Universidad Autónoma de Baja California

Ingeniería en Computación

Facultad de Ciencias Químicas e Ingeniería

Microcontroladores

Práctica 1. Introducción

Alumno: Zavala Román Irvin Eduardo

Docente: Castro Gonzales Ricardo

Grupo: 561

17/02/2022

Periodo 2022-1

Objetivo

El alumno se familiarizará con algunos usos típicos del Lenguaje C en sistemas embebidos.

Material

- Computadora Personal

Teoría

- Lenguaje C y su uso en sistemas embebidos.

- Compilador Cruzado (Cross-Compiler)

Instrucciones

Investigar:

- Tamaño de las primitivas y uso de inttypes.h.
- Representación de literales (1, 0x1, 0b1, 0)
- Operadores Bitwise ({&, |, ~, ^ } vs {&&, ||, !})
- Apuntadores (char *x, int (*cb)(void))
- Estructuras (struct, union, bitfield)
- Palabras claves (keywords: const, static, volatile)
- Condicionales del pre-compilador (#) • git (github)

Desarrollo

Tamaño de las primitivas y uso de inttypes.h.

En C existen diferentes tipos de datos básicos por defectos para poder crear datos más complejos a partir de estos, los tamaños de estos datos se muestran en la tabla 1.

Tipo	Tamaño en bytes
char, unsigned char	1
short int, unsigned short int	2
int, unsigned int, long int, unsigned long int	4
float	4
double	8
long double	12

Tabla 1. Tamaños de los tipos de datos en C en la plataforma Linux/Intel i686

Fuente: https://www.it.uc3m.es/pbasanta/asng/course_notes/data_types_es.html

El header inttypes.h sirve para usar tipos de entero con el tamaño preciso para la aplicación que necesite el programador. Los tipos de entero que deja usar este header están en la tabla 2.

Tipo		Descripción
Sin signo	Con signo	
uint8_t uint16_t uint32_t uint64_t	int8_t int16_t int32_t int64_t	Entero con el tamaño especificado
uint_fast8_t uint_fast16_t uint_fast32_t uint_fast64_t	int_fast8_t int_fast16_t int_fast32_t int_fast64_t	Entero más rápido con el tamaño especificado
int_least8_t uint_least16_t uint_least32_t uint_least64_t	int_least8_t int_least16_t int_least32_t int_least64_t	Entero más pequeño con el tamaño especificado

uintmax_t	intmax_t	Entero del tamaño máximo

Tabla 2. Enteros proporcionados por inttypes.h

Fuente: https://en.cppreference.com/w/c/types/integer

Representación de literales (1, 0x1, 0b1, 0)

Un literal es cualquier valor que puede aparecer en un programa, tanto numérico como alfanumérico. A continuación una lista de distintos ejemplos de literales:

- Decimales: Representación de manera normal, por ejemplo 10, 15, -85, 90
- Octal: En C, para representar números octales se empieza con 0. Por ejemplo 0123, 056, 07
- Hexadecimal: En C, para representar números hexadecimales se empieza con 0x. Por ejemplo 0xA, 0x56.
- Flotantes: Para representar una literal flotante se debe tener una parte entera, punto decimal y parte fraccionario. Por ejemplo 3.14, 2.71.
- Se pueden usar sufijos para estas literales, como u (unsigned int) o l (long int).
- Las literales tipo cadena se encuentran entre comillas dobles.
- Las literales tipo carácter se encuentran entre comillas simples.

Operadores Bitwise ({&, |, ~, ^ } vs {&&, ||, !})

Los operadores y, |, ~, ^ pueden llegar a ser confundidos con &&, ||, !. Esto debido a que comparten nombres y símbolos, pero la verdad es que los primeros sirven para modificar los bits a los datos y los demás para comparar valores de datos.

Operador ~: Cambiar los bits 1 por ceros y viceversa. Por ejemplo ~(1101) = 0010

Operador &: Realiza una comparación AND bit por bit entre operandos. Por ejemplo (1101) & (0110) = 0100. Se usa para “apagar” bits por medio de enmascaramiento.

Operador |: Realiza una comparación OR bit por bit entre operandos. Por ejemplo (1101) | (0110) = 1111. Se usa para “encender” bits por medio de enmascaramiento.

Operador ^: Realiza una comparación XOR bit por bit entre operandos. Por ejemplo (1101) ^ (0110) = 1011. Se usa para “invertir” bits por medio de enmascaramiento.

*Apuntadores (char *x, int (*cb)(void))*

Los apuntadores son variables que sus valores son direcciones de memorias de otras variables. La manera de declarar estos apuntadores es con el formato tipo_de_dato_apuntado *nombre_apuntador. A continuación un ejemplo de programa con apuntadores.

```
#include <stdio.h>
void funcion_1(int variable);
void funcion_2(int *variable);
int main() {
    int variable = 55;
    int *apuntador = &variable;
    printf("Direccion de variable: %d\n", apuntador);
    printf("Valor de variable: %d\n", *apuntador);
    funcion_1(variable);
    printf("Funcion 1: %d\n", variable);
    funcion_2(apuntador);
    printf("Funcion 2: %d\n", variable);
    return 0;
}
void funcion_1(int variable){
    variable = 100;
}
void funcion_2(int *variable){
    *variable = 100;
}
```

También existen apuntadores a funciones. Se declaran de la manera tipo_funcion (*nombre_apuntador)(tipo_dato_entrada) = &nombre_funcion.

```
#include <stdio.h>
void print(int a){ printf(“Valor: %d\n”, a);}
int main() {
     void (*apuntador_print)(int) = &print;
    (*apuntador_print)(15);
    return 0;
}
```

Estructuras (struct, union, bitfield)

C permite agrupar campos con otros tipos de datos en las estructuras. Esto permite al programador crear su propio tipo de dato. La forma más básica de una estructura es la siguiente:

```
struct nombre_estructura{
      tipo_1 nombre_campo_1;
      tipo_2 nombre_campo_2;
      tipo_3 nombre_campo_3;
      …
      tipo_n nombre_campo_n;
};
```

En las estructuras, cada campo tiene sus espacio en memoria, pero en ciertos casos se puede desperdiciar memoria. Para esto existen las uniones, las cuales se definen como una estructura en la que los campos comparten el mismo espacio en memoria, la declaración es igual que una estructura pero poniendo union en vez de struct.

```
union nombre_estructura{
      tipo_1 nombre_campo_1;
      tipo_2 nombre_campo_2;
      tipo_3 nombre_campo_3;
      …
      tipo_n nombre_campo_n;
};
```

Aun existiendo las uniones, la memoria puede seguir siendo desperdiciada por los tipos de datos, para eso existe el bitfield, para usar la memoria de manera eficiente cuando se sabe que no se va a usar el tamaño máximo. Por ejemplo:

```
#include <stdio.h>
struct fecha{
    /*
    Asi sera la declaracion de los campos normal:
    int dia;
    int mes;
    int year;
    */

    //Con bitfield

    //Comos los dias van del 0 al 31, solo ocupo 6 bits y no 4 bytes
    int dia : 6;
```

```
    //Comos los meses van del 1 al 12, solo ocupo 5 bits y no 4 bytes
    int mes : 5;

    int year;


};
int main() {
    struct fecha f1 = {30,12,2001};
      printf("Mi fecha de nacimiento es %d/%d/%d\n", f1.dia, f1.mes,
f1.year);
}
```

Palabras claves (keywords: const, static, volatile)

Las palabras clave son palabras con significado especial en el compilador de C.

La palabra const permite proteger valores contra modificaciones a lo largo del programa. Esto permite crear constantes.

La palabra static permite crear un tipo especial de variable, la cual no es destruida cuando la ejecución saliese de su ámbito y conserva su valor.

Al contrario de const, volatile le dice al compilador que la variable puede ser modificada por una rutina de background, interrupción o dispositivo I/O. Por lo que puede cambiar aun fuera del programa.

Condicionales del precompilador (#)

La primera etapa del proceso de compilación en C es el preprocesamiento, donde se filtra el código de los comentarios y directivas de procesamiento que inician por #.

Directiva	Descripción
#define	Define constantes simbólicas y macros funcionales
#include	Inclusión de ficheros

#if #else #elif #endif #ifndef	Permite incluir código de manera selectiva

Tabla 3. Directivas de precompilador

Fuente:
https://www.it.uc3m.es/~pedmume/asignaturas/2005/LAO/Lab2/tutorial4/www-etsi2.ugr.es/depar/ccia/mp2/old/apoyo/preproc/preproc.html

• *git (github)*

Git es un sistema de control de versiones, lo que significa que sirve para gestionar archivos de un proyecto y mantener un historial de las cosas en las que se trabajaron. Al trabajar generalmente se empieza con una versión básica de un programa y se va trabajando sobre este, por esto git sirve para tener en cualquier momento la oportunidad tanto de guardar cambios como para recuperar datos en caso de errores.

GitHub es un servicio web para el control de versiones con Git, básicamente es una red social para desarrolladores, permitiendo compartir código, modificar, opinar y más.

Conclusiones

La mayoría de cosas investigadas en este reporte se me hacían familiares, aunque cosas como la unión y el bitfield se me hicieron nuevas, entiendo el porqué de las aplicaciones de esto y todo lo investigado para aplicarlo en microcontroladores. Al buscar trabajar con hardware limitado a lo que estamos acostumbrados, el manejar a nuestro criterio los recursos da una flexibilidad para entrar en el área de microcontroladores.

Bibliografía

Bit Fields in C. (2021, 12 18). GeeksforGeeks. Consultado el 12/Feb/2022, desde https://www.geeksforgeeks.org/bit-fields-c/

C - Constants and Literals. (n.d.). Tutorialspoint. Consultado el 12/Feb/2022, desde https://www.tutorialspoint.com/cprogramming/c_constants.htm

Fixed width integer types (since C99). (2021, 06 20). cppreference.com. Consultado el 12/Feb/2022, desde https://en.cppreference.com/w/c/types/integer

GeeksforGeeks. (2018, 09 5). *Function Pointer in C*. GeeksforGeeks. Consultado el 12/Feb/2022, desde https://www.geeksforgeeks.org/function-pointer-in-c/

Gonzalez, L. (2019, 08 8). *¿Qué es Git y GitHub? - Aprende IA*. Aprende IA. Consultado el 12/Feb/2022, desde https://aprendeia.com/que-es-git-y-github/

La diferencia entre palabras clave auto, register, static, const, volatile y extern en lenguaje C. (n.d.). programador clic. Consultado el 12/Feb/2022, desde https://programmerclick.com/article/93731179974/

Prata, S. (n.d.). *C Primer Plus, Fifth Edition - By Stephen Prata*. CL72.org. Consultado el 12/Feb/2022, desde from http://gauss.ececs.uc.edu/Courses/c4029/code/C-primer/c-primer.pdf

3.2.1 Palabras clave. (n.d.). Zator. Consultado el 12/Feb/2022, desde https://www.zator.com/Cpp/E3_2_1.htm

Universidad Carlos III de Madrid. (n.d.). *Capítulo 2. Tipos de datos en C*. Ingeniería Telemática - UC3M. Consultado el 12/Feb/2022, desde https://www.it.uc3m.es/pbasanta/asng/course_notes/data_types_es.html