COP 3503

Programming Fundamentals for CIS Majors II

Spring 2015

Analysis and Design of Group Project

March 20, 2015

Dennis Alas

Sarah Anderson

Grace Friedman

Matthew Griessler

Michelle Palumbo

Valentina Rendon Duque

**Definition:**

* System Definition:

Gator Chart is a computer program that allows users to use shape objects as well as lines to create useful flowcharts. Use cases for flow charts are myriad and include:

* + Explaining programs, or subsections of programs such as algorithms

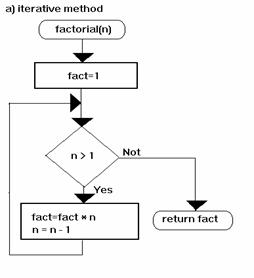


Figure 1 - Factorial Algorithm

* + Many manuals, such as those used in the space shuttle, have processes that are represented with a flow chart

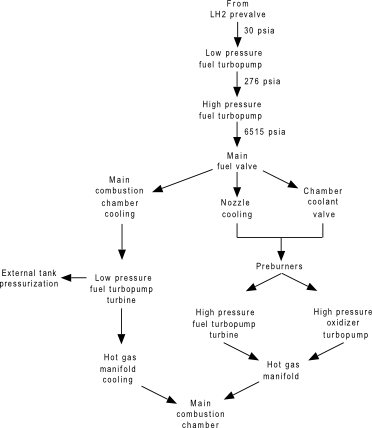
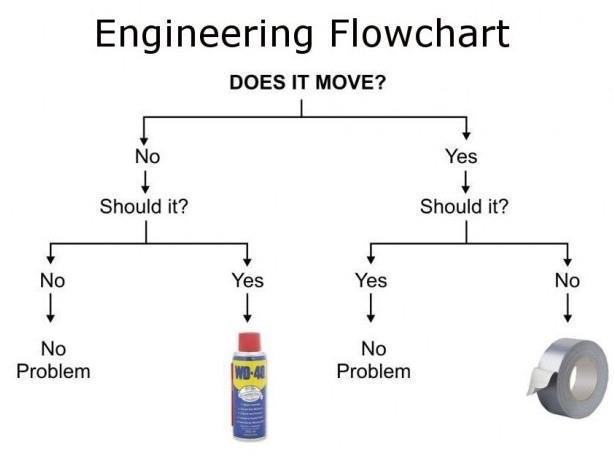


Figure 2 - Shuttle Main Engine Oxidization

* + Choice of an option from a field of options, the flowchart can direct you to the correct choice by answering a series of questions 
  + Modeling real life processes, for example, a traffic light controller
* Importance of System:

It is often easier to understand a complicated idea graphically, flowcharts enable understanding. For example, in an industry in which new, different programs are constantly created to solve problems for customers, a graphical representation that crosses fields of knowledge is important for the programmer, fellow personnel that may be part of a large team, as well as the user, to assure and understand if and how their needs are been fulfilled. The system also clarifies what the user provides to the system (what the inputs will be) and what the program with its multiple tools has to offer (what the outputs will be) in order to fully utilize the product.

Gator Chart has several advantages over the competition, which includes Microsoft Visio, and yEd. Gator Chart is simple to use and single minded in purpose, it is used to create flowcharts. Gator Chart was born out of frustration with the complicated nature of other flowchart software used to make flowcharts for classes.

**Analysis:**

Actors: Actors, as defined by the lectures, are the objects present in the system. Therefore, we shall examine every object and its inputs, outputs,

* Inputs to the System:

The objects presented by the system are:

Lines: logically connect inputs by pointing from one object to another to create flow and show how inputs act with each other.

Parallelogram: represent input data.

Oval: indicates the starting and ending points of a process.

Rhombus: indicates a point where the outcome of a decision dictates the next step. There can be multiple outputs, but usually it consist of just two – yes and no.

Rectangle: represents a step in the process.

Square: indicates a set of steps that combine to create a sub-process that is defined elsewhere, often on another page in the same drawing.

Trapezoid: represents an operation or adjustment to process that can only be made manually.

* Outputs of the System:

Parallelogram: also represents output data.

* Flow/Logic Required for the System:

Constraints: users can only determine the place where the objects will be located, their size (given a maximum), and whether the square and rhombus objects are solid since in logic solid rhombus and squares are used.

Assumptions: the user knows the basic use of shapes to create a flowchart. The user will primarily use the system to create logic flowchart, noting that the same symbols may be interpreted with differently in other industries.

Modifications: ???

Relationships and Effects: ???

Design:

Modules

* Model – the model contains the data. It inherits from the QGraphicsScene (a Qt widget) which enables the drawing of 2D shapes. It maintains a list of all currently drawn shapes
  + Shape Class, contains properties shared by all shapes, location, size. It inherits from QGraphicsItem (a Qt widget) that represents a 2D shape
    - Rectangle Class – inherits from Shape class and knows how to draw a rectangle
    - Trapezoid Class – inherits from Shape class and knows how to draw a trapezoid
    - Parallogram Class – inherits from Shape class and knows how to draw a parallelogram
    - Rhombus Class – inherits from Shape class and knows how to draw a rhombus
    - Line Class – inherits from Shape class and knows how to draw a line
* Main Window – the view allows the user to interact with the data. It initializes the entire application. It creates a view, and a model, and all the other windows that you see
  + Editor “the View” – This is where the user creates a flowchart. The editor inherits from QGraphicsView which is attached to the model. It shows all of the shapes on the screen. It allows you to zoom in and out and pan around so you can view the entirety of your flow chart
  + Menu bar Class – is the File, edit, Help bar at the top of the application
  + Item List Class – lists all of the shapes you can use in the flow chart and allows the user to drag and drop them into the editor
  + Properties List Class – allows the user to edit the properties of selected shapes such as width, height, color, fill pattern, and text