

Interprète d'un mini langage impératif

1) Introduction

Ce projet a pour but de définir un interprète d'un mini langage impératif. Ce langage est défini par un jeu **d'instructions** permettant la consultation et la mise à jour des variables de **l'environnement**. Les valeurs de ces variables sont décrites par des **expressions** arithmétiques ou booléennes. Cependant la représentation de ces valeurs, décrites par un **domaine d'interprétation**, n'est pas fixée et sera un paramètre de l'interprète. Suivant la représentation choisie, l'interprète calculera la valeur exacte des variables ou une valeur approchée indiquant simplement le type de la variable ou le signe de sa valeur. Il sera ainsi possible de vérifier les propriétés sur le programme (typage, terminaison, signe d'une variable, ...)

2) Vue d'ensemble

```
> java Main

+-----+
|   Interprete d'un mini langage imperatif!   |
+-----+

Choisissez un programme :
 1. Pgm 1 (test1:While)
 2. Pgm 2 (test2:While)
 3. Pgm 3 (test3:While)
 4. Pgm 4 (If & Affect)
 5. Pgm 5 (If & Affect)
 6. Pgm 6 (Somme & Produit)
 7. Pgm 7 (Somme & Produit)
 8. Pgm 8 (Des Tests)
 9. Pgm 9 (Des Tests)

 0. Quitter

Votre choix : 3

test3 =
{
var s = 0 ;
{
var x = 1 ;
{
while (<<(x == 0) == FAUX>>)
{
s = (s + x) ;
x = (x + 1) ;
}
x = (x + URAI) ;
}
}
}

Choisissez un environnement :
 1. Executer dans UalDom
 2. Executer dans TypeDom
 3. Executer dans NPDom

 0. Annuler

Votre choix : 3

Execution ...
Boucle infinie a cet endroit :
while (Urai)...
Erreur : somme entre Positif et Urai impossible!

Appuyez sur entree pour continuer!
```

3) Solutions retenues

La partie Java est connectée à un module écrit en C représentant un ensemble par une liste chaînée.
La partie Caml est connectée à un module écrit en C représentant un ensemble par un entier dont les bits positionnés à 1 dans la représentation binaire indiquent les éléments présents.

Pour l'opération de test d'égalité, il faut que les arguments soient de même type sinon il y a une erreur.

On ne peut pas déclarer deux fois la même variable dans deux blocs différents.

On peut affecter une valeur d'un autre type à une variable déclarée.

La condition du If & du While doit être de type booléen.

L'accès à une variable non déclarée provoque une erreur.

L'addition et la multiplication ne peuvent se faire qu'entre deux entiers.

4) Restrictions liées au projet

Ce mini langage permet de déclarer une variable, d'affecter une valeur à une variable, d'écrire des séquences d'instructions, d'effectuer un test « if (condition) then instructions else instructions », de réaliser une boucle tant que « while (condition) instructions » et d'afficher l'état de l'environnement.

Les opérations réalisables sont la comparaison (Egal), la somme et le produit de deux valeurs.

5) Partie Java

5.1) Fichiers C

alloc.h

```
/* Affiche le bilan de la gestion dynamique de la mémoire */
extern void AllocVerify(void);

/* Remplace malloc */
extern void *Malloc(size_t Size);

/* Remplace calloc */
extern void *Calloc(size_t Nmemb, size_t Size);

/* Remplace realloc */
extern void *Realloc(void *ptr, size_t Size);

/* Remplace free */
extern void Free(void *Ptr);
```

alloc.c

```
/* Actions possibles */
typedef enum Action { allouer, liberer, bilan } Action;

/* Fonction permettant la vérification de la gestion de la mémoire */
static void Compta(Action a)

/* Remplace malloc */
void *Malloc(size_t Size)

/* Remplace calloc */
void *Calloc(size_t Nmemb, size_t Size)

/* Remplace realloc */
void *Realloc(void *ptr, size_t Size)

/* Remplace free*/
void Free(void *Ptr)

/* Affiche le bilan de la gestion dynamique de la mémoire */
void AllocVerify(void)
```

listgen.h

```
/* Type Liste */
typedef struct Liste *Liste;

/* Création d'une liste vide, Retourne NULL si problème */
extern Liste ListAllouer(void *(*EltCopier)(void *), void (*EltLiberer)(void *));

/* Libération complète de l'espace mémoire occupé par une liste */
extern void ListLiberer(Liste *pl);

/* Ajout d'un élément en fin de liste, Retourne NULL si problème */
extern Liste ListAjouterEnFin(Liste l, void *pElt);

/* Création d'une copie d'une liste, Retourne NULL si problème */
extern Liste ListCopier(Liste l);
```

```

/* Initialisation de la position courante dans une liste */
extern void ListInitPos(Liste l);

/* Détermine si la fin de liste est atteinte
 * Retourne 0 s'il existe un élément en position courante ou 1 sinon */
extern int ListFin(Liste l);

/* Incrémentation de la position courante dans une liste
 * Sans effet si la fin de liste est atteinte */
extern void ListAvancer(Liste l);

/* Obtention de l'élément qui se trouve en position courante
 * Retourne NULL si la fin de liste est atteinte */
extern void *ListCourant(Liste l);

/* Modification de la valeur de l'élément qui se trouve en position courante
 * Sans effet si la fin de liste est atteinte */
extern void ListModifieurCourant(Liste l, void *pNouveau);

/* Suppression de l'élément qui se trouve en position courante et
 * incrémentation de la position courante
 * Sans effet si la fin de liste est atteinte */
extern void ListSupprimerCourant(Liste l);

/* Nombre d'éléments présents dans une liste */
extern unsigned int ListNbElts(Liste l);

```

listgen.c

```

/* Descripteur de liste */
struct Liste

/* Cellules de la liste */
struct Cellule

/* Création d'une liste vide, Retourne NULL si problème */
Liste ListAllouer(void *(*EltCopier)(void *), void (*EltLiberer)(void *))

/* Libération des zones mémoire occupées par les cellules d'une liste */
static void CellLiberer(struct Cellule *Tete, void (*EltLiberer)(void *))

/* Libération complète de l'espace mémoire occupé par une liste */
void ListLiberer(Liste *pl)

/* Ajout d'un élément en fin de liste, Retourne NULL si problème */
Liste ListAjouterEnFin(Liste l, void *pElt)

/* Création d'une copie d'une liste, Retourne NULL si problème */
Liste ListCopier(Liste l)

/* Initialisation de la position courante dans une liste */
void ListInitPos(Liste l)

/* Détermine si la fin de liste est atteinte
 * Retourne 0 s'il existe un élément en position courante ou 1 sinon */
int ListFin(Liste l)

/* Incrémentation de la position courante dans une liste
 * Sans effet si la fin de liste est atteinte */
void ListAvancer(Liste l)

/* Obtention de l'élément qui se trouve en position courante
 * Retourne NULL si la fin de liste est atteinte */
void *ListCourant(Liste l)

```

```

/* Modification de la valeur de l'élément qui se trouve en position courante
 * Sans effet si la fin de liste est atteinte */
void ListModifieurCourant(Liste l, void *pNouveau)

/* Suppression de l'élément qui se trouve en position courante et
 * incrémentation de la position courante
 * Sans effet si la fin de liste est atteinte */
void ListSupprimerCourant(Liste l)

/* Nombre d'éléments présents dans une liste */
unsigned int ListNbElts(Liste l)

```

listNP.h

```

/* Copie d'un entier. */
extern void NPCopier(void *p) ;

/* Liberation de la zone memoire occupee par un entier. */
extern void NPLiberer(void *p) ;

/* Afficher le contenu de la liste. */
extern void Afficher(Liste l) ;

/* Cree le test d'egalite des elements de lx et l. */
extern Liste egall(Liste l, Liste lx) ;

/* Cree l'addition des elements de lx et l. */
extern Liste ajoutL(Liste l, Liste lx) ;

/* Cree la multiplication des elements de lx et l. */
extern Liste multL(Liste l, Liste lx) ;

/* Generalisation de deux listes. */
extern Liste genL(Liste l, Liste lx) ;

/* Verifie si deux listes sont de meme type. (Booleen & Entier) */
extern int memeType(Liste l, Liste lx) ;

/* Recherche l'element val dans la liste l. */
extern int recherche(Liste l, int val) ;

/* Renvoie la somme des elements contenus dans la liste. */
extern int convertObjet(Liste l) ;

/* Cree une liste a partir d'une valeur. */
extern void convertListe(int val, Liste l) ;

/* Verifie si la liste est de type Entier */
extern int estEntier(Liste l) ;

/* Verifie si la liste est de type Booleen */
extern int estBooleen(Liste l) ;

```

listNP.c

```

/* Copie d'un entier. */
void NPCopier(void *p)

/* Liberation de la zone memoire occupee par un entier. */
void NPLiberer(void *p)

/* Afficher le contenu de la liste. */
void Afficher(Liste l)

```

```

/* Verifie si la liste est de type Entier */
int estEntier(Liste l)

/* Verifie si la liste est de type Booleen */
int estBooleen(Liste l)

/* Cree le test d'egalite des elements de lx et l. */
Liste egalL(Liste l, Liste lx)

/* Cree l'addition des elements de lx et l. */
Liste ajoutL(Liste l, Liste lx)

/* Cree la multiplication des elements de lx et l. */
Liste multL(Liste l, Liste lx)

/* Generalisation de deux listes. */
Liste genL(Liste l, Liste lx)

/* Verifie si deux listes sont de meme type. (Booleen & Entier) */
int memeType(Liste l, Liste lx)

/* Recherche l'element val dans la liste l. */
int recherche(Liste l, int val)

/* Renvoie la somme des elements contenus dans la liste. */
int convertObjet(Liste l)

/* Cree une liste a partir d'une valeur. */
void convertListe(int val, Liste l)

```

NPDom.h

Généré par la machine. (JNI)

NPDom.c

```

/* La methode egal retourne la representation du test d'egalite de deux valeurs du
domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_egal (JNIEnv * env, jobject np, jobject xdom)

/* La methode vrai retourne la representation de la constante VRAI sur le domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_vrai (JNIEnv * env, jobject np)

/* La methode faux retourne la representation de la constante FAUX sur le domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_faux (JNIEnv * env, jobject np)

/* La methode peut_etre_vrai retourne true si la valeur du domaine peut représenter
la valeur vrai. */
JNIEXPORT jboolean JNICALL Java_NPDom_peut_letre_lvrai (JNIEnv * env, jobject np)

/* La methode peut_etre_faux retourne false si la valeur du domaine peut représenter
la valeur faux. */
JNIEXPORT jboolean JNICALL Java_NPDom_peut_letre_lfaux (JNIEnv * env, jobject np)

/* La methode entier retourne la representation d'un entier sur le domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_entier (JNIEnv * env, jobject np, jint i)

/* La methode add retourne la representation de l'addition de deux valeurs du
domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_add (JNIEnv * env, jobject np, jobject xdom)

```

```

/* La methode mul retourne la representation de la multiplication de deux valeurs du
domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_mul (JNIEnv * env, jobject np, jobject xdom)

/* La methode generaliser retourne une valeur plus generale sur le domaine des deux
valeurs du domaine. */
JNIEXPORT jobject JNICALL Java_NPDom_generaliser (JNIEnv * env, jobject np, jobject
xdom)

/* La methode toString retourne une chaine de caractere de la valeur du domaine. */
JNIEXPORT jstring JNICALL Java_NPDom_toString (JNIEnv * env, jobject np)

/* La methode equals retourne vrai si les 2 valeurs sont egales */
JNIEXPORT jboolean JNICALL Java_NPDom_equals (JNIEnv * env, jobject np, jobject xdom)

```

5.2) Fichiers Java

Environnement.java

```

// La classe Environnement generique associe a une variable une valeur.
public class Environnement<T extends Domaine<T>> extends Hashtable<String,T>
{
    private boolean valide ; // valide du domaine
    private T dom ; // type de representation du domaine

    // Constructeur de Environnement : Creation d'un environnement vide.
    public Environnement(T _dom)

    // Constructeur de Environnement : Creation d'un environnement vide avec
    // capacite initiale.
    public Environnement(int capaciteinitiale, T _dom)

    // Constructeur de Environnement : Creation d'un environnement vide avec
    // capacite initiale et facteur de charge.
    public Environnement(int capaciteinitiale, float facteurdecharge, T _dom)

    // La methode ajouter ajoute un couple (variable, valeur) dans l'environnement.
    public void ajouter(String s, T x)

    // La methode modifier modifie la valeur d'une variable dans l'environnement.
    public void modifier(String s, T x)

    // La methode supprimer supprime une variable dans l'environnement.
    public void supprimer(String s)

    // La methode dupliquer duplique l'environnement.
    public Environnement<T> dupliquer()

    // La methode comparer compare deux environnements.
    public boolean comparer(Environnement <T> env)

    // La methode invalider invalide l'environnement.
    public void invalider()

    // La methode estValide teste la validite d'un environnement.
    public boolean estValide()

    // La methode lineariser convertit en chaine de caracteres
    // l'etat de l'environnement.
    public String toString()

    // La methode combiner combine deux environnements.
    public void combiner(Environnement<T> env)

```

```

// La methode getDom retourne le type de domaine.
public T getDom()

// La methode valeur retourne la valeur d'une variable sur le domaine.
public T valeur(String s)
}

```

Domaine.java

```

// Les domaines d'interpretation sont definis par les classes implantant l'interface
// Domaine.
public interface Domaine<T>
{
// La methode egal retourne la representation du test d'egalite de deux valeurs
// du domaine.
public T egal(T x);

// La methode vrai retourne la representation de la constante vrai sur le domaine.
public T vrai();

// La methode faux retourne la representation de la constante faux sur le domaine.
public T faux();

// La methode peut_etre_vrai retourne true si la valeur du domaine peut
// représenter la valeur vrai.
public boolean peut_etre_vrai();

// La methode peut_etre_faux retourne false si la valeur du domaine peut
// représenter la valeur faux.
public boolean peut_etre_faux();

// La methode entier retourne la representation d'un entier sur le domaine.
public T entier(int i);

// La methode add retourne la representation de l'addition de deux valeurs
// du domaine.
public T add(T x);

// La methode mul retourne la representation de la multiplication de deux valeurs
// du domaine.
public T mul(T x);

// La methode generaliser retourne une valeur plus generale sur le domaine des
// deux valeurs du domaine.
public T generaliser(T x);

// La methode toString convertit en chaine de caracteres la valeur du domaine.
public String toString();
}

```

ValDom.java

```

// La classe ValDom represente les entiers et booleens par les entiers et les
// booleens du langage considere.
public class ValDom implements Domaine <ValDom>
{
// La methode egal retourne la representation du test d'egalite de deux valeurs du
// domaine ValDom.
public ValDom egal(ValDom x)

// La methode vrai retourne la representation de la constante vrai sur le domaine
// ValDom.
public ValDom vrai()
}

```

```

// La methode faux retourne la representation de la constante faux sur le domaine
// ValDom.
public ValDom faux()

// La methode peut_etre_vrai teste si la valeur du domaine ValDom peut représenter
// la valeur vrai.
public boolean peut_etre_vrai()

// La methode peut_etre_faux teste si la valeur du domaine peut représenter la
// valeur faux.
public boolean peut_etre_faux()

// La methode entier retourne la representation d'un entier sur le domaine ValDom.
public ValDom entier(int i)

// La methode add retourne la representation de l'addition de deux valeurs du
// domaine ValDom.
public ValDom add(ValDom x)

// La methode mul retourne la representation de la multiplication de deux valeurs
// du domaine ValDom.
public ValDom mul(ValDom x)

// La methode generaliser retourne une valeur plus generale sur le domaine des
// deux valeurs du domaine ValDom.
public ValDom generaliser(ValDom x)

// La methode toString convertit en chaine de caracteres la valeur du domaine
// ValDom.
public String toString()

// La methode equals teste si les deux valeurs du domaine ValDom sont egales.
public boolean equals(Object x)
}

```

Bool.java

```

// La classe Bool represente les booleens par les booleens du langage considere.
public class Bool extends ValDom
{
    private boolean b ;    // Un booleen.

    // Constructeur de Bool : initialisation du booleen.
    public Bool(boolean b)

    // La methode get retourne la valeur du booleen.
    public boolean get()

    // La methode egal retourne la representation du test d'egalite de deux valeurs du
    // domaine ValDom.
    public ValDom egal(ValDom x)

    // La methode peut_etre_vrai teste si la valeur du domaine ValDom peut représenter
    // la valeur vrai.
    public boolean peut_etre_vrai()

    // La methode peut_etre_faux teste si la valeur du domaine ValDom peut représenter
    // la valeur faux.
    public boolean peut_etre_faux()

    // La methode generaliser retourne une valeur plus generale sur le domaine des
    // deux valeurs du domaine ValDom.
    public ValDom generaliser(ValDom x)

    // La methode toString convertit en chaine de caracteres la valeur (booleen) du
    // domaine ValDom.

```

```

public String toString()

// La methode equals teste si les deux valeurs du domaine ValDom sont egales.
public boolean equals(Object x)
}

```

Int.java

```

// La classe Int represente les entiers par les entiers du langage considere.
public class Int extends ValDom
{
    private int i ;    // Un entier.

    // Constructeur de Int : initialisation de l'entier.
    public Int(int i)

    // La methode get retourne la valeur de l'entier.
    public int get()

    // La methode egal retourne la representation du test d'egalite de deux valeurs du
    // domaine ValDom.
    public ValDom egal(ValDom x)

    // La methode add retourne la representation de l'addition de deux valeurs du
    // domaine ValDom.
    public ValDom add(ValDom x)

    // La methode mul retourne la representation de la multiplication de deux valeurs
    // du domaine ValDom.
    public ValDom mul(ValDom x)

    // La methode generaliser retourne une valeur plus generale sur le domaine des
    // deux valeurs du domaine ValDom.
    public ValDom generaliser(ValDom x)

    // La methode toString convertit en chaine de caracteres la valeur (int) du
    // domaine ValDom.
    public String toString()

    // La methode equals teste si les deux valeurs du domaine ValDom sont egales.
    public boolean equals(Object x)
}

```

TypeDom.java

```

// La classe TypeDom represente les entiers par la constante TInt et les booleen par
// la constante TBool.
public enum TypeDom implements Domaine <TypeDom>
{
    // Les deux valeurs du domaine
    TBool, TInt;

    // La methode egal retourne la representation du test d'egalite de deux valeurs du
    // domaine TypeDom.
    public TypeDom egal(TypeDom x)

    // La methode vrai retourne la representation de la constante vrai sur le domaine
    // TypeDom.
    public TypeDom vrai()

    // La methode faux retourne la representation de la constante faux sur le domaine
    // TypeDom.
    public TypeDom faux()
}

```

```

// La methode peut_etre_vrai teste si la valeur du domaine TypeDom peut
// représenter la valeur vrai.
public boolean peut_etre_vrai ()

// La methode peut_etre_faux teste si la valeur du domaine TypeDom peut
// représenter la valeur faux.
public boolean peut_etre_faux ()

// La methode entier retourne la representation d'un entier sur le domaine
// TypeDom.
public TypeDom entier(int i)

// La methode add retourne la representation de l'addition de deux valeurs du
// domaine TypeDom.
public TypeDom add(TypeDom x)

// La methode mul retourne la representation de la multiplication de deux valeurs
// du domaine TypeDom.
public TypeDom mul(TypeDom x)

// La methode generaliser retourne une valeur plus generale sur le domaine des
// deux valeurs du domaine TypeDom.
public TypeDom generaliser(TypeDom x)

// La methode toString convertit en chaine de caracteres la valeur du domaine
// TypeDom.
public String toString()
}

```

NPDom.java

```

// Represente les entiers par leur signe et les booleen par vrai, faux ou quelconque.
public class NPDom implements Domaine <NPDom>
{
    private int val;    // Valeur.

    // Constructeur de NPDom : initialisation de la valeur.
    public NPDom(int v)

    // La methode egal retourne la representation du test d'egalite de deux valeurs du
    // domaine NPDom.
    public native NPDom egal(NPDom x) ;

    // La methode vrai retourne la representation de la constante vrai sur le domaine
    // NPDom.
    public native NPDom vrai () ;

    // La methode faux retourne la representation de la constante faux sur le domaine
    // NPDom.
    public native NPDom faux () ;

    // La methode peut_etre_vrai teste si la valeur du domaine NPDom peut représenter
    // la valeur vrai.
    public native boolean peut_etre_vrai () ;

    // La methode peut_etre_faux teste si la valeur du domaine NPDom peut représenter
    // la valeur faux.
    public native boolean peut_etre_faux () ;

    // La methode entier retourne la representation d'un entier sur le domaine NPDom.
    public native NPDom entier(int i) ;

    // La methode add retourne la representation de l'addition de deux valeurs du
    // domaine NPDom.
    public native NPDom add(NPDom x) ;
}

```

```

// La methode mul retourne la representation de la multiplication de deux valeurs
// du domaine NPDom.
public native NPDom mul(NPDom x) ;

// La methode generaliser retourne une valeur plus generale sur le domaine des
// deux valeurs du domaine NPDom.
public native NPDom generaliser(NPDom x) ;

// La methode toString convertit en chaine de caracteres la valeur du domaine
// NPDom.
public native String toString() ;

// La methode equals teste si les deux valeurs du domaine NPDom sont egales.
public native boolean equals(Object x) ;

// Library
static {
    System.loadLibrary("NPDom") ;
}
}

```

Expression.java

```

// Les expressions sont definies par les classes implantant l'interface Expression.
public interface Expression
{
    // La methode eval permet d'evaluer les expressions dans l'environnement passe
    // en parametre.
    public <T extends Domaine<T>> T eval(Environnement<T> env) ;
}

```

Vrai.java

```

// La classe Vrai permet d'evaluer un boolean vrai dans l'environnement.
public class Vrai implements Expression
{
    private boolean b ; // Un boolean.

    // Constructeur de Vrai : Initialisation du boolean a vrai.
    public Vrai()

    // La methode eval permet d'evaluer un boolean vrai dans l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres le boolean vrai.
    public String toString()
}

```

Faux.java

```

// La classe Faux permet d'evaluer un boolean faux dans l'environnement.
public class Faux implements Expression
{
    private boolean b ; // Un boolean.

    // Constructeur de Faux : Initialisation du boolean a faux.
    public Faux()

    // La methode eval permet d'evaluer un boolean faux dans l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres le boolean faux.
    public String toString()
}

```

Entier.java

```
// La classe Entier permet d'évaluer un entier dans l'environnement.
public class Entier implements Expression
{
    private int ent ;    // Un entier.

    // Constructeur de Entier : Initialisation de l'entier.
    public Entier(int i)

    // La methode eval permet d'évaluer un entier dans l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres l'entier.
    public String toString()
}
}
```

Variable.java

```
// La classe Variable permet d'évaluer une variable dans l'environnement.
public class Variable implements Expression
{
    private String var ;    // Une variable.

    // Constructeur de Variable : Initialisation du nom de la variable.
    public Variable(String s)

    // La methode eval permet d'évaluer une variable dans l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres le nom de la variable.
    public String toString()
}
}
```

Egal.java

```
//La classe Egal permet de tester l'égalite de deux expressions dans l'environnement.
public class Egal implements Expression
{
    private Expression e1 ;    // Expression 1.
    private Expression e2 ;    // Expression 2.

    // Constructeur de Egal : Initialisation des deux expressions.
    public Egal(Expression expr1, Expression expr2)

    // La methode eval permet de tester l'égalite des deux expressions dans
    // l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres le test d'égalite
    // des deux expressions.
    public String toString()
}
}
```

Somme.java

```
//La classe Somme permet d'évaluer la somme de deux expressions dans l'environnement.
public class Somme implements Expression
{
    private Expression e1 ;    // Expression 1.
    private Expression e2 ;    // Expression 2.
```

```

// Constructeur de Somme : Initialisation des deux expressions.
public Somme(Expression expr1, Expression expr2)

// La methode eval permet d'evaluer la somme de deux expressions dans
// l'environnement courant.
public <T extends Domaine<T>> T eval(Environnement<T> env)

// La methode toString convertit en chaine de caracteres l'expression de la somme
// des deux expressions.
public String toString()
}

```

Produit.java

```

// La classe Produit permet d'evaluer le produit de deux expressions dans
// l'environnement.
public class Produit implements Expression
{
    private Expression e1 ; // Expression 1.
    private Expression e2 ; // Expression 2.

    // Constructeur de Produit : Initialisation des deux expressions.
    public Produit(Expression expr1, Expression expr2)

    // La methode eval permet d'evaluer le produit de deux expressions dans
    // l'environnement courant.
    public <T extends Domaine<T>> T eval(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres l'expression du produit
    // des deux expressions.
    public String toString()
}

```

Instruction.java

```

// Les instructions sont definies par les classes implantant l'interface Instruction.
public interface Instruction
{
    // La methode exec permet aux instructions de mettre a jour l'environnement
    // passe en parametre.
    public <T extends Domaine<T>> void exec(Environnement<T> env) ;
}

```

Affect.java

```

// La classe Affect permet d'affecter une expression a une variable.
public class Affect implements Instruction
{
    private String var ; // La variable.
    private Expression val ; // L'expression (la valeur de la variable).

    // Constructeur de Affect : Initialisation de la variable et de l'expression.
    public Affect(String variable, Expression valeur)
    // La methode exec permet de modifier la valeur de la variable var dans
    // l'environnement courant.
    public <T extends Domaine<T>> void exec(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres l'instruction
    // d'affectation.
    public String toString()
}

```

Declare.java

```
// La classe Declare permet de declarer une nouvelle variable.
public class Declare implements Instruction
{
    private String var ;           // La variable.
    private Expression val ;      // L'expression (la valeur de la variable).
    private Instruction inst ;    // L'instruction(definissant la portee de la variable).

    // Constructeur de Declare : Initialisation de la variable, de l'expression et
    // de la portee.
    public Declare(String variable, Expression valeur, Instruction instruction)

    // La methode exec permet d'ajouter une nouvelle variable initialisee dans
    // l'environnement courant.
    public <T extends Domaine<T>> void exec(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres l'instruction
    // de declaration.
    public String toString()
}
}
```

If.java

```
// La classe If permet de realiser un test (si cond alors inst1 sinon inst2).
public class If implements Instruction
{
    private Expression cond ;     // La condition.
    private Instruction inst1 ;   // Instruction 1.
    private Instruction inst2 ;   // Instruction 2.

    // Constructeur de If : Initialisation de la condition et des deux instructions.
    public If(Expression condition, Instruction instruction1, Instruction instruction2)

    // La methode exec permet d'executer la structure de controle If dans
    // l'environnement courant.
    public <T extends Domaine<T>> void exec(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres la structure de
    // controle If.
    public String toString()
}
}
```

While.java

```
// La classe While permet de realiser une boucle (tant_que cond : inst).
public class While implements Instruction
{
    private Expression cond ;     // La condition.
    private Instruction inst ;    // L'instruction.

    // Constructeur de While : Initialisation de la condition et de l'instruction.
    public While(Expression condition, Instruction instruction)

    // La methode exec permet d'executer la structure de controle While dans
    // l'environnement courant.
    public <T extends Domaine<T>> void exec(Environnement<T> env)

    // La methode toString convertit en chaine de caracteres la structure de
    // controle While.
    public String toString()
}
}
```

Seq.java

```
// La classe Seq permet d'executer deux instructions.
public class Seq implements Instruction
{
    private Instruction inst1 ;    // Instruction 1.
    private Instruction inst2 ;    // Instruction 2.

    // Constructeur de Seq : Initialisation de deux instructions.
    public Seq(Instruction instruction1, Instruction instruction2)

    // La methode exec permet d'executer deux instructions dans l'environnement
    // courant.
    public <T extends Domaine<T>> void exec (Environnement<T> env)

    // La methode toString convertit en chaine de caracteres la sequence des
    // deux instructions.
    public String toString()
}
}
```

Trace.java

```
// La classe Trace permet d'afficher le contenu de l'environnement.
public class Trace implements Instruction
{
    private String msg ; // Message affiche avant le contenu de l'environnement.

    // Constructeur de Trace : Initialisation du message.
    public Trace(String message)

    // La methode exec permet d'affiche le message passe en parametre ainsi que
    // le contenu de l'environnement courant.
    public <T extends Domaine<T>> void exec (Environnement<T> env)

    // La methode toString convertit en chaine de caracteres l'instruction trace
    // de l'environnement.
    public String toString()
}
}
```

Menu.java

```
// La classe Menu permet de gerer un systeme de menu pour le programme.
public class Menu
{
    private int no_pgm ;    // Numero du programme choisi.
    private int no_env ;    // Numero de l'environnement choisi.
    private String spgm ;    // Le programme. (chaine)
    private Programme pgm ; // Le programme. (instruction)

    // Constructeur de Menu : Initialisation des variables.
    public Menu()

    // La methode afficher_entete permet d'afficher dans un cadre
    // "Interprete d'un mini langage imperatif!".
    public void afficher_entete()

    // La methode afficher_menu_pgm permet d'afficher un menu presentant les
    // differents programmes diponibles.
    public void afficher_menu_pgm()

    // La methode afficher_menu_env permet d'afficher un menu presentant les
    // differents domaines dans lesquels on peut executer un programme.
    public void afficher_menu_env()
}
```

```

// La methode modifier_spgm permet de modifier la chaine de caracteres
// representant le programme en cours.
public void modifier_spgm(int no)

// La methode afficher_spgm permet d'afficher le programme en cours.
public void afficher_spgm()

// La methode saisir_int permet de recuperer et valider un entier saisi au
// clavier par un utilisateur.
public int saisir_int() throws IOException, NumberFormatException

// La methode lancer permet de gerer et de lancer le systeme de menu.
public void lancer() throws IOException, NumberFormatException
}

```

Programme.java

```

// La classe Programme permet de construire un programme et de l'executer dans
// un environnement.
public class Programme
{
    private Environnement<NPDom> npenv ;           // Environnement NPDom.
    private Environnement<ValDom> valenv ;       // Environnement ValDom.
    private Environnement<TypeDom> typeenv ;     // Environnement TypeDom.
    private Instruction inst ;                   // Instruction.

    // Constructeur de Programme : Initialisation des environnements.
    public Programme ()

    // La methode initialiser permet d'initialiser les environnements.
    public void initialiser ()

    // La methode getInst permet de recuperer l'instruction.
    public Instruction getInst ()

    // La methode construire_instruction permet de construire une instruction
    // en fonction du numero choisi.
    public void construire_instruction(int pgm)

    // La methode executer execute l'instruction courante en fonction de
    // l'environnement choisi.
    public void executer(int env)
}

```

Main.java

```

// La classe Main permet de lancer l'interprete d'un mini langage imperatif.
public class Main
{
    // La methode main debute l'execution de l'application Java.
    public static void main( String[] args ) throws Exception
}

```

6) Partie Caml

6.1) Fichiers C

Cdom.h

```
/* Retourne la representation du test d'egalite de deux valeurs du domaine. */
extern CAMLprim value Cegal(value v1, value v2) ;

/* Retourne la representation de la constante VRAI sur le domaine. */
extern CAMLprim value Cvrai() ;

/* Retourne la representation de la constante FAUX sur le domaine. */
extern CAMLprim value Cfaux() ;

/* Retourne true si la valeur du domaine peut représenter la valeur vrai. */
extern CAMLprim value Cpvrai(value v1) ;

/* Retourne false si la valeur du domaine peut représenter la valeur faux. */
extern CAMLprim value Cpfaux(value v1) ;

/* Retourne la representation d'un entier sur le domaine. */
extern CAMLprim value Centier(value v1) ;

/* Retourne la representation de l'addition de deux valeurs du domaine. */
extern CAMLprim value Cadd(value v1, value v2) ;

/* Retourne la representation de la multiplication de deux valeurs du domaine. */
extern CAMLprim value Cmul(value v1, value v2) ;

/* Retourne une valeur plus generale sur le domaine des deux valeurs du domaine. */
extern CAMLprim value Cgeneraliser(value v1, value v2) ;

/* Retourne une chaine de caractere de la valeur du domaine. */
extern CAMLprim value Cprint(value v1) ;
```

cdom.c

```
typedef unsigned char NPval ;

/* determine si un entier represente un Entier sur NPDom */
int estEntier (int v)

/* determine si un entier represente un Booleen sur NPDom */
int estBooleen (int v)

/* Retourne la representation du test d'egalite de deux valeurs du domaine. */
CAMLprim value Cegal(value v1, value v2)

/* Retourne la representation de la constante VRAI sur le domaine. */
CAMLprim value Cvrai()

/* Retourne la representation de la constante FAUX sur le domaine. */
CAMLprim value Cfaux()

/* Retourne true si la valeur du domaine peut représenter la valeur vrai. */
CAMLprim value Cpvrai(value v)

/* Retourne false si la valeur du domaine peut représenter la valeur faux. */
CAMLprim value Cpfaux(value v)

/* Retourne la representation d'un entier sur le domaine. */
CAMLprim value Centier(value v)
```

```

/* Retourne la representation de l'addition de deux valeurs du domaine. */
CAMLprim value Cadd(value v1, value v2)

/* Retourne la representation de la multiplication de deux valeurs du domaine. */
CAMLprim value Cmul(value v1, value v2)

/* Retourne une valeur plus generale sur le domaine des deux valeurs du domaine. */
CAMLprim value Cgeneraliser(value v1, value v2)

/* Retourne une chaine de caractere de la valeur du domaine. */
CAMLprim value Cprint(value v)

```

6.2) Fichiers Caml

env.ml

```

(* Module Env *)
module Env(D:Domaine.Domaine) = struct
  .....
end

(* Creer un environnement vide. *)
val creer : (string*t) list

(* Renvoie true si la variable x est presente dans l'environnement sinon false. *)
val existe : string -> (string*t) list -> boolean

(* Ajoute le couple(Variable,Valeur) s'il n'existe pas sinon c'est un erreur. *)
val ajouter : string -> t -> (string*t) list -> (string*t) list

(* Modifie la valeur de la variable x par v si la variable x existe sinon c'est une
erreur. *)
val modifier : string -> t -> (string*t) list -> (string*t) list

(* Supprime la variable x de l'environnement si x n'y est pas alors il ne se passe
rien. *)
val supprimer : string -> (string*t) list -> (string*t) list

(* Renvoie la valeur associée a la variable x dans l'environnement si x n'existe pas
alors c'est une erreur. *)
val get : string -> (string*t) list -> t

(* Renvoie une chaine de caracteres contenant les valeurs de l'environnement. *)
val print : (string*t) list -> string

(* Environnement invalide. *)
exception Invalide of string
let err_invalide = Invalide "Environnement invalide.\n"

(* Combinaison de deux environnements *)
val combiner : (string*t) list -> (string*t) list -> (string*t) list

```

domaine.mli

```

(* Module Domaine *)
module type Domaine = sig
  type t
  .....
end

```

```

(* Retourne vrai ou faux. *)
val egal : t -> t -> t

(* Retourne la representation des constantes vrai et faux. *)
val vrai : t
val faux : t

(* Retourne true si l'argument peut représenter la valeur vrai. *)
val peut_etre_vrai : t -> bool

(* Retourne true si l'argument peut représenter la valeur faux. *)
val peut_etre_faux : t -> bool

(* Retourne la representation d'un entier. *)
val entier : int -> t

(* Retourne la somme des 2 arguments. *)
val add : t -> t -> t

(* Retourne le produit des 2 arguments. *)
val mul : t -> t -> t

(* Retourne lorsque c'est possible une valeur plus generale que les deux parametres.
*)
val generaliser : t -> t -> t

(* Transforme l'argument en une chaine de caracteres. *)
val print : t -> string

```

valdom.ml

```

(* Module ValDom *)
module Valdom : Domains.Domaine = struct
  type t = Int of int | Bool of bool
  .....
end

(*Comparaison de 2 entiers ou 2 booleens*)
val egal : t -> t -> t

(*Retourne la representation de la constante VRAI sur le domaine *)
val vrai : t

(*Retourne la representation de la constante FAUX sur le domaine *)
val faux : t

(* Retourne true si l'argument peut représenter la valeur vrai. *)
val peut_etre_vrai : t -> t

(* Retourne true si l'argument peut représenter la valeur faux. *)
val peut_etre_faux : t -> t

(* Retourne la representation d'un entier sur le domaine *)
val entier : t

(* Retourne la somme des 2 arguments. *)
val add : t -> t -> t

(* Retourne le produit des 2 arguments. *)
val mul : t -> t -> t

(* Retourne lorsque c'est possible une valeur plus generale que les deux parametres.
*)
val generaliser : t -> t -> t

```

```
(* Transforme l'argument en une chaine de caracteres. *)
val print : t -> string
```

typedom.ml

```
(* Module TypeDom *)
module Typedom : Domaine.Domaine = struct
  type t = TInt | TBool
  . . . .
end

(*Comparaison de 2 entiers ou 2 booleens *)
val egal : t -> t -> t

(*Retourne la representation de la constante VRAI sur le domaine *)
val vrai : t

(*Retourne la representation de la constante FAUX sur le domaine *)
val faux : t

(* Retourne true si l'argument peut représenter la valeur vrai. *)
val peut_etre_vrai : t -> t

(* Retourne true si l'argument peut représenter la valeur faux. *)
val peut_etre_faux : t -> t

(* Retourne la representation d'un entier sur le domaine *)
val entier : t

(* Retourne la somme des 2 arguments. *)
val add : t -> t -> t

(* Retourne le produit des 2 arguments. *)
val mul : t -> t -> t

(* Retourne lorsque c'est possible une valeur plus generale que les deux parametres. *)
val generaliser : t -> t -> t

(* Transforme l'argument en une chaine de caracteres. *)
val print : t -> string
```

cdom.ml

```
(* Module Cdom *)
module Cdom : Domaine.Domaine = struct
  type t = int
  . . . .
end

(* Comparaison de 2 entiers ou 2 booleens *)
external egal : t -> t -> t = "Cegal"

(* Determine si c'est VRAI sur le domaine *)
external uvrai : unit -> t = "Cvrai"

(* Determine si c'est sur le domaine*)
external ufaux : unit -> t = "Cfaux"

(* Retourne la representation de la constante VRAI sur le domaine *)
let vrai = uvrai ()
```

```

(* Retourne la representation de la constante VRAI sur le domaine *)
let faux = ufaux ()

(* Determine si ca peut etre VRAI sur le domaine *)
external peut_etre_vrai : t -> bool = "Cpvrai"

(* Determine si ca peut etre FAUX sur le domaine *)
external peut_etre_faux : t -> bool = "Cpfaux"

(* Retourne la representation d'un entier sur le domaine *)
external entier: int -> t = "Centier"
(* Retourne la somme des 2 arguments. *)
external add : t -> t -> t = "Cadd"

(* Retourne la multiplication des 2 arguments. *)
external mul : t -> t -> t = "Cmul"

(* Retourne lorsque c'est possible une valeur plus generale que les deux parametres.
*)
external generaliser : t -> t -> t = "Cgeneraliser"

(* Transforme l'argument en une chaine de caracteres. *)
external print : t -> string = "Cprint"

```

langage.ml

```

(* Module Langage *)
module Langage = struct

  (* Expression *)
  type expr =
    Somme of expr * expr
  | Egal of expr * expr
  | Vrai
  | Faux
  | Entier of int
  | Produit of expr * expr
  | Variable of string

  (* Instruction *)
  type inst =
    Declare of string * expr * inst
  | Affect of string * expr
  | If of expr * inst * inst
  | Seq of inst * inst
  | While of expr * inst
  | Trace of string

end

```

meval.ml

```

(* Module Meval *)
module Meval(D:Domaine.Domaine) = struct
  .....
end

(* Evalue une expression dans un environnement. *)
val eval : (string*t) list -> expr -> t

```

mexec.ml

```
(* Module Meval *)
module Mexec(D:Domaine.Domaine) = struct
  .....
end

(* Execute une instruction dans un environnement *)
val exec : (string*t) list -> inst -> (string*t) list
```

mprint.ml

```
(* Module Meval *)
module Mprint = struct
  .....
end

(* Transforme l'expression en une chaine de caracteres. *)
val printex : (string*t) list -> expr -> string

(* Transforme l'instruction en une chaine de caracteres. *)
val printex : (string*t) list -> expr -> string
```

main.ml

```
(* Creation de l'environnement dans les domaine de EnvValDom, EnvTypeDom, EnvCDom. *)
let envValDom = EnvValDom.creer() ;;
let envTypeDom = EnvTypeDom.creer() ;;
let envCDom = EnvCDom.creer() ;;

(* Fonction d'affichage de l'entete. *)
val afficher_entete : unit -> unit = <fun>

(* Fonction d'affichage du menu programme. *)
val afficher_menu_pgm : unit -> unit = <fun>

(* Fonction d'affichage du menu environnement. *)
val afficher_menu_env : unit -> unit = <fun>

(* Fonction affichant le code source d'un programme. *)
val afficher_pgm : unit -> unit = <fun>

(* Reinitialisation des environnements. *)
val initialiser_env : unit -> unit = <fun>

(* Fonction lancant le système de menu. *)
val lancer : unit -> unit = <fun>
```

7) Problèmes rencontrés

Compréhension du sujet.

Utilisation de la version Java 1.5.0 beta2.

Utilisation du JNI (Java Native Interface).

Comparaison de deux environnements.

8) Améliorations effectuées

Réalisation d'un menu permettant de choisir le programme que l'on veut vérifier et sur quel domaine.

Ajout de quelques programmes de test.

Création de la JavaDoc.

Création de Makefile perfectionnés.

9) Améliorations possibles

Enrichissement du langage : types, opérateurs, structures de contrôles, ...

Lecture de programmes à partir d'un fichier.

Interface graphique.