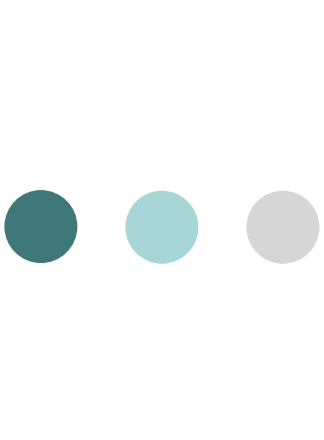


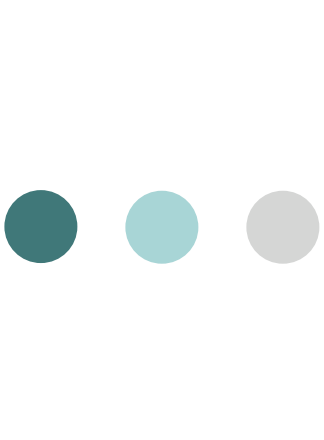
Lenguaje C

Todo el lenguaje ANSI C

Computadores I 2005-2006

Contenido

- Introducción
- Tipos de Datos
- Operadores
- Expresiones
- Sentencias
- El Preprocesador
- Vectores y Punteros
- Funciones
- La Biblioteca de Funciones



Introducción

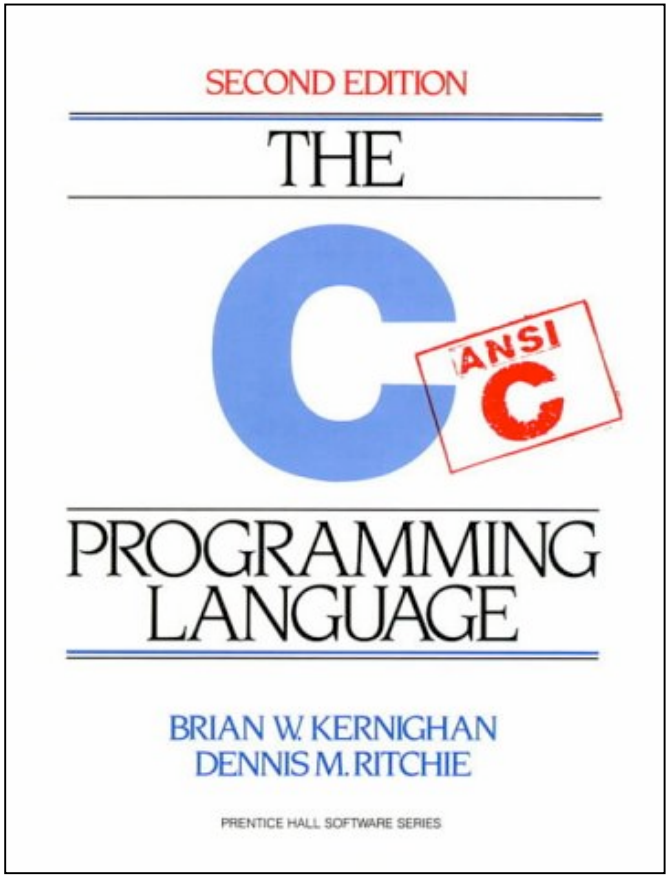
Palabras clave, estructura de programas, proceso básico, etc.

¿Qué es C?

- Un lenguaje de programación de propósito general
- Heredero de B y BCPL
- Creado por Bell en los ‘70 al mismo tiempo que UNIX
- Estandarizado por ANSI/ISO
- Es el lenguaje de elección de los programadores de sistemas

Referencia básica

- *The C Programming Language*, por Brian Kernighan and Dennis Ritchie, 2nd Edition, Prentice Hall
- La segunda edicion describe **ANSI C**
- El lenguaje de la primera edición ce conoce como **C K&R**



39 Palabras Reservadas

- main
- for, while, do
- if, else, switch, case, default
- break, continue, goto
- return, sizeof
- asm, endasm
- void, char, int, short, long, unsigned, float, enum, double, struct, union, typedef, signed
- const, volatile, auto, extern, register, static

Estructura de un Programa

- Colección de declaraciones y funciones
- Organización habitual:
  - definiciones de macros
  - ficheros de encabezamiento propios y de la librería estándar
  - declaración de variables globales y tipos
  - Funciones - incluido main()

Estructura de una función

- Cada una de las funciones
  - parámetros, declaración de variables, sentencias

```
int suma(int a, int b) {
    int c;
    c = a + b;
    return c;
    }
```

Ejemplo de programa

```
#define PI 3.1416
#include <stdio.h>
float calcula(int x) {
    return x*PI/180;
    }
int main(void) {
    int gra=90, rad;
    rad = calcula(gra);
    printf("%d grados son %f radianes\n", gra, rad);
    }
```

Estructura de un Programa

- El programa más sencillo
  - main() { }
- Conceptos básicos sobre sentencias
  - Cada sentencia se cierra con ;
  - ; no es un separador como en otros lenguajes.
    x = 5;
- Bloques : {}
  - Donde haya una sentencia se puede poner un bloque

Estilo de Programación

- Legibilidad
  - nombres
  - indentación
  - espacios
  - una sentencia por línea
  - comentarios
- Hacer buen uso de las posibilidades del lenguaje

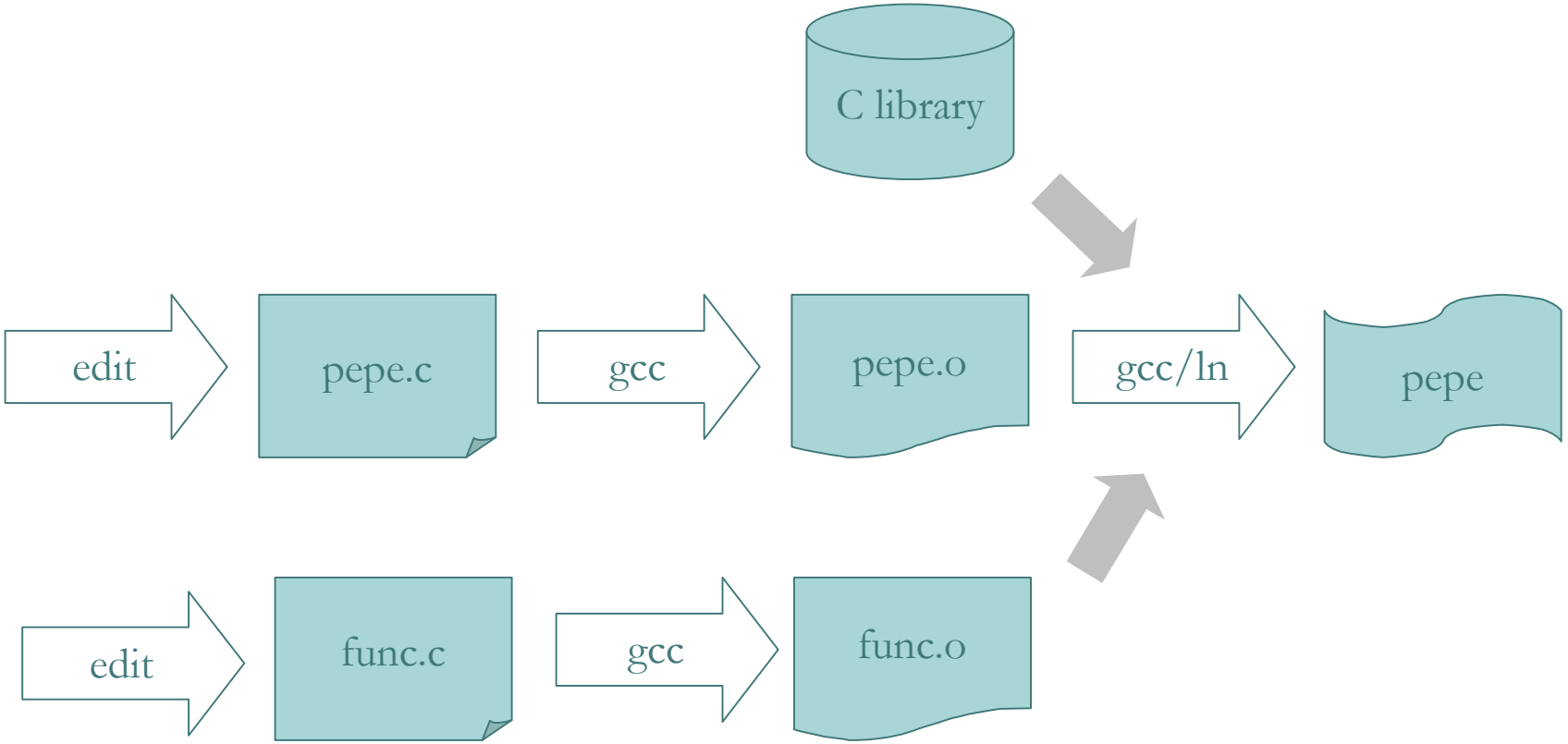
Compilación de Programas

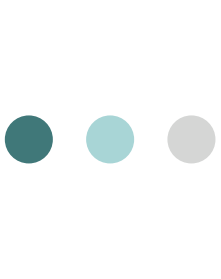
- Lenguajes compilados v.s. Lenguajes Interpretados
- Código máquina
- Tipos de ficheros
  - fuentes
  - objetos y librerías
  - ejecutables

Compilación de Programas

- Compilación en UNIX (gcc)
- Un solo fichero
  - gcc pepe.c (a.out)
- Varios ficheros
  - gcc -c pepe.c
  - gcc -c func.c
  - gcc -o pepe pepe.o func.o

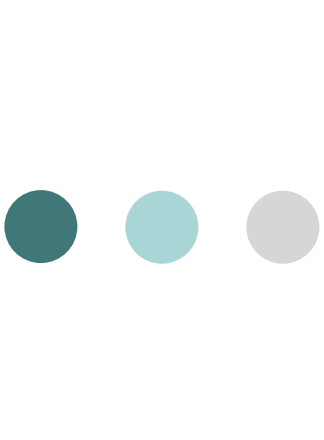
Compilación de Programas





Portabilidad

- Escribir C Limpio
- Experimentar en varias plataformas
- Evitar compiladores peligrosos



Tipos de Datos

Modos de almacenamiento y manejo de la información

Números

- ¿Cómo escribir el número 26?
  - decimal: potencias de 10 (26 en C)
  - notacion científica: exponencial (.26E2 en C)
  - octal: potencias de 8 (032 en C)
  - hexadecimal: potencias de 16 (0x1A o 0X1A en C)

Tamaño de los datos

- Tamaño de almacenamiento
  - bit (0 o 1, 2^1)
  - byte (enteros de 8 bits, 0 a 255, 2^8)
  - palabra (0 a 65535, 2^{16})
    - depende de la máquina !!!

Códigos y Notación

- La codificación interna es siempre binaria
  - Codificación binaria del entero 26
    - potencias de 2 (00011010) 0 a 255
  - Codificación binaria del número -26
    - 1 bit para el signo (01111010) -128 a 127
  - Codificación binaria del carácter A
    - Código ASCII o EBCDIC (’A’=65), 7 bits
    - Códigos extendidos para aprovechar los 8 bits

Códigos y Notación

- Codificación binaria de números en punto flotante (-6.4 o bien -.64E1)
  - signo: 1 bit
  - parte decimal (64): 23bits
  - exponente (1): 8 bits
- Se suele seguir la norma IEEE 754
  - Simple precision
  - Doble precision

Tipos de Datos K&R

- int
  - Enteros con signo (16 o 32 bits) –32768 a 32767
- unsigned int o unsigned
  - Enteros sin signo (16 o 32 bits) 0 a 65535
- short int o short
  - (16 bits) en un PC int = short
- long int o long
  - (32 bits) en una Sun int = long

Tipos de Datos K&R

- short <= int <= long
- En algunas implementaciones existen también
  - unsigned long int o unsigned long
  - unsigned short int o unsigned short
- *Overflow* en enteros
  - Si a short int (8 bits) se le asigna 258 en un PC
  - ¡No es un error! : ES un 3
- No hay tipo booleano

Tipos de Datos K&R

- char
  - Caracteres alfanuméricos (1 byte) -128 a 127 o 0 a 255, depende del compilador
  - Letras, números, y otros como &, %, $, =,)
  - Realmente almacena enteros
  - Código ASCII ('A'=65), 7 bits
- En las implementaciones en las que se almacena con signo, puede especificarse unsigned char

Tipos de Datos K&R

- Punteros
  - apuntan a una dirección de memoria
  - unsigned long 128 Mb = 2^27 (hasta 64 Gb)
- Cadenas de caracteres char [20]
  - Cadena de 20 caracteres (20 bytes)
  - Caracteres alfanuméricos (1 byte) -128 a 127 o 0 a 255, depende del compilador
  - Constantes: “Letizia Ortiz”
  - Acaban en ‘\0’ (carácter nulo, ASCII 0)
  - Es preciso un byte más de los que tiene la cadena

Tipos de datos K&R

- float
  - Números en punto flotante (32 bits) +-(1E-37 a 1E38)
  - Precisión de 6 o 7 cifras decimales
- double
  - Doble precisión (64 bits) macro o microcifras
  - Incrementan la precisión de la parte decimal con el uso de 31 bits
  - Pueden aceptar mayores exponentes incorporando algún bit adicional

IEEE 754

- Estándar de punto flotante
- Es el usado habitualmente en los computadores actuales

	Bits	**Sign**	**Exponent**	**Fraction**	**Bias**
Single Precision	32	1 [31]	8 [30-23]	23 [22-00]	127
Double Precision	64	1 [63]	11 [62-52]	52 [51-00]	1023

Tipos de Datos K&R

- Errores de redondeo en punto flotante
- Overflow en punto flotante
  - depende del sistema
    - error
    - se para
    - lo pone a cero
- Limites de los datos en limits.h y float.h

Tipos de Datos K&R

- struct
  - Bloques de datos
    - Por ejemplo: agrupar un int y un float
  - Tamaño ≥ suma de tamaños cada uno de los datos
  - Campos de bits
- union
  - Uniones de datos
    - un int o un float
  - Utilizan los mismos bytes para almacenar cualquiera de los campos
  - Ocupa lo que ocupe el mayor de los datos

Tipos de datos K&R

- Vectores
  - Las cadenas de caracteres son vectores de caracteres
  - Se pueden definir vectores de cualquier tipo de datos de los vistos
    - float [20]
    - unsigned[20]
  - En estos casos no es preciso reservar un elemento de más

Constantes

- Constantes enteras
  - En notación decimal: 5, -5
  - 5L o 5l, si queremos forzar a que lo almacene como long
  - 5U, en algunas implementaciones, si queremos que lo almacene como unsigned
  - En notación octal: 020
  - En notación hexadecimal: 0xFF

Constantes

- Constantes de tipo carácter
  - ‘A’
  - ‘\007’, es lo mismo que ‘\07’ y que ‘\7’
  - O bien ‘\x07’
  - ASCII ‘4’ es ‘\052’
  - Caracteres especiales para la pantalla o la impresora \n, \t, \b, \r, \f, \\, \’, \”.
- Cadenas de caracteres constantes
  - “Fernando”
  - “Hola\007hola”

Constantes

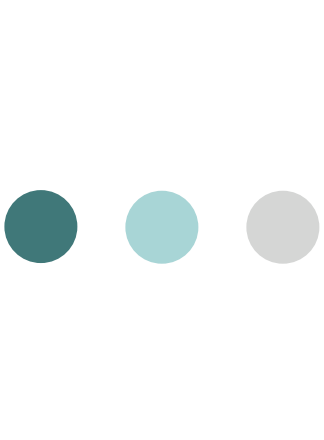
- Constantes en coma flotante
  - 5.0
  - -5.34e-33
  - .2, 1E12 (sin espacios en medio)
  - Las constantes en punto flotante se almacenan en doble precisión
- No son L-values
  - No podremos hacer 5.0 = 33
  - Si podemos hacer x = 33 o x = y

Tipos de Datos ANSI

- void
  - Tipo nulo o vacío
  - Para funciones sin valor de retorno o punteros no inicializados
- const
  - Para variables a las que no se permite modificar su valor
  - Utilidad en vectores pasados como parámetros a funciones

Tipos de Datos ANSI

- volatile
  - Para variables que pueden ser modificadas desde fuera del programa
- enum
  - Para especificar nuevos tipos: rojo, azul y verde
- long double
  - Un nuevo grado de precisión
- signed o signed char
  - Para decir explicitamente que se desea con signo



Operadores

Realizando operaciones con los datos

Tipos de Operadores

- Según el número de operandos
  - Unarios, binarios y ternarios
- Según el tipo de operación
  - Aritméticos
  - Lógicos
  - Relacionales
  - Primarios
  - De bits
  - De asignación

Operadores Aritméticos

+ Adición
- Sustracción
- Signo (unario)
* Multiplicación
/ División
    flotante 10.5/5.0 devuelve 2.1
    entera 7/2 devuelve 3
% Resto (Módulo)
    no funciona en coma flotante

Operadores Aritméticos

++ incremento

- i++ (i=i+1)
- ++i
- 2*++a
- 2*a++

-- decremento

Ojo con los efectos laterales en expresiones y argumentos de funciones

- a*b + c*b++

¡No se pase de listo!

- escribe(x, x*x++)

Operadores Lógicos

Verdadero y falso en C

- Es falso el valor cero
  - 0, 0L, 0x00, 000, '\0', NULL
- Es verdadero cualquier valor distinto de cero
  - 37, -5.8, 0xAF, 'A', 033, 20564204U

Operan verdadero y falso

- && and lógico (no de bits)
- || or lógico
- ! no lógico

Devuelven un 1 si verdadero y un 0 si falso

Operadores Relacionales

Devuelven un 1 si verdadero y un 0 si falso

- < menor que
- > mayor que
- <= mayor o igual que
- >= menor o igual que
- == Igual que
- != distinto que

Operadores Primarios

Acceso a los datos

- * indirección: contenido de la dirección a la que apunta una variable de tipo puntero
- & dirección: en la que se almacena una variable
- () asociación: (a+b)*c
- [] elemento: para acceder a los elementos de un vector
  - en [0] devuelve el primer elemento
  - en [19] devuelve el elemento vigésimo
- . Miembro: acceso al campo de una estructura
- -> miembro indirecto: acceso cuando es un puntero a una estructura

Operadores de Bits

>> desplazamiento a la derecha (x >>2)

<< desplazamiento a la izquierda

& and de bits (ojo distinto de &&)

| or de bits

^ xor de bits

~ negación de bits (unario)

Operadores de Asignación

= asignación

+= asignar suma

x+=5 es x=x+5

i+=1 es i=i+1 es i++

-= asignar resta (ojo con x =- 5)

*= (ojo x *= y, x =* y)

/= asignar división

%= asignar resto

Operadores de Asignación

>>= asignar desplazamiento a la derecha

<<= asignar desplazamiento a la izquierda

&= asignar and de bits

|= asignar or de bits

^= asignar xor de bits

asignar not de bits no es posible, porque es un operador unario

Operadores Varios

, coma (x=5, y=7)

- se evalúa de izquierda a derecha
- El resultado es el de la expresión de la derecha
- En la funciones es un separador
- En las declaraciones de variables también

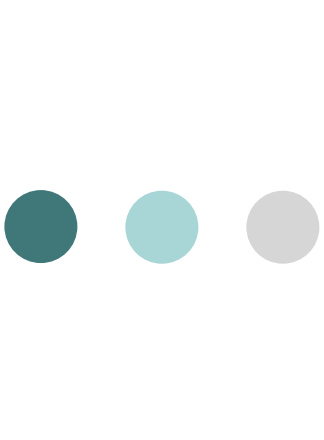
sizeof tamaño (sizeof char da 1 byte)

(*tipo*) operador de moldeado o conversión de tipo

- (int) 5.72 devuelve 5

? : evaluación de condiciones, ternario

- (x<20)? 1: -1
- los paréntesis no son necesarios, pero se recomiendan
- x = x<0? –x : x



Expresiones

Usando los operadores

Expresiones

- Combinación de operadores y operandos (aquello sobre lo que actúa el operador)
- Toda expresión devuelve un valor
  - numérico o lógico (0 falso, otro verdadero)
  - x=7*y tiene por valor el mismo que x
- Ejemplos
  - -4+6
  - c=3+8
  - 5>3
  - 6+(c=3-8)

Expresiones

- La más simple es una constante o variable aislada
  - x
  - 5
- Puede contener subexpresiones
  - a*(b+c/d)/20
- El operando de tipo menor se promueve al de tipo superior
  - 5*2.5

Expresiones

- Conversiones de tipo forzadas
  - 5*(int)2.5
  - x = 3.0 con x entero (el C es permisivo)
- Asociatividad de Asignaciones
  - de derecha a izquierda
  - x^=y^=x^=y

Expresiones

- Tipos de expresiones
  - Aritméticas
  - Lógicas
  - Relacionales
- Aritméticas
  - Operandos numéricos
  - Devuelven un valor numérico
  - x+y

Expresiones

- Lógicas
  - Operandos lógicos
  - Devuelven un valor lógico
  - x&&y
  - x==5 o 5==x da lo mismo (y es distinto de x=5)
  - a<5 && b>7
  - (x&y)==1
  - x devuelve 1 o 0
  - Leyes de DeMorgan
    - (!x && !y) == !(x || y) es verdadero
    - (!x || !y) == !(x && y) es verdadero

Expresiones

- Relacionales
  - Operandos numéricos
  - Devuelven un valor lógico
  - x<y
  - Paradoja aparente: 3<2<1 es verdadero

Precedencia y Asociatividad

- Las expresiones se evalúan en el orden dictado por la precedencia de los operadores
- Primarios
  - () [] -> .
  - de izquierda a derecha
- Unarios
  - ! ~ ++ -- - () * & sizeof
  - de derecha a izquierda

Precedencia y Asociatividad

- Binarios
  - Multiplicación * / %
  - Suma + -
  - Desplazamiento << >>
  - Relación < > <= >=
  - Igualdad == !=
  - Bit and &, bit or | y bit xor ^
  - And && y or ||
  - de izquierda a derecha

Precedencia y Asociatividad

- Ternarios
  - ? :
  - de derecha a izquierda
- Asignación
  - = += -= *= /= %= >>= <<= &= ^= |=
  - de derecha a izquierda
- Coma
  - ,
  - de izquierda a derecha

Precedencia y Asociatividad

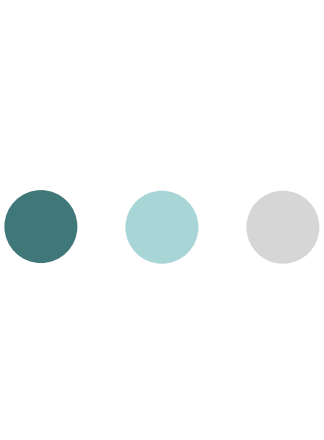
- En a*b/c+d*c
  - se evalúan * y / antes que +
  - a*b/c regla de izquierda a derecha
  - pero no a*b/c antes que d*c
- ojo: a*b+c*(b=b+1) puede dar resultados diferentes dependiendo de qué sumando se evalúe antes (depende del compilador)
- a!=b==c es lo mismo que (a!=b)==c

Precedencia y Asociatividad

- En x=(5+3)*(9+6) no se sabe cual de las dos sumas se hará primero
- Pero se garantiza que las expresiones lógicas se evalúan de izquierda a derecha
  - c=y && c!='0'
  - x!=0 && 12/x==2
  - Acelera el cálculo por evaluación en cortocircuito

Precedencia

Operador	Asociatividad
`()` `[]` `->` `.` `++` *(post)* `--` *(post)*	→
`++` *(pre)* `--` *(pre)* `!` `~` `sizeof` `+` *(unario)* `-` *(unario)* `&` *(dirección)* `*` *(indirección)*	←
`*` `/` `%`	→
`+` `-`	→
`<<` `>>`	→
`<` `<=` `>` `>=`	→
`==` `!=`	→
`&`	→
`^`	→
`|`	→
`&&`	→
`||`	→
`?:`	←
`=` `+=` `-=` `*=` `/=` `%=` `>>=` `<<=` `&=` `^=` `|=`	←
`,`	→



Sentencias

Los ladrillos del programa C

Sentencias

- Sentencias o instrucciones
- Son los elementos con los que se construye un programa
- Acaban en punto y coma: ;
  - X = 4 es una expresión
  - X = 4; es una sentencia

Sentencias

- Deben completar una acción
  - x=6+(y=5); es una sentencia
  - 5; es una sentencia
- ojo = es un operador, no es una sentencia
  - LET x=5 es una sentencia en BASIC
  - x:=5 es una sentencia en Pascal
  - x=5 es una expresión de asignación en C
  - x=5; es una sentencia en C

Tipos de Sentencias

- De declaración: int x;
- De asignación: x=12;
- De función: evalua(x, y);
- De control: if, for
- Nula: ;

- Bloques: { x=5; y=7; }
  - Técnicamente, no son sentencias.

Sentencias de Declaración

- Identificadores
  - Cualquier número de caracteres
  - 31 caracteres son significativos (depende del compilador)
  - Letras, números y “_”
  - El primero debe ser una letra o “_”

- Costumbres y recomendaciones
  - Que tenga significado en el contexto del programa
  - Minúsculas (salvo iniciales)
  - Las variables de biblioteca suelen empezar por _

Sentencias de Declaración

- Declaración de variables
  - En otros lenguajes no se declaran explícitamente
  - P.e. en FORTRAN se hace sobre la marcha y tienen un tipo por defecto según el nombre
  - LOMO y L0M0 son diferentes
  - int z=5; es una declaración y una asignación
  - Hasta ahora declarar equivale a definir
  - En las constantes se presume notación decimal por defecto

Sentencias de Declaración

- Declaraciones múltiples
  - Operador coma (separador)
  - int x, y;
  - int x=4, y;
- Declaración de caracteres
  - char x = 65; es lo mismo que char x = 'A';
  - No es lo mismo A que 'A' que "A"
  - char x = 7; es lo mismo que char x = '\007';

Sentencias de Declaración

- Declaración de vectores
- Se indica el numero de elementos del tipo base
  - int z[10]; // Diez enteros
  - int z [3][3]; // 3x3 enteros
- Se puede inicializar:
  - int z[10] = {234,345,333,33};
  - Diez enteros
  - Cuatro inicializados

Sentencias de Declaración

- Cadenas de caracteres
  - char x[] = “Fernando”; es char x[9] = “Fernando”;
- Otra posibilidad es:
  - char x[9];
  - x[0]=‘F’;
  - x[1]=‘e’;

  ...
  - x[7]=‘o’;
  - x[8]=‘\0’;

Sentencias de Declaración

- Números en punto flotante
  - float x;
  - double x = 3.0;

Sentencias de Declaración

- Estructuras
  - struct complejo {float a; float b;};
  - struct complejo {float a; float b;} x, y;
  - El nombre es opcional
- La primera declaración declara un tipo nuevo
  - struct complejo {float a; float b;};
  - struct complejo x;
- Se pueden inicializar
  - struct complejo x = {3.0, 4};
- Son posibles las estructuras anidadas

Sentencias de Declaración

- Uniones
  - union dato {float peso; int altura;};
- Campos de bits
  - struct puerto {int a : 2; int b : 4;} x;
  - x.a=3; cabe en a
  - x.a=34; no cabe (solo bits inferiores)

Sentencias de Declaración

- Punteros
  - int *p;
  - float *q;
  - Punteros a estructuras: struct complejo *p;
- Inicialización de punteros
  - int x, *p; p=&x;
- Punteros a estructuras
  - struct punto {int a, b;} x, *p;
  - x.a = 7;
  - p = &x;
  - p->a = 7;

Nuevos Tipos de Datos

- Definición de nuevos tipos
  - typedef
- typedef int boolean;
  - Declaración de un nuevo tipo “boolean”
  - Idéntico a “int”
    - boolean pepe;
    - int pepe;

Sentencias Condicionales

- Sintaxis
  - if (expresion) sentencia else sentencia
  - if (expresion) sentencia
- Condiciones anidadas
  - if (expresión) sentencia else if (expresión) sentencia else sentencia
  - If (expresión) if (expresión) sentencia else sentencia (else va con el if mas cercano)

Sentencias Condicionales

- Expresiones lógicas de comparación
  - != >= <= > < ==
- Cuidado con la comparación/asignación
  - Basic (LET = o =)
  - Pascal (:= o =)
  - FORTRAN (= o .EQ.)
  - C (= o ==)
  - if(5==x) y++;

Sentencias Condicionales

- Verdadero y falso en C
  - if (0) x++;
  - if (–33) x++;
  - 0, 0L, 0.0, '\0', NULL
  - if (!x)
  - if (x==0)
  - if (p==NULL)

Sentencias Condicionales

- Prioridad y asociatividad
  - if(x>y+2) y=x; es if(x>(y+2)) y=x;
  - if(x>0 || x<10) y=x; no hace falta paréntesis
  - En z>b && b>c || b>d, && antecede a ||
  - y=(x<10)?1:-1; es if(x<10) y=1;else y=-1;
- Flexibilidad
  - if(x!=7) z--; y=x; es if(y=x!=7) z--;

Sentencias Condicionales

- Elección múltiple
  - switch (expresión) {
    case constante: sentencias
    case constante: sentencias
    default: sentencias
    }
  - default es opcional
  - La sentencia break sale de la última sentencia condicional y se dirige a la siguiente sentencia inmediatamente situada tras la misma

Sentencias Condicionales

- switch (c) {
  case ‘A’:
  case ‘a’: sentencias
  break;
  case ‘B’: sentencias
  break;
  default: sentencias
  }
- break es opcional, pero ojo con olvidarse alguno
- switch es más eficiente que if, pero no siempre puede emplearse if(x<1000 && x>2)

Bucles

- while
  - Sintaxis
    - while(expresión) sentencia
  - Ejemplo: calcula x elevado a n
    - while (--n) x*=x;
  - Bucle infinito
    - while (1) sentencia
  - Sentencias compuestas
    - n--; while (n) {
      x*=x;
      n--;
      }

Bucles

- for
  - Sintaxis
    - for(expresión; expresión; expresión) sentencia
    - for(i=1; i<n; i++) x*=x;
  - Expresiones: inicial, condición, cada vez
  - Mayor flexibilidad que en otros lenguajes
    - for(i=n-1; i; i--) x*=x;
    - for(i=0; i<3*n; i+=3) x*=x;
    - for(c='A'; c<'Z'; c++, i++) x[i]=c;
    - El operador coma (se evalúa de izquierda a derecha)

Bucles

- El bucle típico comienza en 0 y acaba en n-1
- Cualquier expresión es legal
- for(int i=0; i<n; i++) { x*=2; y+=i }
- Dejar alguna de las tres en blanco
- for (;;) sentencia es un bucle infinito
- for(i=0; i<n; i++) x*=2;
- La primera expresión puede hacer cualquier cosa

- do while
  - do sentencia while (expresión);
  - con condición de salida, al contrario que for y while

Bucles

- Cuál elegir?
  - Normalmente (95%) se precisa condición de entrada
  - for permite inicializar, comparar y actualizar en una sola sentencia
  - Si no hacen falta las tres cosas, el while es más simple
- Bucles anidados

Otras Sentencias de Control

- break
  - Sale del último switch, for, while o do while, y dirigiéndose a la siguiente sentencia
  - Para abandonar un bucle
    - while (n--) {
      if(n<5) break;
      y+=n;
      }
    - while (--n<5) y+=n; (a veces se puede evitar)
  - Al usarla en situaciones anidadas, sólo afecta a la sentencia de control más interna

Otras Sentencias de Control

- continue
  - Se salta el resto del bucle for, while o do while, se continua en la siguiente iteración
    - while (n--) {
      if(n>=5) continue;
      y+=n;
      }
  - No se utiliza en el switch
  - Frecuentemente, invirtiendo la expresión del if, se evita el uso de continue

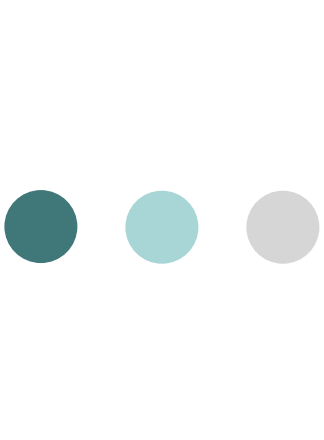
Otras Sentencias de Control

- goto
  - Es la maldición del BASIC y el FORTRAN
  - Sintaxis
    - goto etiqueta;
    - etiqueta:
  - Es prácticamente innecesaria si se saben emplear bien todas las sentencias de control anteriores
  - Crea nefastos hábitos de programación
  - Los buenos programadores lo usan para salir de varios bucles anidados en caso de error

Lectura Recomendada

- **Go To Statement Considered Harmful**
  Edsger W. Dijkstra

  Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148.



Preprocesador C

Antes de compilar

Comentarios

- Sintaxis
  - /*cualquier texto*/
- En cualquier lugar
  - x=/* 5 */ 7;
- Dentro de una cadena de caracteres no tienen efecto
  - char [] =“Fern/*a*/ndo”
- No se pueden anidar
  - /* x=5; /* y+=x; */ x++; */

Macros

- No son lo mismo que las constantes del programa
  - #define PI 3.1416
  - x = 2*PI;
- El nombre sigue las normas de las variables
- En mayúsculas por convención
- No es válido PI=44;
- Pascalizando C

```
#define begin {
#define end }
```

Macros

- #define PI 3.1416 no tiene efecto
  - dentro de una cadena “AGAPITAS”
  - dentro de un token PITAS
- Para el nuevo tipo boolean
  - #define TRUE 1
  - #define FALSE 0
- Puede continuarse en la siguiente línea con \
- Ojo con los nombres de las variables y las funciones

Macros

- Pueden llevar argumentos
  - #define CUADRADO(x) x*x
    - x = 3*CUADRADO(2);
    - se sustituye por x = 3*2*2;
    - x = 3*CUADRADO(y);
    - se sustituye por x = 3*y*y;
  - #define PR(y) escribe("hay", y)
    - PR(3);
    - se sustituye por escribe("ha3", 3);
    - PR(CUADRADO(2));
    - se sustituye por escribe("haCUADRADO(2)", 2*2);

Macros vs Funciones

- Macros y funciones no son equivalentes
- Las macros se expanden en el texto
- Multiplican el código por el número de apariciones

Ficheros de Encabezamiento

- Sintaxis
  - #include “fichero.h”
  - #include <fichero.h>
- <> Permite buscar en directorios por defecto
  - /usr/include, /usr/local/include, /opt/local/include
- El objetivo es incluir el fichero en varios ficheros C

Ficheros de Encabezamiento

- Puede contener
  - Definición de macros
  - Definición de tipos
  - Prototipos de funciones
  - Inclusión de otras ficheros de encabezamiento
- No deben (suelen) contener
  - Código C (sentencias)
  - Declaración de variables
  - Implementación de funciones
- Útiles para compartir definiciones

Otras Directivas

- #ifdef, #else, #endif, #ifndef, #elif, #undef
  - Compilación condicional

    #define Linux definición vacía

    #ifdef Linux

    ...

    #else

    ...

    #endif
  - o con #ifndef Linux o #elif
  - #undef Linux

Otras Directivas

- Util para no incluir dos veces un fichero de encabezamiento

```
#ifndef _FICHERO_H_
#define _FICHERO_H_
...
#endif
```

- #if útil para comentarios provisionales

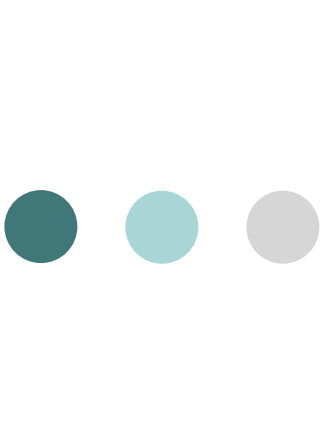
```
#if 0
...
#endif
```

Otras Directivas

- #pragma palabra reservada
  - directivas propias de cada compilador
  - se ignora en caso de no ser reconocida

Pasos de Compilación

- Un solo comando
  - gcc -c programa.c
- El preprocesador
  - Elimina los comentarios
  - Ejecuta las directivas del preprocesador
- El compilador
  - Analizador sintáctico
  - Traducción a código máquina



Vectores y punteros

La sal y la pimienta de C

Vectores

- Como las cadenas de caracteres
  - int x[] = {4, 2 ,7 ,9 } ; es int x[4] = {4, 2, 7, 9};
  - Es aplicable a cualquier otro tipo
  - Otra posibilidad es:
    - int x[4];
    - x[0]=4;
    - x[1]=2;
    - x[2]=7;
    - x[3]=9;
  - Se almacenan en memoria consecutivamente

Punteros y Vectores

- Apuntar al primer elemento del vector
  - int x[4] = {4, 2, 7, 9};
  - int *p;
  - p = x; es p = &x[0];
- O a cualquiera de ellos
  - p = x; p++; es p = &x[1];
  - p++; aumenta p sizeof int bytes
  - p+=2; aumenta p 2*sizeof int bytes
  - *(p+2) devuelve el valor de x[2]
- Tipo base = tipo al que apunta p

Operaciones con Punteros

- Es posible prácticamente cualquier operación con punteros en expresiones
  - p++, p--
  - p == q
  - p<=q
  - p - q no en bytes, en unidades del tipo apuntado
  - Teniendo en cuenta que se entán comparando direcciones de memoria (unsigned long)
- Un error habitual consiste en no inicializar el puntero y tratar de acceder a *p

Matrices

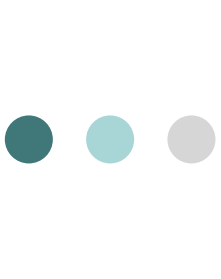
- Vectores de vectores
  - int x[3][2] = { {4, 1} , {7, 9}, {0, 8} };
  - O bien:
    - int x[0] [0] = 4;
    - int x[0] [1] = 1;
    - int x[1] [0] = 7;
    - int x[1] [1] = 9;
    - int x[2] [0] = 0;
    - int x[2] [1] = 8;

Matrices y Punteros

- Se apunta al primer elemento de la matriz
  - int x[3][2] = { {4, 1} , {7, 9}, {0. 8} };
  - int (*p)[2]; que no es int *p[2];
  - p = x; es p = &x[0]; o bien p = &x[0][0]
- O a cualquiera de ellos
  - p = x; p++; es p = &x[1]; o bien p = &x[1][0]
  - (*p)[1]; devuelve x[1][1];
  - p = x[2]; es p = &x[2]; o bien p = &x[2][0]
  - p++; aumenta p 2*sizeof int bytes
  - p+=2; aumenta p 4*sizeof int bytes

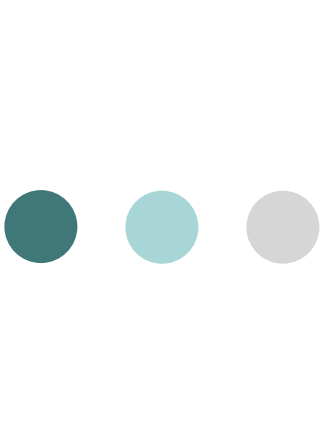
Vectores de Estructuras

- Se definen de manera análoga
  - struct complejo { int a; int b; };
  - struct complejo x[3], *p = x;
  - x[0].a, x[0].b;
  - x[1].a, x[1].b;
  - x[2].a, x[2].b;
- Punteros a vectores de estructuras
  - (p+1)->a devuelve x[1].a
  - (p+3)->a devuelve cualquier cosa



Algoritmos

- Utilidad de vectores y punteros
  - Ordenación de números
  - Listas
  - Árboles



Funciones

Agrupaciones de codigo en C

¿Qué es una función?

- Un trozo de código reutilizable
- Una función puede llamar a otras funciones
  - De hecho todo empieza en main()
- Hay que tener cuidado con los problemas de recursión

Definición de Funciones

- Dos partes:
  - Firma
  - Cuerpo
  - int suma(int a, int b) { return a+b;}
- Firma:
  - Nombre
  - Valor de retorno
  - Argumentos

Nombre de Función

- Sigue las reglas de identificadores
- Un nombre claro es conveniente
- Habitualmente se usan minúsculas
- Problemas de nombres repetidos
  - Funciones de librería
  - Uso de prefijos

Ejemplos de Firmas

- Ejemplo sin argumentos
  - compara()
  - void compara(void)
  - Si no se indica el tipo se presupone int
- Ejemplo con argumentos
  - void compara(int x, int y)

Definición de Funciones

- Prototipo de una función
  - void compara(void);
  - void compara(int, int);

Tipo de una Función

- En Pascal hay funciones y procedimientos.
  En C sólo hay funciones
- Tipo del dato devuelto

```
void compara(int x, int y) {
        ...
        return;   no es imprescindible
        }
int compara(int x, int y) {
        int z;
        ...
        return z; es la forma de devolver un int
        }
```

El Programa Principal

- El programa principal es una función más, aunque especial por ser la primera

```
void main(int argc, char *argv[]) {...}
```

- Devuelve un valor entero

```
int main() {
    int x;
    ...
    return x;
    }
```

Argumentos de Funciones

- Los argumentos en C se pasan por valor
  - int escribe(int);
  - escribe(26); o escribe(13*2); o escribe(x);
- Se declaran los argumentos
  - int escribe(int x) {} o int escribe(x) int x; {}
- Se comprueban los tipos de argumentos (no K&R)

Argumentos de Funciones

- Los vectores se pasan por valor del puntero que los señala (= por dirección)
  - float x[20];
  - ordena(x);
  - ordena(float *y) { ... usa *y ... };
- Se pueden usar punteros para devolver valores por dirección
  - int x;
  - dime(&x);
  - dime(int *x) { ... usa *x ... };

Argumentos Variables

- Funciones con argumentos variables
- Un caso típico es el programa principal
  - void main(int argc, char **argv);
  - argc entero que indica el número de argumentos
  - argv[0] cadena que indica el nombre del programa
  - argv[1] cadena que indica el primer argumento
  - ...
  - argv[argc] cadena que indica el último argumento

Prototipos y Librerías

- Declaración y definición de funciones
  - Se pueden definir en cualquier orden
  - Se declaran con el prototipo
- Utilidad del prototipo
  - Es opcional si se ordenan adecuadamente
  - Permite el chequeo de tipo y argumentos
  - Util cuando hay varios ficheros
  - Util en librerías de funciones
  - Se declaran en el fichero de encabezamiento

Alcance y modo

- Dos aspectos a tener en cuenta en las variables
- Alcance
  - Local
  - Fichero
  - Global
- Modo
  - Automático (auto)
  - Estático (static)
  - Registro (register)
  - Volátil (volatile)

Alcance y Modo

- Usables en el interior del bloque en el que se encuentran

```
main () {
    int x = 6;
    if (y>7) {
        int x;
        x = 5;
        }
    }
```

- Modo:
  - Las variables son de tipo auto (automáticas) por defecto

Alcance

- Alcance dentro de un fichero
  - Variables locales a cada función
  - Variables globales a todas las funciones

```
int x;
escribe () {
    int x;
    x = 5;
    }
```

Variables estáticas

- Declaración
  - static int x;
- Dentro de una función
- Proporcionan almacenamiento permanenente
- Uso en funciones recursivas

Alcance entre varios ficheros

- Variables externas

```
extern int x;
escribe () {
        x = 5;
        }
```

  - es una declaración, pero no una definición
  - definida en otro fichero
- Uso de static en variables globales
  - static int x;
  - variable global a un fichero
  - evita que sean accesibles desde otros ficheros

Variables de tipo registro

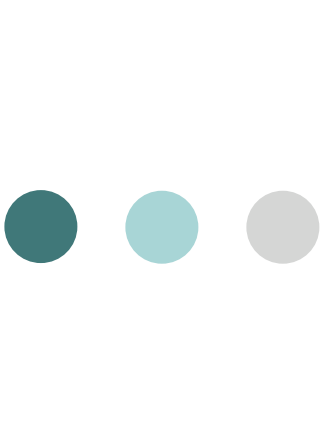
- register int c;
- register char c;
- El compilador procura almacenarlas en registros de la CPU
- Sólo pueden aplicarse a variables automáticas y parámetros formales de una función
- Pueden incluir inicialización

Funciones y Macros

- ¿Macros o Funciones?
  - Las macros pueden inducir errores
  - Las macros de más de una línea deben ser evitadas
  - Es una diferencia entre espacio y tiempo
  - Una macro funciona con distintos tipos de argumentos (enteros o en coma flotante)

Funciones “in Line”

- Permiten compactar el código de las funciones
  - inline float absoluto(float x) { return x>0? x : -x; }
- Se recomienda al compilador sustituir literalmente la llamada a la función por el código correspondiente
- Ventajas
  - Código más rápido, se evita la llamada
  - Código más grande, aparece repetido
- Se usa en funciones muy pequeñas



C Library

La biblioteca de funciones C

Propósito

- Uno de los objetivos del ANSI C fue definir una serie de funciones que fueran comunes a cualquier implementación del lenguaje
- libc.a (se enlaza por defecto)
- libm.a (matemáticas)
- en /usr/lib
- gcc -o programa programa.o -lc -lm

Entrada / Salida

- Prototipos en <stdio.h>
  - printf("Hola %d\n", x);
    - %d, %o, %x (int)
    - %u, %ld, y en algunos sistemas %lu (unsigned)
    - %c, %s (char y char[])
    - caracteres especiales
    - %f (float, en notación decimal)
    - %e (float, en notación exponencial)
    - %g (float, el que sea más corto)

Entrada / Salida

- Modificadores de especificaciones de conversión
  - %4d anchura mínima
  - %3.2f precisión
  - %-10d se pega a la izda y no a la dcha
- Ejemplos
  - printf("%u\n", -336) presenta 65200
  - printf("%c %d\n", 'A', 'A'); presenta A 65
  - printf("%*d\n", ancho, x);

Entrada / Salida

- scanf("%d", &x);
  - No es preciso & para cadenas de caracteres
  - No existe %g
  - Son equivalentes %f y %e
  - Se pueden leer enteros short con %h
  - scanf("%*d %*d %d", x); se salta los dos primeros enteros
- printf() y scanf() devuelven un valor: el numero de datos impresos o leidos

Entrada / Salida

- ch = getchar();
- putchar(ch);
- putchar(getchar());
- Fin de la entrada CTRL-Z o ^Z
- EOF en stdio.h
- while ((ch=getchar())!= EOF) e.g. ^d
- Redirección de ficheros en unix < >
- gets()

Cadenas de Caracteres

- Prototipos en <string.h>
  - Asignación
    - char x[9];
    - strcpy(x, “Fernando”);
  - strcpy(cadena1, cadena2);
  - strlen(cadena);
  - strcat(cadena1, cadena2);
  - Carácter nulo
  - stoi();
  - atoi();

Acceso a Ficheros

- Prototipos en <stdio.h>
  - FILE *fp;
  - fp = fopen(fp, “nombre”)
  - fprintf(fp, “”)
  - fscanf(fp, “”)
  - fclose(fp)
  - Devuelven un valor

Puertos E/S

- En algunos compiladores
  - valor = inp(puerto);
  - outp(puerto, valor);

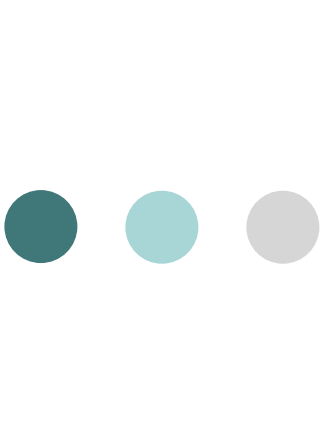
Gestión de Memoria

- Prototipos en <stdlib.h>
  - float *p;
  - p = malloc(sizeof(float));
  - ...
  - free(p);

Resumen

¿Qué hemos visto?

- El lenguaje ANSI C



Ejercicios

Ejercicio 1

- Escribir un programa C que imprima en pantalla el mensaje “Hello World”

Ejercicio 2

- Escribir un programa para hacer doble eco del teclado hasta pulsar <Crtl>-D

Ejercicio 3

- Escribir un programa que imprima en pantalla lo siguiente:

```
*
**
***
****
*****
******
*******
```

Ejercicio 4

- Escribir un programa que vuelque en pantalla el contenido de un fichero cuyo nombre se pasa como primer argumento del programa