Example: Pre-/post- Increment

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
    int *a = malloc (sizeof(int) * 3);
    int b = 3;
    int c = 4;

    a[0] = 1;
    a[1] = 2;
    a[2] = 3;

    *a = *(a++) * b + *(++a) * c;

    printf("%d \n", *a);
    return (0);
}
```

What is the output?
Example: Pre-/post- Increment

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
    int *a = malloc (sizeof(int) * 3);
    int b = 3;
    int c = 4;

    a[0] = 1;
    a[1] = 2;
    a[2] = 3;

    *a = *(a++) * b + *(++a) * c;
    printf("%d \n", *a);
    return (0);
}
```

- Execute all pre-increment operators prior to all other accesses, picking an order L-R or R-L in the expression.
- Execute pre-increment operators when the node is reached during tree traversal.
- Execute all post-increment operators after the expression has been calculated and assigned.
- Execute post-increment operators when the node is reached during tree traversal.
- Execute post-increment operators after the value has been computed, but before assignment.
Example: Pre-/post- Increment

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
    int *a = malloc (sizeof(int) * 3);
    int b = 3;
    int c = 4;

    a[0] = 1;
    a[1] = 2;
    a[2] = 3;

    *a = *(a++) * b + *(++a) * c;
    printf("%d \n", *a);
    return (0);
}
```

- C language specification does not define the behavior of the program where there is more than one pre- or post- increment operator in applied to a single variable within a single expression

- Compiler specific: 15 (gcc version 4.2.1)
  - Execute pre-increment operators when the node is reached during tree traversal.
  - Execute post-increment operators after the value has been computed, but before assignment.
Example: Lazy Evaluation

\[
\text{if } (a \lor (b \land c \land d))
\]

\begin{tabular}{|c|c|c|c|}
\hline
a & b & c & d \\
\hline
T & F & F & F \\
T & T & F & F \\
T & F & T & F \\
T & F & F & T \\
T & T & F & T \\
T & T & F & T \\
T & F & T & T \\
T & T & T & T \\
F & T & T & T \\
\hline
\end{tabular}

a || (b && c && d) is true

exp is true

exp is false

exp is false

exp is false

exp is true
Example: Lazy Evaluation

- Efficient
- The order of operations produces semantically different results

1. if (ptr != NULL && *ptr == a)
2. if (*ptr == a && ptr != NULL)

What are the results of 1 and 2 if ptr == NULL?
Semantics

- The **semantics** of a programming language is a precise definition of the **meaning** of any program that is syntactically and type-wise correct.

- Three ways to define the semantics:
  - **Operational semantics**: the meaning of a program is the output produced by a given architecture/compiler pair.
  - **Axiomatize semantics**: the meaning of a program can be rigorously proven by using a systematic logical argument.
  - **Denotational semantics**: the meaning of a program can be described as a collection of functions operating on the program state.

Why semantics are critical to the design of programming language?