# Forward Slice Testing

# Table of Contents

# Slice Testing Discussion

Slice testing focuses on analyzing specific portions or "slices" of code to evaluate how particular variables, conditions, or statements affect the program's behavior. This approach is useful for isolating and understanding dependencies within a program, especially in more complex systems. Slice testing can be split into two flavors, forward slicing and backward slicing, where each aims to reveal the propagation of data or control influences across a program.

Forward slice testing identifies which statements are affected by a particular statement when the statement is added to a program or modified in any way. Backward slice testing in contrast, identifies all the statements/lines of code that affects a particular statement. Forward slice testing is most useful to determine how much the code will be affected by an addition/modification, helping programmers and testers determine if an addition is worth it and how long it will take to thoroughly test the addition. Backwards slice testing on the other hand is most useful for debugging by isolating particular “slices” of code that may contain the bug. Forward slice testing is performed by starting at a specific point (such as the definition of a particular variable), and identifying all subsequent points in the code that depend on this element, revealing how its value or effect spreads through other components.

# Program Slicing

Variables:
string temp;
string variable;
int end_point;
int line_number;

**temp:**
S(temp, 10)
10 string temp;
16 cin>>temp;
18 ifstream input(temp);
27 cin>>temp;
37 getline(input, temp)
39 if(**temp**.find(variable) != string::npos && line_number >= end_point) output << line_number << " " << **temp** << "\n";

**variable:**
S(variable, 11)
11 string variable;
21 cin>>variable;
30 output << “S(“ << variable << ", " << end_point << ")\n\n";
39 if(temp.find(**variable**) != string::npos && line_number >= end_point) output << line_number << " " << temp << "\n";

**end_point:**
S(end_point,12)
24 cin>>end_point;
39 if(temp.find(variable) != string::npos && line_number >= **end_point**) output << line_number << " " << temp << "\n";

**line_number:**
S(line_number,13)
36 line_number++;

39 if(temp.find(variable) != string::npos && **line_number** >= end_point) output << **line_number** << " " << temp << "\n";

# Sample Output

```
1   sum
2   int main()
3   {
4       int sum;
5
6       int beastmode;
7
8       beastmode = 19;
9
10      sum = beastmode;
11
12
13      int testa;
14
15      testa = sum / beastmode;
16
17
18      if(testa > 5)
19          cout << "one Gorgillion" << endl;
20
21      sum = sum * testa;
22
23      cout << sum << endl;
24
25
26
27      return 0;
28
29
30
31
32
```

This is the file we will be slicing.

In this test, we will be slicing on the variable “beastmode”, starting at line 6.

```
gfors@thomas:~/public_html$ g++ slice.cpp
gfors@thomas:~/public_html$ a.out
Please enter the name of the file you wish to slice:
forward.txt
Please enter the variable you want to perform the slice on:
beastmode
Please enter the endpoint of the slice:
6
Please enter the name of the output file:
rfkbrainworm.txt
gfors@thomas:~/public_html$
```

```
1    S(beastmode, 6)
2
3    6   int beastmode;
4    8   beastmode = 19;
5    10  sum = beastmode;
6    15  testa = sum / beastmode;
7
```

Above is what was written to the file provided by the user. Each time beastmode appears in the program past line 6 it is included.

**Sum:**

```
Please enter the name of the file you wish to slice:
forward.txt
Please enter the variable you want to perform the slice on:
sum
Please enter the endpoint of the slice:
4
Please enter the name of the output file:
test.txt
```

```
S(sum, 4)

4       int sum;
10      sum = beastmode;
15      testa = sum / beastmode;
18      sum = sum * testa;
20      cout << sum << endl;
```

**Testa:**

```
S(testa, 12)

12        int testa;
14        testa = sum / beastmode;
17        if(testa > 5)
20        sum = sum * testa;
```

```
Please enter the name of the file you wish to slice:
forward.txt
Please enter the variable you want to perform the slice on:
testa
Please enter the endpoint of the slice:
12
Please enter the name of the output file:
test.txt
amankikani@Aman-Kikanis-Macbook-Pro-2 Downloads % _
```

# Reflections

Gavin: In this project I designed the slicing program and ensured that it correctly extracted lines that contained the variable entered by the user, keeping the endpoint in mind. Additionally, I created the sample output. The test was done on a .cpp program read in as a text file. Sum is seen at the top of the file because I was testing if the endpoint ensured that that wouldn't be read in, if performing the slicing on "sum". Prior to this assignment, I was unfamiliar with slice testing, despite the fact that I unknowingly perform it when debugging my code. It is an efficient and easy to understand way of analyzing and testing software- and I much prefer it to BVA.

Sean: Aman and I worked on the slicing of the four different variables. Using the forward slicing technique we analyzed the program and identified all the lines that were affected by the different variables. Since there were only four variables the slicing was pretty fast and straightforward, which is a major improvement over BVA. This has been my favorite flavor of software testing that we have learned so far this semester for a few reasons. A reason why I like slice testing the best is because compared to the earlier methods there is less repetitive work so it feels more worthwhile and rewarding to do this over BVA. The other reason I like this method best is because it makes the most sense to me and is more similar to the testing/debugging I normally do with my code.

Aman: Sean and I worked on the slicing of both the program as well as Forward.txt. We used the slice.cpp file to go through the forward.txt file and see where and when each variable was being used. If it was being used then it would log it accordingly. We also had to go through the slicer

manually to show we knew what slicing was. We only used four variables so the manual slicing was quite easy. All one needs to do in this scenario is to look at the file and find out when the variable is first used to show and then mark down when its made and each time its used including its line number. This is useful in software testing because when programs become larger, instead of just looking through millions of lines of code, you can just run a slicer to see when each variable is being used and narrow down where errors could be occuring/