

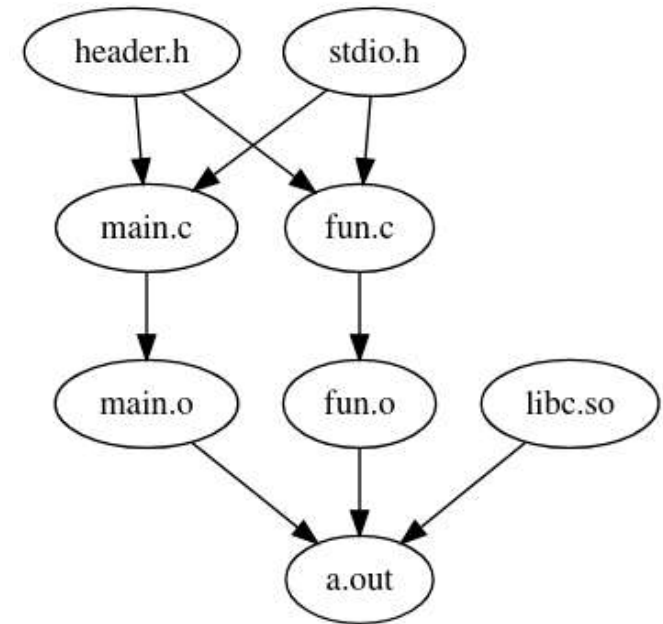
make

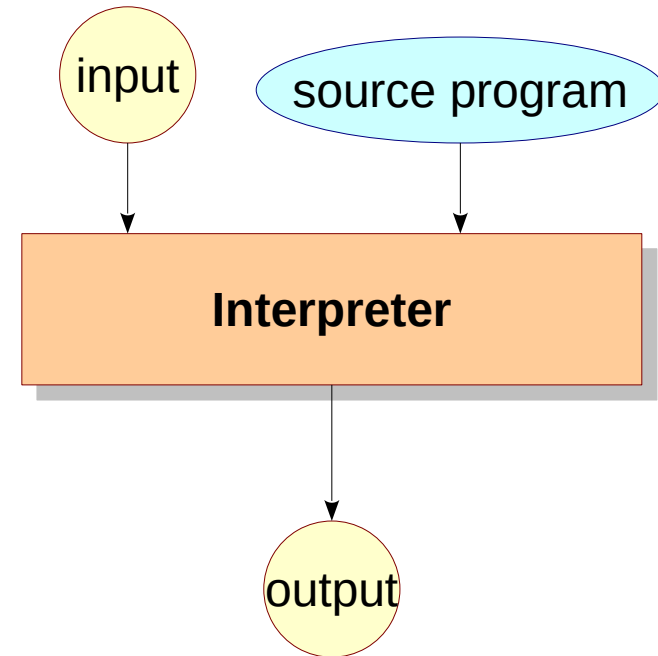
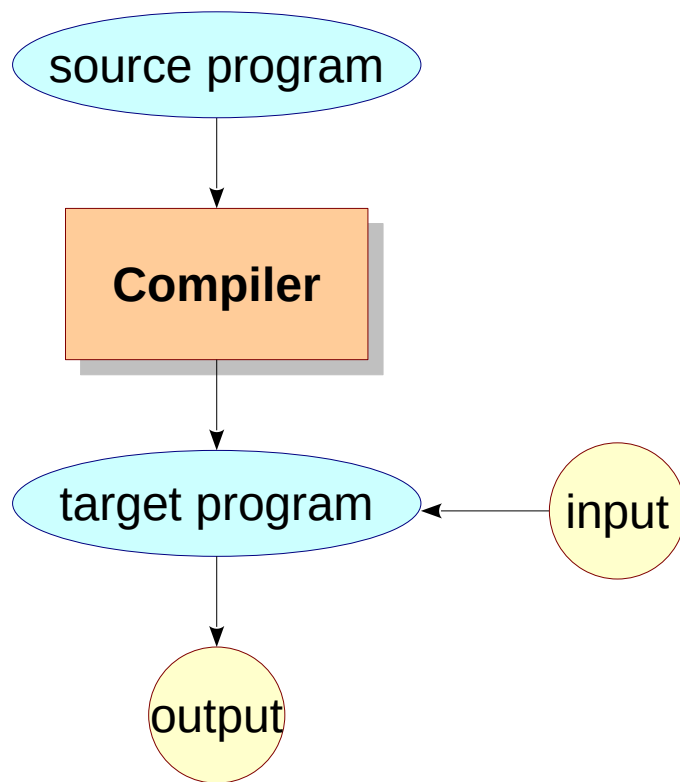
Rupesh Nasre.
rupesh@cse.iitm.ac.in

IIT Madras
January 2026

Compiling Large Projects

- Large projects consist of:
 - Multiple teams
 - Multiple directories
 - Several files
 - Complex dependencies
- One .h file may affect several .c / .cpp files, but not all.
- If one .c file changes, do we need to compile the whole project?





- What does this mean?
 - You may be able to do the following with interpreters.

```

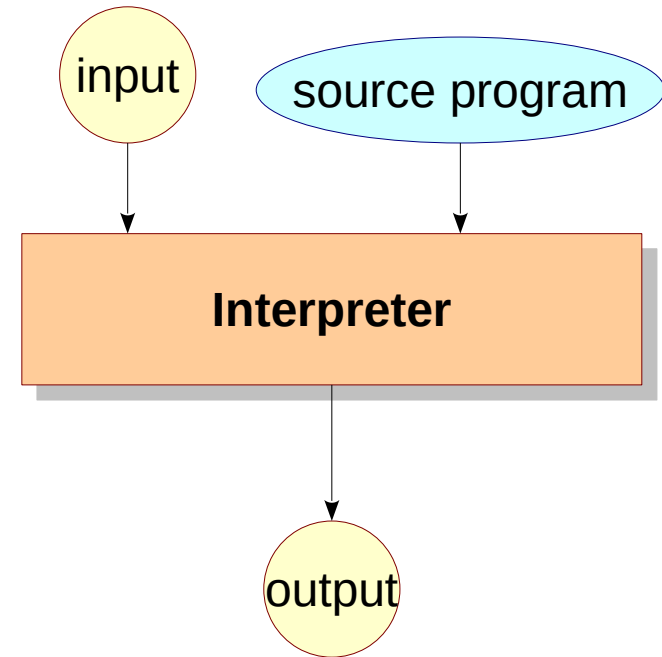
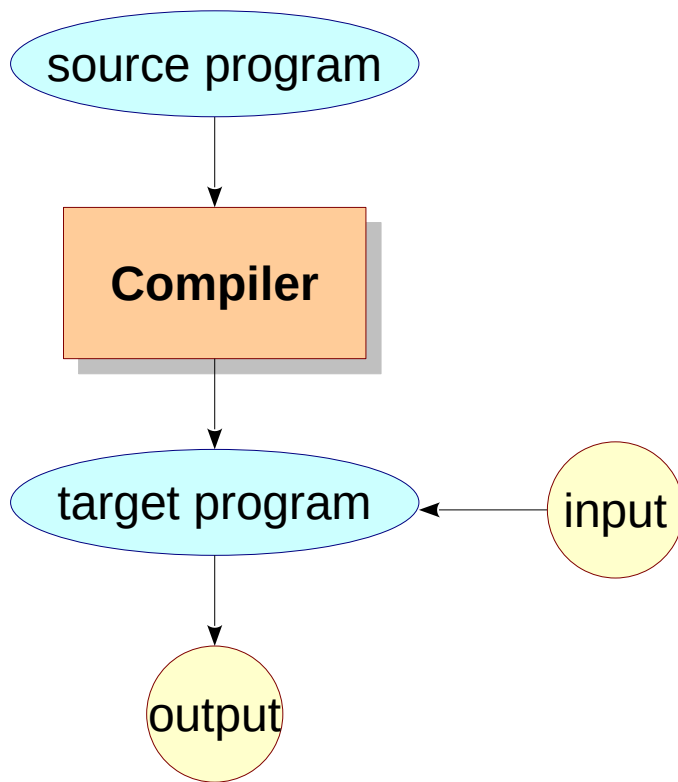
$x = 0; $y = 0;
echo "Enter a variable name: ";
$line = fgets(STDIN);
$line = trim($line);
${$line} = 20;
echo "x=$x, y=$y\n";
  
```

How about C?

```

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

3
Everything is fair in love, war, and C.

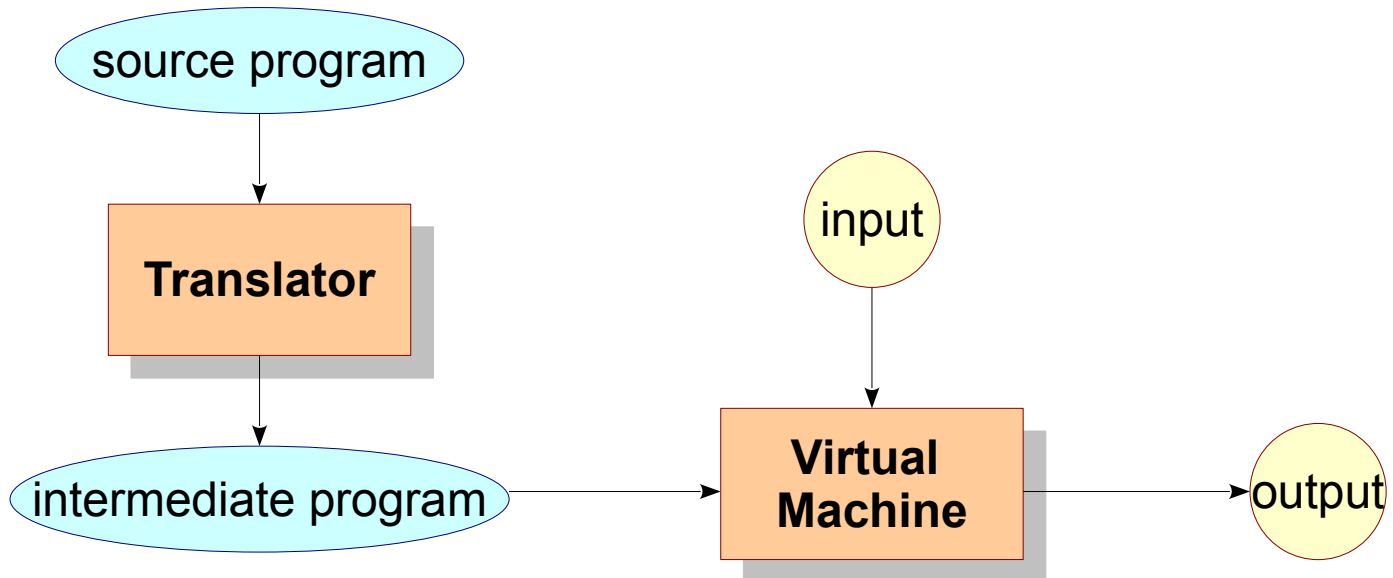
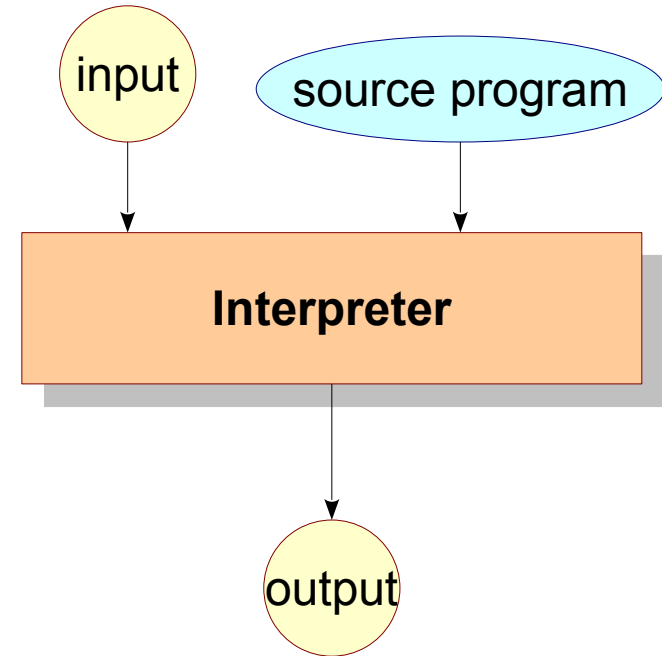
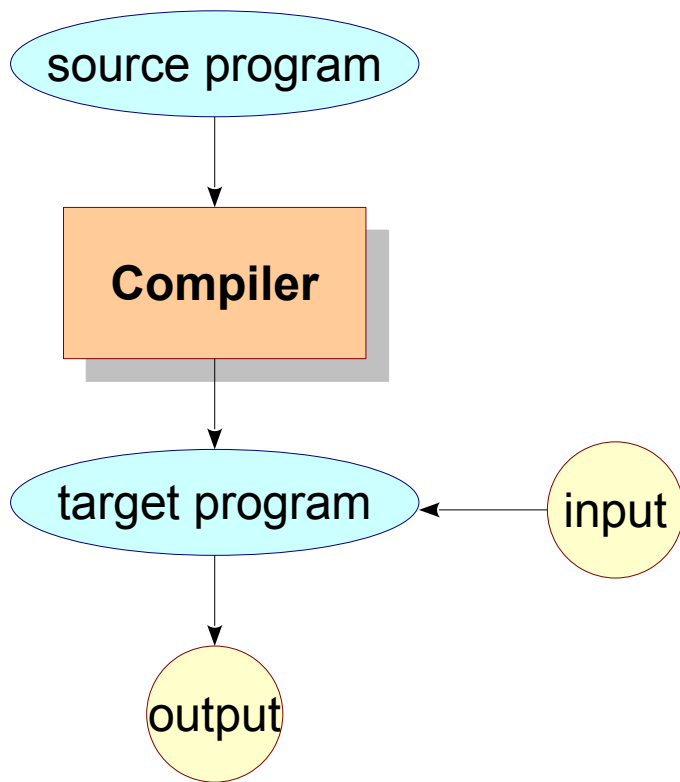


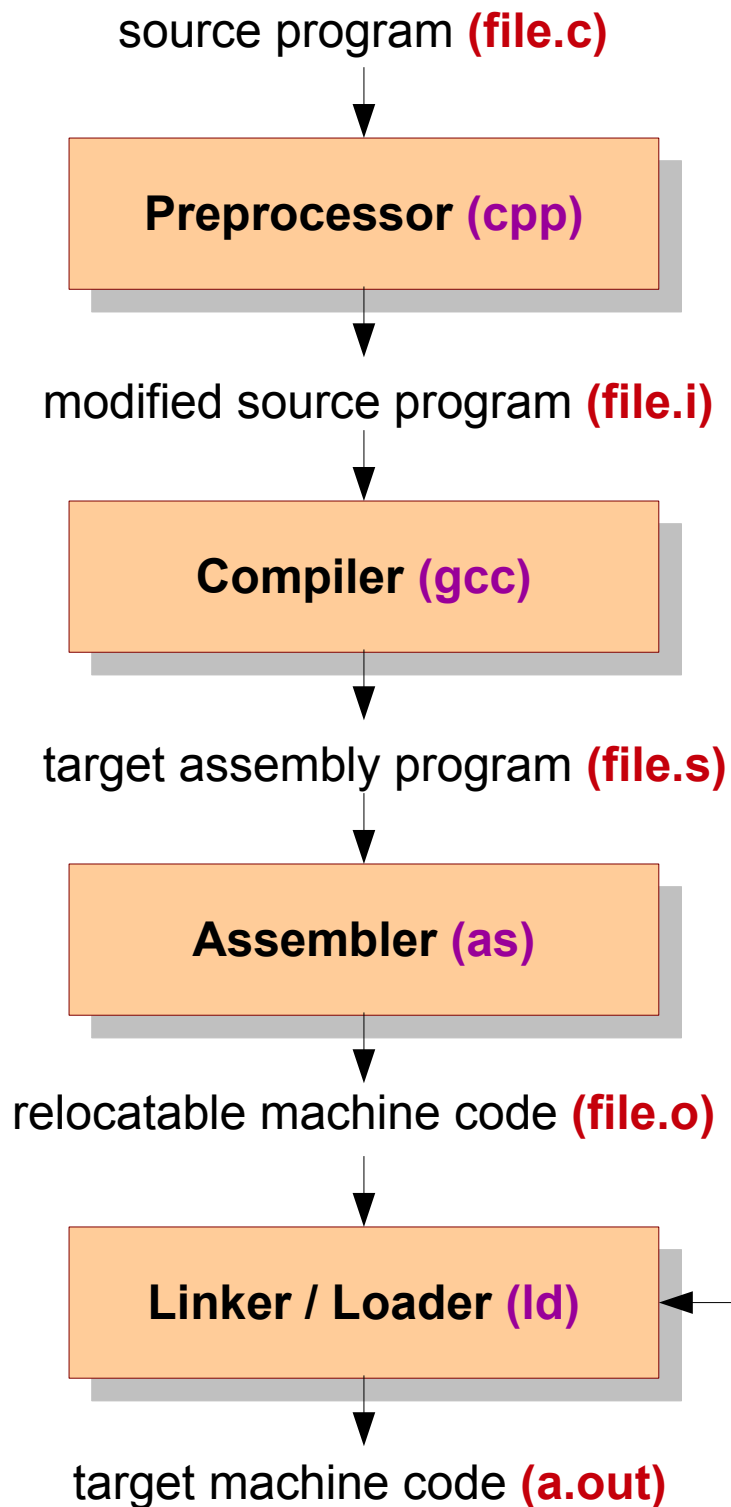
- What does this mean?
 - You may be able to do the following with compilers.

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

is equivalent to

```
x += 18;
```





- `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:

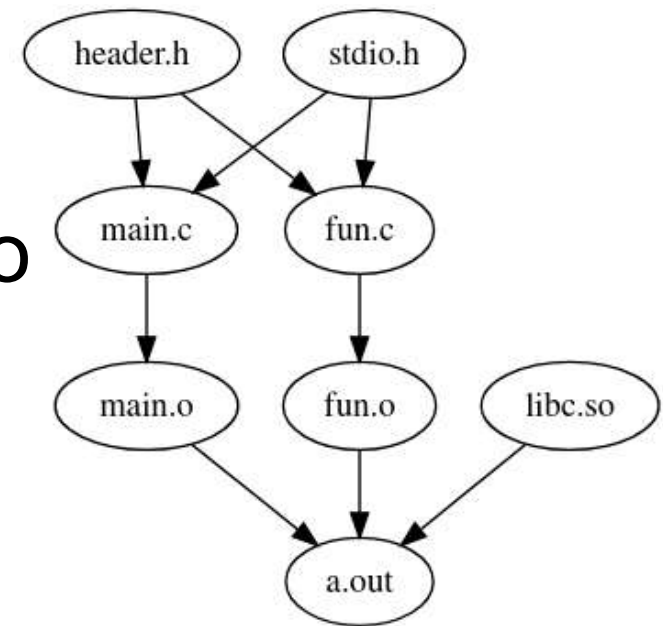
- `gcc -v file.c`
- `gcc -save-temps file.c`

We would like
to save as
much processing
as possible.

library files,
relocatable object files

Compiling Large Projects

- If main.c changes, we would like to compile main.c to main.o, and then link main.o to a.out with fun.o and libc.so.
- **make** permits us to specify these dependencies.
- It uses files' timestamps to decide whether a file is modified.
 - If main.c is modified, it would have a higher timestamp than that of main.o.



Makefile

target prerequisites

```
a.out: list.c main.c
```

RULE

```
gcc list.c main.c
```

must be a tab action / commands



a.out depends upon ...

how to get a.out

\$ make

gcc list.c main.c

\$ make

make: 'a.out' is up to date.

\$ make -f Makefile

reads Makefile (default)

executes the command

same as make, can be

used to specify a different file

Multiple Rules

```
a.out: list.o main.o
    gcc list.o main.o
```

```
list.o: list.c list.h
    gcc -c list.c
```

```
main.o: main.c list.h
    gcc -c main.c
```

```
clean:
    rm *.o a.out
```

As a general programming practice:
avoid repetition.

```
$ make -f Makefile2
```

```
gcc -c list.c
```

```
gcc -c main.c
```

```
gcc list.o main.o
```

```
$ emacs list.c      # modify
```

```
$ make -f Makefile2
```

```
gcc -c list.c
```

```
gcc list.o main.o
```

```
$ make clean -f Makefile2
```

```
rm *.o a.out
```

Make Variables

```
myprog: list.o main.o  
    gcc  $\$^$  -o  $\$@$ 
```

```
list.o: list.c list.h  
    gcc -c list.c
```

```
main.o: main.c list.h  
    gcc -c main.c
```

```
clean:  
    rm myprog *.o
```

Makefile3

- $\$^$ == all prerequisites
- $\$@$ == target
- $\$<$ == first prerequisite
- These variables avoid repetition (and errors).

Make Variables

```
BIN = myprog
```

```
${BIN}: list.o main.o  
gcc $^ -o $@
```

```
list.o: list.c list.h  
gcc -c list.c
```

```
main.o: main.c list.h  
gcc -c main.c
```

```
clean:  
rm ${BIN} *.o
```

- $\$^$ == all prerequisites
- $\$@$ == target
- $\$<$ == first prerequisite
- These variables avoid repetition (and errors).

Is there any other repetition?

Make Template Rules

```
BIN = myprog
```

```
${BIN}: list.o main.o  
    gcc $^ -o $@
```

```
%.o: %.c list.h  
    gcc -c $<
```

```
clean:  
    rm ${BIN} *.o
```

Makefile5

- Each **.o** file is dependent on the corresponding **.c** file and **list.h**.
- Use **gcc -c thatfile.c** to generate **thatfile.o**.
- Recall **\$<** == first prerequisite

Makefile is not C specific.

target: One.class Two.class

One.class: One.java
javac One.java

Two.class: Two.java
javac Two.java

No action

data.output: data.awk data.txt
awk -f data.awk data.txt >data.output

- Additional notes

- `touch` can be used to update the modification time of a file.
- One can write multiple commands for a rule.
- Can one generate Makefile for C programs?

In this course...

Application	Programming	Tools
Puzzle of grass, goat, lion	Pointers, structures	Editor, visualization
Data maintenance	File I/O	I/O redirection, make
Item grouping	Linked Lists	Debugging
Gene sequencing	Recursion, strings	Memory profiling, ulimit
Closest match	Matrices	Compute profiling, plotting
Keep the recent	Arrays	Version control
Cryptography	Strings, bitwise operators	Library (ncurses)
Unit testing	Command-line	Testing
Web-page creation, analysis	HTML, Strings	wget, shell scripting
Game	Graphics	Library (graphics)
Basics	Python	--

Our goal: *to have fun with automation*