PP: Scheme Mini Project

NAME: Alexander Brandborg
STUDENT NUMBER: 20136225
STUDENT MAIL: abran13@student.aau.dk

STATUS: Program lives fully up to requirements and is runnable
SCHEME SYSTEM: R5RS

Overview

The program includes functions to set up a calendar, appointments and points in time. In addition there are functions, which can be applied to the calendar and its constituents. The program is split into five sections, Course Functions, Constructors & Accessors, Calendar Functions, HTML Functions and Tests.

The Course Functions section contains course material used for the program. This includes a handful of higher order functions as well as a sorting function.

The Constructors & Accessors section contains all functions needed to create assignment specific objects and access their parts. Given input, a constructor will evaluate to the internal representation of the given object. The internal representation of a calendar is an associative list. That is, a list in which each element is a pair. In a pair the first element is a name, and the other is a value. This corresponds to the dictionary data structure with key-value pairs. A calendar contains two pairs, one for its list of appointments and one for its list of sub-calendars. These are both accessed by name through an accessor function “access-calendar”. Similar to a calendar, an appointment's internal structure is also given by an associative list, and it is accessed with its own acessor function “access-appointment”. Time is represented as a number rather than an associative list. This was done to make it easier to compare time objects with each other. Thus two time objects can simply be compared using arithmetic comparison operators. An accessor function “access-time” has also been written for this object.

The Calendar Functions section contains most of the requested calendar functionality. The first few functions are used to add/delete appointments and sub-calendars from a calendar. One interesting thing to note is that deletion occurs by the use of a predicate. This gives the function user a lot of freedom to specify what elements should be deleted, while I as a programmer just need to write a simple deletion function. A handful of the functions in this section are about time comparison and overlap. Because of the internal representation of time these have been rather simple to write. Such a function “app-time-less?” is also used to sort appointments chronologically by start time, by being passed to the sort-list function. The “flatten-Calendar” function benefits from the internal representation of a calendar object. The problem is approached recursively, by first flattening all a calendar's sub-calendars before flattening the calendar itself.

The HTML Functions section contains functionality to create an external representation of a calendar in the form an HTML string. This is done by an application of “present-calendar-html”, which enacts a handful of helper functions. The HTML represents part of a calendar in a time period defined by from-time and to-time. The functionality designed works as follows: It sorts the list of all appointments, which appear within the defined period of time. Then for each year month and day in that period places an html header. If an appointment appears for a given day, places a block under that day's header to describe it. If a day has no appointments writes the line “Nothing today!”. Using the “get-next-day” function I make sure to get the right amount of days for each month that is run through, also accounting for leap years. An extra functionality I designed handles appointments, which span several days. In the naive implementation such an appointment would just be placed in the calendar on the day it starts. To get around this, I replace such an appointment with a new appointment, for every day which is spanned. This ensures that the temporal length of an appointment is accurately visualized in the calendar. This functionality is encapsulated in the “multiple-day-apps” function.

The Test section defines a few time,appointment and calendar objects. It also has some uncommented functions-applications using these objects, which show off the program functionality. The string produced by (present-calendar-html (flatten-calendar code-calendar) t1 t2)” can be found in the calendar.html file. It can be loaded into a browser.

Reflections

In this section, I will reflect upon the functional paradigm, which was used to implement the assignment. This will be discussed in the following subsections.

My use of functional programming relative to experiences with imperative and object-oriented programming

I have had no previous experience with functional programming, except for lambda expressions in C# and some functional elements in python. So my background is very much one in the imperative camp. Here I will use the term imperative to refer to both imperative and object-oriented programming. This is because I have yet to work with a pure object-oriented language. Languages like C#, Java and Python, which are labeled as object-oriented, are more multi-paradigm languages, heavily based on the imperative paradigm with elements from the object-oriented world thrown in.

What I notice a lot in my use of functional programming is how small my functions tend to be, as I more often split up my functionality. This is also encouraged in imperative programming, however my functions tend to grow large there, having many different execution paths running through them. I think this is because a large function in a functional language is very difficult to understand. This can in part be blamed on some of the syntax, which is heavily inspired by mathematical functions. But syntax aside, a programmer can only look at so many function-applications nested within function-applications before his understanding of the code shatters. This could be looked upon as a negative, as having to frequently declare new functions can be bothersome. However I do find the end result a lot easier to comprehend. I will try to import this discipline into my imperative work in the future.

Use of recursion is also a lot more central to my functional programming than my imperative. This is of course because you can not have traditional loops in functional, as those require a state to be used and updated. When writing imperatively I do not feel, as if I can be bothered to write a recursive function. Especially when I have to run through some data structure, as that really depends on the structure at hand. Yet, I feel that functional programming supports recursive traversal of collections very well, as every collection of data is a simple list with the associated car and cdr functions. Because of this uniform structure, you always traverse data in the same manner, freeing up focus to think about what the actual base and recursive cases should do. You could do something similar in imperative programming, but that often requires indexing, which opens up a whole realm of possible programming errors. Yet, even when I have to use recursive functions for something else than collection traversal, such as writing factorial, I find it easier to do in the functional paradigm. Again this is because functions tend to be shorter, and it is easier to comprehend the recursion. I would say that my use and understanding of recursion has become better working in the functional paradigm. But I am not sure that it would be my goto way of traversing object collections in imperative programming. What I will definitely bring along is the notion of tail-recursion. Saving stack space can be very beneficial also in imperative programming, when a large amount of iteration is going on. This is a benefit you do not get form foreach.

I also find myself using higher order functions a lot more. In functional programming, functions are treated like any other sort of data and can freely be passed around. This I see as a great benefit. You could do this in something like C through the use of pointers, but that becomes very difficult to comprehend and use. However python does allow for functions to be passed around in a similar manner. Earlier the usefulness of higher order function were not apparent to me. Yet, I think I will use them more often in my python programming from now on.

Overall my use of functional programming here differs a lot from my regular imperative style. While I may not visit the functional world

often, I can use some of its ideas and disciplines in the imperative world.

Evaluation of strength and weaknesses of the functional paradigm in relation to the calendar assignment.

When I first viewed the assignment, I thought that it looked very fitting for an object-oriented approach. A calendar could be an object with associated methods. So could appointment and time. A lot of the functionality could be encapsulated into a few very central elements. Modeling the real world, I think, feels more natural to do using classes and objects rather than functions. I wish I could associate some of my functions with the internal representations of my objects. Right now these functions can be used in ways they were not meant to. An unfamiliar programmer could try to use the “access-calendar” function to access a non-calendar object. This leads the unsuspecting programmer into a world of hurt i.e. runtime errors. If they could be encapsulated in an object like a method, it would be easier to protect both the data and the programmer.

A benefit of not having my functionality encapsulated within objects, is that it is very easy to unit test. An object method can be difficult to unit test. This is because a method's output may not only depend on input, but also the state of the object. As there is no state in functional programming, it is a lot easier to test individual functions. You could of course also write object methods that do not use the object state, but that does take some discipline and can be seen as cumbersome. In functional programming you are forced to write your functions as complete deterministic entities. I have not unit tested my program as is, but I feel it would be easier to do in this form, rather than if I had used an object-oriented approach.

Talking about the state, I was actually surprised at how little I missed it, while writing the assignment. Again, without a state you can get no function side-effects and the program overall becomes more deterministic. This deterministic property is very attractive. Yet, in reality we are trying to think functionally on a machine that thinks imperatively. A computer executes a number of statements in sequence. Depending on the statements and the sequence the physical state of main- and secondary memory is updated. This is something that we can not get around. So when interacting with the computer more directly, we need to think about the state. This is the case for my “write-html” function, as it needs to update the program state behind the scenes, to remember the port which is being written to. In fact, just writing to a file could be seen as updating the state. So while the statelessness of the functional paradigm is attractive, I do not wish to be stuck with it, when working with a machine that does not follow the same mindset. Yet, if we only need to update the state a few times, it should be easy to keep the eventual side-effects in mind. Sometimes, as in this assignment, updating the state is not needed often. But I feel that trying to implement something like a dynamic algorithm in a functional way could prove challenging, as it is all about updating the state to benefit efficiency.

One of the weaknesses with not having a state is also that a functional program probably uses up more space and resources than an imperative one. With an imperatively implemented calendar, we would simply extend the list of appointments, if needed. With functional we are creating an entirely new calendar object. This is going to take resources, both in the frequent instantiation and the need for garbage collection. I am very happy with the garbage collection found in scheme, cause I would want to do functional programming without it! But I figure this is the price to pay, when programming without a state.

In short I see both positives and negatives with the functional paradigm. Like imperative, it is a tool that fits well in some instances, while at other times another tool needs to be used.

Dynamic typing contra static typing
I have actually been using python, a dynamically typed language, for a long time before working functionally with scheme. In both instances I notice the benefits and downfalls of dynamic typing. The downfalls especially became aware to me, during this assignment. Very much because I did not have a debugger handy, to help me find all the type errors I was getting.

The idea with dynamic typing is that we do not have to declare our variables and their type before using them. A variable gets the type of the value assigned to it. This means that we do not have to think about the type of parameters, when declaring a function, nor the type returned by that function. This makes code a lot faster to write. It also allows us to be more flexible, as we can send many different data types to the same function, which it can then choose to handle or not. Thus we do not need to write a lot of type specific versions of the same function.

However, with dynamic typing I spend more time on debugging after compilation. I remember back in C, if your program compiled you were relatively sure that you were not going to see a lot of type errors. Here, compilation does not give you this sort of safety, and you are going to find your faults at runtime instead. What is dangerous is that you may not trace through every logical path of your program, and type errors are left undiscovered until a later point in time. Because of this, I perceive my program to be less stable than it probably is. You need to test and debug more in order to get the same type of safety as you would with static typing.

The idea that you can think less about types because you are working with a dynamically typed language does not hold true either. Yes, you do not need to declare types, but to not get runtime errors you have to think about what type of input, you are giving to each function. Because the compiler does not warn you, you have to be more aware of types.

Yet, I still feel that dynamic typing can be attractive, it just needs more support. In python we have dynamic dispatch, which means that the implementation of an operation is decided upon at runtime. Thus we can create a function that takes an object and calls a method on it. If the object has a method by that name, it will run. Dynamic typing supports this, by not limiting what types of objects are passed to the function. So I feel that while static typing gives us a lot of benefits, dynamic allows us to be really flexible, if we add other features to the mix. If a programmer can comprehend the entire “type space”, which he is working with, dynamic typing can be a powerful tool in the right context.

Sources of inspiration
I have used no source of inspiration, other than the methods presented in the course. These have been marked out in the program.