TP1 : Une première classe

B3111 : Edern HAUMONT et Nicolas SIX

Mercredi 21 octobre 2015

Table des matières

1 Introduction

La classe BoolContainer gère une collection dynamique de booléens non triée. Cette collection a une taille spécifiée à sa création, cependant elle peut être modifiée à la demande ou optimisée automatiquement. La classe est dotée de méthodes daffichage, dajout et de suppression ditems, de réunion dune autre collection de booléens et de modification de capacité. En interne, les booléens sont stockés dans un tableau statique de booléens. Celui-ci présente l'avantage de prendre peu de place en mémoire et de permettre un accès rapide aux éléments. Cependant, cela impose de recréer un tableau et de copier recopier tous les éléments du tableau un à un à chaque fois que sa taille est modifiée. Les attributs et méthodes de la classe sont spécifiés ci-après.

2 Attributs

La classe BoolContainer utilise trois attributs privés pour gérer ses données :

```
unsigned int tailleUtilisee;
unsigned int tailleDispo;
bool *tab;
```

Cela lui permet de connaitre la quantité de données actuelement stockée mais aussi d'avoir une zone de stockage par l'intermédiaire du pointeur.

3 Méthodes publiques à spécifier

3.1 Constructeur avec tableau d'initialisation

```
BoolContainer(bool nouveauTab[], unsigned int nouvelleTaille);
```

Prend en parametre un tableau statique de booleens qui sera copié et sa taille.

Contrat : nouveauTab est de taille supérieure ou égale à nouvelleTaille (si sa taille est supérieure à nouvelleTaille seuls les éléments des cases entre 0 et nouvelleTaille exclue seront copiés dans le tableau).

3.2 Ajouter

```
int Ajouter(bool b);
```

Prend en paramètre le booléen qui sera ajouté a la fin de la liste.

3.3 Retirer

```
int Retirer(unsigned int debut, unsigned int longueur = 1);
```

Prend en paramètre l'index à partir duquel la suppression aura lieu et en paramètre optionnel, la longueur de l'intervalle d'index sur lequel les suppressions auront lieu.

3.4 Réunir

```
int Reunir(const BoolContainer & boolContainerBis);
```

Prends en paramètre une autre collection de booléens qui sera ajoutée à la fin de la collection courante.

4 Codes d'erreur

Nous avons choisi de faire renvoyer un code entier par nos fonctions de façon à détecter plus facilement certaines erreurs dexécution via des codes :

— **const int** ERROR_NON_INTUITIVE_ADJUSTMENT = 100;

Ce code derreur est renvoyé par la méthode Ajuster si la nouvelle taille demandée est inférieure à la taille actuellement utilisée par la collection.

— **const int** ERROR_RESIZE_FAILED = 101;

Échec de redimensionnement du tableau.

— **const int** ERROR_OUT_OF_BOUNDARY = 102;

Ce code derreur est renvoyé par la méthode Retirer si un ou des éléments dont on demande le retrait ne se trouvent pas dans la collection, cest-à-dire au-delà de tailleUtilisee.

5 Tests unitaires à spécifier

Pour des raisons d'encapsulation le programme ne peut pas effectuer la plupart des tests par lui-même. De fait la plupart des tests se font par contrôle visuel : l'utilisateur ayant lancé le programme de test doit indiquer lui-même si les tests ont réussi ou échoué.

5.1 Test de la méthode Ajouter (1)

test visuel :

La fonction crée un BoolContainer vide, puis utilise la méthode Ajouter pour y insérer des éléments. La méthode Afficher est appelée. Lutilisateur na quà comparer le résultat de Afficher à ce qui était attendu.

5.2 Test de la méthode Retirer (2)

test visuel :

La fonction crée un BoolContainer et le remplit. Elle supprime ensuite un élément puis un groupe déléments et permet à chaque fois à lutilisateur de vérifier que les bons éléments ont été retirés.

5.3 Test de la méthode Réunir (3)

test visuel :

La fonction crée deux BoolContainer. Elle les affiche. Puis elle appelle la méthode Reunir sur le premier tableau avec le second comme argument. Enfin elle affiche le résultat et lutilisateur vérifie la correspondance.

5.4 Test d'overflow (4)

test automatisé :

Ce test vérifie qu'il n'y a pas de problème lors de l'insertion dans un tableau plein. Pour cela il crée un tableau de taille un. Puis ajoute des éléments un à un pour essayer de supprimer le même nombre déléments que celui qui a été ajouté. Le test ne vérifie pas lintégrité des données mais permet de s'assurer que les tableaux se remplissent dans ce cas sans perte d'éléments.

Le test étant automatisé il effectue ses tests sur un grand nombre de tableaux. Même si cela ne semble pas nécessaire, nous nous assurons ainsi d'une certaine stabilité du processus sans que l'utilisateur ne puisse voire la différence.

5.5 Test du second constructeur (5)

test visuel :

Ce test génère automatiquement un tableau de booléens et appelle le second constructeur en utilisant ce tableau en paramètre. L'utilisateur doit comparer le tableau passé en paramètre et le retour de la méthode Afficher() du nouvel objet.

6 Annexes

6.1 makefile

```
CC=g++
CFLAGS=
LDFLAGS=
EXEC=main
SRC=BoolContainer.cpp main.cpp
OBJ=$(SRC:.cpp=.o)

all: $(EXEC)

main: $(OBJ)
	$(CC) -o $@ $^ $(LDFLAGS)

main.o: main.cpp
	$(CC) -o $@ -c $< $(CFLAGS)

BoolContainer.o: BoolContainer.cpp BoolContainer.h errorCode.h
	$(CC) -o $@ -c $< $(CFLAGS)

.PHONY:clean

clean:
	rm -rf $(OBJ) $(EXEC)
```

6.2 main.cpp

```
#include <iostream>
#include "BoolContainer.h"

#define TEST_ERROR_MESSAGE "\e[1;31m[Failed]\e[0m"
#define TEST_SUCESS_MESSAGE "\e[1;32m[OK]\e[0m"

using namespace std;

void selfTesting();
void containerSizeTest(unsigned int containerSize, bool testOverflow =false);
void containerAjusterTest(unsigned int containerSize);

void manualTesting();
void manualAjouterTest(unsigned int containerSize);
void manualRetirerTest(unsigned int containerSize);
void manualReunirTest(unsigned int containerSize);
void manualConstructorTest(unsigned int containerSize);

int main()
{
    manualTesting();
    selfTesting();
}

void selfTesting()
{
    cout << "Test d'overflow: ";
    containerSizeTest(200, true);
    cout << "Test de Ajuster: ";
    containerAjusterTest(200);
}

/* 4 - */void containerSizeTest(unsigned int containerSize, bool testOverflow)
{
    unsigned usedSize = 1;
    if(!testOverflow)
    {
        usedSize = containerSize;
    }
    for(unsigned n=0 ; n< usedSize; n++)
    {
        BoolContainer booleanTab(usedSize);
                //remplie le tableau de vrais pour les n première case puis de faux
        for(unsigned i=0 ; i< usedSize; i++)
        {
            if(booleanTab.Ajouter(i<n) != 0)
            {
                cout << TEST_ERROR_MESSAGE << endl;
                return;
            }
        }
        //vérifie que les donnée retirer corespondent au données envoyer
```

```
        for(unsigned i=0 ; i< usedSize; i++)
        {
            if(booleanTab.Retirer(0) != 0)
            {
                cout << TEST_ERROR_MESSAGE << endl;
                return;
            }
        }
    }
    BoolContainer booleanTab(usedSize);
    //test en alternance:
    for(int i=0 ; i< usedSize; i++)
    {
        booleanTab.Ajouter(i%2);
    }
    for(unsigned i=0 ; i< usedSize; i++)
    {
        if(booleanTab.Retirer(0) != 0)
        {
            cout << TEST_ERROR_MESSAGE << endl;
            return;
        }
    }
    cout << TEST_SUCESS_MESSAGE << endl;
}


/* 6 -*/void containerAjusterTest(unsigned int containerSize)
{
    bool tab[containerSize];
    BoolContainer booleanTab(tab,containerSize);
    for (unsigned i = 0; i < 2*containerSize; ++i) {
        if((booleanTab.Ajuster(i) == 0 && i<containerSize) ||\
          (booleanTab.Ajuster(i) != 0 && i>=containerSize))
        {
            cout << TEST_ERROR_MESSAGE << i << " " << booleanTab.Ajuster(i) << endl;
            return;
        }
    }
    cout << TEST_SUCESS_MESSAGE << endl;
}

/// Manual test fonctions:

void manualTesting()
{
    cout << "\e[1;31m/!\\ lors des entré texte veulliez ne donnée\
                    qu'un seul caractère\e[0m";
    cout << endl << endl;
    manualConstructorTest(20);
    manualAjouterTest(20);
    manualRetirerTest(20);
    manualReunirTest(20);
```

```
}
/* 1 - */void manualAjouterTest(unsigned int containerSize)
{
    BoolContainer boolContainer(0);
    cout << "Conteneur vide déclarer:" << endl;
    boolContainer.Afficher(false,true);
    cout << "remplisage du conteneur avec une alternance de 1 et 0 commancant par 0";
    cout << endl;
    for(unsigned i=0 ; i<containerSize ; i++)
    {
        boolContainer.Ajouter(i%2);
    }
    boolContainer.Afficher(false,true);
    cout << "le résultat correspond t'il à ce qui était attendue ? [o/N] ";
    char userAnswer = 'n';
    cin >> userAnswer;
    cout << "test d'ajout à vérification manuel: ";
    if(userAnswer != 'o' && userAnswer != 'O')
    {
        cout << TEST_ERROR_MESSAGE << endl << endl;
    }
    else
    {
        cout << TEST_SUCESS_MESSAGE << endl << endl;
    }
}
/* 2 - */void manualRetirerTest(unsigned int containerSize)
{
    BoolContainer boolContainer(0);
    cout<<"remplisage du conteneur avec une alternance de 1 et 0 commencant par 0";
    cout<<endl;
    for(unsigned i=0 ; i<containerSize ; i++)
    {
        boolContainer.Ajouter(i%2);
    }
    boolContainer.Afficher(false,true);
    cout << "supression de la première valeur:" << endl;
    boolContainer.Retirer(0);
    boolContainer.Afficher(false,true);
    cout << "supression de la dernière valeur:" << endl;
    boolContainer.Retirer(containerSize-2);
    boolContainer.Afficher(false,true);
    cout << "les résultats correspondent t'ils à ce qui était attendue ? [o/N] ";
    char userAnswer = 'n';
    cin >> userAnswer;
    cout <<  "test de retirer pour un élément avec vérification manuel: ";
    if(userAnswer != 'o' && userAnswer != 'O')
    {
        cout << TEST_ERROR_MESSAGE << endl << endl;
        return;
    }
    else
    {
```

```
        cout << TEST_SUCESS_MESSAGE << endl << endl;
    }

    BoolContainer boolContainer2(0);
    for(unsigned i=0 ; i<containerSize ; i++)
    {
        boolContainer2.Ajouter(i<(containerSize/2));
    }
    cout<<"creation du conteneure replie de 1 de 0 à "<<((containerSize/2)-1);
    cout<<" et de 0 ensuite" << endl;
    boolContainer2.Afficher(false,true);
    cout<<"suppression des valeurs de 0 à "<<((containerSize/2)-1)<<":" ;
    cout<<endl;
    boolContainer2.Retirer(0,containerSize/2);
    boolContainer2.Afficher(false,true);
    cout<<"les résultats correspondent t'ils à ce qui était attendue ? [o/N] ";
    userAnswer = 'n';
    cin >> userAnswer;
    cout << "test de retirer pour un élément avec vérification manuel: ";
    if(userAnswer != 'o' && userAnswer != 'O')
    {
        cout << TEST_ERROR_MESSAGE << endl << endl;
        return;
    }
    else
    {
        cout << TEST_SUCESS_MESSAGE << endl << endl;
    }
}
/* 3 - */void manualReunirTest(unsigned int containerSize)
{
    BoolContainer boolContainer1(0);
    cout << "remplisage du conteneur nř1 de 1" << endl;
    for(unsigned i=0 ; i<containerSize/3 ; i++)
    {
        boolContainer1.Ajouter(1);
    }
    boolContainer1.Afficher(false,true);
    BoolContainer boolContainer2(0);
    cout << "remplisage du conteneur nř2 de 0" << endl;
    for(unsigned i=0 ; i<(containerSize*2)/3 ; i++)
    {
        boolContainer2.Ajouter(0);
    }
    boolContainer2.Afficher(false,true);
    cout << "reunion des deux conteneur, le nř2 étant à la fin du nř1" << endl;
    boolContainer1.Reunir(boolContainer2);
    boolContainer1.Afficher(false,true);
    cout << "le résultat correspond t'il à ce qui était attendue ? [o/N] ";
    char userAnswer = 'n';
    cin >> userAnswer;
    cout << "test de Reunir avec vérification manuel: ";
    if(userAnswer != 'o' && userAnswer != 'O')
```

```
    {
        cout << TEST_ERROR_MESSAGE << endl << endl;
    }
    else
    {
        cout << TEST_SUCESS_MESSAGE << endl << endl;
    }
}

/* 5 - */void manualConstructorTest(unsigned int containerSize)
{
    bool baseBooleanTab[containerSize];
    cout << "construction d'un conteneur à l'aide du tableau :" << endl;
    for (int i = 0; i < containerSize; i++)
    {
        baseBooleanTab[i] = i%2;
        cout << baseBooleanTab[i] << " ";
    }
    cout << endl << "le conteneur optenue est :" <<endl;
    BoolContainer conteneur(baseBooleanTab,containerSize);
    conteneur.Afficher(false,true);
    cout << "le résultat correspond t'il à ce qui était attendue ? [o/N] ";
    char userAnswer = 'n';
    cin >> userAnswer;
    cout << "test du constructeur avec vérification manuel: ";
    if(userAnswer != 'o' && userAnswer != 'O')
    {
        cout << TEST_ERROR_MESSAGE << endl << endl;
    }
    else
    {
        cout << TEST_SUCESS_MESSAGE << endl << endl;
    }
}
```

6.3 BoolContainer.cpp

```
/*************************************************************************
                        BoolContainer  -  description
                             -------------------
    debut                : 05/10/2015
    copyright            : (C) 2015 par Edern Haumont & Nicolas Six
*************************************************************************/

//- Realisation de la classe <BoolContainer> (fichier BoolContainer.cpp) -

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
using namespace std;
#include <iostream>

//------------------------------------------------------ Include personnel
#include "BoolContainer.h"
#include "errorCode.h"

//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Methodes publiques
int BoolContainer::Afficher(bool afficherCarateristique, bool afficherContenue) const
// Algorithme :
// Affiche les tailles utilisees et disponibles
// Affiche une a une les valeurs du tableau
{
    if(afficherCarateristique)
    {
        cout << "la taille utilisee est de: " << tailleUtilisee;
        cout << " sur " << tailleDispo << " disponible" << endl;
    }
    if(afficherContenue)
    {
        for(int i=0 ; i<tailleUtilisee ; i++)
        {
            cout << tab[i] << " ";
        }
        cout << endl;
    }

    return 0;
} //----- Fin de Methode

int BoolContainer::Ajouter(bool b)
// Algorithme :
// Ajustement de la taille du tableau si elle n'est pas suffisante
// Mise a jour des variables
{
    if(tailleDispo == tailleUtilisee)
    {
        if(Ajuster(tailleDispo+1) != 0)
```

```
        {
            return ERROR_RESIZE_FAILED;
            //code erreur : echec du redimentionnement du tableau
        }
    }
    tab[tailleUtilisee]=b;
    tailleUtilisee++;
    return 0;
} //------ Fin de Methode

int BoolContainer::Retirer(unsigned int debut,unsigned int longueur)
// Algorithme :
// utilisation d'un pointeur pointant vers le nouveau tableau
// copie un a un des elements de l'ancien vers le nouveau tableau
// sauf celles a eliminer
// destruction de l'ancien tableau
{
    if(debut+longueur > tailleUtilisee)
    {
        return ERROR_OUT_OF_BOUNDARY;
        //code erreur : l'utilisateur demande de supprimer des elements
        // en dehors de la collection
    }
    bool * nouveauTableau = new bool[tailleUtilisee-longueur];
    int positionCourante = 0;
    for(int i=0 ; i<tailleUtilisee ; i++)
    {
        if(i<debut || i>=debut+longueur)
        {
            nouveauTableau[positionCourante] = tab[i];
            positionCourante++;
        }
    }
    delete [] tab;
    tab = nouveauTableau;
    tailleUtilisee -= longueur;
    tailleDispo = tailleUtilisee;
    return 0;
} //------ Fin de Methode

int BoolContainer::Ajuster(unsigned int nouvelletaille)
// Algorithme :
// utilisation d'un pointeur tampon qui permet de conserver le
// tableau d'origine copie une a une des valeurs du tableau vers
// un nouveau tableau de taille ajustee destruction du tableau
// d'origine via le pointeur tampon
{
    if(nouvelletaille < tailleUtilisee)
    {
        //cout << " ["<< tailleUtilisee <<"]";
        return ERROR_NON_INTUITIVE_ADJUSTMENT;
        //code erreur : ajustement impossible a cette taille
        //de maniere intuitive
```

```
    }
    else if(nouvelletaille == tailleUtilisee)
    {
        return 0;
    }
    bool * ansTab = tab;
    tab = new bool[nouvelletaille];
    tailleDispo = nouvelletaille;
    for(int i=0 ; i<tailleUtilisee ; i++)
    {
        tab[i] = ansTab[i];
    }
    delete [] ansTab;
    return 0;
} //----- Fin de Methode

int BoolContainer::Reunir(const BoolContainer & boolContainerBis)
// Algorithme :
// Appel a la fonction d'ajustement de taille
// copie valeur par valeur des variables du tableau du BoolContainer
// passé en parametre vers tab.
{
    if(Ajuster(tailleUtilisee+boolContainerBis.tailleUtilisee) != 0)
    {
        return ERROR_RESIZE_FAILED;
        //code erreur : echec du redimentionnement du tableau
    }
    for(int i=0; i<boolContainerBis.tailleUtilisee; i++)
    {
        Ajouter(boolContainerBis.tab[i]);
    }
    return 0;
} //----- Fin de Methode

/*
//------------------------------------------------------ Surcharge d'operateurs
BoolContainer & BoolContainer::operator = ( const BoolContainer & unBoolContainer )
// Algorithme :
//
{
} //----- Fin de operator =
*/

//-------------------------------------------- Constructeurs - destructeur

BoolContainer::BoolContainer (unsigned int nouvelleTaille):\
        tailleDispo(nouvelleTaille)
// Algorithme :
//
{
    #ifdef MAP
        cout << "Appel au constructeur par defaut de <BoolContainer>" << endl;
    #endif
```

```
    tailleUtilisee = 0;
    tab = new bool[tailleDispo];
} //------ Fin de BoolContainer

BoolContainer::BoolContainer (bool nouveauTab[],unsigned int nouvelleTaille):\
        tailleDispo(nouvelleTaille), tailleUtilisee(nouvelleTaille)
// Algorithme :
// copie valeur par valeur du tableau passe en parametre vers tab
{
        #ifdef MAP
                cout << "Appel au constructeur de <BoolContainer>" << endl;
        #endif
    tab = new bool[tailleDispo];
    for(int i=0; i<tailleDispo; i++)
    {
        tab[i]= nouveauTab[i];
    }
} //------ Fin de BoolContainer


BoolContainer::~BoolContainer ( )
{
    #ifdef MAP
        cout << "Appel au destructeur de <BoolContainer>" << endl;
    #endif
    delete [] tab;
} //------ Fin de ~BoolContainer
```

6.4 BoolContainer.h

```
/*************************************************************************
                        BoolContainer - description
                             -------------------
    debut                : 05/10/2015
    copyright            : (C) 2015 par Edern Haumont & Nicolas Six
*************************************************************************/

//---- Interface de la classe BoolContainer (fichier Boolcontainer.h) ----
#if ! defined ( BOOLCONTAINER_H )
#define BOOLCONTAINER_H

//------------------------------------------------------------ Constantes

const int DEFAULT_CONTAINER_SIZE = 10;

//------------------------------------------------------------------------
// La classe BoolContainer permet de definir des objet dont le but est de
// contenir des booleens. La capacite d'un BoolContainer est dynamique.
// La classe supporte les modifications sur son contenu via des methodes
// dediees
//------------------------------------------------------------------------

class BoolContainer
{
//------------------------------------------------------------------ PUBLIC

public:

    //----------------------------------------------------- Methodes publiques
    int Afficher(bool afficherCarateristique=true,bool afficherContenue=true) const;
    // Mode d'emploi :
    // Peut afficher la capacite utilisée et disponible suivie du contenu
    // du BoolContainer. Si le premier argument est vrais alors il y aura
    // affichage d'une ligne avec les caracterisitiques du tableau. Si le
    // segond est vrais alors il y aura affichage du contenue du tableau.
    // par défaut : affiche les deux bloques.
    // Contrat :
    //

    int Ajouter(bool b);
    // Mode d'emploi :
    // Prend en parametre le booleen qui sera ajoute a la fin de la liste.
    // Contrat :
    //

    int Retirer(unsigned int debut, unsigned int longueur = 1);
    // Mode d'emploi :
    // Prend en parametre l'index e partir auquel la suppression aura lieu
    // et en parametre optionnel, la longueur de l'intervalle d'index sur
    // lequel les suppressions auront lieu.
    // Contrat :
    //
```

```
int Ajuster(unsigned int nouvelletaille);
// Mode d'emploi :
// Prend en parametre la taille de collection desiree
// (superieure e la taille utilisee actuellement).
// Contrat :
//

int Reunir(const BoolContainer & boolContainerBis);
// Mode d'emploi :
// Prend en parametre une autre collection de booleens qui sera ajoutee à
// la fin de la collection courante.
// Contrat :
//



//------------------------------------------------------ Surcharge d'operateurs
BoolContainer & operator = ( const BoolContainer & unBoolContainer );
// Mode d'emploi :
// Non implemente
// Contrat :
// TODO verifier si elle n'est pas a implementer


//-------------------------------------------------- Constructeurs - destructeur
BoolContainer ( const BoolContainer & unBoolContainer );
// Mode d'emploi (constructeur de copie) :
// Non implemente
// Contrat :
//

BoolContainer(unsigned int nouvelleTaille = DEFAULT_CONTAINER_SIZE);
// Mode d'emploi :
// Prend en parametre optionnel la taille de la nouvelle collection
// Contrat :
//

BoolContainer(bool nouveauTab[],unsigned int nouvelleTaille);
// Mode d'emploi :
// Prend en parametre un tableau statique de booleens qui sera copie
// et sa taille
// Contrat :
// nouveauTab est de taille superieur ou égale à nouvelleTaille
// (si sa taille est supérieure à nouvelle taille seuls les élléments
// des case entre 0 et nouvelleTaille exclue seront copié dans
// le tableau)

virtual ~BoolContainer ( );

//------------------------------------------------------------------- PRIVE
```

```
private:

    unsigned int tailleUtilisee;
    unsigned int tailleDispo;
    bool *tab;
//------------------------------------------------------ Attributs prives
}; // class BoolContainer

#endif // BOOLCONTAINER_H
```

6.5 errorCode.h

```
// fichier de correspondance erreurs/codes d'erreur

#ifndef ERRORCODE_H
#define ERRORCODE_H

const int ERROR_NON_INTUITIVE_ADJUSTMENT = 100;
const int ERROR_RESIZE_FAILED = 101;
const int ERROR_OUT_OF_BOUNDARY = 102;

#endif //ERRORCODE_H
```