Université de Strasbourg
UFR de mathématique et d'informatique

L2 informatique
automne 2016

Structures de données et algorithmes

TP3 : entiers naturels

Booléens

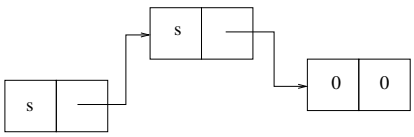
On implante les booléens simplement en respectant les conventions C :

```
// fichier bool.h

#ifndef __BOOL_H
#define __BOOL_H

typedef enum {faux=0,vrai} Bool;

#endif
```

Expliquez cette implantation.

Entiers naturels

On veut maintenant implanter en C les entiers naturels d'une manière fidèle à la spécification vu en cours. Nous avons vu que les formes canoniques pour termes de sorte `Nat` sont de la forme `s(s(s(...0 ...)))` et nous avons indiqué que cela correspondait à la numérotation unaire des entiers. Nous traduisons ceci par une structure C possédant 2 champs, l'un indiquant le symbole fonctionnel (de la forme normale équivalente) : `0` ou `s` et le deuxième indiquant quel est, éventuellement l'argument. Par exemple, l'entier naturel `s(s(0))` est schématiquement représenté en mémoire comme suit :



On obtient le fichier header `nat0.h` suivant :

```
// fichier nat0.h
# include"bool.h"

#ifndef __NAT_H__
#define __NAT_H__

#define _Nat_ struct sn *      // permet de définir à l'avance ce que va être un Nat

struct sn {
         char f; // symbole fonctionnel '0' ou 's'
         _Nat_ p; // argument de s, ou encore prédécesseur
                         };

typedef _Nat_ Nat;              // un Nat est un pointeur sur une structure sn

Nat 0();                        // le naturel 0
Nat s(Nat n);                   // fonction successeur
Nat add(Nat n, Nat m);          // addition
Nat mult(Nat n, Nat m);         // multiplication
Bool zero(Nat n);               // test de nullité
Bool estsucc(Nat n);            // test de non nullité
```

```
Bool egN(Nat n, Nat m);          // test d'égalité
Bool spp(Nat n, Nat m);          // strictement plus petit

// fonction auxiliaires
void printn(Nat n);              // affichage de n en unaire (par ex. s(s(s(0)))
Nat u2nat(unsigned u); // traduction de décimal à unaire
unsigned int nat2u(Nat n);       // transforme un Nat en entier non signé

#endif
```

1. La définition des types a été décomposée en 3 étapes `#define _Nat_ ...`, `struct sn ...` et `typedef _Nat_ Nat`. Lisez, comprenez et expliquez en commentaire comme se fait cette définition de type et pourquoi je l'ai fait de cette manière.
2. Expliquez pourquoi le deuxième champ est un pointeur et non une structure `sn` ? Pourquoi cela entraîne-t-il que le type `Nat` doit être un pointeur ?
3. Programmez dans un fichier que vous nommerez `nat0.c` les fonctions indiquées dans le fichier `nat0.h`. Programmez dans un fichier de test, un programme permettant de vérifier vos fonctions.
4. Qui dit pointeur dit effets de bord : qu'en dites vous dans le cas présent ?
5. Programmez les fonctions classiques de sommation, factorielle, etc. avec cette structure de données en suivant le schéma vu en TD.

Compléments

Un fichier Makefile simple pour ce TP :

```
test : lib/nat.o testn.c
        gcc lib/nat.o testn.c -o test
lib/nat.o : nat.c include/nat.h
        gcc -c nat.c -o lib/nat.o
clean :
        rm lib/*.o
clean_all :
        rm test lib/*.o
```

Pour les étudiants qui ont fini en avance :

1. Montrez qu'en fait, dans la structure de données indiquée, le premier champs n'est pas utile et qu'on peut programmer tout ceci plus simplement. Redéfinissez une structure de données plus simple et reprogrammez le tout dans des fichiers nommés respectivement `nat.h` et `nat.c`
2. Réutilisez le programme dans `testn.c` pour vérifier votre nouvelle implantation. Est-ce qu'il faut l'aménager ?
3. Faites des tests avec des grands entiers et discutez de l'efficacité de cette implantation.