AP Computer Science A Notes

Pranaav Sureshkumar

September 2023

Contents

Unit 1

1.1 Binary & Hexadecimal

You are likely most familiar with the base 10 (decimal) number system in which each digit represents a power of 10. For example, in the decimal number

$$2560_{10}$$[1]

the numeral 0 represents how many ones (10^0) are in the number, the numeral 6 represents how many tens (10^1) are in the number, the numeral 5 represents how many hundreds (10^2) are in the number, etc.

Similarly, in binary, each digit represents a power of 2. For example, in the binary number

$$1011_2$$

the rightmost numeral one represents how many ones (2^0) are in the number, the numeral one directly to the left represents how many twos (2^1) are in the number, the numeral zero represents how many fours (2^2) are in the number, etc.

$$\text{Therefore, } 1011_2 = 1 \cdot 8_{10} + 0 \cdot 4_{10} + 1 \cdot 2_{10} + 1 \cdot 1_{10} = 11_{10}$$

The following page features a binary-decimal worksheet to test your understanding.

[1]The subscript represents the base of the number.

Practice

Binary → Decimal

Convert each 4-digit binary number to decimal.

1. $1101_2 = \underline{\hspace{2em}}_{10}$
2. $1010_2 = \underline{\hspace{2em}}_{10}$
3. $1111_2 = \underline{\hspace{2em}}_{10}$
4. $0100_2 = \underline{\hspace{2em}}_{10}$
5. $0011_2 = \underline{\hspace{2em}}_{10}$
6. $1000_2 = \underline{\hspace{2em}}_{10}$
7. $0110_2 = \underline{\hspace{2em}}_{10}$
8. $1100_2 = \underline{\hspace{2em}}_{10}$

Decimal → Binary

Convert each decimal number to 4-digit binary.

9. $9_{10} = \underline{\hspace{2em}}_{2}$
10. $15_{10} = \underline{\hspace{2em}}_{2}$
11. $6_{10} = \underline{\hspace{2em}}_{2}$
12. $10_{10} = \underline{\hspace{2em}}_{2}$
13. $3_{10} = \underline{\hspace{2em}}_{2}$
14. $12_{10} = \underline{\hspace{2em}}_{2}$
15. $14_{10} = \underline{\hspace{2em}}_{2}$
16. $2_{10} = \underline{\hspace{2em}}_{2}$

Answer Key

Binary → Decimal

Convert each 4-digit binary number to decimal.

1. $1101_2 = 13_{10}$
2. $1010_2 = 10_{10}$
3. $1111_2 = 15_{10}$
4. $0100_2 = 4_{10}$
5. $0011_2 = 3_{10}$
6. $1000_2 = 8_{10}$
7. $0110_2 = 6_{10}$
8. $1100_2 = 12_{10}$

Decimal → Binary

Convert each decimal number to 4-digit binary.

9. $9_{10} = 1001_2$
10. $15_{10} = 1111_2$
11. $6_{10} = 0110_2$
12. $10_{10} = 1010_2$
13. $3_{10} = 0011_2$
14. $12_{10} = 1100_2$
15. $14_{10} = 1110_2$
16. $2_{10} = 0010_2$

Note: leading zeros are suggested but optional; trailing and sandwiched zeros are required.

Hexadecimal is a base 16 number system. It uses the numerals 0–9 to represent their base 10 equivalents, but also uses the numerals A, B, C, D, E, and F to represent numbers 10–15 respectively. RGB colors are often represented as hexadecimals.

1.2 First Programs

Java is a high-level[2] programming language that allows programmers to execute *algorithms*, or series of tasks. Java is the programming language that will be taught throughout this course. In this subsection, we will familiarize ourselves with the coding environment and learn about *comments* and two basic functions: `print()` and `println()`.

1.2.1 Printing

Upon opening a Java repl, you will be greeted with a file explorer, a text editor, and a console. The text editor is where we will enter Java code to get our computer to execute our algorithms, and the console is where we will see our outputs. Your text-editor should be auto-populated with the following code:

```
1 class Main {
2   public static void main(String[] args) {
3     System.out.println("Hello world!");
4   }
5 }
```

We can ignore most of this code for now as we will learn about it later—just know that our code goes between the second set of curly braces. Try running this code. You can do so either by clicking the green "Run" button or by pressing Ctrl Enter. This basic program uses one of our functions of interest (`println()`) on line 3 to output "Hello world!" without quotes to the console when run. Now, try replacing the `println()` with `print()` and running this code. The exact same output is printed. So what's the difference between `println()` and `print()`?

Let's add another `println()` to our code and see what it does. Add a `System.out.println()` statement directly below the existing one and enter any string of text you want to print in quotes between the parentheses. Your finished code should look something like this:

[2]Low-level programming languages are machine-oriented and only a few levels away from machine code. High-level languages are more user-oriented and straightforward, being more distant from machine code.

```java
class Main {
  public static void main(String[] args) {
    System.out.println("Hello world!");
    System.out.println("I love Java!");
  }
}
```

Let's run this code and see what happens. If everything went well, your console should display

```
> sh -c javac -classpath .:target/dependency/* -d . $(find . -type f -name '*.java')
> java -classpath .:target/dependency/* Main
Hello world!
I love Java!
>
```

or whichever string of text you chose to print. (Disregard the first two lines, as those are referring to your Java installation. For the remainder of these notes, extraneous lines will be omitted when displaying outputs.) As expected, "I love Java" was printed to the console. Now, let's try switching the first `println()` to `print()`. Run the code again. What happens? Your output should be

```
Hello world!I love Java!
```

Now, the difference between `println()` and `print()` is clear. `println()` prints the input and appends a new line (essentially pressing Enter after printing) whereas `print()` only prints the input.

1.2.2 Comments

When programming, it is important to ensure your code is readable and easily understandable as it makes fixing problems easier and enables peer review. We can use *comments* to help us acheive this. A comment is something we can insert in our code to "annotate" it. Comments are completely ignored by the computer and only exist to help humans understand their code.

In Java, there are two types of comment: single-line comments and multi-line comments. You can make a single-line comment by typing `//`. When you do so, anything after the double slash and on the same line will be ignored. You can make a multi-line comment by typing `/* */`. In a multi-line comment, anything between the two asterisks will be ignored. Below is an example of both types of comments being used on the code block discussed in 1.2.1:

```java
/* This is a multi-line comment.
Anything typed between the two asterisks will be ignored.
Multi-line comments are used to "annotate" code. */

class Main {
  public static void main(String[] args) {
    System.out.print("Hello world!"); // This is a single line comment.
    System.out.println("I love Java!"); // prints "I love Java!"
  }
}
```

Use comments frequently in your code even if you're the only one who will see them. Comments help tremendously with debugging and allow you follow your algorithm step-by-step without reading code.

1.3 Variables

In Java, there are four distinct types of data: integers, doubles, booleans, and strings. Integers are whole numbers, doubles are decimal values, booleans are either true or false, and strings are a list of characters (like words). We can use *variables* to store data. When we do so, we must first tell Java what type of data we would like to store in the variable. Then, we can tell Java what exactly we want the variable to store. This is shown below.

```java
1 class Main {
2   public static void main(String[] args) {
3     int x;
4     x = 2;
5   }
6 }
```

On line 3, the code above declares a variable `x` and tells Java that `x` will store an integer. On line 4, the code assigns the value `2` to `x`. Anytime Java sees `x` from now on, it will interpret it as `2`. We can confirm that `x` really is `2` by telling Java to print `x`.

```java
class Main {
  public static void main(String[] args) {
    int x; // declares an integer variable x
    x = 2; // assigns x the value 2
    System.out.println(x); // prints x
  }
}
```

(Notice that when we print `x`, we do not put `x` in parentheses. In Java, anything entered in parentheses will be printed verbatim, not interpreted.)

When run, this outputs 2, confirming that `x == 2`.[3] We can make our code more concise by assigning it a type and a value simultaneously. To do so, we can combine lines 3 and 4 in the previous code as such:

```java
class Main {
  public static void main(String[] args) {
    int x = 2; // declares an integer variable x that stores 2.
    System.out.println(x); // prints x
  }
}
```

When run, this code will also output 2 to the console.

We can do the same thing with doubles. Let's create a double `Pi` and assign it the value `3.14`.

```java
class Main {
  public static void main(String[] args) {
    double Pi = 3.14; // declares a double Pi that stores 3.14
    System.out.println(Pi); // prints Pi
  }
}
```

As expected, 3.14 is output to the console. What happens if we declare an integer variable but assign it a double? Let's find out by replacing the `double` in the previous code with `int`. When we run the code, we get the output:

```
error: incompatible types: possible lossy conversion from double to int
    int Pi = 3.14;
             ^
1 error
exit status 1
```

Uh oh! Our code did not run because it encountered an error. When this happens, it is important to read the error message as it often contains useful hints as to what went wrong. This error message is telling us that we used incompatible data types in our code as we tried to assign a `double` to an `int`. But what happens if we assign an integer to a double? In theory, no information is lost when storing an integer as a decimal, but does this hold up in practice? Let's check by running the following code that assigns an `int` (5) to a `double`.

```java
class Main {
  public static void main(String[] args) {
    double num = 5; // declares a double num that stores 5
    System.out.println(num); // prints num
  }
}
```

[3]When assigning variables in Java, a single equal sign (=) is used. When comparing two values in Java, a double equal sign (==) is used. For example, `x == 2` will be evaluated as `true` if `x` is 2 and `false` if `x` is not 2.

We get the output `5.0`. Java converts the integer to a decimal but does not throw an error because no data is lost.

A special property of the `int` is that it always returns an `int` when operated on. For example, try printing `1 / 2` with the following code:

```java
class Main {
  public static void main(String[] args) {
    System.out.println(1 / 2);
  }
}
```

Curiously, `0` is output to the console instead of `0.5`. Why is this? Let's try `5 / 2` instead. This time, `2` is output to the console, ignoring the remainder. In Java, `int` operations *always* return `int` values. So how can we get Java to correctly divide 5 by 2? An easy way to do this is to convert an `int` to a `double` as such:

```java
class Main {
  public static void main(String[] args) {
    System.out.println(1.0 / 2);
    // or
    System.out.println(1 / 2.0);
    // or
    System.out.println(1.0 / 2.0);
  }
}
```

While this property of Java may seem cumbersome to circumnavigate at first, it can be leveraged to our advantage as we will later see.

Now, let's familiarize ourselves with the `String` and `boolean` data types. Let's define and print a `String` variable and a `boolean` variable with the following code:

```java
class Main {
  public static void main(String[] args) {
    String bestActor = "Danny DeVito";
    boolean validity = true;
    System.out.println(bestActor);
    System.out.println(validity);
  }
}
```

When run, this code outputs:

```
Danny DeVito
true
```

Once a variable is defined, you can change its value but not its type. For example, an `int` variable could store `2` then `5` then `1000`, but it can *never* store `0.5` or `"Hello world!"` or `true`. In Java, attempting to change a variable's type will throw an error.

1.4 Expressions & Assignment

Computers are useful because they can operate on data quickly and efficiently. Now, we will learn how to manipulate various data types. In this subsection, it is important to remember that operations on data types will always return the same data type. There are various operators we can apply to different data types. They are listed below[4].

- `int`/`double`: addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), modulus[5] (`%`), increment (`++`), and decrement (`--`)
- `String`: concatenate (`+`)

Let's use our newly acquired knowledge of Java operators to make a program that computes the circumference of a circle given its radius. We will need to use variable assignments to define the radius and π, we will need to use numeric operators to compute the circumference, and we will need to concatenate strings to present the output in an understandable form. This is shown below.

```java
class Main {
  public static void main(String[] args) {
    int radius = 5; // defines the radius of the circle
    double Pi = 3.1415; // defines Pi, a necessary constant
    double circumference = 2*Pi*radius; // computes the
    ↪  circumference of the circle
    System.out.println("The circumference of a circle with radius
    ↪  " + radius + " is " + circumference + "."); // prints the
    ↪  circumference.
  }
}
```

When run, the program correctly computes the circumference of a circle with radius 5 and uses string concatenation to print:

```
The circumference of a circle with radius 5 is 31.415000000000003.
```

[4]Boolean operators will be taught in detail later

[5]In Java, performing the modulus operation on two doubles will return the remainder. In most languages, modulus is exclusive to the `int` data type.