Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 1

Introducción

Objetivos

- Capacidad para encontrar soluciones informáticas a problemas mediante la modelización disciplinada de soluciones y descomposición en módulos.
- Capacidad para traducir eficientemente algoritmos al lenguaje de programación C.
- Habilidad en el uso del lenguaje C.
- Habilidad en el uso de buenos hábitos de programación.
- Capacidad para documentar técnicamente los programas desarrollados (análisis, diseño, implementación, prueba, manuales para el usuario, etc).
- Habilidad en el uso de herramientas básicas de desarrollo de software.
- Capacidad para trabajar en grupo.

Características de la asignatura

- **Carreras**: Analista en Computación – Profesorado en Ciencias de la Computación – Licenciatura en Ciencias de la Computación.

- **Régimen de regularidad:**
  - 80% de asistencia a las clases teóricas en aula común en cada cuatrimestre.
  - 80% de asistencia a las clases prácticas en aula común en cada cuatrimestre.
  - 80% de asistencia a las clases de laboratorio en cada cuatrimestre.
  - Presentación de 1 ejercicio resuelto en Lenguaje C en los trabajo prácticos en que así se indique.
  - Un trabajo evaluable individual por parcial, realizado en el Laboratorio, que aportará un 10% de la valoración total del parcial respectivo.
  - Aprobar cuatro (4) exámenes parciales, dos en primer cuatrimestre y dos en el segundo. Habrá cuatro recuperatorios, uno por cada parcial.
  - Aprobar un proyecto integrador en lenguaje C a fin de año.

Características de la asignatura

- **Asignación de horas semanales:**

  Teóricos: 4hs. Consulta: 2hs.

  Prácticos: Aula común: 4hs. Consulta: 2hs.
  Laboratorio: 2hs. Consulta: 2hs.

- **Exámenes parciales**: escritos e individuales. Dos exámenes parciales durante el primer cuatrimestre y dos durante el segundo.

  Fechas de cada examen: **Primer Parcial jueves 23/04**, el resto a definir luego del 23/04. **Semana de parciales: 21/04 al 30/04**

- **Examen final**: individual. Oral.

Horarios

Teóricos

ARIEL FERREIRA SZPINIAK (aferreira@exa.unrc.edu.ar):
Lunes de 16 a 18hs, Jueves de 16 a 18hs.

Prácticos

Laboratorio: Aula 101 del Pabellón 2

Mañana

Comisión 1 - Aula común: Lunes de 10 a 12hs, Miércoles de 10 a 12hs.
Laboratorio: Miércoles de 8 a 10hs. A cargo de Luis Chávez

Comisión 2 - Aula común: Lunes de 10 a 12hs, Miércoles de 08 a 10hs.
Laboratorio: Miércoles de 10 a 12hs. A cargo de César Cornejo

Tarde

Comisión 3 - Aula común: Lunes de 18 a 20hs, Miércoles de 16 a 18hs.
- **Laboratorio: Miércoles de 18 a 20hs. A cargo de Luis Chávez**

Horarios

Prácticos

Laboratorio: Aula 101 del Pabellón 2

Mañana

Comisión 5 - Aula común: Miércoles de 10 a 12hs, Viernes de 10 a 12hs.
Laboratorio: Lunes de 8 a 10hs. A cargo de César Cornejo

Tarde

Comisión 4 - Aula común: Lunes de 18 a 20hs, Miércoles de 18 a 20hs.
- **Laboratorio: Martes de 16 a 18hs. A cargo de César Cornejo**

Aulas https://sisinfo.unrc.edu.ar/bedepub/

Día	Horario	Asignatura	Docente a cargo	Aula /Pabellón
Lunes	10 a 12	C1 – Práctico	Guillermo Rojo	Aula 105 Pab 2
Lunes	10 a 12	C2 – Práctico	Luis Chávez	Aula 106 Pab 2
Lunes	16 a 18	Teórico	Ariel Ferreira Szpiniak	Aula 35 Pab 4
Lunes	18 a 20	C3 – Práctico	Jorge Guazzone	Aula 8 Pab 4
Lunes	18 a 20	C4 – Práctico	Luciano Putruele	A definir
Miércoles	08 a 10	C2 – Práctico	Luis Chávez	Aula 106 Pab 2
Miércoles	10 a 12	C1 – Práctico	Guillermo Rojo	Aula 106 Pab 2
Miércoles	10 a 12	C5 – Práctico	Facundo Molina	Aula 106 Pab 2
Miércoles	16 a 18	C3 – Práctico	Jorge Guazzone	Aula 8 Pab 4
Miércoles	18 a 20	C4 - Práctico	Luciano Putruele	A definir
Jueves	16 a 18	Teórico	Ariel Ferreira Szpiniak	Aula 35 Pab 4
Viernes	10 a 12	C5 - Práctico	Facundo Molina	Aula 107 Pab 2

Bibliografía de la materia

- Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”, Barcelona, Ed. Masson, 1991.
- Biondi, J. y Clavel, G. “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”, 2° ed., Barcelona: Masson, 1985.
- Clavel, G. y Biondi, J. “Introducción a la Programación. Tomo 2: Estructuras de Datos”, 2° ed., Barcelona: Masson, 1985.
- Quetglás, G. Toledo, F., Cerverón, L. "Fundamentos de Informática y Programación". Valencia, 1995. http://robotica.uv.es/Libro/Indice.htmlhttp://robotica.uv.es/Libro/Indice.html
- De Guisti, A. “Algoritmos, datos y programas. Con aplicaciones en Pascal, Delphi y Visual Da Vinci. Prentice Hall.
- Wirth, N. “Algoritmos + Estructuras de Datos = Programas”. Ediciones del Castillo, 1980.

Bibliografía Secundaria

- Lucas, M., J.-P. Peyrin y P. Scholl, “Algorítmica y Representación de Datos. Tomo 1: Secuencia, Autómata de estados finitos”, Barcelona, Ed. Masson, 1985.
- Aho, A., J. Hopcroft and J. Ullman, “Data Structures and Algorithms”, Reading MA, Addison-Wesley, 1987 (traducción al castellano “Estructuras de Datos y Algoritmos”, Addison-Wesley, 1988).

Aula virtual

Utilizaremos un aula virtual para publicar:

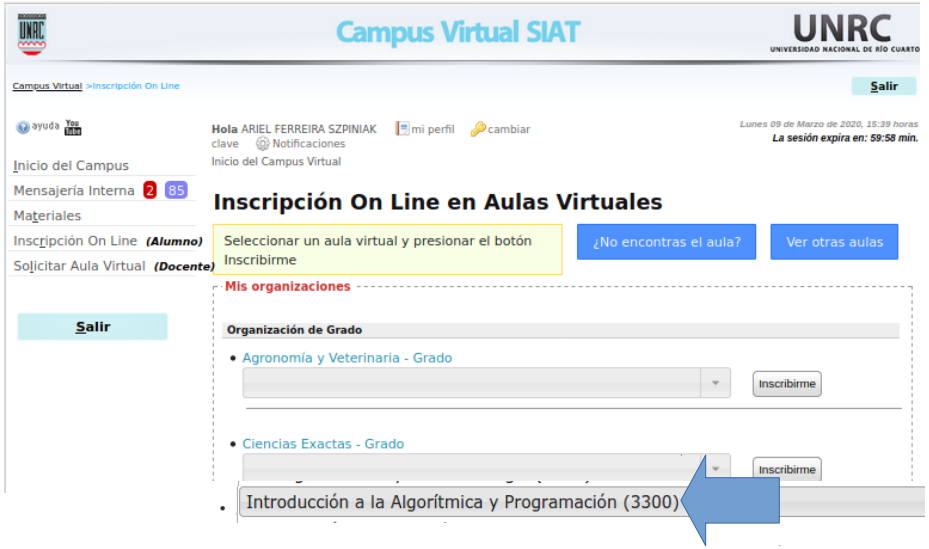
- **Novedades generales, Calendario de actividades**
- **Materiales digitales utilizados para desarrollar las clases teóricas, prácticas y proyectos.**
- **Entrega de trabajos prácticos**
- **Notas de parciales**

Sitio Web de la asignatura (en el Campus Virtual de la UNRC):

www.evelia.unrc.edu.ar

- **Pasos a seguir para utilizar el aula virtual (plazo hasta el 16/03):**
  - **Registrarse**
  - **Ingresar al Campus e inscribirse en la materia: desde el menú principal del Campus, opción Inscripción On Line (Alumno)**
  - **Buscar el aula: Introducción a la Algorítmica y Programación y elegir la Comisión (1, 2, 3, 4 o 5)**
  - **Inscribirse con la siguiente clave de inscripción: 2020**

Aula virtual



Aula virtual

Clave de inscripción: 2020

Problemas
Solución mediante computadoras

No todos los problemas pueden ser resueltos por una computadora.

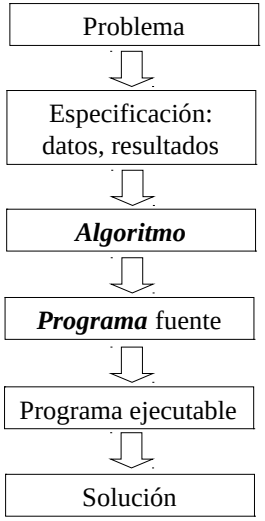
Problema

Un largo camino...

Solución

Problemas
Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Traducción del programa a un lenguaje entendido por la computadora.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Pasos para solucionar un problema
Análisis

- El objetivo del análisis es entender el problema a resolver.
- El problema debe estar bien definido para poder obtener una solución satisfactoria.
- Los datos de entrada y los resultados deben ser precisamente descriptos.

Preguntas orientadoras

Datos de entrada: ¿Cuáles y cuántos son los valores de entrada? ¿Qué nombre significativo puedo darle a esos datos?
Dibujo o esquema que permita entender mejor el problema

Resultados (salida): ¿Cuáles y cuántos son los valores del resultado? ¿Qué nombre significativo puedo darle a esos resultados?

Relaciones o subproblemas: en caso de existir, describir las relaciones existentes entre los datos, los resultados u otra información adicional que sea necesaria para la resolución del problema. O suproblemas en caso de ser un problema más complejo.

Pasos para solucionar un problema
Análisis (cont.)

Podemos dividir el análisis en dos etapas:

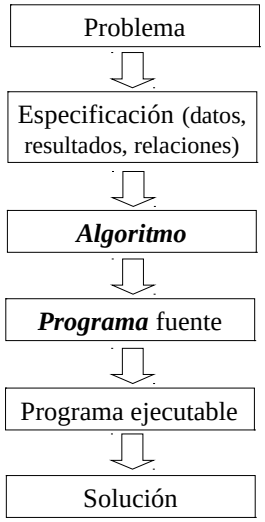
Etapa 1: se analiza el problema libremente y se trata de lograr una síntesis del problema a resolver.

Etapa 2: luego de tener en claro el problema a resolver se tratan de identificar los datos de entrada, los resultados y las relaciones que puedan existir entre datos, resultados u otros componentes del problema. Al finalizar la Etapa 2 se debe obtener lo siguiente:

- Dato/s:
- Resultado/s:
- Relaciones o subproblemas:

Problemas
Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Traducción del programa a un lenguaje entendido por la computadora.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Pasos para solucionar un problema
Diseño

- El objetivo del diseño es desarrollar un algoritmo que solucione el problema de forma genérica, sin considerar los detalles de implementación.

Podemos dividir el diseño en dos etapas:

- **Etapa 1:** definir el entorno de trabajo o léxico (tipos, variables, etc).

- **Etapa 2:** construir un algoritmo.

Pasos para solucionar un problema
Diseño

En la Etapa 2:

- Si el problema es simple: se construye un algoritmo que lo resuelva.
- Si el problema es más complejo: puede ser visto como la composición de varios (sub)problemas de menor complejidad.

Por ejemplo, subproblemas:

(1) obtener datos del hexágono
(2) calcular área del hexágono
(3) informar el área del hexágono

A cada subproblema se le vuelve a aplicar la misma técnica, es decir, un análisis y un diseño.

Noción de Algoritmo
Acción, Entorno y Procesador

Consideremos los siguientes enunciados:

E1: 1/2 docena de huevos revueltos

a) Romper seis huevos en un plato;
b) Batir la clara y la yema con un tenedor;
c) Calentar el aceite en una sartén al fuego;
d) Cuando el aceite esté caliente, verter el contenido del plato;
e) Sacar la sartén del fuego cuando el revuelto esté cocido.

E2: **Cálculo de la media de dos números con una calculadora**

a) pulsar *C ;*
b) teclear el primer número;
c) pulsar + *;*
d) teclear el segundo número;
e) pulsar *% ;*
f) teclear 2;
g) pulsar = . {Aparece el resultado}

Noción de Algoritmo
Acciones primitivas y descomposición

En los enunciados E1 y E2 hemos supuesto que el procesador sabía ejecutar las acciones enumeradas. En ese caso las acciones se denominan ***primitivas***.

Definición: para un procesador dado, una acción es **primitiva** si el enunciado de dicha acción es suficiente para poder ejecutarla sin información suplementaria.

Si una acción no es primitiva debe ser descompuesta en dos o más acciones primitivas.

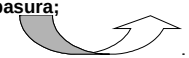
Definición: **descomponer** una acción -no primitiva- es encontrar una serie de **acciones primitivas** que realicen lo requerido por dicha acción.

Noción de Algoritmo
Ejemplo de descomposición

Supongamos que en el enunciado E1, el procesador es un niño y no comprende la acción “a) romper seis huevos en un plato”. Es necesario descomponer la acción.

a) romper seis huevos en un plato;

a1) poner seis huevos en la superficie de trabajo;
a2) tomar un huevo de la superficie de trabajo;
a3) romperlo y verter su contenido en el plato;
a4) tirar las cáscaras a la basura;
a5) tomar otro huevo de la superficie de trabajo;
a6) romperlo y verter su contenido en el plato;
a7) tirar las cáscaras a la basura;
a8) tomar otro huevo de la superficie de trabajo;
a9) romperlo y verter su contenido en el plato;
a10) tirar las cáscaras a la basura;
a11) tomar otro huevo de la superficie de trabajo;
a12) romperlo y verter su contenido en el plato;
a13) tirar las cáscaras a la basura;
a14) tomar otro huevo de la superficie de trabajo;
a15) romperlo y verter su contenido en el plato;
a16) tirar las cáscaras a la basura;
a17) tomar el último huevo de la superficie de trabajo;
a18) romperlo y verter su contenido en el plato;
a19) tirar las cáscaras a la basura;



Noción de Algoritmo
Procesador

Definición: un procesador es la entidad responsable de ejecutar las **acciones primitivas.**

El procesador solo entiende las acciones primitivas, nada más.

Una acción no primitiva puede transformarse en primitiva según quien sea el procesador.

El procesador puede ser una máquina, una persona, etc.

Noción de Algoritmo
Entorno de trabajo, variable y tipo

Entorno de trabajo

Definición: el entorno de trabajo es el espacio donde conviven los distintos utensilios que pueden ser usados por el procesador para ejecutar las **acciones primitivas.** El procesador modifica el entorno de trabajo mediante la ejecución de las acciones.

Dato

Definición: un dato es la **representación** de un **objeto** del **mundo real** mediante el cual podemos modelar aspectos del problema que se quiere resolver. Son los **valores** de información de los que se necesita **disponer** y en ocasiones **transformar**.

Noción de Algoritmo
Entorno de trabajo, variable y tipo

Variable

Definición: son **datos** que tienen la **posibilidad** de **cambiar** su **valor**. Ejemplo: saldo

Constante

Definición: son **datos** que **no pueden cambiar** su **valor**. Ejemplo: Iva = 21

Tipo

Definición: es un **conjunto** de **valores posibles** que se encuentran ligados a un conjunto de operaciones para crearlos y manipularlos. Todo **dato**, tanto **constante** como **variable**, debe **pertenecer** a un **tipo**.
Ejemplos: saldo $\in$ Z (saldo pertenece a Entero). Iva=21 $\in$ Z (Iva es igual a 21 y pertenece a Entero)

Algunos Tipos Simples

- **Entero (Z):** es el conjunto de valores numéricos más simple de todos: ..., -2, -1, 0, 1, 2, …

  Operaciones: +, -, *, /, div, mod, =, <>, <, <=, >, >=

- **Lógico:** conjunto de valores lógicos: **Verdadero** y **Falso.**

  Operaciones: y (conjunción), o (disyunción), no (negación)

- **Caracter:** conjunto de caracteres: letras minúsculas, mayúsculas, cifras, y signos especiales. ‘M’ ‘m’ ‘5’ ‘%’

  Operaciones: =, <>, >, <, >=, <=, ord, chr

- **Cadena:** conjunto de cadenas de caracteres: “promNotas” “5mentarios”.

  Operaciones: =, <>, >, <, >=, <=, +

Noción de Algoritmo Significado de la asignación

- El objetivo de la *asignación* es cambiar el valor almacenado en una **variable**.

  x ← e Guarda en **x** el valor de **e**

- Sintaxis: **<variable> ← <expresion>**

- Ejemplos:

  Sean **i,j ∈ Z**

  i ← 9
  j ← 3 + 4

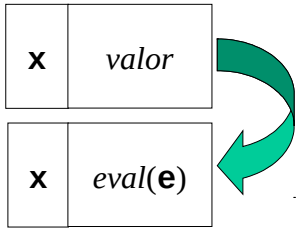
Deben coincidir los tipos de la variable y la expresión

Noción de Algoritmo Significado de la asignación

Una asignación **x ← e** es ejecutada siguiendo estos pasos:

1. Se evalúa la expresión **e**
2. Se reemplaza el valor almacenado en la variable **x**, por el valor de **e**.

x ← e { $x=X_0 \wedge X_0=eval(e)$ }



I ← 9 { i=9 }

j ← **3+2 { j=5 ^ 5=eval(3+2) }**

Noción de Algoritmo Significado de la asignación

- Al producirse la asignación el valor anterior se pierde.
- La ocurrencia de una variable en el lado derecho de una asignación denota su *valor* actual.
- Una misma variable puede aparecer en la parte izquierda y derecha de una asignación.

Por ejemplo: **x ← 1**

¿Qué valor tiene x aquí?

x ← x + 1

¿Qué valor tiene x aquí?

x ← x + 1 **NO** debe interpretarse como una ecuación matemática!
Sólo significa que estamos usando el valor *actual* de la variable **x** para calcular su *nuevo* valor.

Algoritmo
Definición

Definición 1: Un algoritmo es una **sucesión finita de instrucciones** o pasos **no ambiguos** que se pueden ejecutar en un tiempo finito para resolver un problema.

Definición 2: Dado un **procesador, un entorno de trabajo** y un tratamiento a ejecutar por dicho procesador sobre ese entorno, un **algoritmo** es el enunciado de una **secuencia finita de acciones** que resuelve un problema determinado.

Algoritmo
Definición (cont.)

Algoritmo: del árabe al-Jwarizmi, matemático del siglo IX

Características:

- Debe ser preciso e indicar el orden de realización de cada paso.
- Se debe obtener el mismo resultado cada vez que se aplica a los mismo datos.
- Se debe terminar en algún momento.

En la vida cotidiana empleamos algoritmos en multitud de ocasiones. También existen ejemplos de índole matemática (algoritmo de la división, Euclides, Gauss, Valor Medio, etc.)

Algoritmo
Estructura y Notación

```
Algoritmo <nombre>
Léxico
```

Entorno de trabajo

```
  <declaración de variables, tipos,
  acciones, funciones, etc>
```

```
Inicio
  <secuencia de acciones>
Fin
```


Notación Algorítmica
Acciones conocidas por el Procesador

nada	No produce cambio alguno en el entorno
x ← e	Guarda en la variable **x** el valor de **e**
Entrada:x	Guarda en la variable **x** un valor
Entrada:x y	Guarda en la variable **x** un valor **y** en la variable y otro valor (se puede generalizar a más variables)
Salida:e	Informa un resultado a través del valor de **e**. Donde **e** puede ser una o más variables
//	Comentario (el procesador lo ignora). Pueden ir en cualquier lugar del algoritmo

Noción de Algoritmo

Ejemplo

```
Algoritmo SumarUno
Léxico
  x, a ∈ Z  //números enteros
Inicio
  Entrada:x
  a ← x+1
// a contiene la suma de x más 1
  Salida:a
Fin
```

comentarios

Noción de Algoritmo

Ejemplo (2)

```
Algoritmo SumarDosNumeros
Léxico
  x, y, a ∈ Z  //números enteros
Inicio
  Entrada:x y
  a ← x+y
// a contiene la suma de x e y
  Salida:a
Fin
```

comentarios

Pasos para solucionar un problema

Implementación

Es la traducción del algoritmo a un lenguaje entendido por la computadora.
¿Qué lenguaje entiende la computadora?

Código binario	Bajo nivel	Alto nivel	Bloques

01100001
01110110
01100001
01101110
01111010
01100001
	ADD ECX, R9D
	ADD ECX, R8D
	ADD ECX, EDX
	MOVD XMM0, ECX
	CVTDQ2PD XMM0, XMM0
	MOVSD XMM1, realVal
		if username == user1:
		print "Access granted"
		elif username == user2:
		print "Welcome"
		else:
		print "Access denied"
			

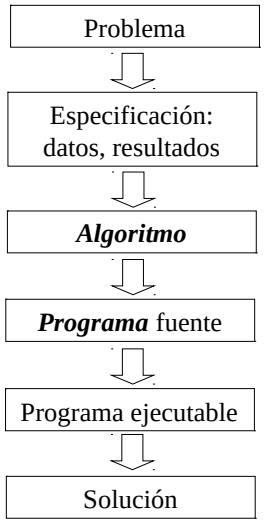
Implementar consiste en traducir el **algoritmo** a un **lenguaje de programación de alto nivel** (C, BASIC, COBOL, Pascal, Java, PHP, C++, FORTRAN, Perl, Pyton, Ruby). Pero hay muchos lenguajes, por lo tanto pueden haber varios programas que solucionen el mismo problema resuelto por un algoritmo.

La gran ventaja de los lenguajes de alto nivel es que consiguen distanciarse del lenguaje máquina y se aproximan al lenguaje algorítmico.

Problemas

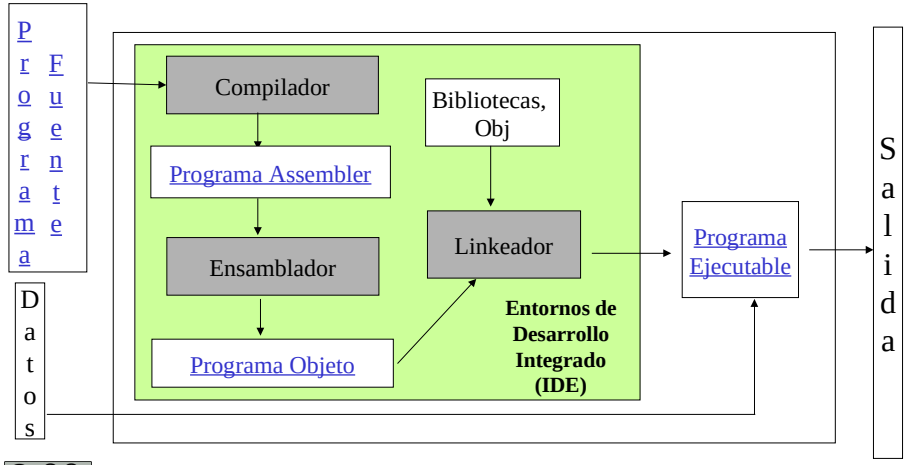
Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Traducción del programa a un lenguaje entendido por la computadora.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Pasos para solucionar un problema
Compilación

"Compilar" (traducir) el programa fuente al lenguaje entendido por la computadora. Detección de errores (sintácticos, …).



Pasos para solucionar un problema
Compilación

El compilador gcc

```
rcanales@freeoszoo:~/c$ more fuente.c
#include "stdio.h"
#include "stdlib.h"

int main()
{
   printf("Hola linux");


}
rcanales@freeoszoo:~/c$ gcc fuente.c
rcanales@freeoszoo:~/c$ ll
total 16
-rwxr-xr-x  1 rcanales users 11517 2005-11-04 17:05 a.out*
-rw-r--r--  1 rcanales users    82 2005-11-04 17:04 fuente.c
```


Pasos para solucionar un problema
Compilación

El compilador gcc

- **gcc hola.c**
  Compila el programa hola.c, y genera un archivo ejecutable llamado a.out
- **gcc -o hola hola.c**
  Compila el programa hola.c, y genera un archivo ejecutable llamado hola

Pasos para solucionar un problema
Compilación

El compilador gcc

- **gcc -c hola.c**

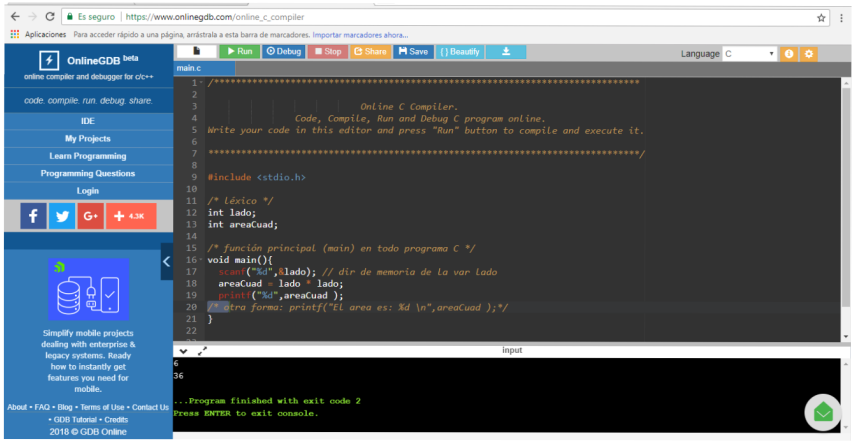
No genera el archivo ejecutable, sino el código objeto, en el archivo hola.o. Si no se indica un nombre para el archivo objeto, usa el nombre del archivo en C y le cambia la extensión por .o

- **gcc -c -o objeto.o hola.c**

genera el código objeto indicando el nombre de archivo.

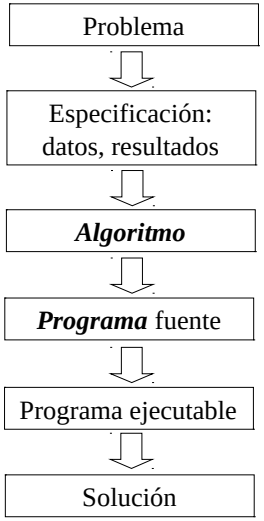
Pasos para solucionar un problema
Compilación

Compilador (y depurador) on line para C y C++
www.onlinegdb.com/online_c_compiler



Problemas
Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Traducción del programa a un lenguaje entendido por la computadora.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Pasos para solucionar un problema
Ejecución y Prueba

• Ejecutar (o “correr”) el resultado de la compilación, es decir, el programa ejecutable.

• Cuando se **ejecuta un programa**, la compjutadora (el sistema operativo) crea un **proceso** donde pone a funcionar ese programa.
- Procesamiento de datos de entrada.
- Detección de errores semánticos.

• Prueba o Testeo
- Probar el programa con una serie de valores de entrada y verificar que produce el resultado esperado en todos los casos.

Pasos para solucionar un problema
Ejecución y Prueba

./a.out

Ejecuta el programa complilado de la siguiente manera:
gcc hola.c
(compila el programa hola.c, y genera un archivo ejecutable llamado a.out)

```
rcanales@freeoszoo:~/c$ ./a.out
Hola linuxrcanales@freeoszoo:~/c$ _
```

./hola

Ejecuta el programa complilado de la siguiente manera:
gcc -o hola hola.c
(compila el programa hola.c, y genera un archivo ejecutable llamado hola)

Ejemplo
Programa Fuente en C

```c
int k, l, m;

void main(){
  l = 3;
  m = 5;
  k = l + m;
}
```


Ejemplo
Programa Assembler

```c
int k, l, m;

void main(){
  l = 3;
  m = 5;
  k = l + m;
}
```

```
.section .bss
# [1] int k,l,m
  .lcomm  _K,2
  .lcomm  _L,2
  .lcomm  _M,2
  .comm   HEAP,262144#
[3] void main(){
.globl    _main
_main:
.globl    CMAIN
CMAIN:
.globl    program_init
program_init:
          pushl     %ebp
          movl      %esp,%ebp
          movb      $1,U_SYSWIN32_ISCONSOLE
          call      FPC_INITIALIZEUNITS
# [4] l=3;
          movw      $3,_L
# [5] m=5;
          movw      $5,_M
# [6] k=l+m;
          movswl    _L,%eax
          movswl    _M,%edx
          addl      %eax,%edx
          movw      %dx,_K
# [7] }
```


Ejemplo
Programa Objeto

```
L□□□□□□□j□□□□□□□□□□□.text□□□□□□□□□□□@□□□@□□□□□□□□□□□□□□□□
□□`.data□□□□□□□□□□D□□□Ì□□□`□□□□□□□□□□□@□□À.bss□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□€
□□ÀU‰å□□□□□□è□□□□fÇ□□□□□□□fÇ□□□□□□□□¿□□□□□□¿□□□□□□Âf‰□□□□□è□□□□ÉÃ□FPC      1.0.10
[2003/06/27]     for     i386     -     WIN32□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□     □□□□□□□□□□   □□□□□$□□□   □□□□□+□□□   □□□□□4□□□
      □□□□□9□□□□□□□□□4□□□□□□□□□.file□□□□□□□þÿ□□g□sumaente.pas□□□□□□_L□□□□□□□□□□□
□□□□□_M□□□□□□□□□□□□□□□□_K□□□□□□□□□□□□□□□□.text□□□□□□□□□□□□□@□□□□□□□□□□□□□□□□□.da
ta□□□□□□□□□□□□□D□□□□□□□□□□□□□□□□□.bss□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□_main□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(□□□□□□□□□□□□5□□□,□□□□□□□□□H
EAPSIZE<□□□□□□□□□__stklen@□□□□□□□□□HEAP□□□□□□□□□□□□□□□□□□□?□□□□□□□□□□□□□□□□□T□□□□
□□□□□□□□□□□□h□□□□□□□□□□□□□□□□□t□□□□□□□□□□□□□ƒ□□□PASCALMAIN□program_init□FPC_RES
OURCESTRINGTABLES□INITFINAL□U_SYSWIN32_ISCONSOLE□FPC_INITIALIZEUNITS□FPC_DO_EXIT
□INIT$$SYSWIN32□
```

Código de máquina y lenguaje assembler del Intel 8088

direcciones de memoria donde se encuentra el código

direcciones de memoria	código de máquina	código assembler	
0CFD:0100	BA0B01	MOV	DX,010B
0CFD:0103	B409	MOV	AH,09
0CFD:0105	CD21	INT	21
0CFD:0107	B400	MOV	AH,00
0CFD:0109	CD21	INT	21

código de máquina

código assembler

La mayoría de los clones del IBM PC y XT usaron el Intel 8088

Ejemplo
Programa Ejecutable

```
□□□ÿÿ□□¸□□□□□□□□@□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□€□□□□□□°
□LÍ!This program cannot be run in DOS mode. $□□□□□□□□PE□□L□□□

□□□□□□□@□□□□□@□□□□□□□□□□□□□□□□□□□□¦□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□
Pè□□□□□ÿ%øpD□U‰å□□□
fÇ□□P@□□□fÇ□□P@□□□□¿□□P@□□¿□□P@□□Âf‰□□P@□è£□□□ÉÃ□U‰å□□□P@□□è
:Eø‹8%}ô¸□□□□%Eü;Eô□%‹Eø‹Uüf<Ð□t□ÿ□Ð‹Eø‹}ü%x□‹Eü;Eô}□ÿEüëÛÉ
P@□è□□□□h□S@□è
```

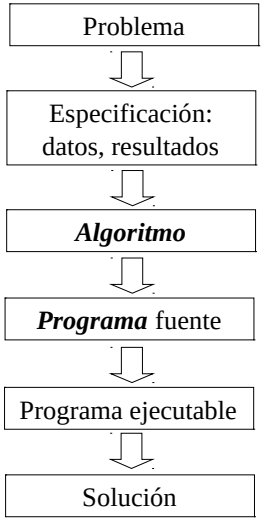
Archivo visto a través de un editor de texto.

FF	D8	FF	E1	1D	FE	45	78	69	66	00	00	49	49	2A	00
08	00	00	00	09	00	0F	01	02	00	06	00	00	00	7A	00
00	00	10	01	02	00	14	00	00	00	80	00	00	00	12	01
03	00	01	00	00	00	01	00	00	00	1A	01	05	00	01	00
00	00	A0	00	00	00	1B	01	05	00	01	00	00	00	A8	00
00	00	28	01	03	00	01	00	00	00	02	00	00	00	32	01
02	00	14	00	00	00	B0	00	00	00	13	02	03	00	01	00
00	00	01	00	00	00	69	87	04	00	01	00	00	00	C4	00
00	00	3A	06	00	00	43	61	6E	6F	6E	00	43	61	6E	6F
6E	20	50	6F	77	65	72	53	68	6F	74	20	41	36	30	00
00	00	00	00	00	00	00	00	00	00	00	00	B4	00	00	00
01	00	00	00	B4	00	00	00	01	00	00	00	32	30	30	34
3A	30	36	3A	32	35	20	31	32	3A	33	30	3A	32	35	00
1F	00	9A	82	05	00	01	00	00	00	86	03	00	00	9D	82
05	00	01	00	00	00	8E	03	00	00	00	90	07	00	04	00

Archivo visto a través de un editor hexadecimal.

Problemas
Pasos para solucionar un problema

- **Análisis**
  El problema debe ser claramente especificado y entendido.
- **Diseño**
  Construcción de una solución general del problema.
- **Implementación**
  Traducción del algoritmo a un lenguaje de programación de alto nivel.
- **Compilación**
  Traducción del programa a un lenguaje entendido por la computadora.
- **Ejecución y Prueba**
  Corrida y prueba de funcionamiento del programa en la computadora.



Pasos para solucionar un problema
Análisis

Ejemplo de Problema: Calcular cuánto espacio ocupa una baldosa cuadrada

Etapa 1 (síntesis): calcular el área de una baldosa cuadrada

Etapa 2 (identificar los datos de entrada, los resultados y las relaciones o subproblemas):

- **Dato**: lado de la baldosa. Es un número. Nombre: **lado**
- **Resultado**: área ocupada por la balsosa. Es un número. Nombre: **areaCuad**
- **Relaciones o subproblemas**: el área de un cuadrado es lado por lado (areaCuad=lado*lado)

Pasos para solucionar un problema
Diseño

```
Algoritmo AreaBaldosa
Léxico
  lado ∈ Z+   // variable dato
  areaCuad ∈ Z+   // variable resultado
Inicio
  Entrada:lado
  areaCuad ← lado * lado
  Salida:areaCuad
Fin
```


Pasos para solucionar un problema
Implementación

Traducción del algoritmo a C (lenguaje que utilizaremos en la asignatura)

```
#include <stdio.h>
/* léxico */
int lado;
int areaCuad;

/* función principal (main) en todo programa C */
void main(){
  scanf("%d",&lado); // dir de memoria de la var lado
  areaCuad = lado * lado;
  printf("%d",areaCuad );
/* otra forma: printf("El area es: %d \n",areaCuad );*/
}
```

Pasos para solucionar un problema

Implementación

Traducción del algoritmo a Pascal (otro lenguaje)

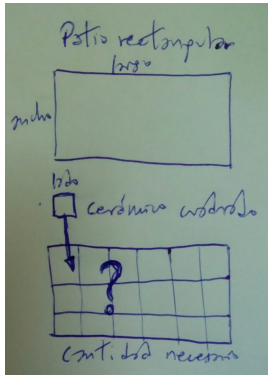
```
PROGRAM AreaBaldosa;
VAR
  lado: Integer;
  areaCuad: Integer;
BEGIN
  Read(lado);
  areaCuad := lado * lado;
  Writeln(areaCuad)
END.
```


Ejemplo de problema más complejo

Doña Rosa tiene un patio de forma rectangular y quiere cubrirlo con cerámicos cuadrados. ¿Puedes ayudarla a calcular cuántos cerámicos necesitará?

Análisis

Etapa 1 (síntesis):



Calcular la cantidad de cerámicos cuadrados necesarios para un patio rectangular

Ejemplo de problema más complejo

Doña Rosa tiene un patio de forma rectangular y quiere cubrirlo con cerámicos cuadrados. ¿Puedes ayudarla a calcular cuántos cerámicos necesitará?

Análisis (cont)

Etapa 2 (identificar los datos de entrada, los resultados y las relaciones o subproblemas):

- **Dato**: Largo y ancho del patio. Son números. Nombres: **largoPatio** y **anchoPatio**. Lado del cerámico. Es un número. Nombre: **ladoCeramico**.
- **Resultado**: cantidad de cerámicos. Es un número. Nombre: **cantCeramicos.**
- **Relaciones o subproblemas**:
  - Calcular el área del patio (areaPatio)
  - Calcular el área de un cerámico (areaCeramico)
  - Calcular la cantidad de cerámicos: cantCeramicos es igual a areaPatio dividido areaCeramico

Ejemplo de problema más complejo

Doña Rosa tiene un patio de forma rectangular y quiere cubrirlo con cerámicos cuadrados. ¿Puedes ayudarla a calcular cuántos cerámicos necesitará?

Diseño

Etapa 1 (definir el entorno de trabajo o léxico: tipos y variables)

ladoCeramico $\in R^+$ // dato de entrada
largoPatio $\in R^+$ // dato de entrada
anchoPatio $\in R^+$ // dato de entrada
areaPatio $\in R^+$ // intermedio
areaCeramico $\in R^+$ // intermedio
cantCeramicos $\in R^+$ // resultado

Ejemplo de problema más complejo

Doña Rosa tiene un patio de forma rectangular y quiere cubrirlo con cerámicos cuadrados. ¿Puedes ayudarla a calcular cuántos cerámicos necesitará?

Diseño (cont)

Etapa 2 (división en subproblemas). A cada subproblema se le vuelve a aplicar la misma técnica, es decir, un análisis y un diseño.

- Calcular el área total del patio rectangular

  Un breve análisis del problema nos lleva a que:

  areaPatio = largoPatio * anchoPatio

- Calcular el área ocupada por cada cerámico

  Un breve análisis del problema nos lleva a que:

  areaCeramico = ladoCeramico * ladoCeramico

- Calcular la cantidad de cerámicos necesarios

  cantCeramicos = areaPatio / areaCeramico

Ejemplo de problema más complejo

```
Algoritmo PatioDeBalsodas
Léxico
  ladoCeramico ∈ R+ // dato de entrada
  largoPatio ∈ R+ // dato de entrada
  anchoPatio ∈ R+ // dato de entrada
  areaPatio ∈ R+ // intermedio
  areaCeramico ∈ R+ // intermedio
  cantCeramicos ∈ R+ // resultado
Inicio
  // Obtener los datos de entrada
  Entrada:ladoCeramico largoPatio anchoPatio
  areaCeramico←ladoCeramico*ladoCeramico // área ocupada por cada cerámico
  areaPatio←largoPatio*anchoPatio // área total del patio rectangular
  cantCeramicos←areaPatio/areaCeramico // cantidad de cerámicos necesarios
  // Informar el resultado
  Salida:cantCeramicos
Fin
```

Nota: Observar la similitud entre este algoritmo y la descripción refinada de la solución del problema.

Ejemplo de problema más complejo

Traducción del algoritmo a C

```c
#include <stdio.h>
/* léxico */
float ladoCeramico, largoPatio, anchoPatio, areaPatio,
areaCeramico, cantCeramicos;

/* función principal (main) en todo programa C */
void main(){
  scanf("%f",&ladoCeramico);
  scanf("%f",&largoPatio);
  scanf("%f",&anchoPatio);
  areaCeramico = ladoCeramico * ladoCeramico;
  areaPatio = largoPatio * anchoPatio;
  cantCeramicos = areaPatio / areaCeramico;
  printf("%f",cantCeramicos);
}
```


Ejemplo de problema más complejo

Compilación (gcc -o patio patio.c)

```c
#include <stdio.h>
/* léxico */
float ladoCeramico, largoPatio, anchoPatio, areaPatio, areaCeramico;

/* función principal (main) en todo programa C */
void main(){
  scanf("%f",&ladoCeramico);
  scanf("%f",&largoPatio);
  scanf("%f",&anchoPatio);
  areaCeramico = ladoCeramico * ladoCeramico;
  areaPatio = largoPatio * anchoPatio;
  cantCeramicos = areaPatio / areaCeramico;
  printf("%f",cantCeramicos);
}
```

stderr

Compilation failed due to following error(s).

```
main.c: In function 'main':
main.c:13:3: error: 'cantCeramicos' undeclared (first use in this function)
   cantCeramicos = areaPatio / areaCeramico;
   ^
main.c:13:3: note: each undeclared identifier is reported only once for each function it appears in
main.c:8:3: warning: ignoring return value of 'scanf', declared with attribute warn_unused_result [-Wunused-result]
   scanf("%f",&ladoCeramico);
   ^
main.c:9:3: warning: ignoring return value of 'scanf', declared with attribute warn_unused_result [-Wunused-result]
   scanf("%f",&largoPatio);
   ^
```

Ejemplo de problema más complejo

Ejecución y prueba (./patio)

```c
#include <stdio.h>
/* léxico */
float ladoCeramico, largoPatio, anchoPatio, areaPatio, areaCeramico, cantCeramicos;

/* función principal (main) en todo programa C */
void main(){
  scanf("%f",&ladoCeramico);
  scanf("%f",&largoPatio);
  scanf("%f",&anchoPatio);
  areaCeramico = ladoCeramico * ladoCeramico;
  areaPatio = largoPatio * anchoPatio;
  cantCeramicos = areaPatio / areaCeramico;
  printf("%f",cantCeramicos);
}
```

input

```
0.3
10.5
5.9
688.333313

...Program finished with exit code 10
Press ENTER to exit console.
```


Problemas

Pasos para solucionar un problema

- **Análisis**

  El problema debe ser claramente especificado y entendido.

- **Diseño**

  Construcción de una solución general del problema.

- **Implementación**

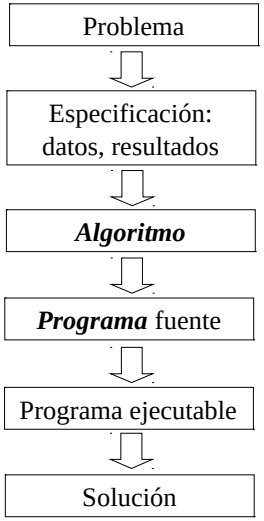
  Traducción del algoritmo a un lenguaje de programación de alto nivel.

- **Compilación**

  Traducción del programa a un lenguaje entendido por la computadora.

- **Ejecución y Prueba**

  Corrida y prueba de funcionamiento del programa en la computadora.



Computadora

Definición: Máquina capaz de aceptar datos a través de un medio de entrada, procesarlos automáticamente bajo el control de un programa y proporcionar la información resultante a través de un medio de salida.

Componentes

HW

- **Unidad Central de Proceso**
- **Unidad de Memoria**
- **Dispositivos de Entrada/Salida**

SW

- **Base (S.O.)**
  - **DOS, UNIX, Windows, MAC, etc.**
- **Aplicación**
  - **Utilitarios**
  - **Lenguajes de programación**
  - **Esparcimiento**
  - **Educativos**
  - **Otros**

HW
Unidad Central de Proceso (UCP/CPU)

Controla el procesamiento de la información

Unidad de Control:
- Carga instrucciones en memoria
- Interpreta y
- Devuelve el resultado de la ejecución

Unidad Aritmética y Lógica:
- Proceso de operaciones aritméticas y lógicas
- Provee decisión a la Unidad de Control

HW
Unidad de Memoria

Memoria principal (o primaria):
- Conjunto de celdas de memoria direccionables vía un único nombre
- Acceso directo por referencia para:
  - Carga
  - Recuperación de información
- Programas residen en este tipo de memoria al ser ejecutados

Memoria secundaria:
- Permiten almacenar gran cantidad de información
- Información persistente
- Ejemplos: Discos duros, diskettes, CDRom, ZIP.

HW
Dispositivos de Entrada/Salida (Input/Output)

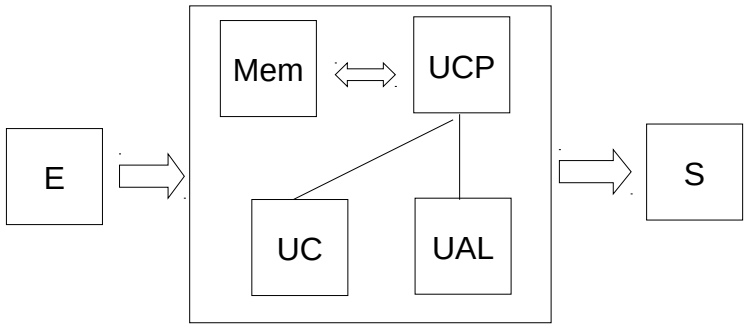
Los dispositivos de E/S permiten la comunicación ente el usuario y el computador.

Dispositivos de entrada:
- Teclados
- Ratón
- Lectores de disco
- etc.

Dispositivos de entrada/salida:
- Placa de red
- Modem
- Placa de sonido
- etc.

Dispositivos de salida:
- Impresoras
- Pantalla
- etc.

HW
Modelo Refinado de Computadora



SW
Lenguajes de Programación

Tipos de instrucciones ejecutadas por una computadora:

- de E/S: lectura/escritura de información
- Lógico-aritméticas
- Secuencia: Primero ejecutar una instrucción y luego otra
- Selección: si ... entonces ... sino ...
- Ciclo: Repetición de una secuencia de instrucciones
- Procedimiento: Grupo de instrucciones que pueden ser referenciadas y ejecutadas

SW
Instrucciones para la computadora

- Contenido de memoria expresado en *bits* (dígitos binarios)
- Datos e instrucciones en el mismo lenguaje
- El lenguaje de máquina depende del diseño y del hardware del computador (por ej., diferentes máquinas pueden representar las instrucciones con códigos diferentes)

SW
Lenguajes de bajo nivel

Lenguajes con un pobre nivel de abstracción, en el sentido de que sus instrucciones se asemejan mucho a las de máquina

Ejemplo: **Lenguaje de Ensamblado (Assembler)**

- En este tipo de lenguajes las instrucciones son nombradas
por códigos mnemotécnicos.
Por ejemplo: ADD, SUB, MPY, DIV, etc.

- Las instrucciones son traducidas a código de máquina mediante un *ensamblador*.

SW
Ejemplo de lenguaje Assembler

```
...
leal U_SYSWIN32_OUTPUT,%edi
movl %edi,-4(%ebp)
pushl     $.L7
pushl     -4(%ebp)
pushl     $0
call FPC_WRITE_TEXT_SHORTSTR
pushl     -4(%ebp)
call FPC_WRITELN_END
pushl     $.L4
call FPC_IOCHECK
...
```

SW
Lenguajes de alto nivel

• Lenguajes con un mayor nivel de abstracción, en el sentido de que sus instrucciones asemejan al lenguaje natural.

• Pueden ser independientes de la máquina. Por lo tanto, pueden ser traducidos a distintos lenguajes de máquina.

Ejemplos
Pascal, C, C++, Java, Visual BASIC, COBOL, Fortran.

SW
Lenguaje C

• Lenguaje de alto nivel que usaremos en la materia.

• Fue creado en 1972 por Dennis Ritchie, en los Laboratorios Bell, como evolución del lenguaje B.

• En 1983 el Instituto de Estándares Americanos estableció un estándar que definiera al lenguaje C, conocido como ANSI C.

• Los principales compiladores de C llevan implementado el estándar ANSI C.

• Utilizado para la implementación del sistema UNIX, y muchas aplicaciones complejas.

SW
Ejemplo de un programa C

Este programa muestra por pantalla el mensaje “Hola mundo”.

```
#include <stdio.h>

int main()
{
        printf("Hola mundo");
        return 0;
}
```


SW
Características de un buen programa

- Confiabilidad
- Adaptabilidad
- Legibilidad

SW
Fracasos - Crisis del software

• **Sistema de Control de Tráfico Aéreo**: FAA-IBMcancelado: 144 M$ - resto: atraso 5 años, 1.400 M$
• **Sonda Mariner 1** (Venus) fallo: error de software
• **Taxi espacial**: 5 computadoras redundantes: demora 2 días ('81), un día ('85), pierde Intelsat 6 ('92),...
• **Sistema telefónico de AT&T**: corte de 1 día, error de tipos
• **Sistema de trenes alemán**: bloqueo de 1 día por Memoria llena
• **American Airlines+Marriott+Hilton+Budget**: 1992 Integración de reservas, abandonado, 165 M$
• **6 fantasmas en el radar:** aeropuerto de San Francisco (EE.UU.) 9/1/01.
• **Tren noruego se detuvo el 31 Dic 2000**: el 2000 tenía 54 semanas en vez de 53.
• **Cohete ucraniano Tsiklon-3**: motores del cohete se apagaron a 367s. del despegue, 6 satélites (Ene/01)

Bibliografía de ésta teoría

• Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”:
- Introducción, Algoritmo, Léxico, Notación algorítmica (pags. 1 - 34)
- Composición secuencial (pags. 35 - 55)

• Biondi, J. y Clavel, G. “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”:
• Notación algorítimica (pags. 1 – 12)
• Entorno, Tipos, Variables, Constantes, Notación algorítmica (pags. 13 – 34)
• Composición condicional (35 - -53)
• Procesadores, Lenguajes (pags. 127 – 140)
• Introducción a Lógica (pags. 203 – 204)

• Wirth, N. “Algoritmos + Estruturas de Datos = Programas”: Presentación y Prólogo muy interesantes. Tipos (pags. 1 – 12).

• Quetglás, Toledo, Cerverón. "Fundamentos de Informática y Programación“. Capítulos 1 y 2. http://robotica.uv.es/Libro/Indice.html.