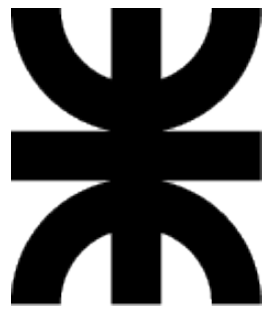
Informática II
Calificadores `volatile` y `const`

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Calificadores `volatile`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado en la mayoría de los casos para el acceso al hardware
- Le indica que la variable no se guarde en cache

Calificadores `volatile`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado en la mayoría de los casos para el acceso al hardware
- Le indica que la variable no se guarde en cache

Ejemplo:

```
salir = 0;
while(!salir)
{
  /* bucle corto completamente
   * visible al compilador */
}
```

Calificadores `volatile`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado en la mayoría de los casos para el acceso al hardware
- Le indica que la variable no se guarde en cache

Ejemplo:

```
salir = 0;
while(!salir)
{
  /* bucle corto completamente
   * visible al compilador */
}
```

- El compilador determina que el cuerpo del bucle no modifica la variable `salir` y convierte el bucle en un bucle `while(1)`.

Calificadores `volatile`

`volatile`

- Le indica al compilador no optimizar lo relacionado a dichas variables
- Utilizado en la mayoría de los casos para el acceso al hardware
- Le indica que la variable no se guarde en cache

Ejemplo:

```
salir = 0;
while(!salir)
{
  /* bucle corto completamente
   * visible al compilador */
}
```

- El compilador determina que el cuerpo del bucle no modifica la variable `salir` y convierte el bucle en un bucle `while(1)`.
- Si la variable `salir` se declara `volatile`, el compilador se ve obligado a cargarla cada vez, ya que puede modificarse en otro lugar.

Calificadores `volatile` – En μC ATmega328

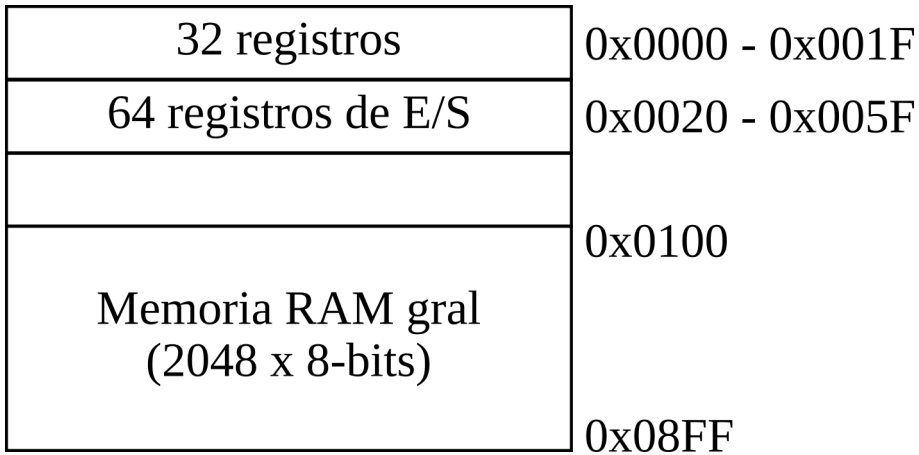
Se pueden definir la siguientes constantes simbólicas:

```
#define DDRB *( (volatile unsigned char *) 0x24 )
#define PORTB *( (volatile unsigned char *) 0x25 )
```

Calificadores `volatile` – En μC ATmega328

Se pueden definir la siguientes constantes simbólicas:

```
#define DDRB *( (volatile unsigned char *) 0x24 )
#define PORTB *( (volatile unsigned char *) 0x25 )
```



Calificadores `volatile` – En μC ATmega328

Se pueden definir la siguientes constantes simbólicas:

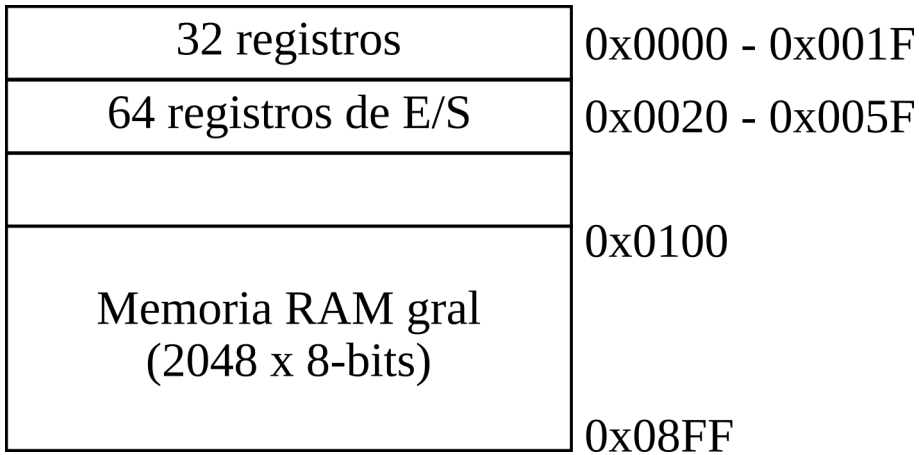
```
#define DDRB *( (volatile unsigned char *) 0x24 )
#define PORTB *( (volatile unsigned char *) 0x25 )
```

Configuración como salida digital:

```
DDRB |= (1 << 5);
```

Escritura de salida digital:

```
PORTB |= (1 << 5);
/* delay */
PORTB &= ~(1 << 5);
```



Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.

Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.
- Le informa al compilador que el valor de una variable no debe modificarse.

Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.
- Le informa al compilador que el valor de una variable no debe modificarse.

Uso de `const`

- Seis posibilidades del uso y no uso de `const` con parámetros de función.

Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.
- Le informa al compilador que el valor de una variable no debe modificarse.

Uso de `const`

- Seis posibilidades del uso y no uso de `const` con parámetros de función.
  - dos al pasar parámetros en llamada por valor, y

Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.
- Le informa al compilador que el valor de una variable no debe modificarse.

Uso de `const`

- Seis posibilidades del uso y no uso de `const` con parámetros de función.
  - dos al pasar parámetros en llamada por valor, y
  - cuatro al pasar parámetros en llamada por referencia (punteros).

Calificadores `const`

`const`

- No existía en la primeras versiones de C. Fue agregado en el ANSI C.
- Le informa al compilador que el valor de una variable no debe modificarse.

Uso de `const`

- Seis posibilidades del uso y no uso de `const` con parámetros de función.
  - dos al pasar parámetros en llamada por valor, y
  - cuatro al pasar parámetros en llamada por referencia (punteros).

Ejemplo:

```
void imprimir_arreglo(const int datos[], const int tam)
{
. . .
}
```

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

3. Puntero constante a datos no constantes. Inicializados al declararlos (nombre de arreglo)

Calificador `const` con punteros

Existen cuatro formas de pasar punteros a funciones:

1. Puntero no constante a datos no constantes, p.e.: `int *pi;` (notación) (no incluye `const`)

2. Puntero no constante a datos constantes

3. Puntero constante a datos no constantes. Inicializados al declararlos (nombre de arreglo)

4. Puntero constante a datos constantes

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo = &var;
```

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo = &var;
```

Puntero constante a entero

```
int const * foo = &var;
const int * foo = &var;
```

Calificador `const` con punteros –Ejemplos

Puntero a entero constante

```
int * const foo = &var;
```

Puntero constante a entero

```
int const * foo = &var;
const int * foo = &var;
```

Puntero constante a un entero constante

```
int const * const foo = &var;
const int * const foo = &var;
```