CIS 343 - Structure of Programming Languages

Nathan Bowman

Based on slides by: Ira Woodring

Language Evaluation Criteria

(Follows the Sebesta Text Chapter 1)

We know now that different languages address different domains. Sometimes that may be enough for us to make a choice. For instance, we wouldn't want to use Java or Python to make an operating system, as they require an interpreter.

Sometimes though, languages may be more general purpose. So then, how can we compare them?

There are a lot of different criteria. The main ones we will discuss in this class are

- Readability
- Writability
- Reliability
- Cost

We will mostly focus on the first three in this class

- **Readability**
- **Writability**
- **Reliability**
- Cost

Readability

Refers to how easily the language can be read and understood. For instance, here is the source code for the original *Prince of Persia* game:

```
* hires
org = $ee00
 tr on
 lst off
*-------------------------------
*
*  PRINCE OF PERSIA
*  Copyright 1989 Jordan Mechner
*
*-------------------------------
 org org

 jmp boot3
 jmp cls
 jmp lay
 jmp fastlay
```

Full source code:

https://raw.githubusercontent.com/jmechner/Prince-of-Persia-Apple-II/master/01%20POP%20Source/Source/HIRES.S

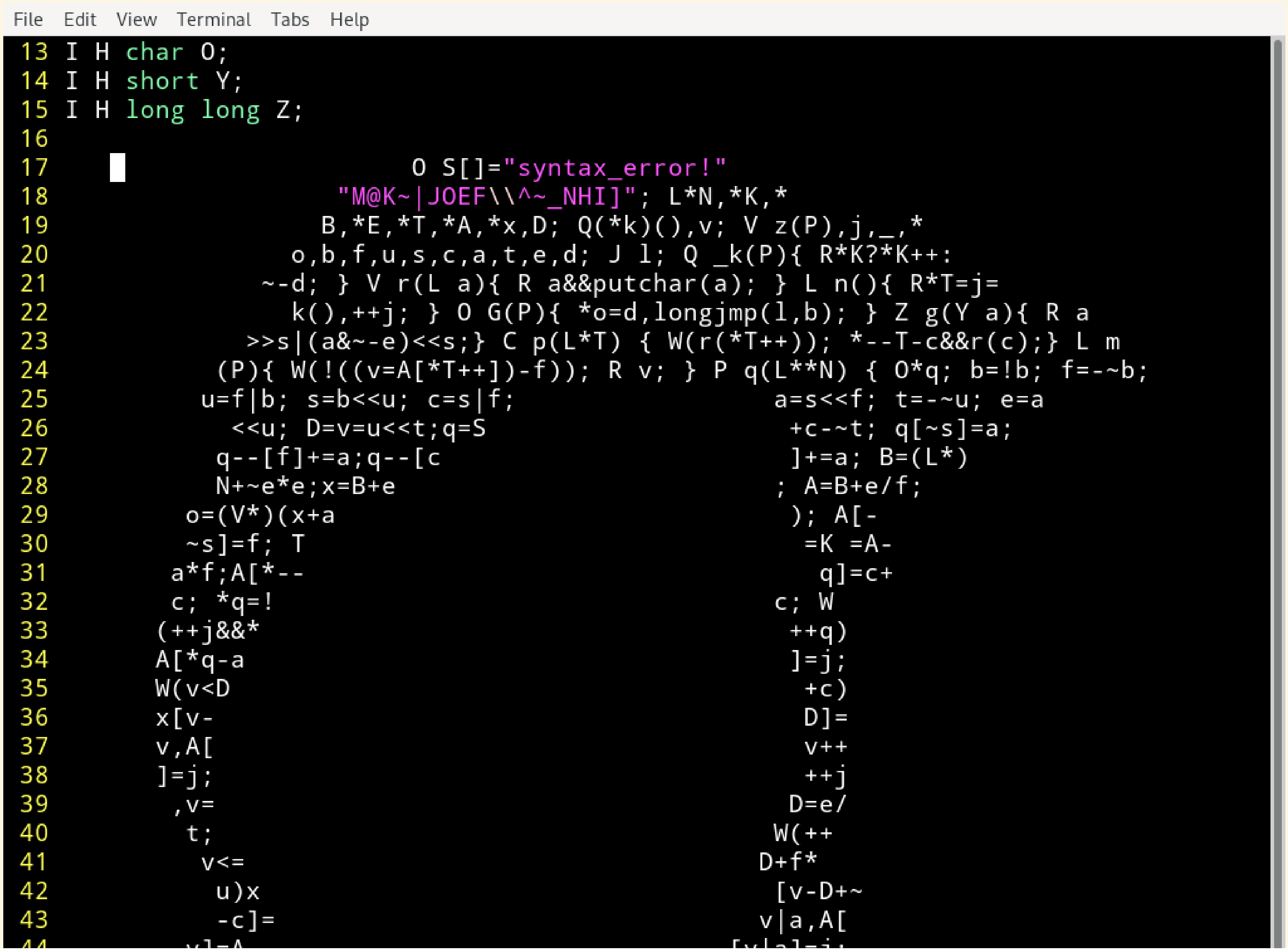
In case you're interested, here is the original game play on the Apple II:

https://www.youtube.com/watch?v=T5-06QnCHKY

Readability

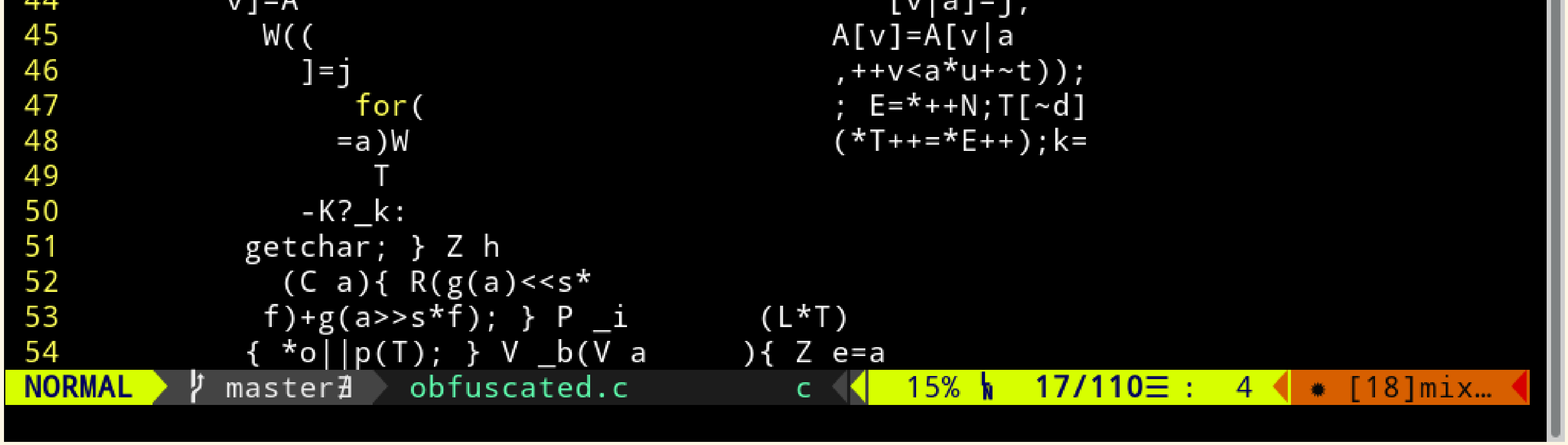
This Assembly Language code was very low-level. It lacked things like for-loops and other higher-level constructs. Because of these deficiencies it is much more cumbersome to express ideas concisely.

However, having higher-level constructs alone does not necessarily make a language more readable. Consider the following code:



```
I H char O;
I H short Y;
I H long long Z;

                       O S[]="syntax_error!"
                  "M@K~|JOEF\\^~_NHI]"; L*N,*K,*
                 B,*E,*T,*A,*x,D; Q(*k)(),v; V z(P),j,_,*
               o,b,f,u,s,c,a,t,e,d; J l; Q _k(P){ R*K?*K++:
             ~-d; } V r(L a){ R a&&putchar(a); } L n(){ R*T=j=
               k(),++j; } O G(P){ *o=d,longjmp(l,b); } Z g(Y a){ R a
            >>s|(a&~-e)<<s;} C p(L*T) { W(r(*T++)); *--T-c&&r(c);} L m
          (P){ W(!((v=A[*T++])-f)); R v; } P q(L**N) { O*q; b=!b; f=-~b;
         u=f|b; s=b<<u; c=s|f;                 a=s<<f; t=-~u; e=a
           <<u; D=v=u<<t;q=S                    +c-~t; q[~s]=a;
          q--[f]+=a;q--[c                       ]+=a; B=(L*)
          N+~e*e;x=B+e                         ; A=B+e/f;
        o=(V*)(x+a                              ); A[-
        ~s]=f; T                                 =K =A-
       a*f;A[*--                                  q]=c+
       c; *q=!                                 c; W
      (++j&&*                                   ++q)
      A[*q-a                                    ]=j;
      W(v<D                                      +c)
      x[v-                                       D]=
      v,A[                                       v++
      ]=j;                                       ++j
       ,v=                                      D=e/
        t;                                     W(++
         v<=                                  D+f*
          u)x                                  [v-D+~
          -c]=                                v|a,A[
        v]=A                                [v|a]=j;
```



```
 v]=A                                [v|a]=j;
   W((                            A[v]=A[v|a
     ]=j                          ,++v<a*u+~t));
        for(                      ; E=*++N;T[~d]
       =a)W                       (*T++=*E++);k=
         T
     -K?_k:
  getchar; } Z h
    (C a){ R(g(a)<<s*
   f)+g(a>>s*f); } P _i        (L*T)
  { *o||p(T); } V _b(V a      ){ Z e=a
NORMAL   master   obfuscated.c         c   15%   17/110 :   4   [18]mix…
```

This code is in `code_samples/obfuscated.c`.

Despite its appearance, it compiles with `gcc` with no errors.

This is from the International Obfuscated Code Contest.

http://www.ioccc.org/years.html#2015

Readability

This program is perfectly valid C code.

It includes higher-level concepts than Assembly Language typically provides, yet is anything but readable.

It is possible to write better or worse code in any language. What makes code from one *programming language* generally more readable than code from another language?

Readability

Simplicity.

- Having a small number of basic constructs
- Feature multiplicity
- Operator overloading

Readability - Simplicity

Having a small number of basic constructs

How many keywords and operators does the language have? The larger the number the less readable.

Consider COBOL:

https://www.ibm.com/support/knowledgecenter/en/SSZ

```
ACCEPT
ACCESS
ACTIVE-CLASS
ADD
ADDRESS
ADVANCING
AFTER
ALIGNED
ALL
ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
```

That's only half of the As!

Compare to Python:

https://www.programiz.com/python-programming/keywords-identifier

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Readability - Simplicity

Feature multiplicity

How many different ways can you accomplish the same operation? For instance, in many languages you may do any of the following to accomplish the same task:

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

The more ways to accomplish a task, the lower the readability.

Readability - Simplicity

Operator Overloading

How many meanings do the operators have?

Consider the C programming language. The * operator can have many meanings. It can be used for multiplication, to denote a pointer, to dereference a pointer, to help signify a comment, etc.

For instance:

```c
#include <stdio.h>
#include <stdlib.h>

/* Let's create some awesome code! */

int main(void){
        int * a = malloc(10 * sizeof(int));
        *(a + 0) = 42;
        *(a + 1) = *a + *a;
        int b = *a + 1;
        printf("%i\n\n", b);

}
```

What does this even output???

Readability - Orthogonality

A language is more orthogonal if the meaning of a primitive **does not depend on context** and if every possible combination of primitives is legal and meaningful.

Eh?

Readability - Orthogonality

For instance, in the Java programming language it is perfectly legal to return any data type from a function; in other languages though (such as C), attempting to return an array data type from a function is invalid code.

Eg:

Java

```java
public Car[] getCars(){
  Car[] cars = this.my_cars;
  return cars;                // Valid!
}
```

C

```c
int[] getValues(){
  int values[10];
  for(int i=0; i<10; i++)}
    values[i] = i;
  }
  return values;             // Not valid!
}
```

Readability - Orthogonality

What that means is that in this instance Java is more orthogonal.

C has a special rule that applies to returning arrays. Java does not treat this as a special case -- the general rule applies.

Readability - Orthogonality

Java is therefore more readable; in C we have to use pointers to return an array, but we also use pointers to return pointers from functions.

Understanding the programmer's original intent becomes harder.

Readability - Orthogonality

But, with everything there is a catch.

Too much orthogonality can be bad too. For instance, are conditionals allowed on the left side of an assignment statement?

```
5 * a = b             // May be valid (in some languages), but what
```

Readability - Data Types

Are they adequate to describe what needs to be described?

For instance, in Python we have a boolean type, but C does not. In C we must use an integer type to simulate a boolean:

```
goAhead = true
```

```c
int goAhead = 1;
```

(Depending on the version of C, there may be a Boolean library available)

Readability - Syntax

Are there easy-to-use facilities for denoting code groups?

Such as:

Many languages use braces and semi-colons

```
for(int i=0; i<10; i++){ ... }
```

Some languages use tabs or spaces

```
for i in range(0,10):
    ....
```

Others use matching words

```
if [ $a $eq 1 ]
then
  ...
fi
```

Readability - Syntax

Can special words be used as names of variables (hopefully not!)? Consider how confusing this might be:

```
int while = 10;
while( while < 100 ){
  ...
}
```

Readability - Syntax

Form and (fairly) obvious meaning.

Meaning should be apparent from the syntax, without the need for context

Consider this C code:

```c
static double pi = 3.14159;             // outside of a function defi
                                        // means only available to co
                                        // this file.

int main(int argc, char** argv){
  static int my_val = 1;                // inside a function; static
                                        // this is a compile time var
}
```

Writability

Measures how easily the language can be used to create programs.

The same issues that affect Readability also affect Writability.

But, there are a few more.

Writability - Simplicity and Orthogonality

The larger and more complex the language, the harder it is to use it to solve problems.

Consider how hard it is to debug a language in which every combination of primitives is legal. For instance, in C it can be very confusing to discover pointer errors.

Writability - Support for Abstraction

Abstraction refers to the ability to define and use complicated structures without worrying about the underlying details. In our Ruby matrix example earlier, we didn't need to know or care to know how the matrix was stored behind the scenes. All we cared about was that we were given a data structure that could efficiently store and operate on a matrix for us.

Writability - Support for Abstraction

There are two types of abstraction in computing, process abstraction and data abstraction.

Writability - Support for Abstraction

Process Abstraction is the use of subprograms (a function or module) to repeatedly solve a problem. For instance, if we can pass a custom comparison operator to a sorting function then we can use the same code to sort our data regardless of the data type.

Writability - Support for Abstraction

Data Abstraction allows us to define a data type and operations that work on that data type (such as a class in Java) that we can then reuse.

Writability - Expressivity

Are there convenient ways to accomplish computations, or do we need to rely on cumbersome methods?

Python, for instance, doesn't allow us to use the (nearly ubiquitous) prefix and postfix operators for variables.

```
count = 0
count++                     // NOT VALID!

count = count + 1           // Valid.
```

Reliability

Does a language always perform to its specifications, under all conditions?

Don't think reliability as in the mechanical sense. It isn't as if the language isn't going to "start-up" one day.

Reliability

Type Checking

Does the language test for type errors either during compilation or execution? For instance, is this an error?

```
int pi = 3.14159
```

Some languages will see this as a type error. Others may not.

Reliability

Exception Handling

If the language provides facilities to catch run-time errors, correct an issue and continue running, then it has exception handling.

```
try {
  ... Open a file on a disk for instance ...
}
catch (FileNotFoundException fnfex){
  ... recover; don't let the program crash ...
}
```

Reliability

Aliasing

Does the language allow for two or more names for the same memory location?

Many languages do, but it can be dangerous and harder to program with! This makes these languages less reliable.

```
int value = 42;
int & alias = value;    // alias now refers to the same location
                        // For all intents and purposes they are
                        // the same variable.
```

Writability - Cost

We won't be discussing this factor much in this class (it is more a Software Engineering topic), but Cost is a factor in language comparison.

Cost

- Training programmers to use the language. Often more powerful languages are more complicated to teach.
- How well the language fits the domain of the application can affect cost. If the domain is not well supported by the language it may take much longer to program the application.

Cost

- Compilation costs such as the price of the compiler (proprietary compilers can be expensive!), downtime waiting for compilation to complete, etc. affect cost.
- Cost can also be influenced by runtimes. Consider for instance renting time on a supercomputer. This is VERY expensive. A less efficient language may require longer run times to complete similar jobs.

Cost

- Reliability can affect cost as well.