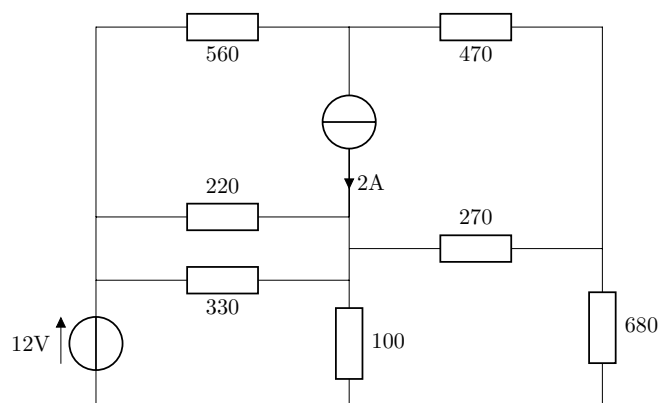


C5

Data Structures & Dynamic Memory Allocation

C has a number of features to aid in breaking up the code into smaller modules with clearly defined interfaces. One such feature is the ability to create data structures which enable you to create your own more complex variables. A challenge when writing programs is often the size of an object is not known until runtime. The only way to handle this robustly is to dynamically allocate the memory. You will use these techniques to develop a circuit simulator for DC analysis of passive circuits.



Schedule

Preparation Time : 3 hours

Lab Time : 3 hours

Items required

Tools :

Components :

Equipment :

Software : gcc, text editor (e.g. notepad++)

Version: October 25, 2020

©2012


Steve R. Gunn

Electronics and Computer Science

University of Southampton

Before the laboratory you should read through this document and complete the preparatory tasks detailed in section [2](#).

Academic Integrity – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

You will be working on your own in this laboratory. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the  symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

For each Task you should create a new directory so that you have a working version of every program at the end of the lab. Remember to place comments in your code.

You will be marked using the **standard laboratory marking scheme**.

Notation

This document uses the following conventions:



An entry should be made in your logbook



Care should be exercised

Remarkable text

A point of note

`command input`

Command to be entered at the command line

`output response`

Response produced at the command line

1 Introduction

This lab introduces you to programming with data structures and dynamic memory management which are powerful and fundamental concepts in C programming. In addition you will learn how to compile projects with multiple source files and read information into the program from the command line.








You will develop vector and matrix objects which can be dynamically sized at runtime. You will use these to underpin a circuit simulator which uses modified nodal analysis to calculate the voltages in a passive resistive circuit with multiple voltage and current sources.

1.1 Outcomes

At the end of the exercise you should be able to:

- ▶ Implement data structures in C.
- ▶ Implement dynamic memory allocation in C to enable objects to be sized at runtime.
- ▶ Build a project with multiple source files.
- ▶ Use library functions provided.
- ▶ Pass arguments from the command line into your programs.

2 Preparation

1. Describe the concepts of a `struct` and an `enum` in C. 
2. By inspecting the source code in `vector.c` determine the appropriate data structure definition to go into `vector.h`. 
3. By inspecting the source code in `matrix.c` determine the appropriate data structure definition to go into `matrix.h`. 
4. Describe the operation of the `malloc` and `calloc` functions, commenting on any differences. 
5. How can you give back memory that you have dynamically allocated? 
6. How do you compile a project with multiple source files? 
7. How can arguments be passed from the command line into your program? 

3 Laboratory Work

Create a new directory ELEC1201/Labs/C5 to store your files for the laboratory. For each task you should create a new sub-directory so that you have a working version of every program at the end of the lab.

3.1 Vectors

1. Take your vector data structure code from the preparation and populate the file `vector.h` to complete the vector header file.
2. Implement the `createVector` and `destroyVector` functions in the implementation file `vector.c` to dynamically allocate a vector and to free up the memory when it is no longer required.
3. Write a main program which calls the `createVectorFromFile` and `printVector` functions to read and display a vector.
4. Test your program to verify it is able to display the `example1.vec` file provided.



3.2 Matrices

1. Take your matrix data structure code from the preparation and populate the file `matrix.h` to complete the matrix header file.
2. Implement the `createMatrix` and `destroyMatrix` functions in the implementation file `matrix.c` to dynamically allocate a matrix and to free up the memory when it is no longer required.
3. Modify your main program from 3.1 to also call the `createMatrixFromFile` and `printMatrix` functions to read and display a matrix.
4. Test your program to verify it is able to display the `example1.mat` file provided.



3.3 Circuit Simulation

In this part you will re-use the files `vector.h`, `vector.c`, `matrix.h`, and `matrix.c` that you have finalised in 3.1 and 3.2. You have also been provided with code to perform the circuit simulation in `circuit.h` and `circuit.c`. The implementation in `circuit.c` is complete and you do not need to modify this file.

1. Populate the data structures in `circuit.h` by inspecting the source code in `circuit.c`.
2. Add to the code in `analyse.c` to call the appropriate functions listed in `circuit.h` to perform modified nodal analysis. Note that the name of the file will be passed in from the command line and will be available to your program in the variable `argv[1]`. You have also been provided with a `makefile` to simplify compilation. Simply type `make` to build the full program.

3. Test the program by analysing some of the example circuits provided and verify that the results are correct for a simple example.



4 Optional Additional Work

1. Create your own netlist files, e.g. by taking examples from ELEC1200 and verify that the program produces the correct results.
2. Can you create a circuit which breaks the simulator?

