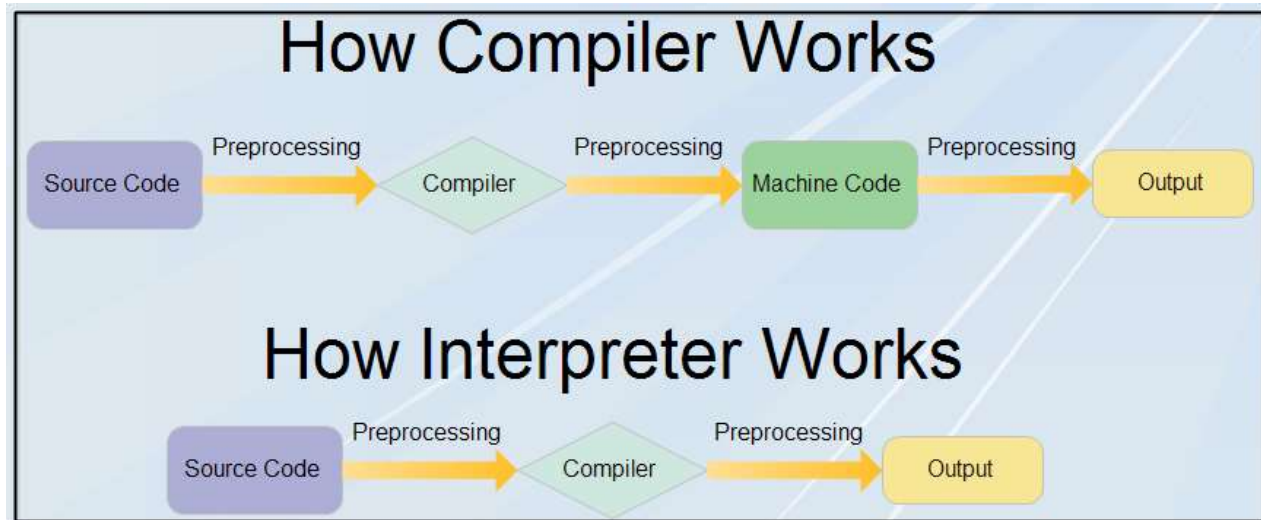
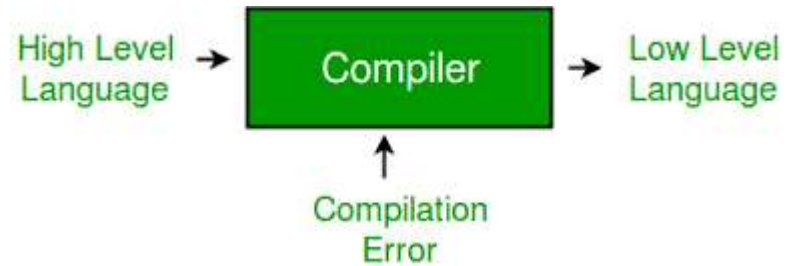




# UNIT II – Compiler & Lexical Analysis

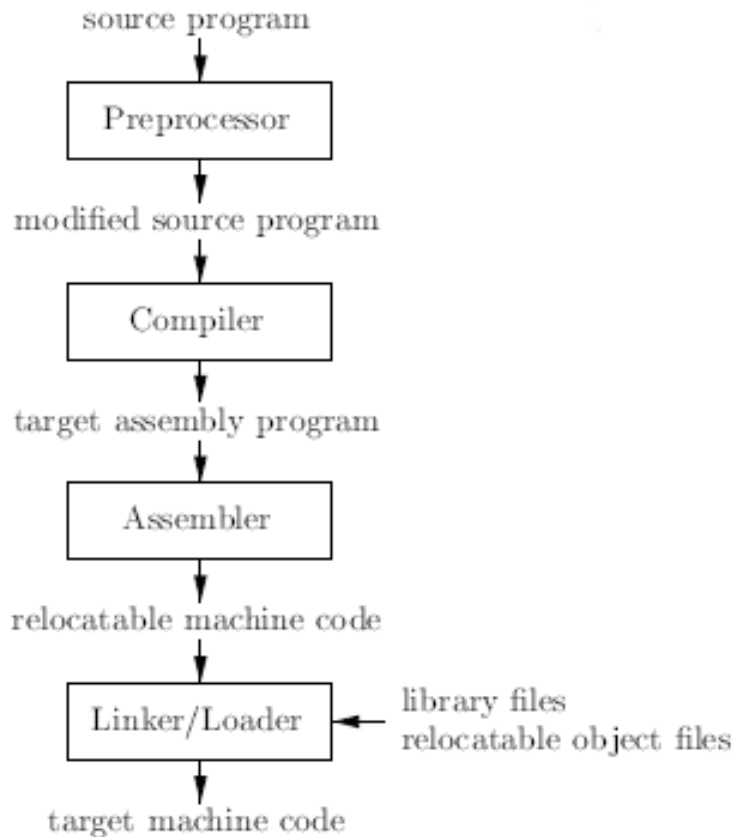


- System Software
  - Compiler
  - Interpreter

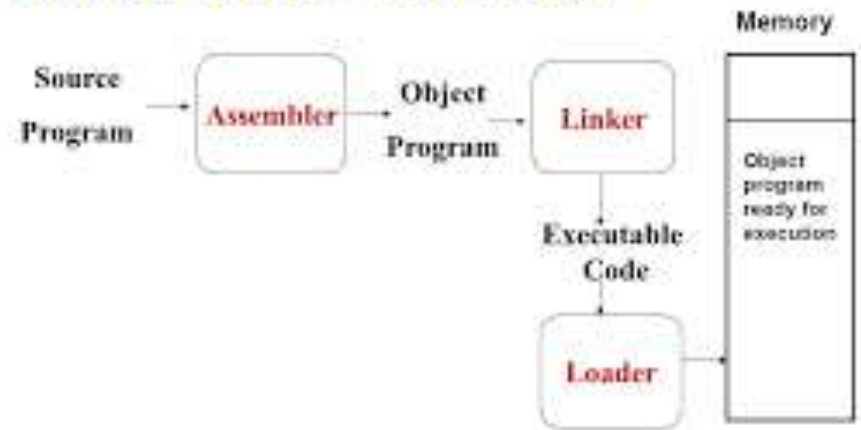




# Cousins of Compiler/language processing system



## Role of Loader and Linker





# Example

## High level language

```
#include<stdio.h>
#define a 10
main()
{
int a,b=20,c;
c=a+b;
printf("\n Add is %d", c);
}
```

## Assembly language

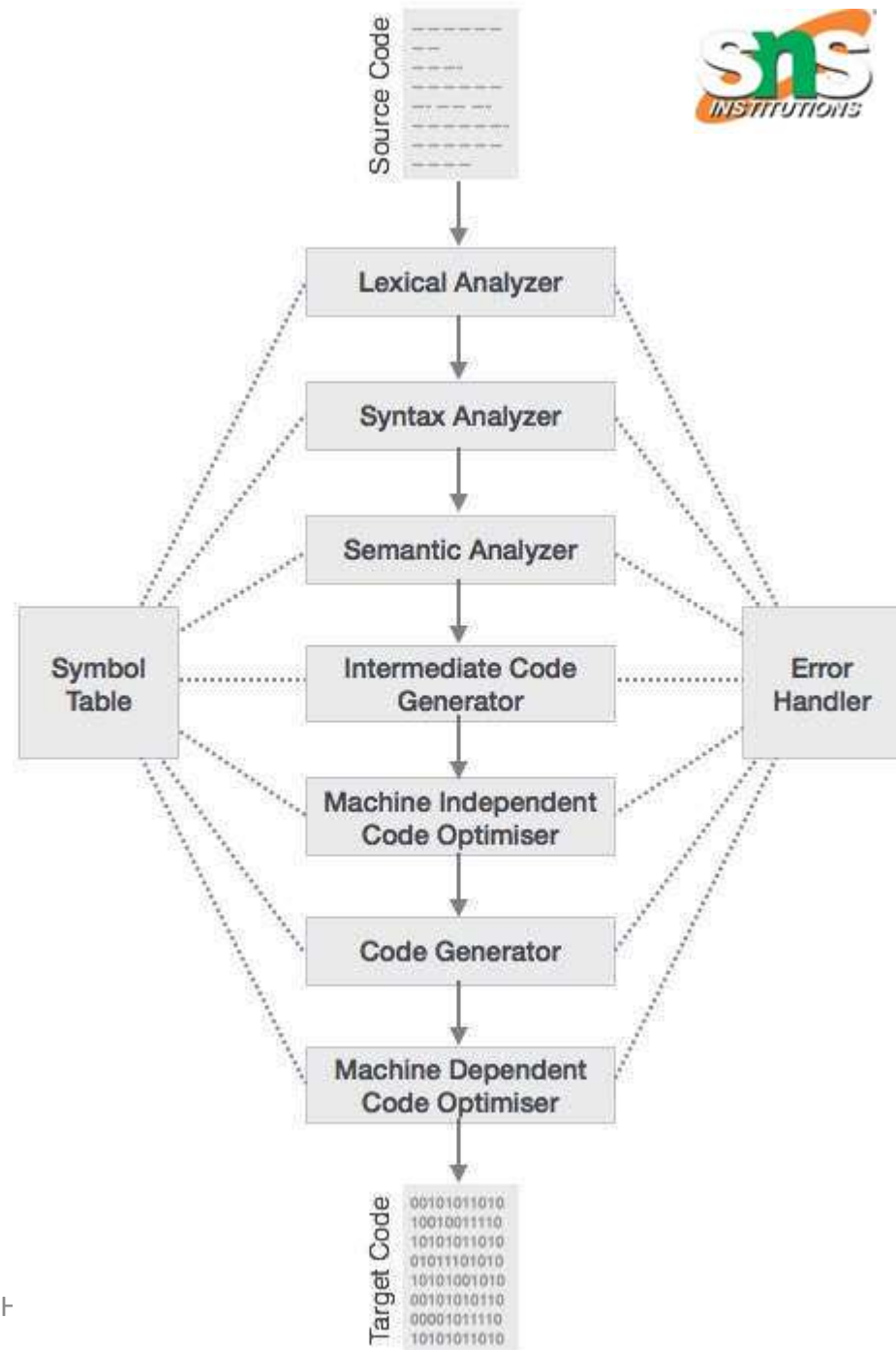
```
A 10
B 20
C=A+B
Display C
```





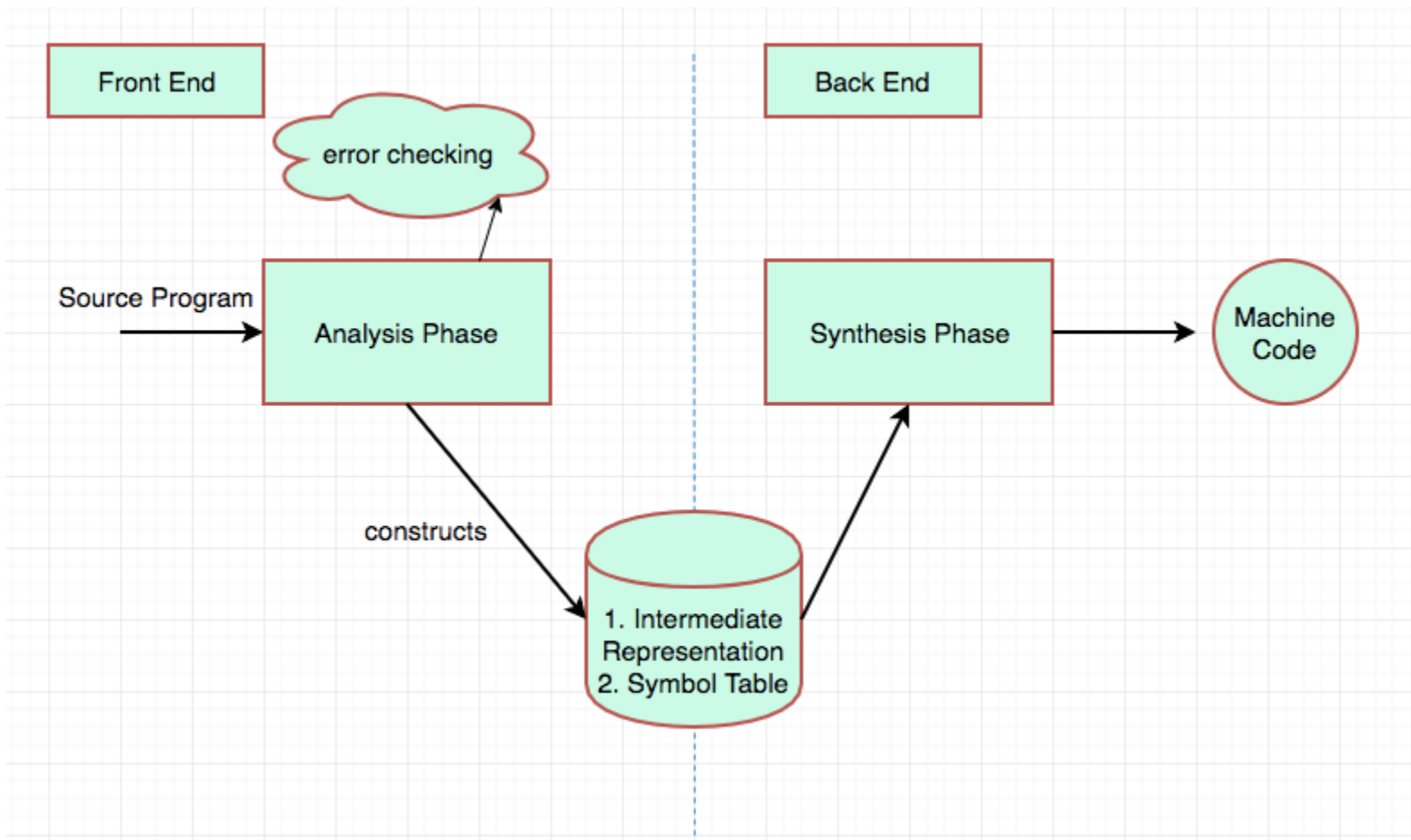
# Phases of Compiler

- COMPILING – 2 phases
  - Analysis (front – end)
    - Grammatical Structure
    - Intermediate code
    - Symbol table
  - Synthesis (back - end)
    - Target program





# Phases of Compiler – Analysis and Synthesis





# Lexical Analyzer/ scanning

- Text scanner
- Source code → stream of characters → meaningful sequence (lexemes)
- Lexeme → token → **<token-name, attribute-value>**
- Example
- Position=initial + rate\*60
- Sequence of tokens
- <id,1><=><id,2><+><id,3><\*><60>

Position	<id, 1>
=	<=>
Initial	<id, 2>
+	<+>
Rate	<id, 3>
*	<*>
60	<60>

# Syntax Analysis / parsing

- Intermediate representation – tree – syntax tree
- Intermediate node  $\rightarrow$  operation
- Child node  $\rightarrow$  arguments

1	position	...
2	initial	...
3	rate	...

SYMBOL TABLE

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



# Semantic Analysis

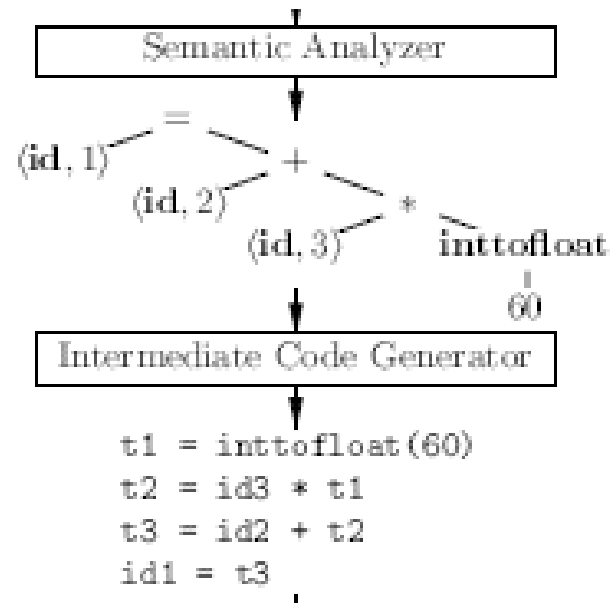
- Semantic Analysis
  - Intermediate representation
  - Symbol table
  - Semantic consistency of source program
  - ***Type checking***
  - Language specification → type conversion (coercions)
  - Example
    - Position, initial, rate ← floating point
    - 60 ← inttofloat





# Intermediate code generation

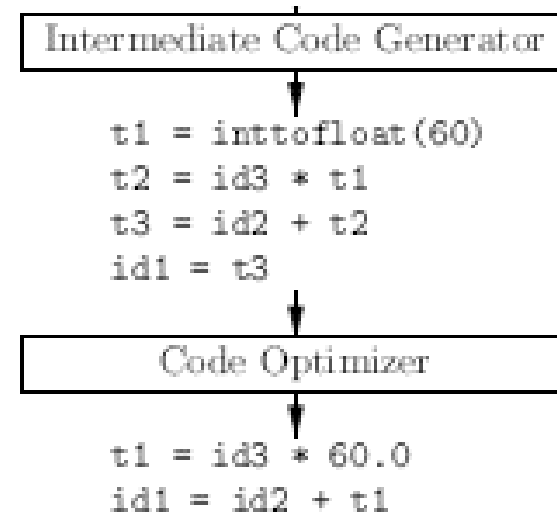
- Low-level/ machine-like intermediate representation





# Code optimization

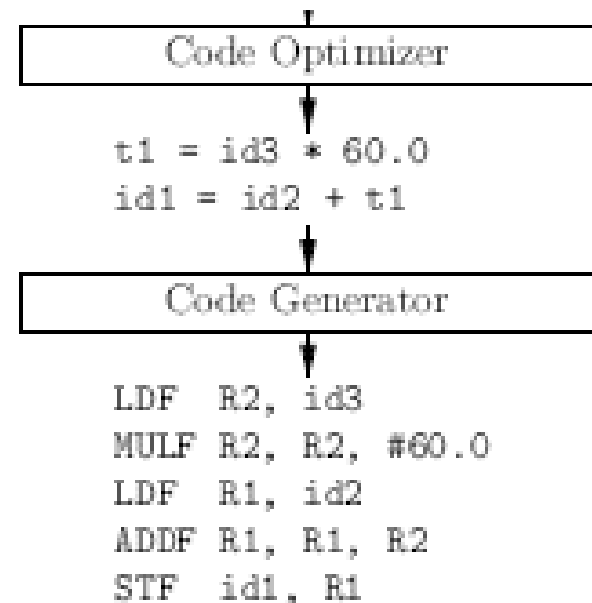
- Improves the intermediate representation to enhance the target code generation
- **Good target code**
- Code generation
  - Each operator  $\rightarrow$  single instruction
- Code optimization
  - `inttofloat`  $\leftarrow$  compile time (`60`  $\rightarrow$  `60.0`)
  - Assignment operation





# Code generation

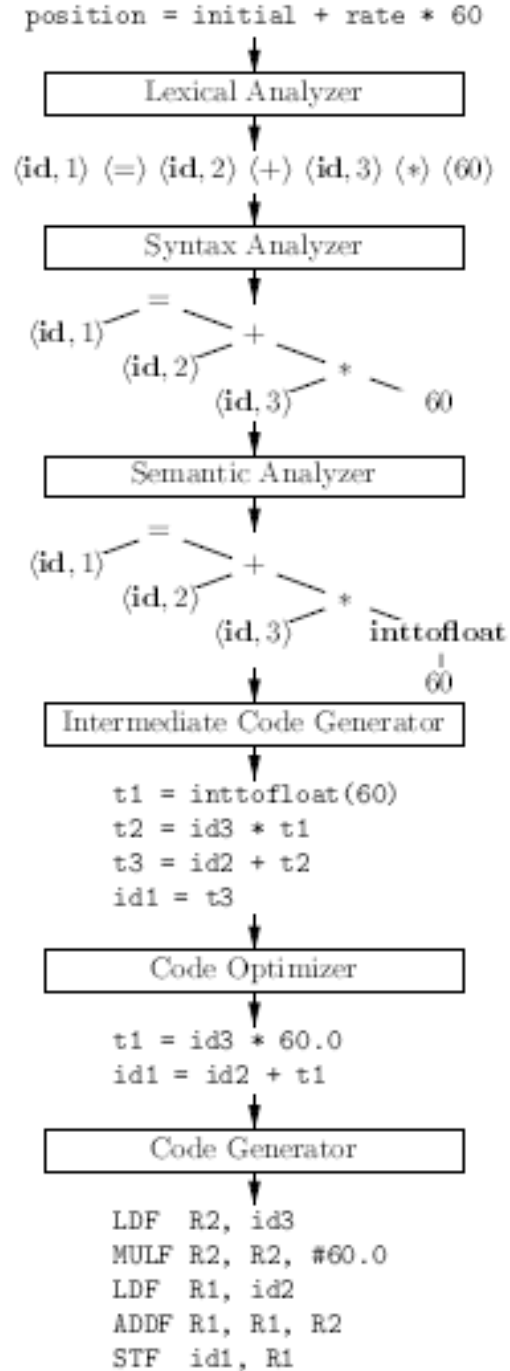
- Target code (for each variable)
  - Registers
  - Memory location
  - *LDF (load floating point value)*
- Example
- $\text{Position} = \text{initial} + \text{rate} * 60$
- Sequence of tokens
- $\langle \text{id}, 1 \rangle \langle = \rangle \langle \text{id}, 2 \rangle \langle + \rangle \langle \text{id}, 3 \rangle \langle * \rangle \langle 60 \rangle$ 
  - $R2 = \text{RATE} = \text{ID3} = 2.0$
  - $R2 = 120.0$
  - $R1 = \text{INITIAL} = 5.0$
  - $R1 = 125.0 \leftarrow \text{POSITION} \leftarrow \text{ID1}$





1	position	...
2	initial	...
3	rate	...

SYMBOL TABLE





## *Grouping of phases into passes*

- One pass
  - Lexical analysis
  - Syntax analysis
  - Semantic analysis
  - Intermediate code generation
- Pass → Code optimization (optional)
- Pass → Code generation