

Macros

In C, a macro is a fragment of code which has been given a name.

Whenever the name is used, it is replaced by the contents of the macro

We have already seen the use of a macro to define constants:

```
#define PI 3.14
```

But with the constant qualifier `const` (from C90 and onward) it's better to use it instead:

```
const double pi = 3.14;
```

There are two types of macros:

Object-like macros

Function-like macros

Object-like macros

Look like a data object in the code where the macro name is replaced

For example with the following macro:

```
#define PI 3.14159
```

The code:

```
printf("pi: %1.15f \n", PI);
```

Is replaced with:

```
printf("pi: %1.15f \n", 3.14159)
```

Function-like macros

Denote macros that look like function calls:

```
#define CELSIUS_TO_FAHRENHEIT( temperature ) \  
    ( ( temperature ) * 1.8 + 32 )
```

The code:

```
fahrenheit = CELSIUS_TO_FAHRENHEIT(celsius);
```

Is replaced with:

```
fahrenheit = ( ( celsius ) * 1.8 + 32 );
```

#undef

We can get rid of macros:

```
#define MY_INT 10
```

```
#undef MY_INT
```

or redefine them later in code:

```
#define MY_INT 10
```

Check whether a macro is defined or not

#ifdef MACRONAME

Or its opposite:

#ifndef MACRONAME

For example:

#ifdef MACRO_1

Returns 1 if MACRO_1 is defined at this spot in the code, otherwise 0

Used in conjunction with `#else`, `#elif` and `#endif`

```
#ifndef HAPPY  
    #define COLOR Yellow  
#elif BORED  
    #define COLOR Gray  
#else  
    #define COLOR Blue  
#endif
```

Useful in various ways

```
#ifndef PI
    #define PI M_PI
    #include <math.h>
#endif
```

```
printf("The value of pi: %1.15f \n",
    PI);
```

Useful for debugging code

```
#include <stdio.h>
#define DEBUG

int i = 0; int arr [10];
int main () {
    for (i=0;i<=10;i++) {
        arr[i]=0;
#ifdef DEBUG
        if (i>9)
            printf("Index variable out of bounds!!
                    Value of i: %d\n",i);
#endif
    }
}
```

Useful for debugging code

```
#include <stdio.h>

int i = 0; int arr [10];
int main () {
    for (i=0;i<=10;i++) {
        arr[i]=0;
#ifdef DEBUG
        if (i>9)
            printf("Index variable out of bounds!!
                    Value of i: %d\n",i);
#endif
    }
}
```

```
gcc -DDEBUG thisfile.c -o thisfile
```

Often used for conditional compiling

```
/* This administrivia gets added to the beginning
   of limits.h if the system has its own version of
   limits.h.  */

/* We use _GCC_LIMITS_H_ because we want this not
   to match any macros that the system's limits.h
   uses for its own purposes.  */

#ifndef _GCC_LIMITS_H_
#define _GCC_LIMITS_H_

#ifndef _LIBC_LIMITS_H_

#include "syslimits.h"
```

Often used for conditional compiling

```
#if DEVICE == IBM
    #include ibmdrv.h
#elif DEVICE == HP
    #include hpdrv.h
#else
    #include gendrv.h
#endif
```

Only constant expressions are allowed in conditional compilation directives. Therefore, in the above code, DEVICE, IBM, and HP must be defined constants.

Useful links

From the Gnu cpp documentation:

<https://gcc.gnu.org/onlinedocs/cpp/Macros.html#Macros>

From Wikipedia

https://en.wikipedia.org/wiki/C_preprocessor