Resolución de Problemas y Algoritmos

Clase 4
Estructura de control condicional.

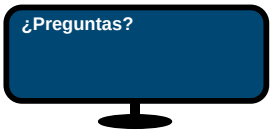
John von Neumann

Dr. Alejandro J. García
http://cs.uns.edu.ar/~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

Conceptos de las clases anteriores

- **Algoritmo. Primitiva. Traza.**
- **Lenguaje de programación. Programa. Código fuente.**
- **Pascal:**
  - **Identificadores reservados y predefinidos**
  - **Constantes, variables y tipos de datos.**
  - **Primitivas: asignación (:=) read, readln, write, writeln**
  - **Tipos predefinidos: real, integer, char, boolean.**
  - **Expresiones. Operaciones y funciones predefinidas.**




```
PROGRAM TextoPregunta;
CONST signo = CHR(168);
BEGIN
WRITE(signo,'Preguntas?');
readln {espera ENTER}
END.
```


Herramientas a disposición de los alumnos

En RPA ofrecemos a los alumnos estas herramientas que se complementan unas a otras:

1. **clase teórica** (presentación grupal, discusión grupal, reflexión, análisis, propuesta de metodologías)
2. **ejercicios en los prácticos** (trabajo individual, puesta en práctica de conceptos y metodologías)
3. **clase práctica** (atención personalizada para discusión y comprobación de los resultados obtenidos, puesta en común, reflexión, consultas ejecutando en computadoras)
4. **material en línea** (http://cs.uns.edu.ar/~wmg/rpa15lz/)
5. **evaluación** en máquina y parciales. Pone una meta en una fecha fija que ayuda a organizarse y poner en práctica las habilidades desarrolladas, como en la vida profesional. Además, el alumno recibe una devolución formal escrita sobre su desempeño.

Objetivos de la materia RPA

El objetivo principal es que los alumnos adquieran la capacidad de desarrollar programas de computadoras para resolver problemas de pequeña escala.

El desarrollo de un programa se concibe como un proceso que abarca varias etapas:

1. La interpretación adecuada del enunciado a través del cual se plantea el problema.
2. El diseño de un algoritmo que especifica la resolución del problema.
3. La implementación del algoritmo en un lenguaje de programación imperativo.
4. La verificación de la solución.

Etapas

El objetivo principal es que los alumnos adquieran la capacidad de desarrollar programas de computadoras para resolver problemas de pequeña escala

El desarrollo de un programa se concibe como un proceso que abarca varias etapas:

1. La interpretación adecuada del enunciado a través del cual se plantea el problema.
2. El diseño de un algoritmo que especifica la resolución del problema.
3. La implementación del algoritmo en un lenguaje de programación imperativo.
4. La verificación de la solución.

Concepto: lenguaje de programación

Un **lenguaje de programación** es un lenguaje artificial creado para expresar procesos que pueden ser llevados a cabo por computadoras. (Ejemplo: Pascal)

Un lenguaje de programación está definido por:

1. **un conjunto de símbolos**,
2. **reglas sintácticas** que definen su estructura, y
3. **reglas semánticas** que definen el significado de sus elementos.

Conceptos: Diagrama Sintáctico

- La **sintaxis** de un lenguaje de programación es un conjunto de reglas que indica la estructura de los programas en ese lenguaje.

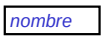
Un **diagrama sintáctico** (syntax diagram or railroad diagrams) es una forma gráfica de representar la sintaxis de un lenguaje de programación. Permite **describir sin ambigüedad** la sintaxis de un lenguaje de una manera simple y formal.

- Los diagramas sintácticos de Pascal que usaremos en RPA se encuentran en la página de la materia siguiendo este enlace: http://cs.uns.edu.ar/~wmg/rpa15lz/index.php?accion=download&dir=downloads//Practicos
- La sintaxis original escrita por N. Wirth está en la pág. 47 de : http://e-collection.library.ethz.ch/eserv/eth:3059/eth-3059-01.pdf

Elementos de un diagrama sintáctico

(1) **Un nombre y flecha** indican el comienzo de un diagrama para la definición de nombre.

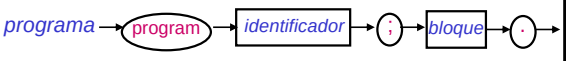
(2) **Las figuras "redondeadas"** indican que texto se debe incluir tal cual como aparece.



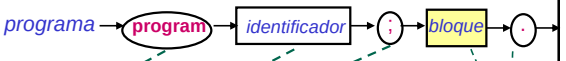
(3) Los **rectángulos** indican que nombre está definido en algún otro diagrama sintáctico.

(4) Las **flechas** indican el orden de lectura en el diagrama.

Ejemplo:

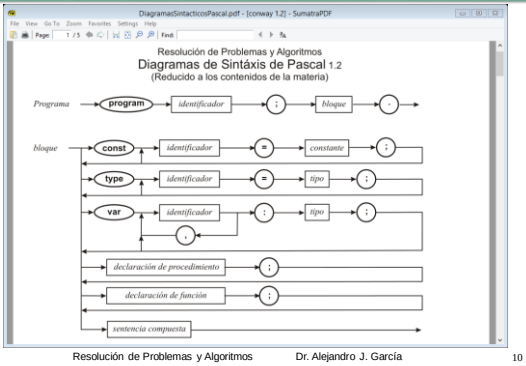


Diagramas sintácticos

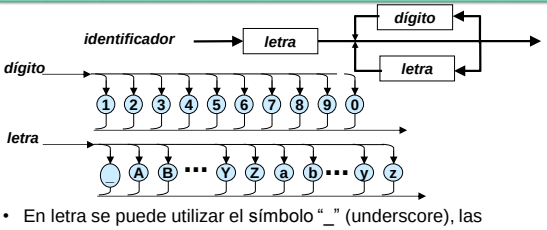



```
PROGRAM AreaCirculo;
CONST pi = 3.1416;
VAR area,radio: REAL;
BEGIN
    read(radio);
    area := pi * radio * radio;
    write("Area es", area);
END.
```


Parte del archivo disponible en la página de RPA



Diagramas sintácticos para "identificador"

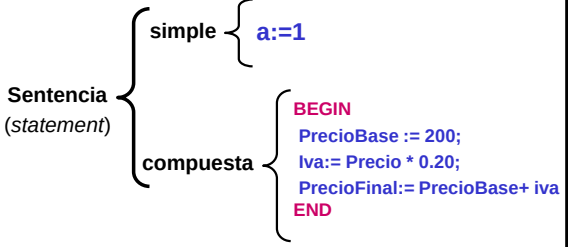


- En letra se puede utilizar el símbolo "_" (underscore), las mayúsculas: ABCDEFGHIJKLMNOPQRSTUVWXYZ y las minúsculas: abcdefghijklmnopqrstuvwxyz
- No pueden utilizarse por ejemplo: á, ó, Ú, ñ, Ñ, -.

IMPORTANTE: Los diagramas sintácticos completos pueden encontrarse en el material de la página web de la materia.

Sentencias en Pascal

Las sentencias *(statements)* en Pascal pueden ser simples o compuestas. En computación statement se suele traducir al castellano como sentencia, instrucción o proposición.



Sentencia compuesta en Pascal

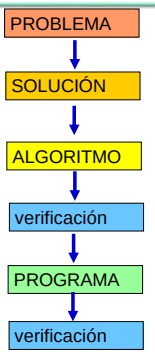
Una sentencia compuesta comienza con **BEGIN** y termina con **END** **y permite definir una secuencia de sentencias como si fuera una única sentencia.**
Por ejemplo, la siguiente es una sentencia
(compuesta, a su vez, por tres sentencias simples)

```
BEGIN
  PrecioBase := 200;
  Iva:= Precio * 0.20;
  PrecioFinal:= PrecioBase+ iva
END
```

El punto y coma es un separador de sentencias.

En la última sentencia de una secuencia el ";" no es necesario ya que no hay otra para separar de la última.

Metodología general propuesta



Veremos a continuación como codificar en el lenguaje Pascal una estructura condicional de la forma:

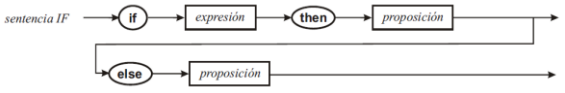
Si …condición…
entonces ….
de lo contrario …

Estructura de control condicional (IF-THEN-ELSE)

```
IF expresión de tipo boolean
THEN Sentencia 1
       (simple o compuesta)
ELSE Sentencia 2
       (simple o compuesta)
```

```
IF expresión boolean
THEN  begin
            …
       end
ELSE  begin
            …
       end
```

Sintaxis: ver el diagrama sintáctico.



Estructura de control condicional (IF-THEN-ELSE)

```
IF expresión de tipo boolean
THEN Sentencia 1
       (simple o compuesta)
ELSE Sentencia 2
       (simple o compuesta)
```

```
IF expresión boolean
THEN  begin
            …
       end
ELSE  begin
            …
       end
```

Sintaxis: ver el diagrama sintáctico.
Semántica:
Si la evaluación de la expresión lógica da resultado true, entonces se ejecuta solamente la sentencia que sigue al "THEN" (sea simple o compuesta) .
Si en cambio la evaluación de la expresión lógica da falso, entonces se ejecuta solamente la sentencia que sigue al "ELSE".

Problema simple propuesto

Problema: Escriba un programa en Pascal para obtener el valor absoluto de un número.

Solución:

Si el número es positivo o cero, el valor absoluto es el mismo número, de lo contrario es el número multiplicado por -1.

Algoritmo:

Leo el número Num
Si Num >= 0
entonces: val_abs es Num
de lo contrario: val_abs es Num * -1
Muestro val_abs en pantalla

Verificación:

Ejemplos significativos para casos de prueba: 3, 0 y -3 (uno positivo, uno negativo y cero)

Programa para valor absoluto

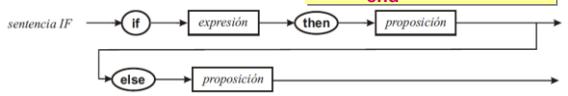
```
program valor_absoluto;
var    numero, val_abs: real;
{Realiza el cálculo del valor absoluto de un número}
begin
Write ('Ingrese un número'); readln(numero);
IF numero >= 0
   THEN val_abs := numero
    ELSE val_abs := (-1) * numero;
writeln(' Su valor absoluto es: ', val_abs);
end.
```

Importante: no lleva ";" antes del ELSE

Estructura de control condicional (IF-THEN)

```
IF expresión de tipo boolean
THEN Sentencia 1
     (simple o compuesta)
```

```
IF expresión boolean
THEN  begin
      ...
      end
```



Sintaxis: vea en el diagrama sintáctico que en la sentencia IF no es obligatorio un ELSE (es opcional).
Semántica: Si la evaluación de la expresión lógica da resultado true, entonces se ejecuta solamente la sentencia que sigue al "THEN" (sea simple o compuesta) .
Si en cambio la evaluación de la expresión lógica da false, se sigue con la ejecución de la sentencias que siguen al IF (si es que existen).

Problema simple propuesto

Problema: Considerando únicamente las letras a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z; escriba un programa que lea un caracter y distinga si se trata de una letra mayúscula o minúscula. Obs: para simplificar no incluimos las vocales con acentos ni la letra Ñ, pero lo haremos más adelante.

Solución: un caracter ASCII entre 'A' y 'Z' es una letra mayúscula, un caracter entre 'a' y 'z' es una letra minúscula.

Algoritmo:

- **leer el caracter**
- **Si está entre 'A' y 'Z' entonces es una mayúscula**
- **Si está entre 'a' y 'z' entonces es una minúscula**

Verificación:

casos de prueba: una mayúscula, una minúscula y un carácter que no sea una legra (ejemplos: 'G', 'g', '3' , '$')

Un programa posible…

```
program mayucula_o_minuscula;
var    ch: char;
{Este programa permite distinguir mayúsculas y minúsculas}
begin
write('Ingrese un caracter:');
readln(ch);
IF  (ch >= 'A') and (ch <= 'Z')
   then writeln(ch, ' es una mayúscula.');
IF  (ch >= 'a') and (ch <= 'z')
   then writeln(ch, ' es una minúscula.');
end.
```

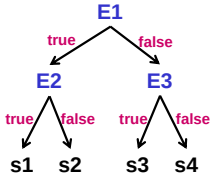
Hacer una traza
¿qué ejemplos usa?

Condicionales "anidados"

```
IF <exp. boolean>
THEN
   Sentencia
   (simple o
   compuesta)
ELSE
   Sentencia
   (simple o
   compuesta)
```

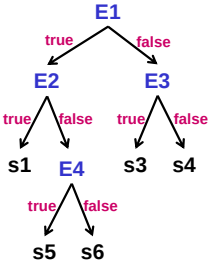
```
IF  <exp. Boolean E1 >
THEN
   IF < exp. boolean E2 >
   THEN < sentencia  >
   ELSE < sentencia  >
ELSE
   IF < exp. Boolean E3 >
   THEN < sentencia  >
   ELSE < sentencia  >
```


Condicionales "anidados"




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


El límite está en su imaginación




```
IF  <exp. boolean E1 >
THEN
   IF < exp. boolean E2 >
   THEN < s1 >
   ELSE  IF < exp.  E4 >
         THEN < s5 >
         ELSE < s6 >
ELSE
   IF < exp. boolean E3 >
   THEN < s3 >
   ELSE < s4 >
```

¿Tienen el mismo efecto?

Secuencia de condicionales

```
IF ( A > 10 )
   THEN write(1);
IF( B = 0 )
   THEN write(2);
IF( C > 20 )
   THEN write(3);
```

Condicionales ANIDADOS:

```
IF( A > 10 )
  THEN write(1)
  ELSE IF( B = 0 )
         THEN write(2)
         ELSE IF( C > 20 )
                THEN write(3);
```

Realice diferentes trazas con los siguientes casos de prueba

1) A = 20, B = 0, C = 100
2) A = 1, B = 0, C = 100
3) A = 1, B = 0, C = 1

¿Qué observa?

¿Tienen el mismo efecto?

Realice una traza con (i) A=20 y B=0; luego con (ii) A=1 y B=0.

```
IF ( A > 10 )
THEN write(1);
IF( B = 0 )
THEN write(2);
```

```
IF( A > 10 )
  THEN BEGIN
         write(1);
         IF ( B = 0 )
           THEN write(2);
       END;
```

¿Por qué con A=1 y B=0 tienen diferente efecto?

En el recuadro de la izquierda (celeste) hay una secuencia de dos sentencias condicionales (if-then) que son independientes entre si (observe que están separadas por un ";").

En cambio, a la derecha (mostaza), como hay un begin-end, el segundo if-then depende del primero ya que está "anidado" dentro del primero: se ejecutará solamente cuando el valor de A sea mayor a 10.

¿Tienen el mismo efecto?

Realice una traza con A=5 y B=6

```
IF A = B
THEN
     IF A = 5
   THEN WRITE('A es 5 ')
ELSE WRITE('DISTINTOS');
```

```
IF A = B
THEN
  BEGIN
     IF A = 5
     THEN WRITE('A es 5 ')
  END
ELSE WRITE('DISTINTOS');
```

El "ELSE" siempre se corresponde con el "IF-THEN" anterior más cercano que no tenga ELSE. Por lo tanto, en el ejemplo de la izquierda el "ELSE" se corresponde con el "IF A=5 THEN".

Sin embargo, utilizando "BEGIN - END" puedo forzar y hacer que se corresponda con otro IF-THEN. Esto ocurre en el ejemplo del bloque de la derecha donde el "ELSE" se corresponde con el "IF A=B THEN".

Nuevo problema propuesto

Escriba un programa en Pascal que lea un carácter (CHAR) y diga si se trata de una letra mayúscula, minúscula, o si se trata de otro símbolo.

Por ejemplo:
'G', es una mayúscula
'g', es una minúscula
'3' , es otro símbolo
'$', es otro símbolo

Siguiendo la metodología propuesta, escriba un algoritmo y un programa en Pascal que resuelva el problema.
Indique cuales son los casos de prueba que usó.

Problema propuesto: días de un año

Problema: Escribir un programa que dado un año, indique cuantos días tiene.

Solución:

En general son 365 días pero algunos años febrero tiene 29 días (años bisiestos) y son 366 ¿cuáles son años bisiestos? ¿por qué pasa esto?

- Un año "astronómico" tiene 365 días 5 h 48 m 45,25 s
- Un año calendario tiene 365 o 366 días (año bisiesto)

  vea http://es.wikipedia.org/wiki/Año_bisiesto

Definición: un año es **bisiesto** si es múltiplo de 4 y no es múltiplo de 100 o es múltiplo de 400.
Ej. 2016, 2008 y 2000 son bisiestos, 2009, 2010 y 1900 no lo son.

Calcular cuando un año es bisiesto

Definición: un año es **bisiesto** si es múltiplo de 4 y no es múltiplo de 100 o es múltiplo de 400.
Ej. 2004, 2008 y 2000 son bisiestos, 2009, 2010 y 1900 no lo son.

Con una expresión: (observe 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);
```

Con condicionales:

```
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
```

Casos de prueba:
4
100
400
1900
2000
2014
2015

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 (o 29) sólo hay uno, y los demás son de 31"

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
2	2015
2	2016
11	2015
3	2014
12	2014

Posible solución para "días 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.
```


Solución 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) {control de datos ingresados}
THEN 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('La cantidad de días para', mes,' es ', cant_dias);
  END;
END.
```


Otra solución 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;
writeln('La cantidad de días para', mes,' es ', cant_dias);
END.
```


Otra solución 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;
writeln('La cantidad de días para', mes,' es ', cant_dias);
END.
```

- **Hacer una traza**
- **Obs: un único writeln para el resultado**

Conceptos: Hardware [Wikipedia]

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***.

Computadora

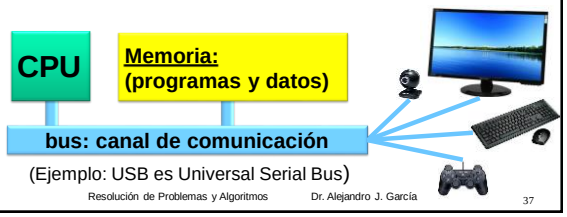
Una computadora es un sistema digital con tecnología microelectrónica compuesta por:

1- CPU (Unidad Central de Proceso)

2- Memoria

3- Dispositivos de Entrada y Salida

Interconectados por un canal de comunicación (bus)



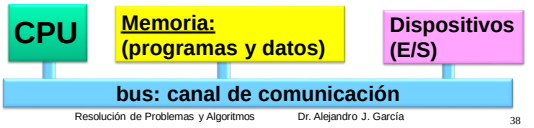
(Ejemplo: USB es Universal Serial Bus)

Computadora con 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 para el diseño de EDVAC (Electronic Discrete Variable Automatic Computer) 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.



Primeras computadoras: ENIAC & EDVAC

ENIAC (Electronic Numerical Integrator And Computer) fue diseñada por el físico John Mauchly, y el ingeniero electrónico John Eckert en 1943. La programación era por hardware (cables) y reprogramarla costaba días. Estaba a cargo de 6 mujeres programadoras con grandes habilidades matemáticas y lógicas, que iban inventando la programación a medida que la realizaban: Betty Snyder Holberton, Jean Jennings Bartik, Kathleen McNulty Mauchly Antonelli, Marlyn Wescoff Meltzer, Ruth Lichterman Teitelbaum y Frances Bilas Spence

Computadora ENIAC (1946) (http://es.wikipedia.org/wiki/ENIAC)

Primeras computadoras: ENIAC & EDVAC

Eckert y Mauchly conscientes de las limitaciones de ENIAC empezaron con un nuevo diseño para una nueva computadora (EDVAC). Eckert propuso una memoria de mercurio para guardar tanto el programa como datos.

El matemático Goldstine también era parte del proyecto EDVAC, y en una charla casual (en una estación de tren) entusiasmó al matemático húngaro John von Neumann a formar parte del proyecto.

La arquitectura de EDVAC incorporó el concepto de programa almacenado en memoria y codificación en binario en lugar de decimal.

Computadora EDVAC (1945) (http://es.wikipedia.org/wiki/EDVAC)

El origen del término arquitectura von Neumann

John von Neumann fue una persona muy inteligente y activa que trabajó en numerosas áreas: matemática, lógica, programación, energía atómica, física y computación teórica. [ver más]

El término arquitectura de von Neumann surgió a partir del primer reporte del diseño de la EDVAC que fue escrito (a mano) en 1945 por John von Neumann mientras esperaba una conexión de tren en una estación. Luego envió el reporte por correo a Goldstine quien lo tipió y distribuyó apresuradamente entre sus colegas por todo el mundo, dejando como autoría solo el nombre de Von Neumann (sin incluir a Mauchly y Eckert).

John von Neumann

Placa madre con CPU memoria y buses

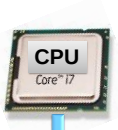
Memoria (RAM):

Computadora con arquitectura von Neumann

Memoria principal
RAM (Random Access Memory)
contiene: programas en ejecución y datos

Memoria secundaria
(ej: disco rígido, pen-drive, dvd, unidad de estado sólido, "la nube")

Contiene: Archivos de programas, textos, datos, fotos, música, etc.



bus: canal de comunicación

Memoria RAM

RAM concreta y real: módulo 4Gb DDR3

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

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

abstracción de alto nivel (una traza de RPA)

NUM:	123
ES_PAR:	FALSE
LETRA:	A
PRECIO:	12.02

Representación de información: bit y byte

- 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.

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

Las variables de los siguientes tipos usan un espacio en memoria que es fijo que no cambia en ejecución (pero puede variar de un compilador a otro).

En Free Pascal (Lazarus) generalmente 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

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 permite 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.

Continuará