Tipos, operadores y expresiones

Andrés Alcarraz

11 de marzo de 2018

Resumen

En esta clase se ven los conceptos más básicos del lenguaje C como ser expresiones, operadores y control de flujo.

1 Adelanto [0:10]

Todo lenguaje de programación tiene un conjunto de reglas básicas a partir de las cuales se pueden construir sus sentencias, funciones y programas. Para que un lenguaje sea estándar e independiente de la plataforma (o lo más independiente posible) es necesario definir con la mayor precisión posible la sintaxis y la semántica de cada una de sus construcciones. De esta forma si el compilador respeta estas reglas, el programa funcionará de la misma manera en todos los entornos.

En esta clase, sin ser exhaustivos, vemos en construcciones básicas que componen un programa en C. A saber:

- Nombres de variables
- Tipos y tamaños de datos
- Constantes
- Declaraciones
- Operadores
- Expresiones Condicionales
- Estructuras de control de flujo.

2 Nombres de Variables [0:20]

Reglas que deben cumplir los nombres de las variables y convenciones de estilo.

- No se pueden usar palabras reservadas.
- Convenciones de estilo. Para identificar claramente si un literal es una constante o una variable se suelen diferenciar unas de otras, las variables se escriben todas en minúsculas usando el _ para separar y las constantes en mayúsculas también usando el _ para separar. Estas convenciones pueden variar ligeramente según el grupo de trabajo.

  Cuando se trabaja en un grupo sobre un mismo programa, es deseable que se acuerden distintas convenciones de estilo. Para que los distintos participantes no se sientan incómodos cuando tengan que leer o modificar código que escribió otro.
- Es deseable usar nombres de variables que expresen lo más claramente posible su propósito.
- Los ejemplos del libro por su longevidad no son necesariamente representativos de la forma de programar moderna.

3 Tipos de datos [0:30]

Tipos de datos básicos en C:

char un solo byte, a pesar del nombre

int entero de tamaño natural

float punto flotante de precisión normal

double punto flotante de precisión doble

Además para los int se pueden usar los modificadores short y long, para asegurarnos de usar menos o mas memoria.

Para los doubles se puede usar el modificador long para aumentar el rango.

Para todos los enteros se pueden usar los modificadores signed y unsigned para aumentar el rango en caso que no se vayan a representar números negativos por ejemplo.

El tamaño de cada tipo de datos depende del entorno de ejecución. Estos modificadores son necesarios cuando o bien sabemos que necesitaremos almacenar números realmente grandes, o bien necesitamos manejar la memoria (y el procesador) lo más eficientemente posible. Esto ultimo aplica sobre todo a sistemas embebidos o programación de microprocesadores de bajo consumo.

3.1 Constantes o literales [0:40]

La sección del libro 2.3 que se llama Constants en realidad se refiere a expresiones literales, o sea como se especifican en el lenguaje los valores de dichas constantes y no a variables constantes que son algo diferente.

Se suele usar la macro `#define` para definir referenciar expresiones constantes y no tener que replicarlas en cada aparición. Esto es en realidad algo del preprocesador que se verá ,ás adelante, pero que bñásicamente reemplaza en el código todas las apariciones de la definición por su constante literal antes de compilar.

Representaciones de constantes de los distintos tipos:

enteros

- L para long,
- U para unsigned,
- Formato:
  - Empieza con 0 para formato octal,
  - Empieza con 0x para formato hexadecimal
  - Entre ' ' char especificado en forma de carácter
  - '\ooo' char especificado en octal
  - '\hh' char especificado en formato hexadecimal.

decimales punto decimal o exponencial para double. f para especificar float, y l para especificar long double

cadenas se definen poniendo la cadena entre " ", por ejemplo `''soy una cadena''`

3.2 enumerados [0:50]

Tipo especial de constante que se usa para representar variables que pueden tomar un conjunto finito de valores.

Ventajas respecto a `#define` que se pueden declarar como tipos de datos y usar por ejemplo como declaración de variables o tipos de parámetros de funciones. En realidad internamente se representan por un entero, por lo cual pueden ser asignados a uno y los enteros pueden ser asignados a una variable de tipo enumerado.

En la declaración del enumerado se puede especificar a que valor entero corresponde cada elemento sucediendo un signo de igualdad y un entero. Por defecto el primero arranca en 0, y consecutivamente los demás toman su valor siguiente.

Ejemplo: ejemplos/enum.c imprimir una fecha

```c
#include<stdio.h>
/**
 * Imprimir una fecha
 */
 /*definicion de tipo enum mes*/
enum mes {ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO, AGOSTO, SETIEMBRE,
    NOVIEMBRE, DICIEMBRE};
int main(){

    enum mes mes = MARZO; /*declaracion de varaiable mes de tipo enum mes*/
    unsigned short int anio = 2018;
    int dia = 8;
    printf("%d/%d/%d\n",dia, mes, anio);
}
```

Este programa, imprime lo que se espera? Cómo lo arreglamos?

3.3 arreglos [1:00]

Un arreglo es una forma de almacenar una cantidad de elementos de determinado tipo, un arreglo almacena en una porción consecutiva de memoria tantos valores de ese tipo como el tamaño del arreglo.

Por ahora solo veremos arreglos de tamaño fijo, para tratar con arreglos de tamaño variable hay que lidiar con manejo dinámico de memoria. Ejemplo: máximo de un array.

Se definen especificando el tipo seguido de el tamaño entre corchetes. Opcionalmente Por ahora el tamaño puede ser una constante entera, del formato definido antes.

Comentar que se pueden usar para representar vectores o señales unidimensionales como sonido.

Ejemplo: ejemplos/arreglos.c Agregamos impresión del mes en texto

```c
#include<stdio.h>
#include<stdlib.h>
/**
 * Imprimir una fecha con mes en modo texto.
 */
 /*definicion de tipo enum mes*/
enum mes {ENERO=1, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO, AGOSTO, SETIEMBRE,
    OCTUBRE, NOVIEMBRE, DICIEMBRE};
char *nombres_mes[] = {NULL, "ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO", "JUNIO"
    , "JULIO", "AGOSTO", "SETIEMBRE"
    "OCTUBRE", "NOVIEMBRE", "DICIEMBRE"};
int main(int argc, char* argv[]){
    int dia = atoi(argv[1]);
    enum mes mes = atoi(argv[2]); /*declaración de variable mes de tipo enum mes*/
    unsigned short int anio = atoi(argv[3]);

    printf("%d de %s de %d\n",dia, nombres_mes[mes], anio);
}
```

También se pueden definir arreglos multidimensionales, pero no dejan de ser una sección continua de memoria que se accede de cierta forma.

4 Operadores y expresiones [1:10

1. Operadores aritméticos (`+,-,*,/,%`)
2. Operadores lógicos (`!, &&, ||`) y de relación (`>, >=, <, <=, ==, !=`), se usan para resultados booleanos, pero su resultado es siempre un entero que evalúa a 0 si la condición es falsa y 1 si la condición es verdadera.
3. Conversiones de tipo, casteo.
4. Operadores de incremento y decremento (`++,--`)
5. Operadores de asignación y expresiones
6. Expresiones condicionales: `condicion ? expr1 : expr2`
7. Operadores de manejo de bits

   Operaciones bit a bit `&, |,^`: and, or y xor respectivamente

   Operaciones de corrimiento `<<, >>`: Equivalen a multiplicar/dividor por 2 elevado al parámetro de la derecha

   Negación bit a bit `~`

   Ejemplo: ejemplos/bits.c distintas operaciones de bits

```c
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[]){
    if (argc < 3) {
        printf("parÃ¡metros insuficientes\n");
        exit(EXIT_FAILURE);
    }
    unsigned  a = strtoul(argv[1], NULL, 16), b=strtoul(argv[2], NULL, 16);

    /*bit a bit*/
    printf("%x & %x=%x\n", a, b, a & b);
    printf("%x | %x=%x\n", a, b, a | b);
    printf("%x ^ %x=%x\n", a, b, a ^ b);

    printf("~%x=%x, ~%x=%x\n", a, ~a, b, ~b);

    /*corrimiento*/
    printf("%d*2^%d=%d\n",a,b,a<<b);
    printf("%d/2^%d=%d\n",a,b,a>>b);

    return EXIT_SUCCESS;
}
```

5 Control de flujo [1:15]

5.1 sentencias y bloques

En C la unidad básica de ejecución es una sentencia, que es básicamente una expresión seguida de un ";"

En todo lugar donde pueda ir una sentencia se puede sustituir por una sentencia compuesta llamada bloque que consta de una secuencia de sentencias entre llaves {}

5.2 Condiciones booleanas [1:20]

En C existen básicamente las mismas construcciones de control de flujo que en la mayoría de los lenguajes estructurados como por ejemplo Pascal. La principal diferencia es que en C no existe el tipo booleano y por lo tanto las expresiones que determinan la condición de parada de un bucle pueden ser de cualquier tipo numérico aunque se aconseja que sean de algún tipo entero. Como vimos recién los operadores booleanos y de comparación devuelven un entero como resultado de la expresión.

5.3 Bifurcaciones [1:25]

Las bifurcaciones en C son implementadas como en cualquier otro lenguaje, tiene básicamente las variantes if, if-else y switch-case.

5.3.1 if-else

En los casos de if e if-else, se evalúa la condición de bifurcación y se ejecuta la sentencia del `if` si se cumple y si no se cumple, y existe el `else`

5.3.2 switch-case

Para el caso del switch-case, se compara una variable de algún tipo entero con distintos valores constantes posibles. Es importante destaca que una vez que se entra por un caso, sólo se sale por break.

Ejemplo: ejemplos/switch.c dias de un mes

```c
#include <stdio.h>
#include <stdlib.h>
enum mes {ENERO=1, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO, AGOSTO, SETIEMBRE,
    OCTUBRE, NOVIEMBRE, DICIEMBRE};
int main(int argc, char** argv){

    int mes, cant_dias;
    if (argc < 2) {
        printf("parametros insuficientes \n");
        return 1;
    }
    mes = atoi(argv[1]);
    switch (mes) {
      case ENERO:
      case MARZO:
      case MAYO:
      case JULIO:
      case AGOSTO:
      case OCTUBRE:
      case DICIEMBRE:
        cant_dias = 31;
        break;
      case ABRIL:
      case JUNIO:
      case SETIEMBRE:
      case NOVIEMBRE:
        cant_dias = 30;
        break;
      case FEBRERO:
        cant_dias = 28;
        break;
      default:
        printf("no es un mes valido\n");
        exit(1);
    }
```

```
    printf("el mes %d tiene %d dias\n", mes, cant_dias);
    return 0;
}
```

5.4 Bucles [1:40]

Existen los bucles `while` y `do-while` que son muy similares a los de otros lenguajes de programación, con la diferencia que la condición de parada es una expresión entera.

Otra particularidad de C es que no implementa de la forma habitual el for. El for de C consiste en tres partes:

1. La primera que se ejecuta una sola vez al comienzo y se suele utilizar para asignar el valor del índice en la primera iteración.
2. La segunda es la condición de parada, es decir la consdición que cuando se cumple hace que el bucle deje de ejecutarse, en un for estandar se verifica que el indice de iteración halla llegado al valor máximo.
3. La tercera es una instrucción de actualización, en un for estándar, solo incremente el indice de iteración.

Finalmente está el cuerpo del for que es la sentencia/bloque que viene luego.

5.5 Integrando [1:50]

A continuación modificamos el ejemplo sobre el que hemos venido trabajando para:

- Verificar que la fecha ingresada sea válida
- Tener en cuenta los años bisiestos
- Que se pueda ingresar el parámetro mes por nombre.

Ejemplo: ejemplos/loop.c Cálculo del dia del año de una fecha

```
#include<stdio.h>
#include<stdlib.h>
#include<strings.h>
/**
 * Imprimir día del año dada una fecha.
 */
 /*definicion de tipo enum mes*/
enum mes {ENERO=1, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO, AGOSTO, SETIEMBRE,
    OCTUBRE, NOVIEMBRE, DICIEMBRE};

int main(int argc, char *argv[]){
    char dias_por_mes[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,31};
    char *nombres_mes[] = {NULL, "ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO",
        "JUNIO", "JULIO", "AGOSTO", "SETIEMBRE", "OCTUBRE", "NOVIEMBRE",
        "DICIEMBRE"};
    if (argc < 4) {
        printf("Mal uso del programa, sintaxis: %s día mes año\n",argv[0]);
        exit(EXIT_FAILURE);
    }

    /*Leo el dia*/
    int dia = atoi(argv[1]);
    /*Leo el mes*/
    char *strmes=argv[2];
    int i=atoi(strmes); /*atoi devuelve 0 si el parámetro no empieza con un número
        */
```

```
    if (i == 0) {
        i = 1;
        while (i<=DICIEMBRE && strcasecmp(nombres_mes[i],strmes)) {
            i++;
        }
    }
    /*Verificamos el mes*/
    if (i<ENERO || i>DICIEMBRE) {
        printf("mes inválido: %s\n", strmes);
        exit(EXIT_FAILURE);
    }
    enum mes mes = i; /*vemos que a un enumerado se le puede asignar un entero, el
         compilador no verifica y en tiempo de ejecución tampoco se verifica que
        el entero sea una constanteválida del enumerado*/

    /*Leemos el año*/
    int anio = atoi(argv[3]);
    /*verificamos el año, no puede ser 0 ya que este no existe, en realidad el
        calendario gregoriano vale solo desde 1582 dependiendo además del país*/
    if (anio <= 0) {
        printf("año inválido: %s\n", argv[3]);
        exit(EXIT_FAILURE);
    }
    /*Si es bisiesto le agregamos un día más a ferbero*/
    dias_por_mes[FEBRERO] += (anio % 4 == 0) && (anio % 100 != 0 || anio % 400 ==0
         )  ;

    /*verificamos el día*/
    if (dia < 1 || dia > dias_por_mes[mes]) {
        printf("día inválido para el mes %s; %s\n", nombres_mes[mes],argv[1]);

    }
    int dia_del_anio=dia;

    for (i = 1; i<mes; i++){
        dia_del_anio += dias_por_mes[i];
    }

    printf("La fecha %d de %s de %d es el día número %d del año\n",dia,
        nombres_mes[mes], anio, dia_del_anio);
}
```