# CSC 172– Data Structures and Algorithms

Lecture 2

Spring 2018

TuTh 3:25 pm – 4:40 pm

# Agenda

- Administrative aspects

- Number Systems
- Memory Management
- OneNote Discussion (if required)
- Java Generics

Chapter 1

# ADMINISTRATIVE ASPECTS

# Workshops

- Workshops
  - Workshops begin on this Sunday (January 28th)
  - Remember: Workshops run on Sun, Mon, Tues, Wed
  - One Session every week

# Labs

- Labs
  - Already started
    - Runs on Mon, Tues, Wed, Thur
    - Two Sessions every week.
  - Lab 1
    - If you have already given demo of your code, you are all set.
  - Lab 2
    - Will be released on Sunday (Jan 28)
    - Due on Sunday (Feb 04) 11:59 pm (Submit on Blackboard)
  - Continues…..

# Lab Schedule (may change)

| Date | Day | Events | Release Lab # | Due Lab # | Graded Lab # | No further dispute accepted for Lab # |
|---|---|---|---|---|---|---|
| 1/21/2018 | Sun | | 1 | | | |
| 1/28/2018 | Sun | | 2 | 1 | | |
| 2/4/2018 | Sun | | 3 | 2 | 1 | |
| 2/11/2018 | Sun | | 4 | 3 | 2 | 1 |
| 2/18/2018 | Sun | | 5 | 4 | 3 | 2 |
| 2/25/2018 | Sun | | | | 4 | 3 |
| 3/4/2018 | Sun | | 6 | 5 | | 4 |
| 3/11/2018 | Sun | Recess | | | 5 | |
| 3/18/2018 | Sun | | 7 | 6 | | 5 |
| 3/25/2018 | Sun | | 8 | 7 | 6 | |
| 4/1/2018 | Sun | | 9 | 8 | 7 | 6 |
| 4/8/2018 | Sun | | 10 | 9 | 8 | 7 |
| 4/15/2018 | Sun | | 11 | 10 | 9 | 8 |
| 4/22/2018 | Sun | | | 11 | 10 | 9 |
| 4/29/2018 | Sun | | | | 11 | 10 |
| | | | | | | 11 |

# Course policy

- Applicable to quizzes, projects, labs and exams:

You have one week after grading to dispute your grade.

# Quizzes

- Quizzes
  - In-class:
    - 10 to 20 minutes
    - Anyone with special needs, contact and schedule with Disability Resources

Quiz 1:

1. Coding on Paper
   --- Java Basics (Arrays, Strings)
   --- All of you should get full points
   --- Otherwise, it's probably time to review CSC 171 materials
2. Class ID
3. Questions on Academic Honesty

# ClassID, Workshop and Lab Schedule

- TA office hours:

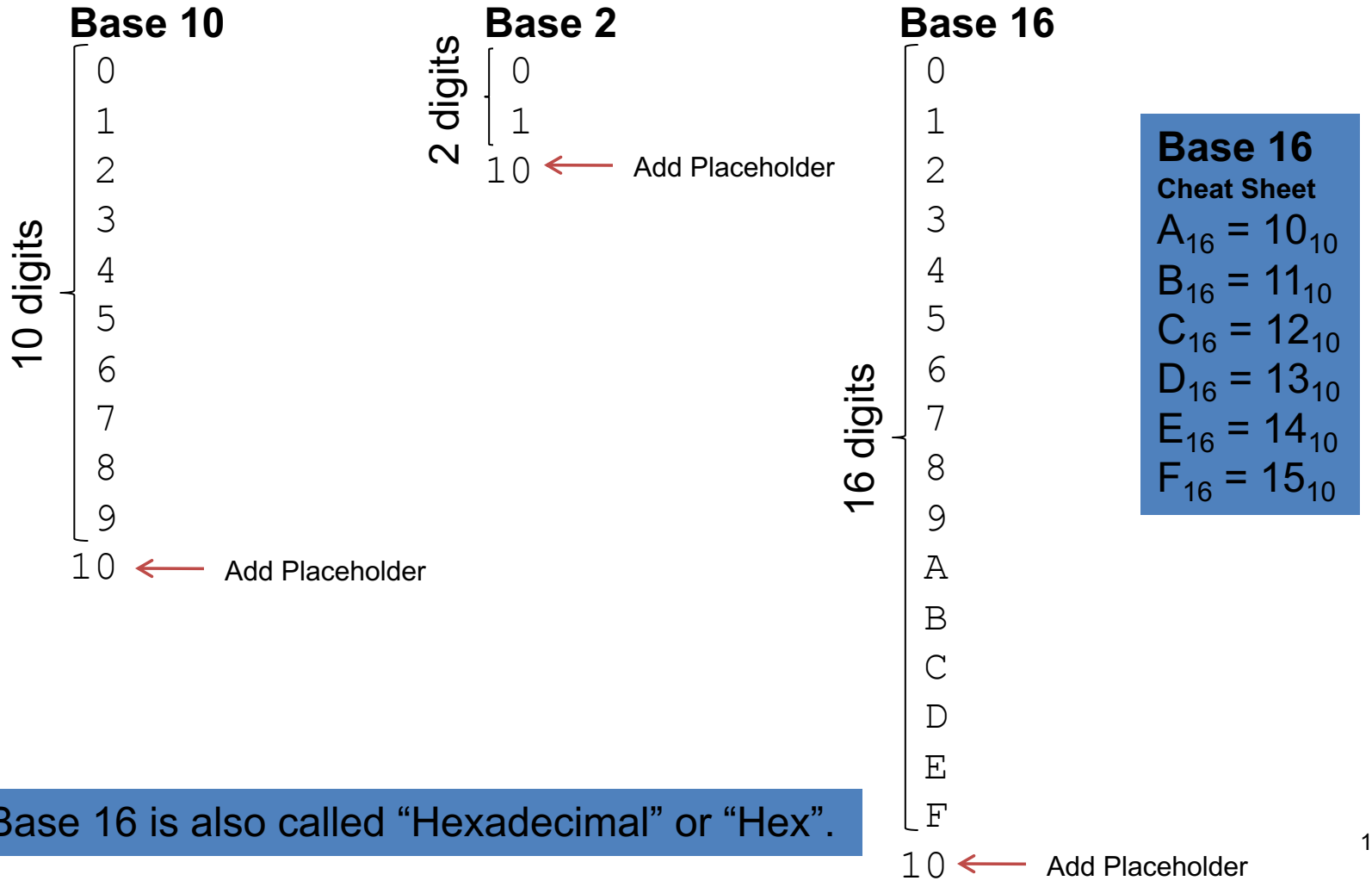
Grad TA: Divya Ojha
Office: Wegmans 3209
Office Hours: Tu 5pm - 6pm. Th 2:15 pm - 3:15 pm
Email: dojha@cs.rochester.edu

- ClassID, WSL and Lab TA info:
  - https://docs.google.com/spreadsheets/d/1bkWjhry2VNJaDC6idE7QbQTvebdf5cq9HkyPsQUkuIs/edit?usp=sharing

# NUMBER SYSTEMS

there are 10 types of people in this world, those who understand binary and those who don’t

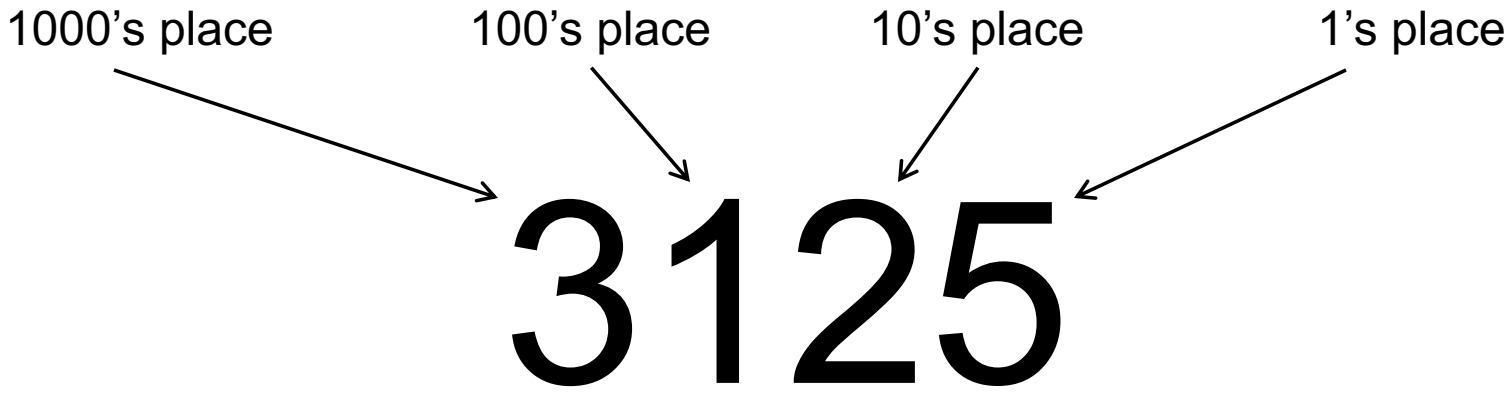
# Which Digits Are Available in which Bases




Note: Base 16 is also called “Hexadecimal” or “Hex”.

# Review of Placeholders

You probably learned about placeholders in the 2nd or 3rd grade. For example:




So this number represents

- 3 thousands
- 1 hundred
- 2 tens
- 5 ones

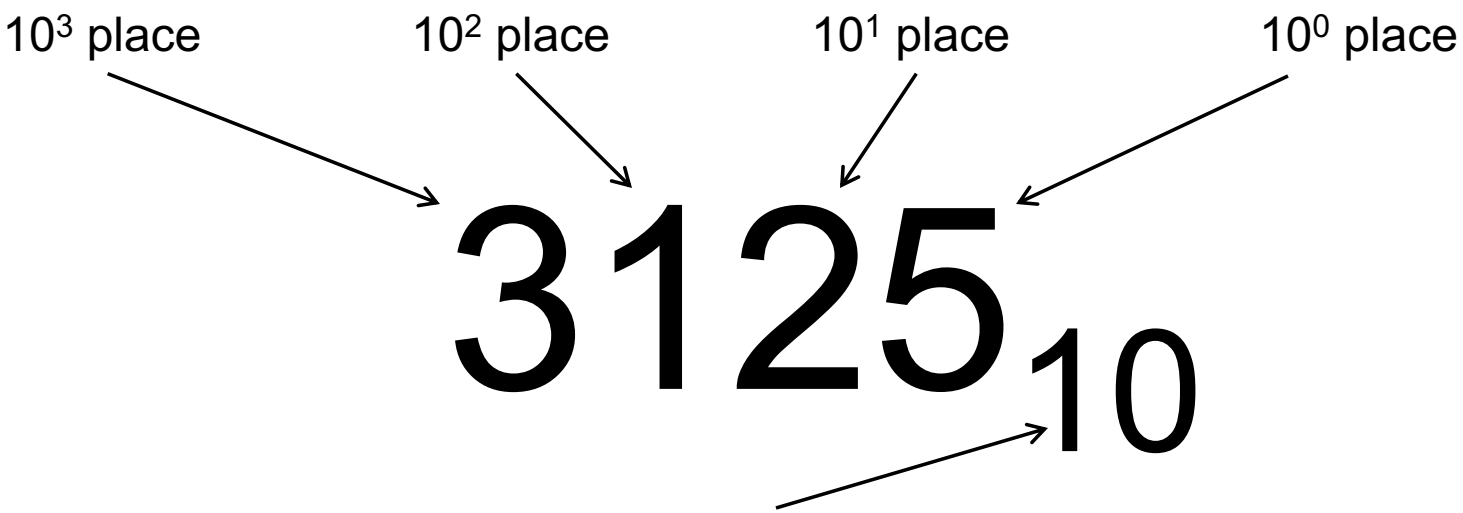
Mathematically, this is

$$(3 \times 1000) + (1 \times 100) + (2 \times 10) + (5 \times 1) = 3000 + 100 + 20 + 5 = 3125$$

But why are the placeholders 1, 10, 100, 1000, and so on?

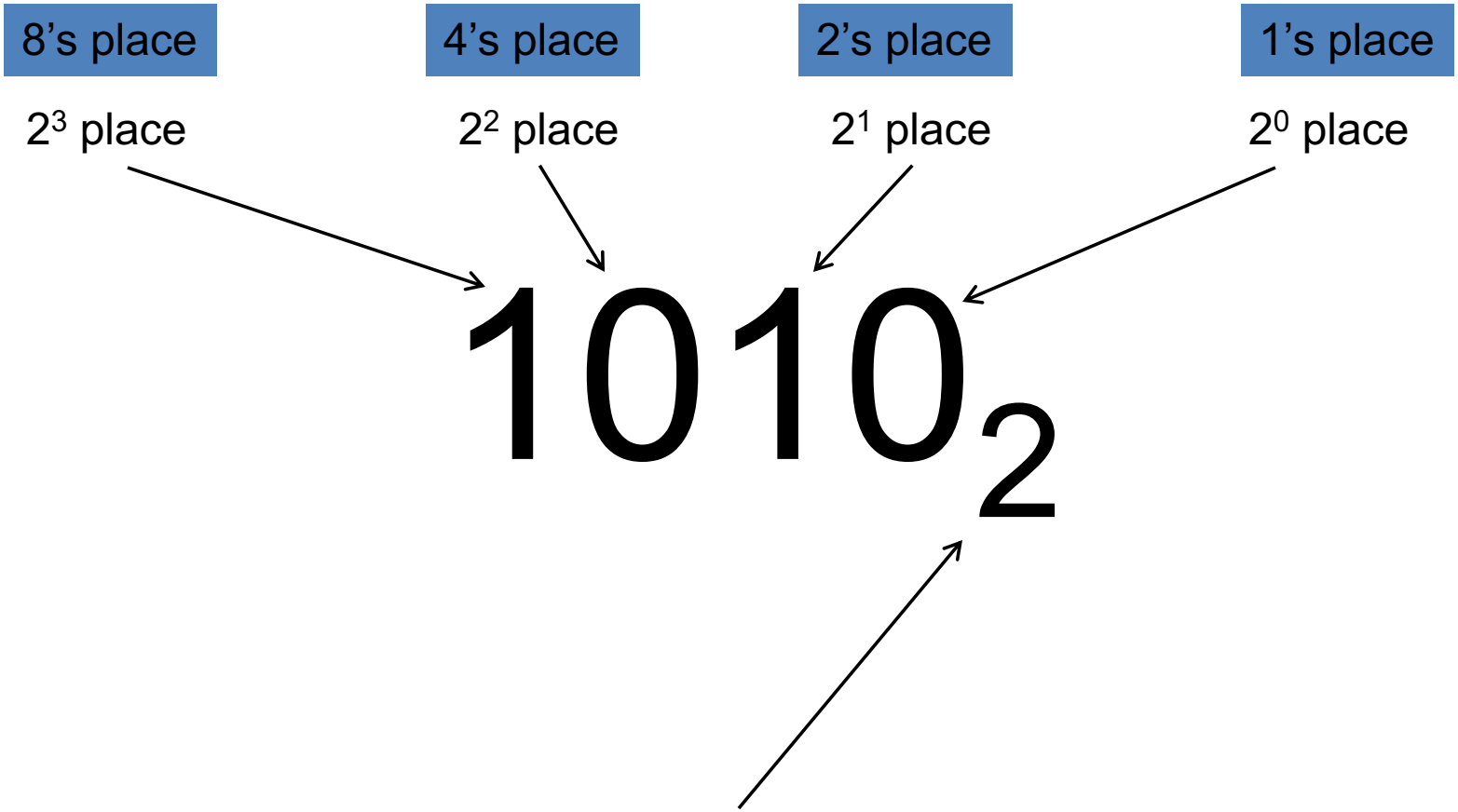
# More on Placeholders

- The numbers commonly used by most people are in Base 10.
- The Base of a number determines the values of its placeholders.

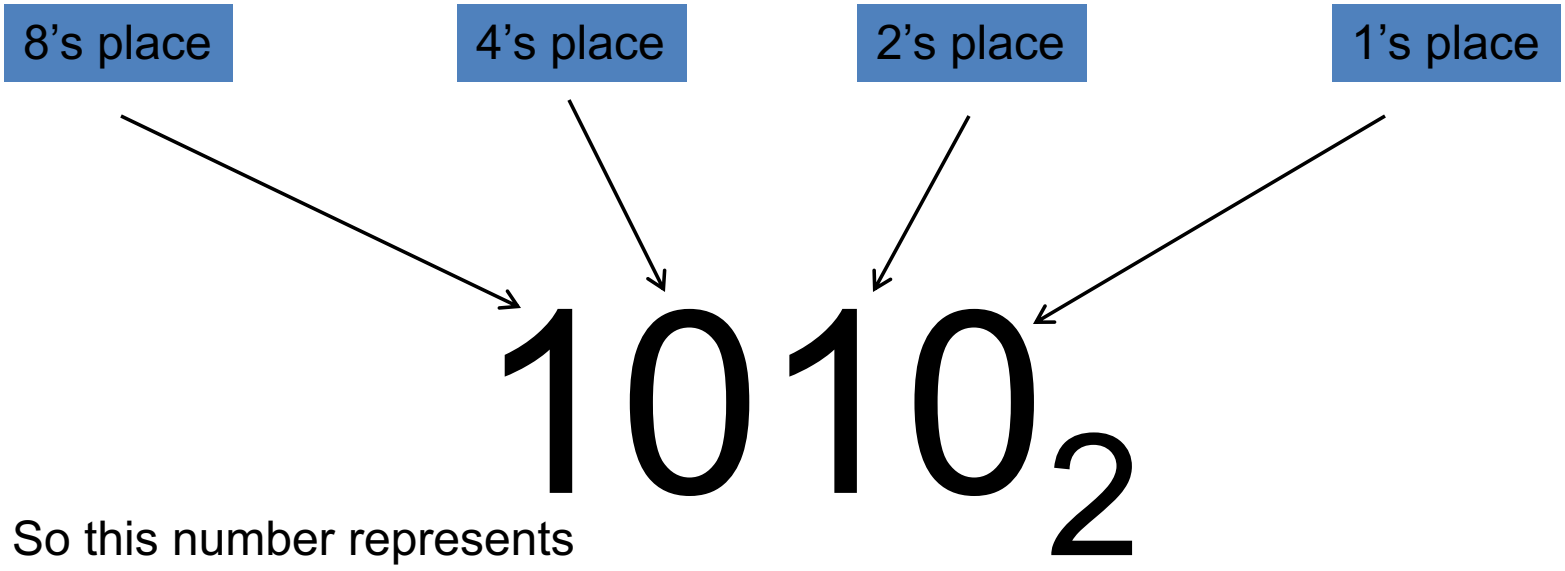



To avoid ambiguity, we often write the base of a number as a subscript.

# Binary Numbers - Example



# Binary Numbers - Example



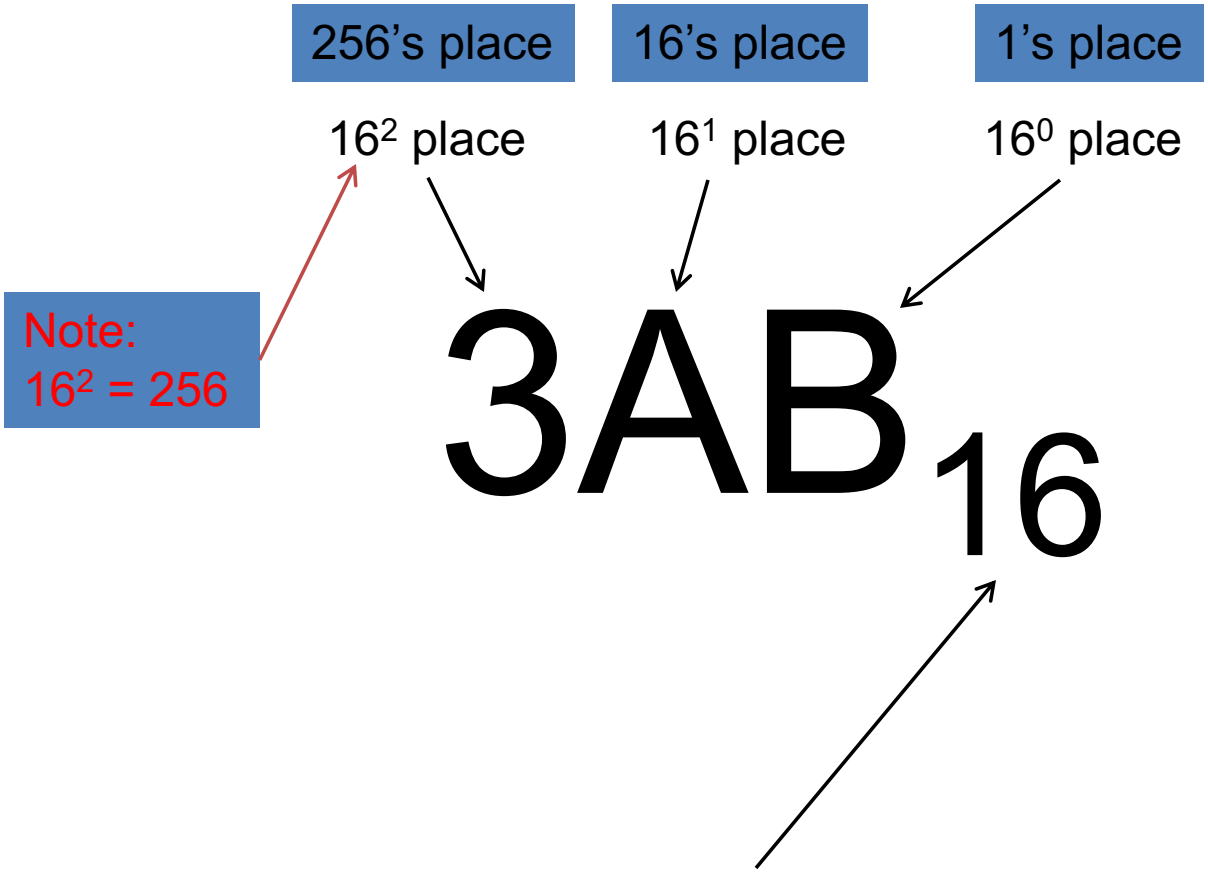

So this number represents

- 1 eight
- 0 fours
- 1 two
- 0 ones

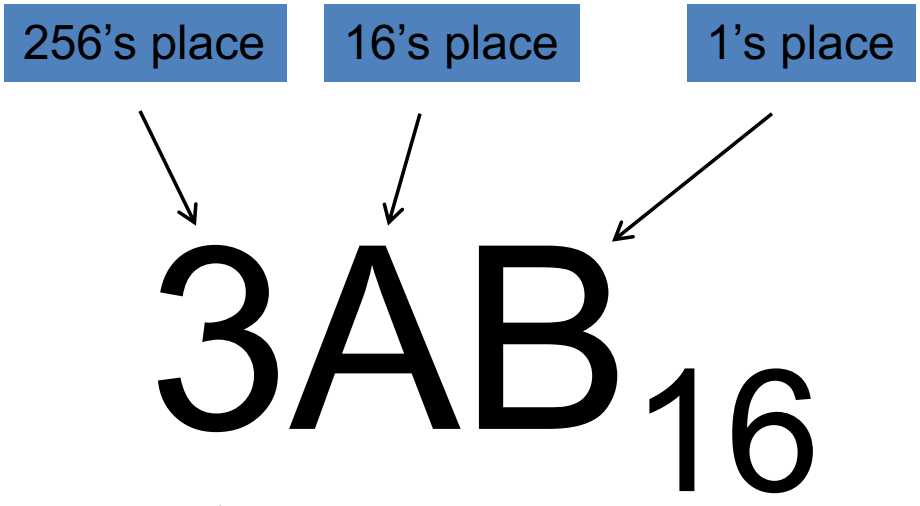
Mathematically, this is

$(1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)$
$= 8 + 0 + 2 + 0 = 10_{10}$

# Hexadecimal Numbers - Example



# Hexadecimal Numbers - Example




**Base 16**
**Cheat Sheet**

$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

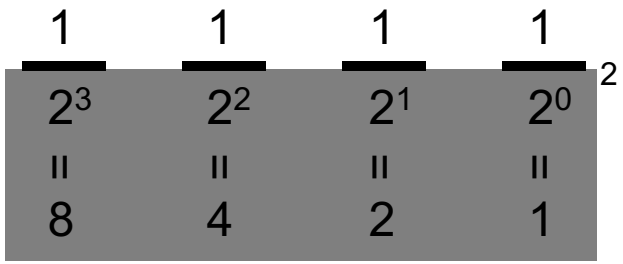
So this number represents
- 3 two-hundred fifty-sixes
- 10 sixteens
- 11 ones

Mathematically, this is

$(3 \times 256) + (10 \times 16) + (11 \times 1)$
$= 768 + 160 + 11 = 939_{10}$

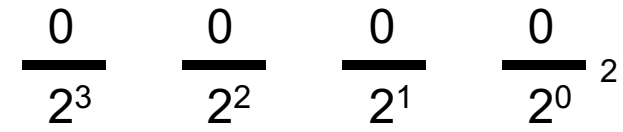
# Why Hexadecimal Is Important

What is the largest number you can represent using four binary digits?




$8 + 4 + 2 + 1 = 15_{10}$

… the smallest number?




$0 + 0 + 0 + 0 = 0_{10}$

What is the largest number you can represent using a single hexadecimal digit?

$F_{16} = 15_{10}$

… the smallest number?

$0_{16} = 0_{10}$

**Base 16**
**Cheat Sheet**

$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

Note: You can represent the same range of values with a single hexadecimal digit that you can represent using four binary digits!

# Why Hexadecimal Is Important

## Continued

It can take a lot of digits to represent numbers in binary.

Example:
$51794_{10} = 1100101001010010_2$

Long strings of digits can be difficult to work with or look at.

Also, being only 1's and 0's, it becomes easy to insert or delete a digit when copying by hand.

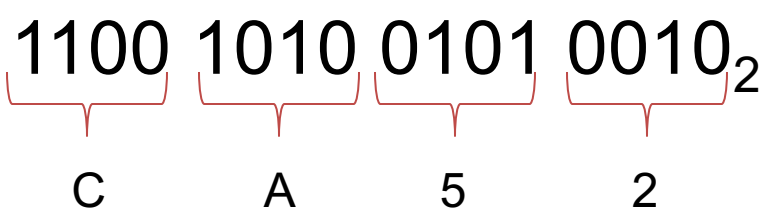
Hexadecimal numbers can be used to abbreviate binary numbers.

Starting at the least significant digit, split your binary number into groups of four digits.

Convert each group of four binary digits to a single hex digit.

# Converting Binary Numbers to Hex

Recall the example binary number from the previous slide:
$1100101001010010_2$

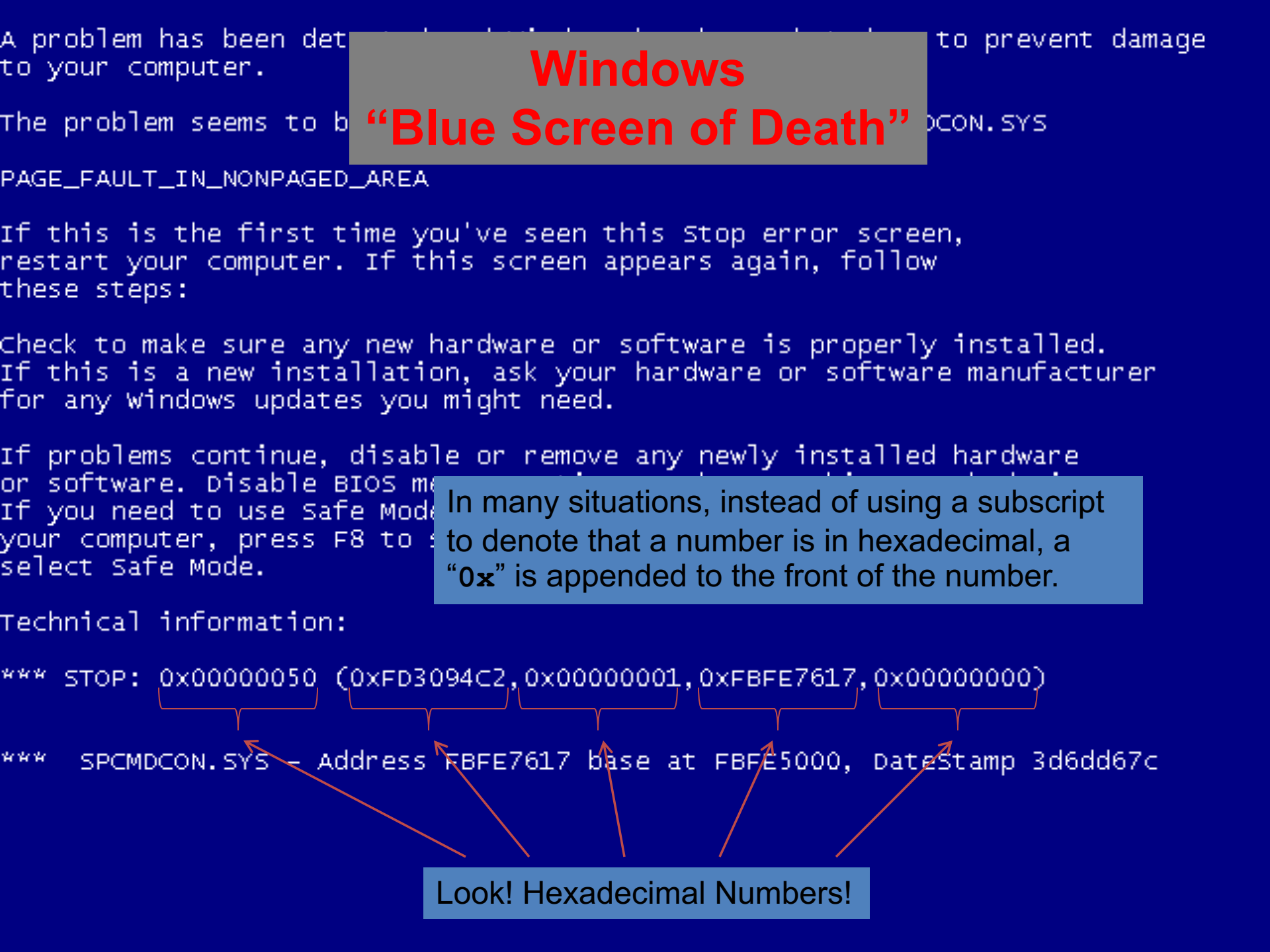



**Base 16**
**Cheat Sheet**
$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

First, split the binary number into groups of four digits, starting with the least significant digit.

Next, convert each group of four binary digits to a single hex digit.

Put the single hex digits together in the order in which they were found, and you're done!




# Windows "Blue Screen of Death"

In many situations, instead of using a subscript to denote that a number is in hexadecimal, a "**0x**" is appended to the front of the number.

Look! Hexadecimal Numbers!

there are 10 types of people in this world, those who understand binary and those who don’t

# MEMORY MANAGEMENT IN JAVA

# Heap vs Stack – Memory Allocation in Java

- **Java Heap Space**
  - Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it's always created in the Heap space.

- **Java Stack Memory**
  - Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method.
  - Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.
  - As soon as method ends, the block becomes unused and become available for next method.
  - Stack memory size is very less compared to Heap memory.

# Note

When stack memory is full:

Java runtime throws java.lang.StackOverFlowError

whereas

When heap memory is full:

it throws java.lang.OutOfMemoryError: Java Heap Space error.

# BigObject

```
public class BigObject {

    int[]  largeArray;

    BigObject() {
        largeArray = new int[10000000];
    }

}
```

# Errors

```
public class Errors {
    public static void stackOverflow(int i) {
        System.out.println(i);
        stackOverflow(i+1);
    }

    public static void outOfMemory(int j) {
        ArrayList<BigObject> boa = new ArrayList<>();

        for (int i = 0; i < 100; i++) {

            BigObject anotherBigObject = new BigObject();
            boa.add(anotherBigObject);
            System.out.println(i);
        }   }

    public static void main (String[] args) {

        stackOverflow(1);
        outOfMemory(1);

    }
}
```

# Memory Management

- One strength of the Java is that it performs automatic memory management
  - shielding the developer from the complexity of explicit memory management.

# Explicit vs. Automatic Memory Management

- Memory management is the process of recognizing when allocated objects are no longer needed, deallocating(freeing) the memory used by such objects, and making it available for subsequent allocations.

- In some programming languages (Ex. C, C++), memory management is the programmer's responsibility.
  - Leads to many common errors that can cause unexpected or erroneous program behavior and crashes.
  - A large proportion of developer time is often spent debugging and trying to correct such errors.

# Problems with Explicit Memory Management

- Dangling references
  - It is possible to deallocate the space used by an object to which some other object still has a reference. If the object with that (dangling) reference tries to access the original object, but the space has been reallocated to a new object, the result is unpredictable and not what was intended.

- Memory leaks
  - These leaks occur when memory is allocated and no longer referenced but is not released. For example, if you intend to free the space utilized by a linked list but you make the mistake of just deallocating the first element of the list, the remaining list elements are no longer referenced but they go out of the program's reach and can neither be used nor recovered. If enough leaks occur, they can keep consuming memory until all available memory is exhausted.

# Garbage collector --- Java's way to fix the problem

- Avoids the dangling reference problem, because an object that is still referenced somewhere will never be garbage collected and so will not be considered free.

- Garbage collection also solves the memory leak problem since it automatically frees all memory no longer referenced.

- Reference and Further reading: http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf (Not required for quizzes or exams)

# JAVA GENERICS

- Resources:
  - https://docs.oracle.com/javase/tutorial/java/generics/index.html
  - https://docs.oracle.com/javase/tutorial/extra/generics/index.html

# Why use Generics

- Stronger type checks at compile time
  - A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety.
  - Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

- Elimination of casts

- Enabling programmers to implement generic algorithms
  - Programmers can implement generic algorithms
    - work on collections of different types
    - can be customized
    - are type safe
    - easier to read.

# Error or no error?

```
1  package lec3;
2
3  import java.util.*;
4
5  public class Gen1 {
6
7      public static void main(String[] args) {
8          List list = new ArrayList();
9          list.add("hello1");
.0         String s = list.get(0);
.1         System.out.println(s);
.2     }
.3
.4 }
.5
```

# Solution 1

```
1  package lec3;
2
3  import java.util.*;
4
5  public class Gen2 {
6
7      public static void main(String[] args) {
8          List list = new ArrayList();
9          list.add("hello2");
10         String s = (String) list.get(0);
11         System.out.println(s);
12     }
13
14 }
```

# Solution 2

```
1  package lec3;
2
3  import java.util.*;
4
5  public class Gen3 {
6
7      public static void main(String[] args) {
8          List<String> list = new ArrayList<>();
9          list.add("hello3");
10         String s = list.get(0);    // no cast
11         System.out.println(s);
12     }
13
14 }
15
```

# Another Example

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }

    public static void main(String[] args) {
        Box box_int = new Box();
        Box box_String = new Box();

        box_int.set(5);
        int x = (int)box_int.get();
        System.out.println(x);

        box_String.set("Hello");
        String y = (String)box_String.get();
        System.out.println(y);

        /*
        box_int.set("CSC172");
        int x1 = (int)box_int.get();
        System.out.println(x1);
        */
    }

}
```

- What if we uncomment the last block

```
21          // Comment vs Uncomment
22          /*
23          box_int.set("CSC172");
24          int x1 = (int)box_int.get();
25          System.out.println(x1);
26          */
```

```
Exception in thread "main" Hello
java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer
        at lec3.Box.main(Box.java:23)
```

# Generic Methods

- *Generic methods* are methods that introduce their own type parameters.

- Similar to declaring a generic type, but the type parameter's scope is limited to the method where it is declared.

- Static and non-static generic methods are allowed, as well as generic class constructors.

# Example

```
public class Util {
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}

public class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public void setKey(K key) { this.key = key; }
    public void setValue(V value) { this.value = value; }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```

# Example (cont.)

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.<Integer, String>compare(p1, p2);
```

The type has been explicitly provided, as shown in bold. Generally, this can be left out and the compiler will infer the type that is needed:

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.compare(p1, p2);
```

# Coding Time

- Write a method which takes 2 arguments:
  - 1. An array of integer `anArray`
  - 2. An integer `elem`

  - Find how many integers in this array are greater than `elem`

- Write a main method to test your code

# Method (Not Generic)

```
public static int countGreaterThan(int[] anArray, int elem)
{
    int total = 0;
    for (int val:anArray) {
        if (val > elem) {
            total++;
        }
    }
    return total;
}
```

# Method (Generic)

```
public static <T extends Comparable<T>> int
              countGreaterThan(T[] anArray, T elem)
{
    int total = 0;
    for (T val:anArray) {
        if (val.compareTo(elem) > 0) {
            total++;
        }
    }
    return total;
}
```

# Acknowledgement

- ORACLE Java Documentation
- Numerous resources available on the Web