CIS 343 - Structure of Programming Languages

Nathan Bowman

Based on slides by: Ira Woodring

Class Overview (Follows the Sebesta Text, Chapter 1)

What's the point?

Why do we study programming languages?

First Reason

Languages give us the ability to express what we are thinking.

It can be hard to convey something we don't have a direct translation for.

Using English for instance, how do you describe:

A relationship by fate or destiny.

Yuanfen

Chinese

The act of tenderly running your fingers through someone's hair.

Cafuné

Brazilian Portuguese

The happiness of meeting again after a long time.

Retrouvailles

French

The heart-wrenching pain of wanting someone you can’t have.

La Douleur Exquise

French

The sense upon first meeting a person that the two of you are going to fall into love.

Koi No Yokan

Japanese

A declaration of one’s hope that they’ll die before another person, because of how difficult it would be to live without them.

Ya’aburnee

Arabic

All of these were from http://bigthink.com/harpys-review/the-top-10-relationship-words-that-arent-translatable-into-english

On a similar note, the Sami people of Scandinavia and Russia have over 180 words for snow.

They have so many different ways to express snow, each with subtle nuances so that they can perfectly express ideas!

On the opposite end of the spectrum...

"By 2050—earlier, probably—all real knowledge of Oldspeak will have disappeared. The whole literature of the past will have been destroyed. Chaucer, Shakespeare, Milton, Byron—they'll exist only in Newspeak versions, not merely changed into something different, but actually contradictory of what they used to be. Even the literature of The Party will change. Even the slogans will change. How could you have a slogan like Freedom is Slavery when the concept of freedom has been abolished? The whole climate of thought will be different. In fact, there will be no thought, as we understand it now." ~ George Orwell, "1984"

Let's consider a computer example:

What does this code do?

```
mov eax, $x               // Move the value in $x to the register
beginning:
cmp eax, 0x0A             // Compare the value in eax to 0x0A (10
jg end                    // If they are the same, go to end.
inc eax                   // Increase the value in eax
jmp beginning             // Go to the beginning
end:
mov $x, eax               // Move the value in eax to $x
```

It is the same as this!

```
while(x <= 10){
  x++;
}
```

Why was it so much more complicated in the first example??

Assembly language lacks the concept of looping!

Looping is a higher-order idea.

Higher-order languages allow us to express these ideas more easily and succinctly.

Second Reason

Different programming languages have different tools. The tools we are provided with help us "attack the problem".

i.e., you *can* use a wrench as a hammer, but doesn't mean it is ideal (or that you should...)

For instance, some people may want to write web programs with HTML forms via C.

(taken from https://www.cs.tut.fi/~jkorpela/forms/cgic.html)

```c
int main(void)
{
    char *lenstr;
    char input[MAXINPUT], data[MAXINPUT];
    long len;
    printf("%s%c%c\n",
            "Content-Type:text/html;charset=iso-8859-1",13,10);
    printf("<TITLE\>Response</TITLE\>\n");
    lenstr = getenv("CONTENT_LENGTH");
    if(lenstr == NULL || sscanf(lenstr,"%ld",&len)!=1 || len > MA
        printf("<P\>Error in invocation - wrong FORM probably.");
    else {
        FILE *f;
        fgets(input, len+1, stdin);
        unencode(input+EXTRA, input+len, data);
        f = fopen(DATAFILE, "a");
```

Code is a nightmare!

Relies on hard coding some values sent, such as headers. This can cause problems as things change.

C doesn't natively understand the web. You will need to re-invent the wheel continuously to get anything done - even then your code is not likely to be safe.

So, we *can* use any language we want, but it doesn't mean we should. By using C we can't take advantage of well-written, robust methods that other languages may provide for web programming.

In C we don't have a great way to deal with matrices, for example. Ruby (and many, many other languages) provide us with custom classes and overloaded operators that make working with such constructs a breeze.

In C, we need to use multidimensional arrays (which technically don't even exist...).

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

int main(int argc, char** argv){
  int height = atoi(argv[1]);
  int width = atoi(argv[2]);
  int* matrix = malloc(width * height * sizeof(int));
  for(int i=0; i<height; i++){
    for(int j=0; j<width; j++){
      matrix[i][j] = 0;
    }
  }
}
```

Ruby (much like many higher order languages) provides better facilities (often through libraries).

```
require "Matrix"

h=Integer(ARGV[0])
w=Integer(ARGV[1])

m = Matrix.zero(h, w)
```

What if you are stuck using a specific language (for example, at your job?)

It can still be helpful to know what features are in other languages. Imagine you had never used an associative array (same thing as a dictionary or `HashMap`). That's an incredibly useful data structure! Once you know it exists, it may be worth implementing it in whatever language you have to use.

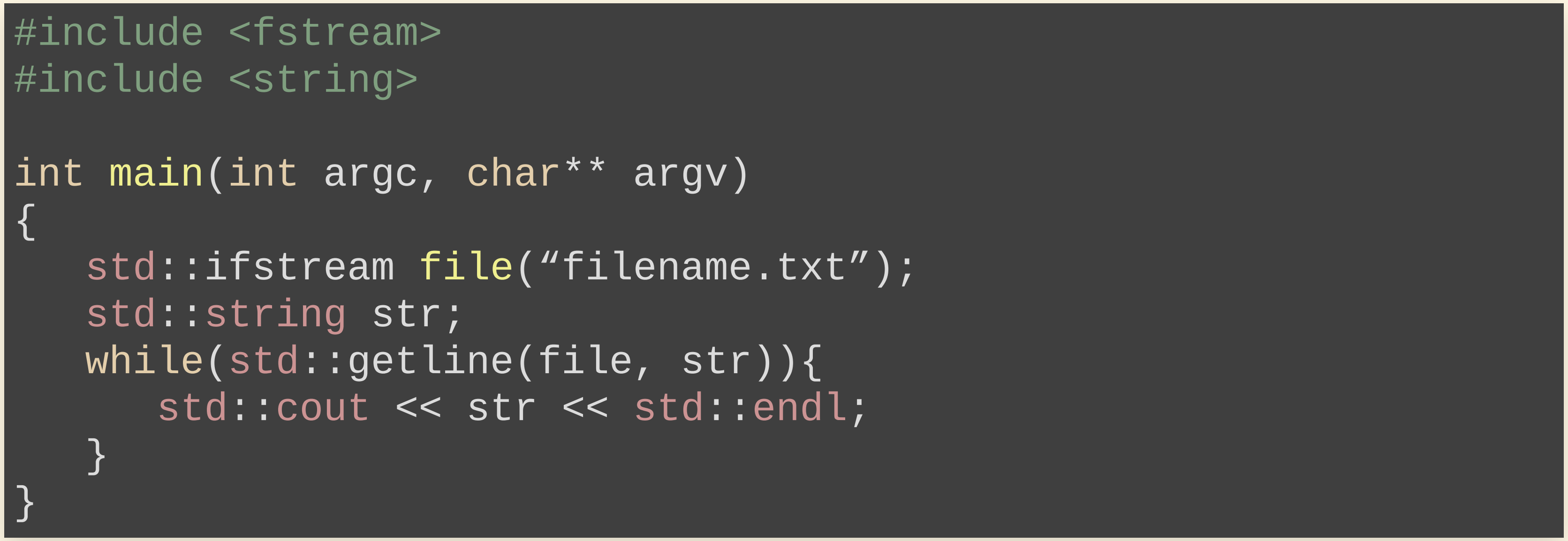
Though, as we've seen, it's better to use a language with built-in support than to "roll your own" whenever possible.

Third Reason

Programming languages address specific domains. No language is perfect for every domain. Studying languages help us to choose the most appropriate language for a job.

For instance, here are Python and C++ samples for how to read a file and print each line:

```
textFile = open(”filename.txt”, ”r”)
lines = textFile.readlines()
for l in lines:
    print l
```



```
#include <fstream>
#include <string>

int main(int argc, char** argv)
{
   std::ifstream file(“filename.txt”);
   std::string str;
   while(std::getline(file, str)){
      std::cout << str << std::endl;
   }
}
```

Python seems to be more English-like, while C++ includes some constructs that may be confusing (What is "::" for instance?).

C++ has a focus on speed and low-level hardware control.

Python is a more general-purpose language for the masses.

So what domains are there?

Science

- Must be fast
- Works with a lot of floating point numbers
- Must be precise

Business

- Emphasis is on reporting and output
- Lots of records and data

Artificial Intelligence

- Primary data structure may be lists, facts, or other special types.
- Symbolic

Systems Programming

- Must be fast
- Efficient
- Focuses on hardware

Many, many other specialized domains

- scripting tools
- file parsing
- custom tasks

Reason Four

It makes it easier for us to learn new languages.

Most of you should know Java; with only your Java knowledge you can probably still guess what the Python code is doing:

```python
class House:
  def __init__(self, s, c, p):
    self.sq_ft = s
    self.color = c
    self.price = p
```

It is the same as this:

```
public class House {
  public int sq_ft;
  public Color color;
  public float price;

  public House(int s, Color c, float p){
    this.sq_ft = s;
    this.color = c;
    this.price = p;
  }
}
```

Imagine you had never seen object-oriented programming before. Would the meaning of the Python code have been at all obvious?

```python
class House:
  def __init__(self, s, c, p):
    self.sq_ft = s
    self.color = c
    self.price = p
```

Think about how much easier it is to learn each of the following in a new language if you have seen the idea before (even if they are written sligthly differently in the new language):

- Basic control statements (`if`, `switch`, `for`)
- `HashMap` (Java)/`dict` (Python)
- Lambda functions
- Classes
- ...

Same thing holds true in natural languages. Assume you are a native English speaker learning Spanish for the first time. You need to learn all sorts of concepts (gender of nouns, different past tenses, subjunctive) that English either doesn't have or that English speakers usually ignore.

Learning Spanish doesn't teach you French, but it does make it a lot easier to learn. And, you may learn a lot about English along the way! The same is true of learning programming languages.

Reason Six

Knowledge of language implementations gives clues as to how best to use a particular language.

Allows us to understand why a particular feature or design pattern may be inefficient.

Calling small functions is inefficient, so may not be the best way to program in a context where performance is crucial.

Helps us in choosing languages based on design.

Python is interpreted (mostly), so it does not make sense to use as a systems language.

Understanding how a language is implemented can help identify and fix bugs that are otherwise difficult or impossible to fix.

Reason Seven

Helps us advance the field.

By understanding what has been created before, we have a better idea of what we don't need to recreate, what we can make better, and what we still need to create.