```cpp
//////// Lecture 81: Make Functions

#include <cstdlib>
#include <iostream>
#include <memory>
// Modern C++ discourages manual memory management. So we should avoid using new and delete operators directly in our code, but if we want to create a smart pointer, then we have to use the new operator to allocate the memory on the heap. This can be avoided, the smart pointers provide global functions through which you can construct a smart pointer without having manually alloctate memory for the underlying resource.

// those global functions are MAKE FUNCTIONS, and are provided for both unique_ptr and shared_ptr.

int main()
{
    auto p = std::make_unique<int>(5)};
    //Make unique will do the following:
    /*
    1. Create an integer on the heap
    2. Initialise it with a value 5
    3. Store the address of this integer inside a unique_ptr
    4. Return the ptr.
    */
    // In this way, you don't have to use new operator in your code.
}
// make_unique() is a VARIADIC FUNCTION TEMPLATE, that means it can accept an arbitrary type and number of arguments. If the type we want to construct has a constructor that accepts multiple arguments, you can pass those arguments here.
// For example, we can create a class called Point.

class Point{
public:
    Point(int x, int y){}
```

```
}
int main(){
    auto pt = std::make_unique<Point>(1,2);
}

// You may even use make_unique to create a unique_ptr for dynamic arrays.
int main(){
    auto array = std::make_unique<int[]>(5); // the argument represents the SIZE of the dynamic array.
    // You are not allowed to initialise the dynamic array using make_unique(), but you can always initialise it using the subscript.
    // But from C++17, we can initialise it after declaration using the subscript operator.
    array[0] = 1; // this works!
}

////// All this will work for shared_ptr.
// To construct a shared_ptr, use 'make_shared'.
// make_shared for arrays was introduced in the C++20 standard.

//
// shared_ptr is implemented differently compared to a unique_ptr, and it has to store additional unique information related to the underlying pointer in its control block.
// If you create a shared_ptr yourself, then you will have to use new to allocate the memory for the underlying resource, and then, the shared_ptr will again use new to allocate memory for the control block, but make_shared has knowledge of the internals of the shared_ptr. So when we use it, it will allocate memory for the control block and the underlying resource using one new call.
// During destruction, there will be only 1 delete call to delete both the underlying resource and the control block.
// So if you create a shared_ptr using make_shared,
you will save multiple calls to the new and delete constructors.
// Note that this is not applicable to make_unique because unique_ptr does not store any other information except the pointer, so it will make no difference to the number of calls to new and delete whether you construct it yourself or you use make_unique.

// There is 1 disadvantage when you use make() functions - there is NO WAY TO SPECIFY CUSTOM DELETERS, while using make() functions.
// If you would like to use a custom deleter, you have to constructor the smart ptr and allocate the memory for that resource yourself.
// but if you're not using a custom deleter, then it is recommended to use the make functions.

```