# CodeGraph

Project Repository

Hannah Swan
hswan@cs.utah.edu
u0939840

November 10, 2017

## 1 Background and Motivation

The motivation behind this project is selfish, though somewhat motivated by research. In computer science research, as well as industrial software, the code becomes increasingly complex over time. Code is organized based on programmer's styles and for any given project, often many programmers are contributing code over its lifetime. Even well documented programs can be difficult for new programmers to get started with, or for original code authors to pick back up and work later. While all the fine level details are exposed in the often numerous code files, the overall picture of how all these parts fit together can be difficult to visualize. It's also difficult to trace where any data may flow, what functions may affect it, or what other data it might affect.

## 2 Project Objectives

I would like to explore some ways to better visualize code. This is clearly a complex problem that is dependent on programming languages and techniques. For this project, I'll ignore many of the complexities in order to get an idea of what ways of visualizing code could be helpful, and considers the different context in which they would be helpful (as reference, for debugging, to understanding code for a new programmer, for refressing memory of an older programmer, or for programming assistance in editting, etc.)

I plan to start by looking simply at functions and variables. For each function/variable, what other functions/variables may it affect (what comes after it), or affects it(what comes after it)? Where could the data change? Where does the data flow through the program. How does the program flow?

## 3 Data and Data Processing

I'm planning on using a simple C++ CPU ray tracer as my raw data, but I'll also probably use some very small C++ program as test data. The idea here is that I'd like to be able to draw some of my graphs by hand for debugging purposes, but also want a larger data set that is more interesting to visualize. Obviously, in an ideal world, this project would include a parser for a program, but I think that is beyond the scope of the project. I plan to manually create my data: a list of all the functions, their variables, and their calls to other functions in the program. This data may include the order of the function calls, whether or not each variable changes, and any dependencies a variable has within that function.

I'm planning on ignoring advanced programming techniques (such as recursion), and I'll treat classes simply as separate variables and functions.

## 4 Design

Figures 1, 2, 3, and 4 shows design ideas. In particular, I plan to visualize code through graphs. I'd like to be able to do more than just a call hierarchy graph, which are usually too complex to be helpful (as the program grows, anyway). I'd like to have particular views to help visualize different aspects of the program. For each of these views, I also want to make sure that only the needed information is shown, while everything else is ignored. I've explored this idea and pared down to a few different views that I think would be the most helpful.

Figure 5 shows a possible graph for viewing functions (this is similar to a call hierarchy.) Figure 6 shows a way of visualizing variable dependencies between functions. Figure 7 takes into account that the function calls have an order to them, which may be an important aspect to visualize. Finaly, figure 8 would focus on a particular path between two functions.

## 5 Features

### 5.1 Must have

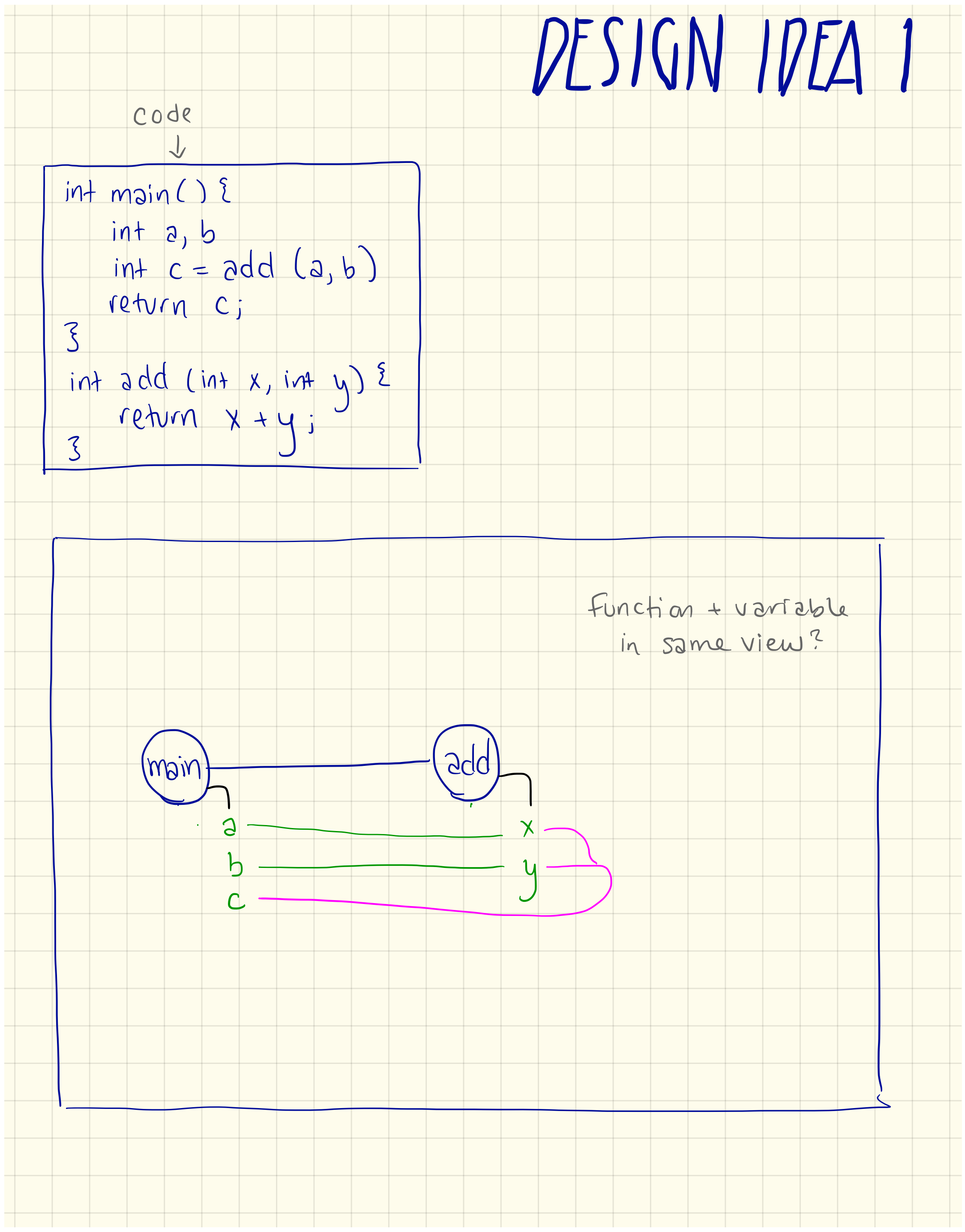
At the very least, I'll need to have a graph (most likely a tree) showing, essentially, the call hierarchy. The interaction will work with a list of functions and/or variables, should show the path(s) of all functions/variables it is dependent on, is dependent on it, and/or both. I think it's also most important to add the path view, at least between a particular call, if not a chain of calls.

### 5.2 Optional

I'd like to have at least one more graph view. (In fact, I think it is a must-have, but I haven't decided which of the alternative views is the must-have.) I'm leaning towards the variable view, but it might become clearer as I progress that another view, like the data flow view, is more necessary. There are many more views that I've considered: a variant of the call hierarchy including call and return paths (where each node would have an enter and exit connection) or the addition of options for showing either all dependencies for a chosen functions/variables or all functions/variable that are have the selected function/variable as a dependency (i.e. all the paths to a function/variable or all the paths from a function/variable.) These kinds of views would be particularly helpful for debugging.

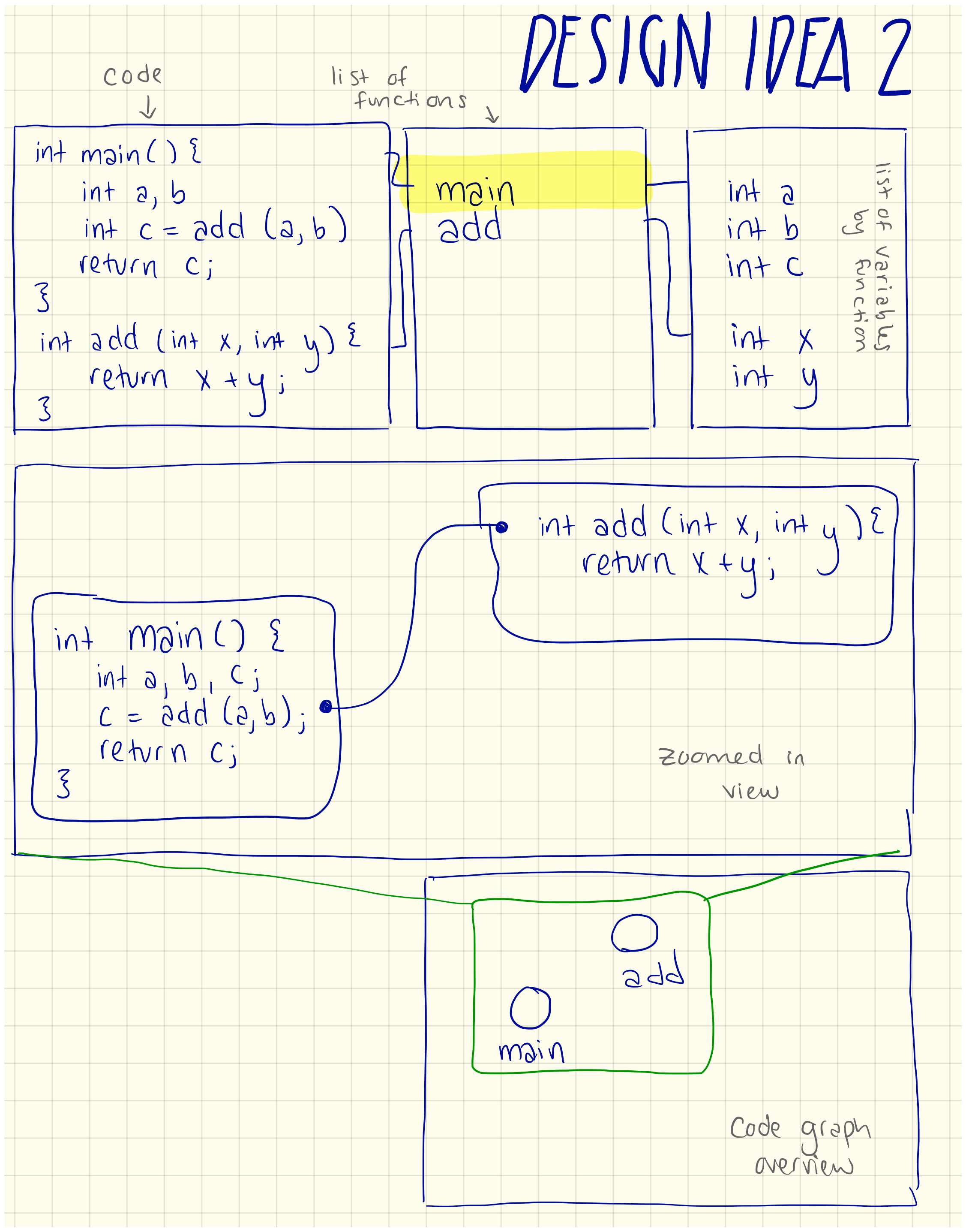
I'm also considering including several lists/text boxes with interactivity between then. For example, at the very least, I'll need a list of all the functions and some list of variables. These could be connected so that when a function is connected, the variable list is sorted or filtered by variables within or affected by that function, and vice versa. Ideally, though it's beyond the scope of the project, it would also be helpful to include a text box with the original code for a selected function, and possibly highlights all calls and declarations of the functions or selected variables. However, this would require the parser that I'll have to ignore.

Ideally, the graph views would have some sort of level of detail processing for zooming in and out, with the most zoomed in view showing the code, and the most zoomed out view simply showing one node for `main()`. Again, this is probably outside the scope of the project.



```
int main() {
    int a, b
    int c = add (a, b)
    return c;
}
int add (int x, int y) {
    return x + y;
}
```

Figure 1: Design 1 Idea



```
int main() {
    int a, b
    int c = add (a,b)
    return c;
}
int add (int x, int y) {
    return x + y;
}
```

```
int add (int x, int y) {
    return x + y;
}
```

```
int main() {
    int a, b, c;
    c = add (a,b);
    return c;
}
```

Figure 2: Design 2 Idea

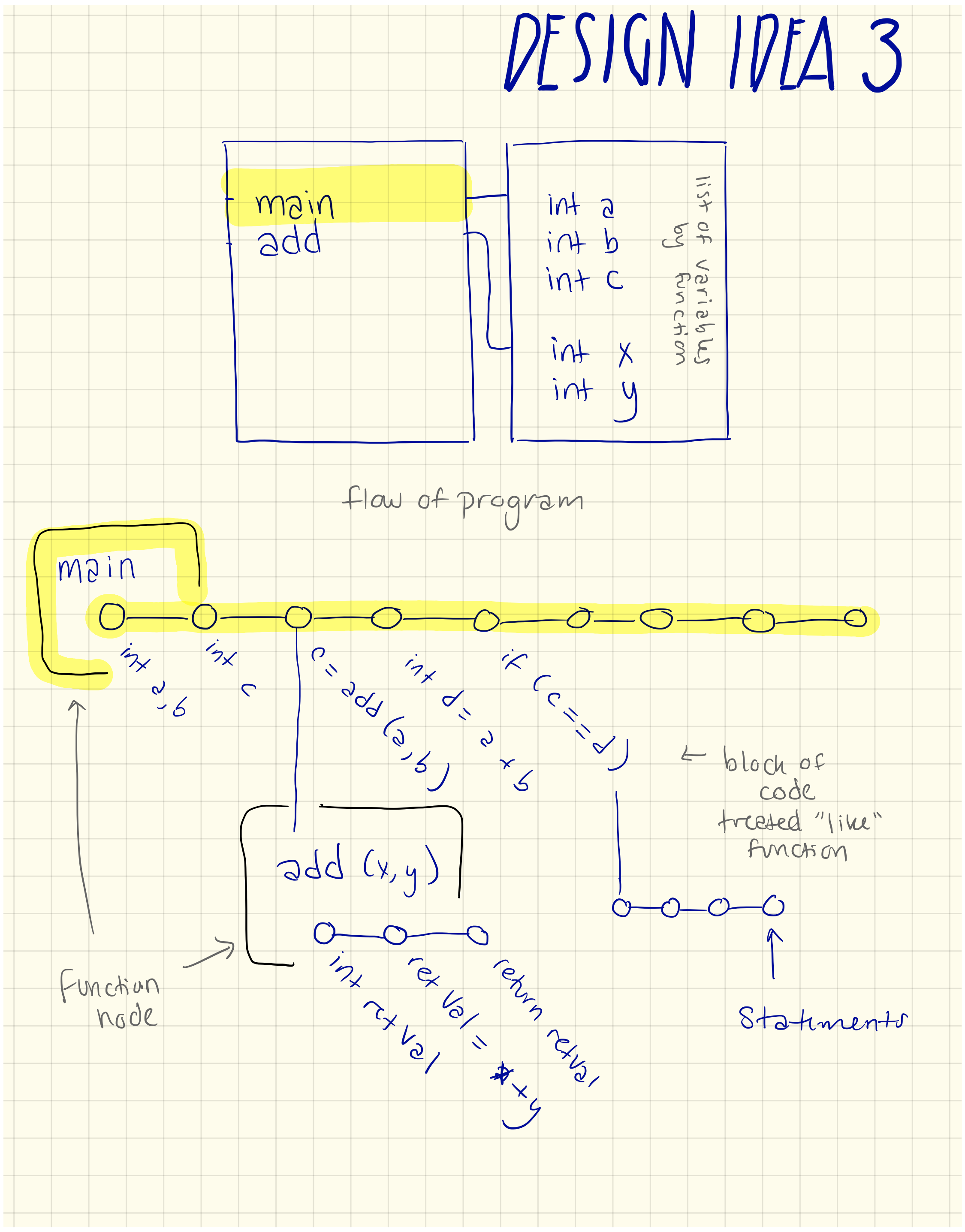


Figure 3: Design 3 Idea

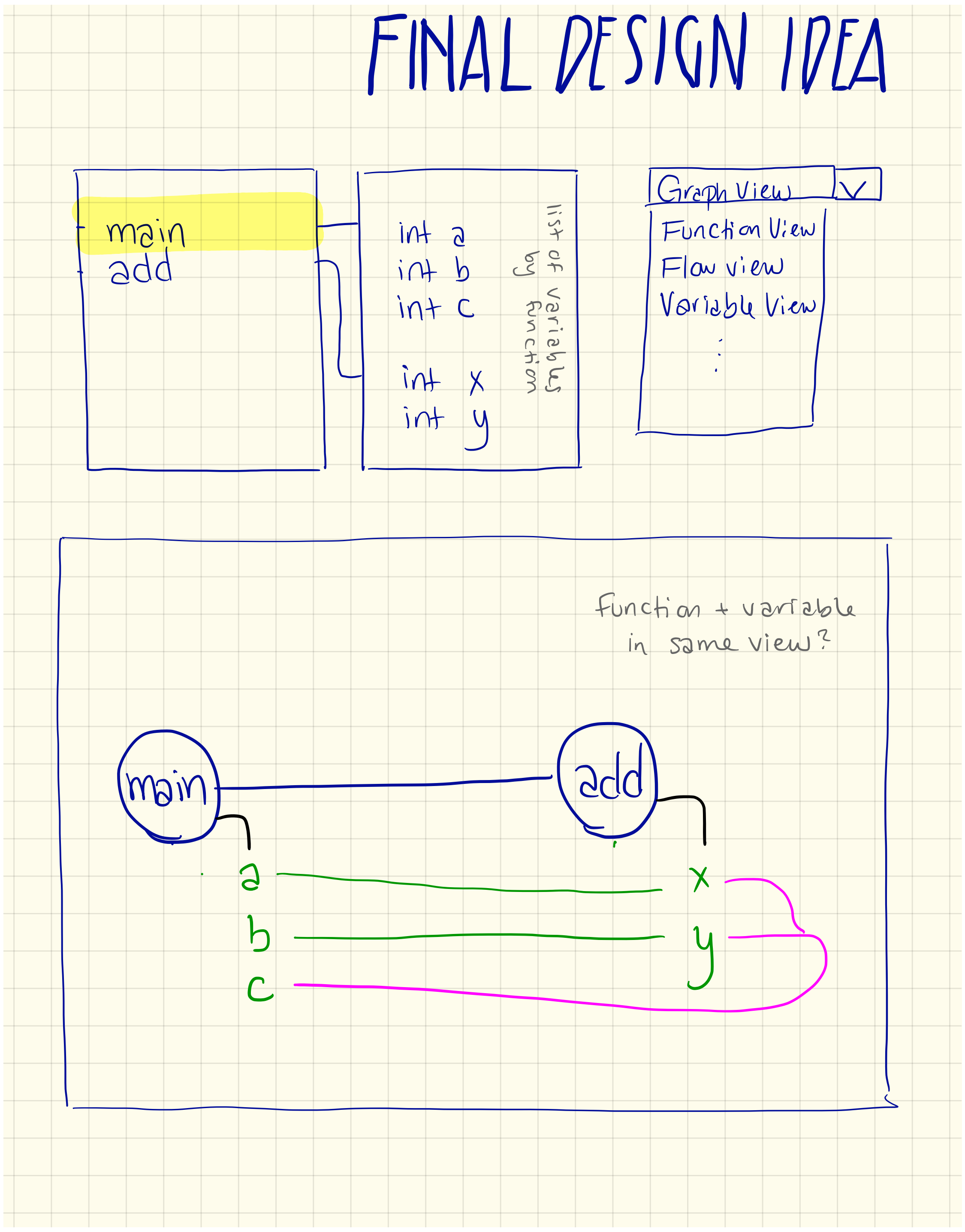


Figure 4: Final Design Idea

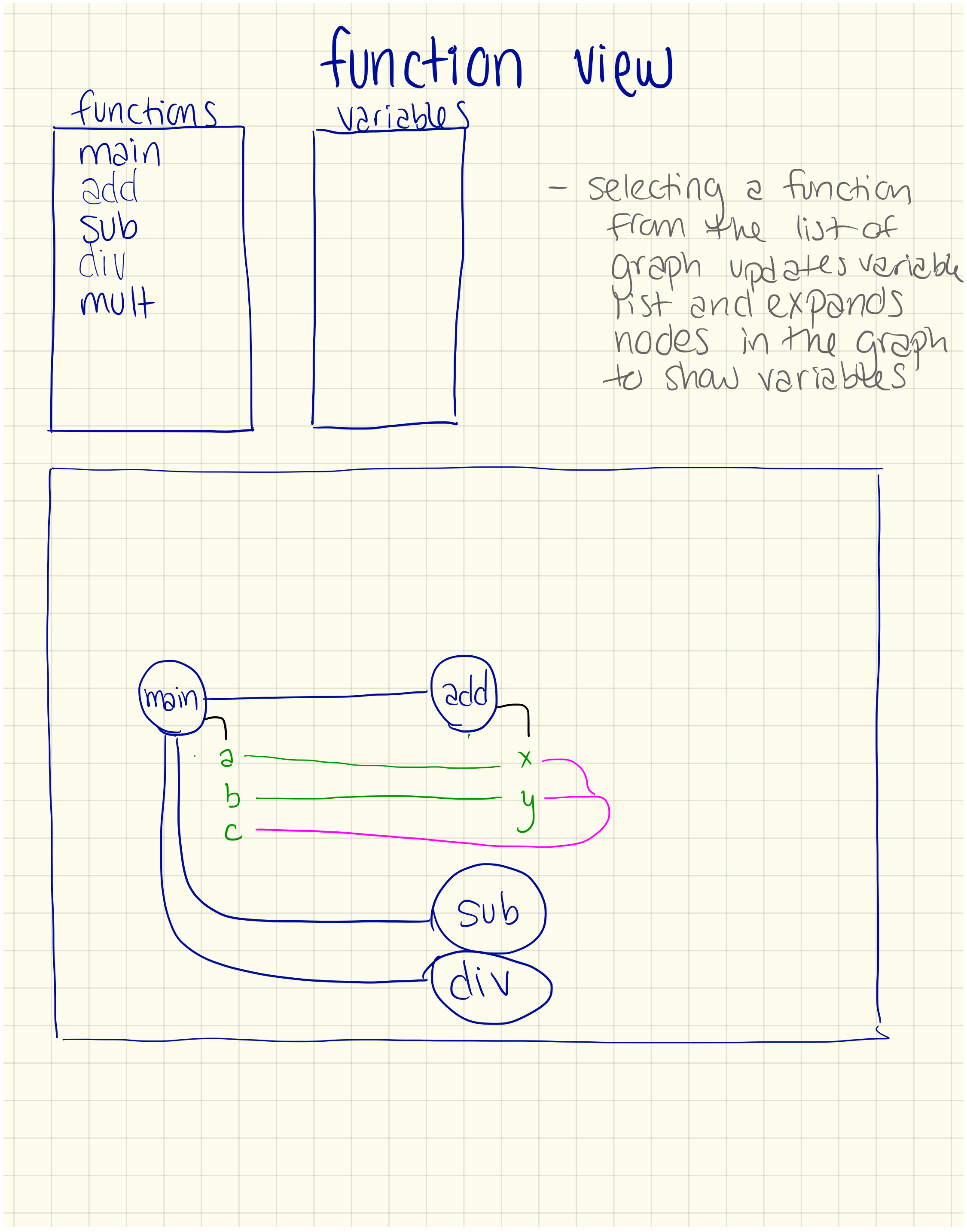

function view
functions
main
add
sub
div
mult
variables
- selecting a function from the list of graph updates variable list and expands nodes in the graph to show variables
main
add
a
b
c
x
y
sub
div

Figure 5: Function View

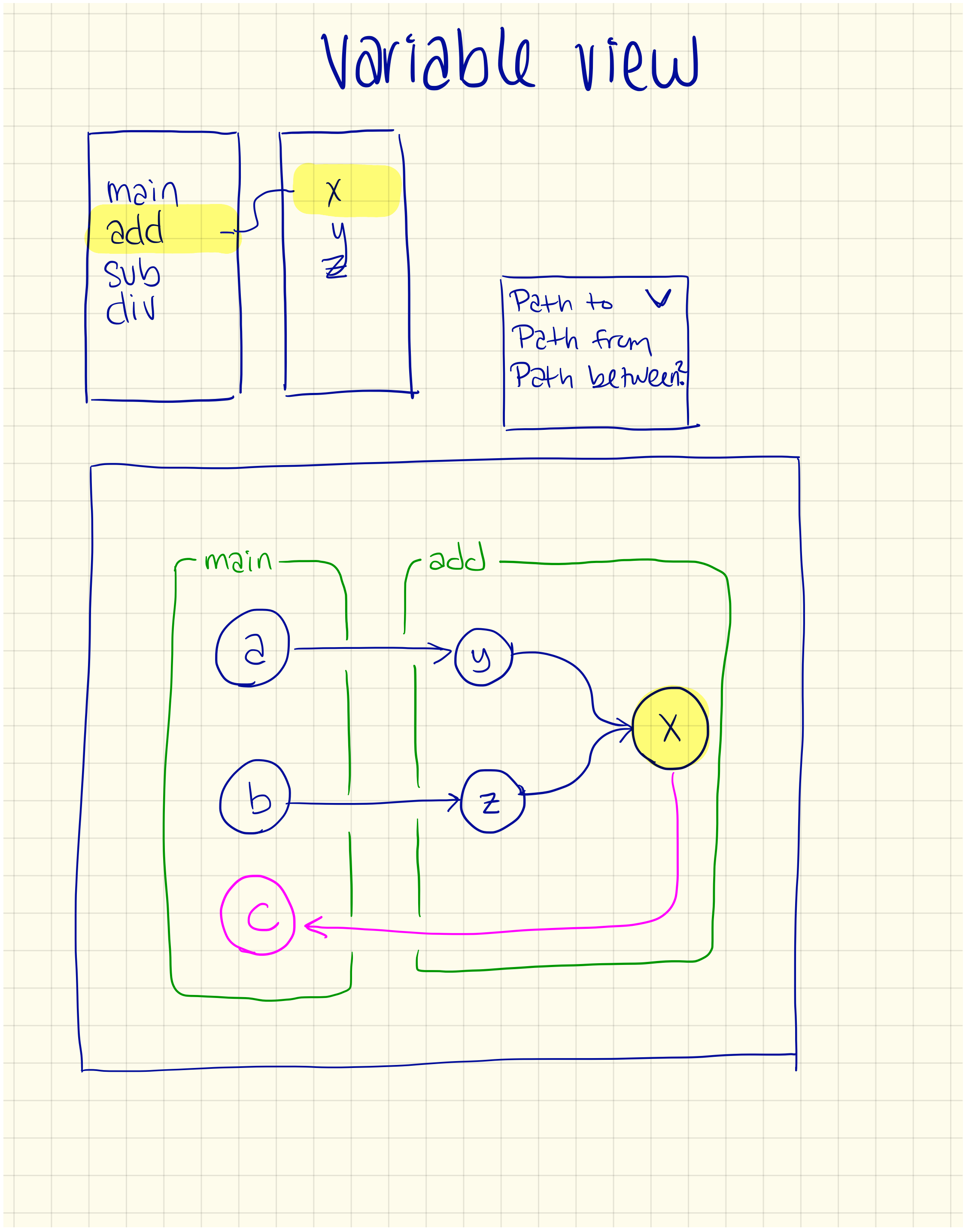


Figure 6: Variable View

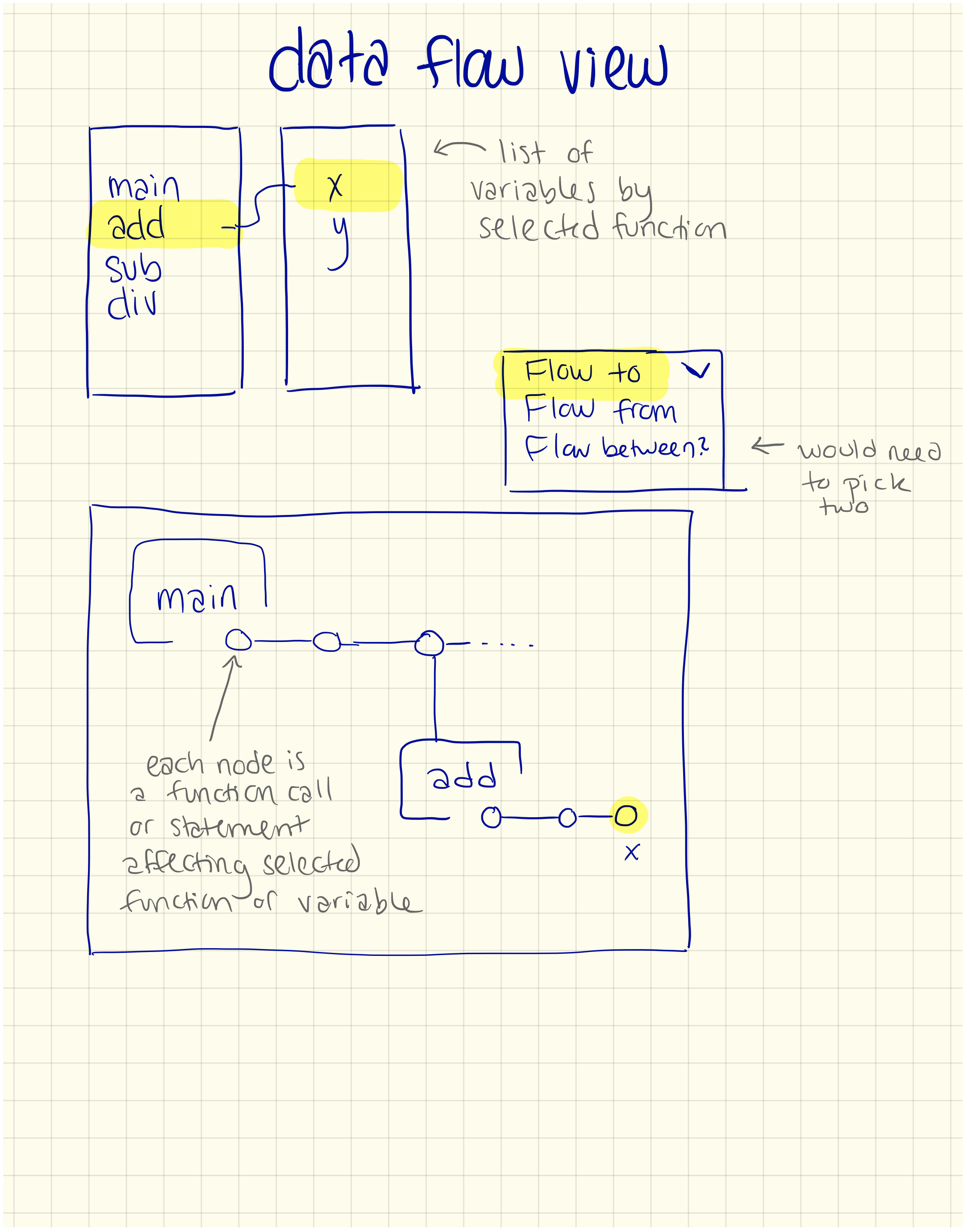

data flow view
main
add
sub
div
x
y
list of variables by selected function
Flow to
Flow from
Flow between?
would need to pick two
main
add
x
each node is a function call or statement affecting selected function or variable

Figure 7: Data Flow View

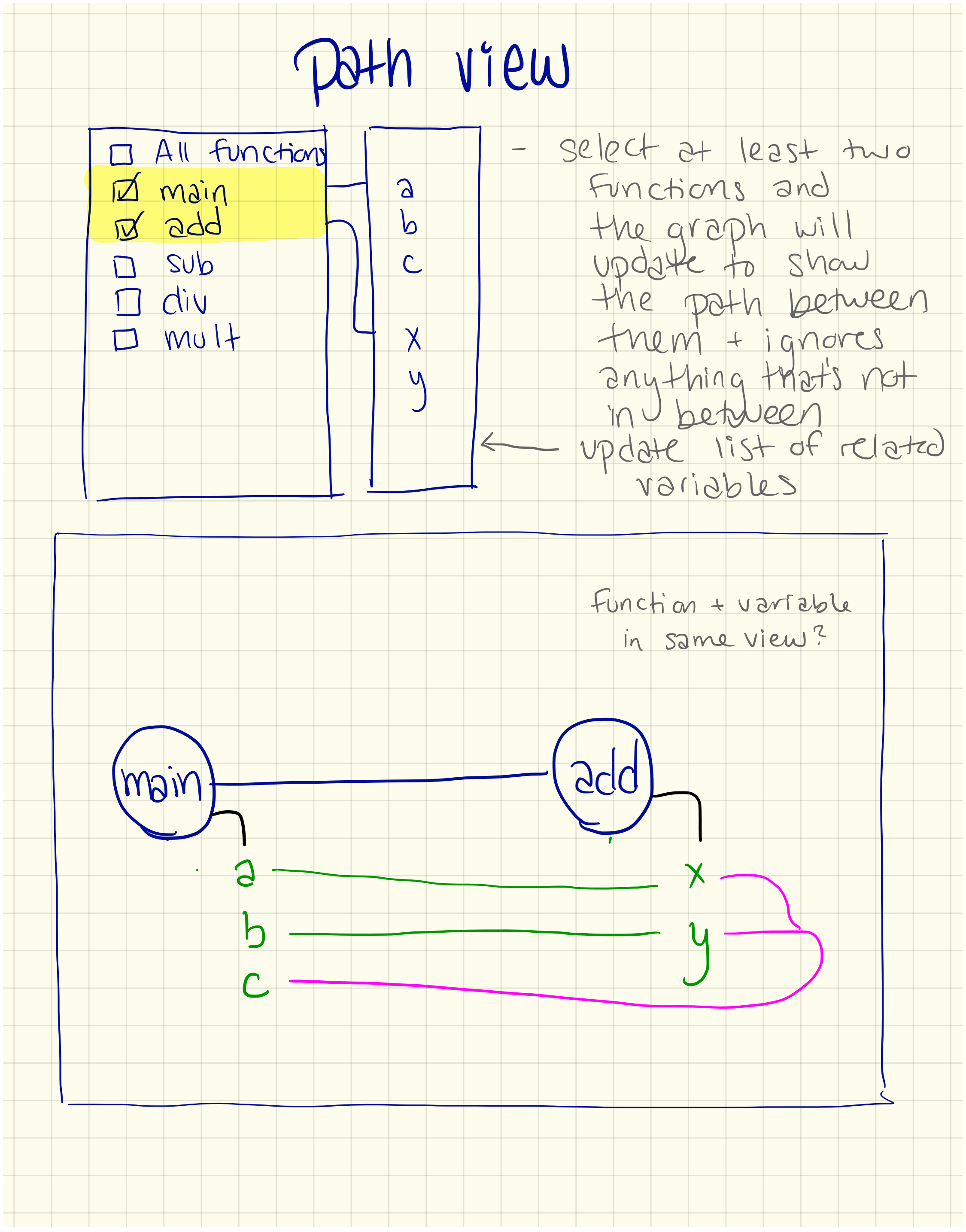

Path view
All functions
main
add
sub
div
mult
a
b
c
x
y
- select at least two functions and the graph will update to show the path between them + ignores anything that's not in between
update list of related variables
Function + variable in same view?
main
add
a
b
c
x
y

Figure 8: Path View

Another thing I'd like to consider is including a class view, or a view that includes classes. I imagine this will become clearer as the must-haves become reality.

## 6 (Rough) Project Schedule

### 6.1 Week One

Data processing and start building and displaying tree/graph (simple call hierarchy)

### 6.2 Week Two

Add list of variables and interactivity with the graph (highlighting relevant paths with relevant colors)

### 6.3 Week Three

Alternate graph view: functions as nodes, but rearranged to show data flow/flow of the program vs. call hierarchy

### 6.4 Week Four

Alternate graph view: variable graph (probably by function)

### 6.5 Week Five

I'll pick the most needed optional feature(s) for the final week