Conditional Compilation

For source code control, you can `#define` a symbol and not give it a value. The preprocessor supports **conditional compilation**, using **preprocessor directives** to conditionally include a section of code based on which **`#defined`** symbols it has "seen":

```
#if defined(A)
    cout << "A is defined." << endl;
#elif defined(B)
    cout << "B is defined." << endl;
#else
    cout << "Neither A or B is defined." << endl;
#endif
```

This code is processed **before the rest of the code is sent to the compiler;** these conditions can **only** refer to **`#defined`** constants, integer values, and arithmetic and logical expressions using those values.

Here we've used the predefined preprocessor function **`defined()`** to check if a constant has been previously been **`#defined`**. You can also use these "shorthand" expressions.

- **`#ifdef`** is short for **if defined**; **`#ifdef symbol`** is the same as **`#if defined(symbol)`**.
- **`#ifndef`** is short for **if not defined**, the opposite of **`#ifdef`**.

Conditional compilation determines whether pieces of code are sent to the compiler. When the preprocessor encounters this, whichever conditional expression is true will have its corresponding code block included in the final program. If your code previously encountered **`#define A`**, then this entire portion of code will go to the compiler as:

```
cout << "A is defined." << endl;
```

The rest of the code will simply be discarded.