FACULTÉ DES SCIENCES

Formulaire pour reprographie d'examen

Le questionnaire d'examen doit parvenir au Centre de reprographie 5 jours ouvrables avant la date de l'examen.

INFORMATIONS POUR LE SECRÉTARIAT DE LA FACULTÉ
(Attacher à chaque questionnaire)

☒ **INTRA**
☐ **FINAL**

Sigle : IFT339

Titre : Structures de données

Professeur : GOULET, Jean

Date et heure de l'examen : Jeudi 8 octobre à 15 h 30

Nombre de pages : 4 en cahier recto-verso

Nombre d'étudiants : 64

☐ **Je désire prendre possession des questionnaires d'examen la veille de l'activité et je me rends responsable de leur mise en sécurité.**

Signature : _____

L'étudiant devra répondre

☐ dans un cahier d'examen
☒ sur le questionnaire

Veuillez inclure

_____ feuilles blanches additionnelles
_____ feuilles de papier graphique

Je consens à ce que deux (2) copies du questionnaire de cet examen soient remises à l'AGES (Association générale des étudiants en sciences) après la fin de la période des examens.

☒ **OUI** ☐ **NON**

Signature : _____

Département d'informatique
Examen intratrimestriel
Automne 2015

IFT339

Structures de données

jeudi 8 octobre 2015
15 h 30 à 17 h 30
Jean Goulet

Cet examen comprend trois questions, toutes d'égale valeur. Lisez d'abord tout l'examen, puis répondez directement sur le questionnaire. Vous avez droit au **Résumé C++** jaune, à une calculatrice, et à une feuille 8 ½ x 11 recto verso de notes personnelles manuscrites. Le surveillant vous fournira des feuilles blanches pour vos brouillons.

Identifiez-vous lisiblement ci-dessous :

Nom, prénom :	
Signature :	

1:	/10
2:	/10
3:	/10
	/30

Question 1 – Représentation des types primitifs (10 points)

Votre matricule:

m7	m6	m5	m4	m3	m2	m1	m0

Arithmétique des types entiers (dans des int16_t). Indiquez s'il y a dépassement de capacité (oui ou non)

e															
	m2	m4	m7	+	m1	m6	a	m3	=						

dépassement?(o/n) _____

e															
	m2	m4	m7	-	m1	m6	a	m3	=						

dépassement?(o/n) _____

Conversion d'un int16_t vers sa valeur décimale

e					
	m2	m4	m7	=	

_____

Conversion d'un nombre décimal en float (exprimé en hexadécimal)

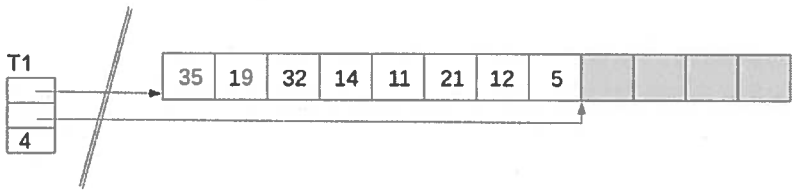
2									
	m4	m5	m2	.	m1	1	=		

Question 2 – Implantation et utilisation abstraite d'un vector (10 points)

On a déjà écrit la fonction `resize` du `vector`, qui en modifie la dimension, et gère au besoin sa capacité, s'assurant qu'il a une réserve pour les prochains `push_back`, ce qui permet de les rendre O(1) (la capacité est le nombre total de places, et non le nombre de places libres). On veut maintenant écrire la fonction `reserve`, qui nous permet de gérer la capacité elle-même. Nous allons lui donner un sens un peu différent de celui de la SL, pour les fins de cet exercice. Cette fonction reçoit un paramètre de type `size_t` et s'assure que la réserve pour les futurs `push_back` et `resize` sera exactement égale à cette valeur. Un paramètre de 0 est donc l'équivalent du `shrink_to_fit` du `deque`. Codez cette fonction avec la représentation telle qu'illustrée ici (le dessin montre un vecteur de dimension 8 avec une réserve de 4. Ce serait le résultat de `T1.reserve(4)`).

On désire aussi coder des fonctions qui nous permettent de manipuler globalement les `vector`, plutôt qu'élément par élément. Par exemple de faire la somme de tous les éléments d'un vector, pour autant que les objets contenus dans le vector possèdent un opérateur d'addition (ce sera à l'usager de s'occuper de cette restriction, cela ne nous regarde pas!). Cette fonctionnalité nous permettra de calculer la moyenne des éléments d'un `vector` de doubles V en faisant : `(+V)/V.size()`. Dans un `vector<string> VS`, `+VS` nous retournerait la concaténation de tous les éléments du vecteur dans une seule `string`. Codez aussi cette fonction, dont la signature vous est donnée.

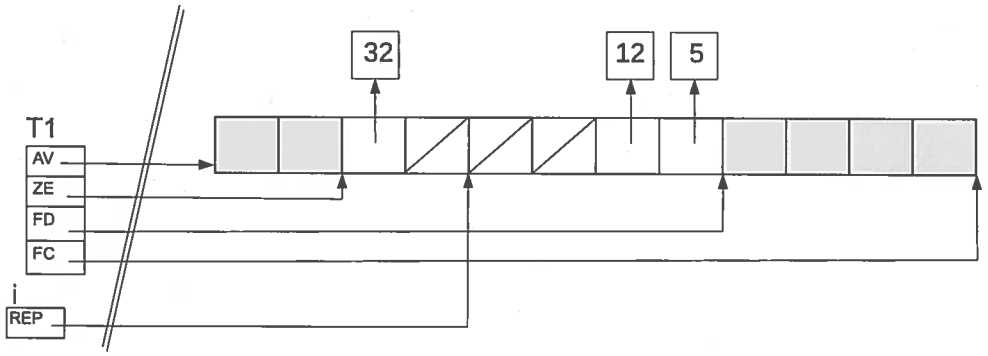
```
template <typename TYPE>
class vector{
private:
   TYPE*  DEBUT;
   TYPE*  FINDIM;
   size_t RESERVE;
public:
   ...
   void reserve(size_t);
   TYPE operator+()const;
   ...
};

template <typename TYPE>
void vector<TYPE>::reserve(size_t nouv_res){
```



```
template <typename TYPE>
TYPE vector<TYPE>::operator+()const{
```

Question 3 – Implanter un deque paresseux (10 points)

Le deque avec les poignées ("*handle*")que nous avons implanté dans le troisième laboratoire instanciait les objets dès le moment du `resize`, que l'usager aie l'intention de les utiliser ou non. Un deque paresseux ("*lazy deque*") attend le plus tard possible avant de faire cette instanciation. Au moment du `resize` (ou dans le constructeur), il laisse les pointeurs à `nullptr`, et attend que l'usager aie besoin d'accéder à cet élément (dans l'opérateur `[ ]`) pour se faire allouer l'espace de cet objet. Un deque général pourrait donc avoir l'air de l'illustration ci-dessous, où certains objets ont été instanciés et d'autres non. On y voit un deque de 6 éléments, et un itérateur qui donne la position du troisième. Pour implanter un tel deque, il faut revoir la fonction `resize` ainsi que l'opérateur `[ ]` (les deux versions, incluant celle qui reçoit le vector non const!). Le `push_front` n'a pas besoin d'être modifié, puisque celui-ci doit toujours instancier un objet de toutes façons. Il faudrait aussi revoir la déréférence des itérateurs, bien sûr, pour faire un travail complet. On ne fera que l'`iterator` ici, pas toutes les autres versions. Avec la représentation donnée ci-dessous, codez les fonctions `resize`, l'`operator[ ]`, et la déréférence de l'`iterator`. Notez que l'opérateur `delete` accepte sans problème un pointeur nul comme paramètre, que cela ne cause pas d'erreur et n'a aucun effet. Toutes les autres fonctions du deque sont à votre disposition.




```
template <typename TYPE>
class deque{
private:
    TYPE **AVANT,**ZERO,**FINDIM,**FINCAP;

public:
    class iterator;
    ...
    void resize(size_t);
    TYPE& operator[](size_t);
    const TYPE& operator[](size_t)const;
    ...
};
```

```
template <typename TYPE>
void deque<TYPE>::resize(size_t nouv_dim){
```

```
template<typename TYPE>
TYPE& vector<TYPE>::operator[](size_t i){
```

```
template<typename TYPE>
const TYPE& vector<TYPE>::operator[](size_t i)const{
```

```
template <typename TYPE>
class vector<TYPE>>iterator{
private:
    TYPE** POINTEUR;
public:
...
    TYPE& operator*()const{

    }
...
};
```