

Estudio de un Sistema Operativo

Diseño de Sistemas Operativos

Problemas de aula 4. El preprocesador de C

Juan Carlos Pérez & Sergio Sáez

Sergio Sáez ssaez@disca.upv.es

Juan Carlos Pérez jcperez@disca.upv.es

4.1 Macros en C

- El preprocesador de C proporciona diversas funcionalidades muy útiles para hacer los programas más sencillos, eficientes y legibles.
- La característica más notable del preprocesador es su capacidad para procesar "macros". Una macro en C se expresa mediante la directiva **#define** seguida por un identificador, que quedará "definido". El resto de la línea se toma como la definición:

```
#define PI          3.141593
#define PI_MEDIOS (3.141593/2)
```

- El identificador definido se sustituirá siempre en el texto, por la definición, literalmente y antes de llegar éste al compilador.
- Es muy importante parentizar las expresiones porque nunca podemos estar seguros a priori de dónde y en qué contexto se emplearán las macros, pudiendo aparecer problemas de precedencia de operadores.


4.2 Macros con parámetros

- Las definiciones pueden llevar parámetros. En ese caso se sustituirá cada parámetro literalmente cuando se use la macro:

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


```
(...)
```

```
printf("Max{%d,%d}=%d\n",x,y,MAX(x,y));
```

- Define una macro que calcule el valor absoluto de un entero. Pruébala en un pequeño programa. **Ojo!** El símbolo de abrir paréntesis debe estar detrás del nombre de la macro **sin ningún espacio en medio.** 
- Define una macro **PRINTVEC(v, a, b)** que, usando la macro anterior, imprima los valores absolutos de los elementos del **a** al **b** del vector de enteros **v**. Para poder

declarar una variable local en la macro, se recomienda que pongas todo el código dentro de una estructura **do { ... } while (0)**. Por ejemplo:

```
#define PRINTVEC(v,a,b) \
do {                    \
    int _i;             \
    ...                 \
} while(0)
```

Observa que para definir una macro que ocupa varias líneas debes poner una barra invertida (\) como **último** caracter de la línea. Pruébala. 

4.3 Cadenización



- Existe un modo de utilizar un parámetro dentro de la definición de una macro que permite interpretarlo como una cadena de caracteres. Se trata de escribirlo precedido por el signo #:

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


```
#define MENSAJE_A(x) printf("Mensaje: %s\n", #x)
#define MENSAJE_B(x) printf("Mensaje: %s\n", x)
#define MENSAJE_C(x,y,z) printf("Mensaje: %s %s %s\n",x,y,z)
```

```
int main (int argc, char * argv[]) {
    char Soy[]="I am", un[]="a", compilador[]="compiler";
```

```
MENSAJE_A(Soy un compilador);
}
```

- Compila y ejecuta el programa. Prueba ahora a cambiar en la última línea **MENSAJE_A** por **MENSAJE_B**. Borra el ejecutable generado anteriormente y compila de nuevo. 
- Prueba a poner comas entre las 3 palabras **Soy, un, compilador** y a usar **MENSAJE_C**. ¿Entiendes todo lo que ocurre? 
- Teniendo en cuenta que en **C** las cadenas consecutivas se concatenan automáticamente, también podríamos definir la siguiente macro.

```
#define MENSAJE_A(x) printf("Mensaje: " #x "\n")
```


Pruébala. 


4.4 Concatenación

- Otro operador que puede usarse en una macro es **##**. Se coloca entre dos parámetros y su resultado es la concatenación de los valores de ambos

- Añade a tu programa la siguiente definición y pruébala con el **printf** indicado:


```
#define UNIR(a, b) a ## b
...
printf("Cadena: %s\n",UNIR(arg,v[0]));
```

Compila y ejecuta, e intenta comprender perfectamente el resultado. 

- El resultado de las macros vuelve a preprocesarse y se sustituyen de nuevo las macros que han aparecido nuevas. Prueba lo siguiente: 

```
#define MACRORARA "Soy una macro rara"
...
printf("Cadena: %s\n",UNIR(MA,CRORARA));
```


4.5 Precauciones

- Es importantísimo tener muy en cuenta la agrupación y precedencia de los operadores al definir una macro. En caso de duda, utiliza siempre paréntesis alrededor de cada parámetro.
- Comprueba el resultado de la siguiente definición y, en caso necesario, mejora o corrígela. 

```
#define CUADRADO(x) (x * x)

(...)

printf("2^2 = %d\n",CUADRADO(2));
printf("(3+2)^2 = %d\n",CUADRADO(3+2));
```

- También es muy importante, al invocar una macro, evitar o controlar el uso de expresiones con efectos "secundarios". Prueba lo siguiente: 

```
int i=2;
printf("2^2 = %d\n",CUADRADO(i++));
printf("3^2 = %d\n",CUADRADO(i));
```

4.6 Compilación Condicional

- El preprocesador de C también permite el uso de la compilación condicional, lo que permite optimizar el código para diversas situaciones.
- Una de las directivas más utilizadas es la construcción

```
#ifdef ALGO
... /* Hacer una cosa */
#else
```

```
... /* Hacer otra */
#endif
```


donde **ALGO** es una macro. Si **ALGO** esta definida se compilará *una cosa* y si no se compilará *otra*.


- o La macro **ALGO** debe estar definida con anterioridad o pasarsela al compilador con la opción **-D**.

```
gcc -DALGO macros.c -o macros
```

- o La siguiente macro para imprimir una variable de tipo entero.

```
#define VAR_INT(s) printf("Variable "#s " = %d\n", s)
```

Pruébala para imprimir alguna variable local. 

- o Modifica el código anterior para que la macro **VAR_INT** sólo imprima el valor de la variable si la macro **DEPURACION** está definida. En caso contrario no debe generar ningún código. 

4.7 Problema

- o Queremos definir en un programa en C un conjunto de 100 funciones que impriman los números del 0 al 99. El código de las funciones se generará con la macro **NUM(x,y)** que deberá imprimir el número **xy**. Ojo! Como la macro **NUM** genera la definición de una función, deba *invocarse* fuera de la función **main()**. Nos ayudaremos de la macro **DECENA** para ahorrarnos código:

```
#define DECENA(x) NUM(x,0) NUM(x,1) NUM(x,2) NUM(x,3) NUM(x,4) \
NUM(x,5) NUM(x,6) NUM(x,7) NUM(x,8) NUM(x,9)
```

- o Incluye en tu programa, al final de **main()** por ejemplo, las siguientes llamadas a varias de las funciones:

```
imprimir_0_1();
imprimir_0_9();
imprimir_1_1();
imprimir_2_7();
imprimir_3_8();
```

- o Define la macro **NUM** y utiliza **DECENA** adecuadamente para que el programa se compile e imprima los números 1,9,11,27 y 38. 