

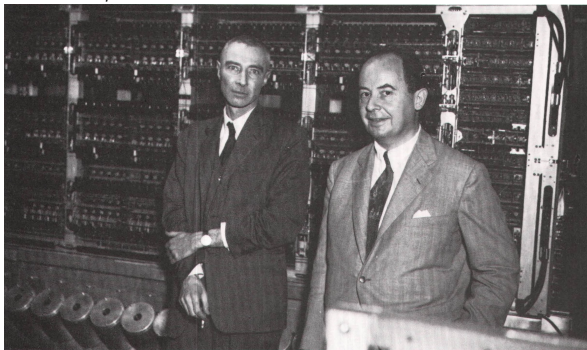
# Computational thinking

Kevin Schmidt, Susan Lindsey, Charlie Dey

Spring 2019

# Earliest computers

Historically, computers were used for big physics calculations, for instance, atom bomb calculations



# Hands-on programming

Very early computers were hardwired



# Program entry

Later programs were written on punchcards



# The first programming language

Initial programming was about translating the math formulas; after a while they made a language for that: FORmula TRANslation



# Programming is not simple

Programs can get pretty big:



*Margaret Hamilton, director of the Software Engineering Division, the MIT Instrumentation Laboratory, which developed on-board flight software for the Apollo space program.*



It's not just translating formulas anymore.

# Programming is everywhere

Programming is used in many different ways these days.

- You can make your own commands in *Microsoft Word*.
- You can make apps for your *smartphone*.
- You can solve the riddles of the universe using big computers.

This course is aimed at people in the last category.

# Is programming a craft or a science?

How about a 'discipline'?



# Computational thinking: elevator scheduling

Mathematical thinking:

- Number of people per day, speed of elevator  $\Rightarrow$  yes, it is possible to get everyone to the right floor.
- Distribution of people arriving *etc.*  $\Rightarrow$  average wait time.

Sufficient condition  $\neq$  existence proof.

Computational thinking: actual design of a solution

- Elevator scheduling: someone at ground level presses the button, there are cars on floors 5 and 10; which one do you send down?

Coming up with a strategy takes creativity!

# Exercise 1

Algorithms are usually not uniquely determined. There can be cleverness involved.

Four self-driving cars arrive simultaneously at an all-way-stop intersection. Come up with an algorithm that a car can follow to safely cross the intersection. If you can come up with more than one algorithm, what happens two cars using different algorithms meet each other?

# Computation and complexity

Looking up a name in the phone book

- start on page 1, then try page 2, et cetera
- or start in the middle, continue with one of the halves.

What is the average search time in the two cases?

Having a correct solution is not enough!

# Abstraction

- The elevator programmer probably thinks: 'if the button is pressed', not 'if the voltage on that wire is 5 Volt'.
- The Google car programmer probably writes: 'if the car before me slows down', not 'if I see the image of the car growing'.
- ... but probably another programmer had to write that translation.

A program has layers of abstractions.

# Data abstraction

What is the structure of the data in your program?

Stack: you can only get at the top item



Queue: items get added in the back, processed at the front



A program contains structures that support the algorithm. You may have to design them yourself.

# Do you have to know much about hardware?

Yes, it's there, but we don't think too much about it in this course.

<https://youtu.be/JEpsKnWZrJ8>

Advanced programmers know that hardware influences the speed of execution (see TACC's ISTC course).

# What is an algorithm?

*An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time*  
[A. Levitin, *Introduction to The Design and Analysis of Algorithms*, Addison-Wesley, 2003]

The instructions are in some language:

- We will teach you C++ and Fortran;
- the compiler translates those languages to machine language

# Program steps

- Simple instructions: arithmetic.
- Complicated instructions: control structures
  - conditionals
  - loops



# Program data

- Input and output data: to/from file, user input, screen output, graphics.
- Data during the program run:
  - Simple variables: character, integer, floating point
  - Arrays: indexed set of characters and such
  - Data structures: trees, queues
    - Defined by the user, specific for the application
    - Found in a library (big difference between C/C++!)

# Comparing two languages

Python vs C++ on bubblesort:

```
for i in range(n-1):           for (int i=0; i<n-1; i++)
    for j in range(n-i-1):      for (int j=0; j<n-1-i; j++)
        if numbers[j+1]<numbers[j]: if (numbers[j+1]<numbers[j])
            swaptmp = numbers[j+1]    int swaptmp = numbers[j+1]
            numbers[j+1] = numbers[j] numbers[j+1] = numbers[j];
            numbers[j] = swaptmp      numbers[j] = swaptmp;
                                     }
```

```
[] python bubblesort.py 5000
```

```
Elapsed time: 12.1030311584
```

```
[] ./bubblesort 5000
```

```
Elapsed time: 0.24121
```

# The right language is not all

Python with quicksort algorithm:

```
numpy.sort(numbers,kind='quicksort')
```

```
[] python arraysort.py 5000  
Elapsed time: 0.00210881233215
```

# Don't reinvent the wheel

- Can you choose a language that has the right tools?  
Python is way better than C++/Fortran for text processing and file manipulation.
- Is your algorithm part of a standard library or a library you can download/buy?  
Millions of programmers, just like you, have needed linear algebra algorithms. Most of what you could need already exists!