RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

CLASE 5

Hardware y Software.

Buenas practicas de programación

Luciano H. Tamargo
http://cs.uns.edu.ar/~lt
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2016

CONCEPTOS: HARDWARE

- ***Hardware:*** es una palabra inglesa (literalmente: partes duras); como no posee una traducción adecuada, fue admitida por la Real Academia Española que lo define como:

«Conjunto de los componentes que integran la parte material de una computadora».

- ***Hardware*** corresponde a todas las partes tangibles de un sistema informático; sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos.
- Son cables, gabinetes, periféricos de todo tipo y cualquier otro elemento físico involucrado; contrariamente, al soporte lógico que es intangible y es llamado ***software***.

[Wikipedia]

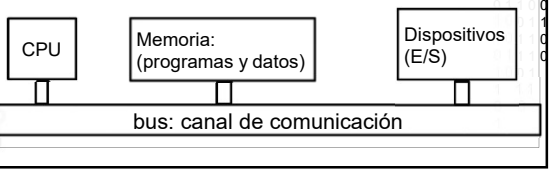
COMPUTADORA: ARQUITECTURA VON NEUMANN

- Una computadora es un sistema digital con tecnología microelectrónica compuesta por:
  1. CPU (Unidad Central de Proceso)
  2. Memoria
  3. Dispositivos de Entrada y Salida

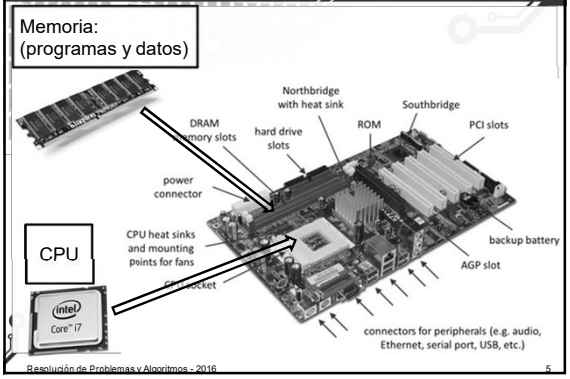
  Interconectados por un canal de comunicación (bus).

COMPUTADORA: ARQUITECTURA VON NEUMANN

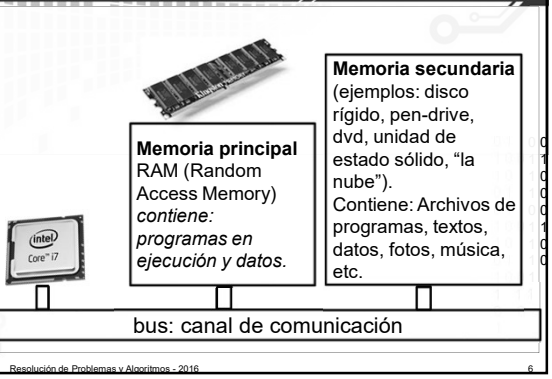
- Esta estructura de una computadora no fue siempre así…
- Esta arquitectura de computadora fue ideada por un grupo de científicos de la universidad de Pennsylvania en 1945. http://es.wikipedia.org/wiki/EDVAC
- *Una **arquitectura de computadora** es un modelo conceptual que define la estructura de una computadora, indicando (generalmente en forma gráfica) los elementos que la componen y como se relacionan.*



PLACA MADRE CON CPU, MEMORIA Y BUSES



COMPUTADORA: ARQUITECTURA VON NEUMANN



CONCEPTOS: UNIDADES DE REPRESENTACIÓN DE INFORMACIÓN

- Un **bit** es la unidad mínima para representar información en un sistema informático.
- Permite almacenar dos valores diferentes (generalmente representados con 0 y 1).
- La palabra bit es el acrónimo **Binary digit ('dígito binario').**
- **Byte** se utiliza como unidad en la medida de capacidad de almacenamiento de un dispositivo.
- Un **byte equivale a 8 bits** y permite 2^8 = 256 valores diferentes.
- Byte proviene de bite (en inglés "**mordisco**"), como la cantidad más pequeña de datos que una CPU podría "morder" a la vez (en los tiempos que fue creada esta palabra).

MEMORIA RAM

RAM concreta y real: módulo 4Gb DDR3

Abstracción de bajo nivel: representación binaria (secuencia de bytes)

0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0

CONCEPTOS: VALORES DE VARIABLES EN PASCAL

- Las variables de los tipos de datos SIMPLES vistos hasta el momento en esta materia tiene las siguientes características:

1. Residen en **memoria principal** (RAM, *random-access memory o memoria de acceso aleatorio*).
2. Los **valores** que contienen **no perduran** cuando termina la ejecución del programa. (El espacio de memoria utilizado por esas variables es liberado y usado por otros programas).
3. La **cantidad** de **memoria** que usan es **fija** (no cambia en ejecución) y el compilador usa este dato para reservar lugar en memoria.

ALMACENAMIENTO EN MEMORIA

Tipo de Dato: define el **conjunto de valores** posibles que puede tomar una variable, **las operaciones** que pueden aplicarse, y cual es la **representación interna**.

- Las variables de los siguientes tipos usan un espacio fijo en memoria (pero puede variar de un compilador a otro).
- Al ser fijo no cambia durante la ejecución del programa.
- En Free Pascal (Lazarus) ocupan:

Tipo de dato	Tamaño	Valores posibles
Char	1 byte	2^8 = 256
Boolean	1 byte	2
Integer	4 bytes	2^{32}= 4.294.967.296
Real	6 bytes	2^{48}= 281.474.980.000.000

1 byte = 8 bits 4bytes = 32 bits 6 bytes = 48 bits

MEMORIA RAM

RAM concreta y real: módulo 4Gb DDR3

Abstracción de bajo nivel: representación binaria (secuencia de bytes)

0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0

Abstracción de alto nivel (una traza en RPA)

LETRA:	A
ES_PAR:	FALSE
SIMBOLO:	+
NUM:	2016

UN PROGRAMA EN EJECUCIÓN

- Un programa debe estar en la memoria principal (RAM) para que sea ejecutado por una CPU.
- Durante la ejecución del programa los valores de las variables también se almacenan en la memoria.
- La declaración de las variables en Pascal permiten indicar que valores puede tomar y de estar forma conocer el espacio que debe **reservarse** en memoria para cada variable.
- Las primitivas de asignación, *read* y *readln*, son las encargadas de dar o cambiar el valor almacenado en el espacio reservado.

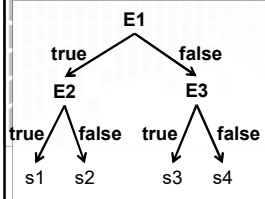
CONCEPTOS DE LAS CLASES ANTERIORES

¿Qué ocurre si a MAXINT le sumo 1?

```
PROGRAM Enteros;
var N:integer;
BEGIN
writeln('    Maxint = ',maxint);
N:=maxint+1;
writeln('Maxint + 1 = ',N);
END.
```

```
Maxint=2147483647
Maxint+1=-2147483648
```


CONDICIONALES "ANIDADOS"




```
IF <exp. boolean E1> THEN
  IF <exp. boolean E2> THEN
    <s1>
  ELSE
    <s2>
ELSE
  IF <exp. boolean E3> THEN
    <s3>
  ELSE
    <s4>
```

¿Bajo que condiciones se ejecutará la sentencia s3?

¿El resultado de E2 afecta a la ejecución de s3?

PROBLEMA

Problema: Escribir un programa que dado un mes y un año, muestre cuantos días tiene ese mes.

Solución: "30 días trae noviembre, con abril, junio y septiembre; de 28 sólo hay uno, y los demás son de 31", (si el año es bisiesto es 29)

Algoritmo

Obtener los valores de mes (1 a 12) y año
SI el mes es 2 (febrero) ENTONCES:
  SI año es bisiesto ENTONCES
    son 29 días,
  DE LO CONTRARIO 28
DE LO CONTRARIO (esto es, no es febrero)
  SI el mes es 11, 4, 6 o 9 ENTONCES
    son 30 días
  DE LO CONTRARIO
    son 31

Casos de prueba:

mes	año
11	2015
3	2014
12	2014
2	2015
2	2016

POSIBLE SOLUCION PARA "DIAS DE UN MES"

```
PROGRAM CantDiasMes;
VAR mes, anio, cant_dias: INTEGER;
BEGIN {computa la cantidad de días de un mes para un año dado}
 write(' Ingrese mes (1 a 12) y año: ');
 readln(mes, anio);{no realiza control de ingreso de datos}
 IF (mes = 2) THEN {… febrero depende si es año bisiesto …}
   IF (anio mod 4=0) and (anio mod 100<>0) or
      (anio mod 400=0) THEN
     cant_dias := 29
   ELSE
     cant_dias := 28
 ELSE {… en los demás meses depende sólo del mes…}
   IF (mes = 11) OR (mes = 4) OR (mes = 6) OR
      (mes = 9) THEN
     cant_dias := 30
   ELSE
     cant_dias := 31;
 writeln('La cantidad de días para', mes,' es ',
         cant_dias);
END.
```

SOLUCION CON CONTROL DE INGRESO DE DATOS

```
PROGRAM CantDiasMes;
VAR mes, anio, cant_dias: INTEGER;
BEGIN {computa la cantidad de días de un mes para un año dado}
 write(' Ingrese mes (1 a 12) y año: ');
 readln(mes, anio);
 IF (mes<1) OR (mes > 12) THEN {control de datos ing.}
   write('el MES ingresado es incorrecto')
 ELSE
   BEGIN
     IF (mes=2) THEN {febrero depende si es año bisiesto}
       IF (anio mod 4=0) and (anio mod 100<>0) or
          (anio mod 400=0) THEN
         cant_dias := 29
       ELSE
         cant_dias := 28
     ELSE {… en los demás meses depende sólo del mes…}
       IF (mes=11) OR (mes=4) OR (mes=6) OR (mes=9) THEN
         cant_dias := 30
       ELSE
         cant_dias := 31;
     writeln('Cant. de días para', mes,' es ', cant_dias);
   END;
END.
```

OTRA SOLUCION SIN ANIDAMIENTO DE IF

```
PROGRAM CantDiasMes;
VAR mes, anio, cant_dias: INTEGER; bisiesto: boolean;
BEGIN {computa la cantidad de días de un mes para un año dado}
 write(' Ingrese mes (1 a 12) y año: ');
 readln(mes, anio);
 bisiesto:= (anio mod 4=0) and (anio mod 100<>0) or
            (anio mod 400=0) ;
 IF (mes = 2) AND bisiesto THEN
   cant_dias := 29;
 IF (mes = 2) AND NOT bisiesto THEN
   cant_dias := 28;
 IF (mes=11) OR (mes=4) OR (mes=6) OR (mes=9) THEN
   cant_dias := 30;
 IF (mes=1) or (mes=3) or (mes=5) or (mes=7) or
    (mes=8) or (mes=10) or (mes=12) THEN
   cant_dias := 31;
 IF (mes < 1) OR (mes > 12) THEN {control de datos ing.}
   write(' el MES ingresado es incorrecto ')
 ELSE
   writeln('Cant. de días mes', mes,' es',cant_dias);
END.
```

OTRA SOLUCION SIN ANIDAMIENTO DE IF

```
PROGRAM CantDiasMes;
VAR mes, anio, cant_dias: INTEGER;
BEGIN
 write('Ingrese mes (1 a 12) y año: ');
 readln(mes, anio);
 IF (mes = 2) AND (anio mod 4=0) and
    (anio mod 100<>0) or (anio mod 400=0) THEN
   cant_dias := 29;
 IF (mes = 2) AND NOT ((anio mod 4=0) and
    (anio mod 100<>0) or (anio mod 400=0)) THEN
   cant_dias := 28;
 IF (mes=11) OR (mes=4) OR (mes=6) OR (mes=9) THEN
   cant_dias := 30;
 IF (mes=1) or (mes=3) or (mes=5) or (mes=7) or
    (mes=8) or (mes=10) or (mes=12) THEN
   cant_dias := 31;
 IF (mes < 1) OR (mes > 12) THEN {control de datos ing.}
   write(' el MES ingresado es incorrecto ')
 ELSE
   writeln('Cant. de días mes', mes,' es',cant_dias);
END.
```

EXPRESIONES Y CONDICIONALES ANIDADOS

Definición: un año es **bisiesto** si es múltiplo de 4 y no es múltiplo de 100 o es múltiplo de 400.

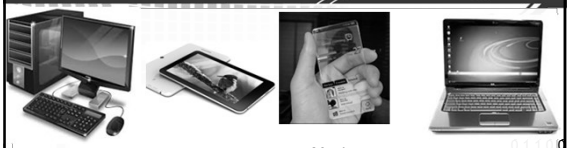
Con una expresión: (Note que si es mult. de 400 también es de 100 y de 4)
VAR anio: integer; bisiesto: boolean;
bisiesto := (anio mod 4=0) and not (anio mod 100=0) or (anio mod 400=0);

Casos de prueba:
1900
2000
2014
2016

Una expresión como la de arriba, en la cual se obtienen un resultado para la variable bisiesto, también puede programarse con condicionales anidados:

```
IF anio mod 4 = 0 THEN
  IF anio mod 100 = 0 THEN
    IF anio mod 400 = 0 THEN
      bisiesto := true
    ELSE
      bisiesto := false
  ELSE
    bisiesto := true
ELSE
  bisiesto := false
```


PROGRAMACION DE COMPUTADORAS A BAJO NIVEL



Acme Chimtel Maple Achepe

- Cada CPU de una computadora es capaz de ejecutar un único lenguaje llamado **lenguaje máquina**.
- Generalmente, cada marca de CPU tiene su propio lenguaje máquina.

¿Tengo que aprender el lenguaje máquina de cada CPU?

- Afortunadamente **NO** (como verá a continuación)

PROGRAMACION DE COMPUTADORAS A ALTO NIVEL

De esta manera, el **mismo código fuente** puede compilarse y luego ejecutarse en **distintas CPU**.

Código ejecutable en Acme | Código ejecutable en Chimpel | Código ejecutable en Achepe

Compilador para Acme | Compilador para Chimptel | Compilador para Achepe

Código fuente

El texto de un programa de computadoras se conoce como **código** fuente *(source code)*.

CONCEPTOS: COMPILACIÓN

Un **compilador** es un programa que traduce un programa, que está escrito en un lenguaje de programación, a otro lenguaje de programación.

- Este proceso de traducción se conoce como **compilación**.
- Un **compilador** permite traducir el código fuente de un programa, a otro lenguaje (típicamente lenguaje de máquina) permitiendo generar **código ejecutable**.

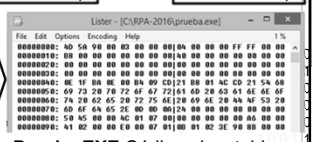
El **código ejecutable** es una secuencia de código que la computadora puede ejecutar directamente al ser invocado, (sin necesidad que el compilador esté presente.) Generalmente de extensión EXE o COM.

EDICION / COMPILACION / EJECUCIÓN

Edición: (por ejemplo con "Notepad") → Compilación (por ejemplo con "Free Pascal") → Ejecución (por ejemplo en PC c/linux)

```
program prueba;
var  N:integer;
begin
writeln('Maxint = ',maxint);
N:=maxint+1;
writeln('Maxint + 1  = ',N);
readln;
end.
```

Prueba.PAS Código fuente escrito en Pascal



Prueba.EXE Código ejecutable por la computadora

- Existen aplicaciones (como Lazarus) que brindan un **entorno de programación**, donde se puede editar, compilar (con Free Pascal) y ejecutar programas en Pascal dentro del mismo entorno.

ELEMENTOS PREDEFINIDOS

- La **definición** de un lenguaje de programación es algo **teórico.** A la definición original se la llama **Estándar.**
- Los **elementos predefinidos** (ej. el tipo **INTEGER**) son generalmente propuestos en la definición del lenguaje y luego provistos por el compilador (ej. Free Pascal).
- Así en un lenguaje de programación puede haber tipos predefinidos, constantes predefinidas, primitivas predefinidas, operaciones o funciones predefinidas.
- Las organizaciones o compañías que implementan un **compilador** deben respetar a la definición estándar.
- Pero muchas veces estos compiladores agregan elementos predefinidos y **extienden** al estándar. Como por ejemplo el tipo **LONGINT** en Free Pascal.

SOFTWARE

- **Software**: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

[Extraído del estándar 729 del IEEE]

- Considerando esta definición, el concepto de ***software*** va más allá de los programas de computación; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software: es decir, *abarca todo lo intangible*, todo lo «no físico» relacionado; en contraposición a los componentes físicos, que son llamados ***hardware***.

CONCEPTOS: SOFTWARE

- ***Software*** es una palabra inglesa (literalmente: partes blandas o suaves); como en español no posee una traducción adecuada, fue admitida por la Real Academia Española que lo define como:
- Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora .
- Se conoce como ***software*** *al equipamiento lógico o soporte lógico* de un sistema informático; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas.

SOFTWARE: CICLO DE VIDA

idea | desarrollo | uso y mantenimiento

análisis | diseño | implementación

- El software surge de una **necesidad**, **problema** o **idea**. Por ejemplo: páginas web para la materias del DCIC.
- Luego se **desarrolla**.
- Finalmente se **usa** durante un tiempo y de ser necesario se **modifica** (según la experiencia el 60% del ciclo de vida lo constituye esta etapa).
- Es **importante** utilizar **metodologías** y **técnicas** que simplifiquen el desarrollo y faciliten el mantenimiento (algunas de ellas vamos a comenzar a ver en RPA)

ETAPAS EN EL DESARROLLO DE SOFTWARE

desarrollo

DOCUMENTACIÓN → ←

1. Definición de requerimientos
2. Análisis del sistema
3. Diseño de sistemas.
4. Implementación de programas.
5. Pruebas unitarias (programas).
6. Pruebas de integración.
7. Entrega del sistema y Mantenimiento.

- En todas las etapas debe realizarse una documentación

RPA EN LAS ETAPAS DEL DESARROLLO DE SOFTWARE

RPA

1. Definición de requerimientos
2. Análisis del sistema
3. Diseño de sistemas.
4. Implementación de programas.
5. Pruebas unitarias (programas).
6. Pruebas de integración.
7. Entrega del sistema y Mantenimiento.

CONCEPTOS: PAUTAS PARA UN BUEN ESTILO DE PROGRAMACIÓN

- En la vida profesional del desarrollo de software, es sumamente importante que un programa (o algoritmo) pueda ser **entendido rápidamente** por una **persona** que lo tiene que leer.
- Para lograr esto, al escribir programas (por más modestos que sean), hay que seguir ciertas **pautas** de "**buen estilo de programación**".
- En las evaluaciones de RPA tendremos en cuenta las siguientes pautas de buena programación:
  1. Usar tipos de datos adecuados.
  2. Uso de nombres representativos en identificadores.
  3. Indentación (del inglés "indent").
  4. Comentarios en el código fuente.
     {…entre llaves…} // o al finalizar una línea

BUEN ESTILO DE PROGRAMACION: TIPOS DE DATOS APROPIADOS

- Tener tipos de datos para las variables permite **claridad y abstracción**: dos conceptos fundamentales en el desarrollo, mantenimiento y futuras actualizaciones del software.
- Usar nombres representativos ayuda a comprender el uso que se le da a un dato en un programa.
- Ejemplos

Dato a representar	Nombre de la variable	Tipo de dato predefinido
Sueldo de un empleado	Sueldo_empleado	REAL
Letra inicial del apellido	Inicial_apellido	CHAR
Si es o no es año bisiesto	Es_bisiesto	BOOLEAN
Año de nacimiento	Anio_nacimiento	INTEGER

USO DE NOMBRES REPRESENTATIVOS

```
IF (x1 > x2)
THEN
IF (x1 > w)
THEN z8:= x1
ELSE z8 := w
ELSE IF (x2 > w)
THEN z8 := x2
ELSE z8 := w
```

```
IF (num1 > num2)
THEN
IF (num1 > num3)
THEN maximo := num1
ELSE maximo := num3
ELSE IF (num2 > num3)
THEN maximo := num2
ELSE maximo := num3
```


INDENTACIÓN

```
IF (num1 > num2)
THEN
IF (num1 > num3)
THEN maximo := num1
ELSE maximo := num3
ELSE IF (num2 > num3)
THEN maximo := num2
ELSE maximo := num3
```

```
IF (num1 > num2) THEN
  IF (num1 > num3) THEN
    maximo := num1
  ELSE
    maximo := num3
ELSE
  IF (num2 > num3) THEN
    maximo := num2
  ELSE
    maximo := num3
```


COMENTARIOS EN EL CÓDIGO

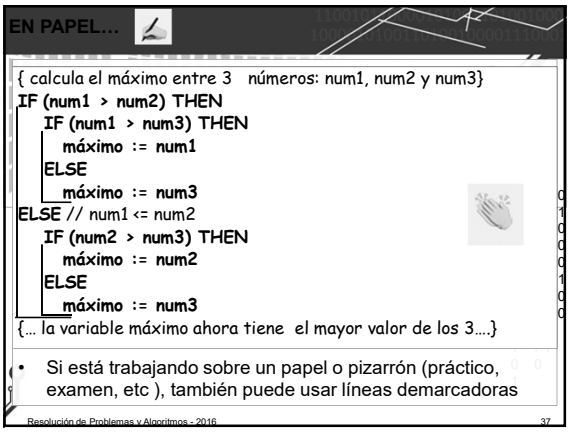
```
IF (num1 > num2) THEN
  IF (num1 > num3) THEN
    maximo := num1
  ELSE
    maximo := num3
ELSE
  IF (num2 > num3) THEN
    maximo := num2
  ELSE
    maximo := num3
```

```
{calcula el máximo entre 3
números: num1, num2 y num3}
IF (num1 > num2) THEN
  IF (num1 > num3) THEN
    maximo := num1
  ELSE
    maximo := num3
ELSE //  num1 <= num2
  IF (num2 > num3) THEN
    maximo := num2
  ELSE
    maximo := num3
{… la variable máximo ahora
tiene el mayor valor de los
3….}
```


COMPARE…

```
IF (x1 > x2)
THEN
IF (x1 > w)
THEN z8:= x1
ELSE z8 := w
ELSE IF (x2 > w)
THEN z8 := x2
ELSE z8 := w
```

```
{calcula el máximo entre 3
números: num1, num2 y num3}
IF (num1 > num2) THEN
  IF (num1 > num3) THEN
    maximo := num1
  ELSE
    maximo := num3
ELSE //  num1 <= num2
  IF (num2 > num3) THEN
    maximo := num2
  ELSE
    maximo := num3
{… la variable máximo ahora
tiene el mayor valor de los
3….}
```


EN PAPEL...
{ calcula el máximo entre 3 números: num1, num2 y num3}
IF (num1 > num2) THEN
IF (num1 > num3) THEN
máximo := num1
ELSE
máximo := num3
ELSE // num1 <= num2
IF (num2 > num3) THEN
máximo := num2
ELSE
máximo := num3
{... la variable máximo ahora tiene el mayor valor de los 3....}
• Si está trabajando sobre un papel o pizarrón (práctico, examen, etc), también puede usar líneas demarcadoras
Resolución de Problemas y Algoritmos - 2016
37