécution de la ligne...

```
h.affiche();
```

...produit l'affichage suivant :

```
0
1
2
3 +
4
5 +++
6 ++
7 +
8
9
10
```

Le nombre de `+` affichés en face de chaque note représente le nombre de fois que cette note a été ajoutée à l'histogramme, c'est à dire le nombre d'occurrences de cette note. On voit que, par exemple, la note 5 a trois occurrences et que la note 6 en a deux.

G1 (1 point)

Définissez la méthode...

```
void HistoNotes::addNote(int note);
```

...qui ajoute une note à l'histogramme courant. Cette méthode ne produit aucun affichage. Référez-vous à l'exemple d'utilisation plus haut pour bien comprendre son rôle. Si la note est négative ou est supérieure à la plus grande note pouvant être enregistrée, la fonction ne doit avoir aucun effet.

G2 (2 points)

Définissez la méthode...

```
void HistoNotes::affiche();
```

...qui affiche l'histogramme courant conformément à l'exemple donné plus haut.

Partie H

Dans le cadre de cet exercice, on s'intéresse aux **arbres étiquetés** dans le sens suivant :

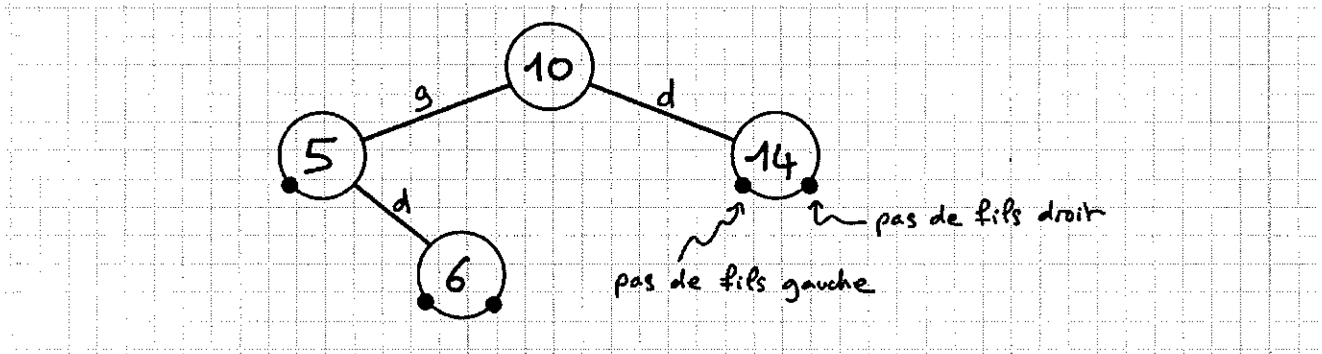
- Tout arbre étiqueté a une **valeur racine** de type `int`.
- Tout arbre étiqueté peut avoir (ou pas) un fils gauche. Ce fils gauche, s'il existe, est un arbre étiqueté.
- Tout arbre étiqueté peut avoir (ou pas) un fils droit. Ce fils droit, s'il existe, est un arbre étiqueté.

On dit qu'un entier x est **une valeur** d'un arbre étiqueté t si et seulement si x est la valeur racine de t ou x est une valeur d'un des fils de t (si applicable).

On appelle **arbre trié** tout arbre étiqueté t vérifiant les propriétés suivantes :

- Si t a un fils gauche g, alors g est un arbre trié et toutes les valeurs de g sont inférieures ou égales à la valeur racine de t.
- Si t a un fils droit d, alors d est un arbre trié et toutes les valeurs de d sont strictement supérieures à la valeur racine de t.

Voici un exemple d'arbre trié.



La classe suivante représente des arbres triés :

```
class Arbre
{
private:
    int val;
    Arbre* gauche;
    Arbre* droit;

public:
    Arbre(int val);
    ~Arbre();

    int getVal() const;
    void addVal(int x);
    void affiche() const;
};
```

L'attribut `val` représente la valeur racine. L'attribut `gauche` a la valeur `nullptr` s'il n'y a pas de fils gauche, sinon il pointe sur le fils gauche. L'attribut `droit` a la valeur `nullptr` s'il n'y a pas de fils droit, sinon il pointe sur le fils droit.

Le constructeur crée un arbre trié contenant une seule valeur, sans fils.

La méthode `addVal` ajoute une valeur dans l'arbre courant en le maintenant trié. Par exemple, l'arbre représenté par la figure donnée plus haut peut être obtenu de la manière suivante :

```
Arbre a(10);
a.addVal(5);
a.addVal(14);
a.addVal(6);
```

La méthode `affiche` permet l'affichage en ordre croissant de toutes les valeurs de l'arbre courant. Par exemple, à la suite des lignes de code précédente, la ligne...

```
a.affiche();
```

...produit l'affichage :

```
5 6 10 14
```

H1 (1 point)

Définissez la méthode :

```
void Arbre::affiche() const;
```

H2 (1 point)

Définissez la méthode :

```
void Arbre::addVal(int x);
```

info4A - mai 2022 - Fiche de réponses

A1(2)

```
int f(int x)
{
}
```

B1(1)

B2(2)

```
uint16_t g()
{ uint16_t out = 0;
  for (int i=0; i<16; i++)
  {
    if (rand()%3 == 0)
    {
    }
  }
  return out;
}
```

E2(0,5)

C2(2)

```
void sums (cont int* tab, int n, int* s1, int* s2)
{
}
```

D2(1,5)

```
void insert( char* str, char c)
{
}
```

D1(0,5)

```
s[ ] = s[ ];
s[ ] = s[ ];
s[ ] = s[ ];
s[ ] = s[ ];
s[ ] =      ;
```

E1(1,5)

C1(1)

No anonymat :

```
void empile(pileStr* p, const char* str)
{
}
```

F1 (1,5)

```
void Arbre::affiche() const
{
}
```

H1 (1)

F2 (1,5)

```
void HistoNotes::addNote(int note)
{
}
```

G1 (1)

```
void HistoNotes::affiche()
{
}
```

G2 (2)

```
void Arbre::addval(int x) const
{
}
```

H2 (1)