Estructuras de Datos

Repaso del lenguaje de programación C

Repaso de conceptos básicos de C (I)

- Tipos de datos
  - Tipos de datos primitivos: `char`, `int`, `float`, `double`...
  - Arrays y punteros
  - Estructuras de datos (`struct`), definición de tipos (`typedef`) y enumeraciones (`enum`)
- Operaciones básicas
  - Operadores aritméticos: `+`, `-`, `*`, `/`...
  - Operadores relacionales: `<`, `>`, `==`, `!=`...
  - Operadores lógicos: `&&`, `||`, `!`
  - Control del flujo de ejecución: `if`, `else`, `for`, `while`...
- Reglas de precedencia y asociatividad de operadores
- Entrada/salida básica: `printf`, `scanf`, `gets`, `puts`, `fopen`, `fclose`, `fscanf`, `fprintf`...

Repaso de conceptos básicos de C (II)

Tipos de datos primitivos

Tipo de dato	Memoria (bytes)	Rango	Formato
`short int`	2	$-32\,768$ a $32\,767$	`%hd`
`unsigned short int`	2	0 a 65 535	`%hu`
`unsigned int`	4	0 a 4 294 967 295	`%u`
`int`	4	$-2\,147\,483\,648$ a $2\,147\,483\,647$	`%d`
`long int`	4	$-2\,147\,483\,648$ a $2\,147\,483\,647$	`%ld`
`unsigned long int`	4	0 a 4 294 967 295	`%lu`
`long long int`	8	-2^{63} a $2^{63}-1$	`%lld`
`unsigned long long int`	8	0 a 18 446 744 073 709 551 615	`%llu`
`signed char`	1	-128 a 127	`%c`
`unsigned char`	1	0 a 255	`%c`
`float`	4	$1{,}2 \times 10^{-38}$ a $3{,}4 \times 10^{38}$	`%f`
`double`	8	$1{,}7 \times 10^{-308}$ a $1{,}7 \times 10^{308}$	`%lf`
`long double`	16	$3{,}4 \times 10^{-4932}$ a $1{,}1 \times 10^{4932}$	`%Lf`

Extraído de `https://www.geeksforgeeks.org/data-types-in-c/`

Funciones (I)

Cabecera de la función:

```
[tipo] nombre ([tipo1 [arg1]] [, resto_de_args]);
```

- `tipo`: (opcional) tipo del retorno, `int` por defecto
- `nombre`: nombre de la función
- `tipo1`: (opcional) tipo del primer argumento
- `arg1`: (opcional) nombre del primer argumento

Implementación (cuerpo de la función):

```
[tipo] nombre ([tipo1 [arg1]] [, resto_de_args]) {
    ...
}
```

Funciones (II)

Ejemplo: Función que recibe 2 enteros y devuelve su suma

Cabecera de la función

```
int sum(int x, int y);
```

Cuerpo de la función

```
int sum(int x, int y) {
    return x+y;
}
```

Funciones (III)

Paso por valor:

- Los valores de los argumentos se copian en variables locales

Ejemplo: Función *swap*

Incorrecto

```
void swap(int a, int b) {
    int aux;

    aux = a;
    a = b;
    b = aux;
}
```

Correcto

```
void swap(int *a, int *b) {
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}
```

Punteros (I)

Un **puntero** es una variable que almacena la dirección de memoria de un dato (de un cierto tipo)

```
int i = 1;
int *pi;

pi = &i;
(*pi)++;
printf("%d", i);
printf("%d", *pi);
```

Punteros (II)

Operadores sobre punteros:

- Operador de dirección `&`: devuelve la dirección de memoria de una variable

```
pi = &i; // Asigna la dirección de la variable i al puntero pi.
```

- Operador de indirección o desreferencia `*`: devuelve el valor almacenado en la dirección a la que apunta el puntero

```
(*pi)++; // Incrementa el valor almacenado en la dirección a la que apunta pi.
```

Punteros (III)

Ejemplos: ¿Cuál es la salida de los siguientes programas?

Programa 1

```
int a, *pa;
a = 3;
pa = &a;
*pa = 4;
a = 5;
printf("%d", *pa);
```

Programa 2

```
int a, *pa , **ppa;
a = 3;
ppa = &pa;
*ppa = &a;
*pa = 4;
printf("%d", a);
```

Programa 3

```
int a, *pa , **ppa;
a = 3;
ppa = &pa;
*ppa = a;
*pa = 4;
printf("%d", a);
```

Arrays (I)

Arrays unidimensionales:

```
int t[10];
t[2] = 4;
```

Arrays multidimensionales:

```
int mat[10][10];
mat[1][1] = 4;
```

El nombre del array es un puntero constante que almacena la dirección del primer elemento del array:

```
int x[10];
*x = 4; // Equivalente a x[0] = 4.
```

Arrays (II)

Arrays y punteros:

```
int n[10];
double d[10];
int *p;
double *pd;

// Asignación correcta (n contiene la dirección de un entero):
p = n;

// Asignación incorrecta (n es constante, no se puede modificar):
n = p;

// Acceso a los datos, aritmética de punteros:
*(p + 1) = 2;
p[1] = 3;
pd = d;
*(pd + 3) = 2.5;
```

Estructuras de datos (I)

Ejemplo: Definición de una estructura para almacenar un número complejo

Opción A

```
struct{
    float re, im;
} c;
```

Opción B

```
struct _ComplexNumber{
    float re, im;
};

struct _ComplexNumber c;
```

Opción C

```
struct _ComplexNumber{
    float re, im;
};

typedef struct _ComplexNumber
                ComplexNumber;
ComplexNumber c;
```

Opción D

```
typedef struct {
    float re, im;
} ComplexNumber;

ComplexNumber c;
```

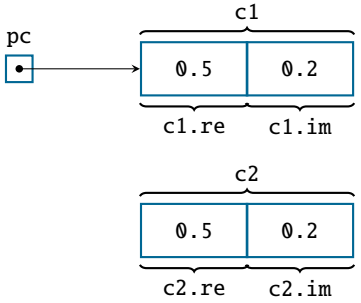
Estructuras de datos (II)

Operador de asignación:

```
typedef struct {
    float re, im;
} ComplexNumber;

ComplexNumber c1, c2;
ComplexNumber *pc;
pc = &c1;

c1.re = 0.5; // pc->re = 0.5;
c1.im = 0.2; // pc->im = 0.2;
c2 = c1;
```



Estructuras de datos (III)

Arrays de estructuras:

- Semejantes a los arrays de tipos de datos primitivos

```
ComplexNumber ca[10];
```

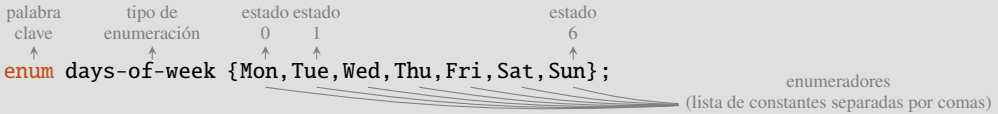
Entrada/Salida:

- Siempre campo a campo

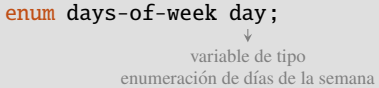
```
ComplexNumber c;
scanf("%f %f", &(c.re), &(c.im));
```

Enumeraciones

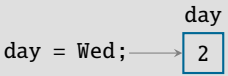
Declaración



Instancia



Operación



Extraído de https://www.geeksforgeeks.org/enumeration-enum-c/

Reglas de precedencia y asociatividad (I)

Operador	Descripción	Asociatividad
`()`	Agrupamiento/llamada a función	Izquierda a derecha
`[]`	Corchetes (indexado de arrays)	
`.`	Selección de campo desde nombre	
`->`	Selección de campo desde puntero	
`++`, `--`	Incremento/decremento postfijo	
`++`, `--`	Incremento/decremento prefijo	Derecha a izquierda
`+`, `-`	Más/menos unario	
`!`, `~`	Negación/complemento por bits	
`(tipo)`	Casting	
`*`	Desreferencia	
`&`	Dirección	
`sizeof`	Tamaño	
`*`, `/`, `%`	Multiplicación/división/módulo	Izquierda a derecha
`+`, `-`,	Suma/resta	Izquierda a derecha
`<<`, `>>`,	Desplazamiento de bits a izquierda/derecha	Izquierda a derecha
`<`, `<=`, `>`, `>=`	Comparaciones de desigualdad	Izquierda a derecha
`=`, `!=`	Comparaciones de igualdad	Izquierda a derecha

...

Extraído de `http://web.cse.ohio-state.edu/~babic.1/COperatorPrecedenceTable.pdf`

Reglas de precedencia y asociatividad (II)

...

&	AND por bits	Izquierda a derecha
^	XOR por bits	Izquierda a derecha
|	OR por bits	Izquierda a derecha
&&	AND lógico	Izquierda a derecha
||	OR lógico	Izquierda a derecha
?:	Condicional ternaria	Derecha a izquierda
=	Asignación	Derecha a izquierda
+=, -=, *=...	Asignación con operación	
,	Separación de expresiones	Izquierda a derecha

Ejemplo 1

```
int a;
a = 5 + 3 * 4 % 5 - 1;
```

Ejemplo 2

```
int a, b;
b = a = (5 + 3) * 4 / 5 || 1;
```

Organización del código (I)

Fichero único:

1. Órdenes para el preprocesador: `#include`, `#define`
2. Estructuras de datos y definición de tipos: `struct`, `typedef`
3. Prototipos de las funciones
4. Variables globales
5. Función `main`
6. Definición de las funciones

Organización del código (II)

Múltiples ficheros:

- Biblioteca:
  - Fichero de cabecera (`library.h`): Definición de TAD, prototipos de primitivas y funciones específicas
  - Fichero fuente (`library.c`): Definición de EdD, cuerpo de primitivas y funciones basadas en esas EdD
- Fichero `main.c`:
  - Función `main`
  - Incluye el fichero de cabecera: `#include "library.h"`
  - Usa únicamente las funciones en el fichero `library.h`, no accede directamente a las EdD
- Compilación y enlazado: `Makefile`

Variables y su alcance

Alcance de una variable: Parte del código desde donde la variable puede ser accedida
Variables globales:

- Su alcance es todo el fichero
- Se almacenan en el segmento de datos

Variables locales:

- Su alcance es el bloque o función donde se declaran
- Se almacenan en la pila, se destruyen cuando se sale del bloque o función

Variables estáticas:

- Variables locales cuyo valor persiste entre llamadas a la función, se almacenan en el segmento de datos

Reserva de memoria

Reserva de memoria: `malloc`

```
int *p;
p = (int *)malloc(10 * sizeof(int));
```

Realojo de memoria: `realloc`

```
p = (int *)realloc(p, 20 * sizeof(int));
```

Liberación de memoria: `free`

```
free(p);
```

Ejercicios (I)

- Escribir el código C que lee (de entrada estándar) un número entero `n` y reserva memoria para un array de `n` enteros
- Escribir el código C que lee (de entrada estándar) dos números enteros `n`, `m` y reserva memoria para un array bidimensional de `n` filas y `m` columnas
- Escribir el código C para liberar toda la memoria reservada en los dos puntos anteriores

Ejercicios (II)

Escribir una función `main` que:

- Declare un array de 10 números de precisión doble en coma flotante
- Inicialice los 10 elementos del array a 0
- Llame a la función auxiliar `readData`, que lee los elementos del array por entrada estándar
- Llame a la función auxiliar `computeStatistics`, que calcula el máximo, mínimo y el valor medio de los elementos del array
- Imprima estos valores por salida estándar

Los prototipos de las funciones auxiliares son:

```
int readData(double *x);

int computeStatistics(const double *x, double *min, double *max, double *ave);
```