CS327: Compilers

Lecture 1

Shouvick Mondal

shouvick.mondal@iitgn.ac.in

January 2026

Learning Outcomes

Students should understand the basics of programming language design, as well as the execution environment. Students should be able to build a compiler for a subset of a realistic programming language (C/C++/Java), implementing the basic building blocks of a compiler.

For each student, there will be TWO lectures, and ONE lab session per week. The L-T-P-C for this course is (3-0-1.5-5) which means a weekly load of: 3 hrs of lecture, and 1.5 hrs of lab, and overall 5 credits for this course. Prerequisites: (i) Data Structures and Algorithms-I (ES 242), (ii) Basics of Regular Languages and Context Free Languages.

About the Course

Instructor: **Shouvick Mondal**

TAs: Devansh Yadav, Madhav Raj Verma

Course email (*all announcements will be made*):
cs327_compilers-jan-may-2026.pvtgroup@iitgn.ac.in

Google classroom (*all lab assignments will be posted*):
https://classroom.google.com/c/ODMxMzQ4Mjk3MzQ5?cjc=kfbqvhwv

GitHub: **https://github.com/SET-IITGN/CS-327-Compilers**

Lectures: (Slot **F1+G2**): Mon + Fri, 11:30 AM – 12:50 PM @ AB1/101 (SCSLT)

Lab sessions: (Slot **F2**): Thu, 11:30 AM – 12:50 PM

- Batch 1.1 (@ AB10/203)
- Batch 1.2 (@ AB10/204)

Overall course load: {**23L** (x1)} + {**14P** (x1)}

- **Blue (lec.)**: AB1/101 (150x)
- **Red (lab.)**: AB10/203 (70x), AB10/204 (70x)

Timeline:
Excluding holidays, exam weeks, and breaks.

Month	Day
January	5, ~~9,~~ 12, 16, 19, 23, 30 {8, 15} **(A1)**, {22, 29
February	2, 6, 9, 13, 16 5, 12} **(A2)**, {19
March	9, 13, 16, 20, 23, 27, 30 12, 19, 26} **(A3)**
April	6, 10, 13, 17, 20 {2, 9, 16, 23} **(A4)**

Course Evaluation

relative grading

- [A1: 7 marks] Lab Assignment #1 [Deadline: Jan 20]
- [A2: 13 marks] Lab Assignment #2 [Deadline: Feb 17]
- [E1: 20 marks] Exam I [Feb 20 – 27]
- [A3: 20 marks] Lab Assignment #3 [Deadline: Mar 31]
- [A4: 20 marks] Lab Assignment #4 [Deadline: Apr 23]
- [E2: 15 marks] Exam II [Apr 24 – May 01]
- [B: 5 marks] Attendance (TAs must record attendance in physical signature for both lectures and lab. sessions)

There will be no makeup exams, re-exams, or reschedules.
There will be no further changes to the lecture/lab slots.
All deadlines are firm, set at 23:59:59 hrs IST

Software Tools to be used: lex, yacc, gcc, gas, and other relevant tools as and when necessary.

***Lab Assignments will be linked**, i.e., built upon the previous one, and should be performed in groups of size at most three. Each group will develop a compiler for a custom language of their choice, or a subset of any popular programming language. Compilation phases collectively implemented (spanning across all four lab assignments) need not be exhaustive.*

Student Honour Code: [https://iitgn.ac.in/students/honourcode]

Course Contents

(tentative and subject to change at the sole discretion of the instructor)

1. Overview of Compiler; Brief discussion on Assembler, Linker, Loader
2. Lexical Analysis; Syntax Analysis: Parsers, Top-down, Bottom-up, LR Parsers, Parser generators;
3. Syntax directed translation; Intermediate code generation and type checking; Run-time Environments; Type checking;
4. Code generation and optimization (data flow analysis, instruction optimization, loop optimization, memory optimization etc).
5. Tools: Flex, Bison (or similar software).

(GATE CS 2026 Syllabus) *for those interested...*

Section 7: Compiler Design

Lexical analysis, parsing, syntax-directed translation. Runtime environments. Intermediate code generation. Local optimization, Data flow analyses: constant propagation, liveness analysis, common sub expression elimination.

Texts/References to be followed

1. Alfred Aho, Monica Lam, Ravi Sethi, Jeff Ullman. *Compilers: Principles, Techniques, and Tools*, Pearson Education India; 2nd Edition, 2013.
2. Steven Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann; 1st edition 1997.
3. John R. Levine, Tony Mason, and Doug Brown, *Lex & Yacc* (2nd ed.). O'Reilly & Associates, Inc., USA, 1992.
4. Jonathan Bartlett, *Programming from the Ground Up*, 2003.
5. Wikibooks, *x86 Assembly*, *GNU assembly syntax* (online).

What is written here?

المعهد الهندي للتكنولوجيا جانديناجار بالاج، جانديناجار -382055 الهند، ولاية غوجارات ،

Индийский технологический институт Гандинагар Паладж, Гандинагар - 382055, Гуджарат, Индия

རྒྱ་གར་གྱི་ལག་རྩལ་སློབ་གྲྭ་གནས་པས་ལྷན་རྫོགས་ཁྱིམ་ - 382055, གུ་ཇ་རཏ།

ᏥᏅᏯᏗ ᏬᏅᏥᏅᏁᏗ ᏏᏚᏂ ᏏᏝᏍ ᏋᏗᏐ ᏝᏍᏍ ᏍᏗ ᏍᏗ ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ, ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ - ᏕᏫᏫᏐᏫᏫ, ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ, ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ - ᏕᏫᏫᏐᏫᏫ, ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ, ᏗᏅᏗ ᏅᏝᏍᏍᏗᏍᏗ - ᏕᏫᏫᏐᏫᏫ

Languages

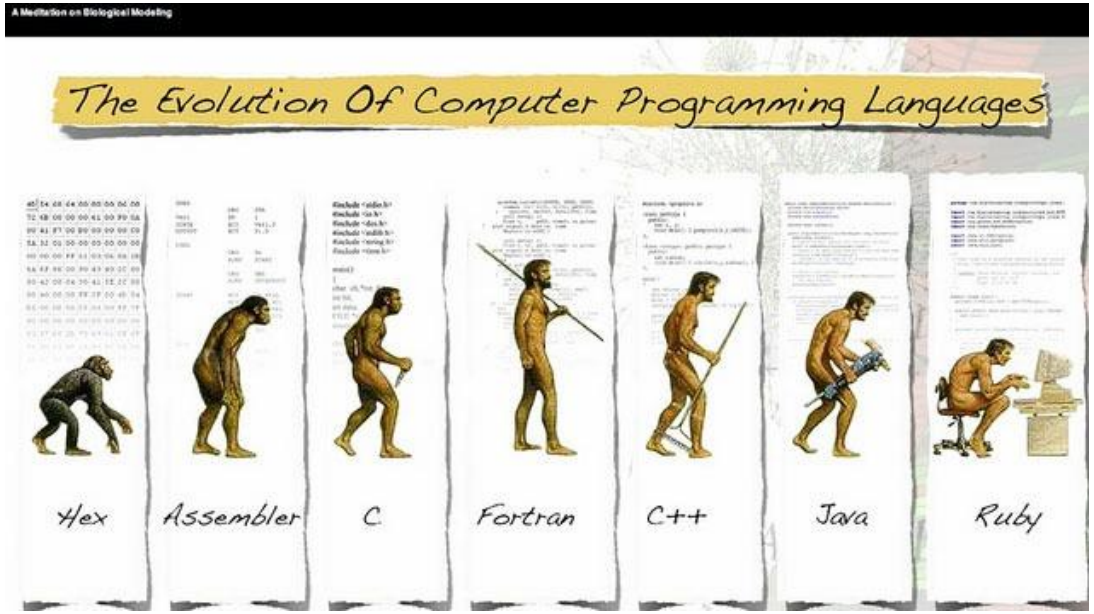
Wall of Love, Paris
Source: google images

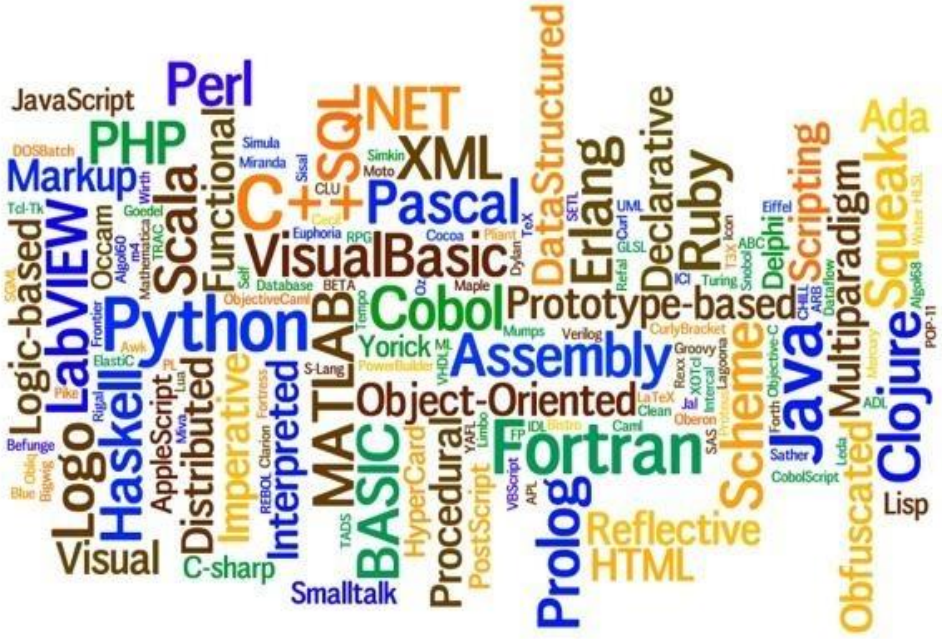
Languages

- Why do we need languages?
  - Humans communicate
    - sign language, body language, braille
  - Birds communicate
    - mark territories, attract for mating, warn danger
  - Animals communicate
    - mark territories, convey need, preparation for attack
  - Aliens?

Programming Languages

- Why do we need programming languages?
- And why so many?
  - What is your first language?
  - عربي, हिंदी, বাংলা, ગુજરાતી, தமிழ், తెలుగు
  - C, C++, Java, Python





Source: google images

Programming Languages

- There are some special purpose languages
- And why so many?
  - HTML for webpages
  - LaTeX for document formatting
  - `ps` for postscript files; `sql`, VHDL
  - Shell scripts, `awk`, `grep`, `sed`
  - Makefile has a language; `smtp`
  - How about google search?
    - `filetype:pdf`, `link:www.iitgn.ac.in`
  - Gmail: `in:unread`, `in:starred`
  - vi: `:se ai`, `:wq`, `:se ft=c`
  - What about `ls -l`, `ls -Ri`, `ls --color`, `ls -1 dir1 dir2` ?

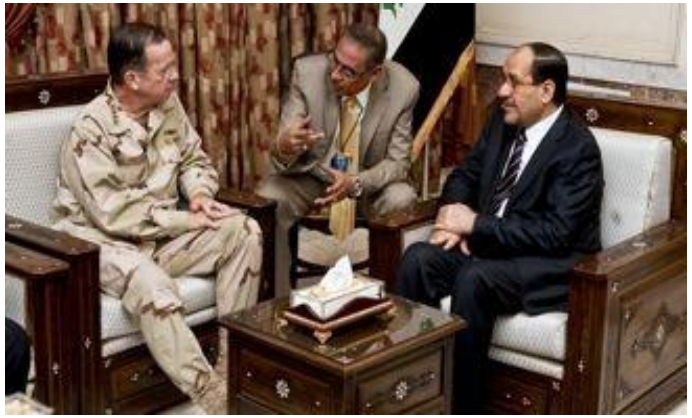
Language is for Communication

- Using mobile buttons
- Using ipad touch
- Using a calculator
- Using a fan switch
- Using a remote for projector / laser

... some of these are not programmable today.
They have a limited abstraction.
We will work with programming languages.

Compiler

- When do we need a compiler?
  - 私は日本語を知っています
  - मुझे हिंदी आता है
  - أنا أعرف اللغة العربية
  - আমি বাংলা জানি
  - I know English



... some of these are not programmable today.
They have a limited abstraction.
We will work with programming languages.

Source: google images

Jobs of a Compiler

- Translate: input language, output language
- Maintain correctness
  - चाचाजी अजमेर गए
  - Uncle died today
- Be efficient
  - Why are you laughing?
  - I understood yesterday's joke.
- Generate a good language
  - I got books but more than that I got your letter
  - मैं किताबें, लेकिन मैं अपने पत्र मिला है कि अधिक से अधिक मिला है

Good Language (2015)

I got books but more than that I got your letter.
मैं किताबें, लेकिन मैं अपने पत्र मिला है कि अधिक से अधिक मिला है।

I have books, but I got your letter got more than that.
मैं किताबें हैं, लेकिन मैं अपने पत्र है कि अधिक से अधिक मिला।

I have books, but that's more than I got your letter.
मैं किताबें हैं, लेकिन लगता है कि मैं अपने पत्र मिला है की तुलना में अधिक है।

I have books, but I have received your letter is more than.
मैं किताबें हैं, लेकिन मैं अपने पत्र की तुलना में अधिक है प्राप्त हुआ है।

Good Language (2018)

I got books but more than that I got your letter.
मुझे किताबें मिलीं लेकिन उससे ज्यादा मुझे आपका पत्र मिला।

I got the books but I received your letter more than that.
मुझे किताबें मिलीं लेकिन मुझे उससे अधिक पत्र मिला।

I got books but I received more letters from him.
मुझे किताबें मिलीं लेकिन मुझे उससे ज्यादा पत्र मिले।

I got the books but I received more letters from him.
मुझे किताबें मिलीं लेकिन मुझे उससे अधिक पत्र प्राप्त हुए।

Good Language (2020)

I got books but more than that I got your letter.
मुझे किताबें मिलीं लेकिन उससे ज्यादा मुझे आपका पत्र मिला।

I received books but more than that I received your letter.
मुझे किताबें मिलीं, लेकिन इससे ज्यादा मुझे आपका पत्र मिला।

I received books, but more than that I received your letter.
मुझे किताबें मिलीं, लेकिन इससे भी ज्यादा मुझे आपका पत्र मिला।

I received books, but more than that I received your letter.
मुझे किताबें मिलीं, लेकिन इससे भी ज्यादा मुझे आपका पत्र मिला।

Good Language (2024)

I got books but more than that I got your letter.
मुझे किताबें तो मिलीं हीं, उससे भी अधिक मुझे आपका पत्र मिला।

Not only did I get the books, but more than that I got your letter.
मुझे न केवल किताबें मिलीं, बल्कि उससे भी अधिक मुझे आपका पत्र मिला।

Not only did I get the books, but more than that I got your letter.
मुझे न केवल किताबें मिलीं, बल्कि उससे भी अधिक मुझे आपका पत्र मिला।

Good Language (2026)

I got books but more than that I got your letter.
मुझे किताबें मिलीं, लेकिन उससे भी ज्यादा मुझे आपका पत्र मिला।

Not only did I get the books, but more than that I got your letter.
मुझे न केवल किताबें मिलीं, बल्कि उससे भी बढ़कर आपका पत्र मिला।

Not only did I get the books, but more than that I got your letter.
मुझे न केवल किताबें मिलीं, बल्कि उससे भी बढ़कर मुझे आपका पत्र मिला।

Compilers work with Strings

- Characters, words / tokens, sentences, programs
- Fun with strings
  - quick brown fox jumps over the lazy dog
  - stewardesses
  - typewriter
  - skepticisms
  - quine

Programs as Data

```
char*f="char*f=%c%s%c;main(){printf(f,34,f,34,10);}%c";main(){printf(f,34,f,34,10);}
```

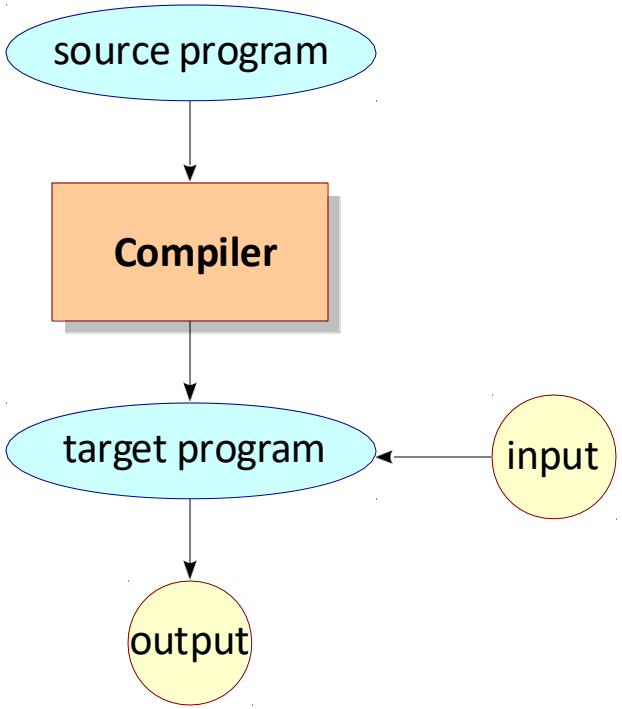
Why should we Design a language?

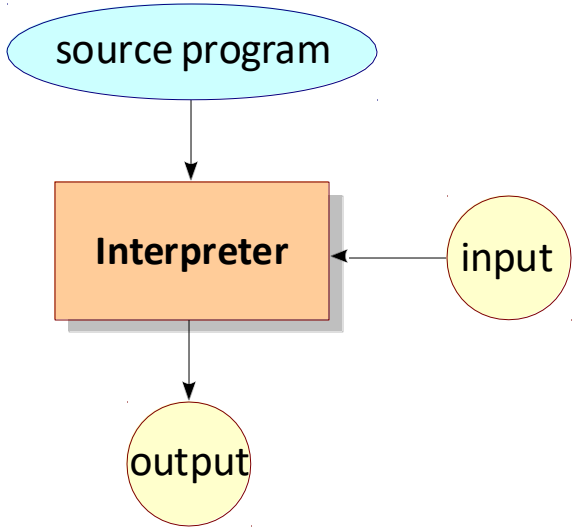
- Language matters!
  - A: Would you accept a gamble that offers a 10% chance to win $95 and a 90% chance to lose $5?
  - B: Would you pay $5 to participate in a lottery that offers a 10% chance to win $100 and a 90% chance to win nothing.
- Outcomes of a treatment for lung cancer. Two descriptions were:
  - C: The one-month survival rate is 90%.
  - D: There are 10% deaths in the first month.
- B fetched many more positives. 84% physicians chose option C.

Source: Thinking Fast and Slow, Daniel Kahneman

Classwork

Write a program that takes a function name as input and calls that function.





What does this mean?

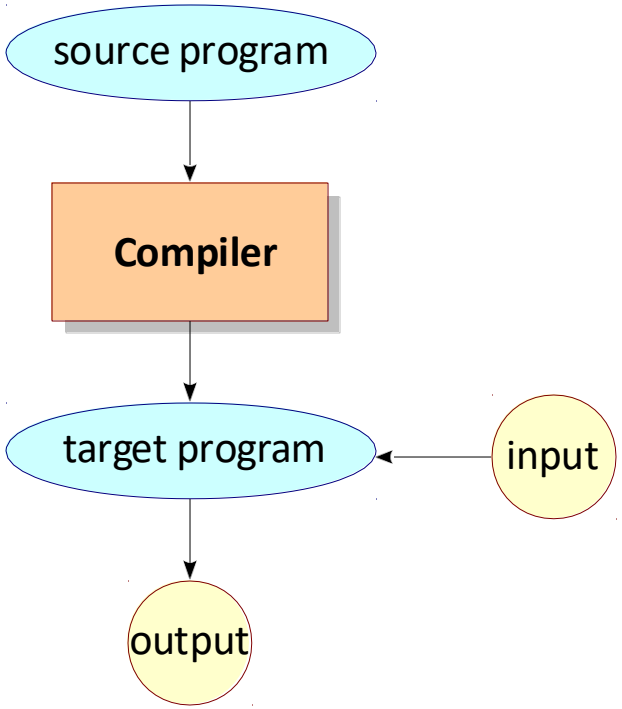
- You may be able to do the following with interpreters.

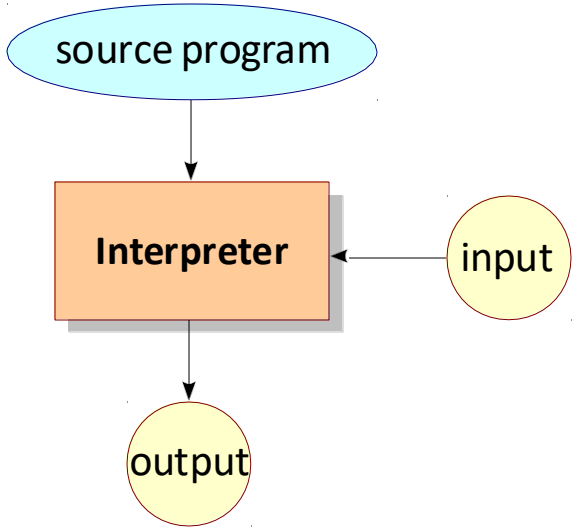
```
$x = 0; $y = 0;
echo "Enter a variable name: ";
$line = fgets(STDIN);
$line = trim($line);
${$line} = 20;
echo "x=$x, y=$y\n";
```

How about C?

```
void main() {
int x = 0, y = 0;
#include "/dev/stdin"
        = 10;
        printf("x = %d, y = %d\n", x, y);
}
```

Everything is fair in love, war, and C.





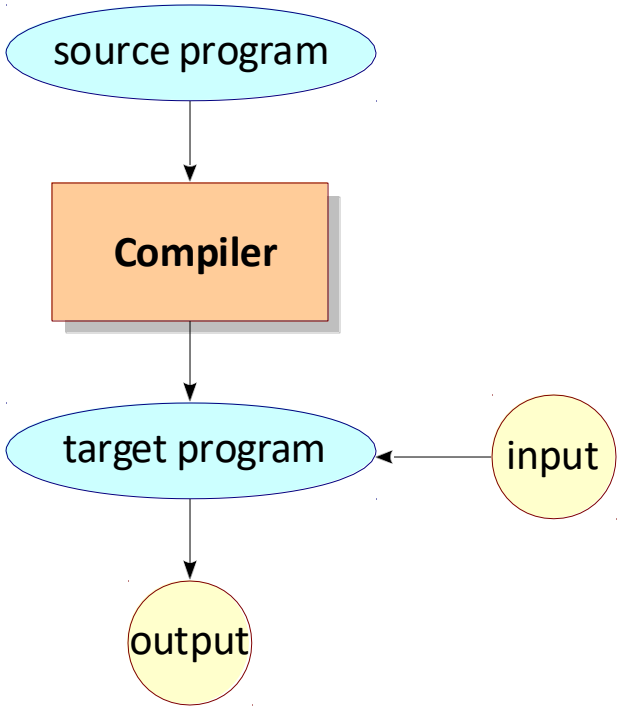
What does this mean?

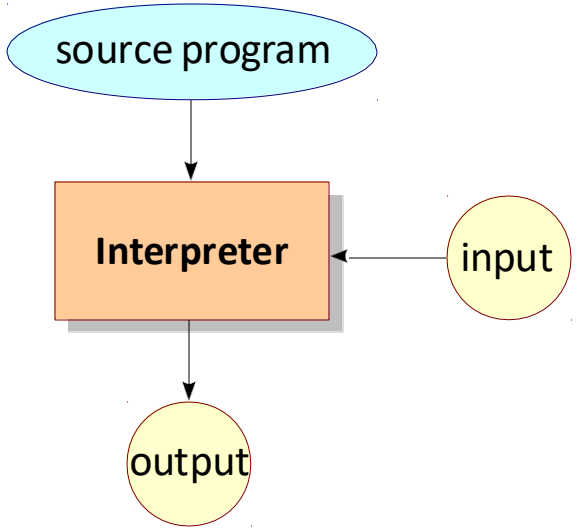
- You may be able to do the following with compilers.

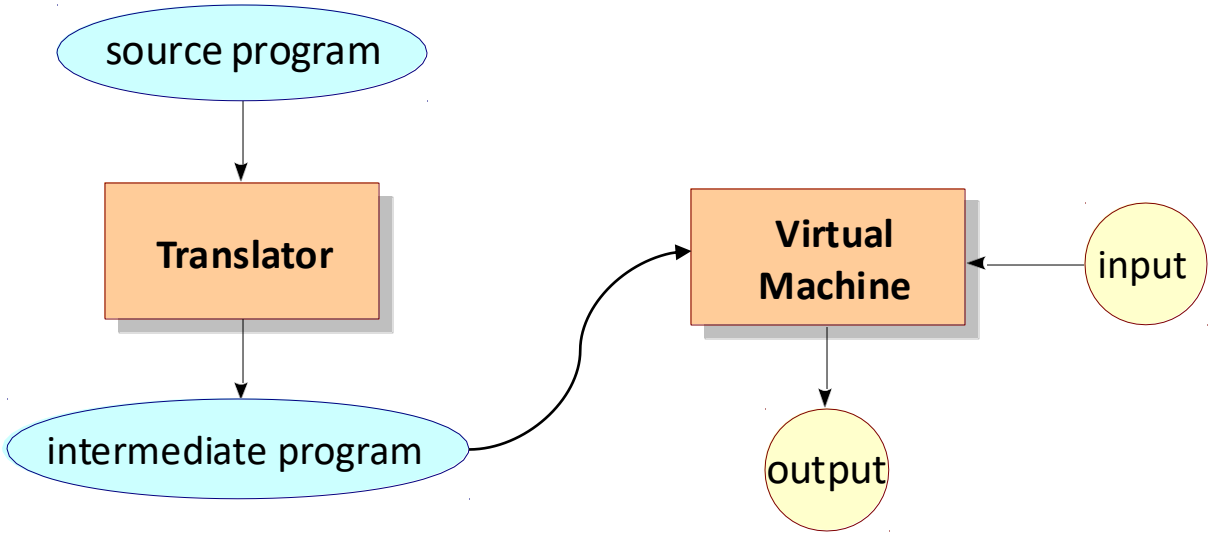
```
x += 2;
x += 2;
--x;
x += 5;
++x;
x += 9
```

is equivalent to ⇒

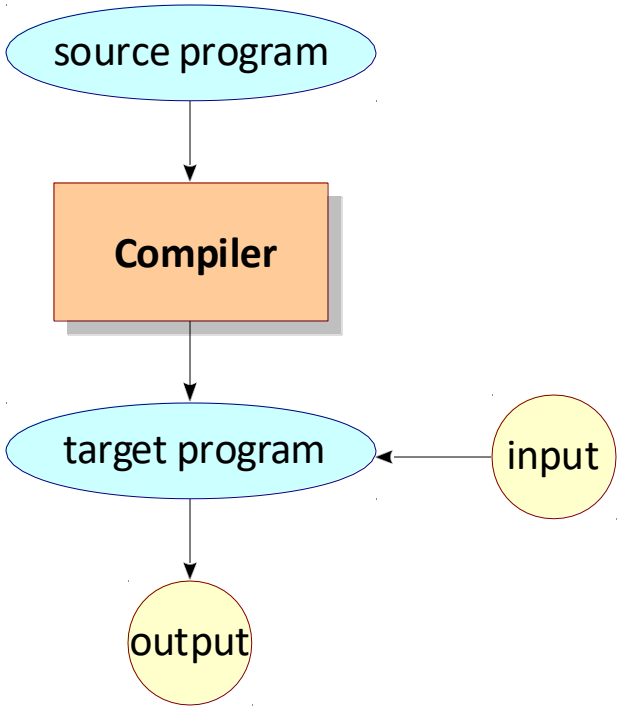
```
x += 18
```

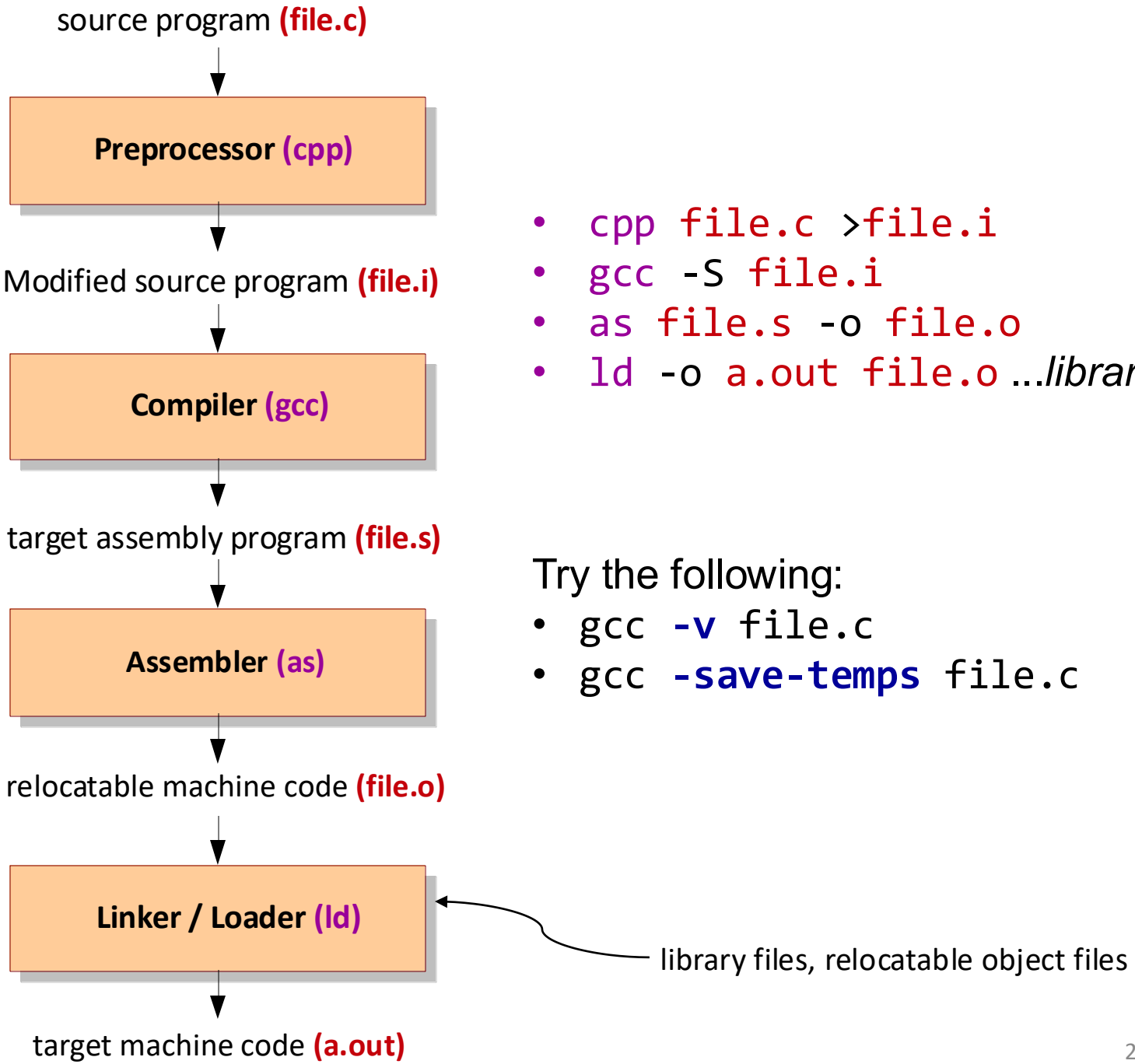

source program
Compiler
target program
input
output


source program
Interpreter
input
output


source program
Translator
intermediate program
Virtual Machine
input
output

```
:~$ gcc file.c
:~$ ./a.out
```





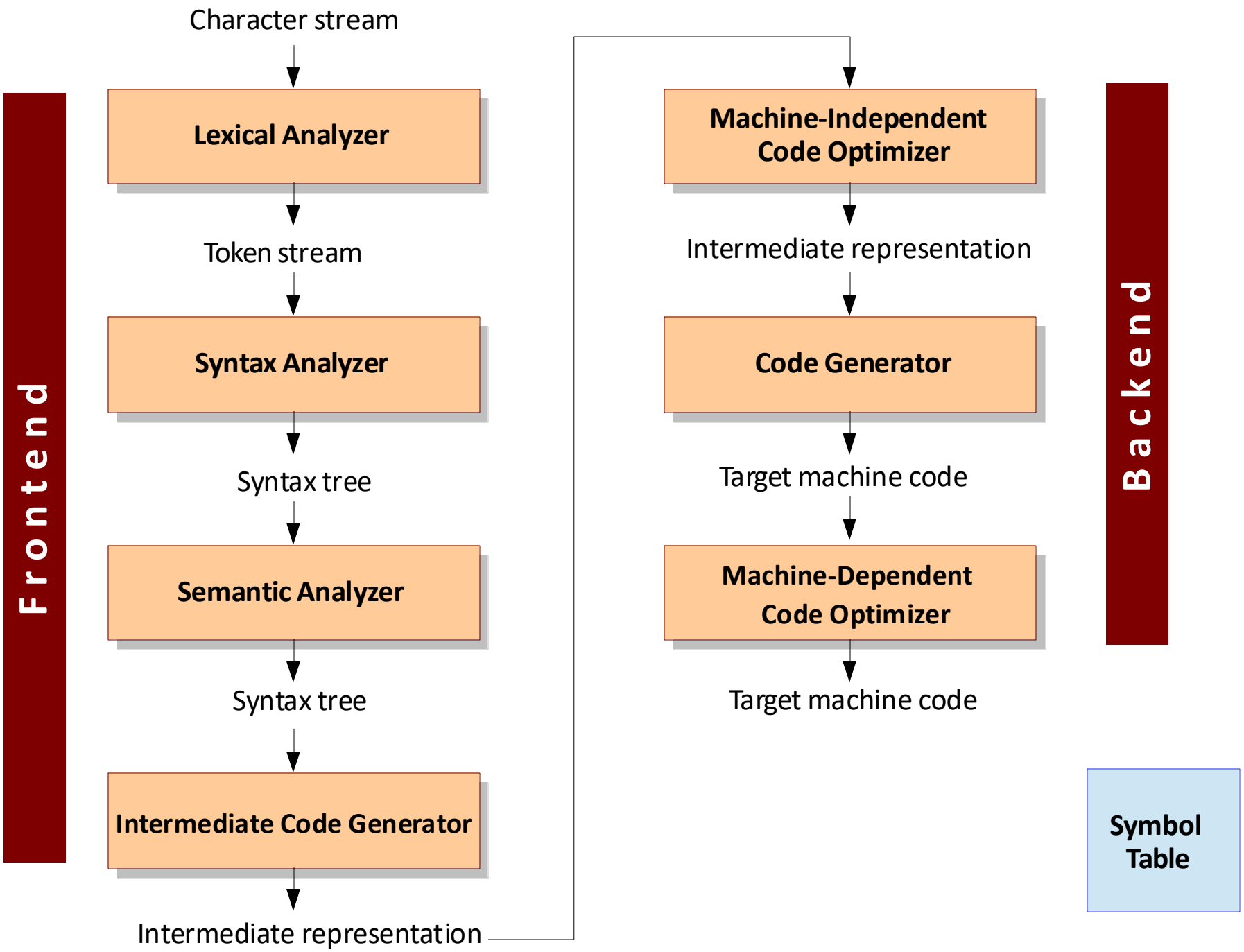
- `cpp file.c >file.i`
- `gcc -S file.i`
- `as file.s -o file.o`
- `ld -o a.out file.o` ...*libraries*...

Try the following:

- `gcc -v file.c`
- `gcc -save-temps file.c`

Language Translators

- **Preprocessor**: collects source programs, expands macros.
- **Compiler**: Translates source program into a low-level assembly.
- **Assembler**: Produces (relocatable) machine code.
- **Linker**: Resolves external references **statically**, combines multiple machine codes.
- **Loader**: Loads executable codes into memory, resolves external references **dynamically**.

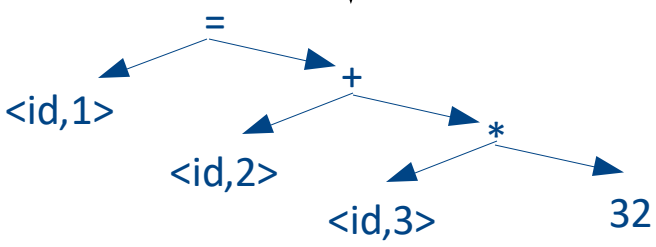

Character stream
Lexical Analyzer
Token stream
Syntax Analyzer
Syntax tree
Semantic Analyzer
Syntax tree
Intermediate Code Generator
Intermediate representation
Machine-Independent Code Optimizer
Intermediate representation
Code Generator
Target machine code
Machine-Dependent Code Optimizer
Target machine code
Frontend
Backend
Symbol Table

z = x + y * 32

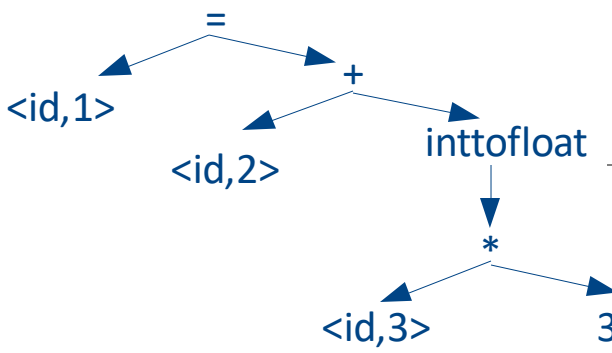
Lexical Analyzer

<id,1> <=> <id,2> <+> <id,3> <*> <32>

Syntax Analyzer



Semantic Analyzer



Intermediate Code Generator

t1 = id3 * 32
t2 = inttofloat(t1)
t3 = id2 + t2
id1 = t3

Machine-Independent Code Optimizer

t1 = id3 * 32
t2 = inttofloat(t1)
id1 = id2 + t2

Code Generator

LD R3, id3
MUL R3, R3, #32
ITOF R2, R3
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1

Machine-Dependent Code Optimizer

LD R3, id3
SHL R3, #5
ITOF R2, R3
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1

1	z	...
2	x	...
3	y	...

Symbol Table

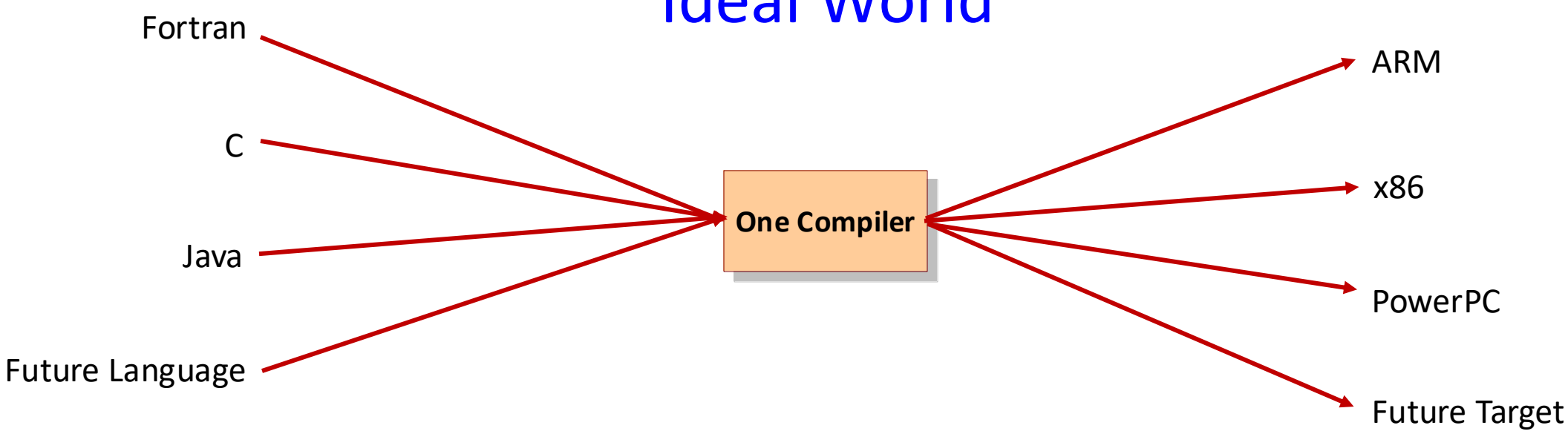
Symbol Table

- Records variable names
- Collects their attributes
  - Type (int, char)
  - Storage requirement ([30], 1)
  - Type modifiers (const, static)
  - Scope (global, static)
  - Information about arguments (for functions)
- Efficient insertion, search (sometimes deletion)
  - C: **int** x, y, z;
  - Pascal: **var** x, y, z: **integer**;

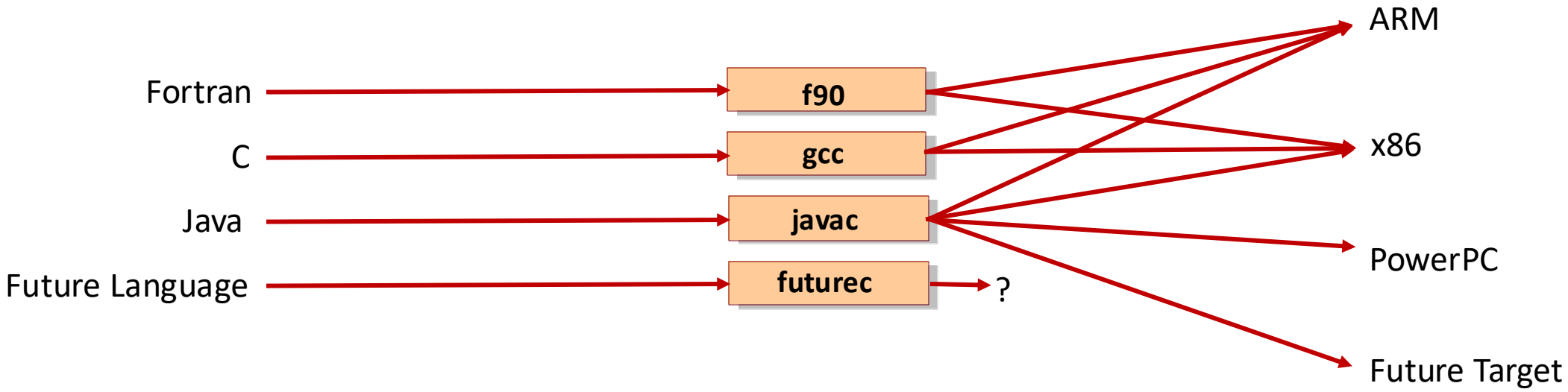
1	z	...
2	x	...
3	y	...

Symbol Table

Ideal World



Reality



Reality getting worse

- I don't have a compiler for this platform.
- My program compiles with an older version of gcc.
- My program compiles with the new version, but does not run on this new platform.
- My program compiles with an older gcc if you disable optimizations.
- My program compiles if you have llvm 5.4, clang 5.5, gcc 5.0.1 on x86_64 with lonestar 1.2 or above on Ubuntu 16 or below.

Evolution of Programming Languages

- First electronic computers in 1940s.
- Programmed in machine language (0 and 1).
  - Move data from one location to another.
  - Add the content of two registers.
  - Compare two values
  - …
- S l o w, T$_{e}$Di^{u}s, and ErorrP run.

Evolution of Programming Languages

- Assembly languages in early 1950s.
  - Initially, only mnemonics for machine instructions
  - Later, support for macros

- High-level languages in late 1950s.
  - **Fortran** for scientific computing
  - **Cobol** for data processing
  - **Lisp** for symbolic computation
  - These were so successful that they are still in use.

PL Classification

- Thousands of languages
  - Need to be categorized
- Based on paradigm
  - Imperative (c, c++, java), declarative (lisp, prolog)
- Based on generation *(think of generation gap)*
  - First (machine),
  - second (assembly),
  - third (fortran, cobol, lisp, c),
  - fourth (sql, ps),
  - fifth (prolog)
- Others
  - OO (c++, c#, Ruby),
  - scripting (awk, js, php, python, ruby)

Compiler Writing

- is challenging.
- A compiler is a large program.
- A compiler must translate correctly potentially infinite set of programs that could be written in the source language.
- The problem of generating the optimal target code from a source program is undecidable.
  - Heuristics and Trade-offs.
- **Compilers is an area where Theory meets Practice.**

Appendix

https://github.com/SET-IITGN/CS-327-Compilers

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% **Other** 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:53 PM
Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi
github.com/SET-IITGN/CS-327-Compilers

README Apache-2.0 license

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash`.
- **mul_add_pred_swap** (`maps`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations with swapped precedence (+,*).
- **yapl** (`yapl`): Scanner, Parser, Analyzer of programs written in a C-like language based on the ANSI C 2011 ISO C standard.

ANSI C 2011 ISO C standard

- ANSI C grammar, Lex specification
- ANSI C Yacc grammar

Lab Assignments will be linked, i.e., built upon the previous one, and should be performed in groups of size at most three. Each group will develop a compiler for a custom language of their choice, or a subset of any popular programming language. Compilation phases collectively implemented (spanning across all four lab assignments) need not be exhaustive.

Our Compilation Pipeline

(Character stream) --> ***Front-end*** --> Intermediate Representation (IR) --> ***Back-end*** --> (Target machine code)

Illustration

Obtained from Rupesh Nasre.

`z = x + y * 32` (Character stream)

Front-end

- Lexical Analyzer

  `<id,1> <=> <id,2> <+> <id,3> <*> <32>`

- Syntax Analyzer

No packages published
Publish your first package

Languages

C 98.9% Other 1.1%

Suggested workflows
Based on your tech stack

SLSA Generic generator Configure
Generate SLSA3 provenance for your existing release workflows

CMake based, single-platform projects Configure
Build and test a CMake based project on a single-platform.

MSBuild based projects Configure
Build a MSBuild based project.

More workflows Dismiss suggestions

README Apache-2.0 license

Our Compilation Pipeline

(Character stream) --> ***Front-end*** --> Intermediate Representation (IR) --> ***Back-end*** --> (Target machine code)

Illustration

Obtained from Rupesh Nasre.

`z = x + y * 32` (Character stream)

Front-end

- Lexical Analyzer

  `<id,1> <=> <id,2> <+> <id,3> <*> <32>`

- Syntax Analyzer

```
          =
        /   \
    <id,1>    +
            /   \
        <id,2>  *
              /   \
           <id,3>  32
```

- Semantic Analyzer

```
          =
        /   \
    <id,1>    +
            /   \
       <id,2> (inttofloat)
                  |
                  *
                /   \
             <id,3>  32
```

- Intermediate Code Generator

Build and test a CMake based project on multiple platforms.

More workflows Dismiss suggestions

Activities Google Chrome Jan 4 5:48 PM
Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi
github.com/SET-IITGN/CS-327-Compilers?tab=readme-ov-file#our-compilation-pipeline
README Apache-2.0 license

- Intermediate Code Generator

```
t1 = id3 * 32
t2 = inttofloat(t1)
t3 = id2 + t2
id1 = t3
```

Back-end

- Machine-Independent Code Optimizer

```
t1 = id3 * 32
t2 = inttofloat(t1)
id1 = id2 + t2
```

- Code Generator AT&T syntax to be followed in `gas`

```
LD R3, id3
MUL R3, R3, #32
ITOF R2, R3
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

- Machine-Dependent Code Optimizer AT&T syntax to be followed in `gas`

```
LD R3, id3
SHL R3, #5
ITOF R2, R3
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Software Tools to be used: `lex`, `yacc`, `gcc`, `gas`, and other relevant tools as and when necessary.

Texts/References

- Alfred Aho, Monica Lam, Ravi Sethi, Jeff Ullman, Compilers: Principles, Techniques, and Tools, Pearson

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers

Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% Other 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:49 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/tree/main/Pro

Files

main

Go to file t

- Pro
  - 20_variables.prl
  - 24_varialbes.prl
  - ReadMe.md
  - lex.yy.c
  - logic1.prl
  - logic10.prl
  - logic11.prl
  - logic12.prl
  - logic2.prl
  - logic3.prl
  - logic4.prl
  - logic5.prl
  - logic6.prl
  - logic7.prl
  - logic8.prl
  - logic9.prl
  - pro
  - pro.l
  - pro.y
  - y.tab.c
  - y.tab.h
- language_processor
- mul_add_pred_swap
- playground
- vapl

CS-327-Compilers / Pro /

Top

ReadMe.md

A Syntactically Correct Sentence (well-formed formula or wff) in Propositional Logic

```
~(P->Q)&((Q&P)->Q)
```

Analysis (output of the language processor: ***ACCEPTED***) ✓

```
 Truth table for ~(P->Q)&((Q&P)->Q)


    P     Q   (P->Q)   ~((P->Q))   (Q&P)   ((Q&P)->Q)      ~(P->Q)&((Q&P)->Q)

-------------------------------------------------------------------------------------------
    F     F      T         F      F          T                      F

    F     T      T         F      F          T                      F

    T     F      F         T      F          T                      T

    T     T      T         F      T          T                      F

The wff ~(P->Q)&((Q&P)->Q) is...

Invalid
Consistent
```

A Syntactically Incorrect Sentence (ill-formed formula) in Propositional Logic

```
(P&Q
```

Analysis (output of the language processor: ***REJECTED***) ✗

```
Error class: syntax error

Error : error in line 1 caused by '(' [unmatched left paranthesis or it has invalid placement]

Error string: (P&Q
```

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers

Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% **Other** 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:49 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/tree/main/language_processor

Files

main

Go to file t

- Pro
- language_processor
  - bash scripts
  - sm scripts
  - sm
  - ReadMe.md
  - Report.pdf
- mul_add_pred_swap
- playground
- yapl
- LICENSE
- README.md

CS-327-Compilers / language_processor /

Add file

shouvick149 Add files via upload e2b11ba · last week History

Name	Last commit message	Last commit date
..		
bash scripts	Collections	last week
sm scripts	Collections	last week
sm	Collections	last week
ReadMe.md	Update ReadMe.md	last week
Report.pdf	Add files via upload	last week

ReadMe.md

Language Processor (`sm`) for "Shell Scripting" using a subset of bash will be having the following features:-

1. Shell script involving arithmetic operations using the syntax of bash
2. Input will be in pain text(.txt) format and fed to the executable of my implementation [a.out]
3. " `display` " command[apart from bash feature] will be used for displaying result on screen. display is compulsory at the end of the script.
4. The only way of input is the `.txt` " `sm script` " and nothing in-between that is user will not be asked to enter a number, enter an array and similar waiting for user inputs.

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers

Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% **Other** 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:49 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/tree/main/playground

Files

main

Go to file t

- Pro
- language_processor
- mul_add_pred_swap
- playground
  - ReadMe.md
  - a.out
  - file.c
  - file.s
- yapl
- LICENSE
- README.md

CS-327-Compilers / playground /

Add file

shouvick149 Update ReadMe.md d96d07e · last week History

Name	Last commit message	Last commit date
..		
ReadMe.md	Update ReadMe.md	last week
a.out	Collections	last week
file.c	Collections	last week
file.s	Collections	last week

ReadMe.md

This is your playground!

```
/*
Filenamne: hello.s
example code written in the x86 assembly language (AT&T syntax)
*/
global:
    .string "hello%d\n"
    .globl  main
global2:
    .string "%d\n"
main:
    push    %rbp
    mov %rsp, %rbp
    sub $16, %rsp
    mov $5, -12(%rbp)
    mov $6, -8(%rbp)
    mov $0, -4(%rbp)
    mov -12(%rbp), %edx
    mov -8(%rbp), %eax
    add %edx, %eax
    mov %eax, -4(%rbp)
    mov -4(%rbp), %eax
    mov %eax, %esi
    lea global2(%rip), %rax
```

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers

Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% Other 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:50 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/tree/main/yapl

Files

main

Go to file t

- Pro
- language_processor
- mul_add_pred_swap
- playground
- yapl
  - testcases
  - Makefile
  - ReadMe.md
  - assign_op.c
  - ex1.c
  - ex2.c
  - lang_spec.odt
  - lang_spec.pdf
  - lex.yy.c
  - y.tab.c
  - y.tab.h
  - yapl
  - yapl.l
  - yapl.y
- LICENSE
- README.md

CS-327-Compilers / yapl / Top

yapl.y	Added scanner and parser yapl: a C-like language.	last week

ReadMe.md

Scanning & Parsing for `yapl`: a C-like Language

The `yapl` specification is available here.

`yapl` in action...

testcase_1.c

```
int *var=6;
struct mystruct *ms=&var;

int auto, static, inline;

extern void *k;
int p;
int p;
int p;

int *hh(char *p);

int main(int b)
{
    int auto=2,b=3,c;
    c=auto+b;
    printf("%d",c);

    struct player
    {
        int a;
        double c;
    };

    int *jj=&auto;

    System.out.print("java here");
    char echo[3]="bash here";
    myprintf("CS3300 here");
```

Activities Google Chrome Jan 4 5:50 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf

Files

main

Go to file

- Pro
- language_processor
- mul_add_pred_swap
- playground
- yapl
  - testcases
  - Makefile
  - ReadMe.md
  - assign_op.c
  - ex1.c
  - ex2.c
  - lang_spec.odt
  - lang_spec.pdf
  - lex.yy.c
  - y.tab.c
  - y.tab.h
  - yapl
  - yapl.l
  - yapl.y
- LICENSE
- README.md

77.7 KB

CS 327: Compilers January 2026

Scanning & Parsing for **yapl**: a C-like Language

Task

Write input files for lex and yacc to generate lexical analyzer and parser for the following language description.

This language specification conforms to the ANSI C-2011 standard with the following requirements/modifications:

• The language constructs:

- ✔ global variables, both normal as well as extern.
- ✔ function declaration, function definition, and function call (including recursion).
- ✔ for loop, while loop, do-while.
- ✔ if, if-else, if-else-if (nested/ladder) {if condition can be any expression including a literals/constants}.
- ✔ switch/case with case labeled with ':', must support fall-through, must support the optional default clause.
- ✔ variable declaration, definition, initialization locally/globally {multiple variables comma-separated}.
- ✔ strings (of the forms "..." (single), "..." "..." (multiple)), character, integer and floating point literals.
- ✔ single (//) and multi-line (/*...*/) comments {multi-line comments should be properly closed lexically}.
- ✔ break, continue, and return statements.
- ✔ a statement must end in a semi-colon ';' like the usual standard of C.

• The supported data types are:

- ✔ primary: int, long, short, float, double, void, char and their pointers.
- ✔ user-defined: structures (struct) {pointer dereferences must support arbitrary level of indirection.}
- ✔ derived: functions, arrays, and pointers: all supported data types. {no need to support function pointers.}

• Supported operators (precedence and associativity same as in C):

- ✔ *relational operators*: <, >, ==, <=, >=, !=
- ✔ *unary operators*: +, −, !, *, &, ~
- ✔ *binary operators*: +, −, *, &, |, /, %, ^
- ✔ *logical operator*: &&, ||
- ✔ *assignment operator*: =
- ✔ *suffix/postfix increment and decrement*: ++, --
- ✔ *structure field access operators*: ->, .
- ✔ *pointers*: *, & {pointer dereferences must support arbitrary/multiple level of indirection, i.e., int *****p;.}
- ✔ *function call*: () {any valid expression inside including blank, constants, and literals}

Activities Google Chrome Jan 4 5:50 PM
Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile
github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf
Files
main
Go to file
Pro
language_processor
mul_add_pred_swap
playground
yapl
testcases
Makefile
ReadMe.md
assign_op.c
ex1.c
ex2.c
lang_spec.odt
lang_spec.pdf
lex.yy.c
y.tab.c
y.tab.h
yapl
yapl.l
yapl.y
LICENSE
README.md
CS-327-Compilers / yapl / lang_spec.pdf
Top
77.7 KB

- ✔ primary: int, long, short, float, double, void, char and their pointers.
- ✔ user-defined: structures (struct) {pointer dereferences must support arbitrary level of indirection.}
- ✔ derived: functions, arrays, and pointers: all supported data types. {no need to support function pointers.}

• Supported operators (precedence and associativity same as in C):

- ✔ *relational operators*: <, >, ==, <=, >=, !=
- ✔ *unary operators*: +, −, !, ∗, &, ~
- ✔ *binary operators*: +, −, ∗, &, |, /, %, ^
- ✔ *logical operator*: &&, ||
- ✔ *assignment operator*: =
- ✔ *suffix/postfix increment and decrement*: ++, --
- ✔ *structure field access operators*: ->, .
- ✔ *pointers*: *, & {pointer dereferences must support arbitrary/multiple level of indirection, i.e., int *****p;.}
- ✔ *function call*: () {any valid expression inside including blank, constants, and literals}
- ✔ *array subscripting*: [] {any valid expression inside including constants and literals}
- ✔ sizeof() {operands: <typename> or unary expression}
- ✔ *typecast*: (<typename>)

• Constructs **NOT** supported by this language:

- ✗ *preprocessors*: none allowed. {no #includes, #defines needed anywhere}
- ✗ ... (ellipsis), ?: (ternary operator) {function vararg list specification not required}
- ✗ *operators*: >>=, <<=, +=, -=, *=, /=, %=, &=, ^=, |=, <<, >>
- ✗ *constructs/keywords*: auto, const, goto, inline, register, restrict, signed, static, typedef, unsigned, enum, union,volatile, _Alignas, _Alignof, _Atomic, _Bool, _Complex, _Generic, _Imaginary, _Noreturn, _Static_assert, _Thread_local, __func__

• The following operator is *introduced*:

Operator	Description
<=>	*Three way comparison.* a<=>b returns the values −1, 0, or 1 depending on whether a < b, a == b, or a > b. Precedence and associativity same as: <, >, <=, >=

Note:

Activities Google Chrome Jan 4 5:50 PM
Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile
github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf

1. The ANSI C-2011 keywords that are not a part of this language can be used as identifiers.
2. No need to handle:
   - Semantics and expression evaluations.
   - Track whether a variable, structure, function is declared/defined before being used.
   - Type of a variable, structure, function, type checking.

Input

The input to the parser will be a program which may or may not be valid according to the above language description. Sample execution format:

```
$ ./a.out [filename.c]
```

Note: input file `[filename.c]` should be passed as command line argument to `a.out`

Output

[1] *If the program is parsed successfully, then the following should be printed*:

```
***parsing successful***
#global_declarations = @
#function_definitions = @
#integer_constants = @
#pointers_declarations = @
#ifs_without_else = @
if-else max-depth = @
```

Note: Replace @ with corresponding counts. There must be one white space before and after '='. See sample testcases.

- `#global_declarations:` Total number of global declarations (enitites) including functions, global variables, extern variables at the global scope. Note that declaration and definition for the same function are counted separately.
-
- `#function_definitions:` Total number functions having bodies, i.e. defined.
- `#integer_constants:` Total number of integer constants that appear *anywhere* throughout the program. Should support hexadecimal constants like `0x1234`. Upper-case 'X' for hex need not be supported.
- `#pointers_declarations:` Total count of pointers that are declared throught the file. Note that if a function returning a pointer, has both prototype (declaration) and definition, they are counted separately. Pointer casts should not be counted.
- `#ifs_without_else:` Total count `if` statements that have no associated `else` clause.

Activities Google Chrome Jan 4 5:50 PM
Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile
github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf
Files
main
Go to file
Pro
language_processor
mul_add_pred_swap
playground
yapl
testcases
Makefile
ReadMe.md
assign_op.c
ex1.c
ex2.c
lang_spec.odt
lang_spec.pdf
lex.yy.c
y.tab.c
y.tab.h
yapl
yapl.l
yapl.y
LICENSE
README.md
CS-327-Compilers / yapl / lang_spec.pdf
Top
77.7 KB

- `#global_declarations:` Total number of global declarations (enitites) including functions, global variables, extern variables at the global scope. Note that declaration and definition for the same function are counted separately.
-
- `#function_definitions:` Total number functions having bodies, i.e. defined.
- `#integer_constants:` Total number of integer constants that appear *anywhere* throughout the program. Should support hexadecimal constants like `0x1234`. Upper-case 'X' for hex need not be supported.
- `#pointers_declarations:` Total count of pointers that are declared throught the file. Note that if a function returning a pointer, has both prototype (declaration) and definition, they are counted separately. Pointer casts should not be counted.
- `#ifs_without_else:` Total count `if` statements that have no associated `else` clause.
- `if-else max-depth:` The max height of `if-else-if` ladder. The height is recursively defined as follows.

  height(Ladder)
  =0, if there is no `if` statement at all.
  =0, if there is `if` but no accompanying `else`.
  =1 (for the accompanying `else`) + *height*(Ladder from this `else`)

  There will be many such ladders, the last line must print the *height of the longest ladder* present in the program. You should keep track of ladders across functions also. Note that there can be ties also.

  `if-else max-depth` = $max\{height(\text{Ladder}_1), \ldots, height(\text{Ladder}_n)\}$ when there are n ladders in the input file.

[2] *On rejection by the parser, the following should be printed*:

```
***parsing terminated*** [syntax error]
```

[3] *On rejection by the lexer when there are unclosed comments, the following should be printed*:

```
***lexing terminated*** [lexer error]: ill-formed comment
```

[4] *On rejection by your a.out file it must handle these cases*:
Case 1: (invalid number of command-line arguments)-

Activities Google Chrome Jan 4 5:50 PM
Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile
github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf
Files
main
Go to file
Pro
language_processor
mul_add_pred_swap
playground
yapl
testcases
Makefile
ReadMe.md
assign_op.c
ex1.c
ex2.c
lang_spec.odt
lang_spec.pdf
lex.yy.c
y.tab.c
y.tab.h
yapl
yapl.l
yapl.y
LICENSE
README.md
CS-327-Compilers / yapl / lang_spec.pdf
Top
77.7 KB

```
***process terminated*** [input error]: invalid number of command-line arguments
```

Case 2: (no such file <filename> exists)-

```
***process terminated*** [input error]: no such file <filename> exists
```

Note: The Makefile should run `lex`, `yacc`, compile the generated code and generate an executable `a.out` file. Please be careful about the naming conventions and structure of the directory.

Sample testcase #1

Input

```
int *var=6;
struct mystruct *ms=&var;

int auto, static, inline;

extern void *k;
int p;
int p;
int p;

int *hh(char *p);

int main(int b)
{
        int auto=2,b=3,c;
        c=auto+b;
        printf("%d",c);

        struct player
        {
                int a;
                double c;
        };

        int *jj=&auto;

        System.out.print("java here");
```

Activities Google Chrome Jan 4 5:50 PM

Inbox - shouvick | Inbox - shouvick | CS327_Lecture_ | SET-IITGN/CS-32 | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile | CS-327-Compile

github.com/SET-IITGN/CS-327-Compilers/blob/main/yapl/lang_spec.pdf

Files

main

Go to file t

Pro
language_processor
mul_add_pred_swap
playground
yapl
testcases
Makefile
ReadMe.md
assign_op.c
ex1.c
ex2.c
lang_spec.odt
lang_spec.pdf
lex.yy.c
y.tab.c
y.tab.h
yapl
yapl.l
yapl.y
LICENSE
README.md

CS-327-Compilers / yapl / lang_spec.pdf

Top

77.7 KB

```
{
        int n=7;

        scanf("%d",&n);

        if(n==0)
                printf("%d\n",n+1);
        else if(n==1)
                printf("%d\n",n+2);

        return(NULL);
}
```

Expected output

```
***parsing successful***
#global_declarations = 10
#function_definitions = 2
#integer_constants = 20
#pointers_declarations = 12
#ifs_without_else = 2
if-else max-depth = 3
```

Sample testcase #2

Input

```
int a, b, c;

int main(int *argc, char* argv[])
{
        int a, i;
        int b[10];
        for(i=1; i<=10; ++i) {
                b[i-1] = i*i;               // square
        }
        int *c = b;
        i = 0;
        int d= 0;
        while(i<10) {
                d += *(c + i);
        }
        printf("%d\n", d);          // print sum of values
        return 0;
}
```

Expected output

```
***parsing terminated*** [syntax error]
```

Activities Google Chrome Jan 4 5:53 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Compi

github.com/SET-IITGN/CS-327-Compilers

SET-IITGN / CS-327-Compilers

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

CS-327-Compilers

Public

Edit Pins | Unwatch 1 | Fork 0 | Star 2

main | 1 Branch | 0 Tags | Go to file | Add file | Code

shouvick149 Update README to reflect additional language processor 6f1f0a4 · last week 27 Commits

Pro	Update ReadMe.md	last week
language_processor	Add files via upload	last week
mul_add_pred_swap	Clarify language specification statement in ReadMe	last week
playground	Update ReadMe.md	last week
yapl	Update ReadMe.md	last week
LICENSE	Initial commit	last week
README.md	Update README to reflect additional language processor	last week

README | Apache-2.0 license

CS-327-Compilers

Jan-May 2026, IIT Gandhinagar

Getting Started

You can use the materials (codebase) in this repository to get started for building your own compiler. Four sample language processors (source code + binary) designed by me are available here.

- **Pro** (`pro`): Scanner, Parser, Analyzer of sentences expressed in Propositional Logic.
- **language_processor** (`sm`): Scanner, Parser, Analyzer of sentences expressing arithmetic operations in `bash` .

About

No description, website, or topics provided.

Readme
Apache-2.0 license
Activity
Custom properties
2 stars
1 watching
0 forks
Audit log
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 98.9% Other 1.1%

Suggested workflows

Based on your tech stack

Activities Google Chrome Jan 4 5:51 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Comp | CS-327-Compilers/Pro | CS-327-Compilers/langu | CS-327-Compilers/mul_

github.com/SET-IITGN/CS-327-Compilers/tree/main/mul_add_pred_swap#context-free-grammar-cfg-with-the-precedence-of-addition-and-multiplication-swapped-this-means--has-higher-precedence-than-

Files

main

Go to file

- Pro
- language_processor
- mul_add_pred_swap
  - Makefile
  - ReadMe.md
  - input.txt
  - lex.yy.c
  - maps
  - mul_add_pred_swap.l
  - mul_add_pred_swap.y
  - y.tab.c
  - y.tab.h
- playground
- yapl
- LICENSE
- README.md

CS-327-Compilers / mul_add_pred_swap / Top

ReadMe.md

Context-free grammar (CFG) with the precedence of addition and multiplication swapped. This means `+` has higher precedence than `*`

```
E -> E * T | T
T -> T + F | F
F -> NUM

| Operator | Precedence | Associativity |
| -------- | ---------- | ------------- |
| +        | Higher     | Left          |
| *        | Lower      | Left          |
```

Context Free Grammar

When a task involves dividing the input into units and establishing some relationship among those units, you should think of Lex and yacc.

Numbers: Arbitrary length via lexer [0-9]+ → single NUM token

Lexical Analyzer (scanner)

```
%{
    #include <stdlib.h>
    #include "y.tab.h"
    extern char mytext[];
%}

%%

"+"     {return ADD;}
"*"     {return MUL;}
[0-9]+  {strcpy(mytext, yytext); return NUM;}
.|\n    {;}

%%

int yywrap(void) {
    return 1;
}
```

part #1: C code (mostly headers)

part #2: "reg-ex patterns" {actions}

`yytext` contains the text that matched the pattern

part #3: C code (other code)

Output: `lex.yy.c` containing `yylex()` **//our lexer**

Syntax Analyzer (parser)

Activities Google Chrome Jan 4 5:51 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Comp | CS-327-Compilers/Pro | CS-327-Compilers/langu | CS-327-Compilers/mul_

github.com/SET-IITGN/CS-327-Compilers/tree/main/mul_add_pred_swap#context-free-grammar-cfg-with-the-precedence-of-addition-and-multiplication-swapped-this-means--has-higher-precedence-than-

Files

main

Go to file

- Pro
- language_processor
- mul_add_pred_swap
  - Makefile
  - ReadMe.md
  - input.txt
  - lex.yy.c
  - maps
  - mul_add_pred_swap.l
  - mul_add_pred_swap.y
  - y.tab.c
  - y.tab.h
- playground
- yapl
- LICENSE
- README.md

CS-327-Compilers / mul_add_pred_swap / Top

```
}
```

Syntax Analyzer (parser)

`lex.yy.c` containing `yylex()` ***//our lexer entry point***

```
%token ADD MUL NUM
%{
    #include<stdio.h>
    #include<stdlib.h>
    void yyerror(char *);
    int yylex(void);
    char mytext[100];
    extern char *yytext;
%}
%start S

%%

S:  E {$$=$1;printf("%d\n",$$);}
    ;

E:  E MUL T {$$=$1*$3;}
    | T {$$=$1;}
    ;

T:  T ADD F {$$=$1+$3;}
    | F {$$=$1;}
    ;

F:  NUM {$$=atoi(yytext);}
    ;

%%

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}

int main(void) {
    yyparse();
    return 0;
}
```

part #1:(tokens, headers, starting non-terminal)

part #2:(CFG productions)

part #3:(`main()`, syntax error handler, other C code)

`yyparse()` is our parser entry point

grand entry point for the lexer+parser combination

Output #1: `y.tab.c` ***//our parser***

Output #2 (opt.): `y.tab.h` ***//token definitions***

Activities Google Chrome Jan 4 5:51 PM

Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Comp | CS-327-Compilers/Pro a | CS-327-Compilers/langu | CS-327-Compilers/mul_

github.com/SET-IITGN/CS-327-Compilers/tree/main/mul_add_pred_swap#context-free-grammar-cfg-with-the-precedence-of-addition-and-multiplication-swapped-this-means--has-higher-precedence-than-

Files

main

Go to file

- Pro
- language_processor
- mul_add_pred_swap
  - Makefile
  - ReadMe.md
  - input.txt
  - lex.yy.c
  - maps
  - mul_add_pred_swap.l
  - mul_add_pred_swap.y
  - y.tab.c
  - y.tab.h
- playground
- yapl
- LICENSE
- README.md

CS-327-Compilers / mul_add_pred_swap / Top

Makefile for building the compiler

gcc

yacc

lex

```
a.out: y.tab.c lex.yy.c
    gcc lex.yy.c y.tab.c -o maps
    @echo "Run the program as ./maps < input.txt"

y.tab.c: mul_add_pred_swap.y
    yacc -d mul_add_pred_swap.y

lex.yy.c: mul_add_pred_swap.l y.tab.h
    lex mul_add_pred_swap.l

clean:
    @rm -f lex.yy.c y.tab.h y.tab.c maps
```

```
$ make
gcc lex.yy.c y.tab.c -o maps
Run the program as ./maps < input.txt
```

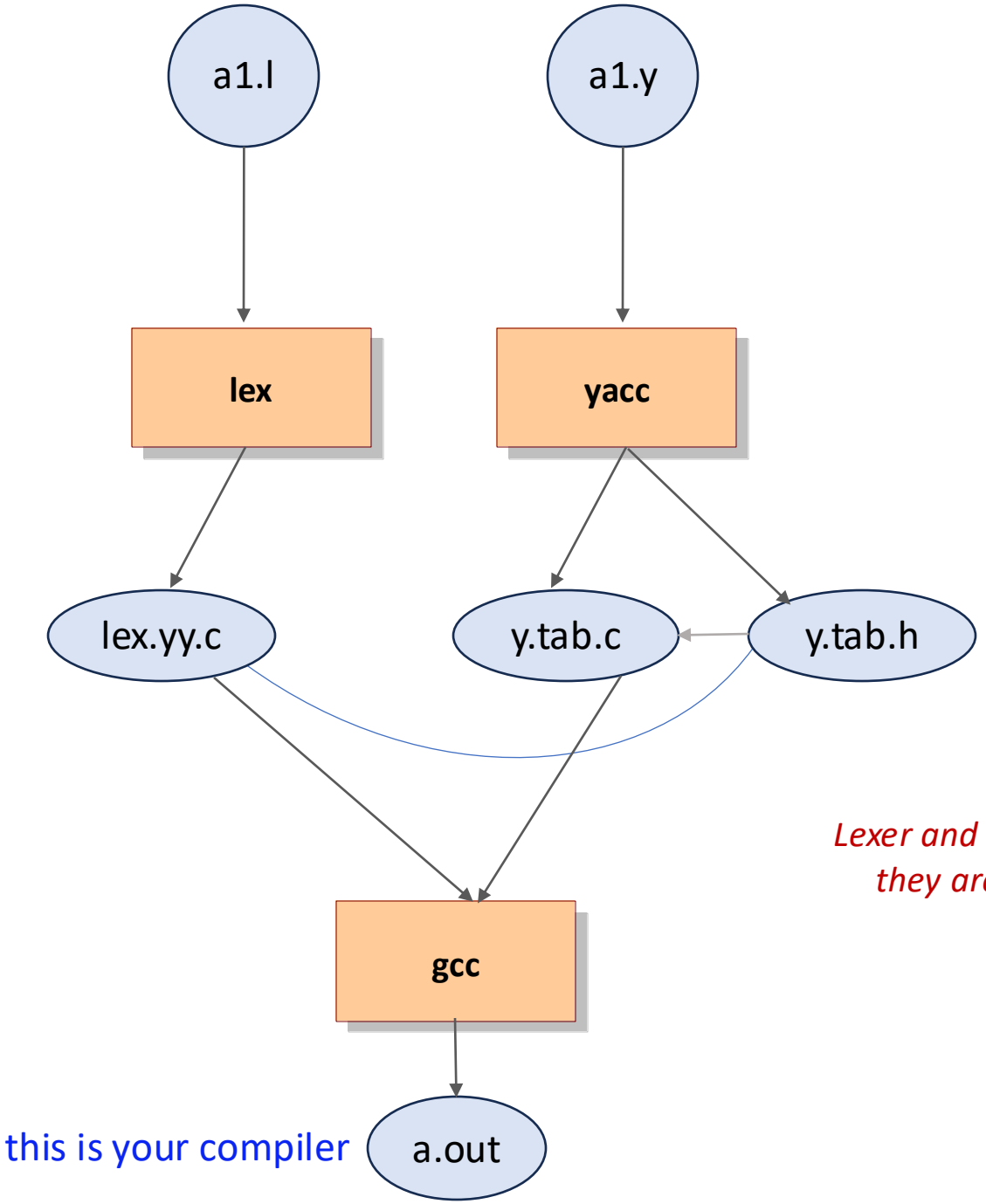
build compiler

Executing the compiler (language processor)

```
$ ./maps
10+5
15
```

execute compiler

```
$ ./maps
10+5*7
105
```



Lexer and parser are not separate binaries; they are part of the same executable.

Activities Google Chrome Jan 4 5:51 PM
Inbox - shouvick.mondal | Inbox - shouvick.mondal | CS327_Lecture_1.pptx | SET-IITGN/CS-327-Comp | CS-327-Compilers/Pro | CS-327-Compilers/langu | CS-327-Compilers/mul_
github.com/SET-IITGN/CS-327-Compilers/tree/main/mul_add_pred_swap#context-free-grammar-cfg-with-the-precedence-of-addition-and-multiplication-swapped-this-means--has-higher-precedence-than-


```
3+
syntax error
```

```
$ ./maps
4+9*(7)
91
```

```
$ ./maps
(8((
8
```

```
$ ./maps
int a=8;
int b=a+8*2;
printf("%d",b);
32
```

```
$ ./maps
a=8;
kuch bhi;
+5*iitgn
4
52
```

Observation

The way you specify a programming language (syntax + semantics) determines what you deploy through software written in that language, including bugs (features).

Texts, References, and Acknowledgements

1. Alfred Aho, Monica Lam, Ravi Sethi, Jeff Ullman. *Compilers: Principles, Techniques, and Tools*, Pearson Education India; 2nd Edition, 2013.
2. Steven Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann; 1st edition 1997.
3. John R. Levine, Tony Mason, and Doug Brown, *Lex & Yacc* (2nd ed.). O'Reilly & Associates, Inc., USA, 1992.
4. Jonathan Bartlett, *Programming from the Ground Up*, 2003.
5. Wikibooks, *x86 Assembly*, *GNU assembly syntax* (online).

Slides obtained from Rupesh Nasre.