TD/TP de Programmation – Correction de la Planche 1
Calendriers julien et grégorien

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <assert.h>
```

1 Fonctions de base

```c
Date // version 1
Date_Make (int day, int month, int year, bool cal_is_julian)
{
  Date d;
  d.day= day; d.month= month; d.year= year;
  d.cal_is_julian= cal_is_julian;
  return d;
}

Date // version 2 (since C99)
Date_Make (int day, int month, int year, bool cal_is_julian)
{
  return (Date) {
    .day= day, .month= month, .year= year,
    .cal_is_julian= cal_is_julian
  };
}
```

```c
Date
Date_Epoch (bool cal_is_julian)
{
  return cal_is_julian ?
    Date_Make (19, 12, 1969, true) :
    Date_Make ( 1,  1, 1970, false);
}
```

```c
bool
Date_HasSameCalAs (Date d1, Date d2) {
  return d1.cal_is_julian == d2.cal_is_julian;
}
```

```c
int // safe version, normalized results
Date_CompareForSameCal (Date d1, Date d2)
{
  assert (Date_HasSameCalAs (d1, d2));
  if (d1.year  < d2.year ) return -1;
  if (d1.year  > d2.year ) return +1;
  if (d1.month < d2.month) return -1;
  if (d1.month > d2.month) return +1;
  if (d1.day   < d2.day  ) return -1;
  if (d1.day   > d2.day  ) return +1;
  return 0;
}
```

```c
int // assuming we can subtract without overflow, non-normalized results
Date_CompareForSameCal (Date d1, Date d2)
{
  assert (Date_HasSameCalAs (d1, d2));
  if (d1.year  != d2.year ) return d1.year  - d2.year;
  if (d1.month != d2.month) return d1.month - d2.month;
  return d1.day - d2.day;
}
```

2 Fonctions de conversion en dates et chaînes

```
bool
Cal_CharIsjulian (char cal) {
  return cal == 'j'
     ||  cal == 'J';
}

bool
Cal_CharIsGregorian (char cal) {
  return cal == 'g'
      || cal == 'G';
}

bool
Cal_CharIsValid (char cal)
{
  return Cal_CharIsGregorian (cal)
    ||   Cal_CharIsJulian    (cal);
}

Date Date_Dummy (void) { return Date_Make (0, 0, 0, false); }

Date
Date_FromString (char const text[])
{
  int day, month, year;
  char cal, extra;
  int nb_convs= sscanf (text, "%d/%d/%d:%c", & day, & month, & year, & cal, & extra);
  if (nb_convs != 4) return Date_Dummy();
  if (! Cal_CharIsValid (cal)) return Date_Dummy();
  return Date_Make (day, month, year, Cal_CharIsjulian (cal));
}
```

```
char Cal_ToChar (bool cal_is_julian) { return cal_is_julian ? 'j' : 'g'; }

void
Date_ToString (Date d, char text [], int capacity)
{
  char cal= Cal_ToChar (d.cal_is_julian);
  snprintf (text, capacity, "%d/%d/%d:%c", d.day, d.month, d.year, cal);
}
```

```
char
Comparison_Symbol (int comp)
{
  return (comp < 0) ? '<'
    :    (comp > 0) ? '>'
    :                 '=';
}

int
main (int argc, char * argv [])
{
  if (argc != 1+1) {
    fprintf (stderr, "usage:␣%s␣day/month/year:cal\n", argv [0]);
    return 1;
  }
  Date date= Date_FromString (argv [1]);
  Date epoch= Date_Epoch (date.cal_is_julian);
  int comp= Date_CompareForSameCal (epoch, date);

  fprintf (stdout, "%s␣", Date_ToStaticString (epoch));
  fprintf (stdout, "%c␣", Comparison_Symbol (comp));
  fprintf (stdout, "%s\n", Date_ToStaticString (date));
  return 0;
}
```

3 Contrôle des plages numériques

```
bool
Int_IsMultipleOf (int value, int factor)
{
  return value % factor == 0;
}

bool
JulianYear_IsLeap (int year)
{
  return Int_IsMultipleOf (year, 4);
}

bool
GregorianYear_IsLeap (int year)
{
  if ( ! Int_IsMultipleOf (year,   4)) return false;
  if ( ! Int_IsMultipleOf (year, 100)) return true;
  if ( ! Int_IsMultipleOf (year, 400)) return false;
  return true;
}
```

```
bool
Year_IsLeap (int year, bool cal_is_julian)
{
  return cal_is_julian ?
    JulianYear_IsLeap    (year) :
    GregorianYear_IsLeap (year);
}
int
Year_Length (int year, bool cal_is_julian)
{
  return Year_IsLeap (year, cal_is_julian) ? 366 : 365;
}
```

```
bool
Date_YearIsLeap (Date d)
{
  return Year_IsLeap (d.year, d.cal_is_julian);
}

int
Date_YearLength (Date d)
{
  return Year_Length (d.year, d.cal_is_julian);
}
```

```
bool Int_IsEven (int value) { return   Int_IsMultipleOf (value, 2); }
bool Int_IsOdd  (int value) { return ! Int_IsMultipleOf (value, 2); }

bool
Month_Has31Days (int month)
{
  return (month <= 7) ?
    Int_IsOdd  (month) :
    Int_IsEven (month);
}

int
Month_Length (int month, int year, bool cal_is_julian)
{
  if (Month_Has31Days (month)) return 31;
  if (month != 2) return 30;
  if (Year_IsLeap (year, cal_is_julian) return 29;
  return 28;
}
```

```
int
Date_MonthLength (Date d)
{
  return Month_Length (d.month, d.year, d.cal_is_julian);
}
```

```
bool
Int_IsBetween (int value, int min_value, int max_value)
{
  return min_value <= value && value <= max_value;
}

bool
Date_IsValid (Date d)
{
  return 1 <= d.year
    &&   Int_IsBetween (d.month, 1, 12)
    &&   Int_IsBetween (d.day  , 1, Date_MonthLength (d));
}
```

```
int
main (int argc, char * argv [])
{
  if (argc != 1+1) {
    fprintf (stderr, "usage: %s day/month/year:cal\n", argv [0]);
    return 1;
  }
  Date date= Date_FromString (argv [1]);
  fprintf (stdout, "date: %s, ", Date_ToStaticString (date));
  if (! Date_IsValid (date)) {
    fprintf (stdout, "invalid date\n");
    return 1;
  }
  fprintf (stdout, "year length: %d, ", Date_YearLength (date));
  fprintf (stdout, "month length: %d\n", Date_MonthLength (date));
  return 0;
}
```

4 Lendemain et veille

```
Date
Date_Add1Day (Date d)
{
  d.day++;
  if (d.day <= Date_MonthLength (d)) return d;
  d.day= 1; d.month++;
  if (d.month <= 12) return d;
  d.month= 1; d.year++;
  return d;
}
```

```
Date
Date_Sub1Day (Date d)
{
  d.day--;
  if (d.day >= 1) return d;
  d.month--;
  if (d.month <= 1) { d.year--; d.month= 12; }
  d.day= Date_MonthLength (d);
  return d;
}
```

```
int
main (int argc, char * argv [])
{
  if (argc != 1+1) {
    fprintf (stderr, "usage: %s day/month/year:cal\n", argv [0]);
    return 1;
  }
  Date date= Date_FromString (argv [1]);
  fprintf (stdout, "date: %s, ", Date_ToStaticString (date));
  if (! Date_IsValid (date)) {
    fprintf (stdout, "invalid date\n");
    return 1;
  }
  Date prev_date= Date_Sub1Day (date);
  Date next_date= Date_Add1Day (date);
  fprintf (stdout, "prev: %s, ", Date_ToStaticString (prev_date));
  fprintf (stdout, "next: %s\n", Date_ToStaticString (next_date));
  return 0;
}
```

5 Compte des jours écoulés et restants dans l'année

```
int /* day of date is excluded from count */
Date_YearElapsedDays (Date d)
{
  int count= 0;
  for (int month= 1; month < d.month; month++)
    count+= Month_Length (month, d.year, d.cal_is_julian);
  count+= d.day - 1;
  return count;
}
```

```
int /* day of date is included in count */
Date_YearRemainingDays (Date d)
{
  return Date_YearLength (d) - Date_YearElapsedDays (d);
}
```

```
int
main (int argc, char * argv [])
{
  if (argc != 1+1) {
    fprintf (stderr, "usage: %s day/month/year:cal\n", argv [0]);
    return 1;
  }
  Date date= Date_FromString (argv [1]);
  fprintf (stdout, "date: %s, ", Date_ToStaticString (date));
  if (! Date_IsValid (date)) {
    fprintf (stdout, "invalid date\n");
    return 1;
  }
  fprintf (stdout, "elpased: %d, ",   Date_YearElapsedDays   (date));
  fprintf (stdout, "remaining: %d\n", Date_YearRemainingDays (date));
  return 0;
}
```

6 Tampon

```
long /* days of older year are included, days of newer year are excluded */
Year_DaysBetween (int older_year, int newer_year, bool cal_is_julian)
{
  long count= 0;
  for (int year= older_year; year < newer_year; year++)
    count+= Year_Length (year, cal_is_julian);
  return count;
}
```

```
long /* older day is included, newer day is excluded */
Date_DaysBetweenForSameCal (Date older, Date newer)
{
  long count= Year_DaysBetween (older.year, newer.year, older.cal_is_julian);
  count += Date_YearElapsedDays (newer);
  count -= Date_YearElapsedDays (older);
  return count;
}
```

```
long
Date_Stamp (Date d)
{
  Date epoch= Date_Epoch (d.cal_is_julian);
  return Date_CompareForSameCal (epoch, d) < 0 ?
    +Date_DaysBetweenForSameCal (epoch, d) :
    -Date_DaysBetweenForSameCal (d, epoch);
}
```

```
int
main (int argc, char * argv [])
{
  if (argc != 1+1) {
    fprintf (stderr, "usage:␣%s␣day/month/year:cal\n", argv [0]);
    return 1;
  }
  Date date= Date_FromString (argv [1]);
  fprintf (stdout, "date:␣%s,␣", Date_ToStaticString (date));
  if (! Date_IsValid (date)) {
    fprintf (stdout, "invalid␣date\n");
    return 1;
  }
  fprintf (stdout, "stamp:␣%+ld\n", Date_Stamp (date));
  return 0;
}
```

7 Différence et comparaison

```
long /* order between d1 and d2 is arbitrary */
Date_Diff (Date d1, Date d2)
{
  long stamp1= Date_Stamp (d1);
  long stamp2= Date_Stamp (d2);
  return stamp1 - stamp2;
}
```

```
int
Stamp_Compare (long stamp1, long stamp2)
{
  return
    (stamp1 < stamp2) ? -1 :
    (stamp1 > stamp2) ? +1 : 0;
}
```

```
int
Date_Compare (Date d1, Date d2)
{
  return (d1.cal_is_julian == d2.cal_is_julian) ?
  Date_CompareForSameCal (d1, d2) :
    Stamp_Compare (Date_Stamp (d1), Date_Stamp (d2));
}
```

```
int
main (int argc, char * argv [])
{
  if (argc != 1+2) {
    fprintf (stderr, "usage:␣%s␣d1/m1/y1:cal1␣d2/m2/y2:cal2\n", argv [0]);
    return 1;
  }
  Date date1= Date_FromString (argv [1]);
  Date date2= Date_FromString (argv [2]);
  if (! Date_IsValid (date1) || !  Date_IsValid (date2)) {
    fprintf (stderr, "invalid␣dates\n");
    return 1;
  }
  int comp= Date_Compare (date1, date2);

  fprintf (stdout, "%s␣(epoch+%ld)␣",
           Date_ToStaticString (date1), Date_Stamp (date1));
  fprintf (stdout, "%c␣", Comparison_Symbol (comp));
  fprintf (stdout, "%s␣(epoch+%ld)\n",
           Date_ToStaticString (date2), Date_Stamp (date2));
  fprintf (stdout, "diff:␣%+ld\n", Date_Diff (date1, date2));
  return 0;
}
```