COMP-202

Introduction to Computing 1

Section 2

Ernesto Posse

ENGMC 11

TTR 10:00 - 11:30

Course website:

`http://www.cs.mcgill.ca/~cs202`

What this course is about

- This course is an introduction to *computer programming*
- Computer programming: solving problems involving information by means of a computer

What this course is *not* about

- This course is ***not*** about...
  - ...how to use a computer
  - ...how to use software applications
  - ...how to use the Operating System
  - ...how to send e-mail
  - ...how to surf the Web
  - ...how to create Web pages
  - ...how to fix your printer
  - ...how to become a hacker
  - ...how to manage a computer system (installing software, fixing problems, etc.)
- There is no course in Computer Science about how to use computers, in the same way that there is no course in Mechanical Engineering that teaches how to drive a car or operate some machinery.

Objectives

- To learn:
  - ...a methodology to understand and solve problems involving information
  - ...how to think computationally
  - ...how to create simple algorithms
  - ...how to design and implement computer programs using the Java programming language
  - ...how to solve problems in an Object-Oriented manner
- This is neither a “computers course” nor a “Java course.”

Fundamental concepts

- Algorithms: An algorithm is a well-defined procedure to solve a problem

- Programming Language: A formal language used to express algorithms

- Programs: The realization of some algorithm in a programming language

Why is computer programming useful

- General benefits
  - Introduces a structured way of thinking, analysing and solving problems
- Applications
  - Engineering and Physical sciences: modelling and simulation
  - Biological sciences: Bioinformatics, Eco-system modelling
  - Geography, Enviromental Studies and Urbanism: Geographic Information Systems
  - Economics: Economic forecasting and analysis, Economic modelling
  - Management: Databases, Information Systems, Process optimization
  - Software development

Who is this course for

- Required for:
  - Major in Software Engineering
  - Major in Computer Engineering
  - Major in Electrical Engineering
  - Minor in Computer Science
  - ...others
- Anyone interested in learning how to develop software

Prerequisites

- An upper-level CEGEP Mathematics course or equivalent
- Logical thinking: being able to reason, to deduct and to infer
- Familiarity with using computers:
  - Editing and saving text files
  - File system: using directories/folders (navigating, copying files, etc.)

Is this course easy?

- No
- This course is considered easy by approximately 5% to 10% of previous students
- The workload is heavy, specially after assignment 2.
- The exams are long
- Course withdrawal: please consult the Undergraduate Course Calendar

Grading system

- The marks will be divided as follows:
  - Assignments: 25%
  - Midterm: 20%
  - Final: 55%
- Assignments:
  - INDIVIDUAL
  - There are 6 assignments
  - To be submitted electronically through WebCT
- Midterm: covers all topics up to the day before the exam
- Final: covers all topics

Plagiarism

- All coursework must be done INDIVIDUALLY
- You may not work in groups: if you need help, contact a TA or instructor
- Each assignment and exam must be marked with your full name and student id
- By putting your name and id you are stating that the assignment is entirely your own work
- Students who put their name on programs, modules, or parts of programs that are not entirely their own work will be referred to the appropriate Associate Dean who will assess the need for further disciplinary action.

Office hours

- Where: McConnell Engineering Building, room 202
- When: Wednesdays from 2:00pm to 4:00pm
- ...or by appointment (e-mail)
- ...but you can come by (almost) anytime
- E-mail: eposse@cs.mcgill.ca (Do not use WebCT e-mail)
- Teaching Assistants (TAs): office hours TBA
- Treat the TAs respectfully

What you will need

- The textbook: Java Software Solutions by John Lewis and William Loftus

- Available at the McGill Bookstore (you may use old and used editions.)

- Access to a computer:
  - Either at home
  - ...or at the Trottier labs (Trottier Building, 3rd floor)
  - ...or anywhere else

- Software:
  - The Java Software Development Kit (j2sdk)
  - An IDE (Integrated Development Environment)

If you use the Trottier labs

- Located at the third floor of the Lorne M. Trottier Building
- All machines are Linux or Unix boxes (no Windows or Macintosh computers)
- Openning an account: (only if you are officially registered)
  - Enter username and password
    - username: newuser
    - password: newuser
  - Answer what you are asked
  - If you need extra help, ask for the consultant
- These machines have already installed the j2sdk and NetBeans, and IDE
- To learn about Linux/Unix, there will be seminars next week at the beginner and intermmediate levels.

If you use another machine

- You need to install the j2sdk:
  - It comes with the book
  - It can also be downloaded for free from
    `http://java.sun.com`
    - Download J2SE, Desktop, any version after 1.3.1
- You need to install an IDE
  - For example a free IDE for Windows is JCreator LE, which can be downloaded from:
    `http://www.jcreator.com`
- Install the IDE after installing the j2sdk

Hints for not suffering in this course

- READ CAREFULLY
- Don't wait until the last minute to do your assignments
- Do not copy any part of anyone else's assignment (current or past students)
- Do not work in groups. If you have difficulties, contact the instructor or the TAs.
- Do not expect to be given every single detail. Expect to deduce things on your own.
- Experiment!

Computers and Information

- What is a computer?
- How computers work?
- How is information stored/represented in a computer?

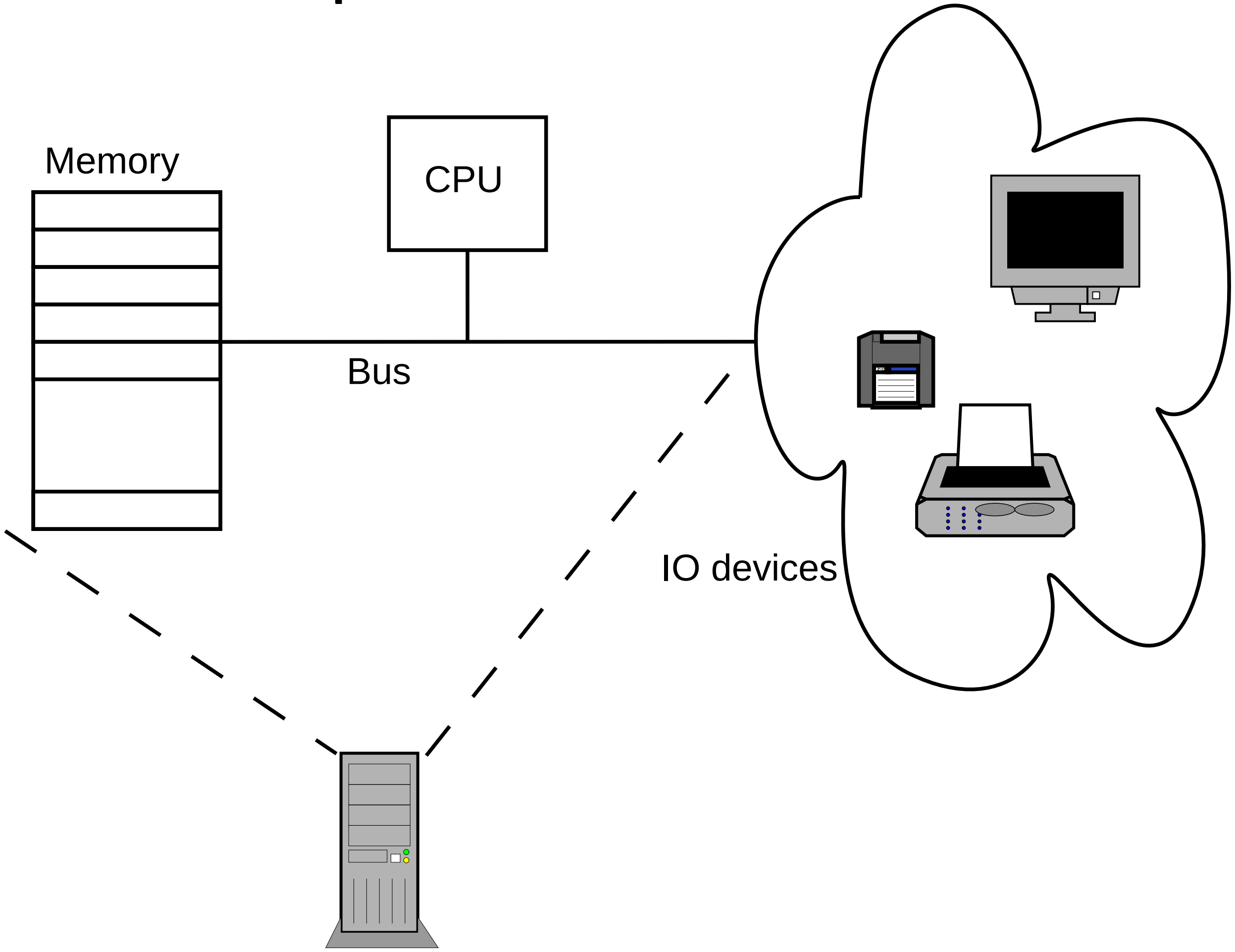
Computers and Information

- A computer is a machine that can perform many different tasks
- ...but the tasks are not predefined
- A computer is a machine which can execute instructions which we give to it
- Therefore, if we can change the set of instructions we can tell the computer to do different things

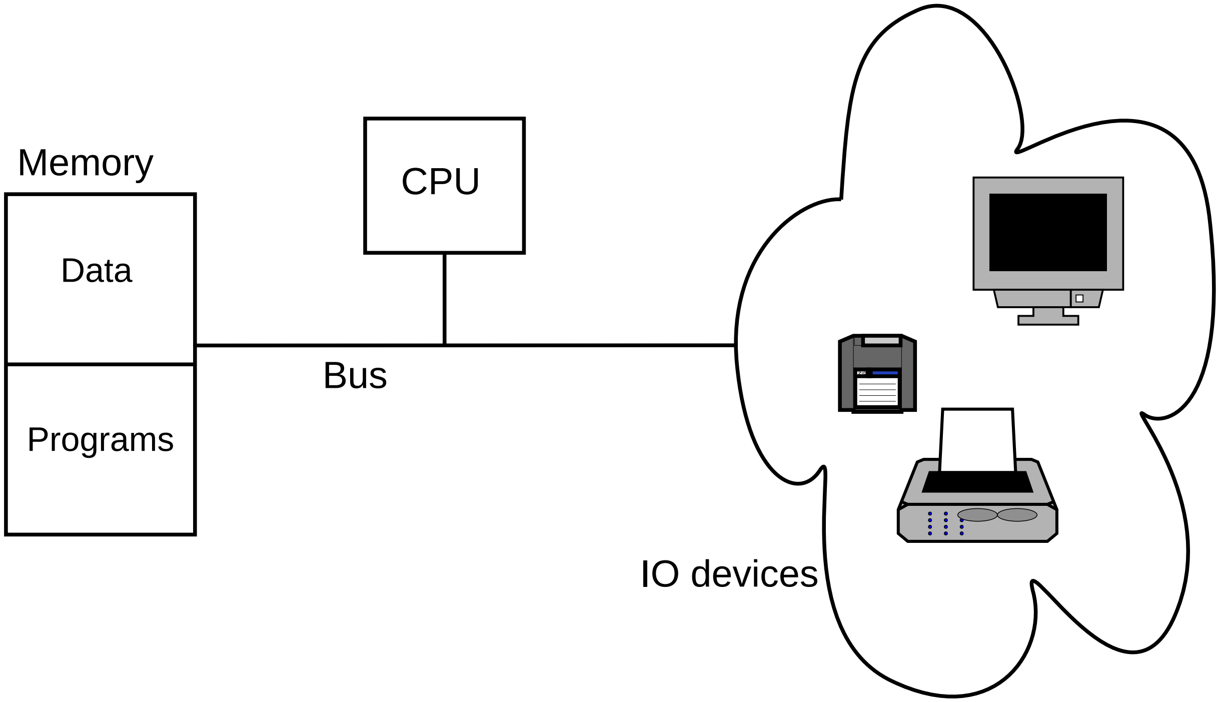
Computers and Information

- Hardware vs Software
  - Hardware: circuits
  - Software: programs
    - Application programs
    - Operating System
- Computer components (Hardware):
  - CPU (Processor)
  - Memory (RAM/ROM/etc.)
  - Input-Output Devices (IO, Keyboard, Screen, Mouse, Printer, etc.)
  - Note: Disks (Hard Disks, CDs, etc.) are IO devices which store data, so they can be seen also as a kind of memory
  - Bus

Computers and Information



Computers and Information

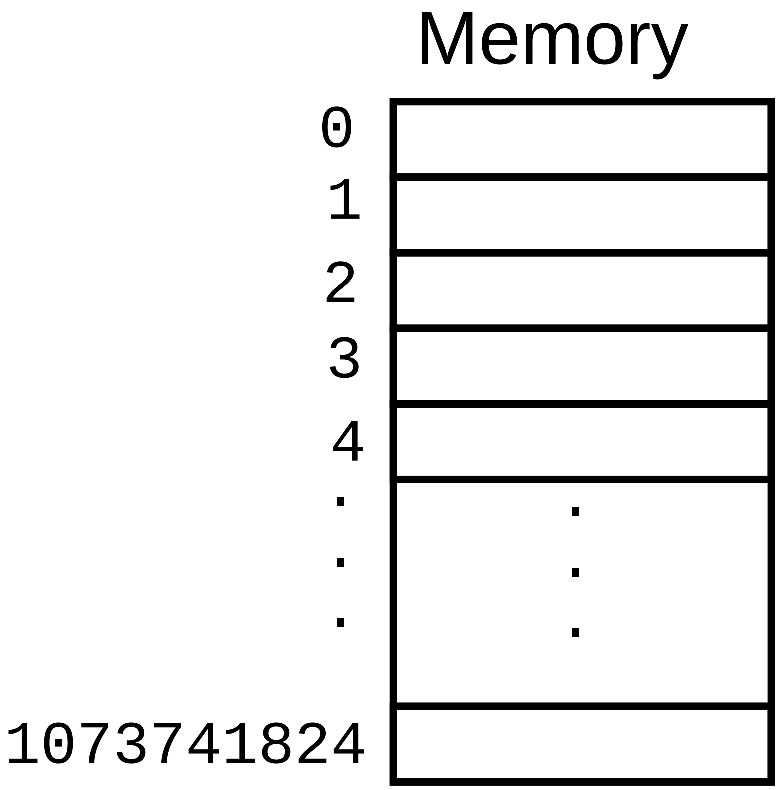


Memory, data and programs

- Memory:
  - Memory is a very long (but finite) list of *cells* or *memory locations*
  - Each cell is assigned a unique *address* (a natural number)
  - Each cell contains some piece of information (of fixed size)
  - Some cells contain just data
  - Other cells contain instructions for the processor

- Programs
  - A program is a sequence of instructions
  - A program can be stored in memory
  - Programs manipulate the data which is stored in other memory locations
  - Programs are data which is *executable* by the processor (Von Neumann Architecture)

Memory, data and programs



Program execution

- The CPU keeps track of the program which it is executing
- The CPU takes each program instruction (one at a time) from memory, ...
- ... and executes the instruction...
- ...which may involve:
  - making an arithmetic computation
  - reading from or writing to memory
  - reading from or writing to an IO device
  - other operations (changing the next instruction to be executed)
- The traffic of data between the components is through the bus

Data representation

- Data is stored in memory
- Memory cells store numbers
- Numbers represent different types of information:
  - letters
  - text
  - graphics/pictures/images
  - sound
  - movies
  - structured data (e.g. databases, tables, etc.)
  - mathematical functions
  - programs

Data representation

- How are numbers represented in memory?
  - A computer is an electronic device made out of wires
  - Wires have a voltage
  - We can think of the voltage of a wire as the *state* of the wire
  - Different voltages can represent different values
- To simplify things, digital circuits have wires with only two possible voltages (e.g. 0 and +3V).
- Hence a single digital wire can represent something that has two possible values: a *bit* (true/false, on/off, up/down, yes/no, ...)
- The bit is the fundamental unit of information: 0 and 1

Data representation

- To represent more complex things, we can form sequences of bits: 000101, 1101001, 00, 111111111111, 1010101010, ...

- Bit sequences represent binary numbers: numbers in base 2:
  - 0 is 0
  - 1 is 1
  - 2 is 10
  - 3 is 11
  - 4 is 100
  - 5 is 101
  - ...

- Binary numbers are ordinary numbers which are written with only two digits (0 and 1) instead of ten (0 to 9).

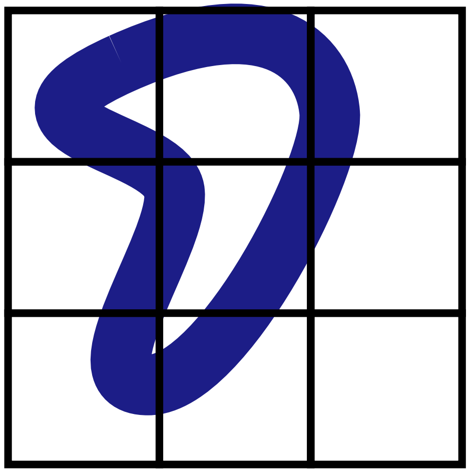
Data representation

- Bit sequences can represent other things: e.g. letters
  - 'a' is 10001001
  - 'b' is 10001010
  - 'c' is 10001011
  - ...
- And therefore text: "bca" is 100010101000101110001001

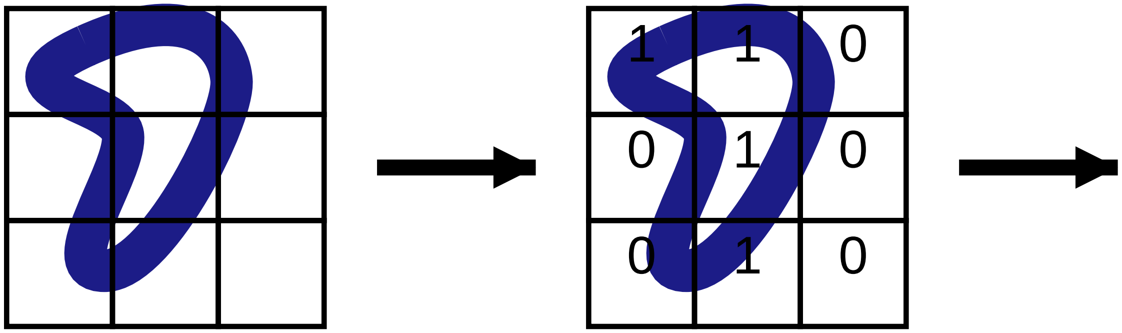
Data representation

- They can also represent images

Data representation



Data representation



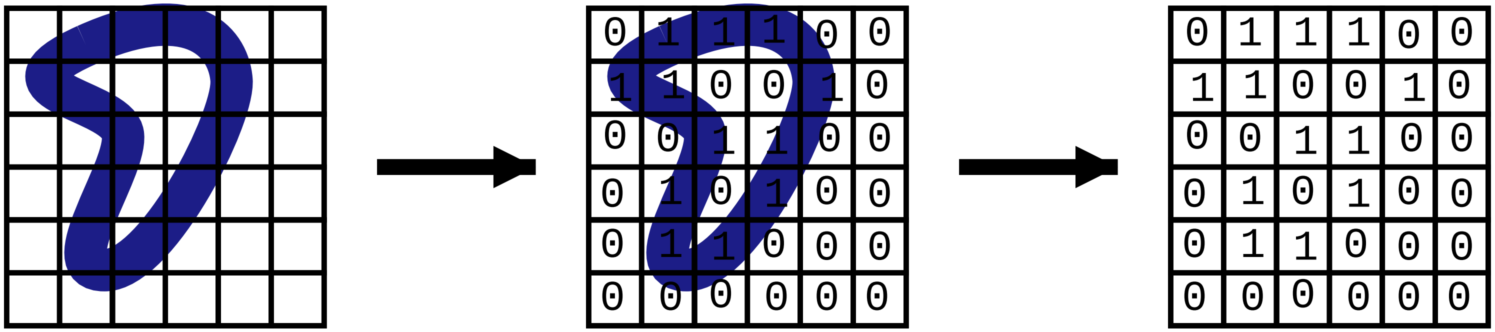
1	1	0
0	1	0
0	1	0

110010010

or

100111000

Data representation



0	1	1	1	0	0
1	1	0	0	1	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	0	0	0	0	0

```
011100110010001100010100011000000000
                 or
010000110110101010101100010000000000
```

Data representation

- Bit sequences can represent other things: e.g. letters
  - 'a' is 01100001 which is 97 in decimal
  - 'b' is 01100010 which is 98
  - 'c' is 01100011 which is 99
  - ...
  - 'e' is 01100101 which is 101
  - ...
- And therefore text: “hello” is 01101000 01100101 01101100 01101100 01101111
- or ... 104 101 108 108 111

Data in memory

- Each memory cell can contain a fixed number of bits: 32 bits, or 64 bits
- Some terminology:
  - A sequence of bits with the size of a memory cell is called a *word*
  - A sequence of 8 bits is called a *byte*
  - A sequence of 1024 bytes is called a *kilobyte* of KB ($1024 = 2^{10}$)
  - A sequence of 1024 kilobytes is a *megabyte* (MB)
  - A sequence of 1024 megabytes is a *gigabyte* (GB)
  - A sequence of 1024 gigabytes is a *terabyte* (TB)

Data in memory

- How much information can be represented by *n* bits?
  - 1 bit: 2 possible values
  - 2 bits: 4 possible values
  - 3 bits: 8 possible values
  - 4 bits: 16 possible values
  - ...
  - n bits: 2^n possible values

- To represent the English alphabet we need ? bits

- If we have q possible values, how many bits do we need?: $\lceil log_2 q \rceil$

- The ASCII code uses 8 bits: letters, decimal digits, symbols, etc.

- Unicode uses 16 bits: accents, different alphabets, more symbols, etc.

Binary to decimal conversion

- Problem: given a sequence of *n* bits, what is the decimal (base 10) representation of the sequence?
- Examples:
  - 0000000000 is 0
  - 1 is 1
  - 0010 is 2
  - 11 is 3
  - 100 is 4
  - ...
- Notation: Let the sequence be $b = b_{n-1}b_{n-2}\cdots b_2b_1b_0$ (indexed from right to left, starting from 0)

Binary to decimal conversion

- Solution:

$$dec(b) = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

- Examples:

$$\begin{aligned} dec(1101) &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\ &= 8 + 4 + 1 \\ &= 13 \end{aligned}$$

$$\begin{aligned} dec(101101) &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\ &= 32 + 8 + 4 + 1 \\ &= 45 \end{aligned}$$

Decimal to binary conversion

- Problem: given a natural number (positive integer or 0) *m*, what is its binary (base 2) representation?

- Analysis:
  - Given *m*, find the sequence of bits $b = b_{n-1}b_{n-2}\cdots b_2b_1b_0$ such that $m = dec(b)$
  - Inputs: a natural number *m*
  - Ouput: a sequence of bits b such that $m = dec(b)$

Decimal to binary conversion

- Algorithm:

1. Divide m by 2. This yields a quotient q_0 and a remainder r_0 which is 0 or 1. (why?)

2. Divide q_0 by 2. This yields a quotient q_1 and a remainder r_1

3. Divide q_1 by 2. This yields a quotient q_2 and a remainder r_2

4. ...

5. ... until you reach 0

6. Then let $b = r_l r_{l-1} \cdots r_2 r_1 r_0$

Decimal to binary conversion

- Example: Consider $m = 114$

1. Divide 114 by 2. The result is 57 and the remainder is 0
2. Divide 57 by 2. The result is 28 and the remainder is 1
3. Divide 28 by 2. The result is 14 and the remainder is 0
4. Divide 14 by 2. The result is 7 and the remainder is 0
5. Divide 7 by 2. The result is 3 and the remainder is 1
6. Divide 3 by 2. The result is 1 and the remainder is 1
7. Divide 1 by 2. The result is 0 and the remainder is 1
8. The result is 1110010

- To check this:

$$\begin{aligned} dec(1110010) &= 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^1 \\ &= 64 + 32 + 16 + 2 \\ &= 114 \end{aligned}$$

Decimal to binary conversion

```
1. Let $b$ be ‘‘’’ (the empty sequence)

2. Let $q$ be $m$

3. While $q$ is not 0 repeat the following:

   (a) Let $new\_q$ be $q$ divided by 2, and
   (b) Let $r$ be the remainder of $q$ divided by 2
   (c) Append $r$ in the front of the sequence $b$
   (d) Set $q$ to be $new\_q$
   (e) Repeat (from line 3)
```

Decimal to binary conversion

- Trace of execution

- Example: Consider the case $m = 44$

iteration	q	new_q	r	b
0	44			“”
1	22	22	0	“0”
2	11	11	0	“00”
3	5	5	1	“100”
4	2	2	1	“1100”
5	1	1	0	“01100”
6	0	0	1	“101100”

Decimal to binary conversion

- Trace of execution
- Example: Consider the case $m = 26$

iteration	q	new_q	r	b
0	26			“”
1	13	13	0	“0”
2	6	6	1	“10”
3	3	3	0	“010”
4	1	1	1	“1010”
5	0	0	1	“11010”

Elements of algorithms

- Variables to store values (such as numbers, sequences, etc.)

- Instructions organized and executed in sequence: order of execution matters

- Instructions for:
  - computing values (e.g. divide by)
  - assigning values to variables
  - repeating a set of instructions
  - etc.

Elements of algorithms

- Solving a problem: (General methodology)

1. Stating the problem
2. Understanding the problem -> Analysis
3. Designing a possible solution -> Algorithm
4. Implementing the algorithm using a programming language

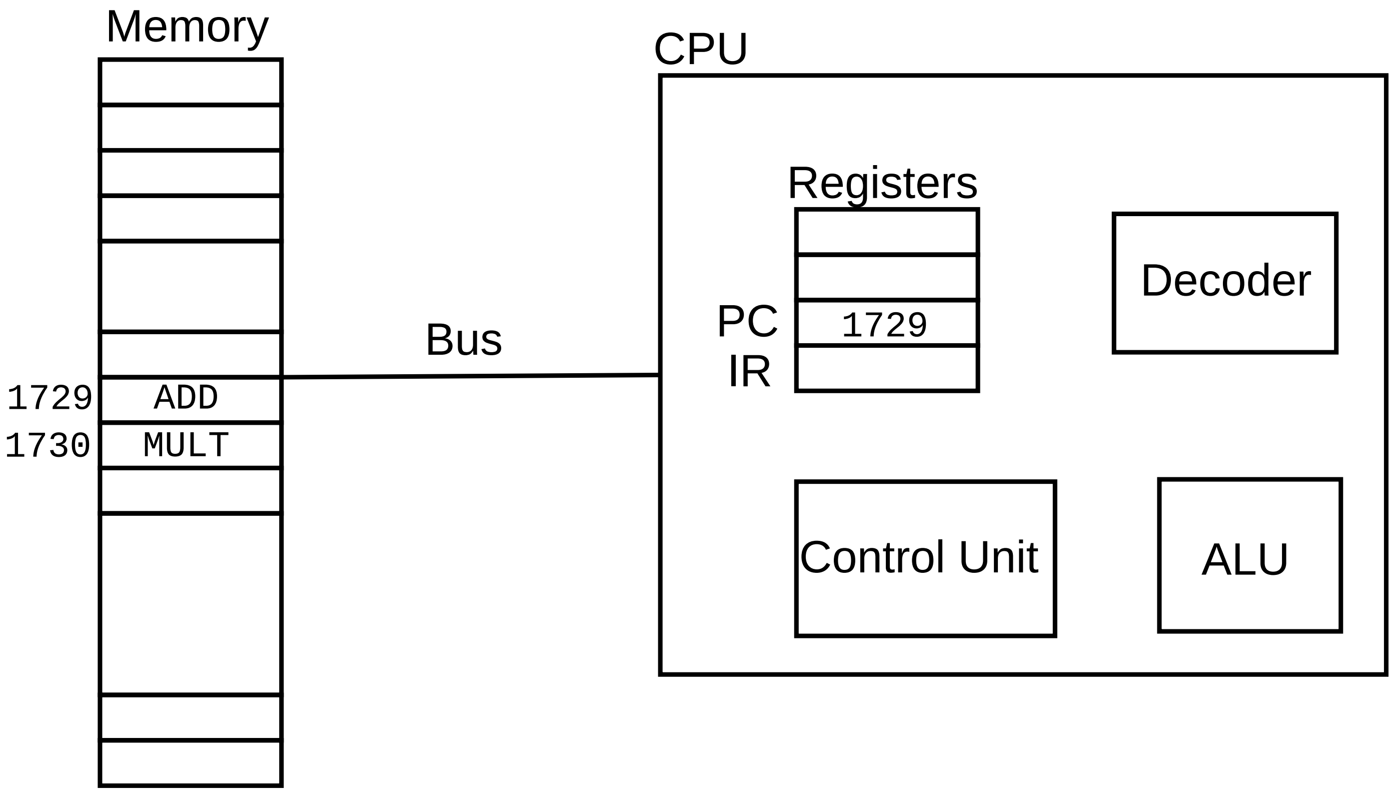
Computer Architecture

- Components:
  - CPU
  - Memory
  - IO devices
  - The Bus
- CPU:
  - Registers (PC, IR, ...)
  - ALU (Arithmetic-Logic Unit)
  - Control Unit
  - Decoder

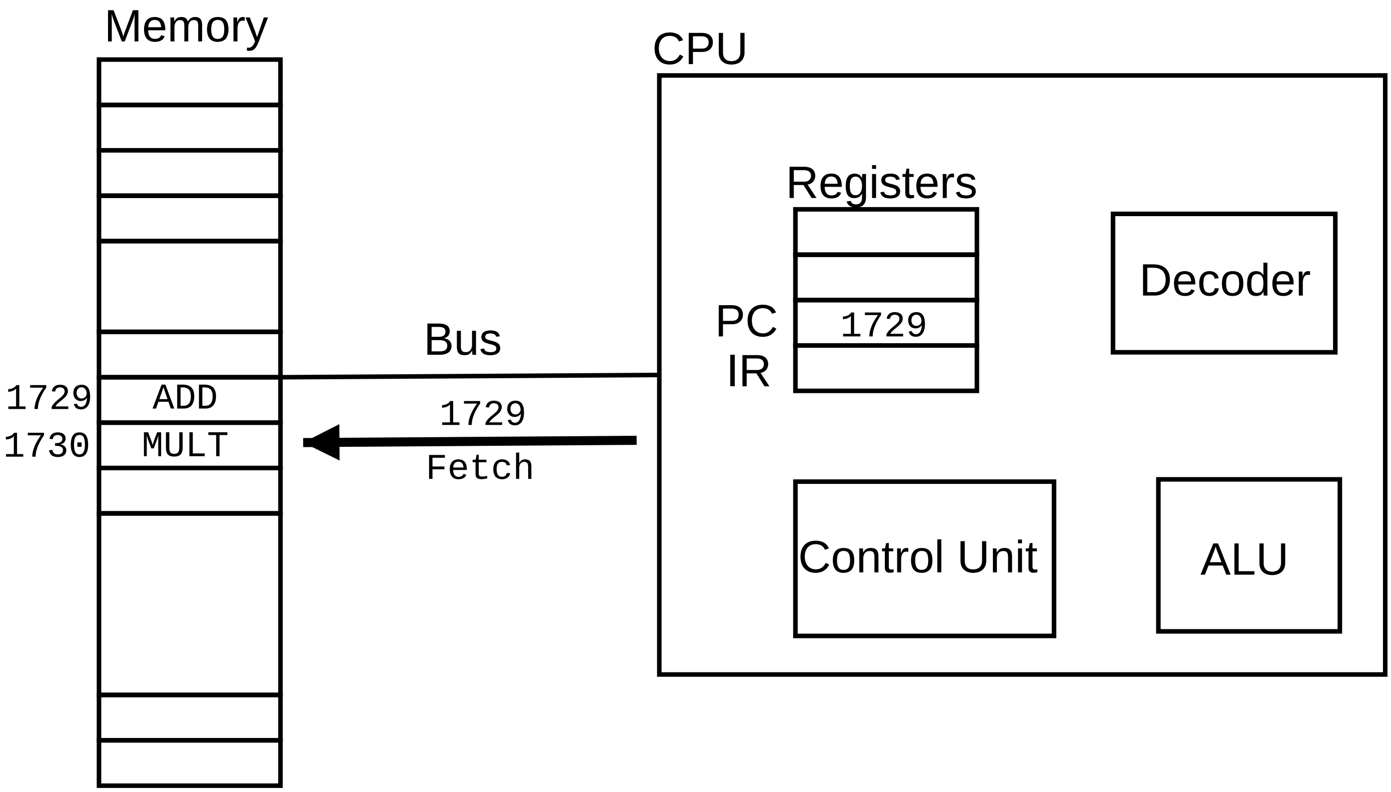
Computer Architecture

- A program is a sequence of instructions stored in memory
- Execution cycle: (Fetch-Decode-Execute)
  1. Fetch: The PC (program counter) register contains the address of the next instruction to be executed
     (a) The Control Unit sends this address to memory
     (b) Memory sends back the instruction stored in that address
     (c) The instruction is stored in the IR (instruction register)
  2. Decode: The instruction in the IR is passed to the Decoder which sends it to the appropriate circuit for execution
  3. Execute: The instruction is performed.
     (a) If the instruction is arithmetic or logic, it is executed by the ALU
  4. The PC register is updated to the next instruction
  5. Repeat

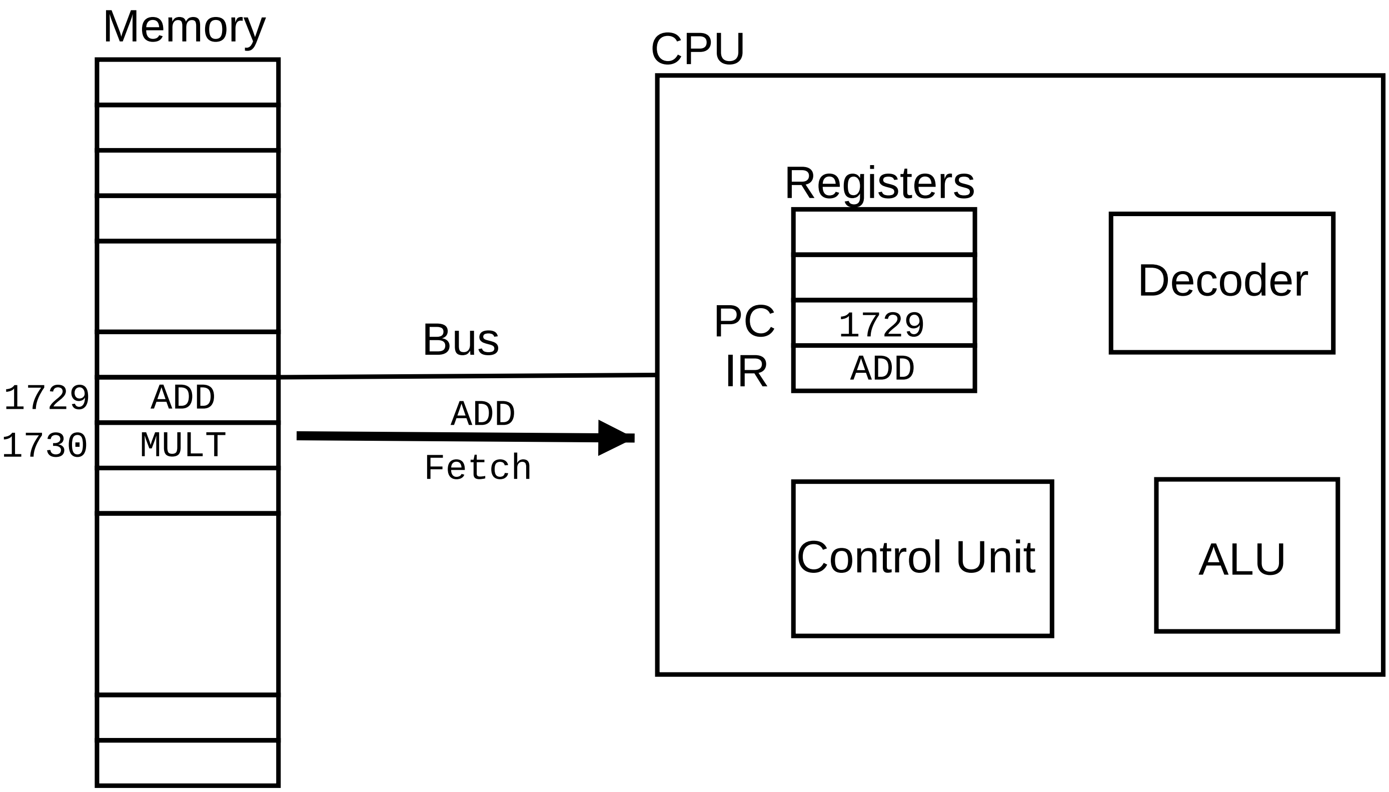
Computer Architecture



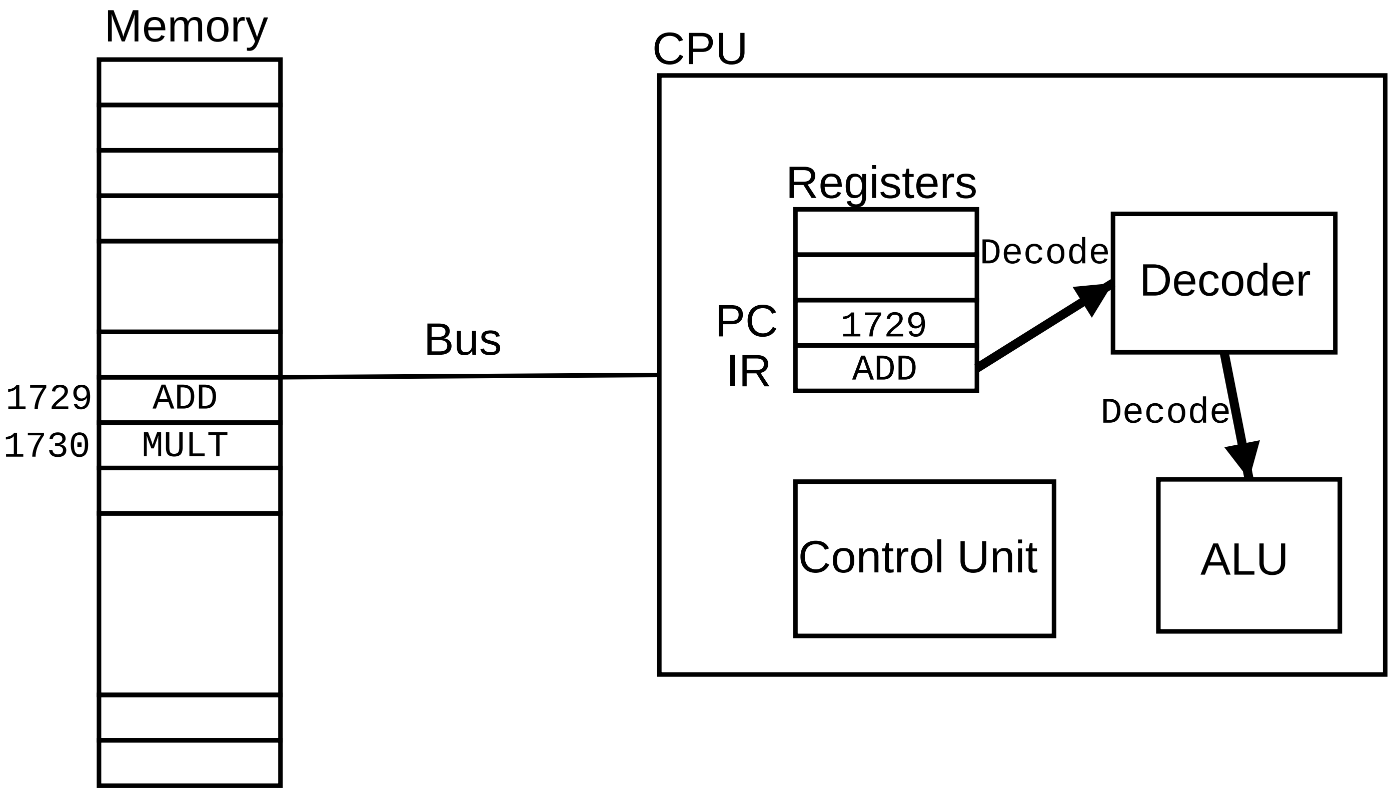
Computer Architecture



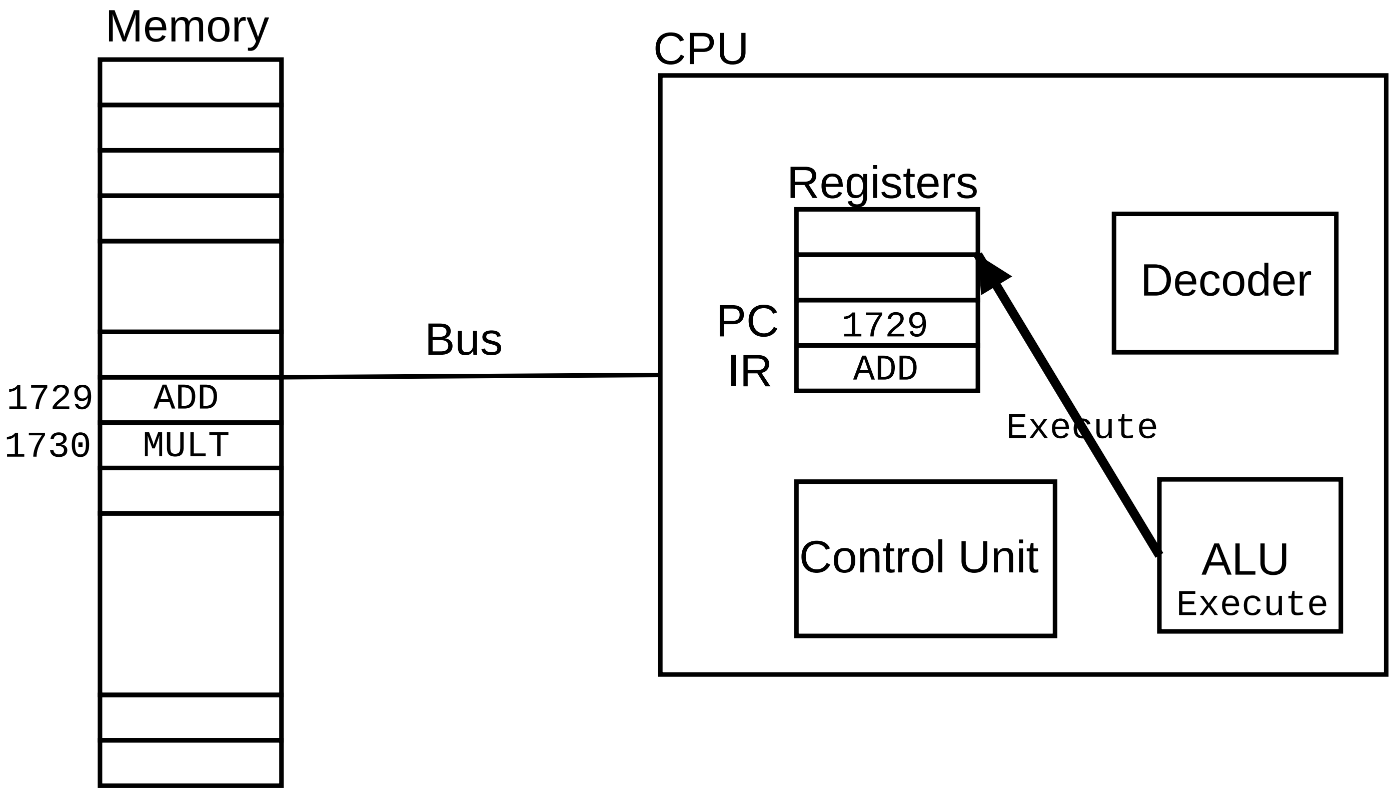
Computer Architecture



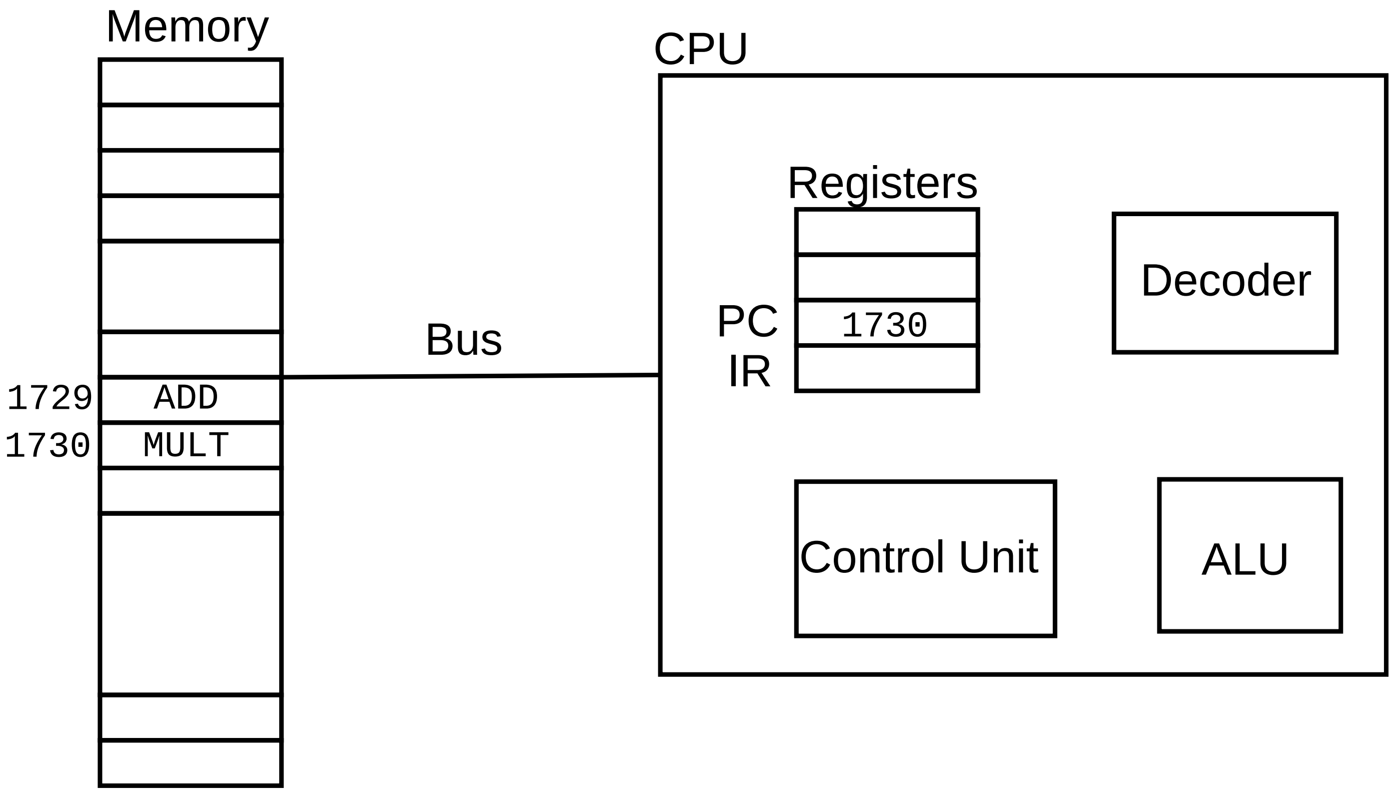
Computer Architecture



Computer Architecture



Computer Architecture



Computer Architecture

- Program instructions:
  - Instructions are numbers (ultimately in binary form)
    - 00110101 represents ADD (adding numbers)
    - 10101100 represents MULT (multiplication)
    - 01010111 represents LOAD (load data from memory to a register)
    - 10100111 represents STORE (stores data from a register to memory)

Computer Architecture

- Instructions, or operators may have parameters
  - Adding the contents of registers R1 and R2 and put the result in R3:

$$\underbrace{00110101}_{ADD}\quad\underbrace{10001001}_{R1}\quad\underbrace{10001010}_{R2}\quad\underbrace{10001011}_{R3}$$

    * Loading data from memory cell 26 and put it in register 2

$$\underbrace{11111001}_{LOAD}\quad\underbrace{00011010}_{26}\quad\underbrace{10001010}_{R2}$$

Computer Architecture

- Different kinds of processors have different *instruction sets* (e.g. Pentium, PowerPC, Alpha, SPARC, Motorola)
  - Each instruction set has different instructions, and associates different numbers to each type of instruction
  - Hence, a program for one type of processor cannot be directly executed by a different processor.
- Portability: the ability to run (execute) a program in more than one type of processor.

Programming Languages

- A program as understood by the computer is a long sequence of words (bits):

  ```
  110110001110100010010001001010010100101001001010
  ```

  – Machine Language

- But each instruction can be written in a fashion readable by humans:

  ```
  LOAD [26], R1
  LOAD 3, R2
  ADD R1, R2, R3
  STORE R3, [1700000029]
  ```

  – Assembly language

- Assembler: a program that translates an assembly language program into its machine language equivalent.

Programming Languages

- Assembly is a low-level language
- High-level languages abstract the components of the machine

  ```
  x = y + 3;
  ```

  - Java, C, C++, Python, Perl, ML, Scheme, Prolog, Ada, Pascal, Basic, Fortran, Cobol, ...
- Abstracting the components is good: when implementing an algorithm you don't have to think about the component of the computer. You focus on the problem.
- Compiler: a program that translates a high-level language program into its machine language equivalent.

A simple Java program

```
// This is a very, very, simple program

public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(‘‘Hello, World!’’);
    }
}
```

A simple java program

- Java is case-sensitive:

  ```
  HelloWorld
  ```

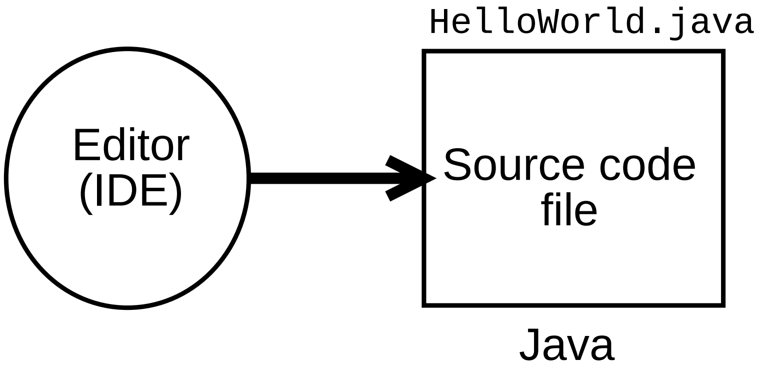
is not the same as

```
helloworld
```

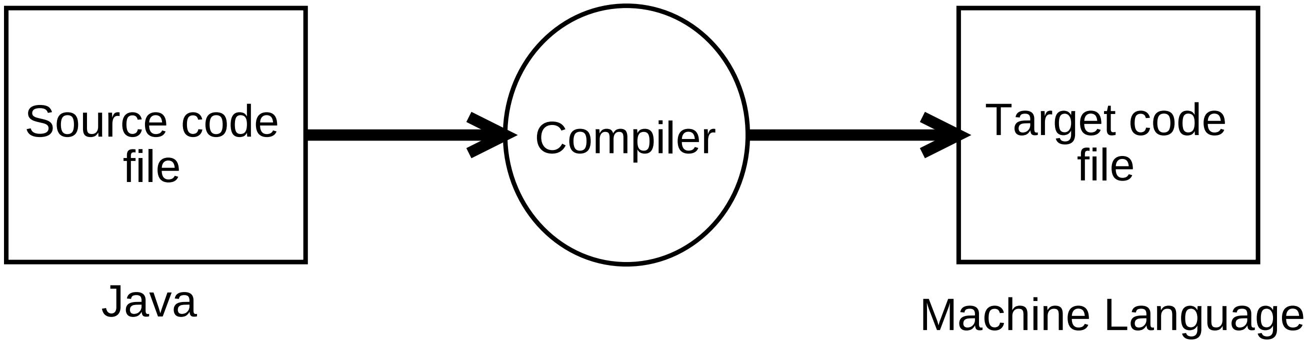
From code to a running program

- Editing
- Compilation/Interpretation
  - Compilation:
    - Translation
    - Execution
  - Interpretation:
    - Execution

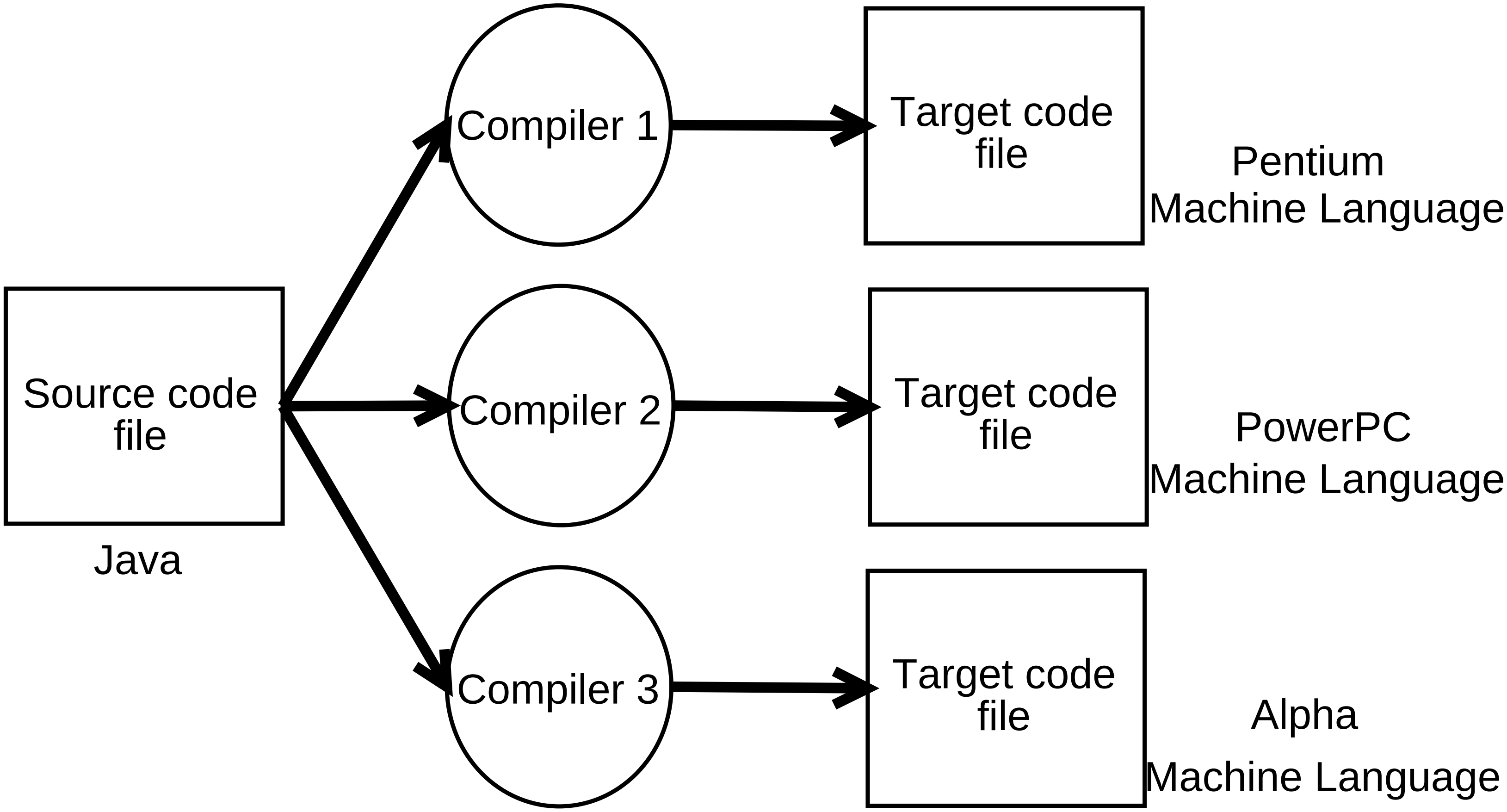
Editing



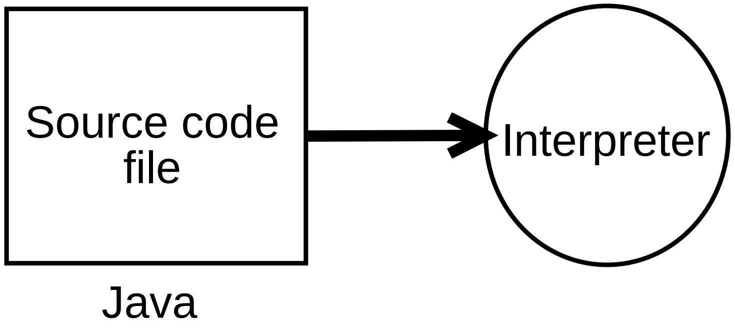
Compilers


Source code file
Compiler
Target code file
Java
Machine Language

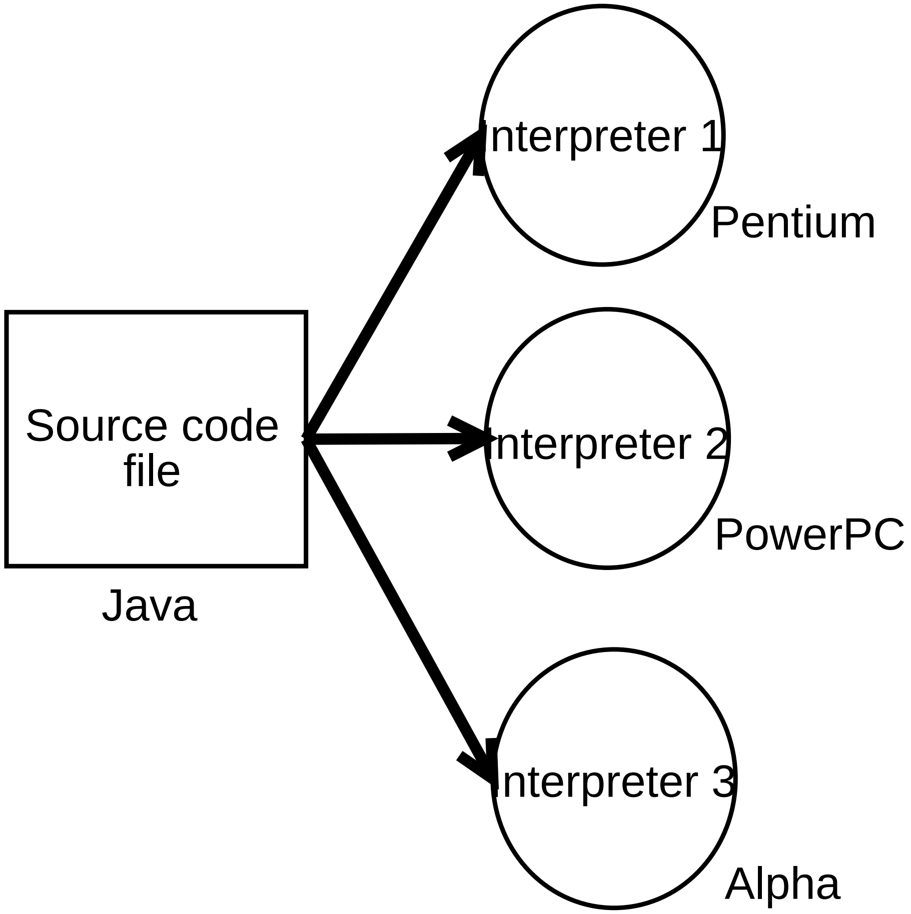
Compilers


Source code file
Java
Compiler 1
Compiler 2
Compiler 3
Target code file
Pentium
Machine Language
Target code file
PowerPC
Machine Language
Target code file
Alpha
Machine Language

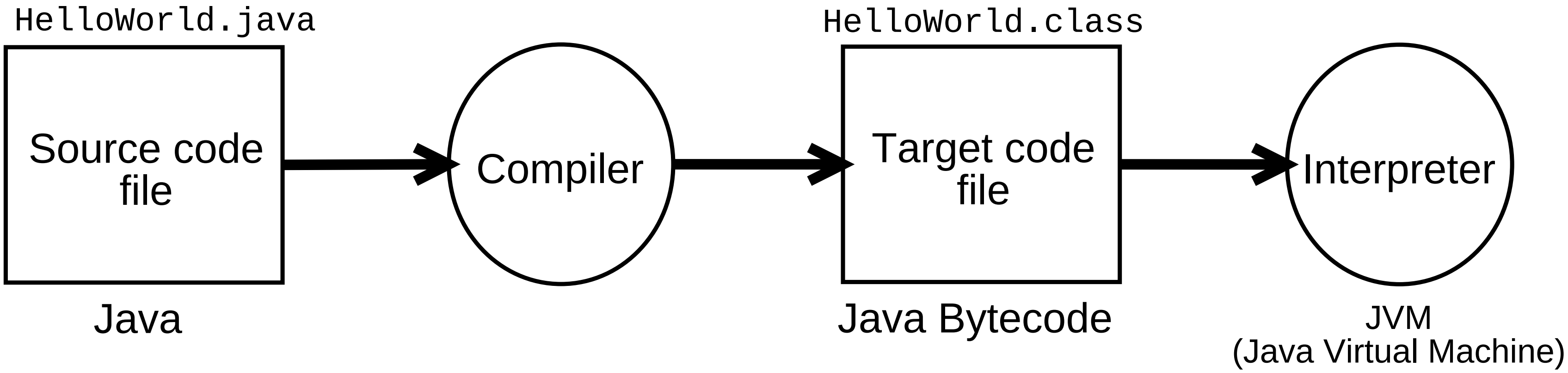
Compilers



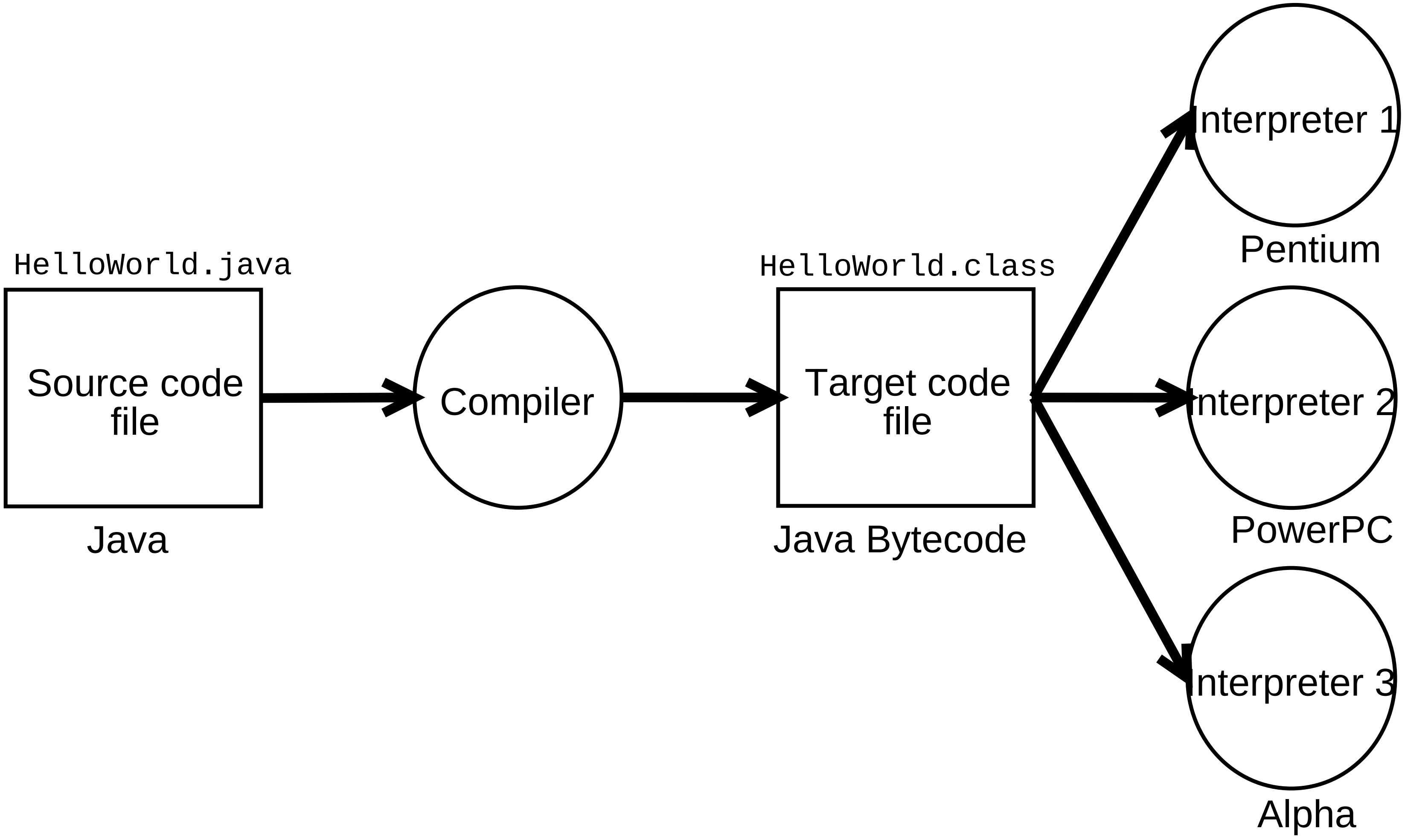
Compilers


Interpreter 1
Pentium
Source code file
Java
Interpreter 2
PowerPC
Interpreter 3
Alpha

Compilers


HelloWorld.java
Source code file
Java
Compiler
HelloWorld.class
Target code file
Java Bytecode
Interpreter
JVM
(Java Virtual Machine)

Compilers


HelloWorld.java
Source code file
Java
Compiler
HelloWorld.class
Target code file
Java Bytecode
Interpreter 1
Pentium
Interpreter 2
PowerPC
Interpreter 3
Alpha

Programming Languages

- A programming language is a formal language to describe algorithms

- A language is a means of communication

- A programming language is a means of communication between a human and a computer, but also between humans

- A programming language is formal: well-defined

Languages

- Elements of a language
  - Alphabet
  - Syntax (grammar)
  - Semantics (meaning)

- Elements of Java:
  - Alphabet of Java: ASCII
  - Syntax: 'constructs'
    - Class definitions
    - Method definitions
    - Statements
    - others
  - Semantics: computation

Errors

- Errors:
  - Compile-time errors
  - Run-time errors
    - Exceptions
    - Logical

Programming Languages

- Machine language (binary, processor dependent)
- Assembly language (textual, low-level, processor dependent)
- High-level languages (textual, abstract, processor independent)
  - There are many high-level languages: Java, C, C++, C#, ML, Haskell, Scheme, Prolog, Python, Perl, etc.
  - Different types of languages:
    - Imperative
      - Procedural
      - Object Oriented
      - Concurrent
    - Declarative:
      - Functional
      - Logic
    - Mixed

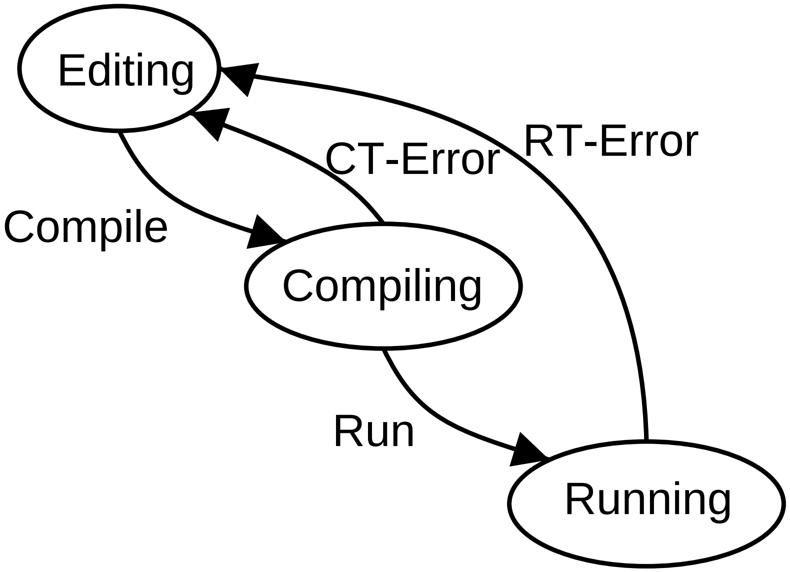
Executing programs

- Editing
- Compilation/Interpretation
  - (Native) Compilation: Translation to machine language + Execution
    - Advantages: Fast, processor specific code is generated
    - Disadvantage: Needs a compiler for each type of processor; generates a different target file for each type of processor
  - Interpretation: Direct execution
    - Advantages: Execution is processor independent. Does not generate a different target file for each possible processor (portability)
    - Disadvantage: Slow execution due to overhead of interpretation.
  - Combined: Translation to bytecode + interpretation of bytecode
    - Best of both worlds: Only one file is generated (portable) and it is faster to execute than direct interpretation (but slower than native compilation.)

Errors

- Errors:
  - Compile-time errors
  - Run-time errors
    - Exceptions
    - Logical

Errors



Basic Java Syntax

- A Java program is made up of one or more *class definitions*

- A class definition is made up of zero or more *method definitions*

- A method definition is made up of zero or more *statements* and *variable declarations*

- Roles:

  - Classes: Modules and Types of objects
  - Methods: procedures, functions, algorithms
  - Statements: instructions

Basic Java Syntax

```
public class ClassName
{
   // Body of ClassName
   // ...
   // List of method definitions
}
```

Basic Java Syntax

```
public class HelloWorld
{
   // Body of ClassName
   // ...
   // List of method definitions
}
```

Basic Java Syntax

```
public class Classname
{
    // method header
    {
        // method body: list of statements
    }
}
```

Basic Java Syntax

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(‘‘Hello’’);
        System.out.println(‘‘Good bye’’);
    }
}
```

Bad Java Syntax

```
public class HelloWorld
{
    System.out.println(“Hello”);
    System.out.println(“Good bye”);
}
```

Bad Java Syntax

```
public static void main(String[] args)
{
    System.out.println(“Hello”);
    System.out.println(“Good bye”);
}
```

Bad Java Syntax

```
public static void main(String[] args)
{
    public class HelloWorld
    {
        System.out.println(‘‘Hello’’);
        System.out.println(‘‘Good bye’’);
    }
}
```

Indentation

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(‘‘Hello’’);
        System.out.println(‘‘Good bye’’);
    }
}
```

Indentation

```
public class HelloWorld
{
public static void main(String[] args)
{
System.out.println(“Hello”);
System.out.println(“Good bye”);
}
}
```

Indentation

```
             public class HelloWorld
{
 public static void main(String[] args)
                            {
        System.out.println(‘‘Hello’’);
System.out.println(‘‘Good bye’’);
    }
               }
```

Indentation

```
public class HelloWorld{public static void main(St
ring[] args){System.out.println(‘‘Hello’’);System.ou
t.println(‘‘Good bye’’);}}
```

User Interface

- The user interface of a program is the way it interacts with the user: keyboard/mouse/windows/text

- Graphical User Interface:
  - Windows: buttons, text boxes, slidebars, graphics, etc.
  - Input with mouse and keyboard.

- Textual User Interface:
  - Console window: plain text
  - Input: keyboard only
  - Output:

    ```
    System.out.println(“text”);
    ```

Introduction to statements

- The print statement

```
System.out.println(string_literal);
System.out.print(string_literal);
```

- String literals:

```
‘‘(almost)any characters’’
‘‘This is a string literal’’
‘‘String literals can contain almost any character,
‘‘a’’
‘‘’’
‘‘24’’
```

Introduction to statements

- String concatenation:

```
string_literal + string_literal
string_literal + number_literal
```

```
“This is a ”+“message”
“This is a message”
“There are ”+70+“ students in this class”
```

- String literals with numbers are not numbers: “17” is not the same as 17

```
“17” + “29”
```

is

```
“1729”
```

while

```
17 + 29
```

is

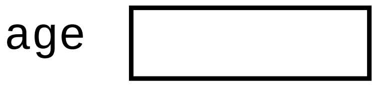
```
46
```

Simple programs

```
// File: PrintingStuff.java
public class PrintingStuff
{
    public static void main(String[] args)
    {
        System.out.println(‘‘This trivial program j
        System.out.println(‘‘prints this text to a
        System.out.println(‘‘Window.’’);
    }
}
```

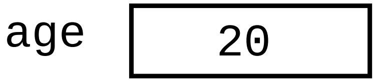
Variables

- A variable is a memory location
- A variable can contain information
- A variable has a symbolic name



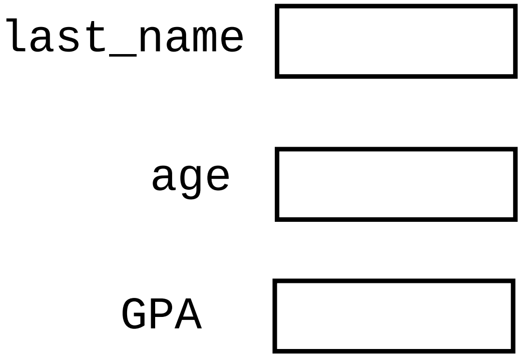
age

Variables



age 20

Variables

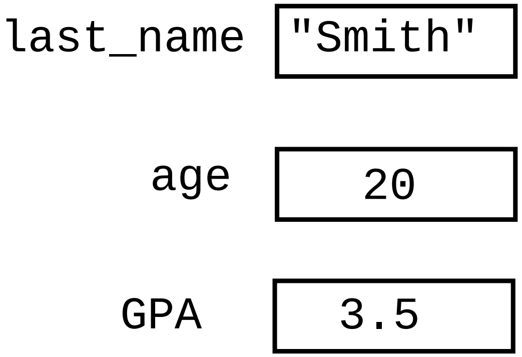


last_name

age

GPA

Variables

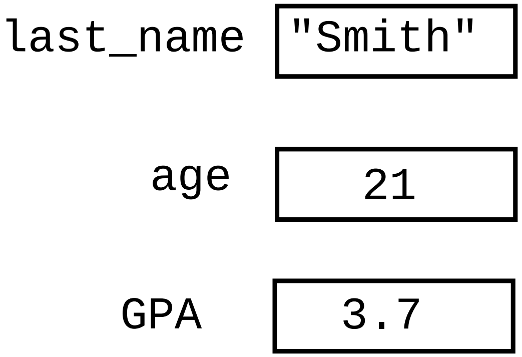


last_name "Smith"

age 20

GPA 3.5

Variables

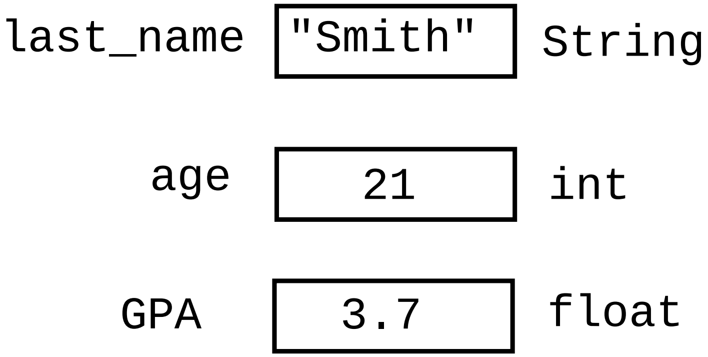


last_name "Smith"

age 21

GPA 3.7

Variables



last_name	"Smith"	String
age	21	int
GPA	3.7	float

Variable declaration

- A *variable declaration* is a statement that declares tha a variable is going to be used.

- A variable declaration goes inside some method

- A variable declaration has the form:

```
type identifier;
```

- Examples:

```
String last_name;
int age;
float GPA;
```

Assignment

- An *assignment* is a statement that gives a value to a variable
- An assignment goes inside some method
- An assignment has the form:

  ```
  variable = value;
  ```

- Its meaning it to put the value into the memory location of the variable
- Examples:

  ```
  last_name = ‘‘Smith’’;
  age = 20;
  ```

- Note that the following are *incorrect*:

  ```
  20 = age;
  ‘‘Smith’’ = last_name;
  ```

Assignment

- The variable must be declared before being assigned a value

```
String last_name;
last_name = ‘‘Smith’’;
```

- But the following is wrong:

```
age = 20;
int age;
```

- The type of the value must be the same as the type of the variable

```
last_name = 20; // Incorrect
age = ‘‘Smith’’; // Incorrect
```

Variables and String expressions

- Variables can be used with concatenation in String expressions

```
“your age is ”+age
```

- is equivalent to

```
“your age is 19”
```

- if the variable age contains the value 19

A simple program

```
public class PrintData
{
  public static void main(String[] args)
  {
    String last_name;
    int age;
    last_name = “Smith”;
    age = 20;
    System.out.println(“Your last name is ” + last_name);
    System.out.println(“You are ” + age + “ years old”);
  }
}
```

The end