CSE 1320 - Intermediate Programming

Macros

Alex Dillhoff

University of Texas at Arlington

Macros

C supports more complicated definitions using `#define` in the form of functions.

These can be called just like functions, but are processed like a preprocessor directive.

Macros

Since macros are created using *#define*, every occurrence of the macro in code is replaced with the definition during compilation.

Macros

The macro code is expanded at each location that it is referenced during the preprocessing phase of compilation.

This provides a performance benefit over a traditional function, which must transfer control to a different part of the object code when called.

Macros

A subtle tradeoff is that macros expand the size of the code.

Excessive usage can create binaries that are bloated compared to using functions.

Macros - Example

A commonly used case is to create min and max macros to produce the minimum or maximum of two values.

```
#define MIN(a, b) (a < b) ? a : b
#define MAX(a, b) (a > b) ? a : b
```

Macros - CAUTION

Be careful when writing functional macros as parameters will expand exactly as you define it.

Example: `abs.c`

Macros - CAUTION

In the previous example, the certain ways of writing the absolute value macro would produce erroneous output.

Macros - CAUTION

Consider the following macro:

```
#define ABS(x) x < 0 ? -x : x
```

If the input is something like `5 - 10`, the resulting expansion will be:

```
ABS(5 - 10) 5 - 10 < 0 ? -5 - 10 : 5 - 10
```

Macros - CAUTION

This evaluates to

```
-5 < 0 ? -15 : -5
```

The resulting output is then -15.

Macros - CAUTION

What's the right way to create such a macro?

```
int ABS_x;
#define ABS(x) (ABS_x = x, ABS_x < 0 ? -ABS_x : ABS_x)
```

Although `int ABS_x;` is declared globally, it is not used in the main program.

Conditional Directives

Conditional directives have already been used when creating a header guard, but there are a few more worth noting:

- *#if*
- *#ifdef*
- *#elif*
- *#else*

Conditional Directives

We can combine these with macros to add debugging or logging statements in our code that only execute under certain builds.

Conditional Directives

For example, we may want leave certain debug statements in the code, but only use them if we build the debugging version of our code.

Conditional Directives

Example: `debug_macro.c`

Conditional Directives

The previous example will only execute the statements if the `DEBUG` macro is defined.

We can pass macros and define them as part of the compilation command.

```
gcc -DDEBUG debug_macro.c
```