CMSC 124

DESIGN AND IMPLEMENTATION OF PROGRAMMING LANGUAGES

CNM PERALTA

LANGUAGE DESIGN ISSUES

LANGUAGE EVALUATION CRITERIA

Most computer scientists agree on four criteria for programming languages.

1.

Readability

How **easily** a language's **syntax** can be **read** and **understood**.

With the advent of the **software life-cycle** in the **1970s**, the **emphasis** on **code efficiency** was lessened in favor of **code maintenance**.

Ease of maintenance
is determined by
code readability.

FACTORS THAT CONTRIBUTE TO READABILITY

1.1.

Overall simplicity

`# of constructs ∝ 1/simplicity`

PROBLEMS WITH LANGUAGE SIMPLICITY

1.1.1.

Programmers usually learn **subsets** of a **large** and **complicated language**.

1.1.2.

Feature multiplicity

Example:

```
count = count + 1;
count++;
count += 1;
```

1.1.3.

Operator overloading

Example:

```
50 + 20
54.3 + 30.5
“hello” + “world”
```

1.1.4.

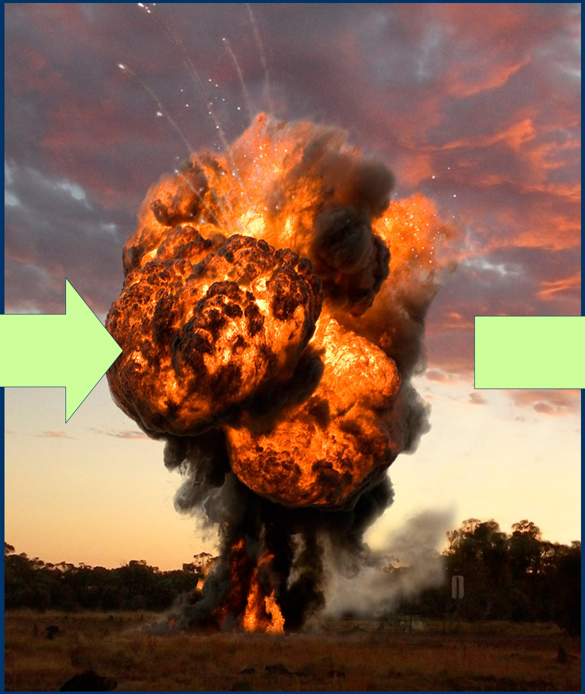
Too much simplicity

```
simple_loop:
# parameter 1: %rdi
..B1.1:                          # Preds ..B1.0
..___tag_value_simple_loop.1:    #2.1
        xorl      %eax, %eax     #3.19
        xorl      %edx, %edx     #5.8
        testq     %rdi, %rdi     #5.16
        jle       ..B1.5         # Prob 10% #5.16
                   #LOE rax rdx rbx rbp rdi r12 r13 r14 r15
..B1.3:                          # Preds ..B1.1 ..B1.3
        addq      %rdx, %rax     #6.5
        addq      $1, %rdx       #5.19
        cmpq      %rdi, %rdx     #5.16
        jl        ..B1.3         # Prob 82%   #5.16
..B1.5:                          # Preds ..B1.3 ..B1.1
        ret                      #8.10
        .align    2,0x90
```

1.2.

Orthogonality

Small set of primitive constructs



Control/data structures

EXAMPLE

{int, float, double, char}

{pointers, arrays}

{int *, float *, double *, char *, int[], float[], double[], char[], int **, float **, ...}

Primitives

must be

symmetric.

That is, **every possible combination** of primitives is **legal** and **meaningful**.

Language rule exceptions are indicative of **poor orthogonality**.

EXAMPLE

Which of the two is orthogonal?

```
A Register1, memory_cell
AR Register1, Register2
```

vs

```
ADDL operand1, operand2
```

More orthogonality begets **readability** by making **language syntax** more **regular**.

As we saw earlier, **pointers** are, in general, an **orthogonal concept**.

However, **C lacks orthogonality** in its **other constructs**.

ORTHOGONAL

- Functions can return any data type

NOT ORTHOGONAL

- Any data type except arrays can be returned, unless the array is inside a structure

ORTHOGONAL

- Array elements can have any data type

NOT ORTHOGONAL

- Array elements can have any data type except void

ORTHOGONAL

- Anything can be passed by value

NOT ORTHOGONAL

- Everything except arrays are passed by value

As always, too much of a good thing is still bad; **too much orthogonality** made **ALGOL 68** too **complicated**.

Functional languages are considered **simple** and **orthogonal** because everything is done using **function calls**.

1.3.

Data types

```
while(1) {...}
```

vs

```
while(true) {...}
```

1.4.

Syntax design

Special words

can make language syntax more readable.

EXAMPLE

```
if(condition) {
...
} else {
…
}
```

VS

```
if condition then
…
else
…
end if
```

Special words should **not** be **allowed** as **identifiers**.

2.

Writability

How **easily** a language can be used to **create programs** for a **chosen problem domain**.

Readability affects writability.

FACTORS THAT AFFECT WRITABILITY

2.1.

Both **simplicity** and **orthogonality** are again key in writability.

`more constructs` → `harder to learn everything`

Still, remember that too much of a good thing is bad.

2.2.

Abstraction

allows **structures** or **operations** to be designed in a way that the **details can be ignored**.

Data abstraction

is achieved using **classes**.

Process abstraction

is achieved using **functions** or **methods**.

2.3.

Expressiveness

means that **specifying operations** is **convenient**, not cumbersome.

EXAMPLE

```
MOVE a to b.
ADD a TO b GIVING c.
```

VS

```
b = a;
c = b + a;
```

3.

Reliability

Programs **perform** to their **specifications under all conditions**.

FACTORS THAT AFFECT RELIABILITY

3.1.

Type checking

by **testing** for **type errors** during **compile-time** or **run-time**.

3.2.

Exception handling

allows programs to **intercept run-time errors.**

3.3.

Aliasing

allows **two or more distinct names** to **access** the **same data cell**.

Aliasing

is

dangerous.

One of the most apparent **implementations** of **aliasing** is the concept of **pointers**.

Languages that **restrict aliasing** are **more reliable**.

3.4.

Readability affects reliability.

Difficult to read → Difficult to write/maintain → Less reliable

3.5.

Writability affects reliability.

Easy to write → Correctness more likely → More reliable

4.

Cost

How much must be invested to use the language.

FACTORS THAT AFFECT COST

4.1.

Cost of **programmers** to **use** the language.

4.2.

Cost of **writing programs** in the language.

The **impact** of this factor is **reduced** by **using** a **good programming environment**, like **IDEs**.

4.3.

Cost of **compiling programs** in the language.

For example, the **cost of Ada** was compounded by the fact that it was **hard to make a compiler for it**.

4.4.

Cost of **executing programs** written in the language.

Programs can be **optimized** (**decease program size** or **increase execution speed**).

4.5.

Cost of the **language implementation system**.

If the **compiler/system/hardware** on which the language needs to run is **expensive**, it will **hamper** the language's **popularity**.

4.6.

Cost of **poor reliability**.

4.7.

Cost of **maintaining programs**.

5.

Portability

Ease with which **programs** in a language can be **moved from one implementation to another**.

Non-standardized languages

are

difficult to port.

6.

Generality

Applicability of a language to a **wide range of applications**.

7.

Well-definedness

Completeness and **precision** of the language's **official defining document**.

In general, the **most important criteria** are **readability** and **writability**.

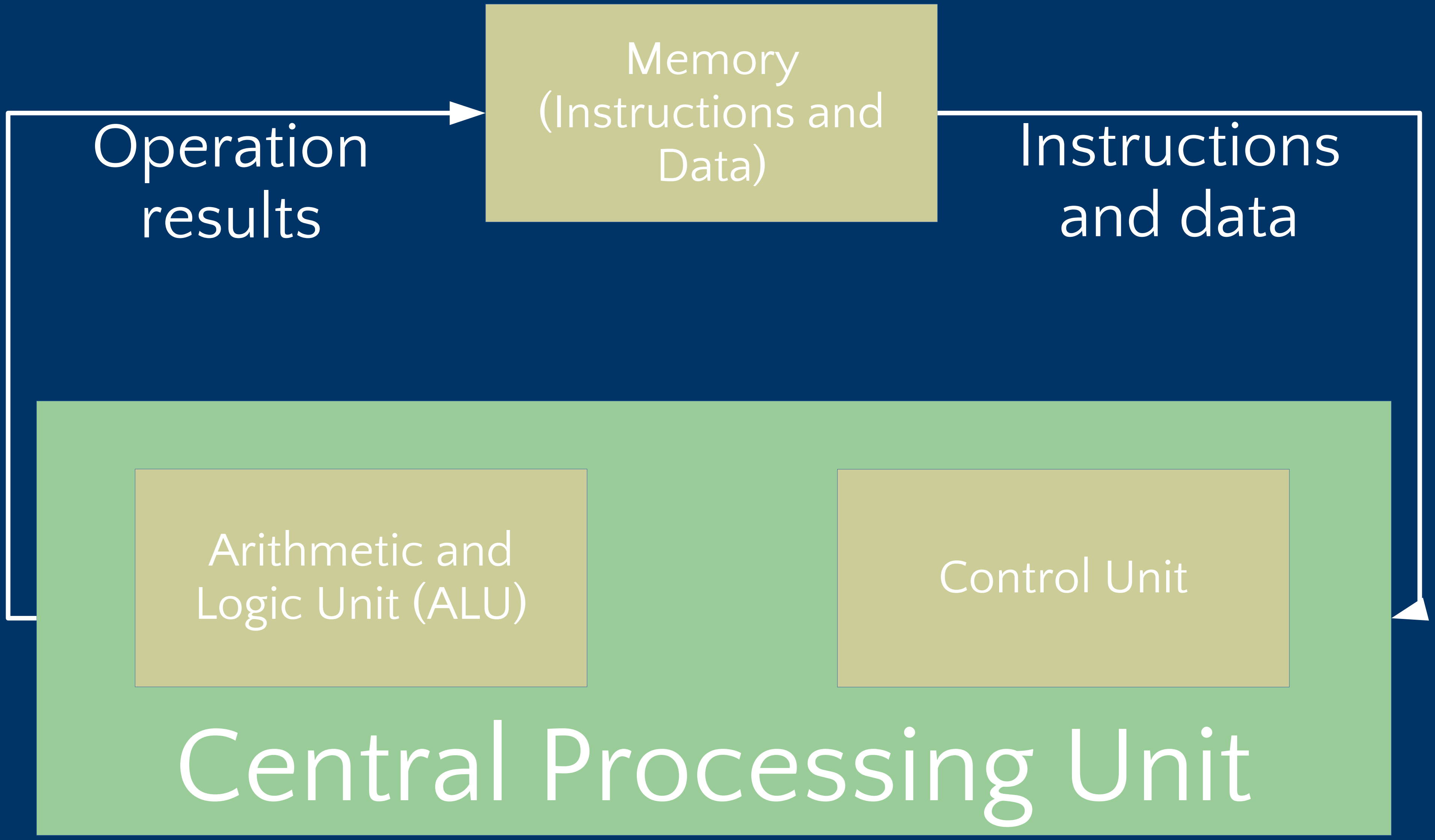
OTHER INFLUENCES ON LANGUAGE DESIGN

1.

Computer architecture

The prevalent computer architecture nowadays is called the **von Neumann architecture**, named after **John von Neumann**.

John von Neumann


Memory (Instructions and Data)
Operation results
Instructions and data
Arithmetic and Logic Unit (ALU)
Control Unit
Central Processing Unit

Programs were executed using the

fetch-execute cycle.

The **memory** was (and still is) a **contiguous array of cells**.

Thus, when a program is loaded for execution, its **instructions** are **stored** in **adjacent memory cells**.

Thus, when **higher-level languages** were being **designed**, the form of **execution** was the same – **step by step**.

These languages were then called

imperative languages.

Imperative languages are **more efficient** than **PLs of other paradigms** because it **conforms** directly to **von Neumann architecture**.

QUIZ

1. Give 2 factors that affect cost.
2. What paradigm of programming languages is the most efficient?
3. Give one PL concept that is orthogonal.
4. What concept is an implementation of aliasing?

QUIZ

5. How is process abstraction achieved?
6. What describes languages that have numerous ways to do the same thing?
7. What indicates poor orthogonality?
8. True or false? C is an orthogonal language.

QUIZ

BONUS: What needs to happen in the FIBA World Cup 2014 for the Philippine Team to advance to the next round?