

## Program 1

```
$
  let a: int;
  input(a);

  let b: int;
  input(b);

  let c: int;
  input(c);

  let max = a;
  check(b > a) {
    max = b;
  }

  check(c>max) {
    max = c;
  }
$
```

### Output:

```
"The program is lexically correct!"
```

### PIF:

```
$ -> -1
let -> -1
a -> 1
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
a -> 1
) -> -1
; -> -1
let -> -1
b -> 2
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
```

```
b -> 2
) -> -1
; -> -1
let -> -1
c -> 3
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
c -> 3
) -> -1
; -> -1
let -> -1
max -> 4
= -> -1
a -> 1
; -> -1
check -> -1
( -> -1
b -> 2
> -> -1
a -> 1
) -> -1
{ -> -1
max -> 4
= -> -1
b -> 2
; -> -1
} -> -1
check -> -1
( -> -1
c -> 3
> -> -1
max -> 4
) -> -1
{ -> -1
max -> 4
= -> -1
c -> 3
; -> -1
} -> -1
$ -> -1
```

**ST:**

Data Structure: Balanced binary search tree, implemented using a red-black tree

a -> 1

b -> 2

c -> 3

max -> 4

**Program 2**

```
$
  let number: int;
  input(number);

  let primeAnswer = "Prime";
  let nonPrimeAnswer = "NonPrime";

  check(number < 2) {
    output(nonPrimeAnswer);
    exit;
  }

  check(number == 2) {
    output(primeAnswer);
    exit;
  }

  check(number % 2 == 0) {
    output(nonPrimeAnswer);
    exit;
  }

  loop(let d = 3; d * d <= number; d = d + 2) {
    check (number % d == 0) {
      output(nonPrimeAnswer);
      exit;
    }
  }

  output(primeAnswer);
$
```

**Output:**

"The program is lexically correct!"

**PIF:**

```
$ -> -1
let -> -1
number -> 1
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
number -> 1
) -> -1
; -> -1
let -> -1
primeAnswer -> 2
= -> -1
"Prime" -> 3
; -> -1
let -> -1
nonPrimeAnswer -> 4
= -> -1
"NonPrime" -> 5
; -> -1
check -> -1
( -> -1
number -> 1
< -> -1
2 -> 6
) -> -1
{ -> -1
output -> -1
( -> -1
nonPrimeAnswer -> 4
) -> -1
; -> -1
exit -> -1
; -> -1
} -> -1
check -> -1
( -> -1
number -> 1
== -> -1
2 -> 6
) -> -1
{ -> -1
output -> -1
```

```
( -> -1
primeAnswer -> 2
) -> -1
; -> -1
exit -> -1
; -> -1
} -> -1
check -> -1
( -> -1
number -> 1
% -> -1
2 -> 6
== -> -1
0 -> 7
) -> -1
{ -> -1
output -> -1
( -> -1
nonPrimeAnswer -> 4
) -> -1
; -> -1
exit -> -1
; -> -1
} -> -1
loop -> -1
( -> -1
let -> -1
d -> 8
= -> -1
3 -> 9
; -> -1
d -> 8
* -> -1
d -> 8
<= -> -1
number -> 1
; -> -1
d -> 8
= -> -1
d -> 8
+ -> -1
2 -> 6
) -> -1
{ -> -1
check -> -1
```

```

( -> -1
number -> 1
% -> -1
d -> 8
== -> -1
0 -> 7
) -> -1
{ -> -1
output -> -1
( -> -1
nonPrimeAnswer -> 4
) -> -1
; -> -1
exit -> -1
; -> -1
} -> -1
} -> -1
output -> -1
( -> -1
primeAnswer -> 2
) -> -1
; -> -1
$ -> -1

```

#### **ST:**

Data Structure: Balanced binary search tree, implemented using a red-black tree

```

"NonPrime" -> 5
"Prime" -> 3
0 -> 7
2 -> 6
3 -> 9
d -> 8
nonPrimeAnswer -> 4
number -> 1
primeAnswer -> 2

```

### **Program 3**

```

$
    let sum = 0;
    let currentNumber: int;

    let n: int;
    input(n);

```

```
    loop(let i = 0; i<n; i=i+1) {
        input(currentNumber);
        sum = sum + currentNumber;
    }

    output(sum);
$
```

**Output:**

"The program is lexically correct!"

**PIF:**

```
$ -> -1
let -> -1
sum -> 1
= -> -1
0 -> 2
; -> -1
let -> -1
currentNumber -> 3
: -> -1
int -> -1
; -> -1
let -> -1
n -> 4
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
n -> 4
) -> -1
; -> -1
loop -> -1
( -> -1
let -> -1
i -> 5
= -> -1
0 -> 2
; -> -1
i -> 5
< -> -1
n -> 4
; -> -1
```

```

i -> 5
= -> -1
i -> 5
+ -> -1
1 -> 6
) -> -1
{ -> -1
input -> -1
( -> -1
currentNumber -> 3
) -> -1
; -> -1
sum -> 1
= -> -1
sum -> 1
+ -> -1
currentNumber -> 3
; -> -1
} -> -1
output -> -1
( -> -1
sum -> 1
) -> -1
; -> -1
$ -> -1

```

### **ST:**

Data Structure: Balanced binary search tree, implemented using a red-black tree

```

0 -> 2
1 -> 6
currentNumber -> 3
i -> 5
n -> 4
sum -> 1

```

## **Program Error 1**

```

$
    let a: lint;
    input(a);

    let b: int;
    input(b);

```



```

let c: int;
input(c);

let max = a;

check(b ~ a) {
    max = b;
}

check(c > max) {
    max = c;
}

output(max);
$

```

### Output:

```

"Lexical Error: token 'lint' cannot be classified, line 2"
"Lexical Error: token '~' cannot be classified, line 13"

```

### PIF:

```

$ -> -1
let -> -1
a -> 1
: -> -1
; -> -1
input -> -1
( -> -1
a -> 1
) -> -1
; -> -1
let -> -1
b -> 2
: -> -1
int -> -1
; -> -1
input -> -1
( -> -1
b -> 2
) -> -1
; -> -1
let -> -1
c -> 3
: -> -1
int -> -1

```

```
; -> -1
input -> -1
( -> -1
c -> 3
) -> -1
; -> -1
let -> -1
max -> 4
= -> -1
a -> 1
; -> -1
check -> -1
( -> -1
b -> 2
a -> 1
) -> -1
{ -> -1
max -> 4
= -> -1
b -> 2
; -> -1
} -> -1
check -> -1
( -> -1
c -> 3
> -> -1
max -> 4
) -> -1
{ -> -1
max -> 4
= -> -1
c -> 3
; -> -1
} -> -1
output -> -1
( -> -1
max -> 4
) -> -1
; -> -1
$ -> -1
```

**ST:**

Data Structure: Balanced binary search tree, implemented using a red-black tree

a -> 1

b -> 2

c -> 3

max -> 4