First Programming Paradigms Miniproject

Marc Tom Thorgersen
October 28, 2016

.I Status

For the project i have used the *lang scheme* in DrRacket v 6.5.0.900 without the use of any libraries.

Now let us review the status of the program i have written, the following is a description of each of the functionalities we were asked to implement. They are listed in the order i decided to develop them, rather than the order they were listed in the assignment. As a sidenote, the way i represent calendars is explained in the overview section, in case you are in doubt of any way i refer to representations that should be explained there.

(appointments-overlap?): Status 100 % complete
: This function makes use of a helper function i constructed called Before? which takes 2 lists of 5 elements representing a date and time and evaluates whether the first list is before the other. It does this twice, once for start time of ap1 and end time of ap2, and again for start time of ap2 and end time ap1. This covers all possible points in which they can overlap, if two Before? calls return true, the appointments overlap.

(calendars-overlap?): Status 100 % complete
: This function makes use of the high-order function ormap, in order to produce the cartesian product of all appointments in the two calenders. This allows me to simply use the already constructed appointments-overlap

(flatten-calendar): Status 100 % complete
: For this function i simply construct a new list that represents a calender, when adding lists that represent appointments i recursively call the function on any lists that represent subcalenders, this is done by using map.

(find-appointments): Status 100 % complete
: I use the existing function filter, which takes a predicate and call that with the list containing the appointments.

(find-first-appointment cal pred): Status 100 % complete
: For this i use the former created function find-appointments, the list returned by this function i then sort by the time representation using a helper function called (starts-first?), and simply use car on the sorted list to obtain the first element, which then is the first appointment that adheres to the predicate.

(find-last-appointment cal pred): Status 100 % complete
: Almost exactly the same as (find-first-appointment) but i simply reverse the list before calling car.

(calendar-add-subcal) and (calendar-add-appointment): Status 100 % complete

For each way i represent a calendar element there is an appropriate construction function, for each of the these add functions i simply use append on the respective lists representing appointments and subcalendars.

(calendar-remove-subcal) and (calendar-remove-appointment): Status 100 % complete

Here i was slightly confused about how to do the removal, with either an index of element to remove, or a predicate, i chose predicate as this is a function a user may wish to call, and index refers to the internal representation, where predicate seems better suited for the external presentation, of course that is just my observation. These functions are called recursively on the list containing their respective representation with their given predicate.

(present-calendar-html): Status 75 % complete

While this function is not technically complete, it is possible to combine other functions to obtain the functionality which is why i believe it to be 75 % complete. For this i would use my existing function (display-calendar cal) and use (flatten-calendar) alongside (calendar-remove-app) on the cal input in order to remove any appointments that do not adhere to the from-time and to-time.

.II Overview

In order to represent the calendar i have the following structure(bottom-up):

Datetime This the lowest level list i use, it contains 5 elements representing; in order; year, month, day, hour, minute

Timeframe As an appointment will need a from time and a to time i use a list, containing two datetime lists to represent this, this list i call a timeframe.

Appointment An appointment is a list containing 2 elements, a string in order to describe the appointment, followed by a timeframe.

Calendar The top level, a calendar has 4 elements, the first two are strings, name and description respectively, the third element is a list of appointments and the last element is a list of calendars.

For each element in each list there are functions for both validation and extraction. the extraction functions are simply named the element they extract, e.g. year, month, timeframe, cal-descript, subcal etc. Furthermore each list representation has a constructor function which have been perpended with "build-". The validation functions have the ? as a postfix, they simply use pattern matching to determine whether the given list is correctly structured, e.g. that a datetime has 5 numerals. As for

the validation and each data representation has an appropriate "with-<representation>" function. These use the validation functions and take their respective data representation and a function as parameters. Their purpose is to be type checkers, this is due to the dynamic typing feature provided, which will be discussed further later. the function (find-day) uses its helpers to determine which day of the week

it is, by a date given purely in numbers. It is based on a formula that helps one determine the day purely by date and works on the gregorian calendar for dates after it's introduction in 1583. Its sole purpose for the system is to provide the day of the week as a string to be used in the html representation output. the HTML generation

part of the program uses a lot of small functions each representing html tag. The (display-calendar) function calls display functions for each element in the list, these calls nest down to the lowest level which is (display-datetime).

Reflections

I.I Functional relative to Imperative and Object Oriented programming

This course is my first interaction with a functional programming language and the functional paradigm so the train of thought while programming as well as approach. As such i feel that while i have successfully produced a calendar language extension providing the ability to output a calendar to html, i would have been able to do so significantly faster in a language such as C or Java, although this is purely by experience. Since starting this mini-project the work i have done have been very exponential, i started out spending a load of time on doing the small stuff, once i had implemented my with-<data representation> functions to use for type checking everything became much more simple. Not because of the type checking it added, but because it forced me to use lambda, and as such gave me a much better understanding of the functional paradigm, or rather the mindset one must be in when programming functionally.

The biggest difference i felt when going from imperative and object oriented to functional is moving to a stateless system. The fact that there are no side-effects is particularly useful when testing allowing me to use the same source input for my tests, knowing that regardless of what i do in a given function, the next time i use it nothing will have changed.

The second big difference i have found is the emphasis on recursion as well as helper functions. While helper functions are nothing new, the syntax in scheme has had me made them from the get go, rather than going back and refactoring my code, which is what often happens in OOP. As for recursion it is mostly a matter of changing ones train of thought from thinking in for-each and for loops to considering recursion in combination with helper functions, at least that was how i "translated" it.

There are also some minor general differences such as how mathematical it feels with the data flow, the use of high order functions and the referential transparency. Another change for me is working with a dynamically typed language, i have mentioned this shortly and will describe this further in the section named dynamic typing.

I.II Strength and Weaknesses

For this, let us first start with the strengths of the language.

Mathematics The mathematical flow of data makes the data flow very easy to follow. Furthermore with there being no side effects and it being stateless simulates mathematics very well, 2 is always 2 and can never be anything else, as such functional programming seems a perfect fit if the program is mathematical.

Performance When we compare performance often we compare to C, however a properly written program in the functional paradigm can rival a properly written C program, so when it comes to performance functional programming is up there.

Pure functions As mentioned functions are free of side effects as a result of functions being pure. Pure functions is a whole topic in itself that i will not discuss further, however it does provide some significant advantages, particularly when considering concurrency and parallelism. Furthermore pure functions provide the advantageous side-effect of referential transparency which can be utilized for caching optimizations. Pure functions also make high-order functions even more powerful, as you know there will be no side-effects.

Free of overhead Contrary to the Object Oriented Paradigm i felt that a lot of overhead was gone, when developing in Java or C, several days can be spent on just readying a build environment with the proper technology stack. That is of cause not the case for smaller projects like developing a calendar for which you can get right into it, but for larger projects overhead is a big part of object oriented programming, and some even trickles down to smaller projects like importing libraries etc. This does also come with its disadvantage however, as we will cover next in the weaknesses section

Now we will move on to the weaknesses before summing up the major points.

Libraries Let us continue where we left off. While a huge overhead can be an annoyance it is also quite useful. For large object oriented projects i would simply import a calendar from somewhere, that would defeat the point of this miniproject but when it comes to library support the functional paradigm is behind. This is not because it cannot follow, it is simply a result of it being the less used. With the emphasis on reuse, not only in a given program but in software development in general, using a largely supported language does have the advantage in this area.

Compatibility As a follow up to libraries, using resources or components written in other languages than the one you are writing in, is not uncommon. For this it is simply easier to adapt from similar paradigms, rather than a restrictive paradigm such as functional, i refer here mainly to the purity of functions and immutability of variables.

Portability Once again this is not actually a fault of the paradigm itself, but rather the software development world. Compared to a language like C, Scheme, SML, ML, ErLang and others simply are not as widespread, and as a result, not as

supported. This means that if your system is written in C it is more likely to be applicable on an arbitrary system.

Overall the functional paradigm has its place, as does any established paradigm. There is a reason that other languages adapt it, a variety of multiparadigm languages such as F, Python and C all allow to mimic the functional paradigm so it obviously has its strengths. However, it also has its weaknesses, which is why multiparadigm languages allows for it to be mimicked where it fits, and use other paradigms for scenarios where it does not. In the end it all comes down to using the proper tools, and for some tasks functional programming is the right tool, and for other it is not.

I.III Dynamic Typing

Personally i am much more familiar with static typing, something which is reflected in my program. Research has shown that the difference static and dynamic typing, when it comes to time spend programming, is negligible since the time is simply allocated differently but and an equal amount; as such i will not go into details about which I believe is better.

As mentioned the real difference comes down to allocation, for dynamic typing it is a lot of type checking, and for static it is a huge overhaul if you did not make a proper type system from the beginning. This is also evident in my program, for which i designed a (with-<data representation>) for each data representation. The purpose for this was simply to "fake" a type system as i after the implementation called it as the first thing in every function i created. Personally I prefer static typing, but neither is really weaker or stronger than the other.

I.IV Sources of Inspiration

The bulk of inspiration for me have been looking up functions in the Racket Documentation. Furthermore a group member gave me an idea of how i could make html output into small functions for each tag and as a result they may look similar.