UNIVERSITÉ DE STRASBOURG

L2 - *Structures de Données et Algorithmes* 1

Examen du 22 mai 2013

Durée : 2H, documents non autorisés

Liste de Dominos

N.B. On suppose prédéfini le module de spécification BASE, pour les sortes, opérations et propriétés des objets de base. Dans la programmation en C, on n'écrira que des fichiers-sources `.c`.

1. — Spécification des dominos (2 points)

Les dominos sont des pièces de jeu rectangulaires sur lesquelles figurent deux ensembles de points représentant un chiffre entre 1 et 6, séparés par un trait.

Ecrire un module de spécification *DOMINO* pour la sorte *Domino,* avec le constructeur de base :

- *dominouv* : tel que *dominouv(x, y)* soit un domino constitué, à gauche du chiffre x et à droite du chiffre y.

Spécifier les observateurs :

- *gauche* et *droite* qui renvoient le chiffre présent à gauche (respectivement à droite) sur le domino.

Spécifier également l'opération :

- *retourne* : qui retourne le domino, c'est à dire que le chiffre à gauche se retrouve à droite et le chiffre initialement à droite se retrouve à gauche.

2. — Spécification d'une suite de dominos (6 points)

Dans une partie (simplifiée) de dominos les dominos sont posés en ligne les uns à la suite des autres, en respectant la condition essentielle que deux dominos placés côte à côte affichent le même chiffre sur leur côté en contact.
En début de partie un premier domino est posé, sans condition. A partir de cet instant la suite de dominos peut être prolongée, à gauche ou à droite, en apposant un nouveau domino, s'il respecte la condition préalable d'avoir un chiffre commun au niveau du contact. Un domino peut être retourné pour respecter cette condition.

Ecrire un module de spécification *SUITE* pour la sorte *Suite,* qui décrit une suite de dominos. Les constructeurs de base sont :

- *suitenouv* : suite vide.
- *adjg* : tel que *adjg(s,d)* est l'adjonction à gauche dans la suite *s* du domino *d*.
- *adjd* : tel que *adjd(s,d)* est l'adjonction à droite dans la suite *s* du domino *d*.

Donner l'expression de construction avec *suitenouv, adjg, adjd* de la suite *s1* contenant les dominos [1-3], [1-6] et [3-4] ajoutés dans cet ordre.

Dans *SUITE*, définir par rapport à *suitenouv, adjg, adjd* les fonctions :
- *eg, ed* : qui renvoie (sans le retirer) le domino à l'extrémité gauche (respectivement droite) de la suite de dominos
- *supg, supd* : pour retirer un domino à la suite à gauche (respectivement à droite)
- *v :* telle que *v(s)* est *vrai* si et seulement la suite *s* est vide
- *l :* telle que *l(s)* renvoie la longueur en nombre de dominos de la suite *s*

3. ─ Implantation des suites de dominos (6 points)

Écrire une structure de données en C permettant de représenter les dominos.

Écrire une structure de données en C offrant une représentation simplement chaînée d'une suite de dominos.

Dessinez dans cette implantation une représentation schématique de la mémoire correspondant à la suite vide, à la suite de dominos [1-3], puis à la suite de dominos [6-1] [1-3] [3-4].

Programmer en C, pour ces deux structures de données, toutes les opérations des questions **1.** et **2,** itérativement, avec effet de bord et apparence fonctionnelle. Vous dessinerez un schéma de la mémoire pour les fonctions *adjg, adjd, supg, supd* détaillant les étapes de leur déroulement.

3. ─Autre opération (2 points)

Spécifier une opération *retournersuite* qui retourne une suite de dominos de telle manière que la suite de dominos résultat soit l'exact miroir de la suite initiale. Par exemple la suite de dominos [6-1] [1-3] [3-4] renverrait la suite [4-3] [3-1] [1-6].

4. ─ Programmation de l'opération (4 points)

Programmer en C avec effet de bord et apparence fonctionnelle la fonction de la question **3.** une première fois récursivement puis une seconde fois itérativement. Détailler à chaque fois par des schémas l'évolution de la mémoire.
Quelle est la complexité dans le pire des cas de vos algorithmes ? Expliquer.

Corrigé

1. –Spécification des dominos (2 points)

spéc DOMINO **étend** BASE
sorte Domino
opérations
dominouv : Nat Nat → Domino /* constructeur */
gauche : Domino → Nat /* observateur */
droite : Domino → Nat /* observateur */
retourne : Domino → Domino
axiomes
gauche(dominouv(x,y)) = x
droite(dominouv(x,y)) = y
retourne(dominouv(x,y)) = dominouv(y,x)
fspéc

2. – Spécification des suites de dominos (4 points)

spéc SUITE **étend** DOMINO,BASE
sorte Suite
opérations
suitenouv : → Suite /* constructeur vide */
adjg : Suite Domino → Suite /* adjonction gauche */
adjd : Suite Domino → Suite /* adjonction droite */
eg,ed : Suite → Domino /* extrémité gauche/droite */
supg,supd : Suite → Suite /* suppression gauche/droite */
v : Suite → Bool
l : Suite → Nat
préconditions
pré adjg(s,d) = gauche(eg(s)) == droite(d) ou
gauche(eg(s)) == gauche(d)
pré adjd(s,d) = droite(ed(s)) == droite(d) ou
droite(ed(s)) == gauche(d)
pré eg(s) = !v(s)
pré ed(s) = !v(s)
pré supg(s) = !v(s)
pré supd(s) = !v(s)
axiomes
eg(adjg(s,d)) = d
eg(adjd(s,d)) = si v(s) alors d sinon eg(s)
ed(adjg(s,d)) = si v(s) alors d sinon eg(s)
ed(adjd(s,d)) = d
supg(adjg(s,d)) = s
supg(adjd(s,d)) = si v(s) alors suitenouv sinon adjd(supg(s),d)
supd(adjg(s,d)) = si v(s) alors suitenouv sinon adjg(supd(s),d)

supd(adjd(s,d)) = s
v(suitenouv) = vrai
v(adjg(s,d)) = faux
v(adjd(s,d)) = faux
l(suitenouv) = 0
l(adjg(s,d)) = l(s) + 1
l(adjd(s,d)) = l(s) + 1
fspéc

*s*1 = adjd(adjg(adjg(suitenouv,dominouv(1,3)), dominouv(1,6)),dominouv(3.4))
ou
*s*1 = adjd(adjg(adjd(suitenouv,dominouv(1,3)), dominouv(1,6)),dominouv(3.4))

3. – Implantation des suites de dominos (6 points)

```
#include "base.h"

typedef struct
{
      int g;
      int d;
} Domino;

typedef struct suitedom
{
      Domino d;
      struct suitedom * next;
} Piece, * Suite;

/* Opérations de Domino */

Domino dominouv(int x, int y)
{
      Domino d;
      d.x = x ;
      d.y = y ;
      return d;
}

int gauche(domino d)
{ return d.x; }

int droite(domino d)
{ return d.y; }

Domino retourne(domino d)
{
      int tmp = d.x;
      d.x = d.y ;
      d.y = tmp ;
      return d;
}

/* Opérations de Suite de dominos */
```

```
Suite suitenouv()
{ return NULL; }

// NB: ci-dessous on choisit Suite referencee par l'extremite gauche

Suite adjg(Suite s, Domino d)
{
      Suite ptr, newpiece;

      if (v(s))
      {
            newpiece=(Suite) malloc(sizeof(Piece));
            newpiece->d = d;
            newpiece->next = NULL;
            return newpiece;
      }
      else
      {
            newpiece=(Suite) malloc(sizeof(Piece));
            // si necessaire retourne le domino avant de le poser
            if (gauche(eg(s)) != droite(d)) d=retourne(d);
            newpiece->d = d;
            newpiece->next = s;
            return newpiece;
      }
}

Suite adjd(Suite s, Domino d)
{
      Suite ptr, newpiece;

      if (v(s))
      {
            newpiece=(Suite) malloc(sizeof(Piece));
            newpiece->d = d;
            newpiece->next = NULL;
            return newpiece;
      }
      else
      {
            newpiece=(Suite) malloc(sizeof(Piece));
            // si necessaire retourne le domino avant de le poser
            if (droite(ed(s)) != gauche(d)) d=retourne(d);
            newpiece->d = d;
            newpiece->next = NULL;
            // deplacement sur la derniere piece de s
            for(ptr=s; ptr->next!=NULL; ptr = ptr->next);
            ptr->next = newpiece;
            return s;
      }
}

Domino eg(Suite s)
{
      return s->d;
}
```

```
Domino ed(Suite s)
{
      Suite ptr;
      // deplacement sur la derniere piece de s
      for(ptr=s; ptr->next!=NULL; ptr = ptr->next);
      return ptr->d;
}

Suite supg(Suite s)
{
      Suite ptr;
      ptr = s->next;
      free(s);
      return ptr;
}

Suite supd(Suite s)
{
      Suite ptr;
      // 1 seule piece
      if (s->next == NULL)
      {
            free(s);
            return NULL;
      }
      // deplacement sur l'avant derniere piece de s
      for(ptr=s; ptr->next->next!=NULL; ptr = ptr->next);
      free(ptr->next);
      ptr->next == NULL;
      return s;
}

Bool v(Suite s)
{
      return (s == NULL);
}

int l(Suite s)
{
      Suite ptr;
      int i = 0;

      for(ptr=s; ptr!=NULL; ptr = ptr->next) i++;
      return i;
}
```