Programación : repaso de C (2)

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS

Agosto 2009

Outline

Outline

Un poco de historia

- Concebido en 1972 por Dennis Richie y Ken Thompson (Bell Labs), después del B.
- Inicialmente era para desarrollar un sistema de explotación UNIX, usando lenguaje de mas alto nivel que el ensamblador.

Un poco de historia

- Formalizado por Brian Kernighan y Dennis Richie en su libro *The C Programming language* (el K&R) en 1978.
- Adquiere muchísima importancia en los años 1980 (tomando el lugar de Fortran).
- Normalizado por el ANSI (American National Standards Institute) 1983-1989 (C89) : el ANSI C.
- Norma recuperada tal cual por la ISO (International Standards Organization) en 1990 (C90).
- Evolución en 1999 : C99 (cambios sobre directivas al preprocesador, unas nuevas palabras llaves, declaraciones dentro del código, inline, arreglos de tamaño variable. . .). Se acerca de C++

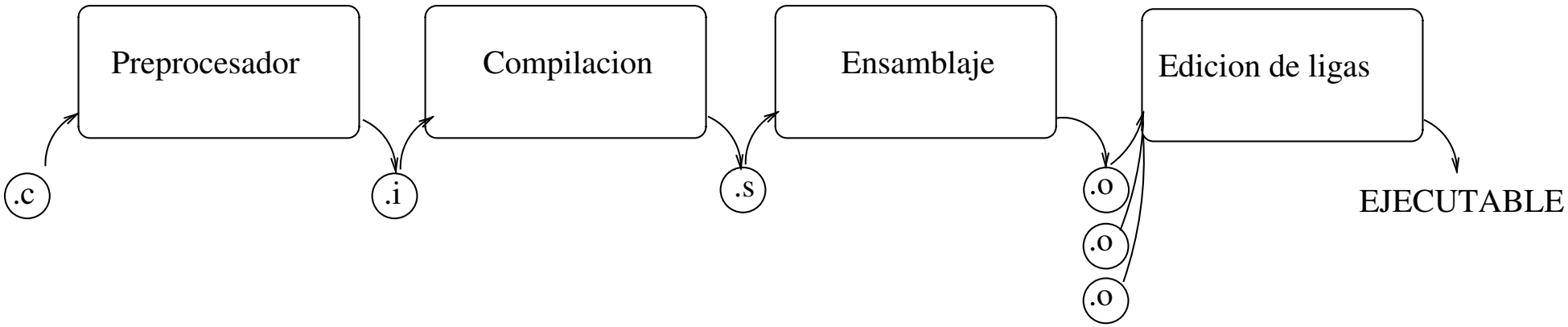
Lenguajes interpretados y compilados

- En un lenguaje interpretado, el codigo es convertido en instrucciones ejecutables por el microprocesador al momento mismo de la ejecución. Necesita un programa interprete. Ejemplos ?
- En un lenguaje compilado, el codigo es traducido en instrucciones ejecutables una vez por todas. Ejemplos ?
- C es una lenguaje compilado (en la gran mayoría del tiempo) pero existen interpretadores también `http://root.cern.ch/drupal/content/cint`

Características del C

+ Lenguaje de bajo nivel entre los de alto nivel.
+ Adaptado para operaciones sistema, accesos directos a la memoria, control del bajo nivel (herencia de su creación...).
+ Gran control y libertad del programador (memoria,...).
- Pero requiere esfuerzos del programador (liberación de memoria, verificación de indices...).
- Tal cual no tiene manera de manejar cadenas de caracteres o estructuras de alto nivel.

Compilación



- El preprocesador trata el código mismo (#define,#include) (.i).
- La compilación transforma el código en ensamblador (.s).
- El ensamblaje transforma el ensamblador en lenguaje maquina, se obtiene un archivo objeto (.o).
- La edición de ligas combina varios archivos objeto para hacer un ejecutable.

Elementos del lenguaje

- Identificadores : nombre de variable, de struct, de label, de funciones.
- Palabras llaves : nombres de tipos, de estructuras de control, calificadores de tipos, de almacenamiento.
- Constantes y nombres de macros sustituidas por el preprocesador : definidas por DEFINE.
- Cadenas de caracteres.
- Operadores : +,*...
- Puntuación.
- Comentarios : /* Comentario */ (o // Comentario a partir de C99)

Expresiones

- Las instrucciones son una serie de expresiones sintácticamente correctas terminadas por un ;.
- Declaración : *int a* = 1;.
- Funciones:

```
tipo mi_funcion ( argumentos ) {
        declaraciones de variables internas
        instrucciones
        return algodetipotipo;
}
```

Constantes enteras

Varias maneras de escribir las constantes

- Decimal

```
int x=100;
```

- Octal (introducido por 0)

```
int x=0144;
```

- Hexadecimal (introducido por 0x)

```
int x=0x64;
```

Se puede poner sufijos para indicar que la constante es LONG (l o L) o UNSIGNED (u o U).

Constantes flotantes

- Representación mantisa-exponente.
- Por default en DOUBLE.
- Puede ser modificada en LONG DOUBLE (con L) o en FLOAT (con F).
- Ejemplos :

```
12.34       double
12.3e-4     double
12.34F      float
12.34L      long double
```

Outline

Operadores aritméticos

- Operadores aritméticos : +,-,/,*,%
- Cuidado al / es a la vez división entera y división en flotantes : si los dos operadores son enteros, hará una división entera; si no, una división entre flotantes.
- Para otros operadores matemáticos, ver MATH.H.
- Error clásico

```
int i=1,j=2;
double medio = i/j;
```

lvalue y rvalue

- Una *lvalue* tiene una dirección a que se puede ir : son las únicas que se puede poner a la izquierda de una afectación $=$.
- i es una *lvalue*, 2*i no es (es una *rvalue*).

Operadores de relación

¿ Qué dice ?

```
int main() {
   int a=0, b=1;
   if (a = b)
      printf("\n a y b son iguales \n");
   else
      printf("\n a y b son diferentes \n");
   return 0;
}
```

Operadores de relación

- Cuidado a la diferencia entre $==$ (relación) y $=$ (afectación).
- Operadores $>$, $<$, $>=$, $<=$, $==$, $!=$.
- Cuidado a valores flotantes muy cercanos (cf. clase 1).

Operadores lógicos

- && : AND
- || : OR
- ! : NOT
- La evaluación regresa un INT que vale 1 si la expresión lógica es TRUE, 0 si no.
- Evaluación de izquierda a la derecha en caso de grupo de expresiones :

```
if ((i >= 0) && (i <= 9) &&
    !(a[i] == 0) && (b[i]>3)) {
...
```

- Pensar el orden de las expresiones !

Operadores lógicos sobre los bits

- & : AND
- | : OR
- ^: XOR
- ~: NOT
- << y >> : *shift* de los bits a la izquierda o a la derecha; añade ceros al final o al principio del numero según el caso.
- No confundir con operadores booleanos.

Operadores lógicos sobre los bits

- Cuidado con el tamaño de los datos

```
unsigned char c1=128;
unsigned char c2=c1<<1;
printf("%d\n",c2);
```

- Cuidado con el signo de los datos

```
char c3=127;
char c4=c3<<1;
printf("%d\n",c4);
```

- Un *shift* hacia la derecha no afecta el signo

```
char c5=-127;
char c6=c5>>1;
printf("%d\n",c6);
```

Aplicaciones : multiplicaciones y divisiones

Usar los operadores de *shift* para multiplicar/dividir por potencias de 2 (pero con cuidado) puede ser mas rápido que usar operadores aritméticos.

Aplicaciones : flags

En los programas se puede querer combinar una serie de opciones que podemos activar o no para nuestras funciones. Una idea es ponerlas cada una correspondiendo a un bit de un número suficientemente grande.

```
#define FLAG_X 0
#define FLAG_1 1
#define FLAG_2 2
#define FLAG_3 4
#define FLAG_4 8
```

Aplicaciones : flags

- Verificar si unos flags estan activados

```
void funcion(unsigned int flags)
{
        if (flags != 0)
        {
            if ((flags & FLAG_1) !=0 )
            // El primer flag es activado
            if ((flags & FLAG_2) !=0 )
            // El primer flag es activado
            ...
         }
         else
         // FLAG_X
```

- Formar un entero de unos flags dados :

```
funcion(FLAG_1 | FLAG_2);
```

Operadores de afectación compuesta

$+=$ $-=$ $*=$ $/=$ $\%=$ $\&=$ $\hat{}=$ $|=$ $<<=$ $>>=$

Por ejemplo, $a+=b$ es equivalente a $a = a + b$. La ventaja es que el término de la izquierda solo es evaluado una vez

Operadores de incrementación

Incrementan/decrementan el valor de enteros. Pueden ser pre-fijos o post-fijos

```
int a = 3, b, c;
b = ++a;        /* a y b valen 4 */
c = b++;        /* c vale 4 y b vale 5 *
```

útiles en estructuras de tipo *loop*.

Operadores de incrementación : remarca

- En c++, puede ser que $++i$ sea mas rápido que $i++$.
- Incluso los enteros son objetos; estos operadores son implementados como métodos de objetos.

Operador coma

Se puede utilizar para combinar varias instrucciones en una, evaluándolas de la izquierda a la derecha :

```
int a, b;
b = ((a = 3), (a + 2));
printf("\n b = %d \n",b);
```

Cuidado, no es este operador que aparece por ejemplo en los printf (en que de hecho no hay garantía en el orden de las evaluaciones).

Operador condicional ?

Es un operador a tres operandos :

```
condición?haz1:haz2;
```

Si la condición se cumple 'haz1', si no 'haz2'.

```
m = ((a > b) ? a : b);
```

Operador de cast

Para convertir objetos de un tipo a otro se usa el operador de *cast*, que se usa :

```
type1 b = ...;
type2 a = (type2) b;
```

Cast automático

Operadores pueden estar aplicados con operandos de diferentes tipos, implicando que las variables de tipo mas pequeño están convertidas en el tipo mas grande entre los operandos. En caso de la afectación, el valor de la derecha esta convertido en el tipo del operando de la izquierda.

Cast automático

```
int i = 8;
int j;
float x = 12.3;
double y;
y = i * x;
j = i * x;
```

Cast automático

Reglas:

1. Entre dos enteros : char y short están convertidos en int; luego, la computadora elige el tipo mas largo.

   int , unsigned int , long , unsigned long

2. Entre un entero y un flotante, se convierte el entero en el tipo flotante.
3. Entre dos flotantes, se convierte el operando menos largo en el tipo mas largo :

   float , double , long double

Cuidado a los overflows.

Cast automático

Efectos de borde

```
unsigned long a = 23;
signed char b = -23;

printf( "a %c b\n", a<b ? '<' : (a==b ? '=' : '>') );
```

Operador de dirección

Se consigue la dirección memoria de un objeto por el operador & :

```
type1  b = ...;
type1 *a = &b;
```

Prioridades entre operadores

Prioridad	Operadores
1 (la mas fuerte)	()
2	! ++ --
3	* / %
4	+ -
5	< <= > >=
6	== !=
7	&&
8	||
9 (la menos fuerte)	= += -= *= /= %=

Con operadores de misma prioridad, evaluación de la izquierda a la derecha.

Si dudas, usar paréntesis !

Prioridades entre operadores

Ejemplo

```
int a=2;
int b=2;
printf("%d",!--a==!b);
```

Para mejor lisibilidad, usar paréntesis.

Prioridades entre operadores

Ejemplo

```
X *= Y + 1;          <=>          X = X * (Y + 1);
X *= Y + 1;     no equivale a     X = X * Y + 1;
```

Outline

if...then...else...

Evaluación condicional

```
if (expresion1 )
  instruccionbloque1
else if (expresion2 )
  instruccionbloque2
    ...
else if (expresionn )
  instruccionbloquen
else
  instruccionbloquedefault
```

P.D. : Un bloque con mas de una instrucción tiene que estar rodeado de {}

switch

¿Cuánto vale b al final ?

```
int a=1;
int b;
switch (a) {
  case 1:
        b=5;
  case 2:
        b=9;
  default:
        b=3
        break;
 }
```

switch

```
switch (expresion ) {
  case valor1:
       instruccionbloque1
       break;
  case valor2:
       instruccionbloque2
       break;
     ...
  case valorn:
       instruccionbloquen
       break;
  default:
       instruccionbloquedefault
       break;
}
```

while

Evaluación mientras una expresión es *true*

```
i = 2;
while (i < 5) {
    printf("\n i = %d", i++);
}
No entra en la evaluación si la condición no es OK.
```

do...while

```
i = 2;
do {
    printf("\n i = %d", i++);
} while (i < 5);
```

Entra en la evaluación de todos modos

for

Instrucción(es) de principio - Condición(es) de terminación - Instrucción(es) de incremento

```
for (i = 0; i < 5; i++)
   printf("\n i = %d", i);
```

break

Al interior de ciclos, permite salir de estos.

```
i = 2;
do {
      printf("\n i = %d", i++);
      if (i==6)
          break;
} while (i < 100);
```

continue

Al interior de ciclos, permite saltar al próximo ciclo.

```
for (i = 0; i < 5; i++)
    if (i%2==1)
        continue;
    printf("\n i = %d", i);
}
```

Outline

printf

Impresión formateada en la salida estándar

```
for (i = 0; i < 5; i++)
    printf("\n i = %d", i);
}
```

La cadena de control (primer argumento de printf) contiene el formato de la impresión.

printf

Formato	Conversión en	Representación
%d	int	decimal con signo
%ld	long int	decimal con signo
%u	unsigned int	decimal sin signo
%lu	unsigned long int	decimal sin signo
%o	unsigned int	octal sin signo
%lo	unsigned long int	octal sin signo
%x	unsigned int	hexadecimal sin signo
%lx	unsigned long int	hexadecimal sin signo

Ademas se puede especificar un espacio total que imprimir: %10d por ejemplo pone 10 cifras (con ceros donde necesario, y alineado a la derecha)

printf

Formato	Conversión en	Representación
%f	double	decimal coma fija
%lf	long double	decimal coma fija
%e	double	decimal notación exponencial
%le	long double	decimal notación exponencial
%g	double	mas corta entre %f y %e
%lg	long double	mas corta entre %lf y %le
%c	unsigned char	carácter
%s	char*	cadena de caracteres

Ademas, se puede indicar espacio reservado antes y después de la coma : %5.5f significa que se imprimirá el flotante con 5 cifras antes y después de la coma.

scanf

Entrada de datos por el input estándar; funciona un poco como printf, excepto que el segundo argumento lleva la dirección en memoria donde poner el dato leído. Cuidado a que el formato elegido y el tipo de la variable donde guardar el dato sean compatibles.

```
#include <stdio.h>
int main()
{
   int i[10],k;
   for (k=0;k<10;k++)     {
            printf("entra un entero ");
            scanf("%d",&i[k]);
            printf("i[%d] = %d\n",k,i);
   }
   return 0;
}
```

I/O para caracteres

Las funciones getchar() y putchar(char c) realizan las operaciones de lectura/escritura de caracteres en la entrada/salida estándar.

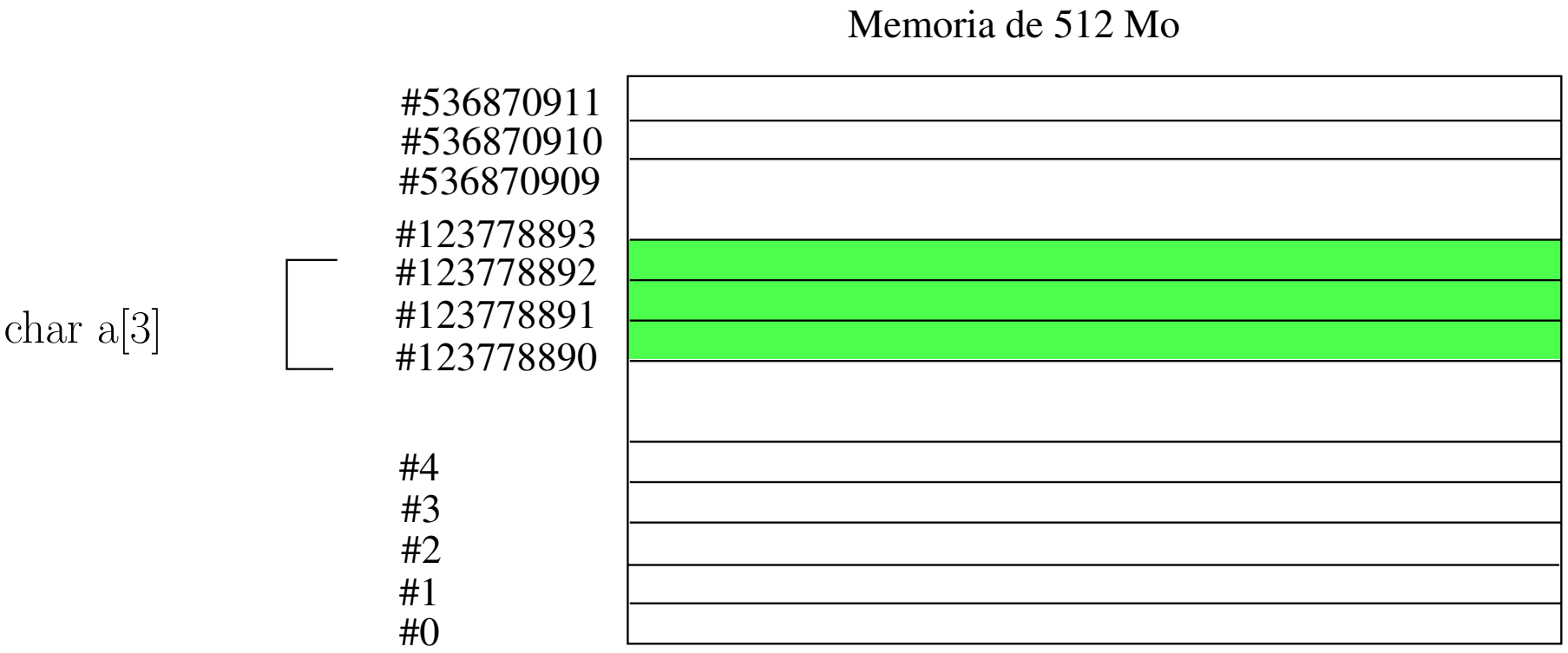
Outline

Arreglos estáticos

Es un conjunto finito de datos del mismo tipo y guardados en espacios contiguos en memoria. No necesitan ser liberados o alocados. Su declaración se hace por :

```
tipo nombreArreglo[tamañoArreglo];
```

Arreglos estáticos



Arreglos estáticos : acceso

El acceso a los valores se hace por el operador corchete, con valores entre 0 y $N - 1$, donde N es el tamaño del arreglo (diferente de MATLAB).

Arreglos estáticos : ejemplo

```
#define N 10
int main()
{
   int tab[N];
   int i;
   ...
   for (i = 0; i < N; i++)
      printf("tab[%d] = %d\n",i,tab[i]);
   return 0;
}
```

Para no tener que poner valores numéricos en todo su código, se puede usar el comando preprocesador define.

Arreglos estáticos

El arreglo en sí mismo es un apuntador hacia el primer valor del arreglo; este apuntador es constante, no es un lvalue (no se le puede afectar un valor). En particular, copiar un arreglo hacia otro implica copiar cada uno de sus elementos :

```
#define N 10
int main()
{
  int tab1[N], tab2[N];
  int i;
  ...
  for (i = 0; i < N; i++)
    tab1[i] = tab2[i];
  return 0;
}
```

Arreglos estáticos : inicialización

Se puede inicializar el contenido del arreglo por un comando :

```
tipo nombreArreglo[tamañoArreglo] = {a1,a2,...aN};
```

donde $a1$, $a2$, ..., aN son constantes.

- Se puede poner $n < N$ constantes, y solo los n primeros elementos del arreglo estarán inicializados.
- Los arreglos no inicializados no están necesariamente con valores $= 0$! Esto es el caso solo si (1) el arreglo es una variable global o (2) si es una variable declarada *static*

Arreglos estáticos : caracteres

Cuidado al inicializar un arreglo de caracteres por una cadena de caracteres :

```
char toto[5] = "hello";
```

No funcionará.

Arreglos estáticos : caracteres

- Toda cadena de carácter esta terminada por un carácter especial \0.
- Se puede inicializar un arreglo sin especificar el tamaño (que estará deducido del numero de elementos dados en la inicialización)

```
char toto[] = "hello";
printf("Tamaño del arreglo %d\n",
                              sizeof(toto));
```

Arreglos estáticos multidimensionales

Es idéntico a lo de los arreglos monodimensionales, en dimensión K :

```
tipo nombreArreglo[tamañoDim1]...[tamañoDimK];
```

Se accede al elemento $(i1, i2, .., iK)$ por $nombreArreglo[i1]\ldots[iK]$; es similar a un apuntador múltiple : los elementos $nombreArreglo[i1]\ldots[iK-1]$ son apuntadores, $nombreArreglo[i1]\ldots[iK-2]$ apuntadores dobles etc...

Arreglos estáticos multidimensionales

```
#define M 2
#define N 5
int tab[M][N] = {{1, -2, 7, 9, 8}, {4, 5, -6, 0, 3}};

int main()
{
   int i, j;
   for (i = 0 ; i < M; i++) {
        for (j = 0; j < N; j++)
          printf("tab[%d][%d]=%d  ",i,j,tab[i][j]);
        printf("\n");
     }
   return 0;
}
```