Compilers

INF-400

Burak Arslan
ext-inf400@burakarslan.com

Galatasaray Üniversitesi

Lecture II
2023-10-12

Course website

burakarslan.com/inf400

Why so many Programming Languages?

Some criteria give way to precise classification.

Others draw fuzzier borders.

And some languages are just multi-paradigm

Why so many Programming Languages?

Programming languages are **tools** used by **engineers**.

Why so many Programming Languages?

... and just like any tool;

- They aim to serve their purpose in the **most ergonomic** way.
- It's always possible to make improvements, but the real trick is to be **good enough**.
- Pain increases as you stray further from the intended purpose.

Why so many Programming Languages?

… and just like any **engineering** tool;

They either **cut costs** or **create value**

Why so many Programming Languages?

... and just like **engineers**;

No single approach is objectively better than another

Why so many Programming Languages?

... and just like **engineers**;

No single approach is objectively better than another

... as long as goals are met!

Why so many Programming Languages?

But we still can evaluate software quality based on some criteria:

- Abstractions (how leaky?)
- Code reuse (how DRY? too general?)
- SOLID
- CMMI

Why so many Programming Languages?

Programming languages are **very** costly to develop:

- Kickass tooling (portable compiler/interpreter, editor, debugger, etc)
- Infrastructure (Package repo, CI, Releases)
- Evangelism/Outreach
- Trademarks, licenses, advertising

Why so many Programming Languages?

Programming languages are **very** costly to develop:

- Kickass tooling (portable compiler/interpreter, editor, debugger, etc)
- Infrastructure (Package repo, CI, Releases)
- Evangelism/Outreach
- Trademarks, licenses, advertising

except **Javascript**!

Why so many Programming Languages?

Programming languages are costly to adopt as well:

- Training costs
- Employee turnover
- Interfacing costs
- Rewriting costs

Why so many Programming Languages?

Yet if you find a niche and serve it well,

you can give way to enormous **cost reductions**.

Why so many Programming Languages?

Yet if you find a niche and serve it well,

you can give way to enormous **cost reductions**.

... and you can have that small corner of the universe to yourself.

Types of Programming Languages

Some criteria give way to precise classification.

Others draw fuzzier borders.

And some languages are just multi-paradigm

Types of Programming Languages

Imperative vs Declarative

Declarative The desired program output is described as a set of constraints

Imperative The desired program output is described as a sequence of instructions

Types of Programming Languages

Imperative vs Declarative

Structured Query Language (SQL)

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

Structured Query Language (SQL)

Declarative

Types of Programming Languages

Imperative vs Declarative

```
SELECT * FROM employees WHERE building='FIT';
```

Types of Programming Languages

Imperative vs Declarative

```
UPDATE employees SET building=NULL
                                WHERE building='FIT';
```

Types of Programming Languages

Imperative vs Declarative

```
CREATE TABLE employees (
  (...)
  building TEXT REFERENCES buildings(name)
                                  ON DELETE SET NULL
)
```

Types of Programming Languages

Imperative vs Declarative

- SQL specification says nothing about implementation
- Compiled to a sequence of actions called the Query Plan
- eg. SQLite's query plan is interpreted by its internal VM (the **Bytecode Engine**)

Types of Programming Languages

Imperative vs Declarative

HyperText Markup Language (HTML)

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

HyperText Markup Language (HTML)

Declarative

Types of Programming Languages

Imperative vs Declarative

Cascading Style Sheets (CSS)

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

Cascading Style Sheets (CSS)

Declarative

Types of Programming Languages

Imperative vs Declarative

Qt Designer's .ui files (XML-Based)

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

Qt Designer's .ui files (XML-Based)

Declarative

Types of Programming Languages

Imperative vs Declarative

Qt Designer's .ui files (XML-Based)

Declarative

Compiled to C++ header files

Types of Programming Languages

Imperative vs Declarative

C++

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

C++

Imperative

Types of Programming Languages

Imperative vs Declarative

x64

Imperative or Declarative?

Types of Programming Languages

Imperative vs Declarative

x64

Imperative

Types of Programming Languages

Imperative vs Declarative

x64

Imperative

What does this tell you?

Types of Programming Languages

Strong vs Weak Typing

A distinction with a fuzzier border compared to others

Types of Programming Languages

Strong vs Weak Typing

Discuss the following C fragment:

```
int a = 0; /* ok */
a = "string"; /* ?? */
```

Types of Programming Languages

Strong vs Weak Typing

Discuss the following C fragment:

```
int *a = NULL; /* ok */
a = "string"; /* ?? */
```

Types of Programming Languages

Strong vs Weak Typing

Discuss the following C++ fragment:

```
SomeClass a = 0; /* ok */
a = "string"; /* ?? */
```

Types of Programming Languages

Strong vs Weak Typing

Weak typing

When the **memory layout** of a variable can be mutated **implicitly**

Types of Programming Languages

Static vs Dynamic Typing

Discuss the following C++ fragment:

```
auto a = 0;
std::vector v{1,2,3};
```

What are the types of a and v?

Types of Programming Languages

Static vs Dynamic Typing

Static typing

When the **type** of a variable can be known at **compile-time**.

Types of Programming Languages

Functional vs Procedural

- Pure functions symbolize values
- Statements modify program state
- C++ is multi-paradigm

C++ Recap
OOP

OOP in C++ has 3 pillars:

- Encapsulation
- Inheritance
- Polymorphism

C++ Recap

OOP - Encapsulation

Objects: When data comes alive

- Public interface, private implementation

C++ Recap

OOP - Encapsulation

Objects: When data comes alive

- Public interface, private implementation
- Accessors: Used to ask questions to an object

C++ Recap

OOP - Encapsulation

Objects: When data comes alive

- Public interface, private implementation
- Accessors: Used to ask questions to an object
- Method calls (C++) vs Message Passing (Smalltalk)

C++ Recap

OOP - Encapsulation

C++ `struct` **vs** `class`;

```
struct S {
  int a; // public
};

class C {
  int a; // private
};
```

C++ Recap

OOP - Inheritance / Polymorphism

Static dispatch: the usual way

```
struct A {
    auto who() { return "A"; }
    auto greet() { return fmt::format("Hello, I am {}", who()); }
};

struct B: public A {
    auto who() { return "B"; }
};

int main() {
  A *a = new A();
  printf("%s\n", a->who());
}
```

C++ Recap

OOP - Inheritance / Polymorphism

Static dispatch: the usual way

```
struct A {
    auto who() { return "A"; }
    auto greet() { return fmt::format("Hello, I am {}", who()); }
};

struct B: public A {
    auto who() { return "B"; }
};

int main() {
  A *a = new B();
  printf("%s\n", a->who());
}
```

C++ Recap

OOP - Inheritance / Polymorphism

Static dispatch: the templates way

```
struct A {
    auto who() { return "A"; }
};

struct B {
    auto who() { return "B"; }
};

template <typename T>
auto greet(const T &t) {
    return fmt::format("Hello, this is {}", who());
}

int main() {
  auto v = new A();
  printf("%s\n", greet(*v));
  ...
```

C++ Recap

OOP - Inheritance / Polymorphism

Static dispatch: the templates way

```
struct A {
    auto who() { return "A"; }
};

struct B {
    auto who() { return "B"; }
};

template <typename T>
auto greet(const T &t) {
    return fmt::format("Hello, this is {}", who());
}

int main() {
  auto v = new B();
  printf("%s\n", greet(*v));
  ...
```

C++ Recap

OOP - Inheritance / Polymorphism

Dynamic dispatch

```
struct A {
    virtual auto who() { return "A"; }
    auto greet() { return fmt::format("Hello, I am {}", who()); }
};

struct B: public A {
    virtual auto who() override { return "B"; }
};

int main() {
  A *a = new A();
  printf("%s\n", a->who());
}
```

C++ Recap

OOP - Inheritance / Polymorphism

Dynamic dispatch

```
struct A {
    virtual auto who() { return "A"; }
    auto greet() { return fmt::format("Hello, I am {}", who()); }
};

struct B: public A {
    virtual auto who() override { return "B"; }
};

int main() {
  A *a = new B();
  printf("%s\n", a->who());
}
```

C++ Recap

Stack vs Heap

Consider the following C++ fragment:

```
int32_t *f(int32_t i) {
    int32_t *r = new int32_t(i);
    return r;
}
int main() {
    auto i = f(50);
    printf("%x %d\n", i, *i);
    return 0;
}
```

- `i` is allocated on the heap: Needs to be manually deleted.

C++ Recap

Stack vs Heap

Consider the following C++ fragment:

```
int32_t *f(int32_t i) {
    int32_t r(i);
    return &r;
}
int main() {
    auto i = f(50);
    printf("%x %d\n", i, *i);
    return 0;
}
```

- `r` is allocated on the stack: It's automatically deleted once out-of-scope

C++ Recap

Stack vs Heap

Consider the following C++ fragment:

```
int32_t *f(int32_t i) {
    int32_t r[1000000000LL];
    return &r[0];
}
int main() {
    auto i = f(50);
    printf("%x %d\n", i, *i);
    return 0;
}
```

- Stack is not infinite! By default, 8MB per thread on Linux

C++ Recap

Smart Pointers

`std::unique_ptr<T>`

- Ties stack behavior to heap memory
- Movable, not copyable
- Calls deallocator when variable goes out of scope.

C++ Recap

Smart Pointers

`std::shared_ptr<T>`

- `unique_ptr` with refcount
- Copyable (you can move it if you want)
- Calls deallocator when the ref# == 0;

Kiraz/COOL

```
class Cons inherits List {
  xcar : Int;
  xcdr : List;

  isNil() : Bool { false };
  init(hd : Int, tl : List) : Cons {
    {
      xcar <- hd;
      xcdr <- tl;
      self;
    }
  }
};
```

Future Work

- C++
- WebAssembly
- flex/bison