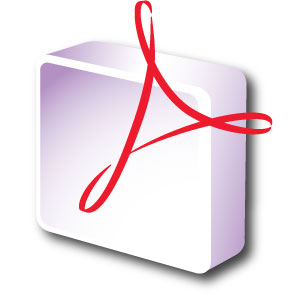
ESTRUCTURA DE DATOS II

PROYECTO VISUALIZADOR DE FICHEROS POSTSCRIPTS

****

**Sinopsis:**

**Documento que trata de explicar y clarificar el código del programa *Visualps*. Este programa es una utilidad para visualizar archivos “simples” en lenguaje postscripts.**

**Autores:**

050007; Ángel Alférez Aroca; [alferez.aroca@gmail.com](mailto:alferez.aroca@gmail.com)

Indice

Introducción.................................................................................Pág.3.

* Algo de historia……………………………………………………………….Pág.3.

Descripción……………………………………………………………………………….Pág.4.

* Estructura modular………………………………………………………….Pág.4.
* Funcionamiento general………………………………………………….Pág.5.
* Bloque principal……………………………………………………………….Pág.6.
* Bloque errores…………………………………………………………………Pág.10.
* Bloque gráfico………………………………………………………………….Pág.10.
* Bloque preparar operador……………………………………………….Pág. 12.

Pruebas………………………………………………………………………………………Pág.13.

Ampliación y mejoras…………………………………………………………………Pág.18.

Curiosidades………………………………………………………………………………Pág.19.

Conclusiones………………………………………………………………………………Pág.19.

Anexo:

Manual de usuario………………………………………………………………………Pág.21.

Demostración……………………………………………………………………………..Pág.25.

Introducción.

Un documento PostScript, en realidad, es un pequeño programa que le dice a una máquina qué, cómo y dónde imprimir, paso a paso. Los documentos PostScript se destinan a imprimirse en aparatos PostScript, es decir, en aparatos que tienen un dispositivo interno capaz de descifrar el código que reciben y convertirlo en simples puntos de impresión (aquí imprimo, aquí no, aquí sí, aquí también, etc…).

Para ser adaptable, PostScript es un lenguaje de los denominados "interpretados". Es decir, no le habla directamente a la máquina, sino que necesita un procesador (un dispositivo físico o un programa residente en el ordenador) que actúe como intérprete traductor entre el código PostScript universal y la máquina. Eso es lo que se llama "intérprete PostScript".

De hecho, el lenguaje PostScript es tan "universal" que su forma más sencilla son simples instrucciones escritas como textos (siguiendo, eso sí, una sintaxis muy rigurosa). Básicamente un fichero PostScript contiene instrucciones que, traducidas al lenguaje humano, dicen cosas del tipo: traza un círculo de 3 cm de radio, dibuja una línea de 4,5 cm. en un ángulo de 45 grados, etc…

Por lo que nuestro programa es un "simple intérprete PostScript ", debe traducir estás instrucciones hasta llegar a la impresión en pantalla de la mismas.

* Algo de historia.

PostScript fue inventado y desarrollado por la firma californiana Adobe hacia 1984. Apple lo adaptó en 1985 a sus impresoras láser y fue un rotundo éxito, gracias en parte a una aplicación de diseño de páginas llamada PageMaker que fabricaba la compañía Aldus.

El hecho es que PostScript no es el único Lenguaje de Descripción de Página (LDP) que existe, pero es el que ha tenido más éxito y el que se ha convertido en un estándar de las artes gráficas.

Descripción.

* Estructura modular.

**BLOQUE\_PREPARAR\_OPERADOR**

TAD ITERADORES (ED1)

LIBREIRÍA JEWL

TAD PILA (ESTADOS GRÁFICOS)

**BLOQUE\_ERRORES**

TAD TABLA

**BLOQUE\_GRAFICO**

**BLOQUE\_PRINCIPAL**

**VISUALPS**

**PROGRAMA CARGADOR**

Ada.Unchecked\_Deallocation

Ada.Command\_Line

TAD PILAS (Pila para postscrip)

ANDRES

ANDRES

ANGEL, ROBERTO

ROBERTO

* Funcionamiento general.

Un lenguaje de programación basado en pila es aquél que se basa en el modelo de una máquina de pila para pasar parámetros. De este tipo es el lenguaje PostScript.

Un lenguaje orientado a pila se trabaja mediante operaciones sobre una o varias pilas, que pueden servir para varios propósitos, en nuestro caso tendremos la pila de operaciones y la pila gráfica. Además por ser un lenguaje a pila operan en notación polaca inversa o postfija (RPN) para simplificar su procesamiento; es decir, los argumentos o parámetros de algunas órdenes se indican antes de la propia orden.

Ejemplo de nuestro programa:

Código

xxx

Def2

Def2

Def1

add

Def1

Tabla

32

45

Def1

Pila *Figura. Pila PostScript*

El código va pasando por la pila respetando el orden de la notación polaca inversa, al llegar a una definición se busca en la tabla si está el código de esta. Se van procesando todos los operandos y realizando con cada uno la acción pertinente. Como describimos a continuación en el bloque principal de nuestro programa.

* Bloque principal.

Este módulo nos permitirá leer el fichero PostScript e ir procesando los distintos elementos contenidos en él, evaluándolos y realizando las operaciones que correspondan.

A la hora de procesar el fichero PostScript necesitaremos una pila que emplearemos para almacenar en ella distintos elementos (del tipo descrita en el apartado anterior) Elemento\_Pila es un registro variante y según qué tipo, tendremos un campo distinto (operan dos, código, booleanos, palabras, ninguno).

Ahora procederemos a comentar los distintos procedimientos que emplearemos en este módulo:

El procedimiento **Coger**, coge del fichero el siguiente elemento significativo del archivo PostScript. Omitiendo los comentarios.

El procedimiento **Coger\_txt**, lo emplearemos para extraer un texto que queramos mostrar; esto lo detectaremos cuando nos encontremos paréntesis “( )”, que corresponde al parámetro que acompaña a Show (operador empleado en PostScript para este fin).

El procedimiento **Cerrar**, cierra el frame iniciado por el Bloque\_Gráfico.

El procedimiento **Ocultar**, oculta el frame iniciado por el Bloque\_Gráfico.

El procedimiento **Procesar** toma Data ([String, Natural]) si no es ni un operador de control, ni una definición y toma los argumentos necesarios y los mete en pila, examinando el posible error que se haya podido generar.

Utilizaremos la siguiente codificación para representar dichos errores:

* 0 = No ha habido errores (salida exitosa)
* 1 = Se ha encontrado un dato invalido (no corresponde al lenguaje PostScript)
* 2 = Se a producido un acceso a pila vacía
* 3 = Se han encontrado operandos de un tipo inesperado
* 4 = Se intenta trazar un camino nuevo sin establecer un origen
* 5 = Intento de acceso a pila gráfica estando vacía

El proceso será el siguiente: Count nos mostrará en todo momento el número de elementos que tiene la pila.

Si DATA es un número, es decir Operando\_pila, lo apilamos.

Si el texto contenido en DATA es "newpath" saltamos a la función newpath.

Todas las operaciones si producen algún tipo de error paran el proceso de evaluación del fichero PostScript y en caso contrario realizan la operación pedida.

Todas las operaciones aritméticas tomarán los argumentos de tipo Operando\_Pila, realizan la operación deseada y apilan el resultado en pila de tipo [Operando\_Pila, Resultado].

Todas las operaciones de comparación tomarán los argumentos, realizarán la operación de comparación deseada y apilan el resultado en pila de tipo [Bool\_Pila, TRUE/FALSE].

Resumen de operaciones:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OPERACIÓN** | **NÚMERO OPERANDOS** | **TIPO DE ARGUMENTOS** | **ERROR DE OPERACIÓN** | **APILA RESULTADO** |
| MOVETO | 2 | Operando\_Pila | - | NO |
| RMOVETO | 2 | Operando\_Pila | 4 | NO |
| LINETO | 2 | Operando\_Pila | 4 | NO |
| RLINETO | 2 | Operando\_Pila | 4 | NO |
| CLOSEPATH | 0 | - | - | NO |
| STROKE | 0 | - | - | NO |
| ROTATE | 1 | Operando\_Pila | - | NO |
| SCALE | 2 | Operando\_Pila | - | NO |
| SHOW | 1 | Palabra\_Pila | - | NO |
| SETLINEWIDTH | 1 | Operando\_Pila | - | NO |
| SETGRAY | 1 | Operando\_Pila | - | NO |
| TRANSLATE | 2 | Operando\_Pila | - | NO |
| GSAVE | 0 | - | - | NO |
| GRESTORE | 0 | - | 5 | NO |
| DEF | 2 | E1: Codigo  E2: Palabra\_Pila | - | NO |
| SHOWPAGE | 0 | - | - | NO |
| ADD | 2 | Operando\_Pila | - | SI |
| SUB | 2 | Operando\_Pila | - | SI |
| MUL | 2 | Operando\_Pila | - | SI |
| DIV | 2 | Operando\_Pila | - | SI |
| IDIV | 2 | Operando\_Pila | - | SI |
| MOD | 2 | Operando\_Pila | - | SI |
| CLEAR | 0 | - |  | NO |
| DUP | 1 | cualquiera | - | SI |
| EXCH | 2 | cualquiera | - | --- |
| POP | 1 | cualquiera | - | NO |
| ROLL | 2 | Operando\_Pila | 2 | --- |
| EQ | 2 | cualquiera | - | SI |
| NE | 2 | cualquiera | - | SI |
| GT | 2 | Operando\_Pila | - | SI |
| GE | 2 | Operando\_Pila | - | SI |
| LT | 2 | Operando\_Pila | - | SI |
| LE | 2 | Operando\_Pila | - | SI |

*Figura. Tabla resumen de operaciones PostScrips.*

En el caso de que la pila no contenga los elementos necesarios para realizar la operación pedida, se producirá un error de tipo 2 (acceso a pila vacía Bloque\_errores).

En el caso de que los argumentos extraídos de la pila, no se correspondan con los tipos requeridos por la operación a realizar, se producirá un error de tipo 3.

Estas dos operaciones las realizaremos siempre en todas las acciones que requieran argumentos para llevarse a cabo.

El procedimiento **Leer\_Iterador**, procesará el fragmento de código almacenado en el iterador.

En el caso de que se trate de un bucle stop pasará a ser “TRUE”, cuando encontremos la sentencia “exit”.Este procedimiento contendrá todas las operaciones necesarias para realizar operaciones con los iteradores:

Si encontramos un operando en el iterador, lo almacenamos en la pila.

En caso de que se tratase de una definición, emplearemos un iterador auxiliar (AUXI) para asignarle el iterador de la definición y llamamos a Leer\_iterador para procesar el iterador AUXI.

Si Elem es de tipo "text" lo apilamos como [palabra\_pila, String].

Si no es ninguna de las anteriores se tratará de un operador; en el caso de que nos encontremos una llave “{”, asignaremos un nuevo iterador (AUXI) con todo lo contenido en dichas llaves, llevando claro está, un control de balanceo de llaves.

Haremos una copia del iterador Auxi que contiene todo lo que se encuentra entre llaves en "puntero", liberaremos Auxi y apilamos [código, puntero] y una vez apilado el código, podremos “puntero” a NULL.

Si encontramos "loop", tendremos que asignar un nuevo iterador (AUXI) con todo lo que contiene el bucle, para ello simplemente asignamos a AUXI el siguiente elemento almacenado en la pila, llamando a Leer\_iterador con AUXI en un bucle hasta encontrar que “stop” pasa a TRUE.

Si encontramos un “repeat” cogemos el primer y segundo elemento de la pila, siendo el primero el número de repeticiones, y el segundo un puntero que señale al código que debemos repetir (se lo asignaremos a AUXI), llamando tantas veces como se solicite a Leer\_iterador, liberando memoria y desapilando dos veces.

En el caso del “IF” cogemos el primer y segundo elemento de la pila (un puntero que señale al código que debemos procesar y la condición booleana) y en caso de que se cumpla, procesaremos el código señalado por AUXI llamando a Leer\_iterador. Cuando hayamos terminado, liberamos memoria y des apilamos dos veces. El caso de “ELSIF” es similar, con la salvedad de que tendremos dos posibles fragmentos de código para poder ejecutar.

Si finalmente no es ninguna de las posibilidades anteriores, no se corresponderá con un fragmento de código y procesamos la operación tratada.

Si hay más elementos en la tabla de iteradores avanzamos al siguiente elemento. Y si por el contrario no hay más elementos en la tabla de iteradores, hemos terminado.

Si se produjo algún error no continuamos y salimos de la evaluación.

El procedimiento **Leer\_Postscript**

Ejecuta el programa PostScript y se reproduce en pantalla. En caso de no ser un archivo PostScript o de haber errores cierra el archivo. En este procedimiento empezaremos iniciando la tabla, creando el iterador, creando la pila y abriendo el archivo PostScript.

Lo primero que compraba eremos será la cabecera del archivo PostScript, de forma que deberá coincidir con "%!PS".

A continuación comenzamos a procesar las operaciones descritas en el archivo PostScript. Para ello, emplearemos dos estructuras básicas de datos: una tabla de iteradores y una pila.

La tabla de iteradores contendrá los punteros que apuntan a los fragmentos de código correspondientes a las definiciones propias siendo su clave el nombre de la definición (/NOMBRE), mientras que en la pila almacenaremos los operan dos de las operaciones a realizar.

Una vez procesado todo el fichero PostScript, borramos la tabla, cerramos el archivo y destruimos la pila.

* Bloque errores.

Se trata de un simple paquete que aglutina todas las posibles salidas de errores.

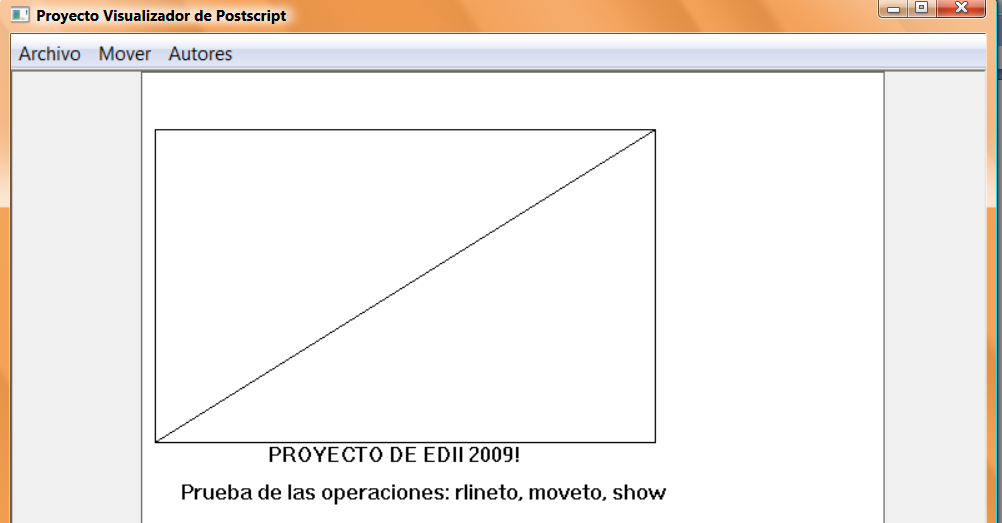
|  |
| --- |
| **Error1:** "Fallo del Sistema: contenido del archivo PostScript no válido" |
| **Error2:** "Fallo del Sistema: Intento de acceso a Pila\_Vacia. Faltan argumentos en la pila" |
| **Error3:** "Fallo del Sistema: en la pila se ha encontrado un tipo no esperado" |
| **Error4:** "Fallo del Sistema: se ha intentado trazar un camino nuevo sin origen definido" |
| **Error5:** "Fallo del Sistema: no es un archivo PostScript" |
| **Error6:** "Fallo del Sistema: debe especificar que archivo desea abrir" |
| **Error7:** "Faltan argumentos en la pila de estados gráficos" |

* Bloque gráfico. Y manejo del interfaz.

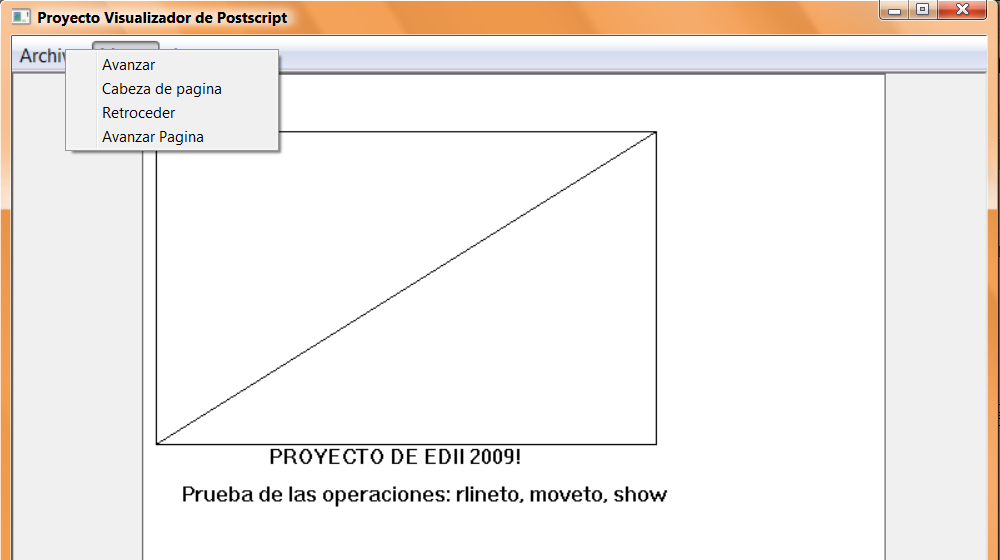
Este bloque es el encargado de la parte gráfica del proyecto. Crea el interfaz de usuario, menú y salida de errores y se encarga de pintar en pantalla él código PostScript.

La librería utilizada ha sido JEWL, un tanto limitada pero fácil de trabajar con ella.

Menú:



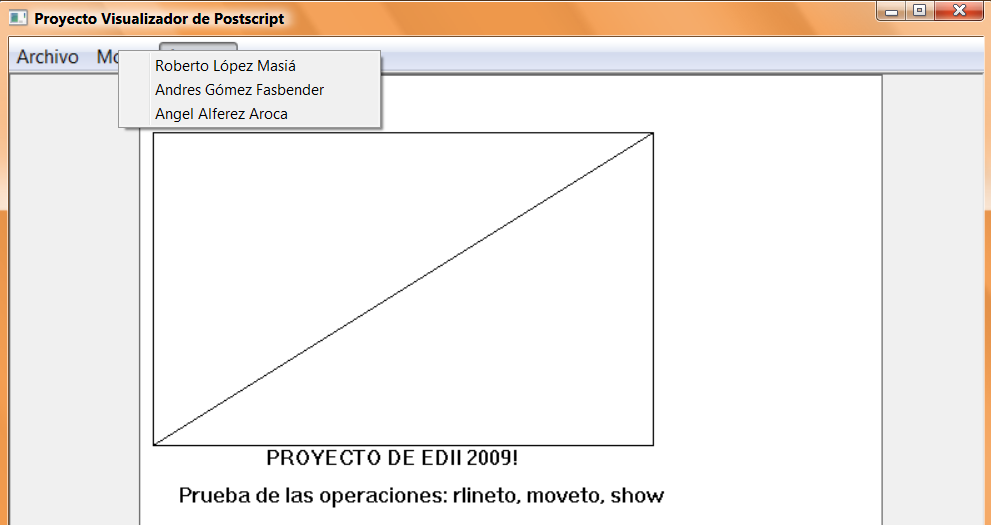
*Figura. Menu.Principal*



*Figura. Sub-Menú Mover*

|  |
| --- |
| *Mover* |
|  | *Avanzar* | *Avance preciso de la página* |
|  | *Cabeza de página* | *Situarnos en el encabezado de la página.* |
|  | *Retroceder* | *Avance preciso de la página* |
| *Avanzar página* | *Pasamos a la siguiente página* |

*Figura. Sub-Menú Mover Utilidad*



*Figura. Sub-Menú Autores*

Tipo:

Estado

Origen,

Actual : Coordenadas := (0.0, 0.0);

Escala : Matrizm := (1.0, 0.0, 0.0, 1.0);

Angulo : Float := 0.0;

Grosor : Integer := 1;

Color : Float := 0.0;

Estado

Estado

Estado

Estado

Figura.Pila de estados gráficos

La pila auxiliar de estados gráficos nos permite saber en cada momento el estado del dibujo, poder salvar éste o movernos entre estados.

* Bloque \_preparar \_operador.

Esta parte contiene las operaciones auxiliares que va utilizar el programa principal y la construcción de algunos tipos.

Operaciones:

1. para procesar el código PostScript. Simples operaciones con String, pasar a float (Es\_Numero, Es\_Def, Valor…)

2. almacenar en los iteradores el código.

3. sobre la tabla de definiciones (Inicia\_Tabla, Borrar\_Tabla, Guardar, Consultar)

Tipos:

A continuación mostramos la tabla que almacena las nuevas definiciones (porciones de código reutilizable), la tabla tiene como *clave* el nombre de la definición y la *información* es el puntero al iterador, es decir el fragmento de código guardado.

TABLA

ITERADOR

OPERANDO

OPERADOR

DEFINICIÓN

TEXTO

INFO

CLAVE

Posibles ELEMENTOS

E

INFO

CLAVE

INFO

CLAVE

Figura. Tipos Bloque\_preparar\_operador

Pruebas.

* Tratamiento de errores

En este apartado nos centraremos en la detección de errores:

**Error de tipo 1:**

Se produce cuando se ha encontrado un dato inválido no correspondiente al lenguaje PostScript.

Un ejemplo de este error se obtendría mediante el siguiente código:

%!PS%

newpath

400 400 moveto

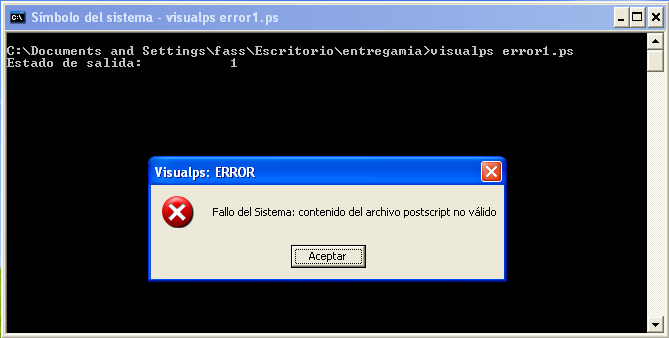
200 0 rlineto

0 200 rlineto

%En este caso “Cerrar\_camino” no pertenece al lenguaje PostScript

Cerrar\_camino

Showpage



Como podemos ver, nos sale un mensaje de error al procesar este código, acompañado del numero de error interno que se ha producido (en este caso el estado de salda 1).

**Error de tipo 2:**

Se produce cuando se realiza un acceso a pila vacía.

Un ejemplo de este error se obtendría mediante el siguiente código:

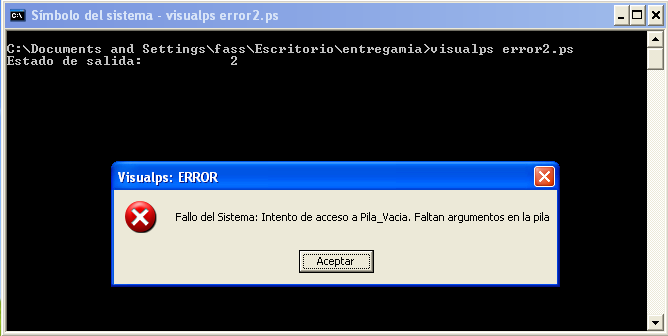
%!PS%

newpath

400 400 moveto

0 200 clear rlineto

Showpage



**Error de tipo 3:**

Se produce cuando se encuentra un operando inesperado.

Un ejemplo de este error se obtendría mediante el siguiente código:

%!PS%

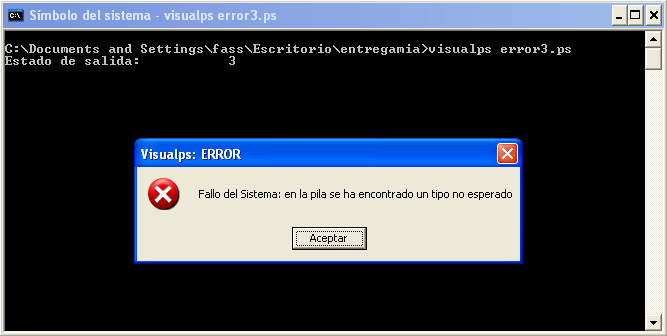
newpath

400 400 moveto

%Pasamos dos argumentos a rlineto pero no se corresponde con el esperado

(esto debería) (ser una coordenada) rlineto

Showpage



**Error de tipo 4:**

Se produce cuando se intenta trazar un camino nuevo sin establecer un origen.

Un ejemplo de este error se obtendría mediante el siguiente código:

%!PS%

Newpath

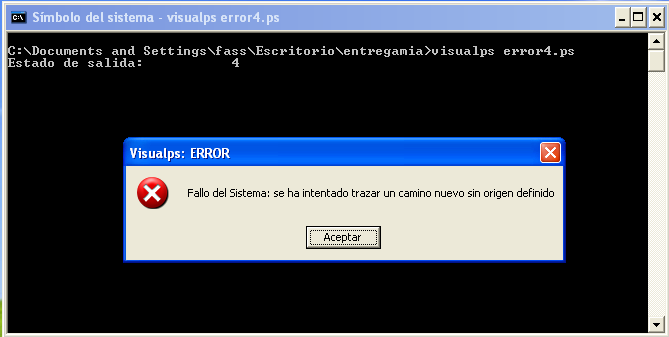
%No hemos definido un origen y ya estamos definiendo un camino

0 200 rlineto

closepath

stroke

showpage



**Error de tipo 5:**

Se produce cuando se intenta acceder a la pila gráfica estando vacía.

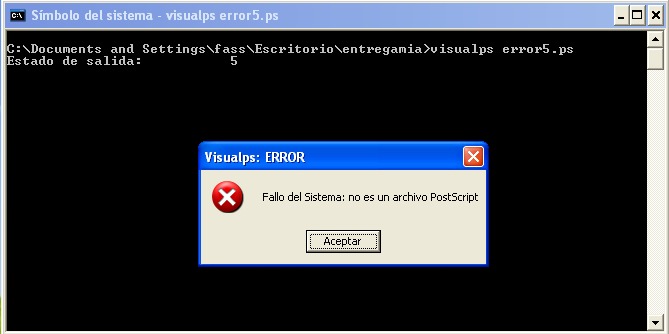
Un ejemplo de este error se obtendría mediante el siguiente código:

%! PS%

2 3 add

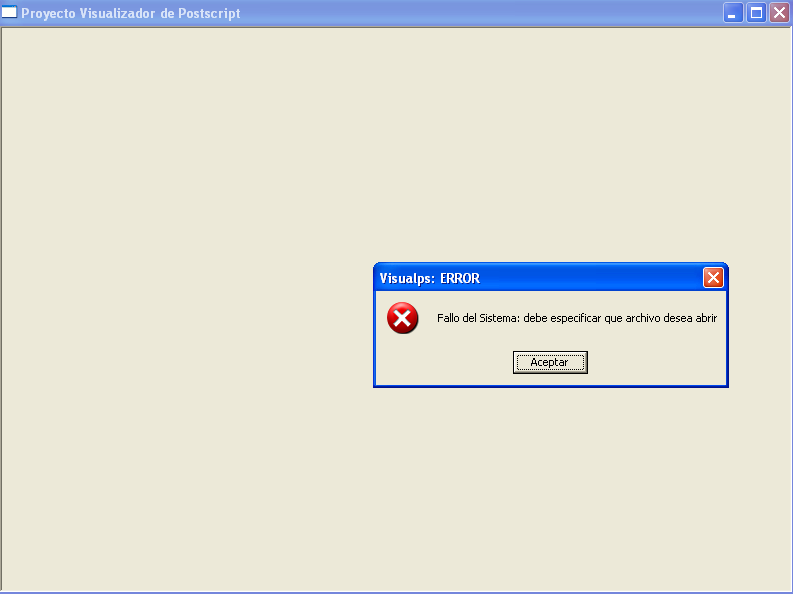
%En este momento estamos intentando acceder a la pila gráfica estando vacía

Grestore



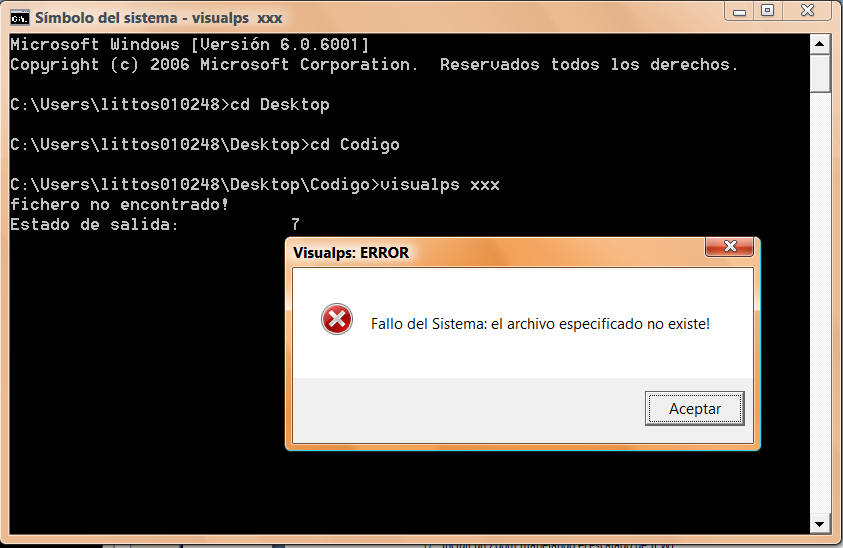
**Erro de tipo 6:**

Se produce cuando se ejecuta el programa sin definir el fichero PostScript que se quiere abrir.

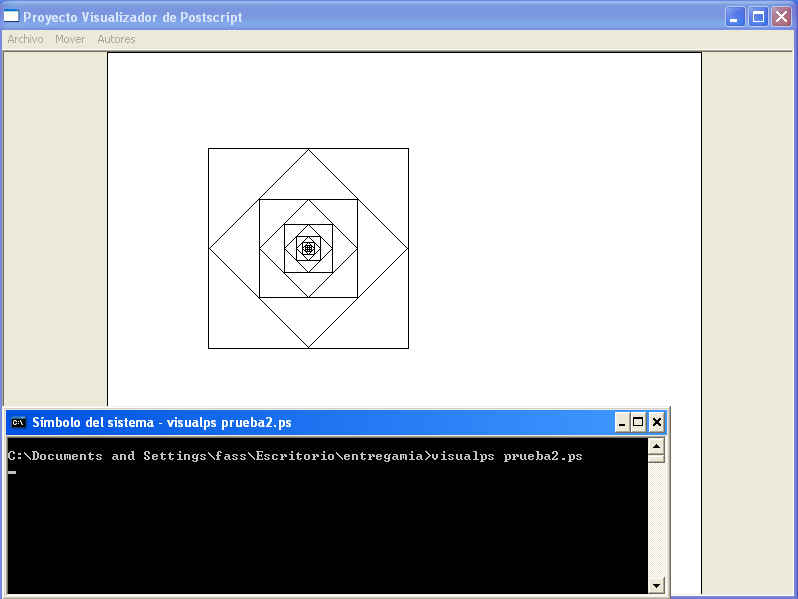


**Erro de tipo 7:**

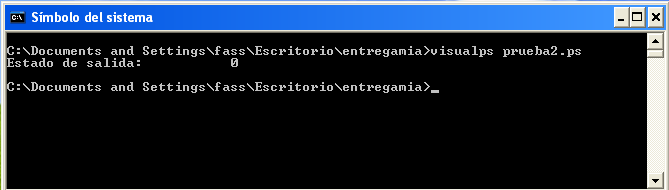
Se produce cuando se ejecuta el programa con un fichero PostScript que no existe.



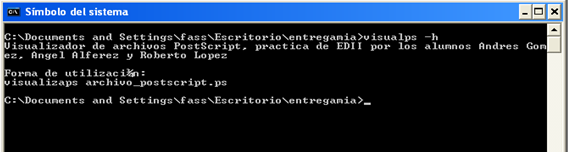
En el caso de que todo vaya bien el resultado sería el siguiente:



Finalmente si cerramos el programa y todo ha salido y no se ha detectado ningún error obtendríamos lo siguiente (Estado de salida 0):



Asimismo, podremos consultar el dialogo de ayuda del programa mediante “-h”.



Ampliación y mejoras.

Este proyecto es una versión un tanto limitada de un intérprete PostScript, en una segunda versión del proyecto se podría ampliar la funcionalidad de este con mejoras tales:

* Manejo de más sintaxis PostScript.
* Realizar un buscador de palabra.
* Mejora del interfaz gráfico.
  + Abrir el fichero de forma gráfica.
  + Incluir un Zoom manejando el escalado de JEWL.
  + Manejar giros de página.

Muchas de estas funcionalidades aparentemente sencillas de programas, se complica al manejar tantas líneas de código, al añadir más funcionalidades deberíamos seguramente hacer más hincapié en la correcta estructura modular.

Curiosidades.

Hace unos cuantos años salieron a la venta unos libros con el nombre de “ojo mágico” se trataba de ilustraciones que alejándolas del ojo humano se conseguía un efecto tridimensional de una figura. Se llaman en realidad estereograma, son imágenes que consiguen ese efecto tridimensional engañando en cierta forma de colocar puntos gemelos a ciertas distancias. Estás imágenes se realizan en lenguaje PostScript

*http://www.sabercurioso.com/2008/10/18/como-funcionan-estereogramas/*

Conclusiones.

La práctica nos ha algún problema más de lo esperado. En un proyecto de un tamaño ya considerable la organización y buen diseño preliminar es fundamental para la correcta realización de éste. Algo que suena tan “típico” y repetido creo que ha sido nuestra lección más importante en esta práctica.

Nuestro grupo formado para la realización de esta empresa (no nos conocíamos antes) Ha tardado en funcionar de manera correcta. La mala comunicación entre los mienbros y alguna circunstancia personal (un miembro del grupo tuvo un accidente grave de tráfico) nos ha hecho trabajar en paralelo y gastar mucho tiempo en encajar trabajos realizados por separado.

Los problemas sobre el código han sido sobre todo el encajar los distintos paquetes y encontrar información sobre la librería JEWL, las primeras tomas de contacto son difíciles al no tener el tan habitual interfaz de dibujar botones, pestañas... ya tan habitual. Luego solo es limitarse a buscar los comandos que usamos para el frame, los menús, punteros…

Es de agradecer en esta práctica el ver el resultado plasmado de la misma en forma de herramienta más o menos eficiente, que incluso los amigos pueden llegar a ver su finalidad (aunque digan solo hace eso) Es un poco frustrante dedicar tantas horas a práctica que solo la gente especializada puede entender y valorar su trabajo. Enseñar una memoria de control microprogramada a tu padre puede ser algo cómico.

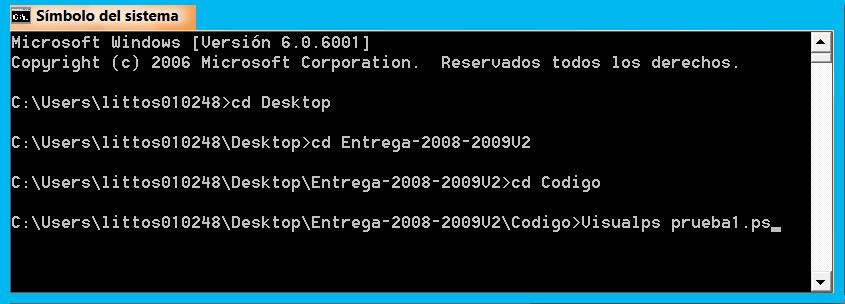
Anexo.

Manual de usuario Visualps.

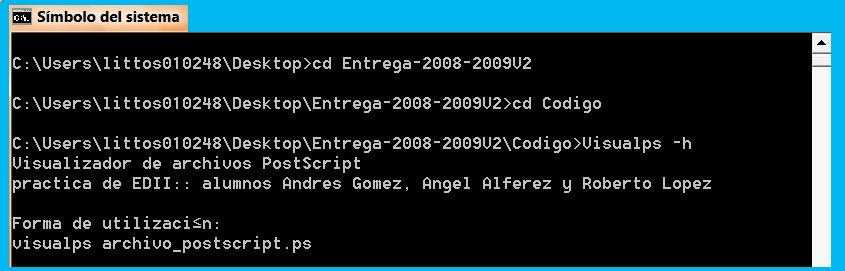
Este manual explica cómo usar el programa Visualps.

PASO11:

INSERTAMOR EN UNA CONSOLA LA RUTA Y EL COMANDO visualps fichero.ps



INSERTAMOS visualps –h PARA CONSULTAR LA AYUDA.



Podemos ver que la ayuda nos dice que para usarlo hay que poner el nombre del programa, espacio y el nombre del archivo ps completo (p.e. archivo\_postscript.ps)

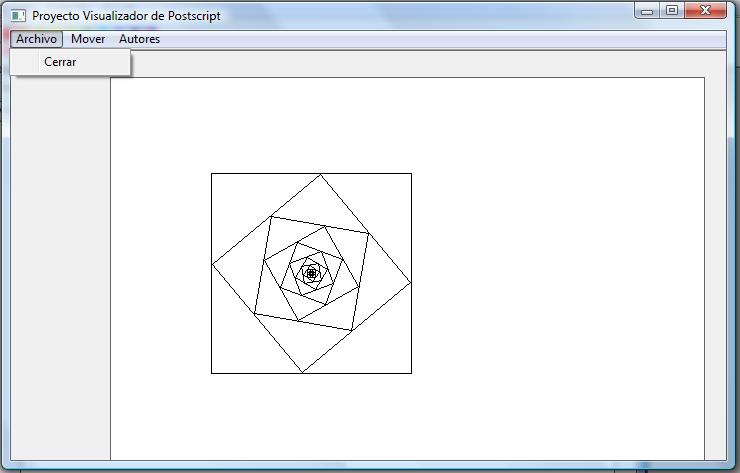
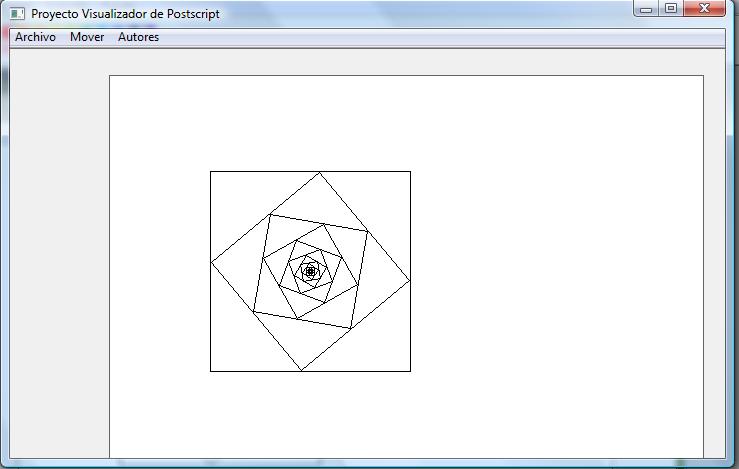
Además, la ayuda da información sobre el programa (que tipo es: Visualizador de archivos Postscript) y que se trata de un proyecto de Estructura de Datos II realizado por esos alumnos.

PASO21:

MANEJO DEL MENÚ PRINCIPAL

**CERRAR**

Como podemos ver la interfaz nos presenta varias pestañas (Archivo, Mover y Autores)



**Click**

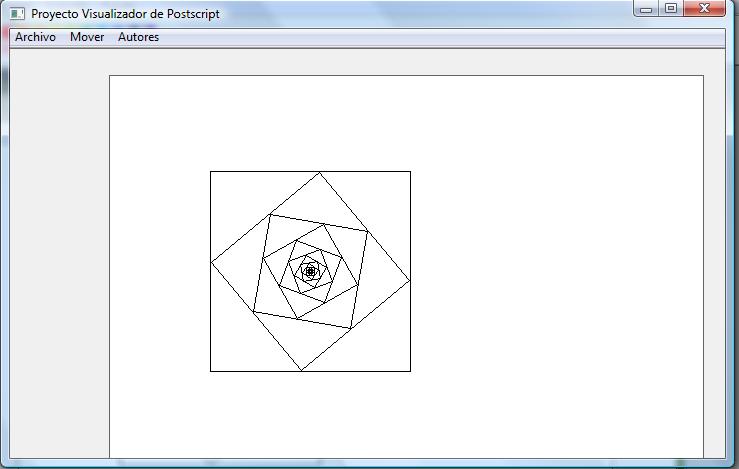
Cerrar nos devuelve a la consola.

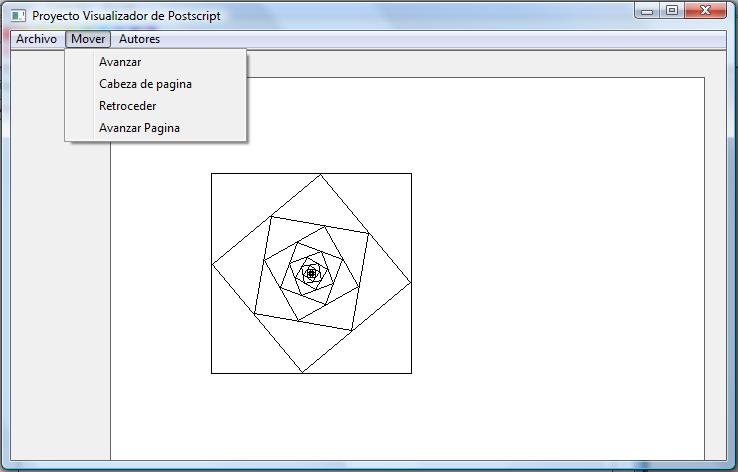
**Click**



***MOVER***

Si pulsamos Mover veremos que se nos despliega otro menú con opciones (Avanzar, Cabeza de página, Retroceder y Avanzar Página

******



**Retroceso preciso de página**

**Cambiar de página**

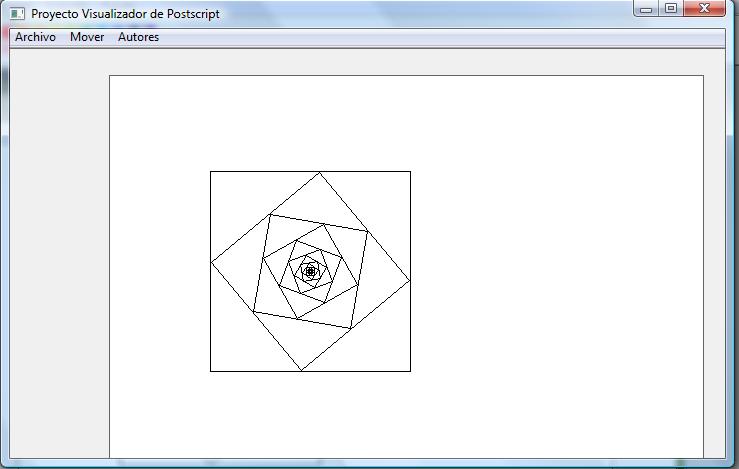
**Situar vista a comienzo de página**

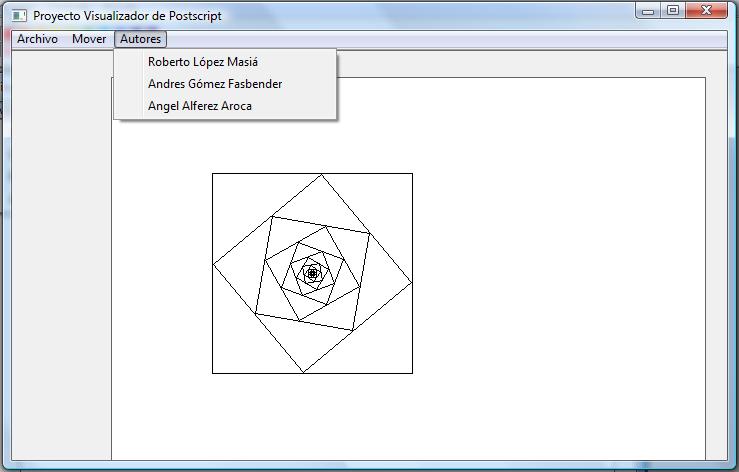
D

**Avance preciso de página**

***AUTORES***

Autores es una pestaña dedicada a los autores y creadores de este proyecto.Al desplegar el menú Autores podemos ver sus nombres:





Demostración.

|  |
| --- |
| **Prueba0**  Prueba de error sin meter ningún archivo |
| //visualps |

|  |
| --- |
| **Prueba1**  Prueba de error introduciendo un archivo que no existe |
| //visualps xxx |

|  |
| --- |
| **Prueba2**  Prueba general del cuadrado\_centrado.  Probar:  Movimiento de la página con el ratón.  Movimiento de precisión.  Movimiento de cabeza de página  Autores.  Cerrar. |
| //visualps cuadrado\_centrado.ps |

|  |
| --- |
| **Prueba3**  Mostrar texto |
| //visualps prueba03.ps |

|  |
| --- |
| **Prueba4**  Otra figura distinta |
| //visualps prueba11.ps |

|  |
| --- |
| **Prueba5**  Prueba de la ayuda |
| //visualps -h |

|  |
| --- |
| **Prueba6**  Se produce cuando se ha encontrado un dato inválido no correspondiente al lenguaje PostScript. |
| //visualps Error1.ps |

|  |
| --- |
| **Prueba7**  Se produce cuando se realiza un acceso a pila vacía. |
| //visualps Error2.ps |

|  |
| --- |
| **Prueba8**  Se produce cuando se intenta trazar un camino nuevo sin establecer un origen. |
| //visualps prueba32.ps |

|  |
| --- |
| **Prueba9**  Prueba de fichero PostScript de varias páginas. |
| //visualps |