DCG_03

11th September 2006
The day before yesterday
Last Saturday
Christmas day
In Winter
In middle of July

1. Propose a representation formalism for dates and time intervals like the ones in these examples.

```
 (type_date, date)
 (longdate, (day,month,year))   // 11th September 2006
(relativeday, (modifierdate, weekday)) // last Friday
(spetial_day, name-day)            // Christmas day
(in, season)   // In Winter
(intervalm, month) // In middle of July
                                                  //(modified, month)  // Next January
```

2) Using the DCG formalism write a simple grammar for detecting in a sentence temporal expressions like these and representing them according to the representation system proposed in 1).

```
date (longdate, (Day,Month,Year)) -> day(Date) month (Month) year (Year)
                                                          // 11th September 2006
date -> relativeday                                      // tomorrow
date -> spetialday                                       // Christmas day
date -> in season                                        // In Winter
date ->  modifierelativeday relativeday                  // The day before yesterday
date ->  modifierdate weekday                            // Next Monday
date -> modifierdate month                               // Next January
date -> modifierdinterval month                          // In middle of July


relativeday ->today | tomorrow | yesterday
specialday -> Christmas day | Thanksgiving day
weekday -> Monday| Tuesday| Wednesday| Thursday| Friday | Saturday | Sunday
month -> January| February
season -> Autum| Winter

modifierdate  -> this | last | next
modifierelativeday ->  the day after | the day before
modifierinterval  -> in middle of| at the end of

others
//date ->  modifierday specialday   //Past Christmas Day
```

Propose a way of normalizing these temporal expressions.
Several examples

```
(day, month, year)
longdate (day,month, year) -- **normalized**(day, month, year))
relativeday (sem(Val))– actualdate (Date), Add( Date,Val,normalized(FinalDate))
                                                        // tomorrow(sem(1)), yesterday(sem(-1))
specialday (day,month)  -- actualdate (year) **normalized**(day, month, year))


modifierdate(Mod) weekday(Day) -- actualdate (Date),
if ((Mod ==this or Mod ==next) && daynumber(weekdate) > daynumber ( Date)){
        finaldaynumer = daynumber(weekdate) - daynumber ( actualdate);
        if(Mod == next) {finaldaynumer =finaldaynumer  +7;
                         actualizemonth(Date, finaldaynumer)   }}
                     }

//Monday  0| Tuesday-1| Wednesday-2| Thursday-3| Friday-4 | Saturday-5 | Sunday-6
```