

# Unique Pointer Solutions

- Briefly describe how the `unique_ptr` type is implemented
  - `unique_ptr` is a class which has a traditional pointer as a private data member
  - It has public member function which implement some of the features of traditional pointers
- In terms of memory usage and efficiency, how does a `unique_ptr` instance compare to a traditional pointer?
  - There is little or no extra overhead from using a `unique_ptr` instead of a traditional pointer

- Explain how `unique_ptr` follows the principles of RAII
  - The traditional pointer member is managed by the class
  - It is allocated in the class's constructor
  - It is released in the class's destructor
  - The class manages transfer of ownership of the traditional pointer from one object to another
  - The programmer does not need to write any code which deals directly with the pointer member

- Give some examples of traditional pointer operations that are supported by `unique_ptr`
  - Dereferencing, boolean and ! operators
- Give an example of a traditional pointer operation that is not supported by `unique_ptr`
  - Copying, assignment, pointer arithmetic

- Write a simple program that creates and initializes an instance of `unique_ptr` and performs some operations on it
- What changes would you need to make your program compile under C++11?
- (Optional) Put your compiler into C++11 mode and check your answer to the previous question

- Explain what is meant by transfer of ownership in the context of `unique_ptr`
  - When ownership is transferred from `unique_ptr` instance A to `unique_ptr` instance B:
  - `delete` is called on B's pointer member
  - B's pointer member is assigned to A's pointer member
  - A's pointer member is set to `nullptr`
  - In effect, the pointer member has moved from A to B

- Why is transfer of ownership useful when returning a `unique_ptr` from a function?
  - The function creates a `unique_ptr` instance as a local variable
  - This will internally allocate memory for its pointer member
  - When the function returns, the pointer member will be transferred from the local variable into a variable in the caller
  - When the caller's variable is destroyed, the memory allocated will be released automatically
  - This allows large objects to be returned efficiently without creating potential memory management issues

- Write a function which creates a `unique_ptr` local variable which is returned by the function
- Write a program that calls the function and prints out the data in the returned `unique_ptr`