Universidad del Valle de Guatemala

Programación de Microcontroladores, Sec 20

Julio Andrés Avila G-S, Carné 19285

Programación en C de Microcontroladores

Tipos de Valores:

Entre los tipos de datos utilizados en el lenguaje C, se encuentran los **Integer**, los cuales representan valores numéricos, pueden ser chat (ASCII), short (16 bits), int (32 bits) y long (64 bits). Los intergers pueden ser precedidos por un *Unsigned*, que desactiva la representación negativa de los números, lo cual duplica el rango en cuanto a valores positivos.

Para los valores de tipo char, basta con colocarlos entre comillas, siendo un sinónimo de su forma numérico en el sistema ASCII. Para los demás comandos, '\n' para una nueva línea, '\t' para tabulación, '\0' como valor nulo y '\012' como una forma de colocar un valor en octal.

Los valores int deben ser seguidos por una L para asignar que serán una constante de tipo long, tambien puede expresarse de la forma 0x para hexadecimal y precedido de un "0" para expresarlo en octal. Es posible mezclar los tipos de valores en los comandos, el compilador tiene la capacidad de analizar valores chat e int como iguales, analizándolos desde una perspectiva numérica antes de realizar el comando. Sin embargo, esto no puede suceder cuando se provoca un overflow, ya que al operar dos valores de forma int que superan los 32 bits, no es posible operar dos valores distintos, por lo que sería necesario expresar uno de estos como Long.

Otros de los tipos de valores utilizados son los Floating point, los **float** tienen un solo número de coma florante de precisión (tamaño de 32 bits), los double tienen dos (64 bits) y los long double aun más grandes. Double es lo más utilizado pero float puede utilizarse especialmente para ahorrar espacio de memoria.

Los comentarios en C se denotan mediante /*…*/ o simplemente con //. Las variables deben ser declaradas con anticipación, debe definirse que tipo de valor será, int, chat, etc. Un aspecto importante sobre las variables es la sensibilidad hacia las letras mayúsculas y minúsculas, ya que "x" es una variable distinta a "X". La asignación de valores se realiza mediante =, como x = 6; x = x +1; etc.

En caso de un truncamiento (diferencia por el tipo de valor que es de menor rango), el valor de baja de rango simplemente. Si se asigna que pi es double con valor de 3.14159 y luego

se dice que int x; y que x = pi, entonces pi vale 3, porque no tiene la capacidad de expresar decimales. El lenguaje C no tiene booleanos, por lo que en lenguaje digital, reconoce 0 como falso y cualquier valor distinto de 0 como cierto

En cuanto a las operaciones matemáticas, C tiene totalmente la capacidad de usar las operaciones aritméticas binarias. Uno de los aspectos que debe cuidarse es el hecho de que la división entre dos int devuelve otro int (6/4)=1, por lo que es mejor hacerlo de forma (6/4.0)=1.5. Simplemente hay que usar + para suma, - para resta, / para división, * multiplicación y % para residuo.

Los operadores ++ y – sirven como incrementadores y decrementadores en los valores de variables. Si en una operación de coloca j = (i++ +10), se sumará 10 al valor de i antes de incrementar su valor, en cambio j = (++i + 10) sumará 10 al valor de i luego de haber incrementado su valor. Pero hacer un incremento o decremento antes o después de la operación es más fácil que aprenderse el orden de los operadores.

Los operadores de relación utilizados son == para igual, != no igual, < > mayor o menor que, <= >= mayor/menor o igual que, ! es not booleano, && and y || or. También está la posibilidad de trabajar a nivel de bits, ~ not, & and, | or, ˆ xor, >> divide por potencia de 2 y << multiplica por potencia de 2. Las operaciones pueden simplificarse, x = x + 10 puede sustituirse por x += 10, están +=, -=, *=, /=, %=, >>=, <<=, &=, |= y ˆ=.

Estructura de Controles:

Se utilizazn {} para agrupar comandos y se ejecutan en orden, las variables pueden declararse en el transcurso de una función, no necesariamente al inicio del código. La instrucción **if** junto con **if-else** se aplican de la misma forma:

```
If (expresión) {
        <comando>
}
Else {
        <comando>
}
```

El operador ternario funciona como un if simplificado, <expresión1> ? <expresión2> : <expresión3>, si la expresión1 es verdad, se toma como verdad la expresión2, de lo contrario se toma la expresión3. La función switch funciona para separar cada comando a realizarse con su respectiva condición a cumplir, ideal cuando se deben analizar varias expresiones con distintos resultados. Cabe resaltar que al final del switch debe haber un comando que se realiza por deault.

```
Switch (expresión) {
        Case <valor1>:
```

```
    Comando
    Break;
```

El ciclo **while** ejecuta una instrucción mientras una condición sea cierta, al momento que deje de serlo, se dejará de repetir el ciclo.

```
While (expresión) {
    Comando}
```

El ciclo **do-while** funciona como el while, solo que primero realiza el comando y luego verifica si se cumple la instrucción para retomar el ciclo.

```
Do { (comando)
    } while (expresión)
```

El ciclo **for** contiene la declaración de un valor inicial, una condición para continuar y una acción a ejecutar.

```
For (i = 0; i < 10; i++) {
    (comando) }
```

Break es un comando que permite detener un ciclo o un switch en cualquier punto, usualmente se coloca luego de un if adentro de otro ciclo, si la condición del if se cumple, el ciclo se interrumpe y salta al final de este. No funciona si el ciclo a trabajar es un if.

<u>Tipos de datos complejos:</u>

C tiene la capacidad de agrupar distintos tipos de datos en arrays que aquí se conocen como **structures**. Un ejemplo de esto es el **struct fraction** que funciona con dos integers.

```
Struct fraction {
    Int numerator;
    Int denominator; };
```

Luego se puede utilizar como:

```
Struct fraction f1, f2;
F1.numerator = 22;
F2.denominator = 7;
F2 = f1;
```

El tipo de array más simple simplemente declara el tamaño de este y da valor a el primer y último elemento del array. Los demás valores serán aleatorios si no se declaran.

```
Int scores[100];

Scores[0] = 13;
Scores[99] = 42;
```

Luego están los arrays multidimensionales, que funcionan similarmente. Siempre declarando desde el principio el primer y último elemento. Se permiten crear arrays que contengan struct fraction.

```
Int board [10] [10];
```

Board[0] [0] = 13;
Board[9] [9] = 13;

Los **pointers** funcionan como una forma de encontrar la localidad de una variable, estos deben definirse con * mientras se definen, no importa la posición del * mientras esté adentro de la declaración. Es posible definir el pointer del pointer de una variable. Mientras que * se utiliza para definir que una variable será pointer, & sirve para definir de cual variable.

```
int* pc, c;
c= 5;
pc = &c;
```

De esta forma se declara que pc es un pointer, específicamente de la variable c. Se le puede asignar el valor nulo a un pointer para definir que por el momento no tiene un valor al cual apuntar.

Los valores de tipo **string** no son más que arrays de datos char, la asignación solo con = no funciona, debe colocarse entre strcpy(), para diferenciar un dato de tipo string, este lleva un valor nulo al final del último valor.

```
{ char localString[10];
  Strcpy(localString, “hola”);}
```

Obteniendo que localString es hola0xxxxxx

Funciones:

“static” define que una función solo estará disponible para llamadas dentro del archivo donde fue creada. Se debe especificar el tipo de valor que se espera regresar. Cabe resaltar que las variables utilizadas dentro de una función solo existen dentro de esta.

```
Static int Twice(int, num) {
        Int result = num * 3;
        result = result – num;
        return(result); }
a = Twice(a)            // para llamar a la función.
```

En caso de que no haya un valor o resultado que se desee obtener de una función, se debe utilizar **void** para indicar que no retorna nada. También puede definirse **const** como parámetro para declarar que el valor de este no cambiará durante la función.

Impares y Ends:

Para la ejecución de todo un programa, debe iniciar con la función main() para que todo lo utilizado en el programa se compile junto en un solo archivo. El main() retorna un int, 0 si el programa se compiló sin problema o cualquier otro valor si presentó algún problema. El

prototipo de una función declara el nombre y los argumentos pero no su contenido, esto ayuda para poder hacer un llamado a la función, esto aplica para funciones non-static, las declaradas como static no lo necesitan.

El pre procesamiento ayuda al compilador a compilar de acuerdo al destino o uso del código. Las dos directivas utilizadas son #define e #include. #if funciona como una forma de definir que parte del código se debe compilar y termina con #endif, es útil para secciones del código que no se desean utilizar pero que se desean mantener en el archivo.