Introduction

Jia Xiaodong

Last revised August 16, 2021

How are programs run? *

- Stored program computer

How are programs run? *

- Stored program computer
- In essence, just 1's and 0's

How are programs run? *

- Stored program computer
- In essence, just 1's and 0's
- Set up macros to ease the pain (assembly)

How are programs run? *

- Stored program computer
- In essence, just 1's and 0's
- Set up macros to ease the pain (assembly)
- Set up more macros (low level languages)

How are programs run? *

- Stored program computer
- In essence, just 1's and 0's
- Set up macros to ease the pain (assembly)
- Set up more macros (low level languages)
- Either compile it back for the computer to read, or get something to interpret it

Expressions

- Programs are made of sequences of *statements*.

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like
  - `1; "hi"; true; ...`

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like
  - `1; "hi"; true; ...`
  - Unary operators `-5; !true;`

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like
  - `1; "hi"; true; ...`
  - Unary operators `-5; !true;`
  - Binary operators `5 - 2; 10 < 5; "a" === "b";`

Expressions

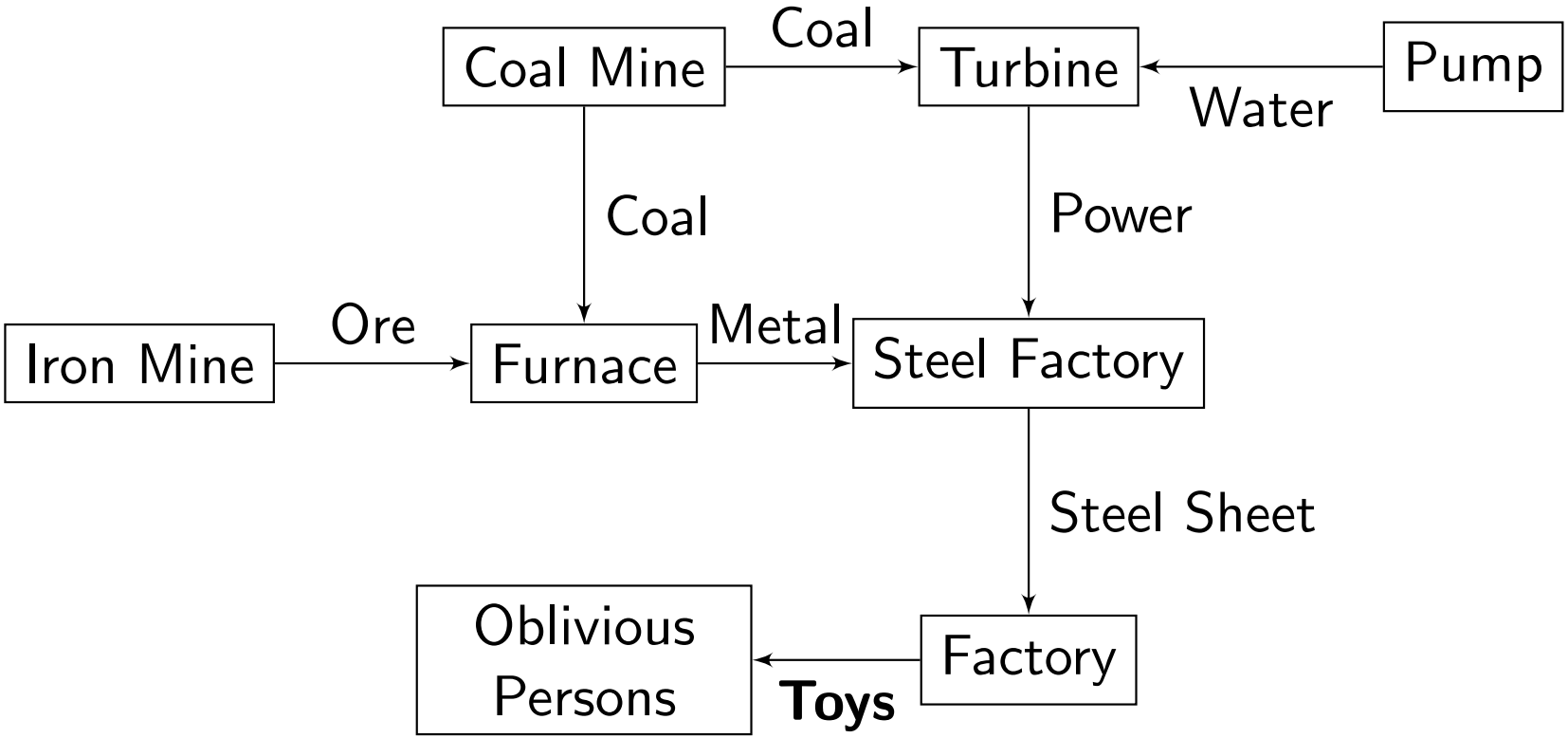
- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like
  - `1; "hi"; true; ...`
  - Unary operators `-5; !true;`
  - Binary operators `5 - 2; 10 < 5; "a" === "b";`
  - Ternary operator `expr1 ? expr2 : expr3`

Expressions

- Programs are made of sequences of *statements*.
- All statements are things such as
  - `const asd = 123;`
  - Blocks `{program}`,
  - Expressions
- And expressions are made of primitives like
  - `1; "hi"; true; ...`
  - Unary operators `-5; !true;`
  - Binary operators `5 - 2; 10 < 5; "a" === "b";`
  - Ternary operator `expr1 ? expr2 : expr3`
- Expressions produce results.

Abstraction

Abstraction, the wagon of progress.



Functions

Example

What does this do?

```
function norm(x, y) {
    return math_sqrt(x * x + y * y);
}
```

Functions

Example

What does this do?

```
function norm(x, y) {
    return math_sqrt(x * x + y * y);
}
```

Good to know

- `display`
- `math_PI`, `math_log`, ...

Conditionals

The **if–else** statement:

```
if (expr) {
    program;
} else if (expr2) {
    program2;
} else {
    program3;
}
```

Ternary operator

```
predicate ? consequent : alternative
```

Ternary operator

```
predicate ? consequent : alternative
```

Examples

- `5 < 2 ? 10 : 100;`

Ternary operator

```
predicate ? consequent : alternative
```

Examples

- `5 < 2 ? 10 : 100;`
- `"a" < "b" ? 1 : 2;`

Ternary operator

```
predicate ? consequent : alternative
```

Examples

- `5 < 2 ? 10 : 100;`
- `"a" < "b" ? 1 : 2;`
- `"a" < "A" ? 1 : 2;`

Short circuiting

```
predicate ? consequent : alternative
```

Example

What does this do?

```
1 === 2 && display("No")
```

Short circuiting

```
predicate ? consequent : alternative
```

Example

What does this do?

```
1 === 2 && display("No")
```

Good to know

Some operators are also lazy!

```
1 === 2 ? display("No") : 1;
```

Modulo

- We can do + - * /

Modulo

- We can do + - * /
- New operator: %