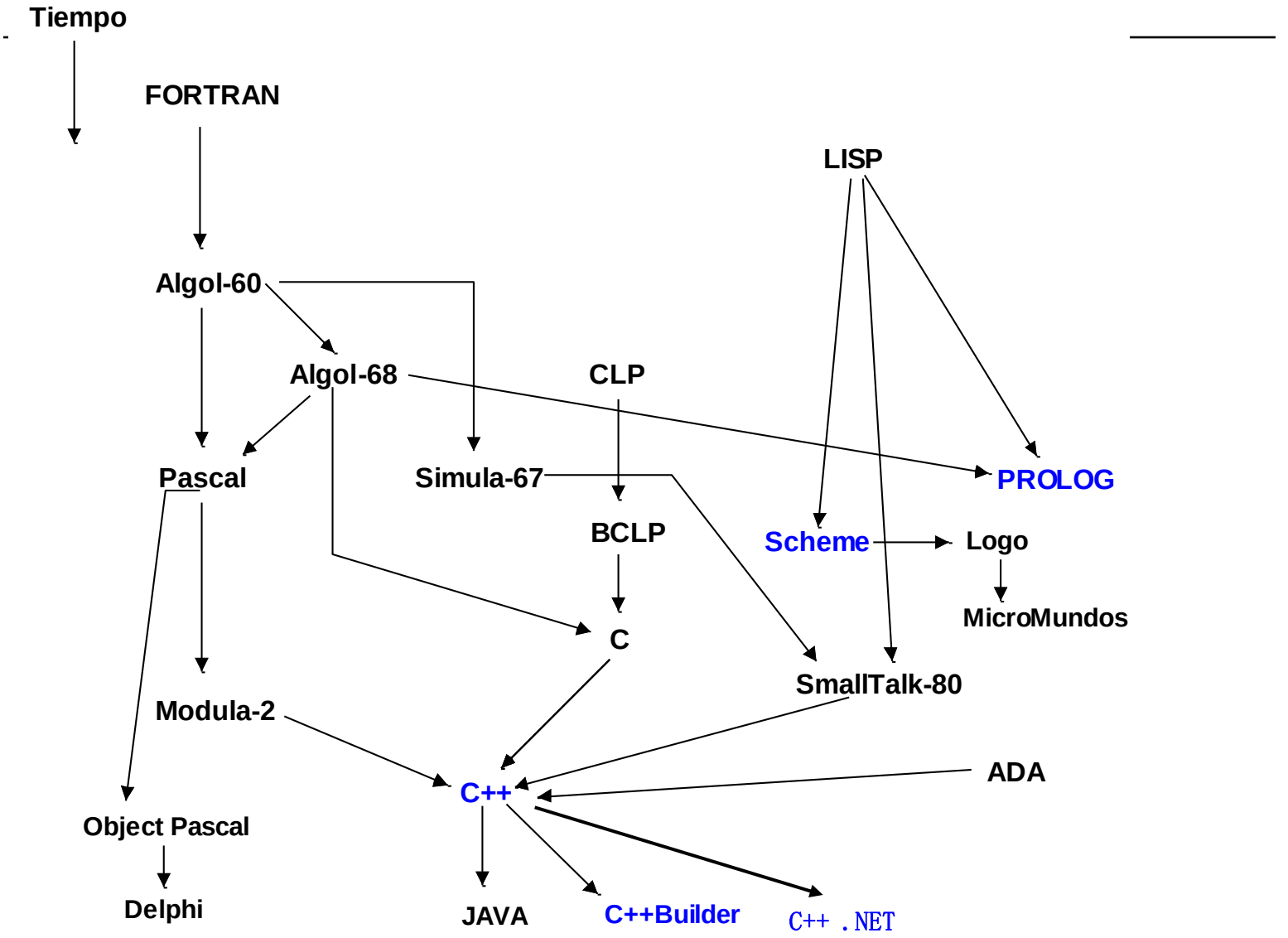
Evolución de los Lenguajes de Programación

Dr. Oldemar Rodríguez
Escuela de Informática
Universidad Nacional

¿Qué es un Paradigma de programación?

- Un **paradigma de programación** representa un enfoque particular o filosofía para la construcción del software.
- No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

Evolución de los Lenguajes de Programación



Origen de los lenguajes

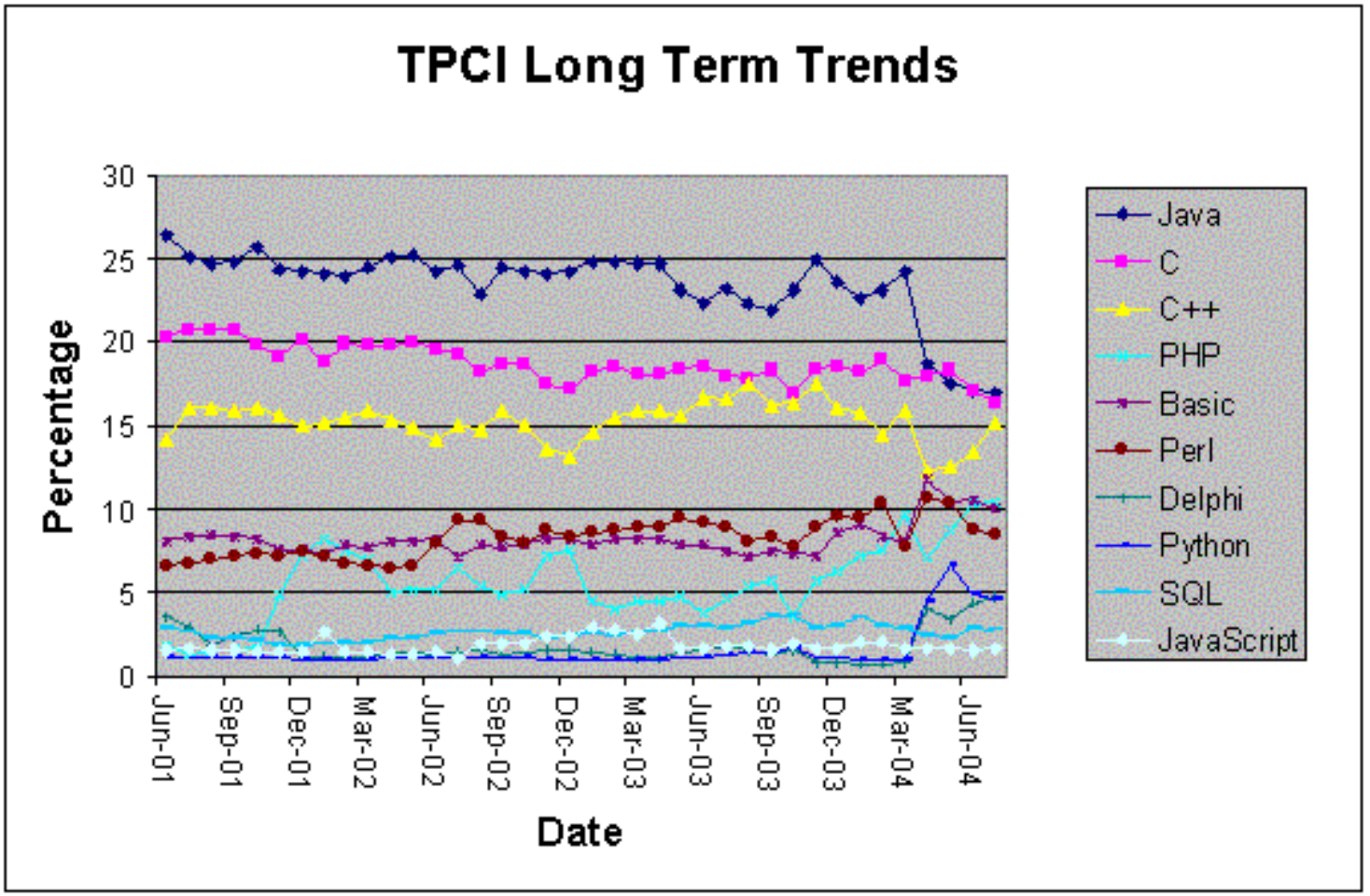
Lenguaje	Creador	Año
Algol-60	Wirth y Hoare	1966
Pascal	Wirth	1971
Modula-2	Wirth	1983
Oberon	Wirth	1988
CPL	Strachey	1966
BCPL	Richards	1969
C	Dennis Ritchie	1972
C++	Bjarne Stroustrup	1986
Lisp	John McCarthy	1958
PROLOG	Alain Colmeraurer y Phillipe Roussel	1972
Logo	Seymour Papert	

Position	(Position)	Programming Language	Ratings	(Ratings)	Status
1	↑	C	19.696%	+1.22%	A
2	↓	Java	16.332%	-8.68%	A
3	=	C++	11.914%	-5.51%	A
4	↑↑	PHP	11.448%	+5.71%	A
5	=	(Visual) Basic	8.168%	+0.93%	A
6	↓↓	Perl	7.420%	-1.51%	A
7	=	SQL	3.445%	+0.52%	A
8	↑↑	Python	3.029%	+1.93%	A
9	↑↑	Delphi/Kylix	2.977%	+2.15%	A
10	↓↓	C#	1.947%	+0.24%	A
11	↓↓	JavaScript	1.595%	-0.14%	A
12	=	SAS	1.416%	+0.61%	A
13	=	COBOL	0.985%	+0.20%	A
14	↑↑↑↑	IDL	0.774%	+0.36%	A--
15	↑↑↑↑↑↑↑↑↑	ABAP	0.735%	+0.49%	A--
16	=	Lisp	0.598%	+0.09%	B
17	↓↓↓	Pascal	0.562%	+0.07%	B
18	↓↓↓	Fortran	0.499%	-0.06%	B
19	↓↓↓	Ada	0.462%	-0.08%	B
20	↑↑↑↑↑	MATLAB	0.444%	+0.20%	B

Año 2000

Position	Delta 1 Year	Programming Language	Ratings	Delta 1 Year	Status
1	=	Java	16.997%	-6.21%	A
2	=	C	16.335%	-1.64%	A
3	=	C++	15.306%	-1.27%	A
4	↑↑	PHP	10.427%	+5.75%	A
5	=	(Visual) Basic	10.136%	+2.67%	A
6	↓↓	Perl	8.440%	-0.54%	A
7	↑↑	Delphi/Pascal/Kylix	4.814%	+2.94%	A
8	↑↑↑	Python	4.704%	+3.41%	A
9	↓↓	SQL	2.856%	-0.14%	A
10	=	JavaScript	1.681%	-0.11%	A
11	↓↓↓	C#	1.633%	-0.37%	A
12	↑	SAS	0.729%	-0.25%	A
13	↓	COBOL	0.526%	-0.54%	B
14	↑↑↑↑↑↑	IDL	0.349%	-0.12%	B
15	=	Lisp	0.328%	-0.34%	B
16	↓↓	Fortran	0.327%	-0.54%	B
17	↑	Ada	0.317%	-0.19%	B
18	↑↑↑↑↑	MATLAB	0.276%	-0.05%	B
19	↓↓↓	RPG	0.276%	-0.38%	B
20	↓	Prolog	0.259%	-0.24%	B

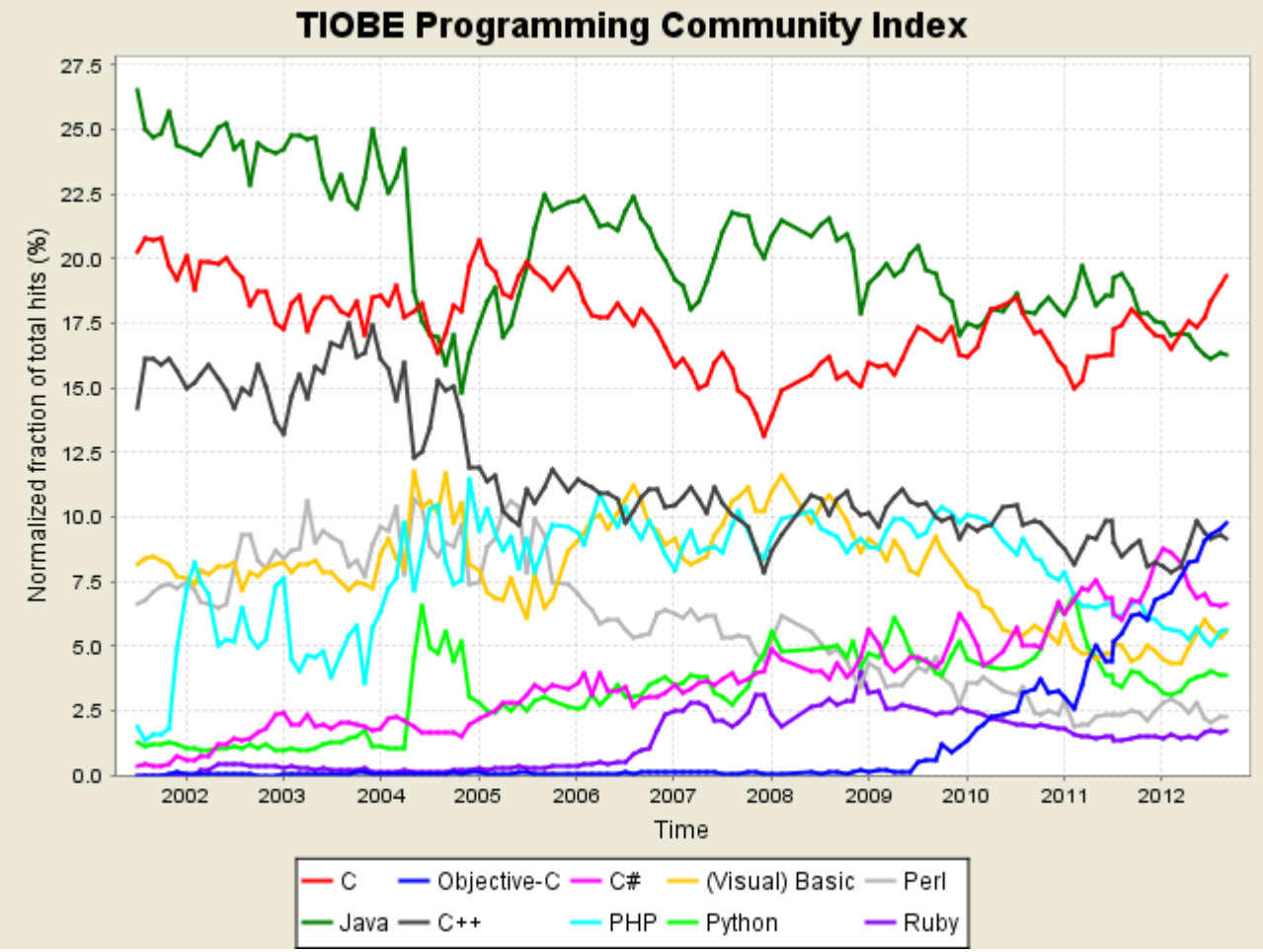
Año 2004



Programming Language Popularity: The TCP Index for 2004

Programming Language Popularity: The TCP Index for December, 2012

Position Sep 2012	Position Sep 2011	Delta in Position	Programming Language	Ratings Sep 2012	Delta Sep 2011	Status
1	2	↑	C	19.295%	+1.29%	A
2	1	↓	Java	16.267%	-2.49%	A
3	6	↑↑↑	Objective-C	9.770%	+3.61%	A
4	3	↓	C++	9.147%	+0.30%	A
5	4	↓	C#	6.596%	-0.22%	A
6	5	↓	PHP	5.614%	-0.98%	A
7	7	=	(Visual) Basic	5.528%	+1.11%	A
8	8	=	Python	3.861%	-0.14%	A
9	9	=	Perl	2.267%	-0.20%	A
10	11	↑	Ruby	1.724%	+0.29%	A
11	10	↓	JavaScript	1.328%	-0.14%	A
12	12	=	Delphi/Object Pascal	0.993%	-0.32%	A
13	14	↑	Lisp	0.969%	-0.07%	A
14	15	↑	Transact-SQL	0.875%	+0.02%	A
15	39	↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	0.840%	+0.53%	A
16	16	=	Pascal	0.830%	-0.02%	A
17	13	↓↓↓↓	Lua	0.723%	-0.43%	A-
18	18	=	Ada	0.700%	+0.02%	A--
19	17	↓↓	PL/SQL	0.604%	-0.12%	B
20	22	↑↑	MATLAB	0.563%	+0.02%	B


TIOBE Programming Community Index
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
Time
C
Objective-C
C#
(Visual) Basic
Perl
Java
C++
PHP
Python
Ruby

Category	Ratings Sept 2012	Delta Sept 2011
Object-Oriented Languages	57.1%	+0.9%
Procedural Languages	38.1%	+0.4%
Functional Languages	3.2%	-1.1%
Logical Languages	1.7%	-0.1%

Category	Ratings Sept 2012	Delta Sept 2011
Statically Typed Languages	71.8%	+0.1%
Dynamically Typed Languages	28.2%	-0.1%

Elementos de un lenguaje de programación

- **Sintaxis**: La sintaxis de un lenguaje especifica cómo están construidos los programas en ese lenguaje.
- S***emántica***: La semántica de un lenguaje especifica el significado del programa.
- Funciones definidas por el programador.
- ***Tipos***: El tipo de una expresión nos indica los valores que ésta puede representar y las operaciones que pueden aplicarse a ella.

Elementos de un lenguaje de programación

- ***Sobrecarga***: un símbolo, y en particular un operador, está sobrecargado cuando posee significados distintos en contextos distintos.
- ***Herencia***.
- ***Backtraking***.
- ***Lazy evaluation***.

Programación Estructurada

- La programación estructura surge ante la necesidad de hacer programas cada vez más entendibles o legibles.
- Surge en el año de 1968.
- Se fundamenta en el flujo de control estructurado, es decir uso constructores de bifurcación, bifurcación, y repetición en lugar de usar los saltos “GO TO”.
- Es decir el flujo de control es evidente del texto del programa, mediante el uso del *if, case, repeat, do, while.*
- Los lenguajes estructurados utilizan también funciones, procedimientos, parámetros y datos estructurados (arreglos, registros, etc.).

Programación Estructurada en C

- While
  - Sintaxis:

```
while(expresion) sentencia
```

- Ejemplo:

```
#include <stdio.h>
 void main() {
     int numero=1;
     while(numero<=100) {
            printf("%d\n",numero);
            ++numero;
     }
 }
```

- do while
  - Sintaxis:

    do sentencia while(expresion)

- Ejemplo:

```
#include <stdio.h>

        void main() {
                int numero=1;
                do
                        printf("%d\n",numero++);
                while(numero<=100);
        }
```

- for
  - Sintaxis:

```
for(expresion1; expresion2; expresion3)
                    sentencia
```

- Una sentencia for es siempre equivalente a la siguiente expresión while:

```
expresion1;
while(expresion2) {
      sentencia;
      expresion3;
}
```

- for
  - Sintaxis:

```
for(expresion1; expresion2; expresion3)
                    sentencia
```

- Ejemplo:

```
#include <stdio.h>
void main() {
    int numero;
    for(numero=1; numero<=100; numero+=2)
        printf("%d\n",numero);
}
```

- if - else
  - Sintaxis:

    `if(expresion) sentencia1 else sentencia2`

- Ejemplo:

```
scanf("%d",&numero);
if((numero%2)==0)
        printf("El número %d es par",numero);
else
        printf("El número digitado no es par");
```

- switch
  - Sintaxis:

    switch (expresion) sentencia

- Ejemplo:

```
switch (color = toupper(getchar())) {
     case 'R':
            printf("COLOR ROJO");
            break;
     case 'B':
            printf("COLOR BLANCO");
            break;
     case 'A':
            printf("COLOR AZÚL");
            break;
     default:
            printf("ERROR");
     }
```

Funciones en C:

- La sintaxis general de una función es la siguiente:

```
tipo nombre_función(declaración de parámetros)
  {
   cuerpo de la función
}
```

Ejemplo:

```
float Potencia(float x,int n) {
   float exp=1;
   if(x<0)
   return -1;
   else
      for(int i=1; i<=n; i++)
         exp*=x;
   return exp;
}
```

Procedimientos:

- **Ejemplo:**

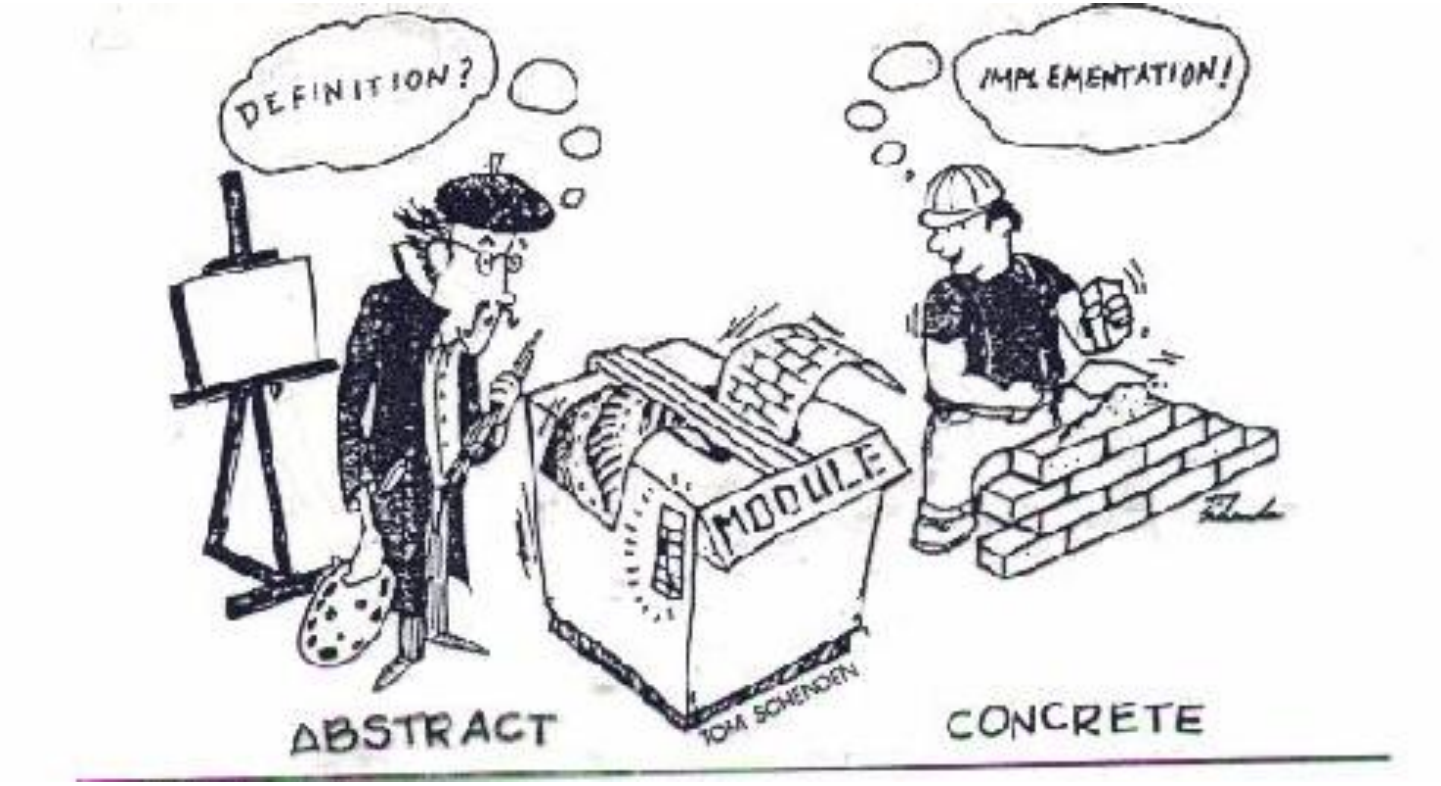
```
void imprime(int n) {
 for(int i=2;i<=n;i+=2)
        printf("%d\n",i);
}
```

Tipos de Datos Abstractos (TDA)

Un TDA incluye:

- Un tipo de datos.
- Un conjunto de operaciones asociadas al tipo.
- Usualmente provee constructores y destructores para el tipo.
- Generalmente se usan "*Tipos Opacos*", es decir la declaración de la estructura de datos se trasladada a la parte implementación. Esto tiene la ventaja de prohíbe importar y accesar directamente los componentes del tipo.

Un TDA incluye:



☞En *Modula2* los TDA se pueden implementar con facilidad gracias a que provee mecanismos para:

1. Un módulo para el programa Principal.
2. Módulos de definición.
3. Módulos de implementación

☞ Sintaxis del Módulo de definición (.DEF)

DEFINITION MODULE Nombre;

[Lista de importación, puede incluir constante, variables, tipos, procedimientos]

[Declaración de constantes]

[Declaración de tipos]

[Declaración de variables]

[Declaración de procedimientos]

END Nombre.

☞ Sintaxis del Módulo de implementación (.MOD)

IMPLEMENTATION MODULE Nombre;

[Lista de importación, puede incluir constante, variables, tipos, procedimientos]

[Declaración de constantes]

[Declaración de tipos]

[Declaración de variables]

[Declaración de procedimientos]

BEGIN

    [Instrucciones]

END Nombre.

```
DEFINITION MODULE LISTA;

TYPE
  Tipo Lista;                          // Se llama un TIPO OPACO
  TipoElemento = REAL;

PROCEDURE CrearLista(VAR L : TipoLista);
(* pre:- L no existe *)
(* pos:- L existe    *)

PROCEDURE EsVacia(L : TipoLista) : BOOLEAN;
(* pre:- L existe  *)

PROCEDURE Posicion(L : TipoLista; Lugar : CARDINAL) : TipoLista;
(* pre:- L existe *)
(* pos:- Retorna un puntero a la posicion Lugar *)

PROCEDURE InsertarEnLista(VAR  L : TipoLista; X : TipoElemento; Lugar :
                                                    CARDINAL);
(* pre:- L existe y   1 <= Lugar <= LargoLista(L) + 1 *)
(* pos:- incluye el elemento X de tipo TipoElemento en la posicion Lugar*)

PROCEDURE EliminarDeLista(VAR L : TipoLista; Lugar : CARDINAL);
(* pre:- L existe y Lugar(L) >= 1 *)
(* pos:- excluye de la lista el elemento ubicado en la posicion Lugar *)

PROCEDURE RecuperarDeLista(L : TipoLista; Lugar : CARDINAL; VAR X :
                                                    TipoElemento);
(* pre:- L existe y Lugar(L) >= 1 *)
(* pos:- Deja en X el elemento recuperado *)

PROCEDURE VisualizaLista(L : TipoLista);
(* pre:- L existe  *)
(* pos:- permite visualizar por pantalla  el contenido de la lista *)

END Lista.
```

```
IMPLEMENTATION MODULE Lista;

FROM Storage IMPORT Available, ALLOCATE, DEALLOCATE;

FROM IO IMPORT WrCard, WrStr, WrLn, WrReal;

TYPE
  TipoLista = POINTER TO Reg;

  Reg = RECORD
        Elemento : TipoElemento;
        Sig      : TipoLista;
       END;

PROCEDURE CrearLista(VAR L : TipoLista);
BEGIN
  L:=NIL;
END CrearLista;

PROCEDURE EsVacia(L : TipoLista) : BOOLEAN;
BEGIN
   IF L=NIL THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   END;
END EsVacia;


.....etc.

END Lista.
```

```
MODULE PrLista;

FROM Lista IMPORT CrearLista, InsertarEnLista, VisualizaLista,
                  TipoLista, EliminarDeLista, RecuperarDeLista;
FROM IO IMPORT WrStr, WrLn, RdReal, RdCard, WrReal;

VAR
   BaseD : TipoLista;
   Num : REAL;
   Pos : CARDINAL;

PROCEDURE HaceLista(VAR L : TipoLista);
BEGIN
  REPEAT
     WrStr('De El Numero a insertar >-  ');
     Num:=RdReal();
     WrLn;
     WrStr('De El Lugar  >-  ');
     Pos:=RdCard();
     InsertarEnLista(BaseD,Num,Pos);
  UNTIL Num=0.0;
END HaceLista;

BEGIN
    CrearLista(BaseD);
    HaceLista(BaseD);
    VisualizaLista(BaseD);
    WrLn;
    WrStr('De El Lugar que desea eliminar >-  ');
    Pos:=RdCard();
    EliminarDeLista(BaseD,Pos);
    VisualizaLista(BaseD);
    WrLn;
    WrStr('De El Lugar que desea visualizar >-  ');
    Pos:=RdCard();
    RecuperarDeLista(BaseD,Pos,Num);
    WrLn;
    WrReal(Num,6,6);
    WrLn;
    VisualizaLista(BaseD);
END PrLista.
```

☞En *C* los TDA se pueden implementar utilizando:

1. Un archivo *.cpp para el programa Principal.
2. Las definiciones en archivos *.h.
3. Las implementaciones en un archivos *.cpp.
4. Un proyecto que unifique todos los archivos extensión *.cpp.