Développement C **L2 CUPGE 2018–2019**

Travaux pratiques Fractions

1 Introduction

Le but de ce thème est de traduire des réels de lintervalle $[0, 1]$ en fractions à une précision donnée près. On va travailler sur 5 fichiers :

- le fichier src/Fractions/fraction.mk pour compiler automatiquement les fichiers ;
- le fichier src/Fractions/fraction.c qui contiendra limplémentation des fractions ;
- le fichier include/fraction.h qui contiendra les en-têtes des fonctions et types définis ;
- le fichier src/Fractions/test_fraction.c qui contiendra les tests sur les fractions ;
- le fichier src/Fractions/recherche.c qui contiendra les sources du programme bin/fraction de recherche de fraction.

2 Compilation

Pour effectuer les tests, on va réutiliser les fichiers `tests.h` et la librairie `entrees`. Assurezvous quils sont bien dans le répertoire ~/src/include. Cela signifie que parmi les options de gcc, lors de la compilation, il faut ajouter :

`-I "~/programmation-en-C/src/include"`

De plus, pour voir plus facilement le résultat des tests, il faut compiler avec :

`-DUSECOLORS`

Ensuite, quand on en sera à la version finale, on enlèvera les tests en compilant avec loption :

`-Dproduction`

Enfin, pour utiliser la fonction `fabsl` : `double` → `double` qui calcule la valeur absolue dun double et est définie dans la bibliothèque `<math.h>`, il faut compiler avec loption :

`-lm`

On paramétrise le fichier `fraction.mk` avec des variables en le commençant par les lignes suivantes :

```
GCC=gcc
LIBS=-lm -lentrees
INCLUDE=-I "~/programmation-en-C/src/include"
FLAGS=-DUSECOLORS
```

On peut utiliser ces variables dans les règles. Par exemple :

```
test_fraction : test_fraction.c
	fraction.c $(GCC) $(FLAGS) -o $@ $@.c $(INCLUDE) $(LIBS)
```

Test du Makefile. On peut préciser ce qui va être fait en marquant le but dune règle. Par exemple :

```
make -f fraction.mk test_fraction
```

3 Fichier `fraction.c`

I Exercice : *Base sur les structures*

(a) Déclarer une structure `fraction_s` qui contient deux entiers, un numérateur p et un dénominateur q

Solution

```
struct fraction_s {
  int p ;
  int  q ;
} ;
```

(b) Déclarer un nouveau type fraction qui est le type des structures `fraction_s`.

Solution

```
typedef struct fraction_s fraction ;
```

(c) Écrire une fonction `nouvelle_fraction` qui prend en entrée deux entiers, et qui renvoie une fraction avec ces entiers comme, respectivement, le numérateur et le dénominateur.

Solution

```
fraction nouvelle\_fraction ( int p , int q )
{
  fraction r ;
  r.p = p ;
  r.q = q ;
  return r ;
}
```

(d) Écrire deux fonctions `numerateur` et `denominateur` qui renvoient respectivement le numerateur et le dénominateur de la fraction passée en entrée.

Solution

```
int
numerateur ( fraction f )
{
return f.p ;
}
int
```

```
denominateur ( fraction f )
{
return f.q ;
}
```

Note de bonne programmation. Dans la suite, utilisez uniquement les fonctions `nouvelle_fraction`, `numerateur`, et `denominateur`.

II Exercice : *Opérations simples sur les fractions*

(a) Écrire une fonction `fractioncmp` qui prend en entrée deux fraction f_1 et f_2 et rend un entier du même signe que $f_1 - f_2$ (pour lopération habituelle de soustraction) et 0 si les deux fractions sont égales.

Solution

```
int fractioncmp ( fraction a , fraction b )
{
  return numerateur ( a )  denominateur ( b )
  - numerateur ( b )  denominateur ( a ) ;
}
```

(b) 'Ecrire une fonction `print_fraction` qui affiche une fraction.

Solution

```
void print_fraction ( fraction f )
{
  printf ( "␣␣%d\n————————\n␣␣␣%d\n" , numerateur ( f ) , denominateur (
}
```

(c) Écrire une fonction `addition_cancre` qui fait laddition des cancres de 2 fractions :

$$\frac{p}{q} \oplus \frac{p'}{q'} = \frac{p+p'}{q+q'}$$

On note quil sagit aussi de laddition des professeurs, puisque

$$\frac{5}{8} \oplus \frac{8}{12} = \frac{13}{20}$$

Solution

```
fraction
addition_cancre ( fraction f1 , fraction f2 )
{
return nouvelle_fraction ( numerateur ( f1 ) + numerateur ( f2 ) ,
```

```
denominateur ( f1 ) + denominateur ( f2 ) ) ;
}
```

(d) Écrire une fonction `eval_fraction` qui renvoie le nombre (de type double) qui est la valeur de la fraction passée en argument.

Solution

```
double eval_fraction ( fraction f )
{
  return ( (  double ) numerateur ( f ) ) / denominateur ( f ) ;
}
```

III Exercice : *Recherche par dichotomie dune fraction*

Laddition des cancres permet de trouver par dichotomie la forme réduite dune fraction approchant un nombre positif. La justification mathématique est donnée par le lemme suivant :

Lemme 1. *Si* $\frac{a}{b} < \frac{p}{q} < \frac{c}{d}$ *et* $ad - bc = -1$ *alors :*

$$\begin{cases} p & \geq & a+c \\ q & \geq & b+d \end{cases}$$

Donc si on cherche un rationnel entre deux autres nombres quon aura calculé, soit il est égal à leur somme des cancres, soit il est entre leur somme des cancres et lun des deux, et ses numérateurs et dénominateurs sont plus grand que ceux de la somme des cancres.

(a) En déduire une fonction `ftofraction` qui prend en entrée un double et rend une fraction irréductible égale à ce double à une précision absolue ε près.

Aide. Il peut commencer par considérer uniquement le cas dun nombre entre 0 et 1 et faire une recherche dichotomique en partant de $0 = \frac{0}{1}$ et $1 = \frac{1}{1}$. On peut généraliser ensuite aux nombres positifs en posant $+\infty = \frac{1}{0}$.

Aide pour le C. Pour calculer la valeur absolue, utilisez la fonction `fabsl` définie dans `math.h`.

Solution

```
fraction ftofraction ( double x , double epsilon )
{
  double e ;

  fraction min = nouvelle_fraction ( 0 , 1 ) , //zero
  max = nouvelle_fraction(1 , 0 ) ; //+infini
  fraction courant ;

  do {
```

```
    courant = addition_cancre ( min , max ) ;
    e = eval ( courant ) ;
   if ( e < x )
    min = courant ;
    else
    max = courant ;
  } while ( fabsl ( e  x ) > epsilon ) ;
  return courant ;
}
```

4 Écriture dun programme

On veut maintenant écrire un programme complet qui va effectuer la mise sous forme de fraction dun décimal. Lors de lappel—*i.e.*, au début de la fonction main—le programme commence par regarder combien il a darguments :

- sil en a 0, il demande (en utilisant `LireDecimal`) à lutilisateur un nombre à chercher et une précision ;
- sil en a 1, on suppose que cest le nombre à chercher, et on utilise la précision par défaut de 0.0000001 ;
- sil en a 3, le premier doit être la chaîne de caractères `"-p"`, le second la précision, et le troisième est le nombre à chercher.

IV Exercice : *Utilisation des arguments*

Implémentez ce programme dans un fichier recherche.c.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fraction.h"
#include "entrees.h"
void
demande_utilisateur ( double  x , double  epsilon )
{
  printf ( "␣Entrez␣un␣nombre␣à␣traduire␣en␣fraction :\n␣" ) ;
  lire_decimal_long ( x ) ;
  printf ( "␣Entrez␣␣l'erreur␣admissible␣epsilon␣ :\n␣" ) ;
  do {
    lire_decimal_long ( epsilon ) ;
  }  while (  epsilon < 0 ) ;
}

int
main ( int argc , char  argv [ ] )
{
  double x , epsilon ;
```

```
  switch ( argc )
  {
    case 1 :
    demande_utilisateur ( &x , &epsilon ) ;
    break ;
    case 2 :
    x = atof ( argv [ 1 ] ) ;
    epsilon = 0.0000001 ;
    break ;
    case 4 :
    if ( strcmp ( argv [ 1 ] , "p " ) == 0 )
    {
      x = atof ( argv [ 3 ] ) ;
      epsilon = atof ( argv [ 2 ] ) ;
    }
    else
    goto erreur_argument ;
    break ;
    default :
    goto erreur_argument ;
  }
  print_fraction ( ftofraction ( x , epsilon ) ) ;
  return 0 ;
  erreur_argument :
  printf ( " Utilisation : %s [[ --p <precision >] <x >]\n " , argv [ 0 ] ) ;
  return 1 ;
}
```

5 Programmation modulaire

Le fichier fraction.h a un défaut, qui est minime pour nous, mais qui est handicapant pour les gros programmes : les programmeurs qui lutilisent peuvent créer de nouvelles structures en dehors des fonctions quon a défini car ils ont accès au format de la structure.

La technique permettant déviter cela consiste à déclarer le type `fraction` non plus comme une structure fraction, mais comme un pointeur vers une telle structure. Si les fichiers ont été écrits correctement, ce changement est très rapide.

V Exercice : *Changement de type des fractions*

(a) Changez le type fraction en un type qui pointe sur des structures dans `fraction.h` et `fraction.c`.

Solution

```
typedef struct fraction_s * fraction ;
```

(b) Changez les fonctions `nouvelle_fraction`, `numerateur`, et `denominateur` du fichier

`fraction.c` pour les adapter au nouveau type.

(c) Recompilez `test_fraction` et `recherche`.

(d) Enlever la déclaration de la structure `fraction_s` du fichier `fraction.h`. Recompilez `test_fraction` et `recherche`.

Solution

```
GCC=gcc
LIBS=-lm ../lib/cours.o
INCLUDE=-I "../include/ "
FLAGS=-DUSECOLORS

fraction.o : fraction.c
    $(GCC) $(FLAGS) c $< $(INCLUDE)

test_fraction : test_fraction.c fraction.c
    $(GCC) $(FLAGS) o $@ $@. c $(INCLUDE) $(LIBS)

fraction : recherche.c fraction.o
    $(GCC) $(FLAGS) o $@ recherche.c $(INCLUDE) $(LIBS) fraction.o

recherche : recherche.c fraction.o
    $(GCC) $(FLAGS) o $@ recherche.c $(INCLUDE) $(LIBS) fraction.o
```