Algorithmique et Structures de Données - Initiation

Plan du cours

- Intro : fonctionnement du cours et projets ;
- Rappels : un peu de C ?
- Démarrage en douceur : une structure de donnée pour gérer les rationnels ;
- La PILE (et la FILE) :
  - à quoi ça sert ?
  - un exemple d'utilisation avec passage d'écriture infixée à postfixée.
- Cherchons un peu :
  - Quand on cherche, il s'agit d'être méthodique !
  - Tri insertion ;
  - Tri fusion.
- Les Listes chaînées :
  - simples, doubles, circulaires
  - et après ?
- Généricité : pile, tris, listes ...

Références

- **Algorithms in C** — par Robert Sedgewick chez Addison-Wesley
- **Algorithms in Java** — par Robert Sedgewick chez Addison-Wesley (en deux parties 1-4 et 5)
- **C Programming Language** — par Brian W. Kernighan et Dennis M. Ritchie chez Prentice Hall

Type	**Occupation mémoire**	**Plage de valeurs**
char	1 octet	-128 à 127
unsigned char	1 octet	0 à 255
int	4 ou 8 octets	Selon l'architecture
unsigned int	4 ou 8 octets	Selon l'architecture
short	2 octets	-32768 à 32767
unsigned short	2 octets	0 à 65535
long	4 octets	-2147483648 à 2147483647
unsigned long	4 octets	0 à 4294967295
long long	8 octets	-2^{63} à $2^{63}-1$
unsigned long long	8 octets	0 à $2^{64}-1$
Type à virgule flottante		
float	4 octets	3.4×10^{-38} à 3.4×10^{38} (IEEE 754)
double	8 octets	1.7×10^{-308} à 1.7×10^{308} (IEEE 754)
long double	10 octets	3.4×10^{-4932} à 3.4×10^{4932} (IEEE 754)

Table 1 – Types de données élémentaires

Format	**Type de donnée**	
%d	Entier	`short, int`
%i	Entier	`short, int`
%u	Entier non signé	`unsigned short, unsigned int`
%o	Entier octal	`short, int`
%x, %X	Entier hexadécimal	`short, int`
%l, %ld, %li, %lu, %lo, %lx, %lX	Entier long	`long`
%lld, %lli, %llu, %llo, %llx, %llX	Entier long 64 bits	`long long`
%c	Caractère ASCII	`char, unsigned char`
%f, %F, %g, %G	Flottant	`float, double`
%e, %E	Flottant (exponentiel)	`float, double`
%Lf, %LF, %Lg, %LG, %Le, %LE	long double	`long double`
%s	Chaine de caractère	`char *`
%p	Pointeur	`(type) *`

Table 2 – (Quelques) Formatages de la fonction `printf`

Séquence d'échappement	**Action**
\n	Nouvelle ligne (`new line`)
\t	Tabulation horizontale
\v	Tabulation verticale
\b	Retour d'un caractère arrière (`backspace`)
\r	Retour chariot (`carriage return`)
\f	Saut de page (`form feed`)
\a	Signal sonore (`alarm`)
\'	Affiche une apostrophe
\"	Affiche un guillemet
\\	Affiche un `Backslash`
\ddd	Affiche les codes ASCII en octale
\xddd	Affiche les codes ASCII en hexadécimale

Table 3 – Séquences d'échappement

```c
#include <stdio.h>
int main(void) {
  printf("Hello World\n");
  return 0;
}
```

FIGURE 1 – Mon premier programme C

```c
#include <stdio.h>
/* Ceci est un commentaire */
static const char _helloWorld[] = "Hello World !\n";
void p01(void) {
  printf(_helloWorld);
}
void p02(void) {
  int i;
  for(i = 0; _helloWorld[i] != '\0'; i++)
    putchar(_helloWorld[i]);
}
void p03(char * ch) {
  printf("%s", ch);
}
void p04(char * ch) {
  while(*ch)
    putchar(*ch++);
}
int main(void) { /* Main est la fonction principale */
  char * chaine = (char *)_helloWorld;
  p01();
  p02();
  p03(chaine);
  p04(chaine);
  return 0;
}
```

FIGURE 2 – Différentes méthodes pour imprimer une chaîne de caractères

```
bash-2.05b$ gcc -Wall helloWorld02.c -o hello
bash-2.05b$ ./hello
Hello World
Hello World
Hello World
Hello World
bash-2.05b$
```

FIGURE 3 – Ma première compilation C

```
1  import java.lang.*;
2  /* Ceci est un commentaire */
3  public class HelloWorld {
4      private static final String _helloWorld = "Hello World !\n";
5      public void p01() {
6          System.out.printf(_helloWorld);
7      }
8      public void p02() {
9          for(int i = 0; i < _helloWorld.length(); i++)
10             System.out.printf("%c", _helloWorld.charAt(i));
11     }
12     public void p03(String str) {
13         System.out.printf("%s", str);
14     }
15     public void p04(String str) {
16         char c;
17         int i = 0;
18         //while((c = str.charAt(i++)) != '\0') ne fonctionne pas !
19         while(i < _helloWorld.length()) {
20             c = str.charAt(i++);
21             System.out.printf("%c", c);
22         }
23     }
24     public static void main(String[] args) {
25         String chaine = _helloWorld;
26         HelloWorld me = new HelloWorld();
27         me.p01();
28         me.p02();
29         me.p03(chaine);
30         me.p04(chaine);
31         return;
32     }
33 }
```

FIGURE 4 – Le même (plus ou moins) en JAVA

```
1  int main(int argc, char *argv[]) {
2    int i;
3    char num[] = "123456"; /* Mettez des chiffres */
4    for(i = 0; num[i] != '\0'; i++)
5      putchar('a' + num[i] - '0');
6    putchar('\n');
7  }
```

FIGURE 5 – Que fait ce programme ?

```
1  /* Pseudo-code
2   * Algorithme d'Euclide (300 av-JC)
3   * Pour algorithme : voir Al-Khawarizmi (8-9e siècle)
4   */
5  pgcd(a, b) {
6          si a = b  ---->> a
7          si a > b  ---->> pgcd( (a - b), b )
8          si a < b  ---->> pgcd( b, (b - a) )
9  }
```

FIGURE 6 – Algorithme d'Euclide pour le calcul du PGCD.

```
1  typedef struct ratio_t *  ratio_t;
2  struct  ratio_t {
3    int p, q;
4  };
5  #define ratio_signe(p, q) ((p) * (q) < 0 ? -1 : 1)
6  #define ratio_num(r) ((r)->p)
7  #define ratio_denom(r) ((r)->q)
8  extern int ratio_pgcd(int p, int q);
9  extern ratio_t ratio_new(int p, int q);
10 extern void ratio_delete(ratio_t r);
11 extern ratio_t ratio_neg(ratio_t r);
12 extern ratio_t ratio_moins(ratio_t r1, ratio_t r2);
13 extern ratio_t ratio_plus(ratio_t r1, ratio_t r2);
14 extern ratio_t ratio_mul(ratio_t r1, ratio_t r2);
15 extern ratio_t ratio_div(ratio_t r1, ratio_t r2);
```

FIGURE 7 – Header `ratio.h` de la bibliothèque de calcul utilisant des nombres rationnels en C.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4  #include "ratio.h"
5  extern int ratio_pgcd(int p, int q) {
6    if(p == q) return p;
7    if(p > q) return ratio_pgcd(p - q, q);
8    return ratio_pgcd(q - p, p);
9  }
10 extern ratio_t ratio_new(int p, int q) {
11   int m = 1, s;
12   ratio_t r;
13   if(q == 0) {
14     fprintf(stderr, "Votre denominateur est nul.\n");
15     return NULL;
16   }
17   s = ratio_signe(p, q); p = abs(p); q = abs(q);
18   if(p) m = ratio_pgcd(p, q);
19   r = malloc(sizeof * r);
20   assert(r);
21   ratio_num(r) = s * p / m;
22   ratio_denom(r) = q / m;
23   return r;
24 }
25 extern void ratio_delete(ratio_t r) {
26   free(r);
27 }
28 extern ratio_t ratio_neg(ratio_t r) {
29   return ratio_new(-ratio_num(r), ratio_denom(r));
30 }
31 extern ratio_t ratio_moins(ratio_t r1, ratio_t r2) {
32   return ratio_new(ratio_num(r1) * ratio_denom(r2) - ratio_num(r2) * ratio_denom(r1),
33                    ratio_denom(r1) * ratio_denom(r2));
34 }
35 extern ratio_t ratio_plus(ratio_t r1, ratio_t r2) {
36   return ratio_new(ratio_num(r1) * ratio_denom(r2) + ratio_num(r2) * ratio_denom(r1),
37                    ratio_denom(r1) * ratio_denom(r2));
38 }
39 extern ratio_t ratio_mul(ratio_t r1, ratio_t r2) {
40   return ratio_new(ratio_num(r1) * ratio_num(r2),
41                    ratio_denom(r1) * ratio_denom(r2));
42 }
43 extern ratio_t ratio_div(ratio_t r1, ratio_t r2) {
44   return ratio_new(ratio_num(r1) * ratio_denom(r2),
45                    ratio_denom(r1) * ratio_num(r2));
46 }
47 #ifdef CHECK
48 int main(void) {
49   ratio_t a, b, c, d, e, f;
50   a = ratio_new(-5, 10);
51   b = ratio_new(-18, -27);
52   d = ratio_moins(c = ratio_neg(b), a);
53   f = ratio_mul(e = ratio_div(d, c), a); /* -1 / 8 */
54   printf("resultat : %d / %d\n", ratio_num(f), ratio_denom(f));
55   ratio_delete(a); ratio_delete(b); ratio_delete(c);
56   ratio_delete(d); ratio_delete(e); ratio_delete(f);
57   return 0;
58 }
59 #endif
```

FIGURE 8 – Fichier source `ratio.c` de la bibliothèque de calcul utilisant des nombres rationnels en C.

```
1  import java.lang.*;
2
3  public class Ratio {
4      private int _p, _q;
5      private static int signe(int p, int q) {
6          return (p) * (q) < 0 ? -1 : 1;
7      }
8      public static int pgcd(int p, int q) {
9          if(p == q) return p;
10         if(p > q) return pgcd(p - q, q);
11         return pgcd(q - p, p);
12     }
13     public static Ratio neg(Ratio r) {
14         return new Ratio(-r.getNumerateur(), r.getDenominateur());
15     }
16     public static Ratio moins(Ratio r1, Ratio r2) {
17         return new Ratio(r1.getNumerateur() * r2.getDenominateur() -
18                          r2.getNumerateur() * r1.getDenominateur(),
19                          r1.getDenominateur() * r2.getDenominateur());
20     }
21     public static Ratio plus(Ratio r1, Ratio r2) {
22         return new Ratio(r1.getNumerateur() * r2.getDenominateur() +
23                          r2.getNumerateur() * r1.getDenominateur(),
24                          r1.getDenominateur() * r2.getDenominateur());
25     }
26     public static Ratio mul(Ratio r1, Ratio r2) {
27         return new Ratio(r1.getNumerateur() * r2.getNumerateur(),
28                          r1.getDenominateur() * r2.getDenominateur());
29     }
30     public static Ratio div(Ratio r1, Ratio r2) {
31         return new Ratio(r1.getNumerateur() * r2.getDenominateur(),
32                          r1.getDenominateur() * r2.getNumerateur());
33     }
34     public Ratio(int p, int q) {
35         int m = 1, s;
36         if(q == 0) {
37             System.err.printf("Votre denominateur est nul.\n");
38             p = 0; q = 1;// ou faire un truc du genre : throw new Exception();
39         }
40         s = signe(p, q); p = Math.abs(p); q = Math.abs(q);
41         if(p != 0) m = pgcd(p, q);
42         _p = s * p / m;
43         _q = q / m;
44     }
45     public String toString() {
46         return _p + " / " + _q;
47     }
48     public int getNumerateur() {
49         return _p;
50     }
51     public int getDenominateur() {
52         return _q;
53     }
54     public static void main(String[] args) {
55         Ratio a, b, c, d, e, f;
56         a = new Ratio(-5, 10);
57         b = new Ratio(-18, -27);
58         d = moins(c = neg(b), a);
59         f = mul(e = div(d, c), a); /* -1 / 8 */
60         System.out.printf("resultat : %s\n", f);
61         return;
62     }
63 }
```

FIGURE 9 – Fichier source `Ratio.java` de la bibliothèque de calcul utilisant des nombres rationnels en JAVA.

```
1  public class Pile {
2      private static final int _max0 = 1 << 8;
3      private int _max, _haut;
4      private int[] _pile = null;
5      private void init(int size) {
6          _haut = 0;
7          _pile = new int[_max = size];
8      }
9      public Pile() {
10         init(_max0);
11     }
12     public Pile(int size) {
13         init(size);
14     }
15     public void push(int v) { /* Empiler */
16         _pile[_haut++] = v;
17     }
18     public int pop() { /* Depiler */
19         return _pile[--_haut];
20     }
21     public boolean vide() { /* Est-ce qu'elle est vide */
22         return _haut == 0;
23     }
24     static private long factR(int n) { /* Utilise la pile du langage */
25         return n > 0 ? n * factR(n - 1) : 1;
26     }
27     static private long factP(int n) { /* Utilise notre pile */
28         Pile p = new Pile();
29         while(n > 0) p.push(n--);
30         long f = 1;
31         while(!p.vide()) f *= p.pop();
32         return f;
33     }
34     static public void main(String[] args) {
35         for(int i = 0; i < 10; i++)
36             System.out.printf("factR(%d) = %7d\tfactP(%d) = %7d\n", i, factR(i), i, factP(i));
37         return;
38     }
39 }
40
```

FIGURE 10 – Création et utilisation d'une Classe Pile : l'exemple de la factorielle.

```c
#ifndef _PILE_H
#define _PILE_H
#define MAX 256
extern void push(int v);
extern int pop(void);
extern int vide(void);
#endif
```

```c
#include <stdlib.h>
#include <assert.h>
#include "pile.h"
static int pile[MAX], haut = 0;
extern void push(int v) { /* Empiler */
  pile[haut++] = v;
}
extern int pop(void) { /* Depiler */
  return pile[--haut];
}
extern int vide(void) { /* Tester si la pile est vide */
  return !haut;
}
```

```c
#include <stdio.h>
#include "pile.h"
static void infixe2postfixe(char * s, char * d) {
  while(*s) {
    if(*s >= '0' && *s <= '9') {
      do {
        *d++ = *s++;
      } while( *s >= '0' && *s <= '9');
      *d++ = ' ';
      if(!*s) break;
    }
    if((*s == ')')  && !vide()) { *d++ = (char)pop(); *d++ = ' '; }
    else if((*s == '+') ||
            (*s == '*') ||
            (*s == '/') ) push((int) *s);
    s++;
  }
  while(!vide()) {*d++ = (char)pop(); *d++ = ' '; }
  *d = '\0';
}
int main(void) {
  char source[MAX], destination[MAX<<1];
  do {
    if(!fgets(source, MAX, stdin)) break;
    infixe2postfixe(source, destination);
    printf("l'expression infixee : %s\n", source);
    printf("s'ecrit : %s en postfixe\n", destination);
  } while(1);
  return 0;
}
```

FIGURE 11 – Transformer une expression infixée en expression postfixée en utilisant la structure de données : pile (`pile.h / pile.c / infix2postfix.c`).