PTI CS 103

Prison Teaching Initiative

Mar 26, 2019

Contents

1 Introduction

Todo:

- Put together all references at the end
- Ask Sedgewick for permission to copy HelloWorld
- Put in references to lecture names for hardware and software instead of lecture 3 and lecture 4 (respectively)
- Add pictures
- Add course expectations (e.g. grading)
- Add more discussion questions
- Add section for the algorithmic thinking activity

1.1 Overview of course

Knowing just a little bit of computer science can get you started right away in actual applications. One of the goals of this course is to learn about the fascinating subject of computer science. Another is to develop algorithmic thinking skills that will help with day-to-day critical problem-solving skills. But perhaps the most important goal of the course is to develop coding skills, which will not only open up new job opportunities but also make you more effective in most areas of business.

In the first semester, we will spend the first two classes of each week on computer science theory and special topics. The final day of each week will be a lab day, where we actually start practicing coding skills.

In the second semester, we will start focusing more on practical coding, with a single day a week for theory and 2 lab periods per week for coding.

Broadly, we will cover the following topics:

- **How modern computers work**
  - Hardware
  - Software
  - Computer networks and information systems
- **Algorithms for quickly solving complex problems**

  - Searching
  - Sorting
- **Data structures**
  - Arrays
  - ArrayLists
- **Applications of Computer science**
  - Basic coding in Java
  - How to use productivity software

1.2 Brief history of computer science

Timeline:

- Invention of the abacus (2700-2300 BC, Sumerians)
- Design of first modern-style computer (Charles Babbage, 1837)
- Design of first computer algorithm (Ada Lovelace, 1843)
- Invention of first electronic digital computer (Konrad Zuse, 1941)
- Invention of the transistor (Bell labs, 1947)
- Invention of the first computer network (early Internet) (DARPA, 1968)
- Invention of the World Wide Web (Sir Tim Berners-Lee, 1990)

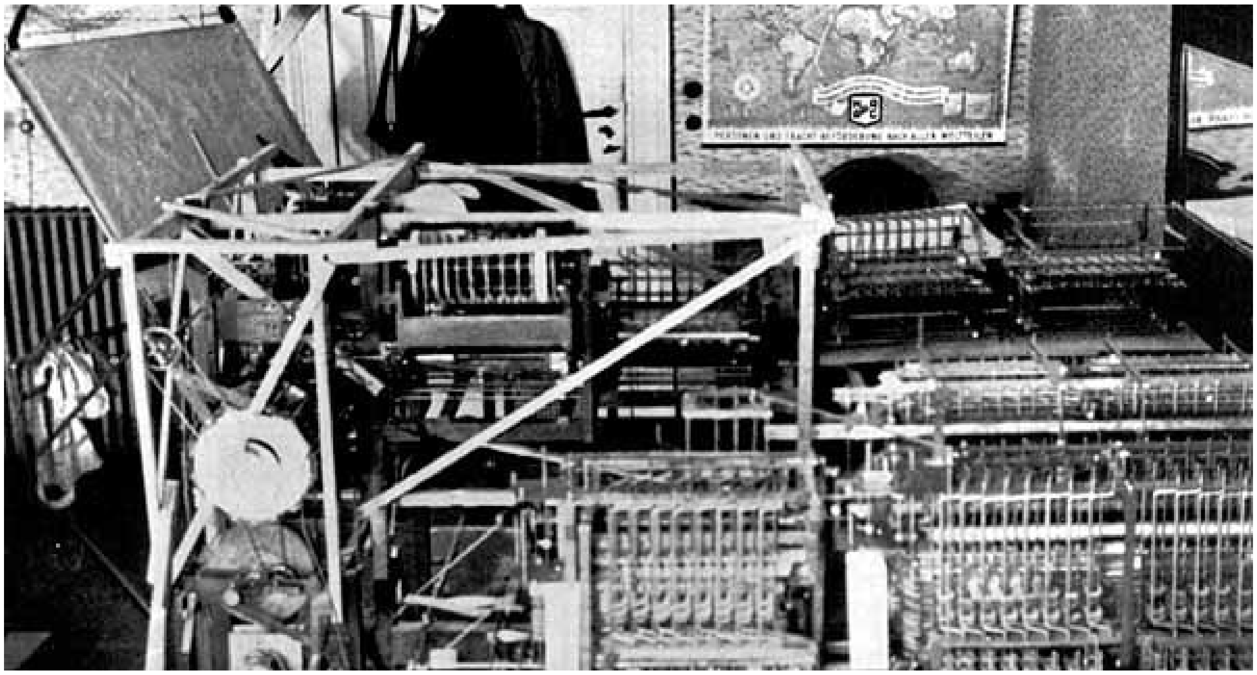

Fig. 1: Construction of Konrad Zuse's Z1, the first modern computer, in his parents' apartment.

1.3 Components of a computer

A computer is an electronic device used to process data. Its basic role is to convert data into information that is useful to people.

There are 4 primary components of a computer:

- Hardware
- Software
- Data
- User

1.3.1 Hardware

Computer hardware consists of physical, electronic devices. These are the parts you actually can see and touch. Some examples include

- Central processing unit (CPU)
- Monitor
- CD drive
- Keyboard
- Computer data storage
- Graphic card
- Sound card
- Speakers
- Motherboard

We will discuss these components in more detail in lecture 3.

1.3.2 Software

Software (otherwise known as “programs” or “applications”) are organized sets of instructions for controlling the computer.

There are two main classes of software:

- Applications software: programs allowing the human to interact directly with the computer
- Systems software: programs the computer uses to control itself

Some more familiar applications software include

- Microsoft Word: allows the user to edit text files
- Internet Explorer: connects the user to the world wide web
- iTunes: organizes and plays music files

While applications software allows the user to interact with the computer, systems software keeps the computer running. The operating system (OS) is the most common example of systems software, and it schedules tasks and manages storage of data.

We will dive deeper into the details of both applications and systems software in lecture 4.

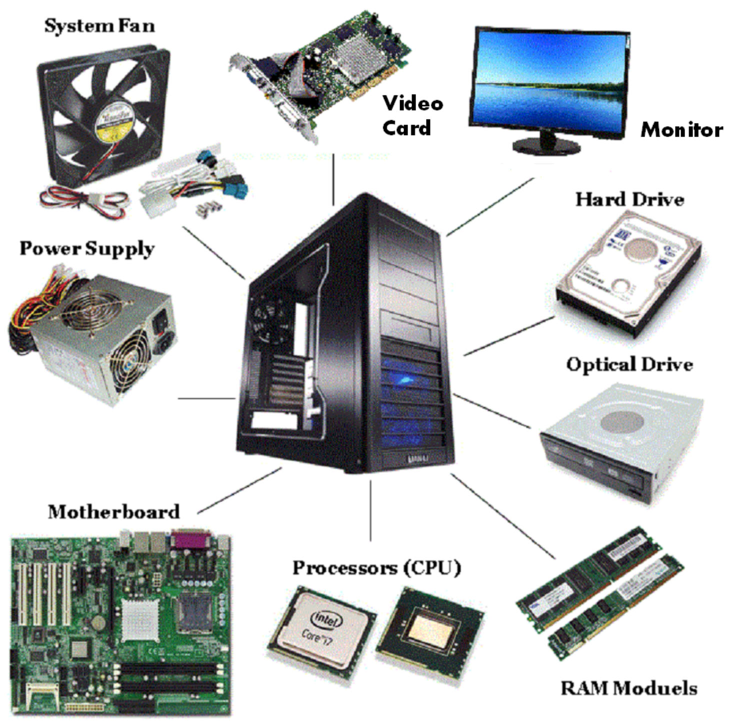

Fig. 2: Examples of hardware components of a personal computer.

1.3.3 Data

Data is fundamentally information of any kind. One key benefit of computers is their ability to reliably store massive quantities of data for a long time. Another is the speed with which they can do calculations on data once they recieve instructions from a human user.

While humans can understand data with a wide variety of perceptions (taste, smell, hearing, touch, sight), computers read and write everything internally as “bits”, or 0s and 1s.

Computers have software and hardware which allow them to convert their internal 0s and 1s into text, numerals, and images displayed on the monitor; and sounds which can be played through the speaker.

Similarly, humans have hardware and software used for converting human signals into computer-readable signals: a microphone converts sound, a camera converts pictures, and a text editor converts character symbols.

1.3.4 Users

Of course, there would be no data and no meaningful calculations without the human user. Computers are ultimately tools for making humans more powerful.

As we will see in the next section, however, different types of computers have different roles for the user.

1.4 Types of computers

1.4.1 Supercomputers

These are the most powerful computers out there. The are used for problems that take a long time to calculate. They are rare because of their size and expense, and therefore primarily used by big organizations like universities, corporations, and the government.

The user of a supercomputer typically gives the computer a list of instructions, and allows the supercomputer to run on its own over the course of hours or days to complete its task.

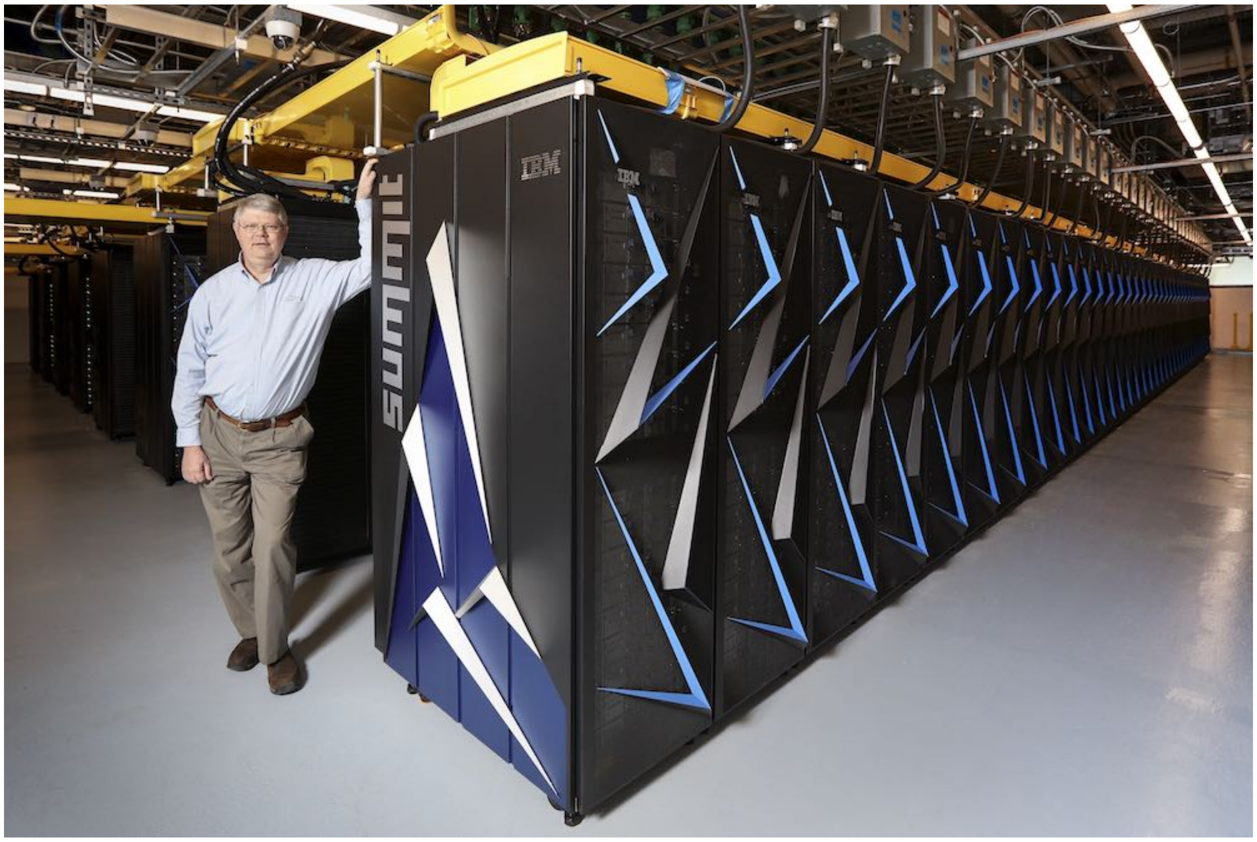

Fig. 3: Summit, a world-class supercomputing cluster at Oak Ridge National Laboratory in Tennessee.

1.4.2 Mainframe computers

Although not as powerful as supercomputers, mainframe computers can handle more data and run much faster than a typical personal computer. Often, they are given instructions only periodically by computer programmers, and then run on their own for months at a time to store and process incoming data. For example, census number-crunching, consumer statistics, and transactions processing all use mainframe computers.

1.4.3 Personal computers

These are the familiar computers we use to interact with applications every day. Full-size desktop computers and laptop computers are examples.

1.4.4 Embedded computers

In the modern "digital" age, nearly all devices we use have computers embedded within them. From cars to washing machines to watches to heating systems, most everyday appliances have a computer within them that allows them to function.

1.4.5 Mobile computers

In the past 2 decades, mobile devices have exploded onto the scene, and smartphones have essentially become as capable as standalone personal computers for many tasks.

1.5 Why computers are useful

Computers help us in most tasks in the modern age. We can use them, for example, to

- write a letter
- do our taxes
- play video games
- watch videos
- surf the internet
- keep in touch with friends
- date
- order food
- control robots and self-driving cars

  Question: What are some other tasks a computer can accomplish?

This is why the job market for computer scientists continues to expand, and why computer skills are more and more necessary even in non-computational jobs.

According to a Stackoverflow survey from 2018, 9% of professional coders on the online developer community Stack Overflow have only been coding for 0-2 years. This demonstrates two things:

1. The job market for people with coding skills is continually expanding
2. It doesn't take much to become a coder

Some examples of careers in computer science include

- IT management / consulting
- Game developer
- Web developer
- UI/UX designer
- Data analyst
- Database manager

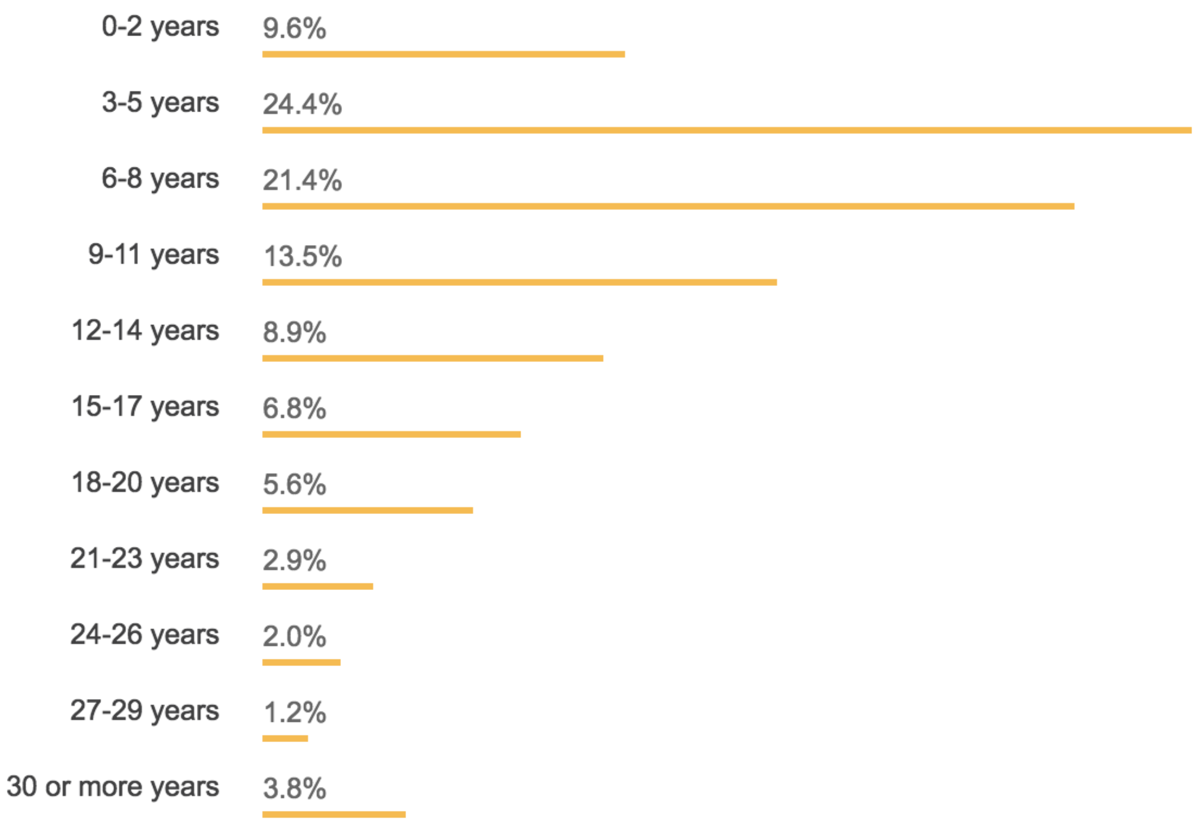

Fig. 4: Percentage of Professional Developers on Stack Overflow with various levels of coding experience.

1.6 First program

Please refer to the attached handout from the online resource corresponding to‘Computer Science: An Interdisciplinary Approach <https://introcs.cs.princeton.edu/java/11hello/>‘_, Robert Sedgewick and Kevin Wayne, “Your First Jaba Program: Hello World”.

> **Exercise:** Create your own program, `HelloMe.java`, that prints out “Hello `name`” with your own name in place of `name`.

1.7 References

- Computer Science: An Interdisciplinary Approach, Robert Sedgewick and Kevin Wayne.
- University of Wisconsin-Madison CS 202 Lectures, Andrea Arpaci-Dusseau.

2

Hardware, Variables and Data Types

Topics

- *Hardware*
  - *Computer Systems*
  - *Central Processing Unit (CPU)*
  - *Memory*
  - *Input and Output Devices*
- *Variables and Data Types*
  - *Variables*
  - *Basic Types*
  - *Operators*

2.1 Hardware

Overivew Sentence.. Todo…

2.1.1 Computer Systems

A computer systems is a collection of components that allows a user to perform the basic operations of input processing, storage, control, and output. Understanding how each of these components combine allows us to create powerful information processing tools. Today we will talk about each of these aspects in turn. Below, we see how some of these parts come together to form a computer system (similar to the ones you'll use to program in this course).

This lecture is meant to serve as an overview of several key parts of computer systems. We'll focus on the central processing unit (CPU), main and secondary memory, and input and output devices.

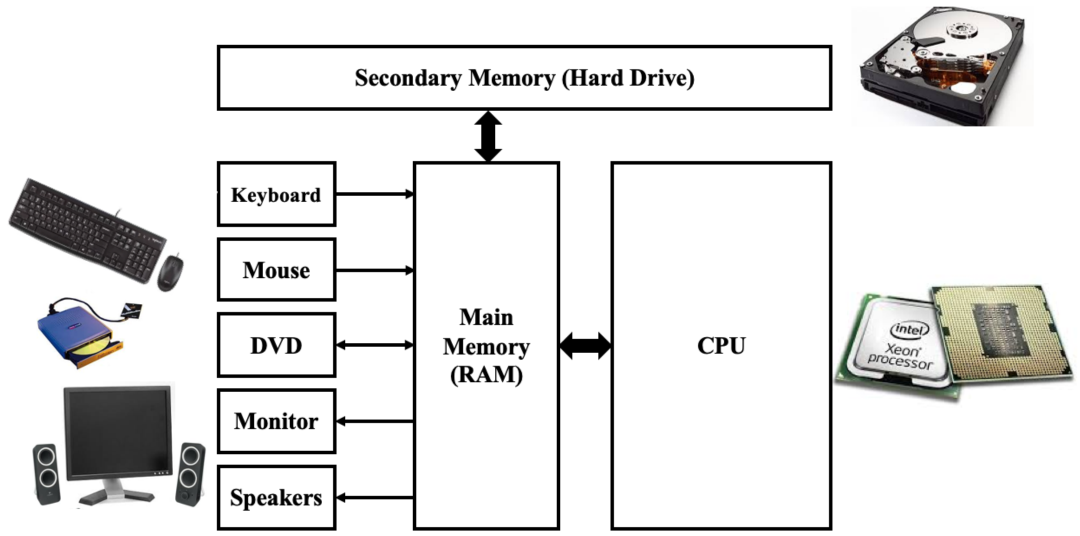

Fig. 1: Interconnected parts of a computer system (keyboard, mouse, monitor, DVD player / burner, speaker, hard drive, CPU).

2.1.2 Central Processing Unit (CPU)

The central processing unit (CPU) — processor, main processor, etc. — of a computer is the physical circuitry of a computer that performs instructions. The CPU is in charge of fetching, decoding, and executing all instructions. The basic building blocks of these instructions include arithmetic, logic, control, input, and output. The set of basic instructions available to the CPU effects, what the computer system can do (and how efficiently). Some CPU's are designed to handle a relatively small number of instructions (integer addition, multiplication, division, conditional branching, store to memory, load to memory). While others CPU's are designed to handle a large number of instructions (e.g. swap content of memory locations, increment and store, etc. in addition to the simpler instructions).

While these instructions can be quite simple, we build up complex behaviours by combining these actions into programs. When reasoning about how our programs will work, we should be aware of the limitations imposed by the hardware we use. In mathematics, we often consider things in a very abstract way. In models of computer systems, we often use unbounded arithmetic and allow a computer to have infinite memory; however, in practice we need to run programs on processors with bounded computational and memory capabilities. In most systems we encounter today, the CPU allows for 32 or 64 bit computations or memory read and writes in a single instruction. Because reading and writing to memory takes a long time (in comparison to arithemetic and logic instructions), almost all CPUs include a very small number of memory blocks on the physical CPU to store intermediate results of instructions, called registers. In modern CPUs this is further increased by allowing even more levels of memory, called caches (Level 1, Level 2, and Level 3), that trade off between speed of access and the size of the memory. We won't go in depth on how these work, but briefly mention them to help you understand how modern computer systems work to acheive greater efficiency.

2.1.3 Memory

One of the most important parts of a computer, is the memory. There are two major types of memory, Main Memory (RAM), and Secondary Memory (e.g. hard dists, solid-state drives, tape drives, etc.) Main memory is voalitile, meaning it only persists while the computer is on. Between turning a computer on and off the main memory is whiped. Secondary memory, on the other hand is persistent, it lasts between restarts of the computer system. Main memory, is where the CPU stores programs and data it needs to execute instructions. Secondary memory, is where persistent data such as files will be stored. When the CPU needs, it can bring data from secondary storage to main memory or persists data in main memory to secondary memory.

2.1.4 Input and Output Devices

The final parts of a computer system we'll talk about today, are input and output devices. Input and output devices are pervasive and necessary to allow humans to interact with computer systems. Otherwise, these systems would be a large block that computes some value, but we would have no method to inspect what this value is. Even the earliest computing systems consisted of a terminal (keyboard and textual interface) and a computer. How do these input and output devices work? Many input and output devices connect to a computer by interfacing with main memory. This is known as memory mapped input and output. The device and computer agree upon a known interface and communicate by storing values in the main memory. The exact method and values depend upon the agreed upon interface. Today there are a great variety of input and output devices including monitors, speakers, keyboards, mice, CD/DVD drives, usb drives, projectors, robotic arms, electric motors, etc.

2.2 Variables and Data Types

Recall that in our discussion about the CPU, we mentioned that we're able to store the results of computations to reference later. This allows us to build up more complex behaviors, and sometimes save a lot of repeated computations (and time). Now, we're going to switch from talking about hardware in general, and talk about some specifics of the Java programming language and how it handles variables and native data types (these align closely with the type of data CPUs can handle in a single instruction). Last week, we saw our first java program that printed out "Hello world!". Today, we'll introduce how to store data and operate on this stored data.

2.2.1 Variables

Variables in programming languages, are similar to the ones we've seen in mathematics. They store a value that can be referenced. Unlike many of the variables in math, we can update the value it holds during the course of the program. Consider the following block of code. We created a variable named x and assigned it the integer value 0. We then print it's value to the screen. Reassign the value of x to 3 and print the new value to the screen. The basic declaration of a variable in java consists of specifying a type, a name, and optionally an initial value (if no value is specified, the default value for the given type is used, instead). In this program, we created a variable with type int (the type of integer values).

```java
public class Example1 {
  public static void main(String[] args) {
    int x = 0;
    System.out.println(x);  // prints 0 to the screen
    x = 3;
    System.out.println(x);  // prints 3 to the screen
  }
}
```

2.2.2 Basic Types

What types of variables can we declare? We will go over a set of basic (or native or primitive) types that can be declared and how they differ. Below we see a table of each native type. Along side each type, we see how many bits each value takes (how much memory required to store a single value of the given type), what the minimum and maximum value / precision we can achieve with the given type. And an example declaration of a variable for the given type. Note that the void type is empty and we cannot create a variable of type void. Void represents the special type of return values for functions that do not return any value. The types byte, char, short, int, and long are integer types and represents integer values upto a given limit. Float and double are floating point numbers (numbers that allow for fraction parts, we will not discuss how these values are stored in memory). Boolean values represent truth (true or false).

Type	Bits	Min Value	Max Value	Precision	Example
byte	8	-128	127	-128 to 127	byte b = -3
char	16	0	2^16-1	0 to 67108863	char c = ‘a’
short	16	-2^15	2^15-1	-33554432 to 33554431	short s = -2
int	32	-2^31	2^31-1	-2,147,483,648 to 2,147,483,647	int i = 1241
long	64	-2^63	2^63-1	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l = -12
float	32	2^-149	(2-2^-23)2^127	1.4E-45 to 3.402,823,5E+38	float f = 3.1
double	64	2^-1074	(2-2^-52)2^1023	4.9E-324 to 1.797,693,134,862,315,7E+308	double d = 3.1
boolean	–	–	–	false, true	boolean b = true
void	–	–	–	–	–

2.2.3 Operators

How do we manipulate values of these types? We can modify values using operators. These values take one or more values and produces a new value. For interger type variables, we have the common arithmetic operators: addition (+), multiplication (x), division (/), modulo (%), increment (++), decrement (++), and assignment (=).

```java
public class Example2 {
  public static void main(String[] args) {
    int x = 0;
    ++x;  // equivalent to x = x + 1  (pre increment)
    System.out.println(x);   // prints 1 to the screen

    System.out.println(x++); // prints 1 to the screen then increments x (post␣
↪increment)
    System.out.println(x);   // prints 2 to the screen

    System.out.println(3 + 7); // prints 10
    System.out.println(4 - 6); // prints -2

    int y = (x * 5) % 3;     // y = (2 * 5) % 3 = 10 % 3 = 1   -- 10 = 3 * 3 + 1

    System.out.println(y + x); // pritns 3 to the screen
  }
}
```

In addition to the common arithmetic operations, we also have logic opeartions: equality (==), inequality (!=), less than (<), greater than (>), and (& or &&), or (| or ||), not (!), and xor (^). Below are table of how these operators perform. The single ampersand and bar and and or operations are non-short circuiting (e.g. they always evaluate both sides of the operator). This is important because of side-effects of these expressions (e.g. (0 == 1) && (1 / 0 != 3) vs (0 == 1) & (1 / 0 != 3) the first will work but the second will throw and errow when run).

A	B	A & B	A | B	A && B	A || B	A ^ B	!A
T	T	T	T	T	T	F	F
T	F	F	T	F	T	T	F
F	T	F	T	F	T	T	T
F	F	F	F	F	F	F	T

x	y	x == y	x != y	x < y	x > y
0	1	F	T	T	F
2	2	T	F	F	F
7	3	F	T	F	T

3

Arrays

Topics

3.1 Motivation

Consider this snippet of code:

```
if      (day ==  0) System.out.println("Monday");
else if (day ==  1) System.out.println("Tuesday");
else if (day ==  2) System.out.println("Wednesday");
else if (day ==  3) System.out.println("Thursday");
else if (day ==  4) System.out.println("Friday");
else if (day ==  5) System.out.println("Saturday");
else if (day ==  6) System.out.println("Sunday");
```

Question: What does this code do?

This code prints the day of the week after conditioning on the value of an integer *day*. But this code is repetitive. It would be useful if we had some way of creating a list of days of the week, and then just specifying which of those days we wanted to print. Something like this:

```
System.out.println(DAYS_OF_WEEK[day]);
```

To achieve this in Java, we need arrays.

3.2 Definition

An *array* is an ordered and fixed-length list of values that are of the same type. We can access data in an array by *indexing*, which means referring to specific values in the array by number. If an array has `n` values, then we think of it as being numbered from `0` to `n-1`:

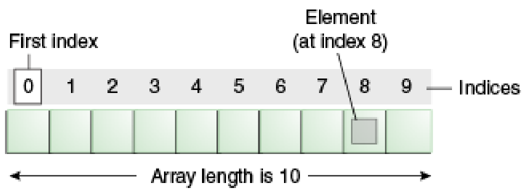

Fig. 1: Diagram of an array (Credit: https://www.geeksforgeeks.org/arrays-in-java/).

To *loop* or *iterate* over an array means that our program accesses every value in the array, typically in order. For example, if we looped over the array in the diagram, that would mean that we looked at the value at the 0th index, then the value at the 1st index, then the value at the 2nd index, and so on.

When we say that the array is "ordered" is that the relationship between an index and its stored value is unchanged (unless we explicitly modify it). If we loop over an unchanged array multiple times, we will always access the same values.

Arrays are *fixed-length*, meaning that after we have created an array, we cannot change its length. We will see in the next chapter [TK: confirm] that `ArrayLists` are an array-like data structure that allows for changing lengths.

Finally, all the values in an array must be of the same type. For example, an array can hold all floating point numbers or all characters or all strings. But an array cannot hold values of different types.

3.3 Creating arrays

The syntax for creating an array in Java has three parts: 1. Array type 2. Array name 3. Either: array size or specific values

For example, this code creates an array of size `n = 10` and fills it with all `0.0` s

```
double[] arr;                      // Declare array
arr = new double[n];               // Initialize the array
for (int i = 0; i < n; i++) {      // Iterate over array
    arr[i] = 0.0;                   // Initialize elements to 0.0
}
```

The key steps are: we first declare and initialize the array. We then loop over the array to initialize specific values. We can also initialize the array at compile time, for example

```
String[] DAYS_OF_WEEK = {
//   Indices:
//   0          1           2             3           4          5            6
    "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"
};
```

Notice the difference in syntax. When creating an empty array, we must specify a size. When initialize an array at compile time with specific values, the size is implicit in the number of values provided.

Finally, in Java, it is acceptable to move the brackets to directly after the type declaration to directly after the name declaration. For example, these two declarations are equivalent:

```
int arr[];
int[] arr;
```

3.4 Indexing

Consider the array `DAYS_OF_WEEK` from the previous section. We can *index* the array using the following syntax:

```
System.out.println(DAYS_OF_WEEK[3]);  // Prints "Thursday"
```

In Java, array's are said to use *zero-based indexing* because the first element in the array is accessed with the number `0` rather than *1*.

Question: What does `System.out.println(DAYS_OF_WEEK[1]);` print?

Question: What does this code do? What number does it print?

```
double sum = 0.0;
double[] arr = { 1, 2, 2, 3, 4, 7, 9 }
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
}
System.out.println(sum / arr.length);
```

3.5 Array length

As mentioned previously, arrays are *fixed-length*. After you have created an array, it's length is unchangeable. You can access the length of an array `arr[]` with the code `arr.length`.

Question: What does `System.out.println(DAYS_OF_WEEK.length);` print?

Exercise: Write a `for` loop to print the days of the week in order (Monday through Sunday) using an array rather than seven `System.out.println` function calls.

3.6 Default initialization

In Java, the default initial values for numeric primitive types is `0` and `false` for the `boolean` type.

Exercise: Consider this code from earlier:

```
double[] arr;
arr = new double[n];
for (int i = 0; i < n; i++) {
    arr[i] = 0.0;
}
```

Rewrite this code to be a single line.

3.7 Bounds checking

Consider this snippet of code.

Question: Where is the bug?

```
int[] arr = new int[100];
for (int i = 0; i <= 100; ++i) {
    System.out.println(arr[i]);
}
```

The issue is that the program attempts to access the value `arr[100]`, while the last element in the array is `arr[99]`.

This kind of bug is called an "off-by-one error" and is so common… well, it has a name. In general, an off-by-one-error is one in which a loop iterates one time too many or too few.

Question: Where is the off-by-one-error?

```
int[] arr = new int[100];
for (int i = 0; i < array.length; i++) {
    arr[i] = i;
}
for (int i = 100; i > 0; --i) {
    System.out.println(arr[i]);
}
```

Exercise: Fill in the missing code in this `for` loop to print the numbers in reverse order, i.e. `5, 4, 3, 2, 1`:

```
int[] arr = { 1, 2, 3, 4, 5 };
for (???) {
    System.out.println(arr[i]);
}
```

3.8 Empty arrays

Question: This code prints five values, one per line, but we never specified which values. What do you think it prints?

```
int[] arr = new int[5];
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

In Java, an unitialized or empty array is given a default value:

- For `int`, `short`, `byte`, or `long`, the default value is `0`.
- For `float` or `double`, the default value is `0.0`.
- For `boolean` values, the default value is `false`.
- For `char`, the default value is the null character `'\u0000'`.

Note that an array can be partially initialized.

Question: What does this code print?

```
char[] alphabet = new char[26];
alphabet[0] = 'a';
alphabet[1] = 'b';
for (int i = 0; i < alphabet.length; i++) {
    System.out.println(alphabet[i]);
}
```

3.9 Enhanced for loop

So far, we have seen how to iterate over arrays by indexing each element with a number:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (int i = 0; i < vowels.length; ++ i) {
    System.out.println(vowels[i]);
}
```

We can perform the same iteration without using indices using an "enhanced `for` loop" or `for-each` loops:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (char item: vowels) {
    System.out.println(item);
}
```

3.10 Exchanging and shuffling

Two common tasks when manipulating arrays are *exchanging two values* and *shuffling* values. (*Sorting* is more complicated and will be address later.)

To exchange to values, consider the following code:

```
double[] arr = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
int i = 1;
int j = 4;
double tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
```

Exercise: What are the six values in the array, in order?

To shuffle the array, consider the following code:

```
int n = arr.length;
for (int i = 0; i < n; i++) {
    int r = i + (int) (Math.random() * (n-i));
    String tmp = arr[r];
    arr[r] = arr[i];
    arr[i] = tmp;
}
```

Question: What does this code do:

```
for (int i = 0; i < n/2; i++) {
    double tmp = arr[i];
    arr[i] = arr[n-1-i];
    arr[n-i-1] = tmp;
}
```

3.11 Exercises

1. **Write a program that reverses the order of values in an array.**
2. **What is wrong with this code snippet?**

```
int[] arr;
for (int i = 0; i < 10; i++) {
    arr[i] = i;
}
```

3. **Rewrite this snippet using an enhanced `for-each` loop (for now, it is okay to re-define the array):**

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (int i = array.length; i >= 0; i--) {
    char letter = vowels[i];
    System.out.println(letter);
}
```

5. **Write a program that uses `for` loops to print the following pattern:**

```
1********

12*******

123******

1234*****

12345****

123456***

1234567**

12345678*

123456789
```

4. **Write a program** `HowMany.java` **that takes an arbitrary number of command line arguments and prints how many there are.**

3.12 References

- Computer Science: An Interdisciplinary Approach, Robert Sedgewick and Kevin Wayne.

4

ArrayLists

Topics

4.1 Motivation

4.2 ArrayLists

A _collection_ is a group of objects. Today, we'll be looking at a very useful collection, the `ArrayList`. A _list_ is an ordered collection, and an `ArrayList` is one type of list.

Create a class NameTracker and follow along in it.

Before we can use an ArrayList, we have to import it:

```
import java.util.ArrayList;
```

Next, we call the constructor; but we have to declare the type of object the `ArrayList` is going to hold. This is how you create a new `ArrayList` holding `String` objects.

```
ArrayList<String> names = new ArrayList<String>();
```

Notice the word "String" in angle brackets: "". This is the Java syntax for constructing an ArrayList of String objects.

We can add a new String to names using the `add()` method.

```
names.add("Ana");
```

Question: Exercise: Write a program that asks the user for some names and then stores them in an ArrayList. Here is an example program:

```
Please give me some names:
Sam
Alecia
Trey
Enrique
Dave
Your name(s) are saved!
```

We can see how many objects are in our ArrayList using the size() method.

```
System.out.println(names.size()); // 5
```

Exercise: Modify your program to notify the user how many words they have added.

```
Please give me some names:
Mary
Judah
Your 2 name(s) are saved!
```

Remember how the `String.charAt()` method returns the `char` at a particular index? We can do the same with names. Just call `get()`:

```
names.add("Noah");
names.add("Jeremiah");
names.add("Ezekiel");
System.out.println(names.get(2)); // "Ezekiel"
```

Exercise: Update your program to repeat the names back to the user in reverse order. Your solution should use a for loop and the size() method. For example:

```
Please give me some names:
Ying
Jordan

Your 2 name(s) are saved! They are:
Jordan
Ying
```

Finally, we can ask our names `ArrayList` whether or not it has a particular string.

```
names.add("Veer");
System.out.println(names.contains("Veer")); // true
```

Exercise: Update your program to check if a name was input by the user. For example:

Please give me some names: Ying Jordan

Search for a name: Ying Yes!

An `ArrayList` can hold any type of object! For example, here is a constructor for an `ArrayList` holding an instance of a `Person` class:

```
ArrayList<Person> people = new ArrayList<Person>();
```

where `Person` is defined as

```
public class Person {

```

(continues on next page)

(continued from previous page)

```
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
        return this.age;
    }
}
```

Exercise: Modify our program to save the user's input names as Person instances. Rather than storing String objects in the ArrayList, store Person objects by constructing them with the input name. You'll need to use the Person constructor to get a Person instance!

4.3 In-class assignment

Write a class BlueBook that tells the user the price of their car, depending on the make, model, and year. You should use Car.java and the stencil file provided, BlueBook.java.

Your program depends on what cars your BlueBook supports, but here is an example program:

```
What is your car's make?
Toyota
What is your Toyota's model?
Corolla
What is your Toyota Corolla's year?
1999
```

Your 1999 Toyota Corolla is worth $2000.

Bonus exercise: Notify the user if the car is not in your BlueBook.

Bonus exercise: Clean up main by putting your code for creating the ArrayList in a separate method. What type should the method return?

Bonus exercise: If the car is not in the BlueBook, ask the user to input the relevant data, construct a new Car instance, add it to your ArrayList.

4.4 References

1. https://github.com/accesscode-2-1/unit-0/blob/master/lessons/week-3/2015-03-24_arraylists.md