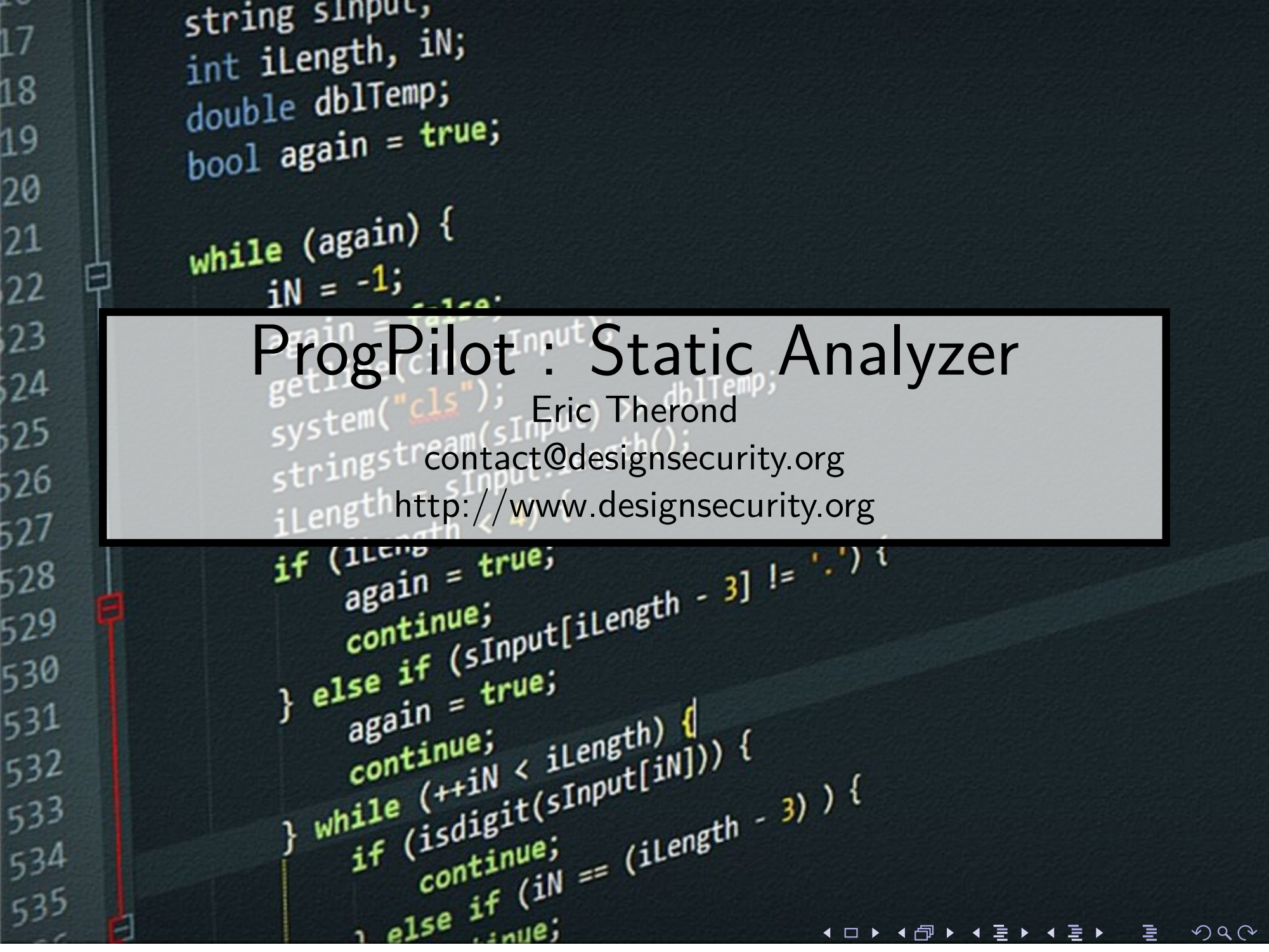


ProgPilot : Static Analyzer

Eric Therond
contact@designsecurity.org
http://www.designsecurity.org


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
system("cls");
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)) {
Introduction

What are the goals of static analysis ?

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

Find lexical defaults

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

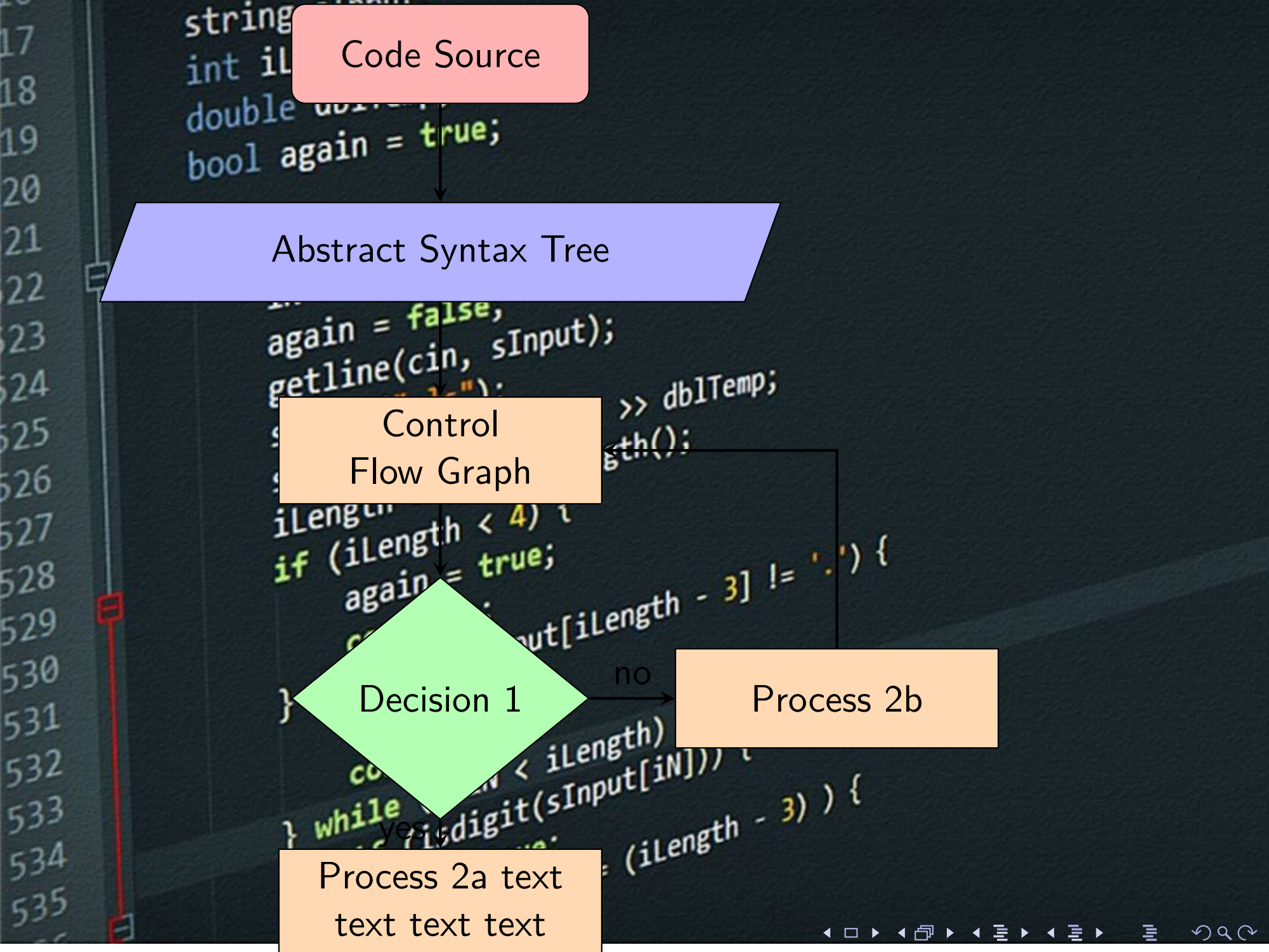
Find syntaxical defaults

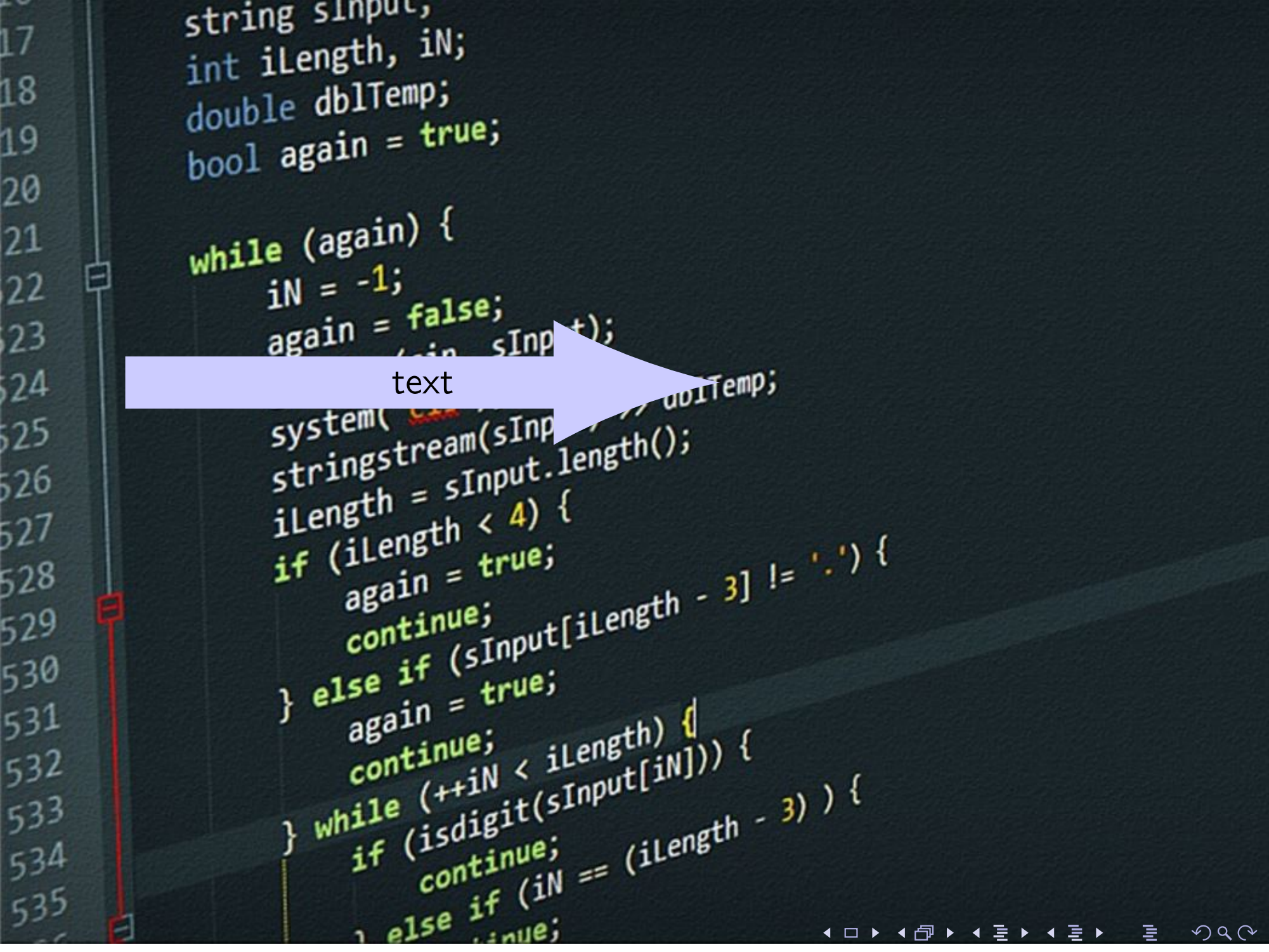
```php
<?php
for(if(true)$i =0;else $i=1; $i < 10; $i++)
{
echo "voila"
}
?>
```

Find semantic defaults

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

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


Code Source
Abstract Syntax Tree
Control
Flow Graph
Decision 1
no
Process 2b
yes
Process 2a text
text text text


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
text
stringstream(sInput.length();
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)) {


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
iLength = sInput.
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)) {
Representations

Abstract syntax tree

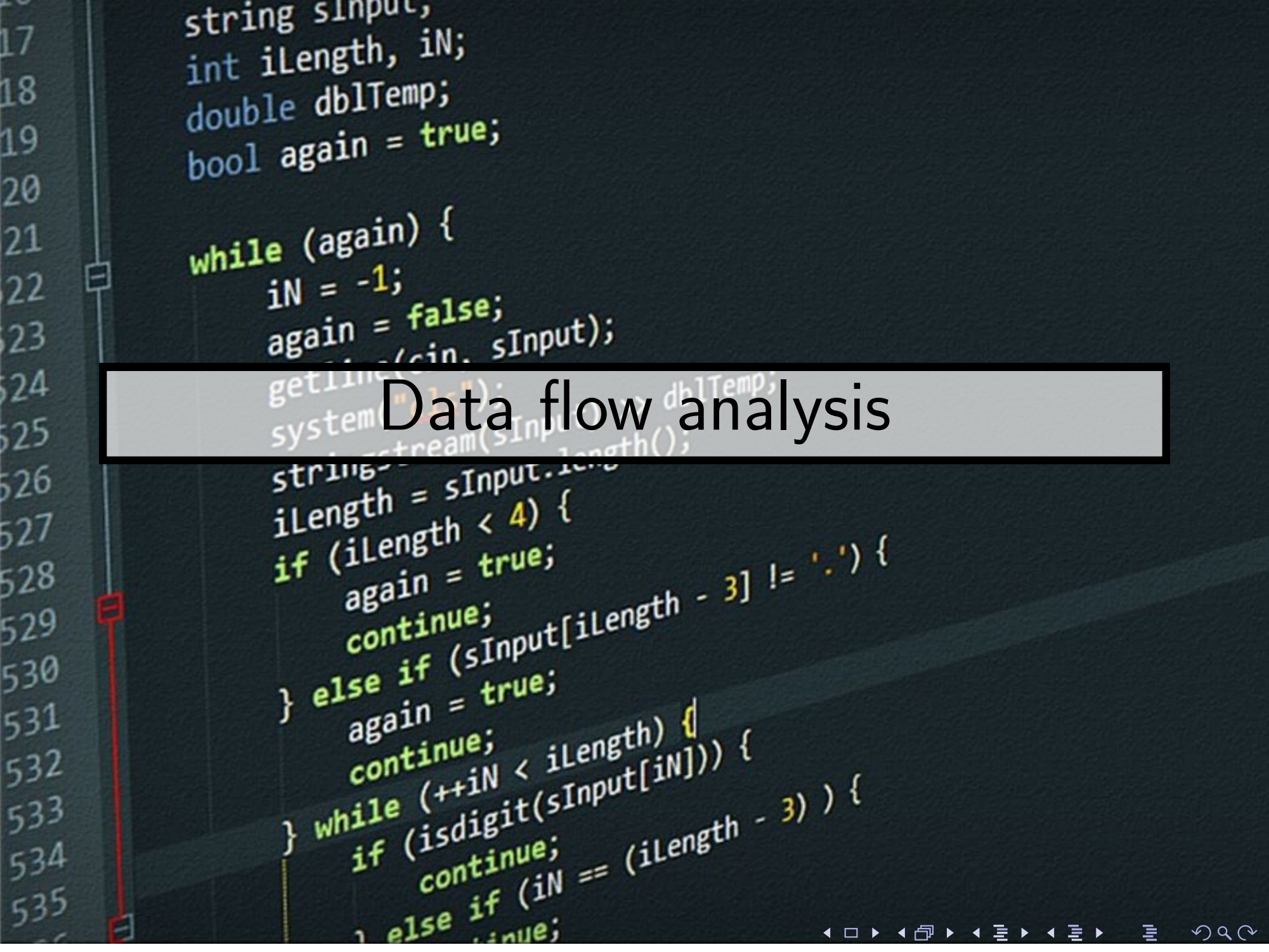
- A set of informations about syntax is stored in a tree.
- Tree, graphs are very simple to use (traversors).
- Preserve semantic of a program (unambiguous syntax).

Control flow graph

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

Call graphs

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


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
iLength = sInput.
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)) {
Data flow analysis

What are the objectives of static analysis ?

- Reaching definitions

Reaching definitions equations

$$kill_{RD}([x := a]^l) = (x, ?) \bigcup \{(x, l') | B^{l'} \text{ is an assignment to } x \text{ in } S_*\}$$

$$gen_{RD}([x := a]^l) = \{(x, l)\}$$

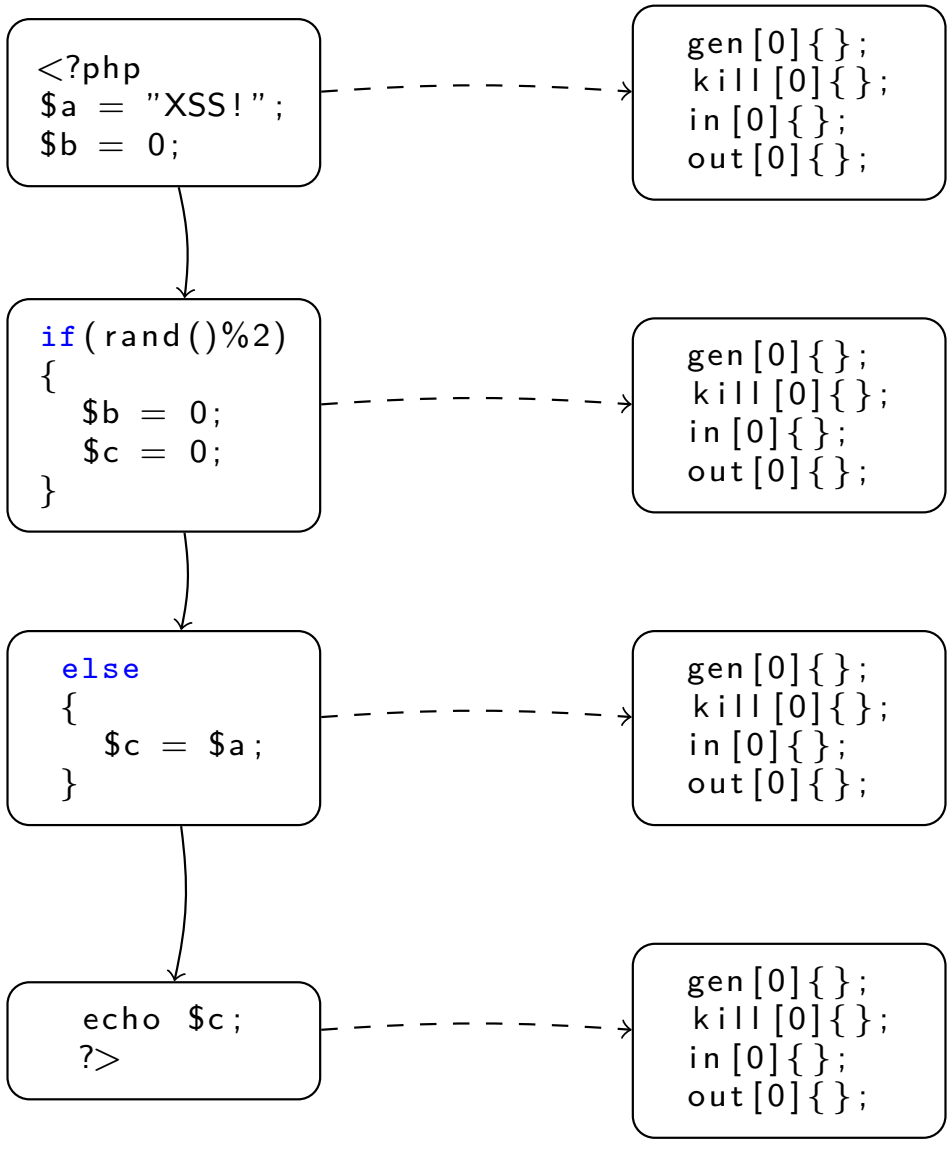
$$RD_{entry}(l) \begin{cases} \{(x, ?) | x \in FV(S_*)\} & \text{if } l = init(S_*) \\ \bigcup \{RD_{exit}(l') | (l', l) \in flow(S_*)\} & \text{otherwise} \end{cases}$$

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

Reaching definitions algorithm

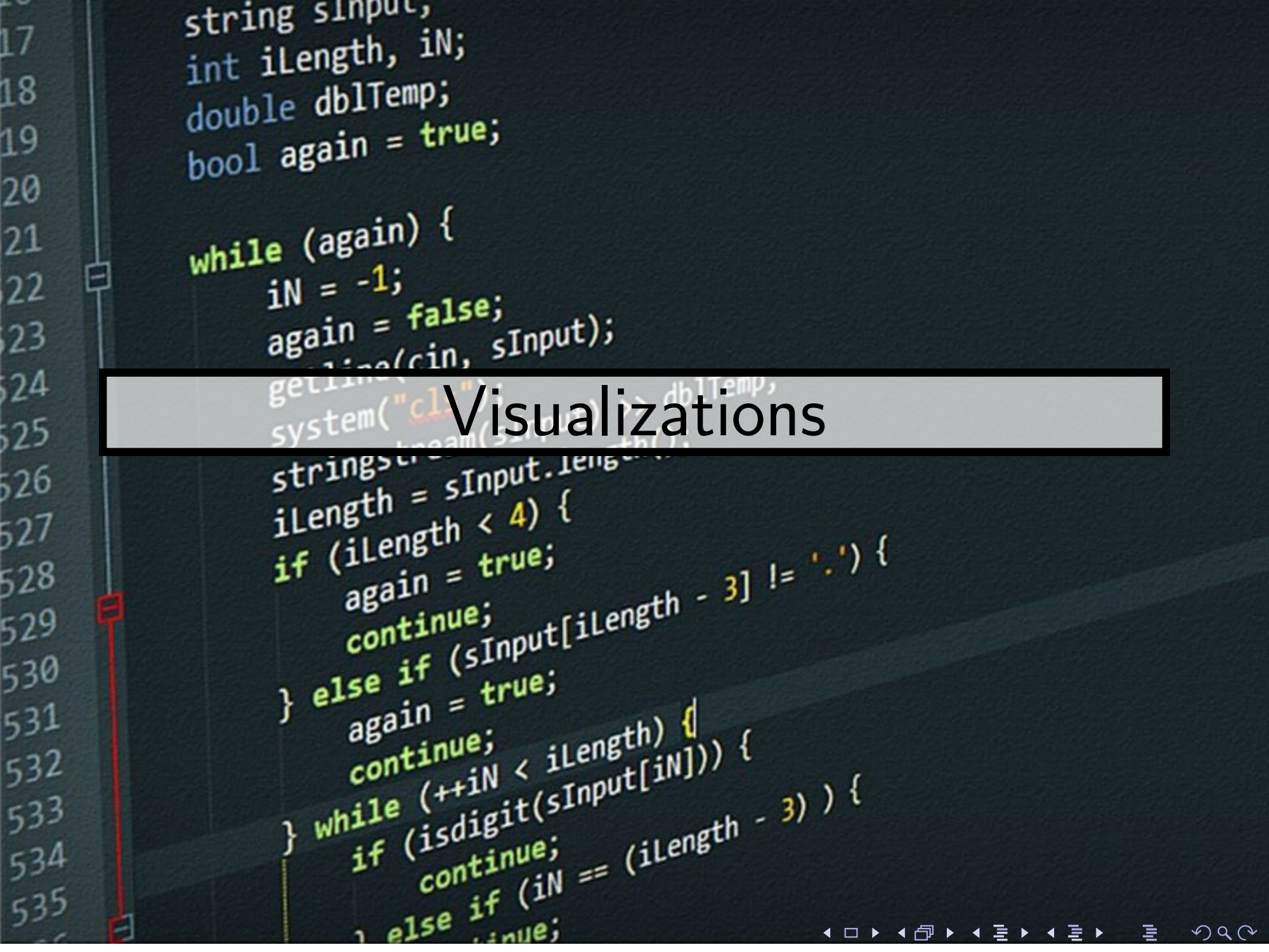
for each node n **do**
$out[n] = gen[n]$
end for
$change = true$
while change **do**
$change = false$
    for each node n **do**
$in[n] = \bigcup out[p]$ where p is an immediate predecessor of n
$oldout = out[n]$
$out[n] = gen[n] \bigcup (in[n] - kill[n])$
        if $out[n] \neq oldout$ **then**
$change = true$
        end if
    end for
end while

Reaching definitions analysis



Control flow graph

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


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
system("cl
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)) {
Visualizations

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