

Tipos de datos derivados y enumeraciones

Agrupación de datos para tratarlos solo como una variable

Ejemplo: supongamos que deseamos crear una función que realice la suma de dos números complejos.

Vamos a crear una función que sume dos números complejos:

```
int sumaComplejos(int realx, int realy, int imgx, int imgy){  
    int sumareal = realx + realy ;  
    int sumaimg = imgx + imgy ;  
    return ???  
}
```



¿Que va aca o como lo solucionamos?

Estructuras de datos

- Son tipos de datos definidos por el programador
- En python o C++ son “similares” a una clase
- La idea es agrupar varios datos y tratarlos como una sola variable
- Esto nos permite definir una variable que se llame “complejo” y tenga una parte real, y otra imaginaria, y que la función en lugar de recibir cuatro parámetros, solo reciba 2 parámetros.

¿Cómo definimos una estructura?

```
struct nombre {  
    tipo_dato par1;  
    tipo_dato par2;  
    ..  
};
```

Ejemplo:

```
struct ejemplo{  
    char c;  
    int l ;  
};
```

CODIGO EJEMPLO:

```
Struct ejemplo{  
    char c;  
    int i ;  
} ;  
.....
```

```
struct ejemplo ej1;  
ej1.i = 30;
```

Existen otras dos formas de definir una estructura. La forma es utilizar un “sinónimo”, mas conocido como alias

Un alias es similar a un apodo para un tipo de dato concreto.

Por ejemplo: supongamos que tengo tres enteros x,y,z que representan edades de personas. Podria en lugar de definir

```
int x,y,z
```

Usando un “apodo” podria decir “edad x,y,z”.

Para hacer esto, utilizamos la palabra “typedef” dentro de C.

Ejemplo:

```
typedef int edad ; // se define de forma global
```

```
....
```

```
edad x,y,z
```

Usando typedef tenemos otras dos formas para definir una estructura.

2° FORMA

```
typedef struct {  
    char c;  
    int i;
```

```
} ejemplo;
```

```
...
```

```
ejemplo ej1 ;  
ej1.i = 20 ;
```

3° FORMA

```
typedef struct ejemplo {  
    char c;  
    int i;
```

```
} ejemplo;
```

```
...
```

```
ejemplo ej1 ;  
ej1.i = 20 ;
```



Retomamos el ejemplo en código

Supongamos que ahora deseamos mas información del número complejo

- Módulo y fase
- Parte real e imaginaria

Simplemente la podrias agregar sin modificar el código:

```
typedef struct {  
    float real ;  
    float img ;  
    float modulo ;  
    float fase;  
}complejo
```

Agregar las funcion que calcule el módulo y fase de un número complejo

Tipo de dato enumerado

Supongamos que quiero realizar una función que calcule las cuatro operaciones básicas a partir de dos números:

- SUMA – OPCION 1
- RESTA– OPCION 2
- MULTIPLICACIÓN – OPCION 3
- DIVISIÓN– OPCION 4

```
float operacion(float n1, float n2, int op) {  
    float result ;  
    switch (op)  
    {   case 1:  
        result = n1 + n2 ;  
        break;  
        case 2:  
        result = n1 - n2 ;  
        break;  
        case 3:  
        result = n1* n2 ;  
        break;  
        case 4:  
        result = n1/n2 ;  
        break;  
        default  
        break;    }  
    return result ;  
}
```

Una opción es utilizar un
switch – case con una función
similar a esta

Es mas dificil recordar numeros y casos que palabras. Entonces se puede definir un nuevo tipo de dato llamado “enumerado”. EL objetivo es “legibilidad de código”.

```
typedef enum{  
    SUMA,  
    RESTA,  
    MULTIPLICACIÓN,  
    DIVISIÓN,  
}operacion;
```



Asigna valores enteros con nombres. Si no se explicita SUMA vale 0, resta vale 1 y asi siguiendo

Deseamos que arranque en 1, ENTONCES SE EXPLÍCITA “haciendo SUMA=1, RESTA=2 y así siguiendo

```
typedef enum{  
    SUMA = 1,  
    RESTA = 2,  
    MULTIPLICACIÓN=3,  
    DIVISIÓN= 4 ,  
}operacion;
```

Veamoslo con el ejemplo en código

Supongamos que queremos hacer una librería para leer el ads1115 (convertor analógico-digital)

- Configurar SPS (muestras por segundo)
- Alertas por tensión
- Polaridad del pin de alerta
- Modo continuo o simple
- Rango de lectura para medición de tensión
- Lectura en forma diferencial o single
- Dirección I2C de lectura

AHORA – NO NOS ACORDAMOS CUALES SON LAS POSIBLES OPCIONES DE SPS
Usamos una enumeración y se le asignan valores respecto a la hoja de datos(pag 28 datashet).

```
typedef enum {  
    SPS_8    = 0,  
    SPS_16   = 1 ,  
    SPS_32   = 2,  
    SPS_64   = 3,  
    SPS_128  = 4,  
    SPS_250  = 5,  
    SPS_475  = 6,  
    SPS_860  = 7,  
}  
ADS1115_sps_t ;
```

				Data rate These bits control the data rate setting. 000 : 8 SPS 001 : 16 SPS 010 : 32 SPS 011 : 64 SPS 100 : 128 SPS (default) 101 : 250 SPS 110 : 475 SPS 111 : 860 SPS
7:5	DR[2:0]	R/W	4h	

Análisis similar con las otras opciones

11:9	PGA[2:0]	R/W	2h	Programmable gain amplifier configuration These bits set the FSR of the programmable gain amplifier. These bits serve no function on the ADS1113. 000 : FSR = $\pm 6.144 \text{ V}^{(1)}$ 001 : FSR = $\pm 4.096 \text{ V}^{(1)}$ 010 : FSR = $\pm 2.048 \text{ V}$ (default) 011 : FSR = $\pm 1.024 \text{ V}$ 100 : FSR = $\pm 0.512 \text{ V}$ 101 : FSR = $\pm 0.256 \text{ V}$ 110 : FSR = $\pm 0.256 \text{ V}$ 111 : FSR = $\pm 0.256 \text{ V}$
------	----------	-----	----	---

```
typedef enum {
    FSR_6144 = 0 ,
    FSR_4096 = 1 ,
    FSR_2048 = 2 ,
    FSR_1024 = 3 ,
    FSR_512 = 4 ,
    FSR_256 = 5 ,
    FSR1_256 = 6 ,
    FSR2_256 = 7 ,
}ADS111x_PGA_values_t ;
```

Input multiplexer configuration (ADS1115 only)

These bits configure the input multiplexer. These bits serve no function on the ADS1113 and ADS1114.

000 : $\text{AIN}_P = \text{AIN}_0$ and $\text{AIN}_N = \text{AIN}_1$ (default)
 001 : $\text{AIN}_P = \text{AIN}_0$ and $\text{AIN}_N = \text{AIN}_3$
 010 : $\text{AIN}_P = \text{AIN}_1$ and $\text{AIN}_N = \text{AIN}_3$
 011 : $\text{AIN}_P = \text{AIN}_2$ and $\text{AIN}_N = \text{AIN}_3$
 100 : $\text{AIN}_P = \text{AIN}_0$ and $\text{AIN}_N = \text{GND}$
 101 : $\text{AIN}_P = \text{AIN}_1$ and $\text{AIN}_N = \text{GND}$
 110 : $\text{AIN}_P = \text{AIN}_2$ and $\text{AIN}_N = \text{GND}$
 111 : $\text{AIN}_P = \text{AIN}_3$ and $\text{AIN}_N = \text{GND}$

```
typedef enum {
    CHANNEL_0_1 = 0, //ONLY USE IF USE DIFFERENTIAL MODES
    CHANNEL_0_3 = 1, //ONLY USE IF USE DIFFERENTIAL MODES
    CHANNEL_1_3 = 2, //ONLY USE IF USE DIFFERENTIAL MODES
    CHANNEL_2_3 = 3, //ONLY USE IF USE DIFFERENTIAL MODES
    CHANNEL_0_GND = 4, //SINGLE MODE
    CHANNEL_1_GND = 5, //SINGLE MODE
    CHANNEL_2_GND = 6, //SINGLE MODE
    CHANNEL_3_GND = 7, //SINGLE MODE
}ADS1115_channel_t;
```


Luego de definir todas las enumeraciones, creamos la estructura de datos para la configuración del sensor. Anidamos una estructura dentro de otra

```
typedef struct {  
    ADS1115_alert_t enableAlert ; // hI TRHESH AND LO TRHESH USE IN On_window  
    ADS1115_polarity_alert_t polarity_alert ;  
    uint16_t HI_Thresh ;  
    uint16_t LO_Thresh ;  
}ADS1115_alert_comparator_t ;
```

|

```
typedef struct {  
    ADS1115_channel_t channel_select ;  
    ADS111x_PGA_values_t setPGA;  
    ADS1115x_mode_measurment_t mode_measurement ;  
    ADS1115_sps_t setSPS ;  
    ADS1115_alert_comparator_t alert_mode ;  
}ADS1115_config_t ;
```

Acceso a toda la libreria