Software Development Part 1 – Lab Session 3B
Algorithms and Problem Solving

Objectives

By the end of this session you should:

• be aware of basic approaches for modelling software development problems as algorithms

• be able to analyse a simple problem and propose an algorithmic solution

• describe a simple algorithm using a flowchart

Introduction

In a previous lecture we looked at the software development life-cycle and basic approaches for modelling an algorithm as a flowchart and pseudocode. In today's lab we will look at how to take a given problem description, analyse it, describe constraints and assumptions and model an algorithmic solution as a flowchart.

During or after the lab session you may wish to do some additional reading into Flowcharts and Algorithms and their use for modelling problem solutions. Some starting points for reading are at the end of this worksheet, but these are just starting points. You should investigate different sources of information and find new ways to search for informative documents, websites and materials.

A simple problem

Imagine we are asked to write an algorithm to control a set of traffic lights at a road junction. The lights must cycle through the correct sequence to indicate to drivers whether or not they may cross the junction. For now we will consider **just one road through** the junction, ignoring other roads and any pedestrian crossing lights – a hefty constraint!

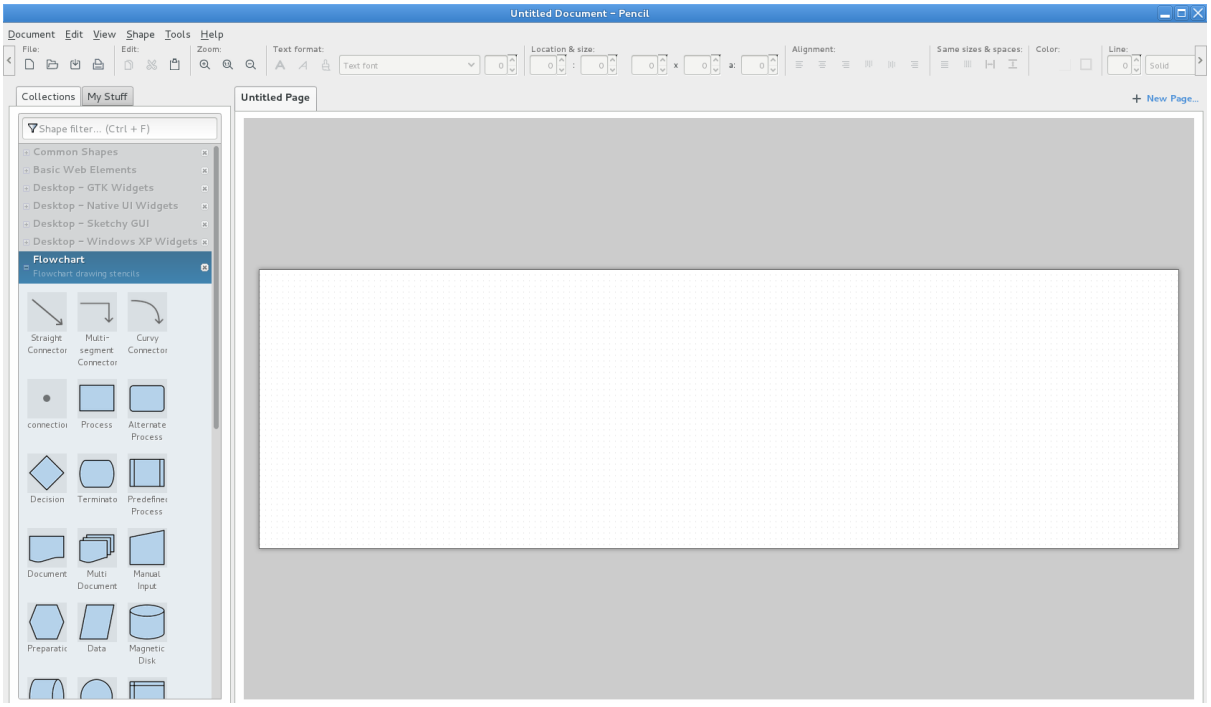
We can begin by defining the problem in words. We can set some assumptions, such as the lights will start at red, and the fact that the physical lights can easily be controlled by our software.

"The algorithm begins with the lights at red. The algorithm should wait 5 seconds, then turn on the yellow light. It should then turn off the red and yellow light and turn on the green light. It should wait 20 seconds and then turn off the green light, also turning on the yellow light. It should wait 5 seconds and then turn off the yellow light, also turning on the red light. It should wait 10 seconds and then repeat the steps above."

This written description serves as a basic starting point for defining an algorithm and eventually creating program code to control hardware. This description can highlight questions about the problem we're trying to solve and failures of the proposed solution. It can also highlight and help us define any assumptions:

- In this algorithm we assume a fixed set of timings for the light sequence
- We assume that hardware will never fail
- We assume only one road (for now)
- Did we forget to include how long the red and yellow lights should be on for near the start?

Let's begin by creating a simple flowchart for our algorithm. In the CentOS menu system, under **Applications -> Programming** you can find a program called **Pencil**. Open it now. The welcome screen will look something like this:



On the right you will see a sidebar with various collections of drawing tools. Later in the session you can take some time to explore these collections and see what else this software is capable of. For now, either collapse (with the minus icon) the collections, or scroll down, until you see the **Flowcharts** collection.

Here you will find a collection of icons that you can use to build a flowchart for our problem.

First right click on the white space in the centre of the screen and choose Resize Canvas -> Fit Screen to make the canvas larger.

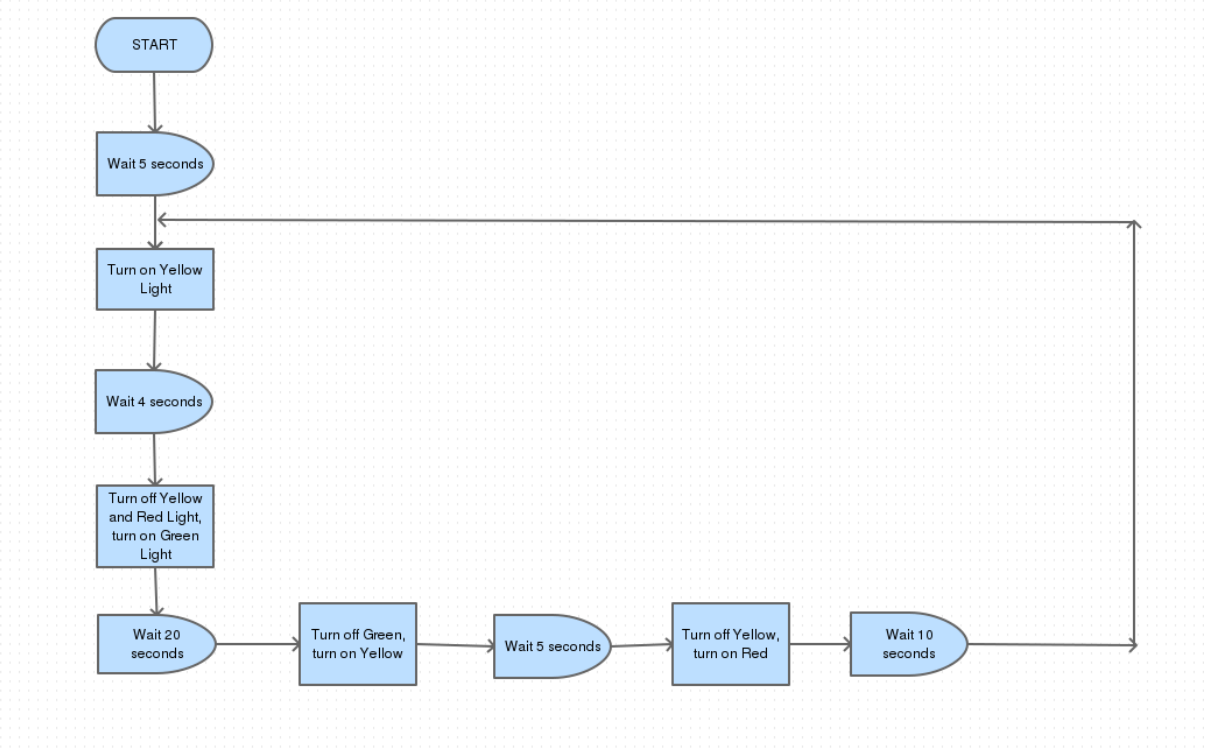
Now to build our flowchart. Begin by dragging a Terminator from the sidebar to the Canvas. Place it somewhere near the top. Double click the Terminator you just placed and type START. We now have the defined beginning of our algorithm!

Our description has us starting with a delay of 5 seconds, so we need to add this step to our flowchart. Formally there is a specific shape used for indicating a delay or pause in an algorithm flowchart, and we may as well use it here. Find the Delay icon and drag it to the canvas, resizing it once there if you like. Double click it and add a label indicating that a pause of five seconds is needed.

Now we must connect our two icons. Drag a Straight Connector to the canvas and connect the two icons – you should find it “snaps” to the icons at set points. The direction of the arrow shows how we move through the algorithm, so ensure it is the right way around.

Next our description states that the yellow light should be turned on. For this we will use a simple and common icon, a Process. A Process describes a step in an algorithm and can give a simple label to a complex operation, allowing us to take a high level view of our algorithm before implementation. Find the Process icon and drag it to the canvas, connecting it to the one we just added. Add an appropriate label again by double clicking.

Continuing with this, making changes to our plan where needed, we can build up a flowchart showing our algorithm, including the fact that we repeat these steps indefinitely. We can use multiple straight connectors to connect the last delay to a point further up in our algorithm, or you may like to try the other types of connectors.



With our first algorithm flowchart complete we can assess it, ensuring it matches our plan, and we can run through the algorithm to test that it meets our expectations and requirements.

Complete the flowchart now and run through it once, keeping track of the light colours at any given moment in time to see if it works as we expected. You can use pencil and paper, or a simple text document to hold the status of the lights, modifying them as Processes in our algorithm occur.

Once you complete this save the document and try also exporting it as an image, saving it in an appropriate place.

A more complex problem

Let's now try a slightly more complicated problem.

Imagine we have to work through a list of numbers provided as an input into our algorithm. The numbers represent amounts of money going into and out of a company bank account. The list of numbers will include both positive and negative values representing money coming in (positive) and money going out (negative).

We must develop an algorithm to count the number of times money was taken out of the account. Our algorithm this time will have an end point and an **output value.**

Let's begin be trying to analyse the problem and create a short description of a proposed algorithm.

Firstly, the problem has some vagueness to it. It doesn't state how many numbers will be in the list. It also doesn't state minimum and maximum values for items in the list. We can make assumptions about these facts, documenting them appropriately and perhaps even seek further information on the problem before we start. Let's assume for now that there could be any number of values in the list (zero to infinity) and the values could be of any size, representing any amount of money. We'll see later if this is actually important or not for our algorithm, though we should definitely consider this if we write code to implement the algorithm!

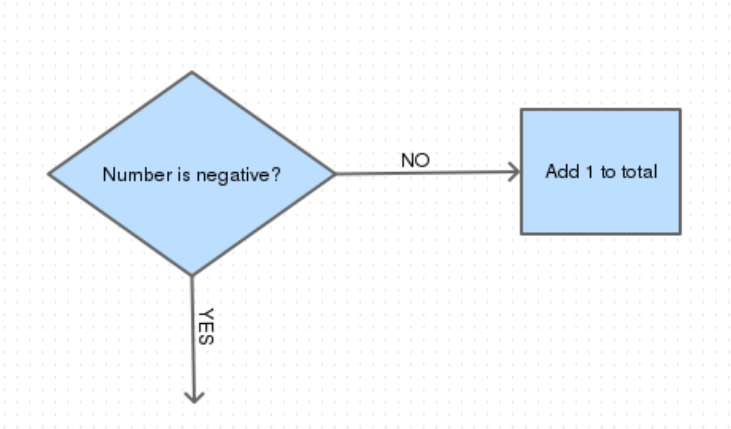
A basic description of an algorithm to solve this problem might be:

“Begin by importing the list of numbers from the source. Create a 'total' to use for counting the negative numbers in the list, setting it to zero initially. Check if there is a number in the list. If so, check if it is positive or negative. If negative add one to the current value of 'total'. Check if there are more numbers in the list, repeating the step(s) above if there are. Return the value of 'total' if the list is finished.”

Assess this description and see if you are happy with it as an initial plan for the algorithm. Make changes if you like, or if you spot any errors.

Try now to develop the flowchart for this algorithm using the same approach as before. This time though you may also need to include a few additional types of icons. A Terminator should be used to signal the end of the algorithm. We also have the notion of a decision (or two) in our algorithm. For these parts a Decision icon should be used with a label that indicated what the decision is. For decisions there should be multiple outgoing connections to show what should happen in different cases.

For example in the algorithm proposed we add one to 'total' if the number we're currently looking at is negative, but doing nothing if it's positive. Connectors can be labelled to show the different decisions by double clicking them. That might look something like this:



Note that this would need to be connected appropriately to the larger flowchart.

Other icons can be used such as the Data icon to indicate that we bring in the list from another source, and even the Display icon if you think we should display a value (total perhaps?) to a user.

- Try now to complete the diagram according to the description above, or your own if you felt it needed modifying. Don't forget to include a Data icon indicating that the final value of total is passed somewhere outside of this algorithm.

Exercise 1

Try to develop an algorithm, including a flowchart representation, for the following problem. Spend some time analysing the problem and don't be afraid to refine your design as you develop it.

You must create an algorithm to work through a standard deck of playing cards (https://en.wikipedia.org/wiki/Standard_52-card_deck) and create two new decks of cards, one with all the number cards (A-10) and another with just the King, Queen and Jack cards.

Exercise 2

Read the following problem and develop a high level description of an algorithm to solve it. Model your solution as a flowchart and test it by creating a few small test cases.

Each morning pallets containing three differently sized boxes are delivered to a warehouse. Each of the boxes must be stacked on top of each other in a single pile. The largest box must go on the bottom and the smallest must go on the top. The boxes may come off the pallet in any order.

Develop your solution and make up a few sets of boxes with different sizes. Test your algorithm using these sets.

- Are there any problems with your analysis of the problem and your solution?
- If so how can you address them?

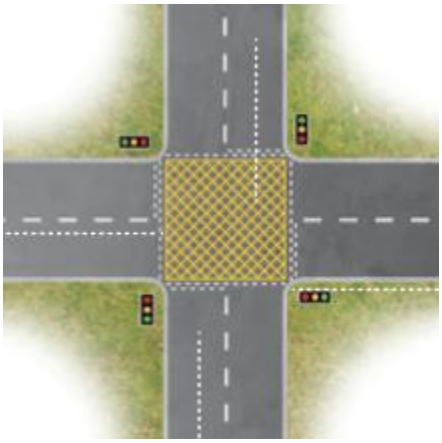
Exercise 2a

Extend your solution above so that it would work for a pallet with any number of boxes on it.

- How much complexity does this add to your algorithm?
- Which parts can be re-used or modified?

Exercise 3

Return to your solution from the first part of this lab where we built an algorithm and flowchart for a simple traffic junction. In this exercise we will add additional parts to the algorithm to incorporate a second road to the junction:



Begin by defining a short written description of the problem, thinking carefully about the rules of which lights should be green/red at which times, and the timings of the changes. For example you should think about how long each road should have for green; how long after one road gets a red light the other gets a green light etc.

Copy and modify your flowchart from the first exercise to add these new details and test your algorithm by running through one or two cycles of lights.

Exercise 3a

Adding pedestrian crossings to the junction – what problems does this add to the situation? Expand your solution to incorporate pedestrians possibly requesting a green light for them to cross while road traffic is held at red lights. Where in your algorithm is an appropriate time to check for a pedestrian crossing request? What must be added to your flowchart to cover this?

Exercise 4

Devise a moderately simple problem from your own interests that you think could be solved by creating an algorithm. Create a problem description in written form. Attempt to create a solution for your problem and an algorithm modelled as a flowchart.

- What information (if any) is missing or vague from your problem description?
- What assumption or constraints must your algorithm define to address these?
- Discuss your problem and solution with a module staff member or a colleague (quietly!); what do they think of your algorithm and do they see any vague or incorrect aspects of it?

Links for further reading and software use:

Algorithms:

- https://en.wikipedia.org/wiki/Algorithm (wikipedia can be an acceptable **starting point!)**
- http://math.hws.edu/javanotes/c3/s2.html
- http://sofia.cs.vt.edu/cs1114-ebooklet/chapter4.html

Flowcharts:

- https://en.wikipedia.org/wiki/Flowchart
- https://xkcd.com/518/ (just for fun)
- http://www.breezetree.com/article-excel-flowchart-shapes.htm
- https://support.office.com/en-us/article/Create-a-basic-flowchart-f8e57ca2-0c24-4760-bc2e-8812d7310c6a (flowcharts for Microsoft Visio)