

# Optional Type Solutions

# std::optional

- Briefly describe std::optional
  - std::optional is a templated type defined in <optional>
  - An std::optional object either contains a single value of the template parameter type, or is empty

# Accessing data in `std::optional`

- Write a simple program which
  - Creates an empty `std::optional` object
  - Stores some data in the object
  - Displays the value of the data

# Optional Return Value with `std::optional`

- Alter the `str2int()` function so that it now returns `std::optional`
- Make the necessary changes to `main()`
- Check that your program compiles and runs correctly
- Make sure that errors are handled correctly
- Use both forms of syntax for checking and accessing an `std::optional` object

# Applications of `std::optional`

- Describe some situations where `std::optional` could be useful
  - Representing a type which can have a NULL value
  - Returning an result when failure is not an error
  - Mathematical calculations which do not always have a solution
  - Lazy initialization
  - Optional arguments to functions
  - Caching

# Disadvantages of `std::optional`

- Give some disadvantages of `std::optional`
  - `std::optional` is implemented using a `bool` internally
  - It uses more memory than a scalar variable
  - Returning `std::optional` can indicate "no data"
  - Cannot provide information about the lack of data