

Computational Programming and Languages in Simple Words

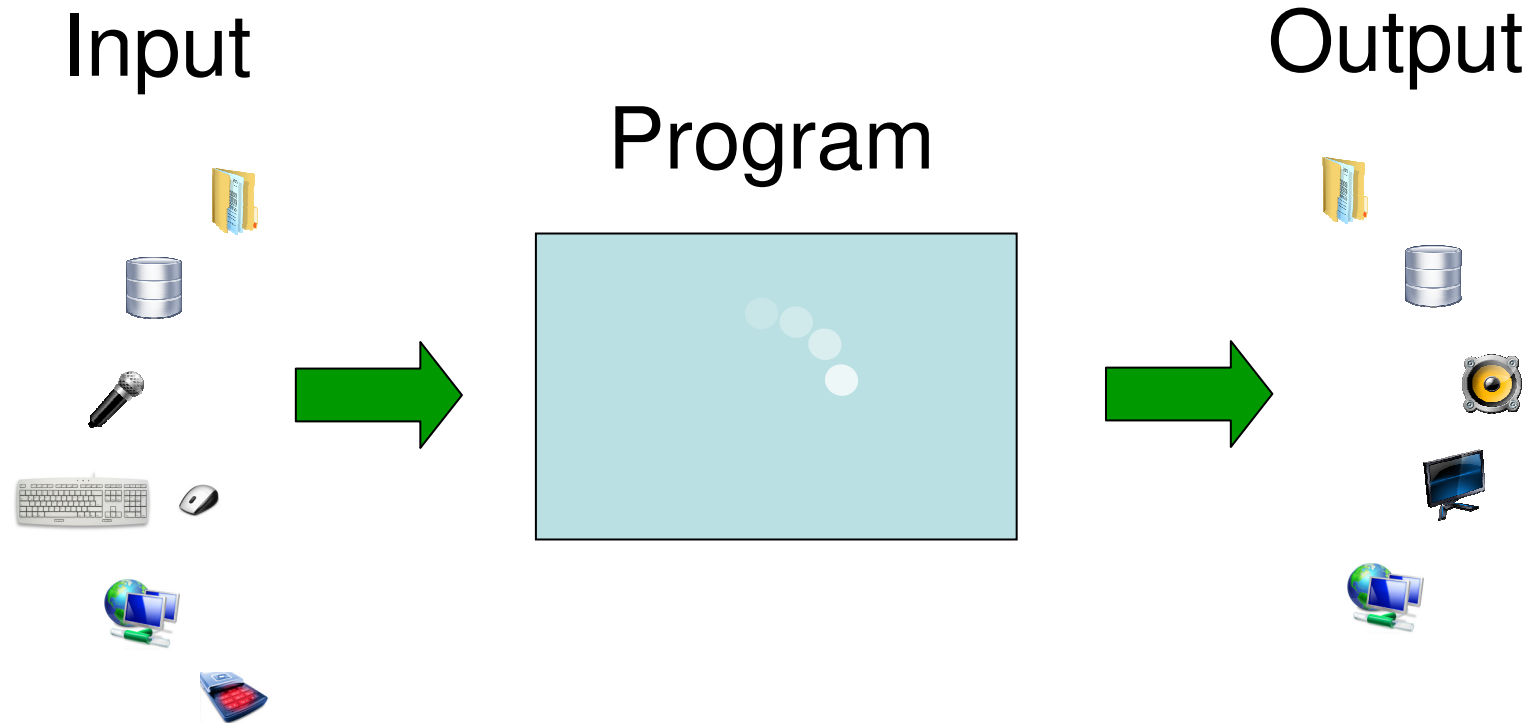
By Mark Burkitt

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - Control Statements
3. Real World Example
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

What is a Computer Program?

- A series of instructions that process input data and produce output data.



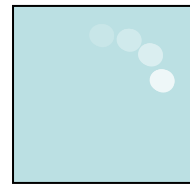
What is a Computer Program?

- Each instruction is processed by the computer, one at a time

Program

```
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret
cmp edx, 2
ja @f
mov eax, 1
ret
push ebx
mov ebx, 1
mov ecx, 1
```

Computer (CPU)



Evolution of Programming Languages

- Modern classification systems recognise 4 generations of programming language:
 - 1st Generation: Machine Code
 - 2nd Generation: Assembly Language
 - 3rd Generation: Higher Level Languages
 - C / C++, Fortran, Pascal, Java, .NET, ...
 - 4th Generation: Purpose Specific
 - Matlab, SQL, R, ...

A Simple Example

Problem: Calculate the Fibonacci Sequence

0, **1**, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

$$F_n = F_{n-1} + F_{n-2}$$

Calculate the Fibonacci Sequence

$$F_n = F_{n-1} + F_{n-2}$$

Machine Code (1GL)

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

Higher Level Language (3GL)

```
long fib(int n) {
    if (n < 2) {
        return n;
    }
    else {
        return fib(n-1) + fib(n-2);
    }
}
```

Assembly (2GL)

```
fib:
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret

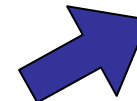
@@:
cmp edx, 2
ja @f
mov eax, 1
ret

@@:
push ebx
mov ebx, 1
mov ecx, 1

@@:
    lea eax, [ebx+ecx]
    cmp edx, 3
    jbe @f
    mov ebx, ecx
    mov ecx, eax
    dec edx

    jmp @b

@@:
pop ebx
ret
```



Talk Outline

1. Background to Programming
2. Programming Language Basics
 - **How Computers View Data**
 - Basic Types
 - Control Statements
3. Real World Example
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

Binary

- Computers store and process data in binary

0 1 0 0 1 1 0 0

Binary

- Computers store and process data in binary

1	2	4	8	16	32	64	128
0	1	0	0	1	1	0	0

$$F_n = 2^{n-1}$$

Binary

- Computers store and process data in binary

1	2	4	8	16	32	64	128
0	1	0	0	1	1	0	0

$$2 + 16 + 32 = \underline{50}$$

Binary

- Computers store and process data in binary

1	2	4	8	16	32	64	128
1	0	1	0	1	0	0	1

$$1 + 4 + 16 + 128 = \underline{149}$$

Binary

- Computers store and process data in binary

1	2	4	8	16	32	64	128
0	0	0	0	0	0	0	0

= 0

Binary

- Computers store and process data in binary

1	2	4	8	16	32	64	128
1	1	1	1	1	1	1	1

$$\begin{aligned} &1 + 2 + 4 + 8 + 16 \\ &+ 32 + 64 + 128 = \underline{255} \end{aligned}$$

-185.25752 0.574 0.053
0.587e⁶
25.06 0.0095 -7.05

Real Numbers (float)

0.456e⁻¹⁰ 1.248226245
30.00575
256.0 0 0.0576
-1.987

“Good Morning”

“Welcome”

“Biology”

“Hello World!”

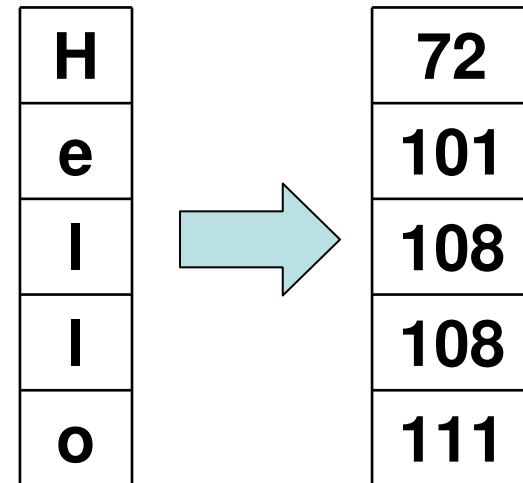
“Press any key”

“22 days”

“Loading Please Wait”

Text (String)

65	A	73	I	97	a	105	i
66	B	74	J	98	b	106	j
67	C	75	K	99	c	107	k
68	D	76	L	100	d	108	l
69	E	77	M	101	e	109	m
70	F	78	N	102	f	110	n
71	G	79	O	103	g	111	o
72	H	80	P	104	h	112	p



Images

Audio

Video

Measurements

Text

Documents

Records

Complex Data

Type / Class

Arrays

MyRecord
Name: John
Age: 24
Height: 1.8
Weight: 18.34
Fav. Colour: Green

Whole Numbers

-12
101
9
108
111

Real Numbers

0.25
12.52
-9.875308
102.551
0.00582

Type / Class

MyRecord
MyRecord
MyRecord
MyRecord
MyRecord

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - **Basic Types**
 - Control Statements
3. Real World Example
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

Variables

- Used by computer programs to store data

Type	Name	Value
<code>int</code>	<code>noOfApples</code>	<code>= 5;</code>
<code>float</code>	<code>appleWeight</code>	<code>= 8.25;</code>
<code>int [8]</code>	<code>applesPerTree;</code>	
	<code>applesPerTree[0]</code>	<code>= 3;</code>

Constants

- Similar to Variables, the value can only be set once, and never changed.
- Useful for values which will not change
 - global constants such as PI.

Type	Name	Value
<code>const int</code>	<code>noOfApples</code>	<code>= 5;</code>

Using Variables & Constants

- Variables and Constants can be used in the same way as algebraic terms.

$$A = B + C \quad A = B - C \quad A = B * C \quad A = B / C$$

```
float A;  
const int B = 15;  
const float C = 0.35;
```

```
A = B + C;
```

```
A = B - C;
```

```
A = B * C;
```

```
A = B / C;
```



A Simple Problem

John



Paul



John has 3 boxes of apples, each box contains 13 apples. Paul gives John 2 boxes of apples, and then takes 5 apples.

- How many apples does John have?

```
const int APPLES_PER_BOX = 13;

int johnBoxes = 3;

int paulBoxes = 2;

johnBoxes = johnBoxes + paulBoxes;

paulsBoxes = 0;

int paulApples = 5;

int johnApples = (johnBoxes * APPLES_PER_BOX) - paulApples;
```

60

Creating Records

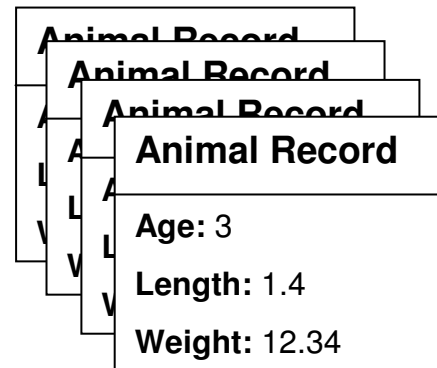
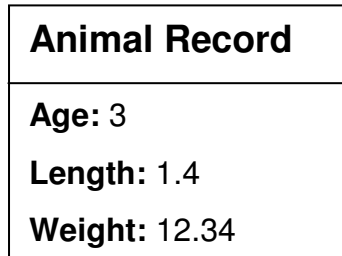
Animal Record
Age: 3
Length: 1.4
Weight: 12.34

```
struct AnimalRecord {  
    int age;  
    float length;  
    float weight;  
};
```

```
int age = 3;  
  
float length = 1.4;  
  
float weight = 12.34;
```

```
AnimalRecord myPig;  
  
myPig.age = 3;  
  
myPig.length = 1.4;  
  
myPig.weight = 12.34;
```

Multiple Records



```
struct AnimalRecord {  
    int age;  
    float length;  
    float weight;  
};
```

```
AnimalRecord myPig;  
  
myPig.age = 3;  
myPig.length = 1.4;  
myPig.weight = 12.34;
```

```
AnimalRecord[100] myPig;
```

```
myPig[0].age = 3;  
myPig[0].length = 1.4;  
myPig[0].weight = 12.34;
```

```
myPig[1].age = 4;  
myPig[1].length = 1.3;  
myPig[1].weight = 14.58;
```

```
myPig[99].age = 5;  
myPig[99].length = 1.2;  
myPig[99].weight = 10.01;
```

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - **Control Statements**
3. Real World Example
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

Boolean Operations

- Relationship between two numbers can be expressed as a series of statements

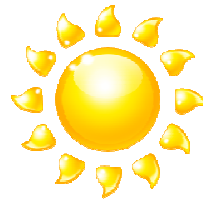
$A = B$ $A \neq B$ $A > B$ $A < B$ $A \geq B$ $A \leq B$

```
bool myCondition;  
int A = 5;  
int B = 10;  
  
myCondition = (A == B);  
myCondition = (A != B);  
  
myCondition = (A > B);  
myCondition = (A < B);  
  
myCondition = (A >= B);  
myCondition = (A <= B);
```

IF Statements

- The IF statement is used to make decisions about what to do next

Jenny



IF Statements

- The IF statement is used to make decisions about what to do next

```
bool theSunIsShining = true;

if (theSunIsShining) {
    //Go to the beach
}
else {
    //Go to the cinema
}
```

```
int x = 10;

if (x < 100) {
    //condition dependent operation
}
else {
    //alternative operation
}
```

FOR Loops

- Loops allow the same section of code to be called multiple times

```
for (int i = 0; i < 10; i++) {  
    //section of code to repeat  
}
```

Using FOR Loops

Task: Set the value of each item in an array of numbers to 100

```
int[6] myNumbers;  
  
myNumbers[0] = 100;  
myNumbers[1] = 100;  
myNumbers[2] = 100;  
myNumbers[3] = 100;  
myNumbers[4] = 100;  
myNumbers[5] = 100;
```

```
int[6000] myNumbers;  
  
for (int i = 0; i < 6000; i++) {  
    myNumbers[i] = 100;  
}
```

Functions

- To perform calculations which will be used throughout the program

```
float simpleCalculation(int a, float b) {  
    float simpleResult = a * b;  
    return simpleResult;  
}
```

```
float v1 = simpleCalculation(10, 0.02);  
float v2 = simpleCalculation(5, 0.03);  
float v3 = simpleCalculation(-5, 0.04);  
float v4 = simpleCalculation(-10, 0.05);
```

Functions

```
float complexCalculation(int a, int b, float c, float d) {  
  
    float calculatedValue = 0;  
  
    if(a > 0) {  
        for(int i = 0; i < a; i++) {  
            calculatedValue += (b * (c + i)) / (d * d);  
        }  
    }  
    else {  
        calculatedValue = ((a * b) + (c * d * d)) / ((a - b) * c * c);  
    }  
  
    return calculatedValue;  
}
```

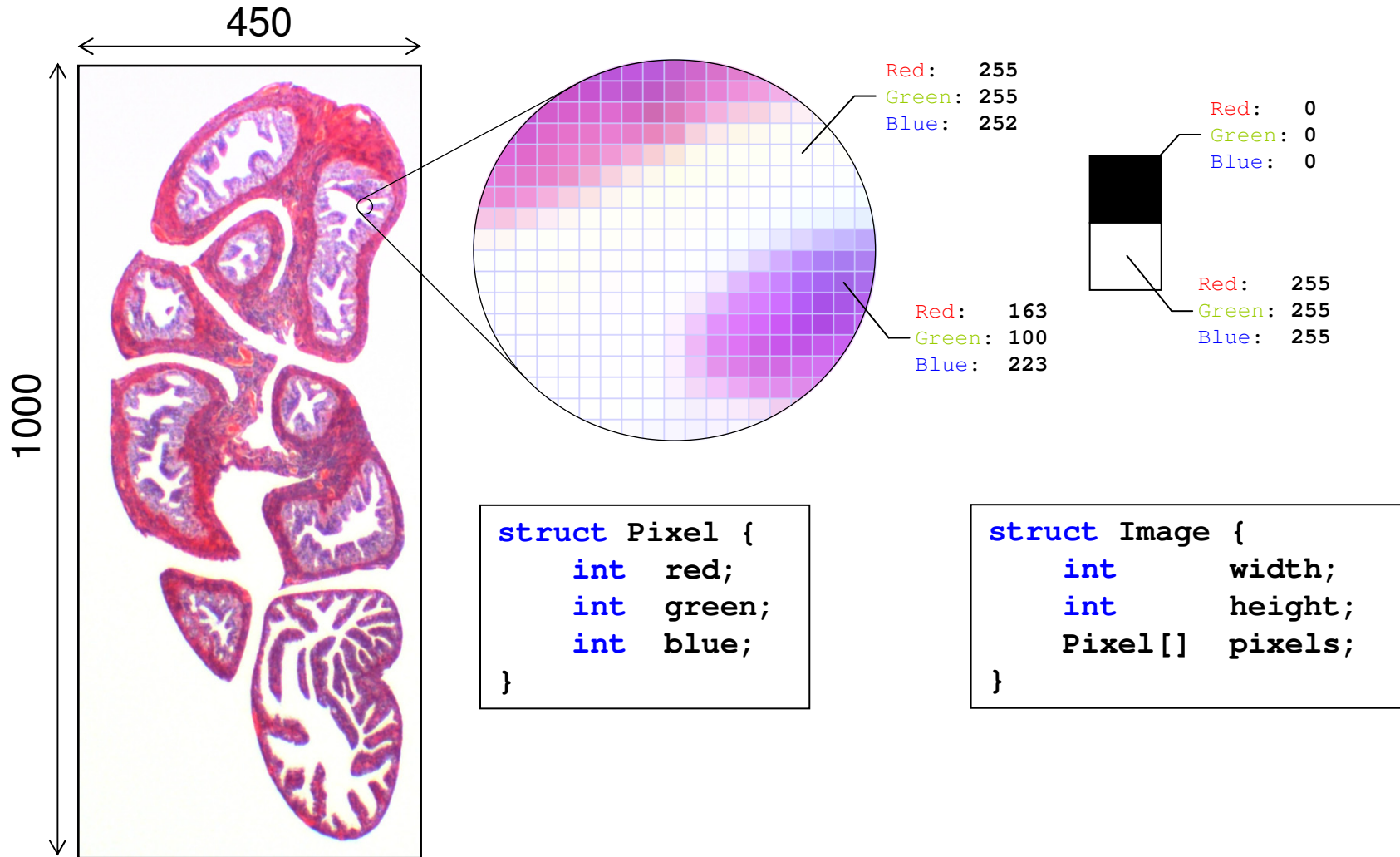
```
float v1 = complexCalculation(10, 4, 2.4, 0.02);  
  
float v2 = complexCalculation(5, 4, 2.4, 0.03);  
  
float v3 = complexCalculation(-5, 4, 2.4, 0.04);  
  
float v4 = complexCalculation(-10, 4, 2.4, 0.05);
```

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - Control Statements
- 3. Real World Example**
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

Calculate the surface area of tissue from a histology image

What does an image look like?

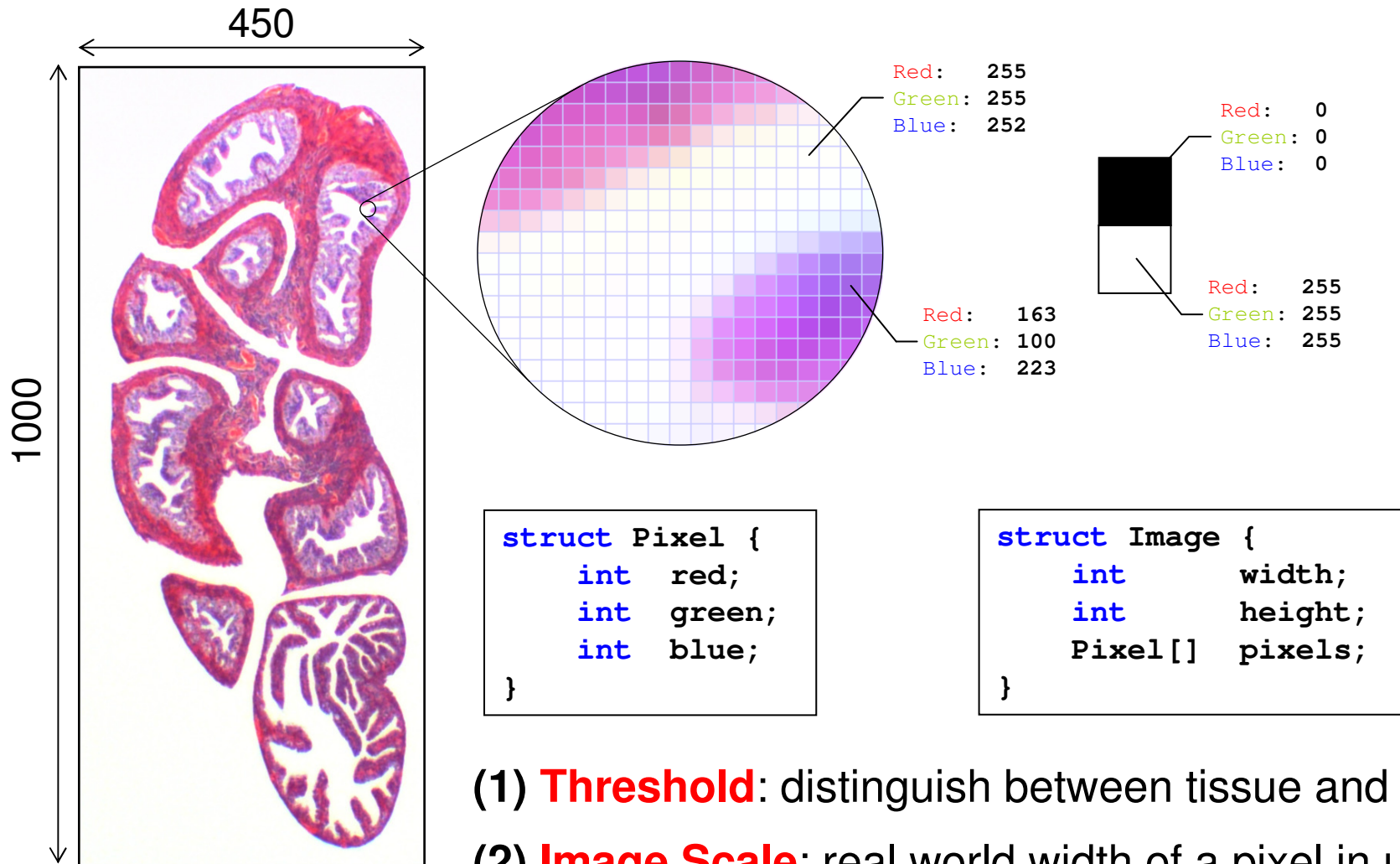


Task: Calculate tissue surface area

A New Program

```
int main() {  
  
    return 0;  
}
```

How to process the image?



- (1) **Threshold**: distinguish between tissue and slide
- (2) **Image Scale**: real world width of a pixel in μm

Task: Calculate tissue surface area

Define Constants & Load Data

```
const int THRESHOLD = 200;
const float UM_PER_PX = 0.23;
const float UM_PER_PX_SQ = UM_PER_PX * UM_PER_PX;

int main() {
    Image img = LoadImage("histology_image.jpg");

    return 0;
}
```

```
struct Pixel {
    int red;
    int green;
    int blue;
}
```

```
struct Image {
    int width;
    int height;
    Pixel[] pixels;
}
```

Task: Calculate tissue surface area

Initialise Variables

```
const int THRESHOLD = 200;
const float UM_PER_PX = 0.23;
const float UM_PER_PX_SQ = UM_PER_PX * UM_PER_PX;

int main() {
    Image img = LoadImage("histology_image.jpg");
    int numberOfPixels = img.width * img.height;
    int pixelArea = 0;

    return 0;
}
```

```
struct Pixel {
    int red;
    int green;
    int blue;
}
```

```
struct Image {
    int width;
    int height;
    Pixel[] pixels;
}
```

Task: Calculate tissue surface area

Process Image Data

```
const int THRESHOLD = 200;
const float UM_PER_PX = 0.23;
const float UM_PER_PX_SQ = UM_PER_PX * UM_PER_PX;

int main() {
    Image img = LoadImage("histology_image.jpg");
    int numberOfPixels = img.width * img.height;
    int pixelArea = 0;

    for(int i = 0; i < numberOfPixels; i++) {
        Pixel px = img.pixels[i];
        float intensity = CalculateLuminance(px);
        if (intensity <= THRESHOLD) {
            pixelArea += 1;
        }
    }

    return 0;
}
```

```
struct Pixel {
    int red;
    int green;
    int blue;
}
```

```
struct Image {
    int width;
    int height;
    Pixel[] pixels;
}
```

Task: Calculate tissue surface area

Calculate Luminance

Based on Luminosity Function

Red: 0.2126

Green: 0.7152

Blue: 0.0722

```
// Y = (R * 0.2126) + (G * 0.7152) + (B * 0.0722)
float CalculateLuminance(Pixel px) {
    float intensity;

    intensity = (px.red * 0.2126) +
                (px.green * 0.7152) +
                (px.blue * 0.0722);

    return intensity;
}
```

```
struct Pixel {
    int red;
    int green;
    int blue;
}
```

```
struct Image {
    int width;
    int height;
    Pixel[] pixels;
}
```

Task: Calculate tissue surface area

Calculate and Print Result

```
const int THRESHOLD = 200;
const float UM_PER_PX = 0.23;
const float UM_PER_PX_SQ = UM_PER_PX * UM_PER_PX;

int main() {
    Image img = LoadImage("histology_image.jpg");
    int numberOfPixels = img.width * img.height;
    int pixelArea = 0;

    for(int i = 0; i < numberOfPixels; i++) {
        Pixel px = img.pixels[i];
        float intensity = CalculateLuminance(px);
        if (intensity <= THRESHOLD) {
            pixelArea += 1;
        }
    }

    float area = pixelArea * UM_PER_PX_SQ;
    printf("Tissue Surface Area: %f μm^2", area );

    return 0;
}
```

```
struct Pixel {
    int red;
    int green;
    int blue;
}
```

```
struct Image {
    int width;
    int height;
    Pixel[] pixels;
}
```

1276.53 μm²

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - Control Statements
3. Real World Example
- 4. Choosing a Language**
5. Developing Complex Systems
6. Learning a Programming Language

Things to Consider

- What do you want to do?
- Who will you use it?
- How much time do you have?
- Who is going to write the program?
- Will it be single use, or could it be extended in the future?

Types of Language

- Compiled
 - C, C++, Fortran, Pascal
- VM Compiled
 - Java, C#, .NET
- Scripted
 - PHP, Perl, Javascript, Ruby, BASIC
- Matlab (4GL)
 - Expensive!
 - Octave is free alternative

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data
Code Security:	Secure	Quite Secure - Can be decompiled (Obfuscation)	Not Secure	Not Secure - Can be Compiled

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data
Code Security:	Secure	Quite Secure - Can be decompiled (Obfuscation)	Not Secure	Not Secure - Can be Compiled
Learning Curve:	Steep	Quite Steep	Easy	Easy for Basics

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data
Code Security:	Secure	Quite Secure - Can be decompiled (Obfuscation)	Not Secure	Not Secure - Can be Compiled
Learning Curve:	Steep	Quite Steep	Easy	Easy for Basics
Library Support:	Good	Best	Limited	Good maths / visualisation / analysis

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data
Code Security:	Secure	Quite Secure - Can be decompiled (Obfuscation)	Not Secure	Not Secure - Can be Compiled
Learning Curve:	Steep	Quite Steep	Easy	Easy for Basics
Library Support:	Good	Best	Limited	Good maths / visualisation / analysis
Debugging:	Good	Best	Poor	Good

Comparison of Languages and Properties

	3GL			4GL
	Compiled	VM Compiled	Scripted	Matlab
Strongly Typed:	Strong	Strong	Weak	Implicit
OS / Arc. Neutral:	No	Yes (requires Virtual Machine)	Yes (requires Interpreter)	Yes (requires Matlab)
Performance:	Fastest	Fast	Slowest	Fast maths
Memory Management:	Low Level Direct Access	Partial Control (Garbage Collection)	None	Load / Unload Data
Code Security:	Secure	Quite Secure - Can be decompiled (Obfuscation)	Not Secure	Not Secure - Can be Compiled
Learning Curve:	Steep	Quite Steep	Easy	Easy for Basics
Library Support:	Good	Best	Limited	Good maths / visualisation / analysis
Debugging:	Good	Best	Poor	Good
Robust Software:	Hard	Easy	Average	Average

Multi-Lingual

- Most languages have the same or similar data types, common functions and libraries.
- Once you have learned one language, learning another is easier.
- If you know several, you will be able to understand code written in any language.

Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - Control Statements
3. Real World Example
4. Choosing a Language
- 5. Developing Complex Systems**
6. Learning a Programming Language

Requirements



- Formal document
- Detailed description of what you want
- Describes inputs and outputs, data formats, parameters and processes
- Any professional software developer can develop the system
- In business, the requirements document can act as a legally binding contract

Design

- Identify areas of common functionality

Record Types
(classes / structs)

Common Processes

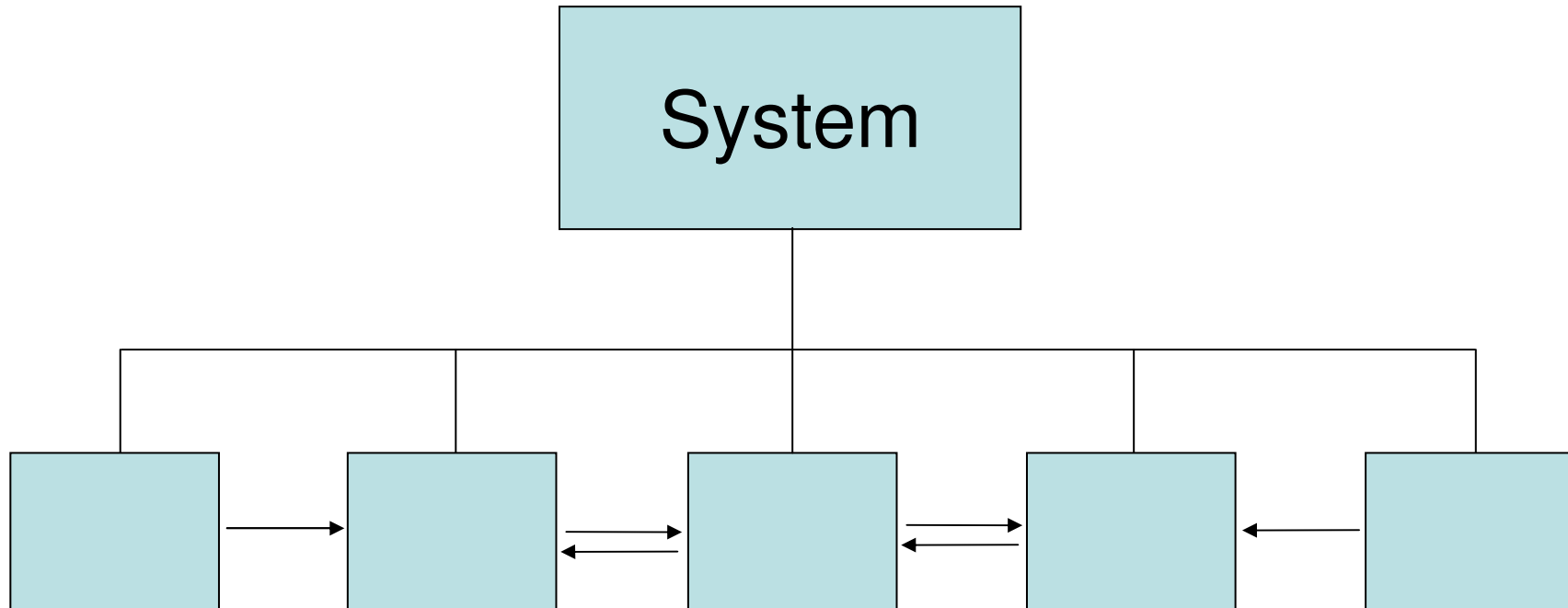
Inputs

Generated Outputs



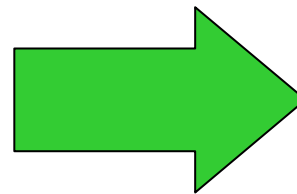
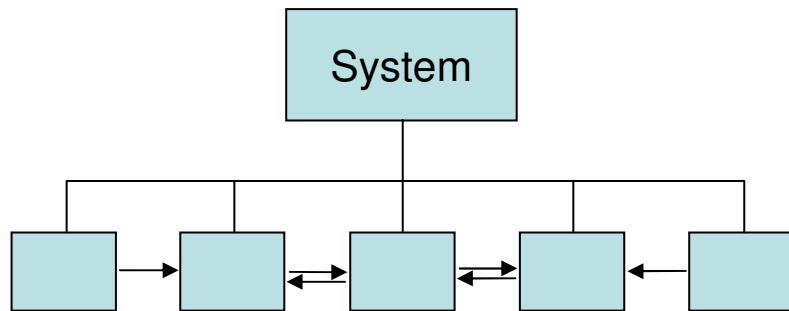
Design

- Break system down into smaller modules



Design

- Document the design using diagrams and formal methods where possible
 - UML Diagrams
 - Data Flow Diagrams
 - Numerous Others



Implementation

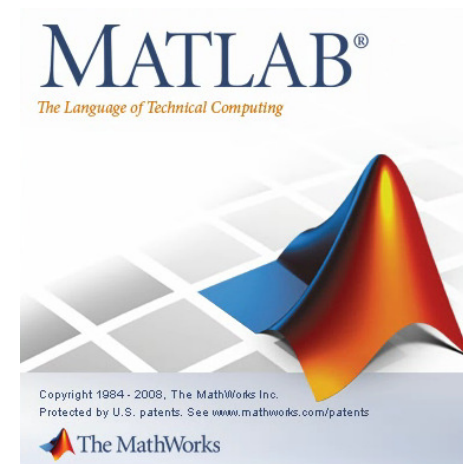
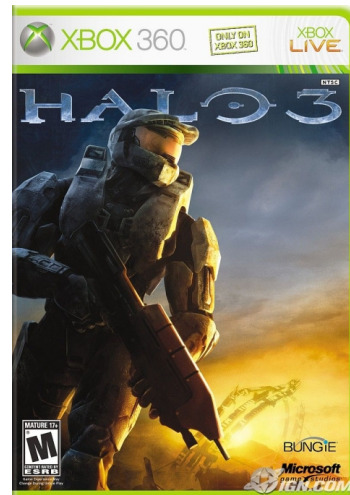
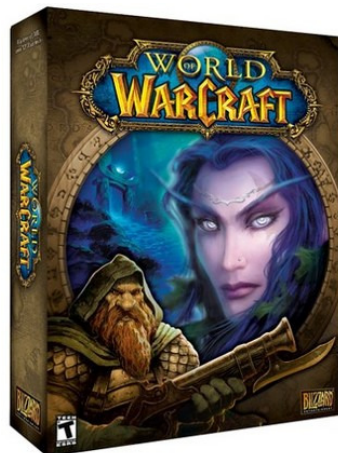
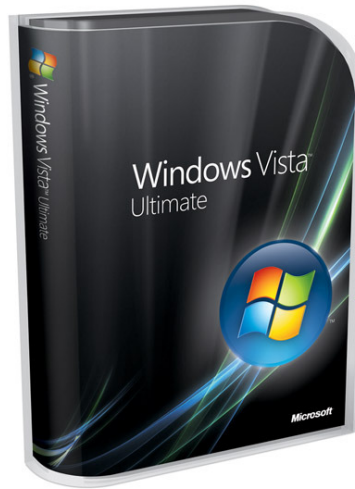
- Implement the system as described in the design documentation
- Use existing libraries where available
 - Reduce development time & bugs
- Use unit testing if appropriate
- Adhere to coding standards
 - Consistent naming conventions, etc.
- Follow common design patterns

Testing

Typically 3 types:

- Does the system work without breaking?
 - Tested during and after implementation
- Does the system match the requirements?
 - Tested by programmer and user
- Does the system do what I want?
 - Tested by user

Is all software developed like this?

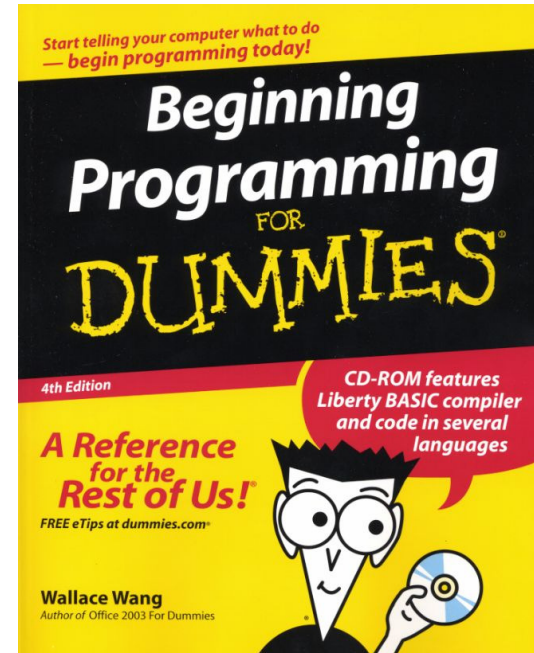


Talk Outline

1. Background to Programming
2. Programming Language Basics
 - How Computers View Data
 - Basic Types
 - Control Statements
3. Real World Example
4. Choosing a Language
5. Developing Complex Systems
6. Learning a Programming Language

How to Learn a Programming Language

- Get a book!
 - Work through all the exercises – don't just read them, do them!
 - You cannot learn how to program from lectures.
- You learn by doing (and failing)
 - If a program you wrote does not work, you learn more by trying to fix it (trial & error) than if someone shows you how to do it.
- Use the Internet
 - Lots of examples, tutorials
 - Type something into Google, and you will find an answer



How to Learn a Programming Language

- Learning to program well takes a very long time
 - Only a small % of a computing degree is on programming, the rest is on designing robust systems and theoretical techniques.
 - Most computer science graduates don't have the experience to write good quality software systems.
- If you are serious about it, do a MSc.

Thank you for your attention

01010100 01101000 01100001 01101110 01101011 00100000 01111001
01101111 01110101 00100000 01100110 01101111 01110010 00100000
01111001 01101111 01110101 01110010 00100000 01100001 01110100
01110100 01100101 01101110 01110100 01101001 01101111 01101110

Are there any questions?

01000001 01110010 01100101 00100000 01110100 01101000 01100101
01110010 01100101 00100000 01100001 01101110 01111001 00100000
01110001 01110101 01100101 01110011 01110100 01101001 01101111
01101110 01110011 00111111