Lenguajes de Programación I

Facultad de Ciencias Exactas

Macros en C - Apunte Práctico

Los ejemplos de MACROS en C podrán probarse por medio del comando GCC.
Pueden emplear para ellos la máquina virtual provista en la clase de Threads.
Por ejemplo, si tenemos el siguiente código, llamado 01.c con el siguiente contenido:

```
#include <stdio.h>

#define BUFFER_SIZE 1024
#define NUMBERS 1, 2, 3

void XX(int x){
      x = 1;
}

void YY(int *x){
      *x = 1;
}

int main(void){
      char * foo = (char*)malloc(BUFFER_SIZE);
      int x[] = { NUMBERS };
      int mivariable = 1;
      BUFFER_SIZE = 1;
      XX(BUFFER_SIZE);
      YY(&VAR);
}
```

Para compilarlo y probarlo basta con hacer:

gcc 01.c -o 01

Esto crea el ejecutable 01 que puede ejecutarse de la siguiente forma:

./01

Existe además una opción para ver el código generado por el preprocesador de C donde las macros son reemplazadas, esta es la opción -E:

gcc -E 01.c

Daría como salida el siguiente texto:

```
int main(void){
      char * foo = (char*)malloc(1024);
      int x[] = { 1, 2, 3 };
      int mivariable = 1;
      1024 = 1;
      XX(1024);
      YY(&mivariable);
}
```

Donde puede verse el reemplazo mencionado.

El código anterior evidencia el problema de usar del lado izquierdo un parámetro pasado por nombre, ya que la sentencia 1024 = 1 es imposible de ejecutar. En el caso de C, la salida del preprocesador es tomada por el compilador y al intentar compilar, la sentencia mencionada va a dar error.

01.c:18:14: error: se requiere un l-valor como operando izquierdo de la asignación

Se adjunta código adicional donde se evidencian otros inconvenientes del uso de Macros en C que sirven para ejemplificar también los problemas de los pasajes de parámetros por nombre

```
#include <stdio.h>

int foo=1;

#define twice(x) (2*(x))
#define call_with_1(x) x(1)
#define min(X, Y) ((X) < (Y) ? (X) : (Y))
#define min2(X, Y) \

 ({   typeof (X) x_ = (X); \
      typeof (Y) y_ = (Y); \
             (x_ < y_) ? x_ : y_; })

#define foo (4 + foo)

int XX(int z){
 return z;
}

int main(void){
 int next, x, y, z;
 call_with_1 (twice);
 /**
```

```
 *     call_with_1 (twice)
              ==> twice(1)
              ==> (2*(1))
 */
 next = min (x + y, XX (z));
 next = min2 (x + y, XX (z));
 z = foo;
}
```

Código preprocesado:

```
int main(void){

 int next, x, y, z;
 (2*(1));
 next = ((x + y) < (XX (z)) ? (x + y) : (XX (z)));
 next = ({ typeof (x + y) x_ = (x + y); typeof (XX (z)) y_ = (XX (z)); (x_ <
y_) ? x_ : y_; });
 z = (4 + foo);
}
```

En la macro min, el problema es que se llama dos veces

next = ((x + y) < (XX (z)) ? (x + y) : (XX (z)));

No solo es ineficiente, sino que además XX(z) puede no arrojar el mismo resultado las dos veces que es invocada.

La solución se encuentra provista en la macro min2, que define una variable donde se almacenan los parámetros y se emplean estas variables internamente. Es algo así como convertir parámetros por nombre en copia valor.

Como ejercicio se sugiere analizar los otros dos problemas presentados en este ejemplo.