# Travail d'Etude et de Recherche
# REALISATION OBJET D'UN MINI-SOLVEUR CSP
# ANNEXES

—

Sébastien RAMON
*Master 1 Mathématiques Informatique (Informatique)*
*Faculté des sciences Jean PERRIN de Lens*

Travail d'Etude et de Recherche encadré par
Stéphane CARDON et Christophe LECOUTRE
*Centre de Recherche en Informatique de Lens (CNRS)*

17 janvier 2007 - 11 mai 2007

# Table des matières


**A Coeur du programme** **5**
A.1 Classe Contrainte . . . 5
A.1.1 Contrainte.c++ . . . 5
A.1.2 Contrainte.h++ . . . 6
A.2 Classe Csp . . . 6
A.2.1 Csp.h++ . . . 6
A.2.2 Csp.c++ . . . 8
A.3 Classe Domaine . . . 9
A.3.1 Domaine.c++ . . . 9
A.3.2 Domaine.h++ . . . 12
A.4 Classe Extension . . . 13
A.4.1 Extension.c++ . . . 13
A.4.2 Extension.h++ . . . 16
A.5 Classe Historique . . . 16
A.5.1 Historique.c++ . . . 16
A.5.2 Historique.h++ . . . 17
A.6 Classe Ordre . . . 18
A.6.1 Ordre.c++ . . . 18
A.6.2 Ordre.h++ . . . 19
A.7 Classe Relation . . . 19
A.7.1 Relation.h++ . . . 19
A.8 Classe Variable . . . 20
A.8.1 Variable.c++ . . . 20
A.8.2 Variable.h++ . . . 21

**B Partie Solveur** **23**
B.1 Classe Ac3rm . . . 23
B.1.1 Ac3rm.c++ . . . 23
B.1.2 Ac3rm.h++ . . . 27
B.2 Classe Filtrage AC . . . 28
B.3 Classe Filtrage . . . 28

**C Outils** **31**
C.1 Classe Int . . . 31
C.1.1 Int.c++ . . . 31
C.1.2 Int.h++ . . . 31
C.2 Classe Queue . . . 31
C.2.1 Queue.c++ . . . 31
C.2.2 Queue.h++ . . . 33

**D Tests unitaires** **35**
D.1 Tests Ac3rm . . . 35
D.1.1 testAc3rm.c++ . . . 35
D.1.2 Résultat testAc3rm . . . 35
D.2 Tests Contrainte . . . 35

D.2.1 testContrainte.c++ . . . 35
D.2.2 Résultat testContrainte . . . 35
D.3 Tests Csp . . . 37
D.3.1 testCsp.c++ . . . 37
D.3.2 Résultat testCsp . . . 37
D.4 Tests Domaine . . . 39
D.4.1 testDomaine.c++ . . . 39
D.4.2 Résultat testDomaine . . . 39
D.5 Tests Extension . . . 41
D.5.1 testExtension.c++ . . . 41
D.5.2 Résultat testExtension . . . 41
D.6 Tests Queue . . . 42
D.6.1 testQueue.c++ . . . 42
D.6.2 Résultat testQueue . . . 42
D.7 Tests Relation . . . 43
D.7.1 testRelation.c++ . . . 43
D.7.2 Résultat testRelation . . . 43
D.8 Tests Variable . . . 44
D.8.1 testVariable.c++ . . . 44
D.8.2 Résultat testVariable . . . 44

**E Choix du langage** **47**
E.1 Test objet langage C . . . 47
E.1.1 Objet C . . . 47
E.1.2 Tri C . . . 48
E.2 Test objet langage C static . . . 49
E.2.1 Objet C static . . . 49
E.2.2 Tri C static . . . 50
E.3 test objet langage C objet . . . 51
E.3.1 Test objet langage C++ . . . 52

# Annexe A

# Coeur du programme

## A.1 Classe Contrainte

### A.1.1 Contrainte.c++

```
#include <assert.h>
#include "Contrainte.h++"

Contrainte::Contrainte (unsigned int i_v1, unsigned int i_v2, Relation* r) {
  ind_v1 = i_v1;
  ind_v2 = i_v2;
  rel = r;
}

Contrainte::~Contrainte () {
  delete rel;
}

unsigned int Contrainte::getV1 () {
return ind_v1;
}

unsigned int Contrainte::getV2 () {
return ind_v2;
}

Relation* Contrainte::getRelation () {
  return rel;
}

void Contrainte::affiche (ostream& out) {
  out << "liste des indices des variables de la contrainte: " << endl;
  out << "ind_v1: " << ind_v1 << " ,ind_v2: " << ind_v2 << endl;

  out << "affichage de la relation" << endl;
  rel->affiche (cout);

  return;
}

bool Contrainte::possedeSupport (unsigned int ind_var, unsigned int ind_val) {
  return rel->support (ind_var, ind_val);
}
```

```
int Contrainte::estDansVars (unsigned ind_var) {
  if (ind_var==ind_v1)
    return ind_v2;
  if (ind_var==ind_v2)
    return ind_v1;
  return -1;
}

unsigned long Contrainte::getMem () {
  return rel->getMem();
}
```

### A.1.2 Contrainte.h++

```
#ifndef __C_CONTRAINTE
#define __C_CONTRAINTE

#include <vector>
#include <iostream>
#include "Relation.h++"

using namespace std;

class Contrainte {
protected:
  unsigned int ind_v1, ind_v2;
  Relation* rel;

public:
  Contrainte (unsigned int i_v1, unsigned int i_v2, Relation* r);
  virtual ~Contrainte ();

  unsigned int getV1 ();
  unsigned int getV2 ();
  Relation* getRelation ();
  int estDansVars (unsigned ind_var);
  void affiche (ostream &out);
  bool possedeSupport (unsigned int ind_var, unsigned int ind_val);
  virtual unsigned long getMem ();
};
#endif
```

## A.2 Classe Csp

### A.2.1 Csp.h++

```
#include "Csp.h++"

Csp::Csp () {
  tpsConstruction = 0;
}

Csp::Csp (unsigned int v_size, unsigned int c_size) {
  lv.reserve (v_size * sizeof (Variable *));
  lc.reserve (c_size * sizeof (Contrainte *));
```

```
  tpsConstruction = 0;
}

Csp::Csp (Csp* csp) {
  lv.assign (csp->lv.begin (), csp->lv.end ());
  lc.assign (csp->lc.begin (), csp->lc.end ());
  tpsConstruction = csp->tpsConstruction;
}

Csp::~Csp () {
  unsigned int i;
  for (i=0;i<lv.size();i++)
    delete lv[i];
  for (i=0;i<lc.size();i++)
    delete lc[i];
}

void Csp::affiche (ostream &out) {
  unsigned int i;

  out << "Affichage Csp" << endl;
  out << "Affichages des variables" << endl;
  for (i=0; i<lv.size (); i++) {
    out << "variable " << i << endl;
    lv[i]->affiche (cout);
  }
  out << endl;

  out << "Affichages des contraintes" << endl;
  for (i=0; i<lc.size (); i++) {
    out << "contrainte " << i << endl;
    lc[i]->affiche (cout);
  }
  out << endl;

  out << "Temps de construction: " << tpsConstruction << " en cs " <<endl;

  return;
}

unsigned int Csp::sizeContrainte () {
return lc.size ();
}

unsigned int Csp::sizeVariable () {
return lv.size ();
}

void Csp::setNombreVariables (unsigned int v_size) {
  lv.reserve (v_size * sizeof (Variable *));
}

void Csp::setNombreContraintes (unsigned int c_size) {
  lc.reserve (c_size * sizeof (Contrainte *));
}
```

```
unsigned int Csp::addVariable (Variable* v) {
  lv.push_back (v);

  return (lv.size ()-1);
}

unsigned int Csp::addContrainte (Contrainte* c) {
  lc.push_back (c);

  return (lc.size ()-1);
}

Variable* Csp::getVariable (unsigned int indice) {
  return lv[indice];
}

Contrainte* Csp::getContrainte (unsigned int indice) {
  return lc[indice];
}

void Csp::setTempsConstruction (unsigned long tps) {
  tpsConstruction = tps;
}

unsigned long Csp::getTempsConstruction () {
  return tpsConstruction;
}

unsigned long Csp::getMem () {
  unsigned int i;
  unsigned long res=0;

  for (i=0;i<lv.size();i++)
    res+=lv[i]->getMem();
  res+=lv.size()*sizeof(Variable*)/NB_BITS_OCTET;
  for (i=0;i<lc.size();i++)
    res+=lc[i]->getMem();
  res+=lc.size()*sizeof(Contrainte*)/NB_BITS_OCTET;
  return res;
}
```

### A.2.2 Csp.c++

```
#ifndef __C_CSP
#define __C_CSP

#include <vector>
#include <iostream>
#include "Contrainte.h++"
#include "Variable.h++"

using namespace std;

class Csp {
protected:
```

```
  vector<Variable*> lv;
  vector<Contrainte*> lc;
  unsigned long tpsConstruction;

public:
  Csp ();
  Csp (unsigned int v_size, unsigned int c_size);
  Csp (Csp* csp);
  virtual ~Csp ();

  void setTempsConstruction (unsigned long tps);
  unsigned long getTempsConstruction ();
  void affiche (ostream &out);

  unsigned int sizeContrainte ();
  unsigned int sizeVariable ();
  void setNombreVariables (unsigned int v_size);
  void setNombreContraintes (unsigned int c_size);
  unsigned int addVariable (Variable* v);
  unsigned int addContrainte (Contrainte* c);
  Variable* getVariable (unsigned int indice);
  Contrainte* getContrainte (unsigned int indice);
  virtual unsigned long getMem ();
};
#endif
```

## A.3 Classe Domaine

### A.3.1 Domaine.c++

```
#include <assert.h>
#include <iostream>
#include "Domaine.h++"

Domaine::Domaine (int taille) : Historique () {
  valeursContenues.reserve (taille * sizeof (Int));
  nbValeurs=0;
}

Domaine::Domaine (Domaine* dom) : Historique () {
  valeursContenues.assign (dom->valeursContenues.begin (),
dom->valeursContenues.end ());
  nbValeurs=dom->nbValeurs;
}

unsigned int Domaine::getNombreValeurs () {
  return nbValeurs;
}

unsigned int Domaine::getNombreIndices () {
  return valeursContenues.size();
}

unsigned int Domaine::idValeur (int valeur) {
  Int temp;
  setV1(temp,valeur);
```

```
  for (unsigned int i=0;i<valeursContenues.size();i++)
    if (valeursContenues[i].v1 == temp.v1)
       return i;

  cerr<<"Erreur ! valeur "<<valeur<<" n'existe pas dans ce domaine !";
  return valeursContenues.size();
}

int Domaine::getValeur(unsigned int indice) {
  assert (indice < valeursContenues.size ());
  return getV1(valeursContenues[indice]);
}

void Domaine::affiche (ostream& out) {
  unsigned int i, temp;

  out<<"Valeurs contenues ("<<nbValeurs<<") :"<<endl;
  for (i=0; i < valeursContenues.size (); i++)
    if (valeursContenues[i].b1)
      out<<i<<": "<<getV1(valeursContenues[i])<<endl;

  out<<"Valeurs supprimees :"<<endl;
  for (i=0; i < valeursSupprimees.size (); i++) {
    out<<i<<":";
    if (!valeursSupprimees[i].b2) {
      temp=valeursSupprimees[i].v1;
      out<<temp<<": "<<getV1(valeursContenues[temp]);
      out<<" ";
      while (!valeursContenues[temp].b2) {
temp=valeursContenues[temp].v2;
out<<temp<<": "<<getV1(valeursContenues[temp]);
       out<<" ";
      }
      out<<endl;
    }
    else
      out<<" XXXX";
    out<<endl;
  }

  return;
}

bool Domaine::identique (Domaine* d) {
if (valeursContenues.size()!=d->valeursContenues.size())
return false;
for (unsigned int i=0;i<valeursContenues.size();i++)
if (valeursContenues[i].v1!=d->valeursContenues[i].v1)
return false;
return true;
}

bool Domaine::estVide () {
  return !nbValeurs;
}
```

```
bool Domaine::contient (unsigned int indice) {
  return valeursContenues[indice].b1;
}

void Domaine::ajoute (int valeur) {
  Int temp;
  setV1(temp,valeur);
  temp.b1=1;
  temp.v2=0;
  temp.b2=0;
  valeursContenues.push_back(temp);
  nbValeurs++;
}

void Domaine::supprime (unsigned int indice) {
  assert(indice < valeursContenues.size ());

  Int temp;
  valeursContenues[indice].b1=0;
  if (valeursSupprimees.size () == nbrEtapes) {
    temp.v1=indice;
    temp.b1=0;
    temp.v2=0;
    temp.b2=0;
    valeursSupprimees.push_back (temp);
    valeursContenues[indice].b2=1;
  }
  else {
    valeursContenues[indice].v2=valeursSupprimees[nbrEtapes].v1;
    valeursContenues[indice].b2=0;
    valeursSupprimees[nbrEtapes].v1=indice;
  }
  nbValeurs--;
}

void Domaine::commit () {
  Historique::commit ();

  if (valeursSupprimees.size () < nbrEtapes) {
    Int temp;
    temp.v1=0;
    temp.b1=0;
    temp.v2=0;
    temp.b2=1;

    valeursSupprimees.push_back (temp);
  }

  return;
}

void Domaine::rollback () {
  assert (nbrEtapes > 0);

  nbrEtapes--;
```

```
  if (valeursSupprimees[nbrEtapes].b2) {
    valeursSupprimees.pop_back ();
    return;
  }
  unsigned int ind=valeursSupprimees[nbrEtapes].v1;

  valeursContenues[ind].b1=1;
  nbValeurs++;
  while (!valeursContenues[ind].b2) {
    ind=valeursContenues[ind].v2;
    valeursContenues[ind].b1=1;
    nbValeurs++;
  }
  valeursSupprimees.pop_back ();
}

unsigned long Domaine::getMem () {
  unsigned long res=Historique::getMem();
  res+=valeursContenues.size()*sizeof(Int)/NB_BITS_OCTET;
  return res;
}
```

### A.3.2 Domaine.h++

```
#ifndef __C_DOMAINE
#define __C_DOMAINE

#include <vector>
#include <iostream>
#include "Historique.h++"
#include "Int.h++"

using namespace std;

class Domaine : public Historique {
protected:
  vector<Int> valeursContenues;
  unsigned int nbValeurs;

public:
  Domaine () {};
  Domaine (int taille);
  Domaine (Domaine* dom);
  virtual ~Domaine () {};

  void affiche (ostream& out);
  bool estVide ();
  bool contient (unsigned int indice);
  unsigned int getNombreIndices ();
  unsigned int getNombreValeurs ();

  /**
   * Retourne l'indice de la valeur reelle 'valeur'
   * contenue dans ce domaine.
   */
  unsigned int idValeur (int valeur);
```

```
  bool identique (Domaine* d);
  int getValeur (unsigned int indice);

  void ajoute (int valeur);
  void supprime (unsigned int indice);
  virtual void commit ();
  virtual void rollback ();
  virtual unsigned long getMem ();
};
#endif
```

## A.4 Classe Extension

### A.4.1 Extension.c++

```
#include "Extension.h++"

Extension::Extension (Extension* ext) : Relation (),BitMatrice(ext) {
  autorisees=ext->autorisees;
  nbTuples=ext->nbTuples;
  if (!ext->valeursSupprimees.empty())
    valeursSupprimees.assign (ext->valeursSupprimees.begin (),
ext->valeursSupprimees.end ());
}

Extension::Extension (bool autorisation,unsigned int col,unsigned int lig)
: Relation (),BitMatrice(autorisation,col,lig) {
  autorisees=autorisation;
  setDimensions(col,lig);
}

Extension::Extension (bool autorisation) {
  autorisees=autorisation;
}

Extension::~Extension () {
}

void Extension::setDimensions (unsigned int col,unsigned int lig) {
  if (autorisees)
    nbTuples=0;
  else
    nbTuples=col*lig;
  BitMatrice::setDimensions(autorisees,col,lig);
}

void Extension::affiche (ostream &out) {
  unsigned int nC, nL;
  bool versG, versD;

  out<<"Tuples actuels ["<<nCol<<"x"<<nLig<<"] ("<<nbTuples<<") :"<<endl;
  for (nC=0;nC<nCol;nC++)
    for (nL=0;nL<nLig;nL++) {
      versG=((nL<nCol) && (nC<nLig) && (get(nL,nC)));
      versD=get(nC,nL);
      if ((versG || versD)
```

```
  && ((!versG) || (!versD) || (nC<=nL))) {
out<<nC;
if (versG)
  out<<"<";
else
  out<<" ";
out<<"-";
if (versD)
  out<<">";
else
  out<<" ";
out<<nL<<endl;
      }
    }

  out << "Tuples supprimes" << endl;
  for (unsigned int i=0; i < valeursSupprimees.size (); i++) {
   if (valeursSupprimees[i].b1) {
      out << valeursSupprimees[i].v1;
      out<<" ->";
      out << valeursSupprimees[i].v2;
      out << endl;
   } else {
   out << "Etape vide " << endl;
   }
  }
  out << endl;
}

bool Extension::estVide () {
  return (!nbTuples);
}

bool Extension::appartient (unsigned ind_v1, unsigned ind_v2) {
  return get(ind_v1,ind_v2);
}

bool Extension::support (unsigned int ind_var, unsigned int ind_val) {
  if (ind_var) {
    for (unsigned int nC=0;nC<nCol;nC++)
      if (get(nC,ind_val))
return true;
    return false;
  }

  for (unsigned int nL=0;nL<nLig;nL++)
    if (get(ind_val,nL))
return true;
  return false;
}

void Extension::ajouteBrut (int v1,int v2) {
  Int temp;
  temp.b1=0;
  temp.b2=0;
  setV1(temp,v1);
```

```
  setV2(temp,v2);
  valeursSupprimees.push_back(temp);
}

void Extension::setDomaines (Domaine* d1,Domaine* d2) {
  setDimensions(d1->getNombreIndices(),d2->getNombreIndices());
  Int temp;
  while (!valeursSupprimees.empty()) {
    temp=valeursSupprimees.back();
    valeursSupprimees.pop_back();
    ajoute(d1->idValeur(getV1(temp)),d2->idValeur(getV2(temp)));
  }
}

void Extension::ajoute (unsigned int ind_v1, unsigned int ind_v2) {
  set(ind_v1,ind_v2,autorisees);
  if (autorisees)
    nbTuples++;
  else
    nbTuples--;
}

void Extension::supprime (unsigned int ind_v1, unsigned int ind_v2) {
  set(ind_v1,ind_v2,false);
  nbTuples--;

  Int temp;
  temp.v1=ind_v1;
  temp.b1=1;
  temp.v2=ind_v2;
  temp.b2=0;
  valeursSupprimees.push_back(temp);
}

void Extension::commit () {
Int temp;
temp.v1=0;
temp.b1=0;
temp.v2=0;
temp.b2=0;
  valeursSupprimees.push_back(temp);
}

void Extension::rollback () {
  Int temp;

temp=valeursSupprimees.back();
valeursSupprimees.pop_back();
  while (!valeursSupprimees.empty()) {
    temp=valeursSupprimees.back();

    if (!temp.b1)
      break;
    set(temp.v1,temp.v2,true);
    nbTuples++;
    valeursSupprimees.pop_back();
```

```
  }
}

unsigned long Extension::getMem () {
  return Historique::getMem()+BitMatrice::getMemMatrice();
}
```

### A.4.2 Extension.h++

```
#ifndef __C_EXTENSION_M
#define __C_EXTENSION_M

#include <vector>
#include <iostream>
#include "Relation.h++"
#include "Domaine.h++"
#include "../tools/BitMatrice.h++"

using namespace std;

class Extension : public Relation,public BitMatrice {
private:
  void setDimensions (unsigned int col,unsigned int lig);

protected:
  bool autorisees;
  unsigned long nbTuples;

public:
  Extension (bool autorisation);
  Extension (bool autorisation,unsigned int col,unsigned int lig);
  Extension (Extension* ext);
  virtual ~Extension ();

  virtual void affiche (ostream &out);
  virtual bool estVide ();
  virtual bool appartient (unsigned ind_v1, unsigned ind_v2);
  virtual bool support (unsigned int ind_var, unsigned int ind_val);

  void ajouteBrut (int v1,int v2);
  void setDomaines (Domaine* d1,Domaine* d2);

  virtual void ajoute (unsigned int ind_v1, unsigned int ind_v2);
  virtual void supprime (unsigned int ind_v1, unsigned int ind_v2);
  virtual void commit ();
  virtual void rollback ();
  virtual unsigned long getMem ();
};
#endif
```

## A.5 Classe Historique

### A.5.1 Historique.c++

```
#include "Historique.h++"
```

```
Historique::Historique () {
  nbrEtapes = 0;
}

void Historique::commit () {
  nbrEtapes++;
}

unsigned long Historique::getMem () {
  return valeursSupprimees.size()*sizeof(Int)/NB_BITS_OCTET;
}
```

### A.5.2 Historique.h++

```
#ifndef __C_HISTORIQUE
#define __C_HISTORIQUE

#include <vector>
#include "Int.h++"

using namespace std;

class Historique {
protected:
  unsigned int nbrEtapes;
  vector<Int> valeursSupprimees;

public:
  Historique ();
  virtual ~Historique () {};

  virtual void commit ();
  virtual void rollback ()=0;
  virtual unsigned long getMem ();
};
#endif


#ifndef __C_INT
#define __C_INT

struct Int {
  unsigned int v1 : NB_BITS_VAL;
  unsigned int b1 : 1;
  unsigned int v2 : NB_BITS_VAL;
  unsigned int b2 : 1;
};

void setV1 (Int& i,int v);
void setV2 (Int& i,int v);
int getV1 (Int& i);
int getV2 (Int& i);

#endif
```

## A.6 Classe Ordre

### A.6.1 Ordre.c++

```
#include "Ordre.h++"

Ordre::Ordre (unsigned int taille) {
  ordre.reserve ((taille+(taille%2)) * (sizeof (Int)/2));

  Int temp;
  for (ind_next=0; ind_next<taille; ind_next++) {
   if (ind_next%2) {
   ordre[ind_next/2].v2=ind_next;
   } else {
   temp.v1=ind_next;
   temp.b1=0;
   temp.v2=0;
   temp.b2=0;
   ordre.push_back (temp);
   }
  }

  if (ind_next%2) {
   ordre[ind_next/2].b2=1;
  }

  ind_next = 0;
}

unsigned int Ordre::next () {
return (ind_next%2 ? ordre[ind_next++/2].v2 : ordre[ind_next++/2].v1);
}

unsigned int Ordre::backtrack () {
ind_next--;
return (ind_next%2 ? ordre[ind_next/2].v2 : ordre[ind_next/2].v1);
}

void Ordre::ordonne (const vector<unsigned int>& newOrdre) {
unsigned int i;

for (i=0; i<newOrdre.size (); i++) {
    if ((i + ind_next) % 2) {
     ordre[(i+ind_next)/2].v2=newOrdre[i];
    } else {
     ordre[(i+ind_next)/2].v1=newOrdre[i];
    }
  }

return;
}

unsigned long Ordre::getMem () {
  return ordre.size()*sizeof(Int)/NB_BITS_OCTET;
}
```

### A.6.2 Ordre.h++

```
#ifndef __C_ORDRE
#define __C_ORDRE

#include <iostream>
#include <vector>
#include "Int.h++"

using namespace std;

class Ordre {
protected:
  vector<Int> ordre;
  unsigned int ind_next;

public:
  Ordre (unsigned int taille);
  virtual ~Ordre () {};

  unsigned int next ();
  unsigned int backtrack ();
  void ordonne (const vector<unsigned int>& newOrdre);
  virtual unsigned long getMem ();
};
#endif
```

## A.7 Classe Relation

### A.7.1 Relation.h++

```
#ifndef __C_RELATION
#define __C_RELATION

#include "Historique.h++"
using namespace std;

class Relation : public Historique {
protected:
  Relation() : Historique () {};

public:
  virtual ~Relation() {};

  virtual void affiche (ostream &out)=0;
  virtual bool estVide ()=0;
  virtual bool appartient (unsigned int ind_v1, unsigned int ind_v2)=0;
  virtual bool support (unsigned int ind_var, unsigned int ind_val)=0;

  virtual void ajoute (unsigned int ind_v1, unsigned int ind_v2)=0;
  virtual void supprime (unsigned int ind_v1, unsigned int ind_v2)=0;
  virtual unsigned long getMem ()=0;
};
#endif
```

## A.8 Classe Variable

### A.8.1 Variable.c++

```
#include "Variable.h++"

Variable::Variable (Domaine* d) : Ordre(d->getNombreIndices ()) {
  this->d = d;
}

Variable::~Variable () {
  delete d;
}

Domaine* Variable::getDomaine () {
  return d;
}

void Variable::affiche (ostream &out) {
unsigned int i;

  out << "Variable, domaine :" << endl;
  d->affiche (out);

  out << "        , ordre affectations :" << endl;
  out << "indice prochain: " << ind_next << endl;
for (i=0; i<ordre.size (); i++) {
cout <<" -> "<< ordre[i].v1;
if (!ordre[i].b2) {
cout <<" -> "<< ordre[i].v2;
}
}
out << endl;
}

bool Variable::estAffectee () {
return (ind_next > 0);
}

unsigned int Variable::affectation () {
unsigned int ind = next ();
while (!(ind) && (!d->contient (ind))) {
ind = next ();
}

return ind;
}

unsigned long Variable::getMem () {
  unsigned long res=Ordre::getMem();
  res+=d->getMem();

  return res;
}
```

### A.8.2 Variable.h++

```
#ifndef __C_VARIABLE
#define __C_VARIABLE

#include <iostream>
#include "Domaine.h++"
#include "Ordre.h++"

using namespace std;

class Variable : public Ordre {
protected:
  Domaine* d;

public:
  Variable (Domaine* d);
  virtual ~Variable ();

  void affiche (ostream &out);
  Domaine* getDomaine ();
  bool estAffectee ();
  unsigned int affectation ();
  unsigned int affectInd ();
  virtual unsigned long getMem ();
};
#endif
```

# Annexe B

# Partie Solveur

## B.1 Classe Ac3rm

### B.1.1 Ac3rm.c++

```
#include "Ac3rm.h++"
#include "../tools/Queue.h++"

#ifdef __STATS
#include <time.h>
#endif

Ac3rm::Ac3rm (Csp* c) : Filtrage_ac () {
  this->csp = c;
  unsigned int max, temp;
  Int valeur;
  valeur.b1=0;
  valeur.b2=0;
  supp.reserve(csp->sizeContrainte()*sizeof(vector<Int>));
  for (unsigned int i=0;i<csp->sizeContrainte();i++) {
    max=csp->getVariable(csp->getContrainte(i)->getV1())
->getDomaine()->getNombreIndices();
    temp=csp->getVariable(csp->getContrainte(i)->getV2())
->getDomaine()->getNombreIndices();
    if (temp>max)
      max=temp;
    supp.push_back(vector<Int>());
    supp[i].reserve(max*sizeof(Int));
    for (temp=0;temp<max;temp++)
      supp[i].push_back(valeur);
  }
}

bool Ac3rm::filtrer () {
#ifdef __STATS
  struct timespec deb, fin;
  long res;

  clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&deb);
#endif
  unsigned int indC, indCp, indX;
  int indV;
  Queue q(csp->sizeContrainte (), csp->sizeVariable ());
```

```
  indC = 0;
  while (indC < csp->sizeContrainte ()) {
    q.push (indC, (csp->getContrainte (indC))->getV1 ());
    q.push (indC, (csp->getContrainte (indC))->getV2 ());
    indC++;
  }

  while (!q.empty ()) {
    q.pop (indC,indX);
#ifdef __DEBUG_AC3
    cout<<"Travail avec "<<indC<<","<<indX<<endl;
#endif

    if (revise (indC, indX)) {
#ifdef __STATS
  nbRevisions++;
#endif
      if (((csp->getVariable (indX))->getDomaine ())->estVide ()) {
#ifdef __STATS
clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&fin);

res=fin.tv_sec*100-deb.tv_sec*100;
if (fin.tv_nsec<deb.tv_nsec) {
  res-=100;
  res+=(1000000000-deb.tv_nsec+fin.tv_nsec)/10000000;
}
else
  res+=(fin.tv_nsec-deb.tv_nsec)/10000000;
tempsFiltrage.push_back((unsigned long)res);
#endif
return false;
      }

      indCp = 0;
      while (indCp < csp->sizeContrainte ()) {
  indV=csp->getContrainte(indCp)->estDansVars(indX);

  if ((indV!=-1) &&
      (!q.contains(indCp,(unsigned)indV))
      && (indCp!=indC)) {
#ifdef __DEBUG_AC3
    cout<<"Ajout de "<<indCp<<","<<indV<<endl;
#endif
    q.push (indCp,(unsigned)indV);
  }
indCp++;
      }

    }
#ifdef __STATS
    else {
      nbRevisionsInutiles++;
    }
#endif
  }
```

```
#ifdef __STATS
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&fin);

  res=fin.tv_sec*100-deb.tv_sec*100;
  if (fin.tv_nsec<deb.tv_nsec) {
    res-=100;
    res+=(1000000000-deb.tv_nsec+fin.tv_nsec)/10000000;
  }
  else
    res+=(fin.tv_nsec-deb.tv_nsec)/10000000;
  tempsFiltrage.push_back((unsigned long)res);
#endif
  return true;
}

bool Ac3rm::revise (unsigned int indC, unsigned int indV) {
  unsigned int nbElements;
  Domaine * domV,* domAutre;
  int t=-1;
  bool first=(indV == csp->getContrainte(indC)->getV1());

#ifdef __DEBUG_AC3
  unsigned int indAutre=csp->getContrainte(indC)->estDansVars(indV);
#endif
  if (first)
      domAutre=(csp->getVariable (csp->getContrainte(indC)->getV2()))
->getDomaine ();
  else
      domAutre=(csp->getVariable (csp->getContrainte(indC)->getV1()))
->getDomaine ();
  domV = (csp->getVariable (indV))->getDomaine ();
  nbElements = domV->getNombreValeurs ();

  for (unsigned int i=0; i<domV->getNombreIndices(); i++)
    if (domV->contient(i)) {
      if (first) {
       if ((supp[indC][i].b1) && (domAutre->contient(supp[indC][i].v1))) {
#ifdef __DEBUG_AC3
  cout<<"Dernier support trouve pour "<<indC<<","<<indV<<","<<i<<" = "
<<supp[indC][i].v1<<endl;
#endif
  continue;
       }
      } else {
       if ((supp[indC][i].b2) && (domAutre->contient(supp[indC][i].v2))) {
#ifdef __DEBUG_AC3
  cout<<"Dernier support trouve pour "<<indC<<","<<indV<<","<<i<<" = "
<<supp[indC][i].v2<<endl;
#endif
  continue;
       }
      }

      t = seekSupport (indC, indV, i, domAutre, first,(first?(supp[indC][i]
.b1?supp[indC][i].v1+1:0):(supp[indC][i].b2?supp[indC][i].v2+1:0)));
```

```
      if (t == -1 ) {
#ifdef __STATS
nbSuppressions++;
#endif
#ifdef __DEBUG_AC3
  cout<<"Suppression dans "<<indC<<","<<indV<<", de la valeur "<<i<<endl;
#endif
       domV->supprime(i);
      } else {
       if (first) {
  supp[indC][i].v1=(unsigned)t;
  supp[indC][i].b1=1;
#ifdef __DEBUG_AC3
  cout<<"MARK : dernier support trouve pour "<<indC<<","<<indV<<","<<i
<<" = "<<supp[indC][i].v1<<endl;
#endif
  if (!supp[indC][(unsigned)t].b2) {
    supp[indC][(unsigned)t].v2=i;
    supp[indC][(unsigned)t].b2=1;
#ifdef __DEBUG_AC3
    cout<<"MARK : dernier support trouve pour "<<indC<<","<<indAutre<<","<<t
<<" = "<<supp[indC][(unsigned)t].v2<<endl;
#endif
  }
       } else {
  supp[indC][i].v2=(unsigned)t;
  supp[indC][i].b2=1;
#ifdef __DEBUG_AC3
  cout<<"MARK : dernier support trouve pour "<<indC<<","<<indV<<","<<i<<" = "
<<supp[indC][i].v2<<endl;
#endif
  if (!supp[indC][(unsigned)t].b1) {
    supp[indC][(unsigned)t].v1=i;
    supp[indC][(unsigned)t].b1=1;
#ifdef __DEBUG_AC3
    cout<<"MARK : dernier support trouve pour "<<indC<<","<<indAutre<<","<<t
<<" = "<<supp[indC][(unsigned)t].v1<<endl;
#endif
  }
       }
      }
    }

  return (nbElements != domV->getNombreValeurs ());
}

int Ac3rm::seekSupport (unsigned int indC, unsigned int indV, unsigned int indA,
Domaine* dom,bool first,unsigned int depart) {
  Relation* rel=csp->getContrainte(indC)->getRelation();

  for (unsigned int i=depart;i<dom->getNombreIndices();i++)
    if (dom->contient(i)) {
      if (first) {
#ifdef __DEBUG_AC3
if (!supp[indC][i].b2)
  cout<<"Pas de dernier support inverse trouve pour "<<indC<<","<<indV<<","
```

```
<<i<<endl;
else
  cout<<"Dernier support inverse trouve pour "<<indC<<","<<indV<<","<<i<<" = "
<<supp[indC][i].v2<<endl;
#endif
if ((supp[indC][i].b2) && (supp[indC][i].v2>indA))
  continue;
      }
      else {
#ifdef __DEBUG_AC3
if (!supp[indC][i].b1)
  cout<<"Pas de dernier support inverse trouve pour "<<indC<<","<<indV<<","
<<i<<endl;
else
  cout<<"Dernier support inverse trouve pour "<<indC<<","<<indV<<","<<i<<" = "
<<supp[indC][i].v1<<endl;
#endif
if ((supp[indC][i].b1) && (supp[indC][i].v1>indA))
  continue;
      }
#ifdef __STATS
nbTestAppartenanceContrainte++;
#endif
      if (first) {
if (rel->appartient(indA,i)) {
#ifdef __STATS
  ccks++;
#endif
  return i;
}
      }
      else {
if (rel->appartient(i,indA)) {
#ifdef __STATS
  ccks++;
#endif
  return i;
}
      }
    }

  return -1;
}
```

### B.1.2 Ac3rm.h++

```
#ifndef __C_AC3RM
#define __C_AC3RM

using namespace std;

#include "Filtrage_ac.h++"
#include "../core/Csp.h++"

class Ac3rm : public Filtrage_ac {
protected:
```

```
  Csp *csp;
  vector<vector<Int> > supp;

public:
  Ac3rm (Csp* c);
  virtual ~Ac3rm () {};

  virtual bool filtrer ();
  virtual bool revise (unsigned int indC, unsigned int indV);
  virtual int seekSupport (unsigned int indC, unsigned int indV,
unsigned int indA,Domaine* dom,bool first,unsigned int depart);
};
#endif
```

## B.2 Classe Filtrage AC

```
#ifndef __C_FILTRAGE_AC
#define __C_FILTRAGE_AC

#include "Filtrage.h++"
#include "../core/Contrainte.h++"
#include "../core/Variable.h++"
#include "../core/Tuple.h++"
#include "../core/Int.h++"

using namespace std;

class Filtrage_ac : public Filtrage {
protected:
  Filtrage_ac () : Filtrage () {};

public:
  virtual ~Filtrage_ac () {};

  virtual bool revise (unsigned int indC, unsigned int indV)=0;
  virtual int seekSupport (unsigned int indC, unsigned int indV,
unsigned int indA,Domaine* dom,bool first,unsigned int depart)=0;
};
#endif
```

## B.3 Classe Filtrage

```
#ifndef __C_FILTRAGE
#define __C_FILTRAGE

#include "../core/Csp.h++"

#ifdef __STATS
#include <vector>
#endif

using namespace std;

class Filtrage {
protected:
```

```
#ifdef __STATS
  Filtrage () {
    nbRevisions=0;
    nbRevisionsInutiles=0;
    nbSuppressions=0;
    nbTestAppartenanceContrainte=0;
    ccks=0;
  };
#else
  Filtrage () {};
#endif

public:
#ifdef __STATS
  unsigned long nbRevisions;
  unsigned long nbRevisionsInutiles;
  unsigned long nbSuppressions;
  unsigned long nbTestAppartenanceContrainte;
  unsigned long ccks;

  vector<unsigned long> tempsFiltrage;
#endif
  virtual ~Filtrage () {};

  virtual bool filtrer ()=0;
};
#endif
```

# Annexe C

# Outils

## C.1 Classe Int

### C.1.1 Int.c++

```
#include "Int.h++"

void setV1 (Int& i,int v) {
  if (v<0)
    v=-v+MASQUE_SIGNE;
  i.v1=(unsigned)v;
}

void setV2 (Int& i,int v) {
  if (v<0)
    v=-v+MASQUE_SIGNE;
  i.v2=(unsigned)v;
}

int getV1 (Int& i) {
  if (i.v1&MASQUE_SIGNE)
    return -(int)(i.v1-MASQUE_SIGNE);
  return (int)i.v1;
}

int getV2 (Int& i) {
  if (i.v2&MASQUE_SIGNE)
    return -(int)(i.v2-MASQUE_SIGNE);
  return (int)i.v2;
}
```

### C.1.2 Int.h++

## C.2 Classe Queue

### C.2.1 Queue.c++

```
#include <cassert>
#include "Queue.h++"

Queue::Queue (unsigned int e, unsigned int n) {
  assert(((unsigned)e*2*n)<((unsigned)MAX_QUEUE));
  q.reserve ((e*2*n) * (sizeof (int)));
```

```
  for (unsigned int i=0;i<e*2*n;i++)
    q[i]=-1;

  nbVar = n;
  debut = -1;
  fin = -1;
}

bool Queue::empty () {
  return ((debut == -1) && (fin == -1));
}

void Queue::affiche (ostream& out) {
  int ind;
  out << "Affichage queue: " << endl;

  if ((debut == -1) && (fin == -1)) {
    out << "vide";
  } else {
    ind = debut;

    while (ind != q[ind]) {
      out << ind << " < ";
      ind = q[ind];
    }

    out << q[ind] << endl;
  }

  return;
}

unsigned int Queue::bijection (unsigned int c, unsigned int v) {
  return ((c*nbVar)+v);
}

void Queue::bijectionInv (unsigned int bij, unsigned int& c, unsigned int& v) {
  c = (bij/(nbVar));
  v = (bij%(nbVar));

  return;
}

void Queue::push (unsigned int ind_c, unsigned int ind_v) {
  unsigned int bij;
  bij = bijection (ind_c, ind_v);

  q[bij] = bij;

  if ((debut == -1) && (fin == -1)) {
    debut = bij;
    fin = bij;
  } else {
    q[fin] = bij;
    fin = bij;
```

```
  }

  return;
}

void Queue::pop (unsigned int& ind_c, unsigned int& ind_v) {
  bool vide = false;
  int tmp;

  if (debut != -1) {
    if (debut == fin ) {
      vide = true;
    }

    bijectionInv (debut, ind_c, ind_v);
    tmp=q[debut];
    q[debut]=-1;
    debut=tmp;

    if (vide) {
      debut = -1;
      fin = -1;
    }
  }

  return;
}

bool Queue::contains (unsigned int ind_c, unsigned int ind_v) {
  return (q[bijection (ind_c, ind_v)]>=0);
}
```

### C.2.2 Queue.h++

```
#ifndef __C_QUEUE
#define __C_QUEUE

#include <iostream>
#include <vector>

using namespace std;

class Queue {
protected:
  vector<int> q;
  unsigned int nbVar;
  int debut, fin;

public:
  Queue (unsigned int e, unsigned int n);
  virtual ~Queue () {};

  void affiche (ostream& out);
  unsigned int bijection (unsigned int c, unsigned int v);
  void bijectionInv (unsigned int bij, unsigned int& c, unsigned int& v);
```

```
  bool empty ();
  void push (unsigned int ind_c, unsigned int ind_v);
  void pop (unsigned int& ind_c, unsigned int& ind_v);
  bool contains (unsigned int ind_c, unsigned int ind_v);
};
#endif
```

# Annexe D

# Tests unitaires

## D.1 Tests Ac3rm

### D.1.1 testAc3rm.c++

### D.1.2 Résultat testAc3rm

```
Parse instance ...
OK, in 0 cs
Using memory in 0 : 38
Nombre de revisions : 4
Nombre de revisions inutiles : 13
Nombre de valeurs supprimees : 7
Nombre de tests d'appartenance a une contrainte : 30
ccks : 16
Filtrage 0 : 0 cs
```

## D.2 Tests Contrainte

### D.2.1 testContrainte.c++

### D.2.2 Résultat testContrainte

```
%%%% affichage contrainte %%%
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes

%%%% etape 1 %%%%
on ne fait rien
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide
```

```
appartient (0,1)?
oui

possede support (0,0)?
oui

%%%% etape 2 %%%%
suppression de (0,1)
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (3) :
0<->0
0<- 1
1 ->0
1<->1
Tuples supprimes
Etape vide
0 ->1

suppression de (0,0)
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (2) :
0<- 1
1 ->0
1<->1
Tuples supprimes
Etape vide
0 ->1
0 ->0

suppression de (1,0)
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (1) :
1<->1
Tuples supprimes
Etape vide
0 ->1
0 ->0
1 ->0

appartient (0,1)?
non

possede support (0,0)?
non

%%%% etape 3 %%%%
on ne fait rien
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
```

```
affichage de la relation
Tuples actuels [2x2] (1) :
1<->1
Tuples supprimes
Etape vide
0 ->1
0 ->0
1 ->0
Etape vide
Etape vide

%%%% roolback 2 %%%%
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (1) :
1<->1
Tuples supprimes
Etape vide
0 ->1
0 ->0
1 ->0
Etape vide

%%%% roolback 1 %%%%
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide

%%%% roolback 0 %%%%
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
```

## D.3 Tests Csp

### D.3.1 testCsp.c++

### D.3.2 Résultat testCsp

```
ajout variable X, indice: 0
ajout variable Y, indice: 1
ajout variable Z, indice: 2
ajout contrainte c1, indice: 0
```

```
ajout contrainte c2, indice: 1
ajout contrainte c3, indice: 2

Affichage Csp
Affichages des variables
variable 0
Variable, domaine :
Valeurs contenues (2) :
0: 0
1: 1
Valeurs supprimees :
        , ordre affectations :
indice prochain: 0
 -> 0 -> 1
variable 1
Variable, domaine :
Valeurs contenues (2) :
0: 0
1: 1
Valeurs supprimees :
        , ordre affectations :
indice prochain: 0
 -> 0 -> 1
variable 2
Variable, domaine :
Valeurs contenues (2) :
0: 0
1: 1
Valeurs supprimees :
        , ordre affectations :
indice prochain: 0
 -> 0 -> 1

Affichages des contraintes
contrainte 0
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 1
affichage de la relation
Tuples actuels [2x2] (2) :
0<->1
Tuples supprimes

contrainte 1
liste des indices des variables de la contrainte:
ind_v1: 1 ,ind_v2: 2
affichage de la relation
Tuples actuels [2x2] (2) :
0<->1
Tuples supprimes

contrainte 2
liste des indices des variables de la contrainte:
ind_v1: 0 ,ind_v2: 2
affichage de la relation
Tuples actuels [2x2] (2) :
0<->1
```

```
Tuples supprimes


Temps de construction: 0 en cs
```

## D.4 Tests Domaine

### D.4.1 testDomaine.c++

### D.4.2 Résultat testDomaine

```
%%%% etape 1 %%%%
on ne fait rien
Valeurs contenues (4) :
0: -1
1: 4
2: 7
3: 9
Valeurs supprimees :
0: XXXX
existe indice(0)?
oui

%%%% etape 2 %%%%%
supression indice 0
Valeurs contenues (3) :
1: 4
2: 7
3: 9
Valeurs supprimees :
0: XXXX
1:0: -1

supression indice 2
Valeurs contenues (2) :
1: 4
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

existe indice(0)?
non

%%%% etape 3 %%%%
on ne fait rien
Valeurs contenues (2) :
1: 4
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

2: XXXX
%%%% etape 4 %%%%
supression indice 1
```

```
Valeurs contenues (1) :
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

2: XXXX
3:1: 4

domaine non vide
%%%% etape 5 %%%%
supression indice 3
Valeurs contenues (0) :
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

2: XXXX
3:1: 4

4:3: 9

domaine vide
%%%% rollback etape 4 %%%%
Valeurs contenues (1) :
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

2: XXXX
3:1: 4

%%%% rollback etape 3 %%%%
Valeurs contenues (2) :
1: 4
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

2: XXXX
%%%% rollback etape 2 %%%%
Valeurs contenues (2) :
1: 4
3: 9
Valeurs supprimees :
0: XXXX
1:2: 7 0: -1

%%%% rollback etape 1 %%%%
Valeurs contenues (4) :
0: -1
1: 4
2: 7
3: 9
```

```
Valeurs supprimees :
0: XXXX
%%%% rollback etape 0 %%%%
Valeurs contenues (4) :
0: -1
1: 4
2: 7
3: 9
Valeurs supprimees :
Indice de -1 = 0
Valeur a 0 = -1
```

## D.5 Tests Extension

### D.5.1 testExtension.c++

### D.5.2 Résultat testExtension

```
%%%% etape 1 %%%%
on ne fait rien
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide

appartient (0,1)?
oui

%%%% etape 2 %%%%
suppression de (0,1)
Tuples actuels [2x2] (3) :
0<->0
0<- 1
1 ->0
1<->1
Tuples supprimes
Etape vide
0 ->1

suppression de (1,0)
Tuples actuels [2x2] (2) :
0<->0
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0

appartient (0,1)?
non

%%%% etape 3 %%%%
on ne fait rien
Tuples actuels [2x2] (2) :
```

```
0<->0
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0
Etape vide
Etape vide

%%%% roolback 2 %%%%
Tuples actuels [2x2] (2) :
0<->0
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0
Etape vide

%%%% roolback 1 %%%%
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide

%%%% roolback 0 %%%%
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
```

## D.6 Tests Queue

### D.6.1 testQueue.c++

### D.6.2 Résultat testQueue

```
Affichage queue:
vide

contient (1,1): non

ajout (1,1)
Affichage queue:
4
ajout (2,1)
Affichage queue:
4 < 7
ajout (1,0)
Affichage queue:
4 < 7 < 3
```

```
contient (1,1): oui

retire tete (1,1)
Affichage queue:
7 < 3

retire tete (2,1)
Affichage queue:
3

retire tete (1,0)
Affichage queue:
vide
contient (1,1): non

ajout (1,0)
Affichage queue:
3

ajout (2,1)
Affichage queue:
3 < 7
```

## D.7 Tests Relation

### D.7.1 testRelation.c++

### D.7.2 Résultat testRelation

```
%%%% etape 1 %%%%
on ne fait rien
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide

appartient (0,1)?
oui

%%%% etape 2 %%%%
suppression de (0,1)
Tuples actuels [2x2] (3) :
0<->0
0<- 1
1 ->0
1<->1
Tuples supprimes
Etape vide
0 ->1

suppression de (1,0)
Tuples actuels [2x2] (2) :
0<->0
```

```
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0

appartient (0,1)?
non

%%%% etape 3 %%%%
on ne fait rien
Tuples actuels [2x2] (2) :
0<->0
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0
Etape vide
Etape vide

%%%% roolback 2 %%%%
Tuples actuels [2x2] (2) :
0<->0
1<->1
Tuples supprimes
Etape vide
0 ->1
1 ->0
Etape vide

%%%% roolback 1 %%%%
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
Etape vide

%%%% roolback 0 %%%%
Tuples actuels [2x2] (4) :
0<->0
0<->1
1<->1
Tuples supprimes
```

## D.8 Tests Variable

### D.8.1 testVariable.c++

### D.8.2 Résultat testVariable

```
Variable, domaine :
Valeurs contenues (5) :
0: 1
```

```
1: 2
2: 3
3: 4
4: 5
Valeurs supprimees :
        , ordre affectations :
indice prochain: 0
 -> 0 -> 1 -> 2 -> 3 -> 4

variable non affectee
ordonne 4,3,2,1,0
Variable, domaine :
Valeurs contenues (5) :
0: 1
1: 2
2: 3
3: 4
4: 5
Valeurs supprimees :
        , ordre affectations :
indice prochain: 0
 -> 4 -> 3 -> 2 -> 1 -> 0
affectation: 4
Variable, domaine :
Valeurs contenues (5) :
0: 1
1: 2
2: 3
3: 4
4: 5
Valeurs supprimees :
        , ordre affectations :
indice prochain: 1
 -> 4 -> 3 -> 2 -> 1 -> 0

variable affectee
reordonne 0,1,2,3
Variable, domaine :
Valeurs contenues (5) :
0: 1
1: 2
2: 3
3: 4
4: 5
Valeurs supprimees :
        , ordre affectations :
indice prochain: 1
 -> 4 -> 0 -> 1 -> 2 -> 3
```

# Annexe E

# Choix du langage

## E.1 Test objet langage C

### E.1.1 Objet C

```
#include <stdio.h>
#include <stdlib.h>

struct _A {
  int (*f1)();
};

struct _B {
  int (*f1)();
  int (*f2)();
};

typedef struct _A A;
typedef struct _B B;

typedef void* Objet;

int f () {
  printf("Je suis dans f !\n");
  return 1;
}

int g () {
  printf("Je suis dans g !\n");
  return 2;
}

int main (int argc,char** argv) {
  Objet a;

  a=malloc(sizeof(B));
  ((B*)a)->f2=&g;
  ((A*)a)->f1=&f;

  // Si l'ordre est le même on a le droit !
  ((A*)a)->f1();
  ((B*)a)->f1();
  ((B*)a)->f2();
```

```
  return 0;
}
```

### E.1.2 Tri C

```
#include <stdio.h>
#include <stdlib.h>

// FONCTIONS
void remonte (int* tab,int t) {
  int i, imax;

  imax=0;
  for (i=1;i<t;i++)
    if (tab[i]>tab[imax])
      imax=i;

  i=tab[t-1];
  tab[t-1]=tab[imax];
  tab[imax]=i;
}

void tri (int* tab,int t) {
  while (t>1) {
    remonte(tab,t);
    t--;
  }
}

void creer (int** tab,int taille) {
  int t;

  *tab=(int*)malloc(taille*sizeof(int));
  for (t=0;t<taille;t++)
    (*tab)[t]=rand();
}

// TEST
int main (int argc,char** argv) {
  unsigned int seed;
  int taille;
  int* tab;

  sscanf(argv[1],"%d",&taille);
  sscanf(argv[2],"%u",&seed);

  srand(seed);
  creer(&tab,taille);
  tri(tab,taille);

  return 0;
}
```

## E.2 Test objet langage C static

### E.2.1 Objet C static

```
#include <stdio.h>
#include <stdlib.h>

// DEFINITIONS
typedef void Objet;

struct _A {
  int (*f1)(Objet);
};

struct _B {
  int (*f1)(Objet);
  int (*f2)(Objet);
};

typedef struct _A A;
typedef struct _B B;


// FONCTIONS
int f (Objet o) {
  printf("Je suis dans f pour l'objet %p!\n",&o);
  return 1;
}

int g (Objet o) {
  printf("Je suis dans g pour l'objet %p!\n",&o);
  return 2;
}

void creer (char v,int (**fct)(Objet)) {
  if (v=='f')
    *fct=&f;
  if (v=='g')
    *fct=&g;
}

// TEST
int main (int argc,char** argv) {
  Objet a;

  a=*(malloc(sizeof(B)));
  creer('g',&(((B)a).f2));
  creer('f',&(((A)a).f1));

  // Si l'ordre est le même on a le droit !
  printf("TYPAGE A :\n");
  printf("(f1) : ");
  ((A)a).f1(a);
  printf("(f2) : n'existe pas !\n");
  printf("TYPAGE B :\n");
  printf("(f1) : ");
```

```
  ((B)a).f1(a);
  printf("(f2) : ");
  ((B)a).f2(a);

  return 0;
}
```

### E.2.2 Tri C static

```
#include <stdio.h>
#include <stdlib.h>

int* tab;

// FONCTIONS
void remonte (int t) {
  int i, imax;

  imax=0;
  for (i=1;i<t;i++)
    if (tab[i]>tab[imax])
      imax=i;

  i=tab[t-1];
  tab[t-1]=tab[imax];
  tab[imax]=i;
}

void tri (int t) {
  while (t>1) {
    remonte(t);
    t--;
  }
}

void creer (int taille) {
  int t;

  tab=(int*)malloc(taille*sizeof(int));
  for (t=0;t<taille;t++)
    tab[t]=rand();
}

// TEST
int main (int argc,char** argv) {
  unsigned int seed;
  int taille;

  sscanf(argv[1],"%d",&taille);
  sscanf(argv[2],"%u",&seed);

  srand(seed);
  creer(taille);
  tri(taille);

  return 0;
```

```
}
```

## E.3 test objet langage C objet

```
#include <stdio.h>
#include <stdlib.h>

// DEFINITIONS
typedef void* Objet;

struct _A {
  int* tab;
  int taille;

  void (*remonte)(Objet,int);
};

struct _B {
  int* tab;
  int taille;

  void (*remonte)(Objet,int);
  void (*tri)(Objet);
};

typedef struct _A A;
typedef struct _B B;


// FONCTIONS
void remonte (Objet o,int t) {
  A* a=(A*)o;
  int i, imax;

  imax=0;
  for (i=1;i<t;i++)
    if (a->tab[i]>a->tab[imax])
      imax=i;

  i=a->tab[t-1];
  a->tab[t-1]=a->tab[imax];
  a->tab[imax]=i;
}

void tri (Objet o) {
  B* b=(B*)o;
  int t=b->taille;

  while (t>1) {
    b->remonte(b,t);
    t--;
  }
}

void creerA (Objet o,int t) {
```

```
  A* a=(A*)o;

  a->taille=t;
  a->tab=(int*)malloc(t*sizeof(int));
  for (t=0;t<a->taille;t++)
    a->tab[t]=rand();
  a->remonte=&remonte;
}

void creerB (Objet o,int t) {
  creerA(o,t);
  ((B*)o)->tri=&tri;
}

// TEST
int main (int argc,char** argv) {
  B* b;
  unsigned int seed;
  int taille;

  sscanf(argv[1],"%d",&taille);
  sscanf(argv[2],"%u",&seed);

  srand(seed);
  b=(B*)malloc(sizeof(B));
  creerB(b,taille);
  b->tri(b);

  return 0;
}
```

### E.3.1 Test objet langage C++

```
#include <iostream>

class A {
public :
  int* tab;
  int taille;

  A (int t) {
    taille=t;
    tab=new int[t];
    for (t=0;t<taille;t++)
      tab[t]=rand();
  };

  void remonte (int t) {
    int i, imax;

    imax=0;
    for (i=1;i<t;i++)
      if (tab[i]>tab[imax])
        imax=i;

    i=tab[t-1];
```

```
    tab[t-1]=tab[imax];
    tab[imax]=i;
  };
};

class B : public A {
public:
  B (int t): A(t) {};

  void tri () {
    int t=taille;

    while (t>1) {
      remonte(t);
      t--;
    }
  };
};

int main (int argc,char** argv) {
  B* b;
  unsigned int seed;
  int taille;

  sscanf(argv[1],"%d",&taille);
  sscanf(argv[2],"%u",&seed);

  srand(seed);
  b=new B(taille);
  b->tri();
}
```


UNIVERSITÉ D'ARTOIS
cril
centre de recherche en informatique de lens
CNRS
CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE