Highlights

- const & passing-by-reference

```c
void foo(person a, const person& b);
```

- pointers

```c
int x = 6;
int* xp;
xp = &x;
```

- new and delete

```c
int *xp;
xp = new int;
*xp = 5;
delete xp;
```
An object is simply a box in memory and if you pass this into a function it makes a copy.

A memory address is where a box is located and if you pass this into a function, you can change the variable everywhere.

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Object (box)</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrays</td>
<td>int, double, char, ...</td>
</tr>
<tr>
<td>using &amp;</td>
<td>classes</td>
</tr>
</tbody>
</table>
What is the difference between these two?

```c
int sum(int x, int y);
int sum(const int &x, const int &y);
```
What is the difference between these two?

```c
int sum(int x, int y);
int sum(const int &x, const int &y);
```

First one copies the values into x and y, thus these values exist in multiple places.

The second creates a link but does not let you modify the original (see: callByValue.cpp)
const call-by-reference

Classes can be rather big, so in this case using const and '&' can save memory

So a better way to write:

```cpp
bool equals(Point first, Point second)
```

... would be: (function definition the same)

```cpp
bool equals(const Point & first, const Point & second)
```

In fact, without & creates a copy, which is a new object and thus runs a constructor
typedef

Side note: If you want to rename types, you can do that with `typedef` command:

```c
typedef int DefinatelyNotAnInt;
DefinatelyNotAnInt x;
x=3;
int y = x;
cout << y;
```

If you have always been bothered that we use “double” instead of “real”, go ahead and fix it!
Review: address vs value

Consider the following:

```cpp
int x=6;
cout << x << "\n";
cout << &x << endl;
```

x is a variable (a box containing value 6)

&x is a memory address (sign pointing to box)
- Rather than giving the value inside the box, this gives the whole box
Review: address vs value

Similar to a URL and a webpage
-A URL is not a webpage, but a link to one

Webpage g;
cout << &g;
Pointers

Just as `&` goes from value (webpage) to address (url), `*` goes the opposite:

Webpage `g`;
URL `u = &g`;
Webpage `g2 = *u`;
Pointers

You can also think of pointers as “phone numbers” and what they point to as “people”
Pointers

If multiple people have the same “phone number”, they call the same person (object)

1-800-presdnt (pointer/memory address)

Obama (object)

1-800-presdnt
A pointer is used to store a memory address and denoted by a * (star!)

```cpp
int x = 6;
int* xp;
xp = &x;
```

Here variable xp is a integer pointer

```cpp
cout << *(&x); // *(&x) same as x
```

The * goes from address to variable (much like when you hit ENTER on a url)
(See: pointerBasics.cpp)
Pointers

It is useful to think of pointers as types:

```c
int* xp;
```

Here I declared a variable “xp” of type “int*”

Just like arrays and [], the use of the * is different for the declaration than elsewhere:

Declaration: the * is part of the type

```c
int* xp;
```

Everywhere else: * follows the pointer/address

(i.e. `*xp = 2;` puts 2 where xp is pointing to)
Pointers and references allow you to change anything into a memory address that you want. This can make it easier to share variables across functions.

You can also return a pointer from a function (return links to variables) (see: returnPointer.cpp)
Pointers

Why do we need pointers? (memory addresses are stupid!!!)

Suppose we had the following class:

```cpp
class Person{
    string name;
    Person mother;
    Person father;
};
```

Will this work?
Pointers

As is, it will not... it is impossible to make a box enclose two other equal sized boxes.

The only way it can enclose something like itself is that thing is smaller.
Pointers

To do this we can use pointers instead!

A pointer does not store the whole class data, it only remembers where it is (like a URL)

```cpp
class person{
    string name;
    person* mother;
    person* father;
};
```

(See: person.cpp) (more on this shortly)
When dealing with classes, often you need to deference (*) and access a member (.).

There is a shortcut to de-reference and call a member (follow arrow and go inside a box)

You can replace (*var).x with var->x, so...

```cpp
(*me.mother).name;
```

... same as ...

```cpp
me.mother->name;
```
Person class

How would you make your grandmother? How could you get your grandmother using only yourself as a named object?

```cpp
class person{
    string name;
    person* mother;
    person* father;
};
```

(See: personV2.cpp)
Pointers and memory

```c
#include <stdio.h>
static char *ptr = "okay.
int main(int

THE SEGFALUT
IN OUR
CHAR STARS

ARG C.CH AR**A WHILE PRINTF("%C
RGV) { (PTR) { "N", *PTR++};
```
Boxes

What is comes next in this pattern?

Basic programming: \texttt{int x;}
Ask for one box with a name

Intermediate programming: \texttt{int x[20];}
Ask for multiple boxes with one name

Advanced programming: ???
???
Boxes

What is comes next in this pattern?

Basic programming: int x;
Ask for one box with a name

Intermediate programming: int x[20];
Ask for multiple boxes with one name

Advanced programming: new int;
Ask for a box without giving it a name
Pointers are also especially useful to use with the `new` command.

The `new` command will create a variable (box) of the type you want. Here is an example:

```cpp
int *xp;
xp = new int;
*xp = 4;
```

The new integer has no separate name, just part of `xp` (as array boxes part of array name). (See: `newMemory.cpp`)
What does this do?

```c
int main()
{
    while(true)
    {
        int *x = new int;
    }
    return 0; //totally going to get here!
}
```
What does this do?

```cpp
int main()
{
    while(true)
    {
        int *x = new int;
    }
    return 0; //totally going to get here!
}
```

Asking for a lot of boxes there...  
(See: memoryLeak.cpp)
When your program exits, the operating system will clean up your memory.

If you want to clean up your memory while the program is running, use `delete` command.

```cpp
int *imaPointer; // pointer box (holds address)
imaPointer = new int; // point here!
// do some stuff...
delete imaPointer; // goodbye pointer
```

(See: deleteMemory.cpp)
This is also a memory leak:

```cpp
int *ptr; // make a pointer
ptr = new int; // point here
ptr = new int; // more the merrier
delete ptr; // ERASE
```

By the 3rd line, there is no link back to the box on the 2nd line (dangling pointer)

There should be a “delete” for every “new”
As you can manage how you want to create new variables/boxes, using new/delete is called **dynamic memory**

Before, the computer took care of memory by creating variables/boxes when you use a type then deleting when the function ends.
Memory management is a hard part of C++

You need to ensure you delete all your boxes after you are done with them, but before the pointer falls out of scope (see: lostPointer.cpp)

Some other languages manage memory for you
Person class

The ability to have non-named boxes allows you to more easily initialize pointers

```cpp
class person{
    string name;
    person* mother;
    person* father;
};
```

(See: personV3.cpp)