Program Slicing

Program slicing is a technique used to help simplify the reading of code for the purpose of making it more readable for maintenance and debugging.

Dependency graph

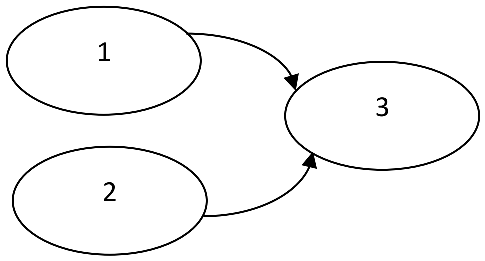
A dependency graph, is a representation of the dependencies within a piece of code. It shows the relationship between code statements and how some code statements can effect the execution of other code statements. To illustrate here is a couple of simple examples:

```
float taxRate = 0.25;                    // line 1

float total = 400;                       // line 2

float taxPayable =total * taxRate;       // line 3
```

We can see the taxPayable value calculated in line 3 depends on code executed in lines 1 and 2.

If we change line 1 as follows : float taxRate = 0.30;   the result calculated in line 3 will change.

We can draw this as a simple graph as follows: (nodes are labelled with line numbers).



Some statements have what are called control dependencies with other statements. This means they control if they execute or not. Look at this example:

```
float taxRate=0.25;              // line 1

float total=400;                 // line 2

if (isfood) {                    // line 3

        taxRate=0.00;            // line 4 don’t charge tax on food items

}
float taxPayable =total * taxRate;       // line 5
```

We can see that line 3 controls if line 4 is executed or not, we can see that line 4 depends on line 3.

We represent the control dependencies by straight lines on the dependency graph.

We can now redraw the graph with the control and data dependencies.

$$\begin{tikzcd}
1 \arrow[rrr, bend left] & & & 5 \\
& 3 \arrow[r] & 4 \arrow[ru] & \\
2 \arrow[rrruu, bend right] & & &
\end{tikzcd}$$

Program slicing using the dependency graph

We can use the graph above to slice up the program to help with maintenance and debugging of the code of the code.

Forward slicing (to help when maintaining code so we can see what lines are effected by changing this line)

So to take a forward slice from node 3, we will include 3, 4 and 5 and end up with this:

```
if (isfood) {                           // line 3

        taxRate=0.00;                   // line 4 don't charge tax on food items

}
float taxPayable =total * taxRate;              // line 5
```

To forward slice from node 1 (float taxRate=0.25;)

We get 1 and 5.

```
float taxRate=0.25;                     // line 1

float taxPayable =total * taxRate;              // line 5
```

Backward slicing

In backward slicing we start at a node and work right to left picking following all nodes in the dependency graph. So if we start with line 4, we get 4 and 3 and the following slice.

```
if (isfood) {                   // line 3
        taxRate=0.00;           // line 4 don’t charge tax on food items
}
```

Backward slicing is used to determine what lines effect the line we are currently working on.