

Prénom :

Correction prof

Evaluation 1/3

Structures, tableaux,
classes, objets

Un intervalle est soit vide, soit défini par deux bornes inf et sup qui, dans le contexte de cet exercice, sont des valeurs entières. Par convention, tout intervalle ayant une borne inf plus grande que sa borne sup sera considéré comme vide.

Complétez les déclarations et définitions suivantes permettant de gérer des intervalles en langage C.

```
struct intervalle
{
    int inf;
    int sup;
};
```

Déclarez ici la structure intervalle.

0.5pt

```
int min(int a, int b)
{
    return a < b ? a : b;
}
```

ou if(a < b) return a;
else return b

0.75pt

Donnez ici la définition complète d'une fonction min acceptant deux entiers en paramètres et retournant le plus petit des deux.

```
void printIntervalle(intervalle* w)
```

```
{
    if (w->inf > w->sup)
        printf("[ ]"); // cout << "[ ]";
    else
        printf("[%d, %d]", w->inf, w->sup);
}
```

Affiche un intervalle sous la forme [min,max] ou [] si l'intervalle est vide.

ok avec cout <<

1pt

Retourne vrai si l'intersection des deux intervalles n'est pas vide, faux sinon.

Réalisez un code concis en utilisant la fonction min définie précédemment et une fonction max dont on supposera l'existence.

```
int testIntersect(intervalle* a, intervalle* b)
```

```
{
    if (max(a->inf, b->inf) < min(a->sup, b->sup))
        return 1;
    else
        return 0;
}
```

1.5pt

Structures

Définition
teur de

Retourne l'intervalle (éventuellement vide) résultat de l'intersection de deux intervalles. Utilisez la fonction testIntersection présentée précédemment

intervalle intersection(intervalle* a, intervalle* b)

```
intervalle i; // 0.25 avec return i
if ( testIntersection(a,b) ) // 0.25
{
    i.inf = max(a->inf, b->inf); // 0.25
    i.sup = min(a->sup, b->sup); // 0.25
}
return i;
```

1pt

void testIntervalle()

```
{
    intervalle a, b, c, ab, bc, ac;
    0.25 a.inf = 1; a.sup = 3; PrintIntervalle(&a);
    0.25 b.inf = 2; b.sup = 4; // (&b);
    0.25 c.inf = 5; c.sup = 6; // (&c);
    0.25 ab = intersection(&a, &b); // (&ab);
    0.25 bc = // (&b, &c); // (&bc);
    0.25 ac = // (&a, &c); // (&ac);
}
```

Écrivez ici le code permettant de créer les intervalles [1,3], [2,4], et [5,6], de les afficher, et d'afficher leurs intersections.

[1,3] 1.5 pt
[2,4]
[5,6]
[] structures
[]

Vous devez maintenant réaliser une classe Intervalle en langage C++ avec les mêmes fonctionnalités que celles décrites précédemment. Seuls la déclaration de la classe et les codes complets du constructeur et de la méthode d'affichage sont demandés.

1.25pt

Classes

class Intervalle

```
{
    private:
    0.25 int inf, sup; // Variables d'instance
    public:
    0.25 intervalle (int a=0, int b=0); // Constructeur
    0.25 void PrintIntervalle(); // Affichage
    0.25 int testIntersection(intervalle* b); // Test d'intersection
    0.25 Intervalle Intersection(intervalle* b); // Calcul d'intersection
};
```


Définition complète du constructeur de la classe Intervalle

0.5pt

```
0.25 Intervalle::Intervalle(int a, int b)
{
    inf = a;
    sup = b;
}
```

Classes

```
void Intervalle::PrintIntervalle()
{
    0.25 if (inf > sup) cout << "[ ]\n";
    else
    0.25 cout << "[" << inf << ", " << sup << "]" << "\n";
}
```

Définition complète de la méthode d'affichage d'un intervalle.

0.75pt

On déclare de la manière suivante une classe ListeInter représentant une liste d'intervalles. Complétez les définitions demandées.

class ListeInter

```
{
    private:
        Intervalle* tab;
        int n;
        int capacite;

    public:
        ListeInter(int capa);

        void add(Intervalle i);
        void retireVides();
};
```

n est le nombre d'intervalles actuellement enregistrés dans la liste et capacite est le nombre maximum d'intervalles pouvant y être stockés.

Le constructeur crée une liste vide, mais ayant une capacité de stockage précisée en paramètre.

La méthode add ajoute un intervalle à la liste. La méthode retireVides retire de la liste tous les intervalles vides.

ListeInter::ListeInter(int capa)

```
{
    n = 0;
    capacite = capa;
    tab = new Intervalle[capa];
}
```

Classes
Tableaux 1D

0.75pt

2pt

Adoptez un algorithme simple et efficace, de complexité en temps linéaire, qui peut modifier l'ordre des intervalles restants dans la liste.

Il faut utiliser ici une seule boucle et un passage si le code est juste mais avec 2 boucles il faut compter 1pt au lieu de 2pt

```
void ListeInter::retireVides()
{
    int i = 0;
    while (i < n)
    {
        if (tab[i].inf >= tab[i].sup)
        {
            a = tab[i];
            tab[i] = tab[n-1];
            tab[n-1] = a;
            if (tab[i].inf >= tab[i].sup) i--;
        }
        n--;
        i++;
    }
}
```

3/5

vous devez à présent définir en langage C une structure listeInter et implanter deux fonctions: initListeInter et retireVides qui remplissent les rôles respectifs du constructeur et de la méthode retireVide de la version C++.

Déclaration de la structure listeInter

```
struct ListInter
{
    int capacite; 0.25
    int n; 0.25
    intervalle * tab; 0.5
};
```

1pt

0.25 void init (ListInter & L, int cp);

0.25 L.n = 0;

0.25 L.capacite = cp;

0.5 L.tab = new intervalle [cp];

1.25pt

Définition complète de la fonction initListeInter

Structures
Tableaux 1D

2pt

Définition complète de la fonction retireVide.

```
void retireVide (ListInter & L)
{
    int i = 0; intervalle a;
    while (i < L.n)
    {
        if (L.tab[i].inf >= L.tab[i].sup)
        {
            a = L.tab[i]; L.tab[i] = L.tab[L.n-1];
            L.tab[L.n-1] = a;
            if (L.tab[i].inf >= L.tab[i].sup) i--;
            L.n--;
        }
        i++;
    }
}
```

Le problème des reines consiste à placer n reines sur un damier de n par n de manière à ce que chaque ligne, chaque diagonale et chaque colonne ne contienne pas plus d'une reine. Lorsque deux reines sont sur une même ligne, colonne ou diagonale, on dit qu'elles sont en conflit. Une classe Damier est définie de la manière suivante.

```
class Damier
{
private:
```

```
    char** tab;
    int n;
```

```
public:
```

```
    Damier(int n);
```

```
    void random();
    void print();
```

```
};
```

Permet l'accès à un tableau à deux dimensions de caractères. Une cellule contenant une reine a la valeur 1, sinon elle a la valeur 0.

Le constructeur crée le tableau 2D représentant le damier vide. Les cellules du tableau doivent donc être initialisées à 0.

random place n reines au hasard sur le damier.

print affiche le damier en représentant les reines par le caractère 'O' et les cases vides par le caractère '-'.

Compléter les définitions des méthodes indiquées.

Damier::Damier(int n)

```
{
    this->n = n; 0.25
    tab = new char*[n]; 0.25
    for(int i=0; i<n; i++)
        tab[i] = new char[n]; 0.25
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++) 0.25
            tab[i][j] = 0;
}
```

1pt

Crée et initialise à 0 un damier de n par n.

Classes/
Tableaux 2D

void Damier::random()

```
{
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            tab[i][j] = 0; 0.25
    int x, y;
    for(int i=0; i<n; i++)
    {
        x = rand() % n; 0.25
        y = rand() % n;
        tab[y][x] = '-'; 0.25
    }
}
```

0.75pt

Remet le damier à 0 puis place n reines aléatoirement dans des cases distinctes.

Pour mémoire, rand() % n retourne un entier aléatoire compris entre 0 et n-1 inclus.

void Damier::print()

```
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            cout << tab[i][j]; 0.25
        }
        cout << endl; 0.25
    }
}
```

0.5pt

Affiche le damier sous forme de caractères.