UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Macro Reemplazos en C

Muchos lenguajes de programación (sobre todos los más antiguos) incorporan el concepto de macro.

Antes del desarrollo de los lenguajes de programación de alto nivel y de las técnicas de compilación, los programadores de ensamblador debían escribir bloques de código repetitivos en los que cambiaban unos pocos elementos. Para facilitar esta tarea, muchos ensambladores proporcionaban herramientas sofisticadas para tratar estos bloques utilizando expansión de macros. En lugar del código repetitivo, el programador escribía la macro. Después el ensamblador se encargaba de realizar la expansión y sustituir la macro por el código.

De forma muy genérica, **una macro consiste en una plantilla o meta-expresión que define un patrón de sustitución**. El patrón define unas variables libres y unas expresiones textuales. Cuando usamos la macro, le damos un valor a las variables libres. La macro se expande, aplicando el patrón a los variables y generando código en el lenguaje de programación de la macro.

Una característica fundamental de la macro es que se trata de una técnica de generación de código. En el caso en que el lenguaje sea compilado (como C o C++) la expansión de la macro se realiza en la fase de **preprocesamiento** del programa, antes de su compilación. El preprocesador expande todas las macros utilizadas y el compilador compila el programa con el código ya expandido.

Definición de un macro

`#define <nombre_de_macro> <texto_de_reemplazo>`

El precompilador reemplazará cada ocurrencia de **nombre_de_macro** por **texto_de_reemplazo**. El nombre en un #define tiene la misma forma que un nombre de variable; el texto reemplazado es arbitrario. Cualquier nombre puede definirse con cualquier texto de reemplazo.

Algunos ejemplos de macros en C:

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 80
#define CICLO_INFINITO for(;;)
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define SWAP(a,b) {int t = (a); (a) = (b); (b) = t;}
```

Como vemos, estas macros contienen expresiones literales (también llamados objetos) y expresiones variables (también llamadas funciones). Las expresiones literales se sustituirán tal cual en la expansión mientras que las expresiones variables se sustituirán por su valor en la llamada a la macro.

En el ejemplo anterior, **TAM** es una expresión textual que se reemplazará por 80 al expandir la macro. Lo mismo ocurre con **CICLO_INFINITO**. Las dos macros siguientes utilizan las "variables" libres **a** y **b**. Si uno observa el ejemplo anterior de MAX, dentro de las macros se puede utilizar los operadores **ternarios**, ¡pero se debe prestar mucha atención al uso de los paréntesis y recordar que las condiciones se evalúan dos veces!

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Si hacemos la llamada a la macro MAX en un programa se hará de la siguiente forma:

```
x = MAX(p+q, r+s);
```

y se sustituirá por:

```
x = ((p+q) > (r+s) ? (p+q) : (r+s))
```

El preprocesador analiza textualmente la llamada y aplica la regla definida por la macro, generando el texto resultante. Este texto debe ser una expresión correcta del lenguaje de programación en el que se expande la macro.

Veremos que mediante las macros podemos extender el lenguaje de programación, saliéndonos de su sintaxis y creando un lenguaje propio.

Como, por ejemplo:

Supongamos que nos gusta un lenguaje del estilo de **Pascal**, en lugar de **C**. Podemos definir las siguientes macros

UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

```c
#include <stdio.h>
#include <stdlib.h>
#define then
#define begin {
#define end ;}


int main()
{
    int i=5,a=0,b=0;
    if (i > 0) then
       begin
        a = 1;
        b = 2
       end

   printf("a:%d \t b:%d",a,b);
   return 0;
}
```

El código anterior, lo que hará es que uno podrá programar como si fuera un IF en Pascal y el propio compilador luego va a hacer los reemplazos para convertirlo al lenguaje C.

¿Ahora, por qué utilizamos macros?

Las macros no siguen la sintaxis del lenguaje de programación en el que se programan, sino que son **metaprogramas** que generan código. Tienen su propia sintaxis de definición y de expansión. Por ello es posible utilizarlos para extender un lenguaje de programación, añadiéndole nuevas características.

Macros multilínea

En C, es posible dividir un macro en varias líneas si se requiere agregando un \ al final de cada instrucción. El siguiente ejemplo muestra un int x seguido de una cadena.

```
#define MOSTRAR(x, str) ({\
   printf("%d", x);\
   printf(" es ");\
   printf(#str);\
   printf("\n");\
})


int main()
{
    int x=1;
    MOSTRAR(x,int);

    return 0;
}
```

Este programa hará un **`printf`** mostrando el valor que estemos pasando a la macro MOSTRAR y el tipo de dato que le enviemos.

Referencias:

[1] B. W. Kernighan y D. M. Ritchie, El lenguaje de programación C, Pearson Educación, 1991.

[2] H. M. Deitel y P. J. Deitel, Cómo programar en C/C+, Pearson Educación, 1995.