1. Estructura Básica de un Programa en Pascal

```
program [Nombre del programa];

  (* Declaracion de Constantes y Variables *)

begin

  (* Entrada de datos *)

  {Precondicion}

  (* Calculos *)

  {Postcondicion}

  (* Salida de datos *)

end.
```

2. Declaración de Constantes y Variables

Puede escribirse *const* y *var* para cada constante/variable a definir/declarar:

```
const [nombre] = [valor];
const [nombre] = [valor];
      .
      .

var [nombre] : [tipo];
var [nombre] : [tipo];
      .
      .
```

O también puede hacerse a manera de lista:

```
const
  [nombre] = [valor];
  [nombre] = [valor];
      .
      .

var
  [nombre] : [tipo];
  [nombre] : [tipo];
      .
      .
```

Ambas opciones son equivalentes.

Los tipos de datos básicos son:

Entero	
integer	desde -32768 hasta 32767
byte	desde 0 hasta 255
shortint	desde -128 hasta 127
word	desde 0 hasta 65535
longint	desde -2147483648 hasta 2147483647

Real	
single	7 decimales
double	15 decimales
extended	19 decimales

Caracter	
char	Caracteres de la codificación ASCII.
string	Cadena de hasta 255 caracteres.

Boolean
true
false

3. Precondición y Postcondición

La precondición se coloca luego de recibir los datos de entrada, puesto que se debe verificar que su valor sea el esperado antes de ejecutar alguna instrucción.

La postcondición se coloca antes de retornar los datos de salida al usuario, dado que se debe verificar que las instrucciones hayan sido ejecutadas correctamente y los valores de salida sean los esperados.

```
begin

  (* Entrada de datos *)

  {precondicion}

        .
        .

  {postcondicion}

  (* Salida de datos *)

end.
```

Ambas se escribirán comentadas y se especificarán como se hace en la teoría (GCL), pero utilizando los operadores de GaCeLa:

GCL	Gacela
Operadores Booleanos Sintaxis: ([expresión] [operador] [expresión])	
conjunción	/\
disyunción	\/
negación	!= (también se usa como operador aritmético para 'distinto de')
implicación	==>
consecuencia	<==
equivalencia	==
true	true
false	false
Operadores Matemáticos (Los mismos de GCL) Sintaxis: ([expresión] [operador] [expresión])	
Cuantificadores Sintaxis: (% [cuantificador] : [rango] : [cuerpo])	
para todo	forall
existe	exists
sumatoria	sigma
productoria	pi
contador (#)	count

4. Cálculos e Instrucciones

1. Instrucciones de Lectura

   - *READ*
     Introducción de datos por consola.

     Lee la línea en la cual se introdujo el dato por consola. Si hay más datos que variables se produce un error. De lo contrario, asigna cada dato a cada variable.

     read ([variable])

     Pueden leerse también una o más variables en la misma línea.

     read ([variable1] , [variable2] , ...)

   - *READLN*
     Introducción de datos por consola.

     Le asigna a la variable el primer dato encontrado en la línea de la consola y descarta cualquier otro dato.

     readln ([variable])

2. Instrucciones de Escritura

   - *WRITE*
     Escritura de datos y/ó texto en consola.

     Escribe en consola sin hacer un salto de línea después de la escritura.

```
write('[texto]')

write([variable])

write([constante])
```

También pueden escribirse textos, variables y constantes en una mísma línea.

```
write(['texto1'] , [variable1] , ['texto2'] , [variable2] , ...)
```

WRITE también permite controlar el espaciado y el número de decimales (en caso de números reales) que se imprimirán por consola.

```
write([variable]:[espaciado antes de la variable]:[cantidad de decimales])
```

- *WRITELN*

  Escritura de datos y/ó texto en consola.

  Funciona igual que write, pero hace un salto de línea después de la escritura.

> Ejemplos:

(*i*)
```
write('Hello ');
write('World!')
```

Salida por pantalla:

```
Hello World!
```

Imprime cada texto sin hacer un salto de línea.

(*ii*)
```
writeln('Hello ');
write('World!')
```

Salida por pantalla:

```
Hello
World!
```

Imprime el primer texto y hace un salto de línea para imprimir el segundo.

(*iii*)
```
writeln('Hello ' , Name , '!');
write('From Planet ', PlanetName, '.');
```

Salida por pantalla:

```
Hello Jou!
From Planet Earth.
```

Imprime texto y variable en la misma línea.

(*iv*)
```
const
  h = 2.5467382;
  .
  .
  write(h:0:2)
```

Salida por pantalla:

```
2.54
```

Imprime la variable sin espaciado y con dos decimales.

3. Asignación

   Asignación de valores a las variables.

   ```
   [variable] := [valor];
   ```

4. Punto y Coma

   El punto y coma se utiliza para indicar que termina una instrucción.

   También debe ser colocado después de cada definición de una constante, declaración de una variable e instrucción del programa.

5. Comentarios

   Se seguirá la siguiente convención:

   - Llaves

     Para las precondiciones y postcondiciones.

     ```
     {precondicion}

     {postcondicion}
     ```

   - Paréntesis

     Para comentarios de documentación del código y encabezado del programa.

     ```
     (* Encabezado *)

     (* Documentacion *)
     ```

   - Slash

     Para los comentarios de cola de las variables.

     ```
     // Comentario de cola
     ```

5. Operadores Aritméticos en Pascal

Operador	Descripción	Retorno
=	igual	booleano
<>	distinto	booleano
div	división entera	entero
mod	división entera	entero (Resto de la división)
/	división real	real

6. Palabras Reservadas

Ciertas palabras en Pascal tienen un sólo propósito y por lo tanto no pueden ser usadas en los programas para nombres de constantes, variables, etc. A estas palabras se les llama *Palabras Reservadas*.
Por ejemplo:

PROGRAM, CONST, VAR, INTEGER, BOOLEAN, REAL, CHAR, STRING, BEGIN, END.

Otras palabras reservadas son nombres de funciones u operadores predeterminados en Pascal, como por ejemplo:

READ, READLN, WRITE, WRITELN, DIV, MOD, ABS, SQR, SQRT.

El uso de mayúsculas o minúsculas para las palabras reservadas en Pascal es indiferente, por ejemplo, *PROGRAM*, *Program* o *program* son equivalentes. Esto hace al lenguaje un poco mas flexible, pero debe tomarse en consideración el buen estilo al momento de programar y cuidar que el código se mantenga siempre uniforme y limpio.

7. Documentación del Código y Buen Estilo de Programación

1. Documentación del Código

   Los códigos realizados deben tener al inicio un comentario con cierta información acerca del mismo.

```
(*
    [Nombre del programa]
    [Descripcion]
    Autor: [autor]
    Ultima fecha de modificacion: [fecha]
*)

PROGRAM ...
```

   La parte de documentación también se refiere a los comentarios explicativos acerca del código que se escribe. La documentación del código es obligatoria.

2. Buen Estilo de Programación

   Es muy importante cuidar en el código:

   - Indentación:
     Buen espaciado del código.
   - Nombres mnemónicos para las constantes y variables:
     Nombres que representen claramente los datos que almacenen.
   - Documentación:
     Comentarios explicando el funcionamiento del programa.

Como material de referencia puede consultarse la *"Guía de Estilo"* de GaCeLa.

http://wiki.lal.labf.usb.ve/GacelaWiki/Wiki.jsp?page=GuiaDeEstilo1-4

Material realizado por Michelle Fernández y Carlos Aponte.
Universidad Simón Bolívar, enero 2012.