Algorithmique et Programmation 2
Travaux Pratiques – Séance 7

À déposer à la fin de la séance 7 sur Arche

Introduction

Soit b un entier naturel supérieur ou égal à 2. Tout entier naturel n s'écrit comme suit :

$$n = c_0 \times b^0 + c_1 \times b^1 + \cdots + c_p \times b^p$$

où $p+1$ est le nombre de chiffres significatifs de n dans la base b : les entiers c_i, tous compris entre 0 et $b-1$, sont appelés les chiffres de n en base b. On suppose en général que $c_p \neq 0$.

Par exemple, si $b = 10$ et $n = 2021$ alors :

$$2021 = \underbrace{1}_{c_0} \times 10^0 + \underbrace{2}_{c_1} \times 10^1 + \underbrace{0}_{c_2} \times 10^2 + \underbrace{2}_{c_3} \times 10^3$$

Nous représenterons l'entier 2021 par la liste $L = (1\ 2\ 0\ 2)$ (le premier élément à gauche est l'unité). Cette représentation n'est pas unique : nous pouvons ajouter autant de 0 qu'on veut en fin de liste pour représenter le même entier : la liste $L = (1\ 2\ 0\ 2\ 0\ 0\ 0)$ représente également l'entier 2021. En particulier, l'entier naturel 0 sera représenté par n'importe quelle liste ne contenant que des 0, y compris la liste vide.

Une telle représentation s'avère utile lorsque nous devons représenter des entiers naturels de très grand taille et que l'on veut s'affranchir des limites des types `unsigned int` voire `unsigned long int` en C. Le plus grand entier naturel représentable sur 64 bits est $2^{64} - 1 = 18446744073709551615$

La librairie `liste.h`

Dans cette séance de TP, nous utiliserons l'enregistrement `struct Liste` pour représenter de très grands entiers naturels.

```
typedef struct Liste* liste;
struct Liste{
  unsigned int  premier ;
  liste suivant;
};
```

L'enregistrement est muni des opérations primitives suivantes :

```
/* SIGNATURES DES OPERATIONS PRIMITIVES */
// constructeurs
liste l_vide () ;
liste cons (int x, liste L) ;
// acces
```

```
bool est_vide (liste L) ;
int prem (liste L) ;
liste reste (liste L) ;
void liberer_liste (liste L) ;
```

Dans ce travail, nous désignerons par *grand entier* une liste d'entiers naturels tous strictement inférieurs à 10. Vous complèterez le fichier ci-dessous avec les programmes demandés.

```
/* fichier tp7.c */
/* etudiant : nom prenom */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "liste.h"

int main(int argc, char** argv){
  return EXIT_SUCCESS;
}
```

La commande de compilation/exécution est la suivante :

```
super_etudiant@ubuntu:$ gcc -Wall tp7.c liste.c -o tp7
super_etudiant@ubuntu:$ ./tp7
```

Question 1 — Constante de base
Définissez la constante `b` dans votre programme à l'aide de la directive `#define`.

Question 2 — Grand entier nul
Écrivez et testez une fonction `est_nul` qui teste si un grand entier représente l'entier naturel 0.

Question 3 — Afficher un grand entier
Écrivez et testez une procédure `afficher_grand_entier` qui affiche un grand entier passé en entrée. Par exemple l'exécution de :

```
int main(int argc, char** argv){
  liste L = cons(5,cons(9, cons(6, cons(8, cons(0, cons(0, l_vide()))))));
  afficher_grand_entier(L);
  return EXIT_SUCCESS;
}
```

produira l'affichage

```
super_etudiant@ubuntu:$ gcc -Wall tp7.c liste.c -o tp7
super_etudiant@ubuntu:$ ./tp7
8695
```

Question 4 — Successeur d'un grand entier
Écrivez et testez une fonction `successeur` qui calcule et renvoie le successeur d'un grand entier.

Question 5 — Entier naturel vers grand entier
Écrivez une fonction `entier_naturel_vers_grand_entier` qui prend en entrée un entier naturel et renvoie le grand entier correspondant.

Question 6 — Grand entier vers entier naturel
Écrivez une fonction `grand_entier_vers_entier_naturel` qui prend en entrée un grand entier et renvoie l'entier naturel correspondant.

Question 7 — Somme de deux grands entiers
Écrivez et testez une `somme` qui calcule et renvoie la somme de deux grands entiers.

Remarque : Une solution très inefficace consiste à s'appuyer directement sur les axiomes de la fonction `plus` vus en cours. Ici, nous souhaitons que vous utilisiez la méthode d'addition apprise à l'école élémentaire. Par exemple, $8639 + 4790$ est calculé de la manière suivante :

$$\begin{array}{rr} & {\scriptstyle 1\ 1} \\ & 8\,6\,4\,9 \\ + & 4\,7\,9\,0 \\ \hline & 1\,3\,4\,3\,9 \end{array}$$

TABLE 1 – Comme en CP/CE1...

Question 8 — Produit de deux grands entiers
Écrivez et testez une fonction `produit` qui calcule et renvoie le produit de deux grands entiers.

Question 9 — Factorielle d'un grand entier
Écrivez et testez une fonction `factorielle` qui calcule et renvoie la factorielle d'un entier naturel. Le test se fera entre-autres avec le calcul de 100! et le code suivant :

```
void test_factorielle(){
  unsigned int n = 100 ;
  liste M = fatorielle(n);
  printf("%u!␣=␣", n);
  afficher_grand_entier(M);
}
```

Rappel : La fonction `factorielle`, notée généralement !, est définie ainsi :

$$\begin{array}{rccl} (\,)! & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & n & \longmapsto & \begin{cases} 1 & \text{si } n = 0 \\ \displaystyle\prod_{k=1}^{n} k & \text{si } n > 0 \end{cases} \end{array}$$

En particulier, si $n > 0$, $n! = n \times (n-1)!$.

Question 10 — Coefficient binomial de deux entiers naturels
Écrivez et testez une fonction `CoefficientBinomial` qui calcule et renvoie, sous forme d'un grand entier, le coefficient binomial de deux entiers naturels.

Rappel :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

Par exemple

$$\binom{2021}{13} = 1\,449\,724\,926\,252\,449\,277\,110\,566\,660\,163\,740$$

La représentation en binaire de $\binom{2021}{13}$ nécessite 111 bits Les types `unsigned int` (4 octets = 32 bits) et `unsigned long int` (8 octets = 64 bits) ne peuvent donc pas représenter cet entier.