SQC

BOLAÑOS GUERRERO JULIÁN
CASTILLO SOTO JACQUELINE
GALINDO REYES DANIEL ADRIAN
ISIDRO CASTRO KAREN CRISTINA
ZURITA CÁMARA JUAN PABLO

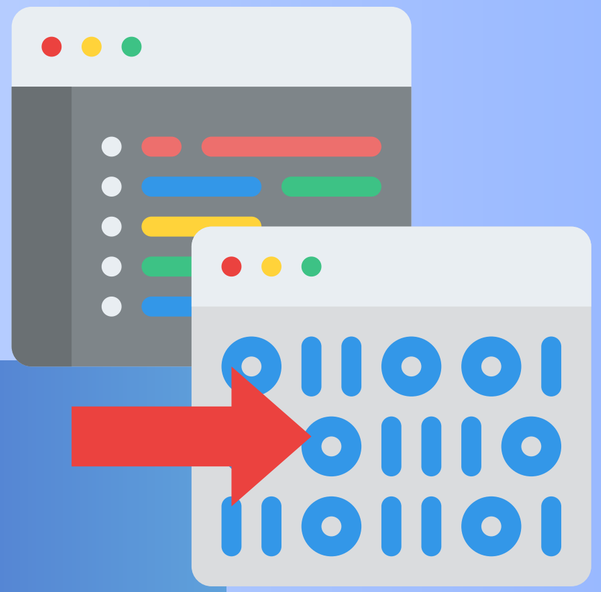
El futuro espera

SQC

Sky Quetz Compiler


A COMPILER DOESN'T JUST TRANSLATE CODE; TRANSFORM IDEAS INTO REALITY
Explore Now

WHY DO WE NEED A COMPILER?



WHAT ABOUT

TO HELP US?

WHY OUR COMPILER?

Lexical Analysis

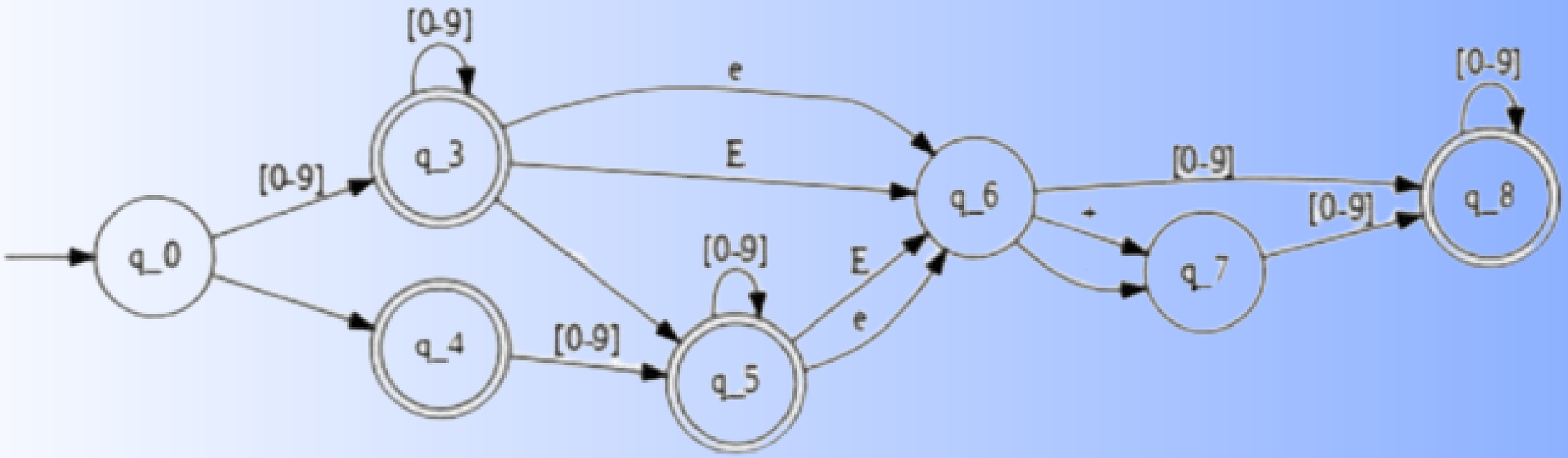
Syntactic analysis

Semantic analysis

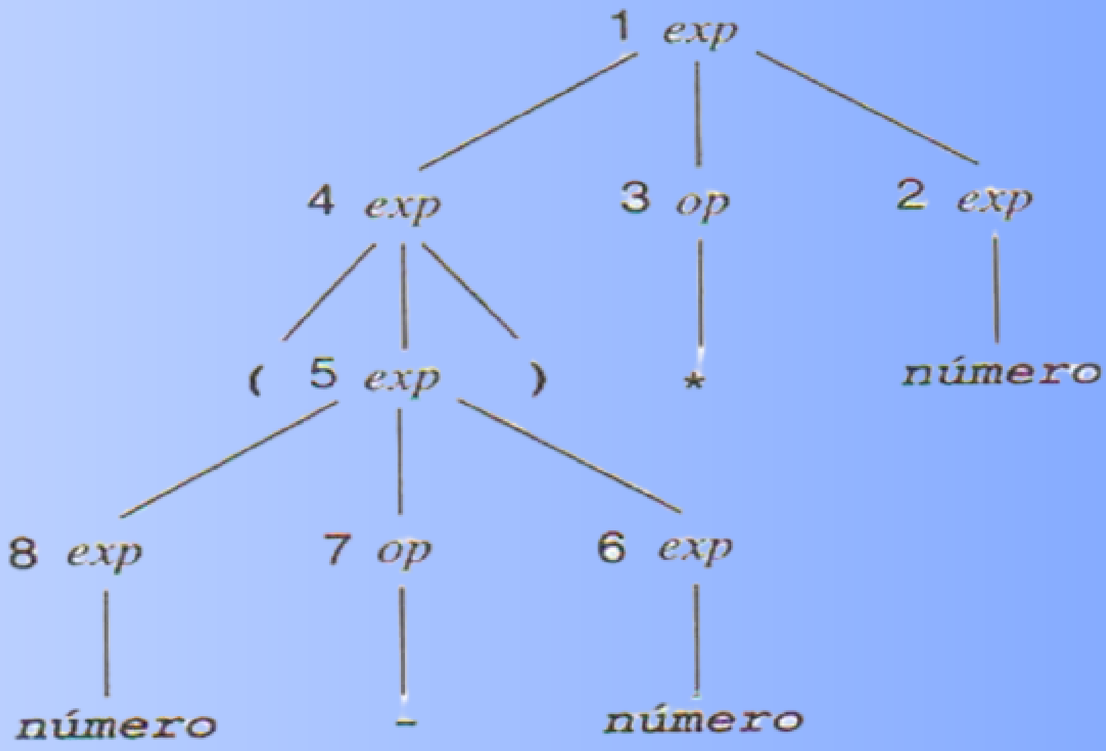
LEXICAL ANALYSIS

A lexical analyzer is the **first phase** of a programming language compiler or interpreter. Its main function is **to read the source code** and split it into basic units called **tokens**.

$$(([0\text{-}9]+ \text{ "." } [0\text{-}9]^*) \mid ([0\text{-}9]^* \text{ "." } [0\text{-}9]+)) (E [+\text{-}]? [0\text{-}9]) ?$$

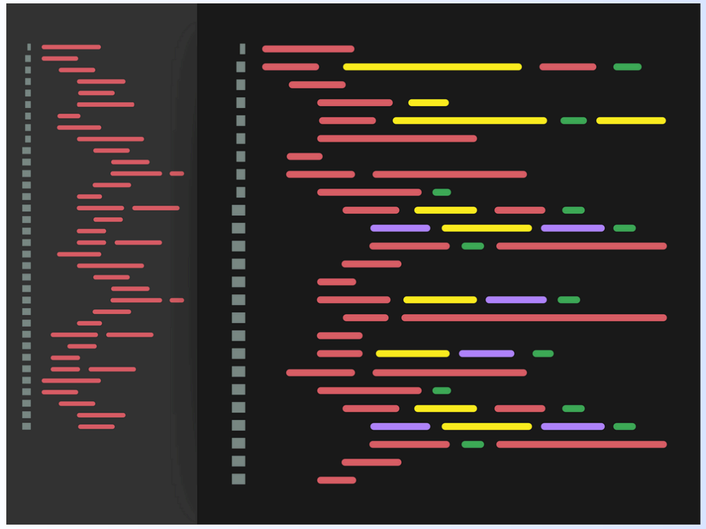


SYNTACTIC ANALYSIS



A parser is the **second phase** of a compiler or interpreter. Its main function is **to take the sequence of tokens** generated by the lexical analyzer and **organize them into a hierarchical structure that reflects the grammar of the language,** known as a **parse tree**.

SEMANTIC ANALYSIS



Semantic analysis is the **third phase** in the process of compiling or interpreting a programming language. Its main function is to verify that the instructions in the program are **semantically correct**, that is, that **they make sense according to the rules of the language and their logical context**.

PLY

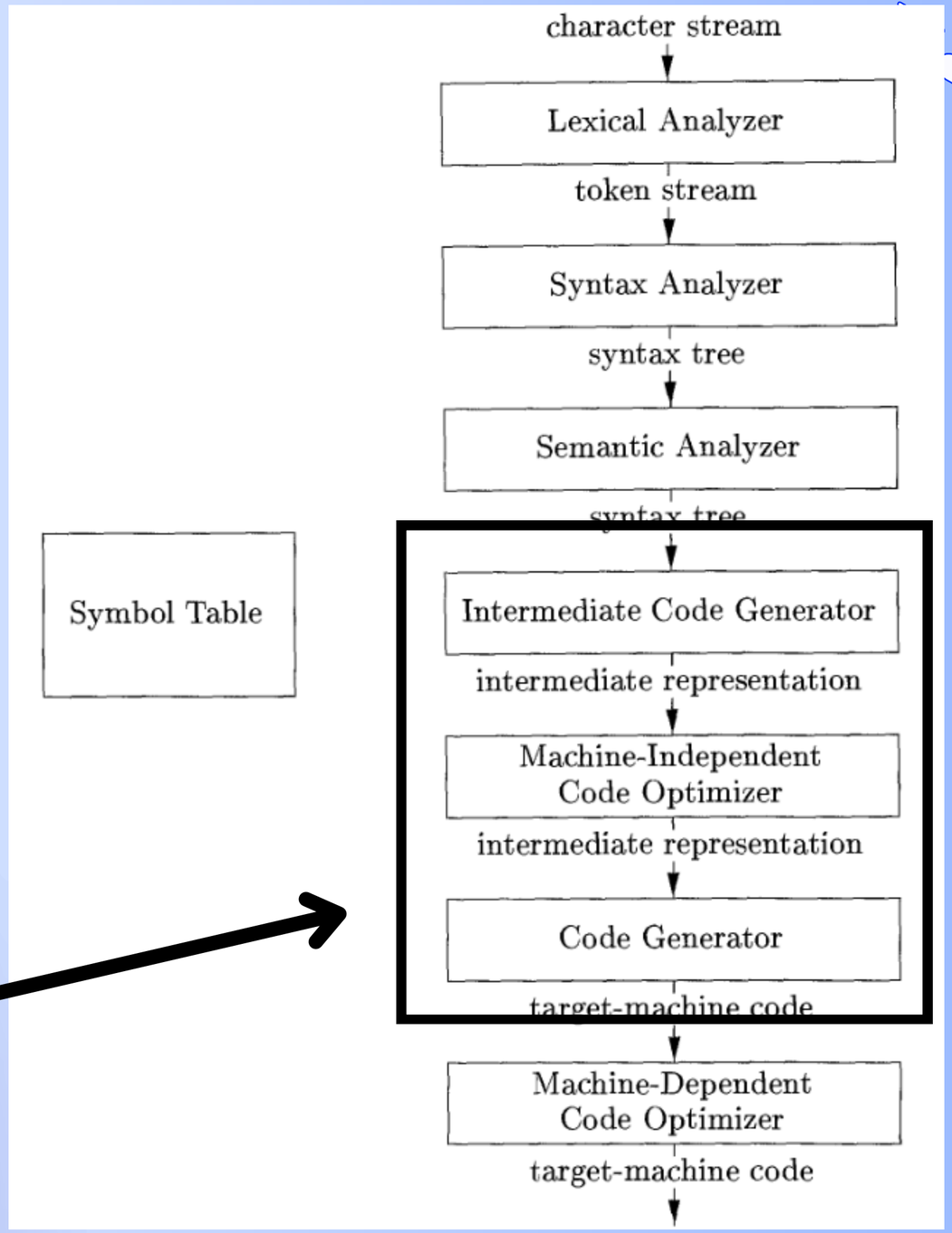
Lex.py

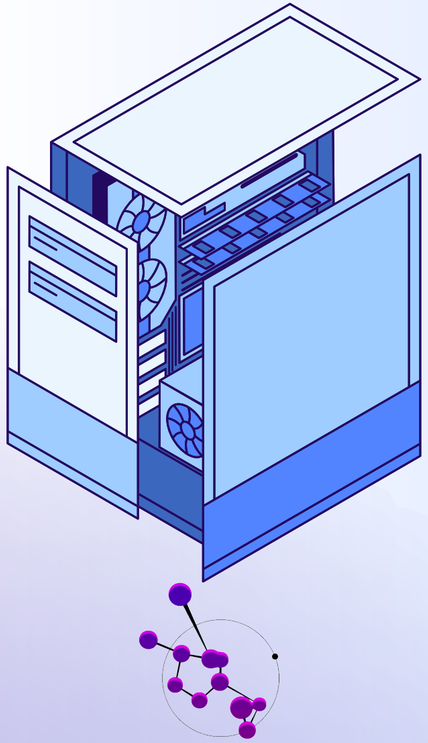
yacc.py

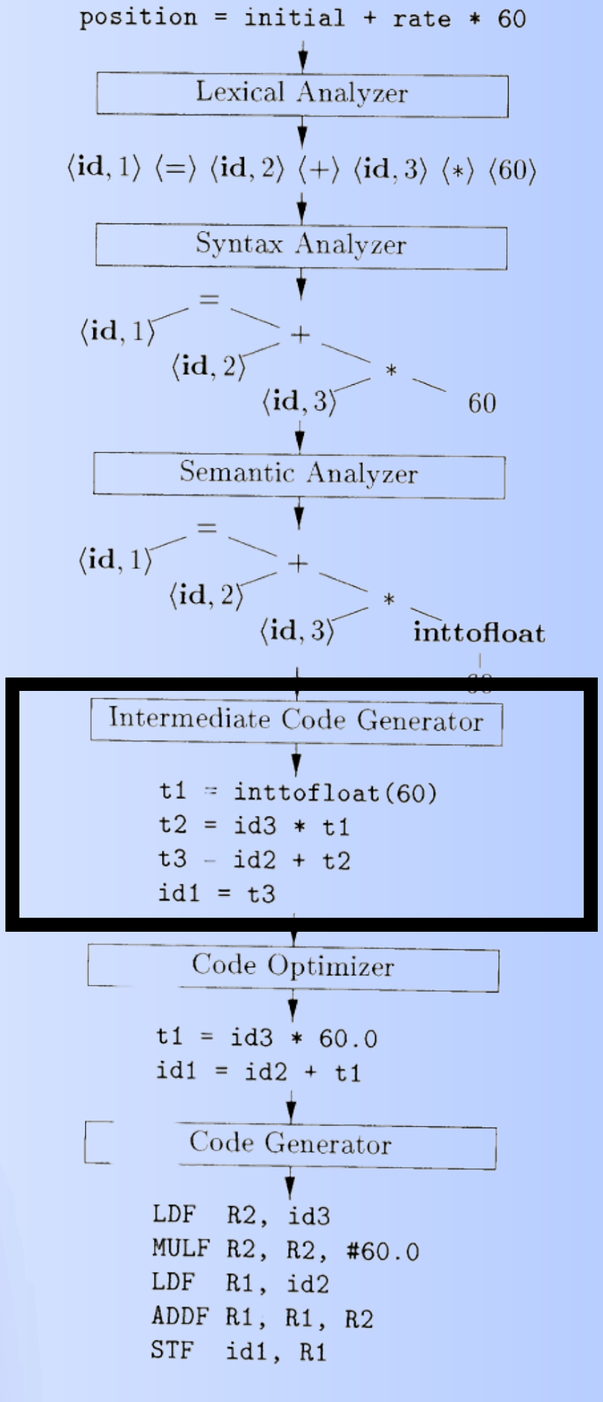
NOW WHERE ARE WE?

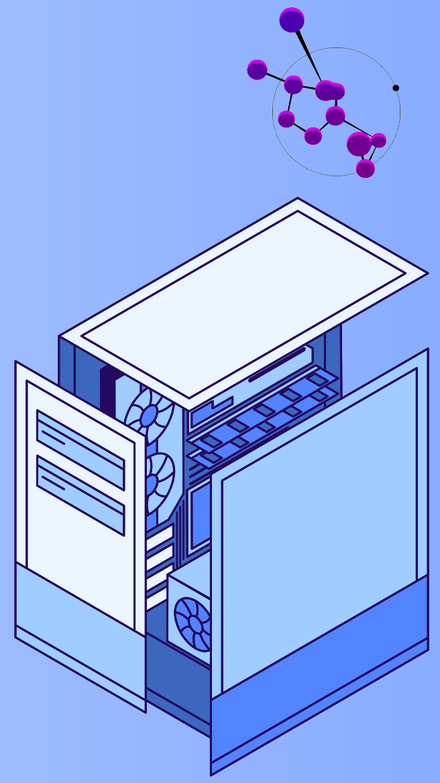
KEY CONCEPTS

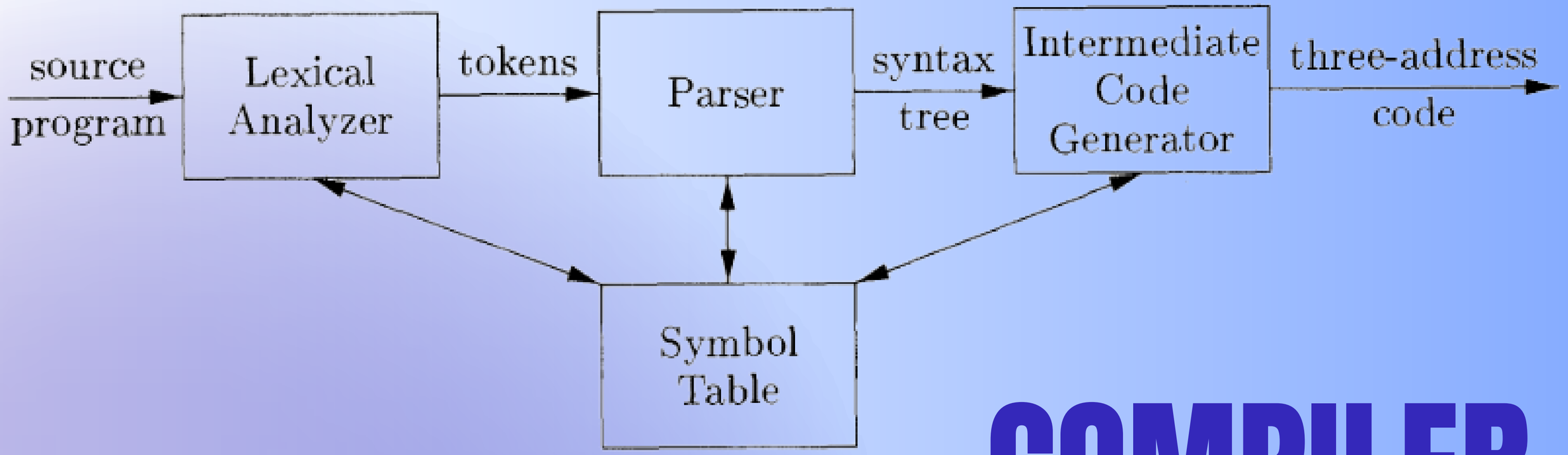
Area of Interest






position = initial + rate * 60
Lexical Analyzer
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
Syntax Analyzer
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
Semantic Analyzer
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
inttofloat
60
Intermediate Code Generator
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
Code Optimizer
t1 = id3 * 60.0
id1 = id2 + t1
Code Generator
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1





COMPILER

FRONT-END

INTERMIDIATE CODE

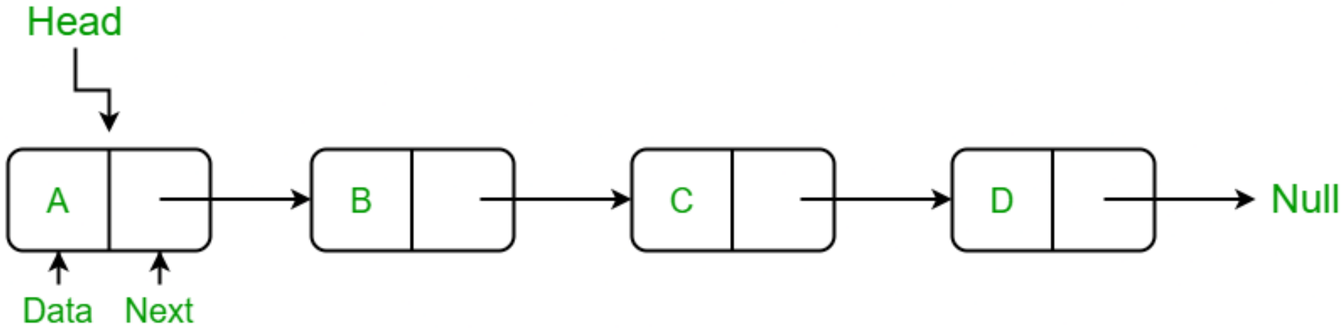
An **intermediate representation** (IR) is a a**bstract machine language** that can express the target-machine operations without committing to too much machine-specific detail. But it is also independent of the details of the source language.

Syntax Directed Translation Scheme With Actions Inside Productions

```
t1 =  enos c
t2 =  b * t1
t3 =  enos c
t4 =  b * t3
t5 =  t2 + t4
 a =  t5
```

	op	arg_1	arg_2	*resultado*
0	enos	c		t_1
1	*	b	t_1	t_2
2	enos	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
	...			

How we did It



THREE ADDRESS CODE

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

Quadruples
representation for
control flow
Backpatching

Code Optimization

Constant Folding

int a = 5 + 3; -> int a = 8;

Constant Propagation

```
int a = 10;
int b = a + 5;

int b = 15;
```

Algebraic Simplification

```
int x = a * 1; // Simplified
int y = a + 0; // Eliminated
```

Evaluation in Logical Operators

```
if (false && x > 5) { ... }

if (false) { ... }
```

EXAMPLE

Source Code

```
int main()
{
    int a = 4 + 56;
    float b= 4.5 + 3;
    char c = 'b';

    int e = b + 2;
    float g = a + e;

    int d = e / a;
}
```

Intermediate Code

```
a=60
b=7.5
c='b'
t1=b+2
e=t1
t1=a+e
g=float(t1)
t1=e/a
d=int(t1)
```

EXAMPLE

Source Code

```
int main()
{
    int a = 4;
    int b = 5;
    int c = 7;
    int d = 10;
    int t = -1;

    if (a < b || c > d)
    {
        t =1;


    }
}
```

1. if (a < b) **goto 5**
2. **goto 3**
3. if (c >d) **goto 5**
4. **goto 6**
5. t = 1
6.

Intermediate Code

```
t1=a<b
if(t1):
    t=1
t1=c>d
if(t1):
    t=1
```

EXAMPLE

Source Code

```
int main()
{
    int a = 4;
    int b = 5;
    int c = 7;
    int d = 10;
    int t = -1;

    if (a < b && c > d)
    {
        t = 1;
    }
}
```

Intermediate Code

1. if (a < b) **goto 3**
2. **goto 6**
3. if (c >d) **goto 5**
4. **goto 6**
5. t = 1
6.

```
t1=a<b
if(t1):
        t1=c>d
        if(t1):
                t=1
```

EXAMPLE

Source Code

```
int main()
{
    int a = 45;
    float b = 3.5;
    int c = (4*a)+(b+5);

    if (a > b && c > 60)
    {
        char p = 'b';
    }

    if (c == 56)
    {
        int i = 23;
    }
}
```

Intermediate Code

```
a=45
b=3.5
t1=4*a
t2=b+5
t3=t1+t2
c=int(t3)
t1=float(a)
t2=t1>b
if(t2):
        t1=c>60
        if(t1):
                p='b'
t1=c==56
if(t1):
        i=23
```

EXAMPLE

Source Code

```
int main()
{
    int a = 56;
    float b = 23;

   if ( 54 < 3)
   {
    a = 23;
   }

   if (3 > 2 && 110 > 99)
   {
    b = 34.4;
   }

   int x = 34;
}
```

Intermediate Code

```
a=56
b=23.0
b=34.4
x=34
```

EXAMPLE

Source Code

```c
int main()
{
    int a = 56;
    float b = 23;
    float c = 45;
    int d = 2;

    float e = (23*(3*d)+45/c)-(4*b/a)*(53/2+b*3/d);
}
```

Execution

```
PS C:\Users\juanz\Downloads\FinalCompiladores> python compiler.py test.c
PS C:\Users\juanz\Downloads\FinalCompiladores> python a.py
a = 56
b = 23.0
c = 45.0
d = 2
e = 38.785714285714285
```

Intermediate Code

```
a=56
print(f"a = {a}")
b=23.0
print(f"b = {b}")
c=45.0
print(f"c = {c}")
d=2
print(f"d = {d}")
t1=3*d
t2=23*t1
t1=45/c
t3=t2+t1
t1=4*b
t2=t1/a
t1=b*3
t4=t1/d
t1=26.5+t4
t4=t2*t1
t1=t3-t4
e=t1
print(f"e = {e}")
```

WHAT WE NEED?
WHAT WE GENERATE?

- **Files Generated by Using the PLY Library.**

parsetab.py

parser.out

- **Necessary Codes for Compiler Execution.**

compiler.py

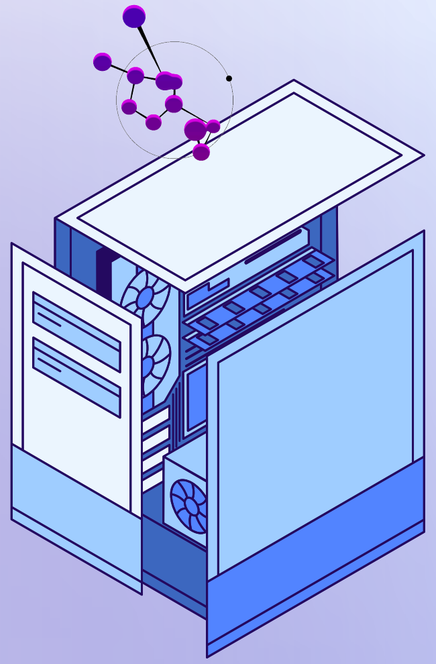
LinkedList.py

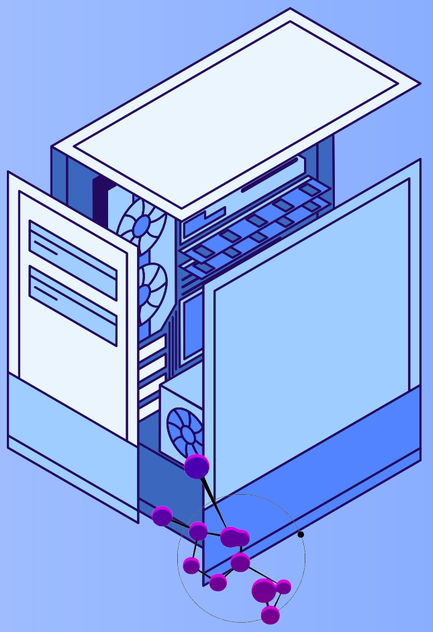
tokrules.py

- **Output File.**

a.py

DEMOSTRATION





WHICH COMPILER WILL MAKE YOUR DREAMS COME TRUE?