



# Introduction to Software Design

# Welcome to CS 3!



# Outline

- 1 Correctness
- 2 Managing Complexity
  - Abstraction
- 3 Introducing C

What does it mean for a program to be “correct”?

What does it mean for a program to be “correct”?

What does this code do?

```
--( __, __, __ ) { __ / __ <= 1 ? ( __, __ + 1, __ ) : ! ( __ % __ ) ? ( __, __ + 1, 0 ) : __ % __ == __ / __ && ! __ ? ( printf ( "%d\t", __ / __ ), ( __, __ + 1, 0 ) ) : __ % __ > 1 && __ % __ < __ / __ ? ( __, 1 + __, __ + ! ( __ / __ % ( __ % __ ) ) ) : __ < * __ ? ( __, __ + 1, __ ) : 0 ; } main () { ( 100, 0, 0 ) ; }
```

What does it mean for a program to be “correct”?

What does this code do?

```
--( __, __, __ ) { __ / __ <= 1 ? ( __, __ + 1, __ ) : ! ( __ % __ ) ? ( __, __ + 1, 0 ) : __ % __ == __ / __ && ! __ ? ( printf ( "%d\t", __ / __ ), ( __, __ + 1, 0 ) ) : __ % __ > 1 && __ % __ < __ / __ ? ( __, 1 + __, __ + ! ( __ / __ % ( __ % __ ) ) ) : __ < * __ ? ( __, __ + 1, __ ) : 0; } main() { ( 100, 0, 0 ); }
```

Programs must be written for people to read, and only incidentally for machines to execute. (Abelson & Sussman)

## What “Program Correctness”?


- The code has the right functionality

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read



## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented 

## What "Program Correctness"?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular



## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use
- ...

## How Can We Determine If A Program is Correct?

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use
- ...

## How Can We Determine If A Program is Correct?

- Write tests

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use
- ...

## How Can We Determine If A Program is Correct?

- Write tests
- Think about “edge cases”

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use
- ...

## How Can We Determine If A Program is Correct?

- Write tests
- Think about “edge cases”
- Use automated tools to catch unintended issues

## What “Program Correctness”?

- The code has the right functionality
- The code is easy to read
- The code is well-documented
- The code is modular
- The code is easy to re-use
- ...

## How Can We Determine If A Program is Correct?

- Write tests
- Think about “edge cases”
- Use automated tools to catch unintended issues
- Ask someone experienced to review your code! (more on this later)



Bigger programs lead to bigger problems. . .

Bigger programs lead to bigger problems. . .

- Small programs are simple and malleable
- Big programs are complex and inflexible

Bigger programs lead to bigger problems. . .

- Small programs are simple and malleable
- Big programs are complex and inflexible

In large programs, **interactions** become unmanagable

Bigger programs lead to bigger problems. . .

- Small programs are simple and malleable
- Big programs are complex and inflexible

In large programs, **interactions** become unmanagable

There are a lot of ways to mitigate this, but two of the most useful are:

- Abstraction
- Specification

The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context. (John V. Guttag)

The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context. (John V. Guttag)

Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts. (Benjamin C. Pierce)

The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context. (John V. Guttag)

Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts. (Benjamin C. Pierce)

## Some Types of Abstraction

- **Procedural Abstraction.** Splitting a program into functions that each have a single purpose
- **Data Abstraction.** Using ADTs and interfaces to make a boundary between client and implementor

The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context. (John V. Guttag)

Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts. (Benjamin C. Pierce)

## Some Types of Abstraction

- **Procedural Abstraction.** Splitting a program into functions that each have a single purpose
- **Data Abstraction.** Using ADTs and interfaces to make a boundary between client and implementor

## Example

The Caltech CS Introductory Curriculum





Programming

Data Structures

Software Design

Python

Java

C

50 LOC

300 LOC

800 LOC

CS 3 is a practical introduction to designing large programs in a low-level language. Heavy emphasis is placed on documentation, testing, and software architecture. Students will work in teams in two 5-week long projects. In the first half of the course, teams will focus on testing and extensibility. In the second half of the course, teams will use POSIX APIs, as well as their own code from the first five weeks, to develop a large software deliverable. Software engineering topics covered include code reviews, testing and testability, code readability, API design, refactoring, and documentation.

# Outline

1 Correctness

2 Managing Complexity

- Abstraction

3 Introducing C

## A String in Java

```
1 String str = "hi";  
2  
3 // what's actually happening:  
4 char[] str = new char[2];  
5 str[0] = 'h';  
6 str[1] = 'i';  
7 System.out.println(str);
```

## A String in Java

```
1 String str = "hi";
2
3 // what's actually happening:
4 char[] str = new char[2];
5 str[0] = 'h';
6 str[1] = 'i';
7 System.out.println(str);
```

for

## A String in C?

```
1 char *str = malloc(2 * sizeof(char));
2 str[0] = 'h';
3 str[1] = 'i';
4 printf("%s\n", str);
5 free(str);
```

## Another Attempt

```
1 (char *str = malloc(2 * sizeof(char));  
2 strcpy(str, "hi");  
3 printf("%s\n", str);  
4 free(str);
```

→ malloc((LEN + 1) \* sizeof(char))

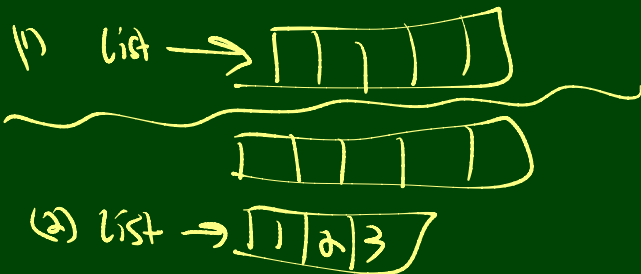
↓

2

→ malloc((strlen("hi") + 1) \* sizeof(char))

## Bad Java Code

```
1 List<Integer> list = new ArrayList<>();  
2 list = List.of(1, 2, 3);
```



## Bad Java Code

```
1 List<Integer> list = new ArrayList<>();  
2 list = List.of(1, 2, 3);
```

## C Code?

```
1 char *str = malloc(2 * sizeof(char));  
2 str = "hi";  
3 free(str);
```