

```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {  
    iN = -1;
```

ProgPilot : Static Analyzer

Eric Therond

contact@designsecurity.org

<http://www.designsecurity.org>

```
    if (iLength < 4) {  
        again = true;  
        continue;  
    } else if (sInput[iLength - 3] != '.') {  
        again = true;  
        continue;  
    } while (++iN < iLength) {  
        if (isdigit(sInput[iN])) {  
            continue;  
        } else if (iN == (iLength - 3)) {  
            continue;  
        }  
    }
```

```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {  
    iN = -1;  
    again = false;  
    getline(cin, sInput);  
    system("cls");
```

Introduction

```
    string sTemp;  
    iLength = sInput.length();  
    if (iLength < 4) {  
        again = true;  
        continue;  
    } else if (sInput[iLength - 3] != '.') {  
        again = true;  
        continue;  
    } while (++iN < iLength) {  
        if (isdigit(sInput[iN])) {  
            continue;  
        } else if (iN == (iLength - 3)) {  
            continue;  
        }  
    }  
}
```

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

What are the goals of static analysis ?

- ▶ Find defaults in software
- ▶ Improve dynamical analysis
- ▶ Compute cyclomatic number
- ▶ And prioritize performance tests on code that have high cyclomatic number

```
529         continue;  
530     } else if (sInput[iLength]  
531         again = true;  
532         continue;  
533     } while (++iN < iLength) {  
534         if (isdigit(sInput[iN])) {  
535             continue;  
536         } else if (iN == (iLength - 3)) {  
537             continue;  
538         }
```

Find lexical defaults

```
<?php
if(true)
return;
else
echo "error";
?>
```

Find syntactical defaults

```
<?php
for(if(true)$i =0;else $i=1; $i < 10; $i++)
{
echo "voila"
}
?>
```

Find semantic defaults

```
<?php
class A
{
public $test1;
private $test2;
};

$var1 = new A;
$var1->test2 = "welcome";
?>
```

Code Source

Abstract Syntax Tree

Control
Flow Graph

Decision 1

Process 2b

Process 2a text
text text text

```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {  
    iN = -1;  
    again = false;  
    (in sInput);
```

text

```
system( "clear" );  
stringstream(sInput);  
iLength = sInput.length();  
if (iLength < 4) {  
    again = true;  
    continue;  
} else if (sInput[iLength - 3] != '.') {  
    again = true;  
    continue;  
} while (++iN < iLength) {  
    if (isdigit(sInput[iN])) {  
        continue;  
    } else if (iN == (iLength - 3)) {  
        continue;  
    }  
}
```



```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {  
    iN = -1;  
    again = false;  
    getline(cin, sInput);  
    system("clear");  
    stringstream ss(sInput);  
    string sTemp;  
    iLength = sInput.length();  
    if (iLength < 4) {  
        again = true;  
        continue;  
    } else if (sInput[iLength - 3] != '.') {  
        again = true;  
        continue;  
    } while (++iN < iLength) {  
        if (isdigit(sInput[iN])) {  
            continue;  
        } else if (iN == (iLength - 3)) {  
            continue;  
        }  
    }  
}
```

Representations

```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {
```

Abstract syntax tree

- ▶ A set of informations about syntax is stored in a tree.
- ▶ Tree, graphs are very simple to use (traversors).
- ▶ Preserve semantic of a program (unambiguous syntax).

```
    again = true;  
    continue;  
} else if (sInput[iLength - 3] != ".")  
    again = true;  
    continue;  
} while (++iN < iLength) {  
    if (isdigit(sInput[iN])) {  
        continue;  
    } else if (iN == (iLength - 3)) {  
        continue;  
    }  
}
```



```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

Control flow graph

- ▶ CFG is a graph where nodes represent basic blocks and edges represent jumps in the control flow graph.
- ▶ Essential tool because all paths of a program during its execution could be represented.
- ▶ Simple analysis of CFG could reveal defects in a program (potential optimizations like unreachable codes).

```
529  
530 } else if (sInput[iN] == '0')  
531     again = true;  
532     continue;  
533 } while (++iN < iLength) {  
534     if (isdigit(sInput[iN])) {  
535         continue;  
536     } else if (iN == (iLength - 3)) {  
537         continue;  
538     }  
539 }
```

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

```
21 while (again) {  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36
```

Call graphs

- ▶ Call graphs is a CFG where relationships between functions are represented.
- ▶ Nice for debugging performance issues.
- ▶ But also to identify malicious code (like backdoors).

```
again;  
continue;  
} else if (sInput[iLength - 3])  
again = true;  
continue;  
} while (++iN < iLength) {  
if (isdigit(sInput[iN])) {  
continue;  
} else if (iN == (iLength - 3)) {  
continue;  
}
```

```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {  
    iN = -1;  
    again = false;  
    getline(cin, sInput);  
    system("clear");  
    stringstream(sInput, &dblTemp);  
    string sTemp = sInput;  
    iLength = sInput.length();  
    if (iLength < 4) {  
        again = true;  
        continue;  
    } else if (sInput[iLength - 3] != '.') {  
        again = true;  
        continue;  
    } while (++iN < iLength) {  
        if (isdigit(sInput[iN])) {  
            continue;  
        } else if (iN == (iLength - 3)) {  
            continue;  
        }  
    }  
}
```

Data flow analysis

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

```
21 while (again) {  
22     iN = -1;  
23     false;
```

What are the objectives of static analysis ?

- ▶ Reaching definitions

```
526 iLength = sInput.length();  
527 if (iLength < 4) {  
528     again = true;  
529     continue;  
530 } else if (sInput[iLength - 3] != '.') {  
531     again = true;  
532     continue;  
533 } while (++iN < iLength) {  
534     if (isdigit(sInput[iN])) {  
535         continue;  
536     } else if (iN == (iLength - 3)) {  
537         continue;  
538     }
```

Reaching definitions equations

$$\begin{aligned} kill_{RD}([x := a]^l) &= (x, ?) \\ &\quad \cup \{(x, l') \mid B^{l'} \text{ is an assignment to } x \text{ in } S_*\} \\ gen_{RD}([x := a]^l) &= \{(x, l)\} \end{aligned}$$

$$RD_{entry}(l) \begin{cases} \{(x, ?) \mid x \in FV(S_*)\} & \text{if } l = init(S_*) \\ \cup \{RD_{exit}(l') \mid (l', l) \in flow(S_*)\} & \text{otherwise} \end{cases}$$

$$RD_{exit}(l) \begin{cases} (RD_{entry}(l) \setminus kill_{RD}(B^l)) \cup gen_{RD}(B^l) \\ \text{where } B^l \in blocks(S_*) \end{cases}$$

Reaching definitions algorithm

for each node n **do**

$out[n] = gen[n]$

end for

$change = true$

while $change$ **do**

$change = false$

for each node n **do**

$in[n] = \bigcup out[p]$ where p is an immediate predecessor of n

$oldout = out[n]$

$out[n] = gen[n] \cup (in[n] - kill[n])$

if $out[n] \neq oldout$ **then**

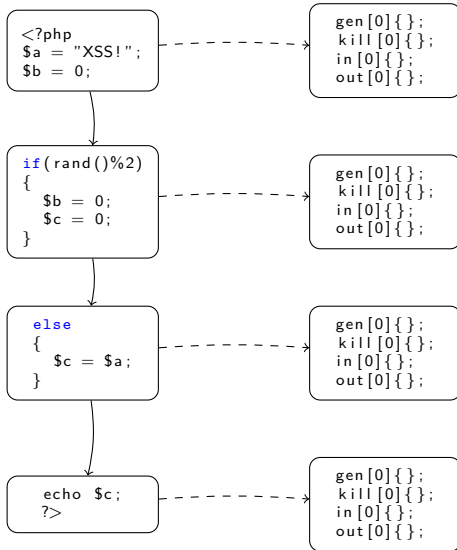
$change = true$

end if

end for

end while

Reaching definitions analysis



```
string sInput;  
int iLength, iN;  
double dblTemp;  
bool again = true;
```

```
while (again) {
```

Control flow graph

	May	Must
Forward	Reaching definitions	Available expressions
Backward	Live variables	Very busy expressions

```
    again = true;  
    continue;  
} else if (sInput[iLength - 3] != '0')  
    again = true;  
    continue;  
} while (++iN < iLength) {  
    if (isdigit(sInput[iN])) {  
        continue;  
    } else if (iN == (iLength - 3)) {  
        continue;  
    }  
}
```

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

```
21 while (again) {  
22     iN = -1;  
23     again = false;  
24     getline(cin, sInput);  
25     system("clear");  
26     string sTemp;
```

Visualizations

```
27     iLength = sInput.length();  
28     if (iLength < 4) {  
29         again = true;  
30         continue;  
31     } else if (sInput[iLength - 3] != '.') {  
32         again = true;  
33         continue;  
34     } while (++iN < iLength) {  
35         if (isdigit(sInput[iN])) {  
36             continue;  
37         } else if (iN == (iLength - 3)) {  
38             continue;  
39         }
```

What kinds of security tests can be automated ?

```
#include <future>
std::map<std::string, std::string> french
{{"hello", "bonjour"}, {"world", "tout le monde"}};
int main()
{
    std::string greet=french["hello"];
    auto f=std::async( [&]{std::cout << greet << ", ";});
    std::string audience=french["word"];
    f.get();
    std::cout<<audience<<std::endl;
}
```

time consuming I/O

next lookup