Call Conventions

Programming Fundamentals

Dr Kevin Chalmers

School of Computing
Edinburgh Napier University
Edinburgh

k.chalmers@napier.ac.uk

Outline

Outline

Outline

Outline

Outline

- Developed in the late 1970s / early 1980s

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes
  - C++17 (2017) hoped to be another major update

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes
  - C++17 (2017) hoped to be another major update
- Biggest advantage of C++ is *object-orientation*

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes
  - C++17 (2017) hoped to be another major update
- Biggest advantage of C++ is *object-orientation*
  - This will be our main use

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes
  - C++17 (2017) hoped to be another major update
- Biggest advantage of C++ is *object-orientation*
  - This will be our main use
  - Also allows reference datatypes (this unit)

Introduction to C++

- Developed in the late 1970s / early 1980s
- Adds new features to C
- ++ indicates that C++ is an increment on C
  - C is more or less a subset of C++
- Other languages are also considered *extensions of C*
  - Objective C (popular with Apple)
  - D (a contender for a C++ replacement)
  - And others
- On a wider scale, C++ language development is a continuous process
  - Standards committee
  - C++11 (2011) was a major update
  - C++14 (2014) has some incremental changes
  - C++17 (2017) hoped to be another major update
- Biggest advantage of C++ is *object-orientation*
  - This will be our main use
  - Also allows reference datatypes (this unit)
  - Other features like generic programming also powerful (outside scope of this module)

Hello World in C

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    return 0;
}
```

Hello World in C and C++

Hello World in C

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    return 0;
}
```

Hello World in C++

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

Hello World in C and C++

Hello World in C

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    return 0;
}
```

Hello World in C++

```cpp
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

- Note that the C version can also be compiled in C++

Questions?

Outline

Passing Data to a Function

- So far we have not discussed in detail how data is *shared* between our main application and the functions it calls

Passing Data to a Function

- So far we have not discussed in detail how data is *shared* between our main application and the functions it calls
- There are three options in total

Passing Data to a Function

- So far we have not discussed in detail how data is *shared* between our main application and the functions it calls
- There are three options in total

  Pass by Value Copy the data to the function

Passing Data to a Function

- So far we have not discussed in detail how data is *shared* between our main application and the functions it calls
- There are three options in total

  Pass by Value Copy the data to the function

  Pass by Reference Provide a *reference* to the value to the function

Passing Data to a Function

- So far we have not discussed in detail how data is *shared* between our main application and the functions it calls
- There are three options in total

  Pass by Value Copy the data to the function

  Pass by Reference Provide a *reference* to the value to the function

  Pass by Pointer Provide the function with the memory location that the value is stored in

Pass by Value

- Creates a copy of the value within the function

Pass by Value

- Creates a copy of the value within the function
  - This means we have the original value and a local copy in memory (so two instances of the value)

Pass by Value

- Creates a copy of the value within the function
  - This means we have the original value and a local copy in memory (so two instances of the value)
- Original value remains the same in the calling function

Pass by Value

- Creates a copy of the value within the function
  - This means we have the original value and a local copy in memory (so two instances of the value)
- Original value remains the same in the calling function
- Local copy within the function is now a local variable

Pass by Value

- Creates a copy of the value within the function
  - This means we have the original value and a local copy in memory (so two instances of the value)
- Original value remains the same in the calling function
- Local copy within the function is now a local variable
  - We can alter it without affecting the original

Pass by Value

- Creates a copy of the value within the function
  - This means we have the original value and a local copy in memory (so two instances of the value)
- Original value remains the same in the calling function
- Local copy within the function is now a local variable
  - We can alter it without affecting the original
  - When the function finishes the local copy is lost

Pass by Value and Scope

- Scope of a called function is different to the calling function

Pass by Value and Scope

- Scope of a called function is different to the calling function
- Local variable within a function are scoped to that function

Pass by Value and Scope

- Scope of a called function is different to the calling function
- Local variable within a function are scoped to that function
- When the function ends, the local variables (copies) are discarded

Pass by Value and Scope

- Scope of a called function is different to the calling function
- Local variable within a function are scoped to that function
- When the function ends, the local variables (copies) are discarded
- So - **any changes to the pass by value variable are not reflected in the calling function!**

Example

Pass by Value

```
#include <iostream>

void foo(int x)
{
    std::cout << "Start of function, x = " << x << std::endl;
    x = 20;
    std::cout << "End of function, x = " << x << std::endl;
}

int main(int argc, char **argv)
{
    int x = 10;
    std::cout << "Before function call, x = " << x << std::endl;
    foo(x);
    std::cout << "After function call, x = " << x << std:endl;
    return 0;
}
```

Peer Learning Exercise

- Get into pairs (person next to you)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by value) to your partner for 1-2 minutes

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by value) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by value) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by value) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening
- Swap partners (person on other side of you) and repeat the exercise

Questions?

Outline

References

- A simple C++ data type (we use & to define)

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

References

- A simple C++ data type (we use `&` to define)

```
    int &x   x is a reference to an int
  float &y   y is a reference to a float
student &s   s is a reference to a student
```

References

- A simple C++ data type (we use `&` to define)

```
    int &x   x is a reference to an int
  float &y   y is a reference to a float
student &s   s is a reference to a student
```

- Less powerful (but safer) than pointers

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

References

- A simple C++ data type (we use `&` to define)

  `int &x` x is a reference to an `int`

  `float &y` y is a reference to a `float`

  `student &s` s is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

  `int &rA = A;` reference to value created

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

  `int &rA = A;` reference to value created

- References are fixed, and *cannot* be reallocated or *reseated* to another variable

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

  `int &rA = A;` reference to value created

- References are fixed, and *cannot* be reallocated or *reseated* to another variable
  - `rA` will always reference `A` above

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

  `int &rA = A;` reference to value created

- References are fixed, and *cannot* be reallocated or *reseated* to another variable
  - `rA` will always reference `A` above
  - Changing the value of `rA` will change the value of `A`

References

- A simple C++ data type (we use `&` to define)

  `int &x` `x` is a reference to an `int`

  `float &y` `y` is a reference to a `float`

  `student &s` `s` is a reference to a `student`

- Less powerful (but safer) than pointers
- Reference always points to a value (no null reference) - *cannot be initialised / created in combination with creation of an object*

  `int A = 5;` value created

  `int &rA = A;` reference to value created

- References are fixed, and *cannot* be reallocated or *reseated* to another variable
  - `rA` will always reference `A` above
  - Changing the value of `rA` will change the value of `A`
  - And vice versa

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*
- Many languages (e.g. Java, Python, C#) use pass by reference as default (except for primitive types) - so you might already be used to working with them

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*
- Many languages (e.g. Java, Python, C#) use pass by reference as default (except for primitive types) - so you might already be used to working with them
  - Many languages get round memory management problems with garbage collection - Java, C#, Python

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*
- Many languages (e.g. Java, Python, C#) use pass by reference as default (except for primitive types) - so you might already be used to working with them
  - Many languages get round memory management problems with garbage collection - Java, C#, Python
- `const` keyword used to define constant values (variables or parameters)

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*
- Many languages (e.g. Java, Python, C#) use pass by reference as default (except for primitive types) - so you might already be used to working with them
  - Many languages get round memory management problems with garbage collection - Java, C#, Python
- `const` keyword used to define constant values (variables or parameters)
  - Can be used with references

Pass by Reference

- When calling a function, we pass a reference to the value - **not a copy**
  - e.g. we use & to specify the type of the incoming parameter
  - The function will have a **reference** to the variable rather than a copy of the variable
- Called function accesses the same value as the calling function - *when called function ends the original value (in the calling scope) has been updated*
- Many languages (e.g. Java, Python, C#) use pass by reference as default (except for primitive types) - so you might already be used to working with them
  - Many languages get round memory management problems with garbage collection - Java, C#, Python
- `const` keyword used to define constant values (variables or parameters)
  - Can be used with references
  - This means that the value stored in the reference cannot be changed

Pass by Reference

```
#include <iostream>

void foo(int &x)
{
    std::cout << "Start of function, x = " << x << std::endl
        ;
    x = 20;
    std::cout << "End of function, x = " << x << std::endl;
}

int main(int argc, char **argv)
{
    int x = 10;
    std::cout << "Before function call, x = " << x << std::
        endl;
    foo(x);
    std::cout << "After function call, x = " << x << std::
        endl;
    return 0;
}
```

Peer Learning Exercise

- Get into pairs (person next to you)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by reference) to your partner for 1-2 minutes

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by reference) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by reference) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by reference) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening
- Swap partners (person on other side of you) and repeat the exercise

Questions?

Outline

Pointers

- Considered difficult

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with
- Most problems come from overlapping meaning of certain symbols

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with
- Most problems come from overlapping meaning of certain symbols
- Also, pointers are more versatile than references

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with
- Most problems come from overlapping meaning of certain symbols
- Also, pointers are more versatile than references
  - This does mean they are more volatile and prone to error

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with
- Most problems come from overlapping meaning of certain symbols
- Also, pointers are more versatile than references
  - This does mean they are more volatile and prone to error
  - More opportunities to introduce bugs

Pointers

- Considered difficult
  - Normally the part that most new C and C++ programmers struggle with
- Most problems come from overlapping meaning of certain symbols
- Also, pointers are more versatile than references
  - This does mean they are more volatile and prone to error
  - More opportunities to introduce bugs
- They are very powerful and useful when you get the hang of them

Pointers

- A pointer is just a location in memory that we treat like a particular type

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

  `int *x` `x` is a pointer to an `int`

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

```
int *x
```
`x` is a pointer to an `int`

```
float *y
```
`y` is a pointer to a `float`

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

```
   int *x  x is a pointer to an int
 float *y  y is a pointer to a float
student *s  s is a pointer to a student
```

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

```
   int *x  x is a pointer to an int
 float *y  y is a pointer to a float
student *s  s is a pointer to a student
```

- We use & to get the *address of* a value (i.e. its location in memory)

Pointers

- A pointer is just a location in memory that we treat like a particular type
  - This is the first important issue - a pointer is a location in memory that *we consider to store a particular type of data*
- Similar to references - also point a particular value in memory
  - The second important issue - a pointer *can change the location it is pointing to*. A reference cannot
- Because of this pointers can be reassigned as required
- Pointer types are defined using an *

```
   int *x  x is a pointer to an int
 float *y  y is a pointer to a float
student *s  s is a pointer to a student
```

- We use & to get the *address of* a value (i.e. its location in memory)
- We use * to get the *value of* the pointer (i.e. the value pointed to)

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.

	A	B	C	D
1				
2				
3				
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets

	A	B	C	D
1				
2				
3				
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this

	A	B	C	D
1				
2				
3				
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this
- In a spreadsheet we store values in cells

	A	B	C	D
1				
2				
3				
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this
- In a spreadsheet we store values in cells
- Each cell may contain a value

	A	B	C	D
1	5			
2	10			
3	8			
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this
- In a spreadsheet we store values in cells
- Each cell may contain a value
- A cell may also contain a formula *relating to other cell addresses (like pointers)*

	A	B	C	D
1	5			
2	10			
3	8			
4				

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this
- In a spreadsheet we store values in cells
- Each cell may contain a value
- A cell may also contain a formula *relating to other cell addresses (like pointers)*
  - e.g. D4 = (A1 * A2) - A3

	A	B	C	D
1	5			
2	10			
3	8			
4				42

OK, let's try and make it simpler...

- Understanding pointers can really twist the brain until you understand.
- There are a number of metaphors for understanding pointers. We will use one involving spreadsheets
  - If you did the programmer's aptitude test you will hopefully understand this
- In a spreadsheet we store values in cells
- Each cell may contain a value
- A cell may also contain a formula *relating to other cell addresses (like pointers)*
  - e.g. D4 = (A1 * A2) - A3
- In a spreadsheet the cell ID is just an address. In C, we can point to a cell in memory to access its value (it's just a little bit more obtuse than a spreadsheet)

	A	B	C	D
1	5			
2	10			
3	8			
4				42

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

  `int v = 10;` declaration of value

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

  `int v = 10;` declaration of value

  `foo(&v);` pass in the *address of* `v` to `foo`

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

  `int v = 10;` declaration of value

  `foo(&v);` pass in the *address of* `v` to `foo`
- Within the function we need to dereference the pointer to get access to it

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

  `int v = 10;` declaration of value

  `foo(&v);` pass in the *address of* `v` to `foo`
- Within the function we need to dereference the pointer to get access to it

  `x` within `foo` this is a pointer - *the address of* `v`

Pass by Pointer

- To use pass by pointer we first have to declare a parameter as a pointer
  - `void foo(int *x)`
- If we want to pass in the variable we have to get its *address* (the pointer)

  `int v = 10;` declaration of value

  `foo(&v);` pass in the *address of* `v` to `foo`
- Within the function we need to dereference the pointer to get access to it

  `x` within `foo` this is a pointer - *the address of* `v`

  `*x` allows access to the value stored in `x` - *the value of* `v`

Pass by Pointer

```
#include <iostream>
// Assume starting address of x = 003FFC30
void foo(int *x)
{
    std::cout << "Address of x in function = " << x << std::endl;
    std::cout << "Start of function, x = " << *x << std::endl;
    *x = 20;
    std::cout << "End of function, x = " << *x << std::endl;
    std::cout << "Address of x at end of function = " << x << std::endl;
}

int main(int argc, char **argv)
{
    int x = 10;
    std::cout << "Starting addres of x = " << &x << std::endl;
    std::cout << "Before function call, x = " << x << std::endl;
    foo(&x);
    std::cout << "After function call, x = " << x << std::endl;
    std::cout << "End address of x = " << &x << std::endl;
    return 0;
}
```

Peer Learning Exercise

- Get into pairs (person next to you)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by pointer) to your partner for 1-2 minutes

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by pointer) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by pointer) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening

Peer Learning Exercise

- Get into pairs (person next to you)
- Speaker - explain current topic (pass by pointer) to your partner for 1-2 minutes
- Listener - listen to explanation. Question anything you don't understand (speaker try to address these questions)
- Swap roles - repeat speaking and listening
- Swap partners (person on other side of you) and repeat the exercise

Questions?

Outline

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))

- Returning from a function is another important area of consideration

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors
  - Advantage of pass-by-reference is that multiple values can be *"returned"*

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors
  - Advantage of pass-by-reference is that multiple values can be *"returned"*
    - Any parameter marked as a reference can be changed

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors
  - Advantage of pass-by-reference is that multiple values can be *"returned"*
    - Any parameter marked as a reference can be changed
- A couple of gotchas

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors
  - Advantage of pass-by-reference is that multiple values can be *"returned"*
    - Any parameter marked as a reference can be changed
- A couple of gotchas
  1. Pass-by-reference modifies values outside the local scope - *potentially troublesome when calling 3rd party code (e.g. a library)*

Pass by Reference vs. Pass by Value

- Despite the complications you might see with pass-by-reference it is the recommended approach when working with C++
- Data copying and memory allocation is an expensive operation

  pass by value — have to allocate and copy the size of the data (possibly millions of bytes)

  pass by reference — copy the address to the value (4 bytes (32 bit), 8 bytes (64 bit))
- Returning from a function is another important area of consideration
  - Returned value is copied. However, referencing a value that goes out of scope leads to errors
  - Advantage of pass-by-reference is that multiple values can be *"returned"*
    - Any parameter marked as a reference can be changed
- A couple of gotchas
  1. Pass-by-reference modifies values outside the local scope - *potentially troublesome when calling 3rd party code (e.g. a library)*
  2. Pass-by-reference enables multiple parties to have access to the same value. This can cause numerous problems with control of a value, which leads to increased complexity (potentially extra code)

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said.
"The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about
  - A reference to the song's name

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about
  - A reference to the song's name
  - The song's name

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said.
"The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about
  - A reference to the song's name
  - The song's name
  - A reference to the song

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said.
"The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about
  - A reference to the song's name
  - The song's name
  - A reference to the song
  - The song

Example

"You are sad," the Knight said in an anxious tome: "Let me sing you a song to comfort you."
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.
"It's long," said the Knight, "but it's very, very beautiful. Everybody that hears me sing it - either it brings the tears to their eyes, or else - "
"Or else what?" said Alice, for the Knight had made a sudden pause.
"Or else it doesn't, you know. The name of the song is called 'Haddock's Eyes.' "
"Oh, that's the name of the song it it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting on a Gate': and the tune's my own invention."

- Distinguishing between the song, what it is called, its name, and what the name is called
  - Song's name is called Haddock's Eyes
  - Song's name is The Aged Aged Man
  - Song is called Ways and Means
  - The song is A-sitting on a Gate
- To think of this in terms that we have just spoken about
  - A reference to the song's name
  - The song's name
  - A reference to the song
  - The song
- A name is another method of referencing the song (like your matriculation number is a way of referencing you). So we have a reference to a reference here!

Questions?

Outline

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*
- We have covered three key concepts

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*
- We have covered three key concepts
  - Passing a value (i.e. copying data)

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*
- We have covered three key concepts
  - Passing a value (i.e. copying data)
  - Passing a reference (i.e. telling where the value is)

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*
- We have covered three key concepts
  - Passing a value (i.e. copying data)
  - Passing a reference (i.e. telling where the value is)
  - Passing by pointer (i.e. copying the memory location)

Summary

- A lot of new ideas today, and a lot to digest and understand. *It will take time so put in the effort - it doesn't come naturally to most people*
- We have covered three key concepts
  - Passing a value (i.e. copying data)
  - Passing a reference (i.e. telling where the value is)
  - Passing by pointer (i.e. copying the memory location)
- The workbook has far more that you can explore around, and looks at some other ideas when calling a function

To do...

1. **Remember** - do the end of unit exercises. Don't ignore these. They help understanding.

To do...

1. **Remember** - do the end of unit exercises. Don't ignore these. They help understanding.
2. Have you started work on the coursework? You should have. If you haven't do so today. If your plan is to leave it any later please come and speak to me so I can explain why you need to start this now.

To do...

1. **Remember** - do the end of unit exercises. Don't ignore these. They help understanding.
2. Have you started work on the coursework? You should have. If you haven't do so today. If your plan is to leave it any later please come and speak to me so I can explain why you need to start this now.
3. Workbook - applied examples of working with references and pointers.

To do...

1. **Remember** - do the end of unit exercises. Don't ignore these. They help understanding.
2. Have you started work on the coursework? You should have. If you haven't do so today. If your plan is to leave it any later please come and speak to me so I can explain why you need to start this now.
3. Workbook - applied examples of working with references and pointers.
4. Tutorial

To do...

1. **Remember** - do the end of unit exercises. Don't ignore these. They help understanding.
2. Have you started work on the coursework? You should have. If you haven't do so today. If your plan is to leave it any later please come and speak to me so I can explain why you need to start this now.
3. Workbook - applied examples of working with references and pointers.
4. Tutorial
5. Next lecture - will cover memory management. This is the part of the module which will take some thought to understand, so ensure you spend the necessary time on it.