Introducción a la programación

Bibliografía:

- Diapositivas de teoría y guiones de prácticas, Introducción a la Programación, Universidad de Murcia (anónimo).

¿Qué tareas puede realizar un ordenador?

Ordenador: Mecanismo digital de propósito general.

- **Hardware:** Le das patadas.
- **Software:** La razón por la que das las patadas.

El ordenador procesa y ejecuta algoritmos. Un **algoritmo** es una secuencia de instrucciones **finita** (termina en algún momento), **definida** (totalmente inambigua), **efectiva** (se puede hacer sólo con papel y lápiz) y **general** (funciona con casos diferentes del problema a tratar y no solo con uno en particular), y que tiene una serie de **entradas** (parámetros, eventos, etc., incluyendo números pseudoaleatorios), **salidas** (resultados) y una determinada **eficiencia** (los recursos que consume, esp. en función del tamaño del problema).

Los problemas se clasifican en **computables**, cuando existe un algoritmo que los resuelve, y **no computables**. Los computables a la vez se dividen en **tratables**, si hay un algoritmo eficiente que lo resuelve, o **intratables** si no, entre los que destacan los NP-completos (no polinomiales).

El ordenador trabaja con **instrucciones máquina**, muy sencillas, y a través del código binario. El bit (valor 0 o 1) es la unidad mínima de información. Los **lenguajes de programación** son una notación de las operaciones a realizar por el ordenador con un nivel de abstracción mayor al lenguaje máquina. El código escrito en estos como representación del algoritmo es convertido al lenguaje máquina por el **compilador** para su posterior ejecución o bien es ejecutado por un **intérprete**, que junta traducción y ejecución.

IMPORTANTE: A la hora de escribir un programa o algoritmo en exámenes o prácticas a entregar, **no usar NUNCA** `return` **a mitad ni tampoco usar** `goto`.

Capítulo 1

Léxico y organización de un algoritmo

Lenguaje algorítmico	Pascal

LÉXICO
v_1: $tipo_1$;
v_2, v_3 : $tipo_2$;
A_1: una acción
PRE { *precondición* A_1 }
POST { *postcondición* A_1 }
LÉXICO
...
ALGORITMO
...
FIN
f_1: función
PRE { *precondición* f_1 }
POST { *postcondición* f_1 }
LÉXICO
...
ALGORITMO
...
FIN
...
ALGORITMO
PRE { *precondición algoritmo* }
POST { *postcondición algoritmo* }
...
FIN
	`program` *nombre*;
	`type`
	$tipo_1$ = ...;
	$tipo_2$ = ...;
	...
	`var`
	v_1 : $tipo_1$;
	v_2, v_3 : $tipo_2$;
	...
	`procedure` A_1(...):
	`var`
	...
	`begin`
	...
	`end;`
	`function` f_1(...):
	`var`
	...
	`begin`
	...
	`end;`
	`var`
	...
	`begin`
	...
	`end.`

La **notación algorítmica** establece la forma de describir las acciones e informaciones y organizarlas en el tiempo, mientras que el código del programa establece la forma en la que estas se implementan de forma interpretable por un ordenador. Está formado por un **léxico,** donde se definen las informaciones u objetos y las acciones, y el **control,** que establece cómo actúan las acciones sobre los objetos. La **abstracción** sirve para dominar la complejidad de un programa.

1.1. Tipos de datos y operaciones primitivos

Especifican un dominio de valores y un conjunto de operaciones aplicables. Los tipos primitivos son:

- **Entero** (`integer`).
- **Real:** Se usa el «.» como separador (`real`).
- **Booleano:** Valores «Verdadero» y «Falso» (`boolean`, `true`, `false`).
- **Carácter:** Se ponen entre comillas simples (`char`).

Las operaciones primitivas son:

- $-$: Entero→Entero, Real→Real
- $+$, $-$, $*$: Entero×Entero→Entero, Real×Real→Real
- /:Entero×Entero→Real, Real×Real→Real
- *DIV*, *MOD*: Entero×Entero→Entero
- $<$, $>$, $=$, $\leq$ (`<=`), $\geq$ (`>=`), $\neq$ (`<>`): Entero×Entero→Booleano, Real×Real→Booleano, Carácter×Carácter→Booleano
- *Predecesor*, *Sucesor*: Entero→Entero, Carácter→Carácter
- *Y*, *O*, *YDESPUÉS* (`and`), *ODESPUÉS* (`or`): Booleano×Booleano→Booleano
- *NO* (`not`): Booleano→Booleano
- *Car* (`chr`): Entero→Carácter
- *Ord* (`ord`): Carácter→Entero

Las acciones primitivas son:

- **Asignación:** *Variable*←*Expresión*. Pascal: *`var := expr`*.
- **Entrada:** Leer(*lista de variables*). Pascal: `read(`*`vars`*`)`, `readln(`*`vars`*`)`.
- **Salida:** Escribir(*lista de expresiones*). Pascal: `write(`*`exprs`*`)`, `writeln(`*`exprs`*`)`.

Todas las declaraciones e instrucciones del léxico y algoritmo terminan por punto y coma, y cada Entero o Real puede ir acompañado de un rango. Ejemplos:

Lenguaje algorítmico	Pascal

m : Entero$[0, 59]$;
d : Entero ≥ 0;
c : Caracter['a', 'z'];
	`m : 0..59;`
	`c : 'a'..'z';`
	`mes : febrero..abril;`

1.2. Organización de las acciones

- **Composición secuencial:** Se introducen estados intermedios para reducir la complejidad, descomponiendo el problema en subproblemas más simples independientes.
- **Análisis de casos:** Se divide el problema en casos según los datos y se resuelve el caso correspondiente en cada situación. La precondición de cada problema debe implicar la precondición inicial, y la postcondición de cada uno debe cumplir la postcondición inicial. Además, se deben considerar todos los casos como mucho una vez.

Lenguaje algorítmico	Pascal
SEGÚN $c_1, \ldots, c_n$ { Nombres de las variables a comprobar } e_1: a_1 ... e_n: a_n EN_OTRO_CASO: $a_{n+1 opcional}$ FIN_SEGÚN	
SI e ENTONCES a SI_NO $b_{opcional}$ FIN_SI	`if` e `then` a [`else` b]

En Pascal, los campos de expresiones solo permiten una acción, pero se pueden agrupar varias en una sola con las etiquetas `begin` y `end`. Además, se debe situar un punto y coma (`;`) para separar las acciones. No es necesario pues introducirlo antes de un `end`, ni tampoco debe usarse antes de un `else`, pues indicaría el final de la sentencia `if` completa.

1.3. Tipos de datos no primitivos

Son las **tablas,** los **registros** o **producto de tipos** (estructuras) y las **secuencias**. Para definir un registro:

Lenguaje algorítmico	Pascal

nombre_del_tipo = TIPO $<a_1, a_2$: T_1; ...; a_n: $T_m>$;
variable : *nombre_del_tipo*
	`type`
	  nombre_del_tipo `= record`
	  a_1, a_2 : T_1;
	  ...;
	  a_n : T_m;
	  `end;`
	`var`
	  variable : *nombre_del_tipo*;

Capítulo 2

Parametrización de acciones

2.1. Acciones

Permiten conseguir **generalidad** representando un conjunto potencialmente infinito de cálculos con el mismo algoritmo, que es una **abstracción** de estos. Las acciones son la base para **descomposición** de programas en el paradigma imperativo. También mejoran la legibilidad, el mantenimiento y la reutilización de partes del programa. La abstracción proporcionada es por **especificación** (separa el qué hace del cómo lo hace) y por **parametrización** (definición general contra caso específico). En lenguaje algorítmico, las acciones se definen como:

Lenguaje algorítmico	Pascal
nombre : una acción (tp_1 par_1 : td_1; ...; tp_n par_n : td_n)	`procedure` *nombre* `(`[`var`] par_1 `:` td_1`;` ... `;` [`var`] par_n `:` td_n`);`

A esto le sigue la implementación, similar a la del algoritmo en sí. Aquí, par_i es el nombre del parámetro, td_i es su tipo de dato y tp_i puede ser «DATO» (entrada), «RESULTADO» (salida) o «DATO-RESULTADO» (entrada y salida), y en Pascal los dos últimos casos se indican con `var`. En par_i se pueden agrupar varios parámetros separados por comas.

Los nombres de los parámetros forman parte de su **léxico local**. Pueden **enmascarar** elementos del léxico global y no son utilizables fuera de la acción. Llamamos **parámetros formales** a los parámetros de una acción ($par_1, \ldots, par_n$) y **argumentos** o **parámetros reales** a los valores con los que se invoca la acción, mediante la notación «$nombre(expr_1, \ldots, expr_n)$».

2.2. Funciones

Se diferencian de las acciones en que:

- Las acciones modifican el estado del proceso, mientras que las funciones establecen una relación entre los elementos del **dominio** y el **codominio** ($f : A \to B$).
- Las acciones definen procedimientos complejos a partir de otros más simples, mientras que las funciones extienden el repertorio de operadores.

Se definen de forma similar a las acciones:

Lenguaje algorítmico	Pascal
nombre: Función (par_1: td_1; ...; par_n : td_n) → tipo_retorno;	`function` *nombre* (*par_1 : td_1; ...; par_n : td_n*): *tipo_retorno*;

Para devolver un valor, se asigna al propio nombre de la función, que se comporta como una variable de tipo «tipo_retorno» de solo escritura. En exámenes y prácticas, no podemos hacer esto a mitad de la función, aun cuando en Pascal este nombre se comporta como cualquier otra variable local.

2.3. Enumerado

Solo en Pascal. Ejemplo:

```
type colors = (red, green, blue);
var color : colors;
begin
      color := red;
end.
```

Capítulo 3

Composición iterativa y noción de secuencia

Una **secuencia** es un conjunto ordenado de valores. Se expresa como $[a_1, \ldots, a_n]$. La función «$long(S)$» devuelve la longitud de la secuencia S. Si $S = [a_1, \ldots, a_n]$ y $S' = [b_1, \ldots, b_m]$ y e es un elemento, definimos conceptualmente las siguientes operaciones:

- **Añadir un elemento:** $S \bullet e = [a_1, \ldots, a_n, e]$; $e \circ S = [e, a_1, \ldots, a_n]$.
- **Concatenar secuencias:** $S\&S' = [a_1, \ldots, a_n, b_1, \ldots, b_m]$.
- $primero(S) = a_1$; $último(S) = a_n$.
- **Sucesor:** $sucesor(S, i) = S_{i+1}$.
- **Cola** y **cabeza:** $cola(S) = [S_2, \ldots, S_n]$; $cabeza(S) = [S_1, \ldots, S_{n-1}]$.
- **¿Vacía?**: $esvacía(S) = \begin{cases} \text{Verdadero} & S = [] \\ \text{Falso} & S \neq [] \end{cases}$

3.0.1. Primer modelo

	E. inicial	E. final	Efecto
Comenzar(S)	Marcada/Consulta	Consulta	Comienza desde el principio.
Avanzar(S)	Consulta	Consulta	Avanza un elemento.
EA(S)	Consulta	Consulta	Obtiene el elemento actual.
Crear(S)	Cualquiera	Creación	Crea una secuencia vacía.
Registrar(S,e)	Creación	Creación	Añade un elemento.
Marcar(S)	Creación	Marcada	Añade una marca de fin de secuencia.

Por defecto una secuencia no está en ningún estado, por lo que solo podemos usar Crear(S). En Pascal, estas operaciones se encuentran en los archivos `unitmse1`, `unitmsc1` y `unitmsr1` para enteros, caracteres y reales. Debemos añadir el código justo después de la línea con el nombre del programa, con el nombre del archivo correspondiente:

```
uses
        unitmsx1;
```

El tipo de dato es MS_x1 (siendo $_x$ una `e`, `c` o `r`), y las funciones se denominan $Comenzar\_MS_x1$, etc. Además, se añaden $Encender\_Maquina\_MS_x1(S)$, que debe ser llamada una y sólo una vez antes de cualquier otra operación sobre la secuencia (el «constructor») y $Cargar\_Fichero\_MS_x1(S, s)$, donde s es una cadena de caracteres (capítulo 5), y carga en la secuencia los datos leídos del fichero indicado en s.

3.0.2. Segundo modelo

	E. inicial	E. final	Efecto
Iniciar(S)	Cr./Inic./Cons.	Consulta	Comienza desde antes del 1^{er} elemento.
Avanzar(S)	Iniciada/Cons.	Consulta	Avanza un elemento. Error si EsÚltimo.
EA(S)	Consulta	Consulta	Obtiene el elemento actual.
EsVacía(S)	Cualquiera	(E. inicial)	Devuelve si la secuencia es vacía.
EsÚltimo(S,e)	Inic./Cons.	(E. inicial)	Devuelve si el elem. actual es el último.
Crear(S)	Cualquiera	Creación	Crea una secuencia vacía.
Registrar(S)	Creación	Creación	Añade un elemento a la secuencia.

Estas secuencias son similares a las del primer modelo, y en Pascal se usan igual, pero no tienen marca de fin y empiezan antes del primer elemento.

3.1. Composición iterativa

Las iteraciones tienen un **invariante** (INV), que se cumple tras cada ciclo y es un subconjunto de la post-condición, además de tener una **precondición** (PRE) y una **postcondición** (POST).

Lenguaje algorítmico	Pascal
MIENTRAS c **HACER** e **FIN_MIENTRAS**	`while` c `do` e
REPETIR s **HASTA_QUE** c	`repeat` s (no es necesario `begin..end`). `until` c

ITERAR
s_1
DETENER c
s_2
FIN_ITERAR
	`repeat`
	s_1`;`
	`if` c `then break;`
	s_2`;`
	`until false`

En exámenes y prácticas, no podemos utilizar `break` salvo en este caso.

Cualquier composición iterativa se puede expresar en términos de cualquier otra, pero hay que saber elegir la más apropiada en cada caso.

Capítulo 4

Esquemas algorítmicos de recorrido y búsqueda. Iteraciones

Un esquema algorítmico es una «plantilla» de algoritmo aplicable no a un problema sino a una *clase* de problemas. Estudiaremos los siguientes:

- **Recorrido** o **enumeración secuencial:** Aplicación del mismo tratamiento a todos los elementos de una colección. Debemos estudiar si podemos tratar la secuencia vacía como el resto (H_1) y si podemos tratar al primer elemento como el resto (H_2). Dado que $H_1 \implies H_2$, tenemos tres esquemas: (1) $H_1 \wedge H_2$, (2) $\neg H_1 \wedge H_2$ y (3) $\neg H_1 \wedge \neg H_2$.

- **Búsqueda:** Encontrar el primer elemento e en la colección P que cumpla cierta propiedad. Puede que dicha propiedad no dependa solo de e sino también de los elementos anteriores (P_{iz}), en cuyo caso habrá que hacer los tratamientos necesarios. Una vez encontrado el elemento buscado, la iteración se detiene.

Es muy importante identificar la clase de cada problema, pues de lo contrario se podría confundir una búsqueda con un recorrido. Un mismo problema puede combinar búsqueda y recorrido, bien de manera secuencial o uno dentro del otro.

Una **iteración** define una sucesión de valores dados por el conjunto de variables implicadas, que denotamos V. La secuencia se caracteriza por un valor inicial V_0 (**inicialización**), una **condición de continuación** $P(V)$ y un conjunto de funciones que modifican el valor de V en cada iteración, $f(V)$ (**cuerpo**). Para saber que la iteración finaliza después de un número finito de pasos, definimos una **función de terminación** T entera que dependa de las variables y sea estrictamente decreciente respecto al progreso de la iteración con una cota inferior.

Existen dos formas de definir una sucesión:

- **Explícita** o **calculada**: a_i se expresa como una fórmula o algoritmo desde i.

- **Implícita** o **recurrente**: a_i se expresa mediante una relación de inducción, a partir de $a_{i-r}, \ldots, a_{i-1}$, donde r es la **profundidad**, y los valores $a_0, \ldots, a_{r-1}$ son dados.

Dada una sucesión a_n, definimos la sucesión de **sumas parciales** o **serie** como $S_1 = a_1$, $S_n = S_{n-1} + a_n$.

Capítulo 5

Tablas

Están formadas por un número fijo de elementos de un determinado tipo. Definición:

Lenguaje algorítmico	Pascal
nombre_tipo = TIPO Tabla [s_1, e_1; ...; s_n, e_n] de *tipo*;	`nombre_tipo = array [`s_1`..`e_1`, ..., `s_n`..`e_n`] of tipo ;`

Si T es una variable de un tipo tabla, podemos acceder a sus elementos con la notación $T_{a_1,\ldots,a_n}$, donde $s_i \leq a_i \leq e_i$, siendo s_i y e_i constantes de tipo entero, caracter o enumerado. En Pascal, `T[`a_1`,...,`a_n`]`.

5.1. Composición k-RECORRIENDO

Lenguaje algorítmico	Pascal
i **RECORRIENDO** [a, b] **HACER** s **FIN_RECORRIENDO**	`for` i `:=` a `to` b `do` s
i **RECORRIENDO** [a, b] **EN_SENTIDO_INVERSO HACER** s **FIN_RECORRIENDO**	`for` i `:=` b `downto` a `do` s

Ejecuta s una vez por cada elemento entre a y b, en sentido directo o inverso, y en este la variable i, de sólo lectura, toma el valor del elemento en cuestión.

5.2. Tipo `string`

Representa una cadena de caracteres, a los que se accede igual que en una tabla definida de 1 a n:

```
var str : string [n]
```

Se representan con comillas simples rodeando el contenido, se accede a los caracteres con `str[i]` y se definen las siguientes operaciones:

- `st1 + st2`
- `concat(st1, ..., stn : string) : string`
- `copy(st : string; start, index : integer) : string`
- `delete(var st : string; start, index : integer)`
- `insert(src : string; var dst : string; pos : integer)`
- `length(s : string) : integer`
- `pos(substr, str : string) : byte`
- `str(in :` (byte, integer...)`; var out : string)`
- `val(in : string; var out :` (byte, integer...)`; var invalidcharpos : integer)`

5.3. Algoritmos de ordenación

Suponemos que todos los algoritmos incluyen el siguiente léxico:

LÉXICO
TipoClave = ENTERO;
TipoDatos = **TIPO** < clave : TipoClave; . . . >;
TipoIndice = **TIPO**[1..n];
a : **TABLA** [TipoIndice] **DE** TipoDatos;

5.3.1. Inserción directa

LÉXICO
q, j : `TipoIndice`;
b : `TipoDatos`
ALGORITMO
q **RECORRIENDO** [2, n] **HACER**
$b \leftarrow a_q$;
$j \leftarrow q - 1$;
MIENTRAS b.clave < a_j.clave **HACER**
$a_{j+1} \leftarrow a_j$;
$j \leftarrow j - 1$
FIN_MIENTRAS;
$a_{j+1} \leftarrow b$
FIN_RECORRIENDO
FIN.

5.3.2. Inserción binaria

```
LÉXICO
        inf, sup, med : TipoIndice;
        x : TipoDatos
ALGORITMO
        i RECORRIENDO [2,n] HACER
                inf ← 1;
                sup ← i − 1;
                x ← a_i;
                MIENTRAS inf ≤ sup HACER
                        med ← (inf + sup) DIV 2;
                        SI x.clave < a_med.clave
                        ENTONCES sup ← med − 1
                        SI_NO inf ← med + 1
                        FIN_SI
                FIN_MIENTRAS;
                j RECORRIENDO [inf, i − 1] EN_SENTIDO_INVERSO HACER
                        a_{j+1} ← a_j
                FIN_RECORRIENDO;
                a_inf ← x
        FIN_RECORRIENDO
FIN.
```

5.3.3. Selección directa

```
LÉXICO
        pos, j, q : TipoIndice;
        min : TipoDatos
ALGORITMO
        q RECORRIENDO [1,n − 1] HACER
                pos ← q;
                min ← a_q;
                j RECORRIENDO [q + 1, n] HACER
                        SI min.clave > a_j.clave ENTONCES
                                pos ← j;
                                min ← a_j
                        FIN_SI
                FIN_RECORRIENDO;
                a_pos ← a_q;
                a_q ← min
        FIN_RECORRIENDO
FIN.
```

Otros algoritmos de ordenación, más avanzados, son el **algoritmo de Shell**, $O(n^{1,25})$, y **Quicksort**, $O(n \log n)$, que Charles Hoare demostró que no existe otro más rápido.