Complex Macros

A complex macro accepts arguments and generates a fragment of code using the values of those arguments. Macros that accept arguments appear to be functions. However, arguments are not typed as in a C function. They are merely replaced by the text passed to the macro when expanded.

Macros with arguments must be defined using the **#define** directive before they can be used. The argument list is enclosed in parentheses and must immediately follow the macro name. Spaces are not allowed between and macro name and open parenthesis. For example:

```
#define MAX(x,y) ((x) > (y) ? (x) : (y))
```

defines a macro named **MAX** that takes two arguments (x and y). When **MAX** is used in your program (or in preprocessor directives) it is replaced with the text **((x) > (y) ? (x) : (y))**. If x and y are numeric constants, the preprocessor can determine the result of the macro and substitute the greater value.

A C statement like

```
int a = MAX(15,20);
```

is expanded by the preprocessor into

```
int a = 20;
```

While a C statement like

```
int a = MAX(myvar,20);
```

is expanded by the preprocessor into

```
int a = ((myvar) > (20) ? (myvar) : (20));
```

- The number of arguments passed to a macro must match the number of arguments specified in the macro definition.
- It is common practice to surround arguments used in a macro definition with parentheses. This is done so that compound expressions, when passed to a macro, do not cause unwanted side-effects. For example:

  ```
  #define MAX(x,y) ((x) > (y) ? (x) : (y))

  int a = MAX(x-5,10);
  ```

  expands as

  ```
  int a = ((x-5) > (10) ? (x-5) : (10));
  ```

  Without the additional parentheses,

  ```
  #define MAX(x,y) x > y ? x : y

  int a = MAX(x-5,10);
  ```

  expands as

  ```
  int a = x-5 > 10 ? x-5 : 10;
  ```

with a potentially different meaning.

- Macros that use arguments more than once can introduce undesired side-effects into your program. For example:

```
#define MAX(x,y) ((x) > (y) ? (x) : (y))

maxval = MAX(a+b, func(c));
```

expands as

```
maxval = ((a+b)> (func(c)) ? (a+b) : (func(c)));
```

The function func appears to be invoked only once in the program, but because of the macro definition, it is actually called twice. Each call may return a different value and the result of MAX may be incorrect.

- Macros may be defined with a null or empty argument list. For example:

```
#define MYMACRO() (func();)
```

To call such a macro, you must specify the macro name along with an empty argument list. For example:

```
MYMACRO()
```

- To pass an empty argument to a macro, you must include at least one whitespace character in the place of that argument.