Google Przetłumaczono na język: angielski Pokaż oryginał Opcje ▼

angielski

Technologia Google Tłumacz

University of Silesia

Department of Technology

Institute of Applied Informatics

MASTER'S THESIS

Topic:

Printer and plotter emulation in graphic mode

Performed by: Supervisor:
Ryszard Czekaj, prof. dr hab. in . Dariusz Badura
Pawe Szymik Consultant:
mgr Ireneusz Gocinak

PINE 1996

Intro

Modern printers, in addition to printing simple texts, allow the use of various scalable fonts and even placing graphic images on the page. Plotters, like printers, are controlled by special sequences, which in many cases can be treated as languages controlling printing. The stream of control codes, together with the data to be printed, can be sent to the printer or saved in a file on the computer's disk. Such a file, containing data for the printer or plotter, is called a print file. This file can later be sent to a device connected to another computer station. The print file contains control sequences, specific only to a specific type of device, by which it will later be correctly interpreted.

The aim of this work is to develop a computer program that emulates the operation of a selected printer and plotter, which will allow for the graphical presentation of the print file on a computer screen.

In achieving this goal, the following assumptions were made:

- there is a print file of selected printers and plotters in the appropriate format for them
- the printer or plotter model for which the print file was created is known
- the print file can contain simple text data and a graphic image
- the developed program will present the image contained in the file as faithfully as possible
- the program will enable printing data on a printer other than the one for which the print file was prepared
- it is recommended to use the Windows 3.xx graphical environment and the Borland C++ programming tools package to build the program .

Technical aspects of printer and plotter operation

1. Classification of printers

1.1. Dot Matrix Printers (Impact Printers)

In a dot matrix printer, the image is created by a head equipped with a series of pins (usually 9, 24, or 48) that strike a ribbon saturated with ink as the head is moved back and forth in front of the paper. The pins are arranged in one or more vertical rows.

Another, lesser known technology, called Comb Matrix or Shuttle Matrix uses a horizontal row of pins (ideally one pin for each horizontal character), because a single row of dots can better reproduce a character. In cases where there are not as many pins as there are horizontal characters, the comb moves slightly forward and backward to print the necessary dots. With this technique, the paper can move almost continuously and is therefore often used in high-speed dot matrix printers. This technology is much more efficient than printing individual symbols.

Impact dot matrix printers are versatile, i.e. the pin matrix can be adjusted to print different fonts and even graphics.

Dot matrix printers can be used to print copies along with the original because the images are created by striking the ink ribbon. These printers use continuous paper, although most models will accept single sheets of paper and labels.

Due to the printing method, dot matrix printers are significantly louder than other models (even in the so-called quiet mode).

Printing speed, usually given in cps (characters per second), varies depending on the fonts used (Draft/LQ) and different cpi (dots per inch) resolution values [16], [15], [8].

1.2. Daisywheel Printers and Typewriters

In daisywheel printers, the print is created using a drum on which all the letters and symbols are embossed. The head moves back and forth across the paper, and an electromagnet starts working when the drum is set in the position of the required letter.

Daisywheel printers were popular because their print quality was better than that of earlier models, such as the nine-pixel dot matrix printers.

Their main disadvantages are: slow printing speed, limited number of symbols on the drum, and the lack of the ability to create graphics.

Daisywheel printers are impact printers. This allows you to print copies simultaneously with the original. Printing is possible on continuous paper, as well as on single sheets and labels.

Print speed is usually given in cps (characters per second) [8].

1.3. Inkjet printers

Inkjet printers work similarly to dot matrix printers, which create a print by arranging dots on paper. Inkjet printers, ink drops are sprayed directly onto the paper. A much denser pattern of dots than in dot matrix printers allows for a better quality print.

In newer inkjet models, the ink is not ejected electrostatically or by tiny pumps, but is converted into vapor by heating and, under the pressure of the expanding steam, is deposited on the paper.

Inkjet models are not impact printers, so copies are printed separately. Printing is possible on cut paper, as in the case of a photocopier. Some models require special quality paper for better printing, although for most printers standard paper is sufficient.

Some printers can use continuous paper and can even print labels, provided they are made from paper that absorbs the ink well.

lynx.AND HP DeskJet 600C

Inkjet printers are quieter and much faster than impact printers. The print speed is given in cps (characters per second), sometimes also, for various fonts, in ppm (pages per minute).

Although the inks of these printers are water-soluble, they differ greatly in their ability to bleed. This can be important if the print is to be used for presentations in a high-humidity environment.

Some types of ink are not water-resistant. These inks include all Canon inks , DEC 520ic black, and TI Micromarc inks . HP DeskJet black ink (Fig. 1) is almost as water-resistant as a laser print. Epson black and color inks seem to be the most water-resistant : the letters disperse but do not smudge when exposed to moisture.

Color (inkjet) prints unfortunately do not last as long as photographs. Those that are lightfast are often damaged by moisture, while others fade quickly in bright light. The durability of the print can be extended by laminating. Unfortunately, modern color inkjet printers have not been in use long enough to test the longevity of the latest inks [8], [10], [11].

1.4. Laser Printers and LED Printers

Laser and LED models are not impact printers, so copies must be printed separately. They usually use single sheets of paper. They can print labels that can withstand the heat of toner burning.

To produce a print in a laser printer, a special drum is electrostatically charged, and the page image is created by discharging the drum with a laser beam or LED (Light Emitting Diode) line at the appropriate points. Toner adheres to the discharged dots. These dots are mirror images of the places where the media is to appear on the final print.

fig. B HP LaserJet 4V

The paper passes a Corona wire, which gives it an electrical charge opposite to the charge of the points on the drum, then passes over a cylinder, from which the toner is retained on its surface. Finally, it is passed under a heating element, which, fusing the toner, fuses it with the paper.

Most current laser printers do not have a Corona wire, and the paper is electrified by a roller. Laser and LED printers produce very clear, precise prints, with a resolution of over 300 dpi (even 1200 dpi). Some printers use various techniques, such as Resolution Enhancement technology to control the dot spacing for smoother contours.

Laser and LED printers are relatively quiet. Laser and LED models' print output is measured in ppm (pages per minute) in copy mode (i.e., the page image, which is already in the printer's memory, is copied onto the paper as quickly as possible) [8], [10], [11].

1.5. Color Printers

Color printers use several methods to produce a color printout.

Regardless of the method used to produce a color print, there are three different ways to specify colors:

RGB (red-green-blue): standard method, used in cooperation with devices such as monitors and televisions. It is an additive process of mixing colors (white is obtained by mixing all three colors to the same intensity).

CMYK (cyan-magenta-yellow-black): A method used especially in commercial printing. It is a subtractive process of mixing colors (black is obtained by mixing all three colors with the same intensity, or by having an additional black color, which gives a richer, darker black).

same intensity, or by having an additional black color, which gives a richer, darker black).

Dithering: A method in which the colors are not actually mixed, but by placing dots of different colors in different shaded areas, an image is made to appear to use more than four colors. This method is similar to the method of producing a grayscale by setting more or fewer black dots.

Color dot matrix printers use a color ribbon. Inkjet printers have color ink cartridges. Some create black by mixing colors (HP DeskJet 600 C Fig. 1), others have a separate black cartridge (HP DeskJet 660 C).

Thermal printers use a film with four colors of wax. The printer moves the paper four times, once for each color. As each color is transferred to the paper, a full color image is created.

Some manufacturers have recently introduced color laser printers with four separate toner containers, one for each primary color.

Color laser printers typically use four toner colors: the three main subtractive colors (cyan, magenta, and yellow) and black.

The visible spectrum is continuous, so you can only approximate the visible colors by mixing the primary, subtractive colors (mixing cyan, magenta, and yellow toners together actually produces a kind of gray, not a strong black). Adding black toner produces more colors and a much more vibrant black. In all color laser printers, the color range is created by mixing toners in one of two basic ways:

1) In continuous tone printers, the amount of toner for each color in each dot can be changed. These printers are quite expensive, but they can also produce near-photographic quality reproduction. For a 32-bit continuous tone color printer, each pixel can have 4,294,967,296 different toner combinations. It is not necessary to print all of them, because, for example, a combination of strong black with C, M, and Y looks identical, and the user only suffers toner waste. Furthermore, the transition from 24-bit RGB to 32-bit CMYK is usually not obvious, so the user actually has access to 16,777,216 colors.

Not all the colors that can be represented on the screen in the RGB palette are created by mixing the four toners. Some of them will be out of scale and can be achieved by replacing them with a printable color that is perceived similarly. In the same way it will be possible to print colors that cannot be reproduced on the RGB screen .

2) In graduated toning printers, the amount of toner per color within each pixel cannot be varied. They are relatively cheaper, but their print quality is much lower, especially when reproducing a real-life image. Each of the four colors may or may not be present in each pixel, so each dot can only have 16 different combinations of toner. Furthermore, black, when mixed with any other color, will appear as black, so 8 of these combinations will appear the same, and each pixel can

actually appear as 9 different colors. The colors that cannot be represented directly are simulated by dithering, which causes the human eye to perceive intermediate colors in the same way as half-tone newspaper photographs. Printing is not possible in the 16,777,216 colors represented by 24-bit RGB. Only 9 different colors can be printed, so it is necessary to use the dithering process to show 24-bit colors. Dithering allows intermediate colors to be shown in a way comparable to a continuous tone. 24-bit data can be supplied to graduated toning printers without affecting the ability to mix different amounts of each toner for an equal range of colors in each dot [8].

1.6. Other types of printers

There are other printing technologies, such as line printer technology, used especially in mainframe systems. Printing is done by means of a rotating cylinder or chain containing all the symbols.

When a rotary cylinder is used, all columns containing the entire character set, the cylinder rotates in the direction of paper advance, and the paper is snapped between the cylinder when the symbol to be printed is in the correct position. One revolution of the cylinder prints a complete line of text.

In the case of a rotating chain, a set of symbols is repeated on the chain several times. The printer designer determines the number of times each symbol is repeated, depending on how often it appears in the text. In American English, this is the sequence ETAON.... If the symbols used are repeated more often, then the entire printout is faster. and the chain rotates perpendicular to the direction of paper movement, and the paper is snapped between the belt and the chain when the symbol to be printed is in the correct position. A variant of the chain printer is the belt printer. The print quality is similar to that of a chain printer, while using less power and making less noise. A standard belt printer uses a high-quality set of chains. On the IBM 3203 and 1403 printers, each stroke prints three symbols from the string. The use of a string of words is deliberate: the three symbols on the module follow one another as the string rotates.

Fig. C IBM 3203

In a belt printer, the symbols are arranged separately on fingers (similar to the arrangement of symbols on a daisywheel) placed on a rubber belt. As in a chain printer, not all the symbols are repeated at the same frequency. Unlike cylinder and chain printers, the belt is snapped between the strips of paper.

Print speed for these printers is usually given in lps (lines per second) or even pps (pages per second). Belt printers, chain printers, and rotary

cylinder printers (up to 70 lps) are recommended for increased print speed.

There are also thermal dot matrix printers that allow printing symbols and graphics (e.g. barcodes) by heating the paper (dots): the heated areas become black.

In the print-making fields, there are several other exotic technologies used, such as the aluminum-coated paper used by Sinclair , and the spark discharge technology. However, they are very rare and mostly outdated [8].

2. Plotter classification in

2.1. Board Plotters

Blackboard plotters are devices of quite large dimensions, close to the maximum drawing format. For example, an A1 or A0 plotter looked like a huge table on which sheets of tracing paper were electrostatically attached. Markers moved on the board on steel rails, and drawing even the smallest element took a very long time because the markers had to travel the entire way back and forth on the rails [11].

2.2. Pen plotters (drum plotters)

Much better are pen plotters (Fig. 4). These devices take up much less space, and the sheet of tracing paper is attached at only two points.

drawing s. D Pen plotter

The principle of operation can be most simply presented as a frame with a stretched paper moving on rollers in a vertical plane, while the carriage with the pens moves in a horizontal plane. The overall dimensions of the pen plotter had to be adapted solely to the width of the format. These were much smaller, quieter and faster devices.

The speed of work was constantly increased thanks to the software optimization of the movement of the pen carriage.

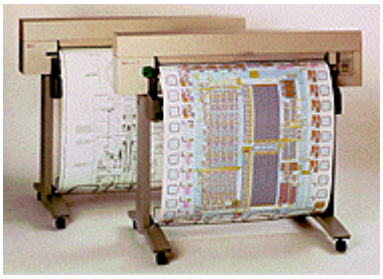
In pen plotters, the following disadvantages could not be overcome: color options were limited to changing pens, and filling fields came down to hatching them. Drawing bitmaps was possible only theoretically. The quality of the drawing depended on a number of factors, such as the quality of the ink and pen and the speed of the pen movement. In the case of complex, multi-layered

drawings in larger formats, inaccuracies of the order of millimeters occurred, caused by the accumulation of mechanical deviations of the pen movement.

The development of pen plotters (Houston Instruments laboratories) eventually led to the creation of a plotter with a scanning head. Installing such a head transforms the plotter into a large-format scanning device [11].

2.3. Inkjet plotters

In parallel with pen plotters, raster inkjet plotters were developed. This device works on the principle of spraying ink droplets on the tracing paper, similarly to an inkjet printer. Inkjet plotters are much more precise and easier to use, and the effects of their work are clearer.



drawing E DesignJet 350 C

Many companies are currently introducing large-format raster plotters to the market. The latest product on this market is the Hewlett-Packard DesignJet 750C . This is an A0 plotter, color, with an extremely high resolution of true 600 dpi in monochrome. Like most modern Hewlett-Packard products, it is a completely maintenance-free device. After loading a roll of paper and inserting ink cartridges into their places, the plotter can be turned on. DesignJet (Fig. 5) already takes care of all the nozzles working properly, of the paper being set correctly, it checks the media parameters and adjusts to the given format.

Hewlett-Packard's original contribution to the development of the raster plotter concept is the use of so-called pigment ink. Until now, the main argument against inkjet plotters was the phenomenon of nozzle drying. HP made sure that the product itself checks whether any nozzle is clogged and tries to fix the failure itself. Only the use of pigment ink brought radical effects and contributed to improving the print quality. This ink, unlike others, is a suspension of a solid body. A cartridge with such ink can no longer be refilled with so-called refills .

The effect of using pigment ink is to increase the clarity of the drawn line or surface. A layer of the substance sprayed onto the paper protects it from moisture and increases the durability and resistance of the drawing.

The fine-grained structure of the ink enables achieving a resolution of around 600 dpi.

Another interesting solution used in the 750C model is the compensation of the play in the carriage track teeth. It eliminates the effect of jagged vertical lines, which in the case of some other devices, especially when drawing large elements with thin lines, creates a less aesthetic visual

effect.

The DesignJet 750C is an extremely quiet and fast device. Although it cannot compete with pen plotters for simple drawings, it surpasses them for complex ones, with an A1 drawing time of around 4 minutes.

A great convenience is the possibility of equipping the plotter with the so-called HP JetDirect card . This is a network card, thanks to which the plotter is seen in the network as a device available to all employees and does not require setting up an additional server. 8 MB RAM, which is standardly equipped with DesignJet , allows for the creation of very large drawings (even of a size of several megabytes).

The drivers are designed in such a way that printing takes place in continuous mode in the computer's memory, simultaneously with the processing of vectors into bitmaps. Additional memory is needed only when we expand the system with PostScript functions [11].

3. Printer and plotter control languages

Modern printers can produce very complex drawings, sometimes competing with photographs or with documents received from a print shop. The way in which the data stream is sent from the printer is itself the format in which the image is stored in the file. There are two general types of print file formats for printers: extended text formats and page description languages.

3.1. Extended Text Formats

Such formats incorporate graphical information into the conventional text data stream. Plain text is printed, and control sequences introduce non-text elements. A widely used format is Printer Control Language (PCL). It has become the standard for low- and medium-performance laser printers [6], [15], [16], [8].

3.1.1.IBM Proprinter

The IBM ProPrinter printer language, originally used for IBM printers (IBM Graphics Printer 5152, IBM ProPrinter XL 4201/4202, IBM ProPrinter X24/XL24 4207/4208), is also currently used by many dot matrix printers such as Epson ESC/P and ESC/P2 . The functionality varies depending on the version used or selected during printer setup (XL, XL24 or AGM are examples of such versions).

The IBM ProPrinter control language in printers from different manufacturers may differ slightly in functionality. Usually the differences are additional features not included as standard.

3.1.2.Epson ESC/P, ESC/P2

ESC/P (Epson Standard Code).

ESC/P2 (Epson Standard Code, Level 2).

The ESC/P printer language was created by Epson for use in early dot matrix printers. It is now used by Epson inkjet and laser printers as well as many other dot matrix printers on the market. ESC/P2 is an improvement on ESC/P , having for example new functions for scaling fonts, for printing raster graphics, etc.

ESC/P or ESC/P2 printers from different manufacturers may differ slightly in their functionality. Typically, these differences are additional features not provided with the original Epson version .

Information about the version of ESC/P and ESC/P2 used can be found in the documentation for the printer from the individual manufacturer. Some of the entries are brief lists of available functions with examples in BASIC . You can also find tables with symbol sets and tables with font widths. The ESC/P2 Reference Manual from Epson contains a list of codes in ESC/P and ESC/P2 as well as a complete description of the differences in each printer's commands. The latest version is from August 1992.

3.2. Page description languages

Another approach to printer control is to define an entirely new print description language. Several such languages have been used in the past, but PostScript has become the current standard [8], [6].

3.2.1.PostScript

PostScript is a page description language produced by Adobe Systems Inc. since the early 1980s. Adobe was founded in 1982 by Dr. John E. Warnock and Dr. Charles M. Geschke. It provides instructions for describing a page of information. Because it requires more memory than most page description languages, it was the first widely available product to control a larger number of fonts and graphics.

The first version published in 1985 is called Level I, the current improvement is called Level II (not to be confused with: PostScript version 47.0 or 2011.110, nor with the number at the beginning of any PostScript output , like %!PS-Adobe-3.0). The PostScript level, and the extension of the version with an interpreter, expand the possible operations.

There are several PostScript clones in circulation , but, because of the cost of the Adobe

interpreter license fee; the best known is GhostScript . Others, built directly into laser printers or adapted by an additional cartridge, include Phoenix Page, BrotherScript , Page Styler, True Image, Turbo PS, PDL, and KPDL . They are said to be completely compatible with PostScript, but this compatibility sometimes breaks down during font loading, font manipulation (i.e. inserting a metric table or new symbols), and other operations. Using clones, however, there is no problem in printing simple text or graphics.

The PostScript symbol set is used to write a PostScript program , not the symbols of a printed PostScript font.

Adobe recommends using only the printable subset of ASCII symbols in PostScript , spaces, tabs, and the CR and LF symbols . PostScript does not prohibit the use of symbols outside this set, but using them can cause transmission problems (e.g., transferring 8-bit symbols over a 7-bit serial line to a printer). To represent 8-bit symbols outside a string, use the formula \ddd .

A file intended for a printer is actually a computer program in the PostScript language that draws the given image. PostScript is used to perform operations closely related to image generation.

PostScript programs draw graphics in two basic ways. The easier way is to draw a pixel map. One of the basic operators in PostScript reads a sequence of pixels and displays them in a rectangular area on the page. This approach is suitable for including scanned or monitor-scanned images in documents, or any other images that already exist in bitmap format.

An alternative way to use PostScript is to create a drawing in vector or metafile format. There are operators for drawing lines, circles, curves, rectangles, etc., and from these elements you can build graphics.

It is also possible to combine these two approaches. For example, the line, rectangle and curve operators can be used to define an area that will then be used as a clip mask to control the display of the bitmap [8], [11].

3.2.2. HP PCL and PJL

The PCL language was created by Hewlett-Packard for the needs of its printers (laser and inkjet). PCL versions are numbered from 1 to the current version 5e.

A Brief History of PCL (Based on HP's Printer Language Technical Reference Manual):

PCL 1 - Print and space functionality is the basis for features provided for a simple, comfortable single-user workstation.

PCL 2 - EDP (Electronic Data Processing) functionality is an improvement over PCL 1. General purpose, multi-user printing system functions have been added.

PCL 3 - Office Word Processing functionality is an improvement on PCL 2. Functions have

PCL 3 - Office Word Processing functionality is an improvement on PCL 2. Functions have also been added to improve print quality and process office documents (HP DeskJet family printers) .

PCL 4 - Page Formatting is an improvement of PCL 3. New page printing capabilities have been added (printers: HP LaserJet, HP LaserJet IIP (PCL 4,5))

PCL 5 - Office Publishing functionality is an improvement over PCL 4. New publishing capabilities include HP-GL/2 font and graphics scaling (HP LaserJet III, HP LaserJet 4 (PCL 5e) printers).

PCL versions differ in functionality (e.g., font types, bitmap fonts, scalable fonts (Intellifonts, True Type), graphics compression methods, graphics support by the HP LaserJet III).

PCL is the most common printer language in the current laser printer market. Most laser printer manufacturers use PCL 4 or PCL 5 for their printers.

PJL (Printer Job Language) was also created by HP to provide a method for changing job level and read status parameters between the printer and the host computer. PJL can be used at the beginning of printing to set certain parameters such as printer language (PCL, PostScript or others), resolution (300 or 600 dpi), number of copies, etc.

PJL is currently used in the following HP printers : LaserJet IIISi , LaserJet 4 family, PrintJet XL 300, and DesignJet.

PJL is also used in the LaserJet 5 series printers [11], [7], [10].

3.3. Other Printer Control Languages

There are many other unique printer control languages on the market. The following list is therefore not complete (the intention is to only mention them, not to describe them in detail). The order in which the languages are listed has nothing to do with their importance in the market.

Advanced Function Printing (AFP): is used in IBM Mainframes for page printers . It is a presentation function of the Mixed Object Document Content Architecture (MO:DCA) set, which is part of the IBM System Application Architecture.

Actually you don't print using MO:DCA , you use IPDS (Intelligent Printer Data Stream).

The information includes PTOCA (Print Text Object Content Architecture), GOCA (Graphic Object Content Architecture), IOCA (Image Object Content Architecture) and others.

IPDS is the IBM SAA printing language . It works with various bitmap fonts, with simple basic graphics, and with bitmap images. Because of the simplicity of the conceptual model, it can be used to drive and operate fast laser printers.

Diablo 630: Originally used with rosette printers and typewriters. Tolerates only tab

sequences, line and symbol spacing, attribute selections (bold, double-strike, underline), horizontal movements in both directions, proportional spacing, and automatic centering and justification - among others. This language is sometimes used by other manufacturers as a basis for their specific emulations.

CaPSL : (Canon Printing System Language) was the previous standard language for Canon laser printers . Another name that has been used is LIPS (Laser-beam Image System). There is a bit of history behind CaPSL - Canon manufactured printer engines for HP , but did not have a license to use HP PCL in its own printers. Hence the need to find its own printer language. Canon lasers traditionally included CaPSL, IBM ProPrinter, ESC/P, and PostScript emulations , but not PCL .) This part of the Canon-HP contract has apparently expired, as Canon now offers printers with PCL 4 and PCL 5.

LIPS supports Diablo 630 (factory setting for command mode), ISO mode (for printing text and raster graphics), and VDM mode (for vector graphics and symbol printing).

RENO : This is the standard control language for Agfa printers (P400, P3400 , etc.). RENO is a kind of page description language. Its functionality is enormous: in addition to printing text with different font scales, you can draw lines, fill icons (Windows) with patterns, use programming expressions (if-then-else, repeat-until, set, use and print variables, push and pop operations), can load and print your own symbols and transfer data to the printer's RAM or to the hard disk or diskette if one is included.

Prescribe : This is a page description language created by Kyocera . Its advantage is that it can be embedded in another current printer emulation on Kyocera devices . Kyocera printers support HP PCL , an HP-GL clone called KC-GL , Epson ESC/P (LQ-850 mode), IBM ProPrinter X24E, Diablo 630 , a general line printer emulation, and as an option KPDL , a PostScript clone .

DEC: DEC has its own unique languages for laser printers (LN03, LN06).

ANSI : Dan McGowan of Mannesmann Tally states that: Mannesmann Tally printers , which have ANSI , provided the basis for the ANSI 3.64 notation . This is a rather loose specification covering general peripheral functions. Printers manufactured in the USA, for the most part, have all the ANSI 3.64 commands pertaining to printer functions. Printers manufactured in Germany are a series of flying serial head printers. They have MTPL (Mannesmann Tally Printer Language), which is based on ANSI 3.64 but includes additional commands [8].

3.4. Plotter Control Languages

The name HP-GL (Hewlett -Packard Graphics Languages) refers to the language

originally used to operate Hewlett-Packard plotters . The year 1976, when the aforementioned plotters appeared on the market, is considered the date of creation of this language. Along with the improvement of these devices, the HP-GL language was enriched with new commands, and currently its second version is marked with the symbol HP-GL/2.

The HP-GL language also has support for vector fonts, which in practice means device control capabilities similar to those provided by PostScript . Much of the HP-GL/2 language has been incorporated into the fifth level of the PCL language , in which there are special commands that switch control from classic PCL to HP-GL and vice versa.

The syntax of the HP-GL/2 language is simple. All instructions are two-character abbreviations of their names. The abbreviation is followed by parameters. They may be mandatory or optional. A space or a comma may be used as a separator between parameters. The comma is the preferred separator. Each instruction ends with a terminator. Terminators may be a semicolon, the first character of the next instruction, or a space [10], [7], [6].

Translation systems

1. Converting between file types

Since there are many formats and languages for controlling printers, it is obvious that there is often a need to convert one print file format to another. Depending on the source and target formats, this may be a trivial task, or it may be impossible to do. Images stored in a print file can take many forms. It can be a bitmap image, a vector image, or even plain ASCII text . Considering the format of the language understood by the printer, it may be necessary to convert the format of the image stored in the print file [6].

1.1. Bitmap to bitmap

Converting one type of bitmap to another is usually easy. Once you get past the details of encoding the file, you're dealing with pixels that are almost always the same, so conversion is simple.

If it is from a more to a less descriptive format, e.g. from color to black and white or grayscale, there are known methods of converting it while maintaining the best possible image quality.

1.2. Vector to Vector Format

The basic issue when converting between vector formats is to adapt the slightly different semantics of the individual formats and also, to some extent, the coordinate system. In the simplest case, commands are translated one-to-one, e.g. a command to draw a line to the same command in the output file. Problems arise when two formats do not have corresponding commands. If the original format has a command to draw an ellipse, and the target format does not, then one of the available translation methods must be used. You can approximate an ellipse with a circle or a polygon made of short segments. If the target format has the ability to scale the x and y axes separately, then a good method may be to set different units on them and draw a circle, which will be scaled into an ellipse.

1.3. Vector to bitmap format

Converting an image from a vector to a bitmap is called rasterization. It involves finding a set of pixels corresponding to each vector in the original image. The basic rasterization algorithm has been known since 1963, when it was published by Bresenham ; circles and arcs can be drawn using the same method.

This type of conversion deals with the same issues as drawing a vector image on a raster screen or laser printer, so it is often possible to adapt the display code to convert to a bitmap.

1.4. Bitmap to vector format

Converting a bitmap to a vector is far more complex than any of the previous conversions. There are now satisfactory edge detection algorithms. They can be used to find lines in a bitmap, but only in the simplest cases. The problem of finding lines in a scanned image (e.g., a faxed document) remains unsolved.

2. Converting programs

2.1. Replacing PostScript with Other Standards

The best known PostScript interpreter is GhostScript . By calling GS with the -help option you can get information about available devices. GhostScript can emulate the following systems: epson, epsonc, necp6, laserjet, ljetplus, ejet2p, ljet3, paintjet, bj10e, djet500, djet500c, pjetxl, lbp8 (these are the names used in the program). It is also possible to add new drivers to the program in [8].

2.2. Changing Other Standards to PostScript

In the case of ASCII text, converting to PostScript is not complicated. There are several publicly available tools for this purpose. The simplest of them is a program called a2ps.

In the case of non- ASCII text (ISO 8859-1 or PC 437), or printer specific control sequences, conversion can be complicated. To convert HP PCL to PostScript, the lj2ps utility is used , which is very helpful in the case of non-proportional fonts (as for HP LaserJet II). The problem, however, becomes more complicated when converting graphics. There is also a converter for HPGL (hpgl2ps), but it is rarely found. To convert Epson to PostScript, the epson2ps filter [8] is used.

EmuLator App

The EmuLator program is designed for the Windows environment from version 3.1. The choice of this system was mainly determined by its simplicity in use and communication with the user via a rich graphical interface. The program is easy to use thanks to the complexity of programming using the API (Application Programming Interface) functions - a set of functions that programmers must use when writing applications for Windows. Windows API contains over 600 functions. Apart from the aforementioned inconveniences, it is necessary to mention the benefits of programming for Windows [13]:

Windows provides hardware independence. The same program can display information on different monitors (EGA, VGA , etc.) and print to different printers, from dot matrix to laser.

From a programmer's perspective, Windows provides many ready-made user interface elements, such as screen buttons, menus, dialog boxes, lists, and edit boxes.

Windows includes an extensive Graphics Device Interface (GDI) for displaying text and graphics. In particular, this interface allows you to draw in your own coordinate system.

The programming language chosen was C++ in the latest version at the time of program creation, the Borland 4.52 compiler [4]. This tool allows for an object-oriented approach to the designed application [2] and enables the use of OWL and CLASSLIB class libraries [1].

The primary purpose of the OWL (Object Windows Library) class library is to provide programmers with a complete framework for Windows applications. Borland's OWL 2.5 was used, which allowed for the rapid creation of a user interface with attractive, graphical controls.

Borland's CLASSLIB container library provided output objects for storing and processing data.

1. Program description

The program is designed to solve the problem of interpreting a graphic image contained in a print file on a computer screen. A large number of printer and plotter control languages and various forms of their recording encourage the programmer to interpret the selected printer language. Such an approach can be observed, for example, in the GhostScript program interpreting the PostScript language . This work consists of writing a program that will interpret various control languages.

It is assumed that the print file contains various commands causing graphic actions such as:

printing text or drawing graphics and sequences changing printer parameters. The results can be seen on the printed sheet, so they can also be presented on the computer screen. Individual control languages cause the same action using different combinations of codes. In order to simulate the operation of a printer or plotter, the program contains a set of basic operations performed by these devices, and enables the execution of these operations.

A significant problem was the use of any printer language by the program in a way that was accessible to the user, while at the same time allowing for the addition of new control languages at any time. The emulator opens the printer driver, which is a text file in a fixed format, checking whether it does not contain any errors in the record. This file also contains the initial settings for a given printer or plotter. The user can create a driver for each printer himself, using some of the operations included in the program, without the need to recompile the program. If a given printer uses mechanisms specific to it, not included in the program discussed, a skilled programmer can quickly expand the program with new functions.

The application uses MDI (Multiple Document Interface) mechanisms, which allow viewing several documents at the same time. The program allows interpreting printouts for only one type of printer at a time.

The final effect can be viewed on the screen in 1:1 scale (taking into account the inaccuracies of the monitor screen), in a full-page print preview, or printed on any printer. Using the hardware independence of the GDI device context , the program allows you to convert the known printer or plotter format to the format of any device available in Windows.

1.1. Construction

The application was created based on the MVC (Model-View-Controller) architecture, commonly used in the Smalltalk-80 language, taking into account the specific capabilities of the OWL 2.5 library . The MVC architecture divides the application into three layers:

1. Model means the application layer, in which all objects dependent on this application are placed. In EmuLator these are all objects representing the data layer and operations performed on them. In particular, it can be a programmable dictionary, enabling translation of code in the print file into graphic operations, and a class processing the print file data into a graphic image, used in the presentation layer.
2. View is the so-called presentation layer, which is responsible for the presentation of data. The presentation layer reads the appropriate information from the application layer and displays it on the screen. This layer also deals with the windows of the graphical user interface.
3. Controller is the so-called control layer, mediating the transfer of information from input devices (keyboard, mouse) to the other two layers.

This model undergoes minor modifications during implementation and is subordinated to new techniques found in OWL 2.5, such as Document/View , depending on the problem being solved.

1.1.1. Control layer

1.1.1.1. Main Application Class

class emuApp: public TApplication - the most important class of the application, the implementation of which is in the files emuapp.cpp and emuapp.h , where the auxiliary class *TFileDrop was also defined,* which deals with handling documents loaded in the drag and drop method . The *emuApp* class is the base for all objects in the application. In the *InitMainWindow* function, objects of *the emuMDIFrame and emuMDIClient classes are created ,* the program's graphic elements (such as icons and menus) are loaded with resources, the status line (*TStatusBar*) and toolbar (*TControlBar) are inserted into the emuMDIFrame* frame , the attributes of the main window are set, and spatial elements of the dialog are activated with the *EnableCtl3d(true)* function . In addition to handling standard Windows messages, the *emuApp* class is responsible for the following tasks for the application in question:

help system support - loading the help file in regular or context mode;

creating a new document and view - when opening a new document and creating a view, the user is informed if the driver file is not loaded; when loading a new driver file, all windows are closed;

passing a control message for the print file document view (*emuView*) - the *emuApp* class passes a request to the current *emuView* view to go to the next or previous page, informing about its current number in the toolbar;

updating information in the status line - the *emuApp* class informs about the currently loaded printer driver or its absence, and when selecting a command from the toolbar or menu, a hint is displayed.

1.1.1.2. MDI Application Control Classes

class emuMDIFrame: public TDecoratedMDIFrame (emmdifrm.cpp, emmdifrm.h) - contains visible controls and handles all control messages intended for *emuMDIChild* child windows by directing them to *emuMDIClient*.

class emuMDIClient: public TMDIClient (mmdiclnt.cpp, mmdiclnt.h) - is responsible for managing *emuMDIChild* window objects in response to requests from the Window menu.

class emuMDIChild: public TMDIChild (mmdichld.cpp, mmdichld.h) - defines the basic

class emuMDIChild: public TMDIChild (emumdi.cpp, emumdi.h) - defines the basic behavior of the MDI window , containing a pointer to a *TWindow* object , which is a visible element of the document view.

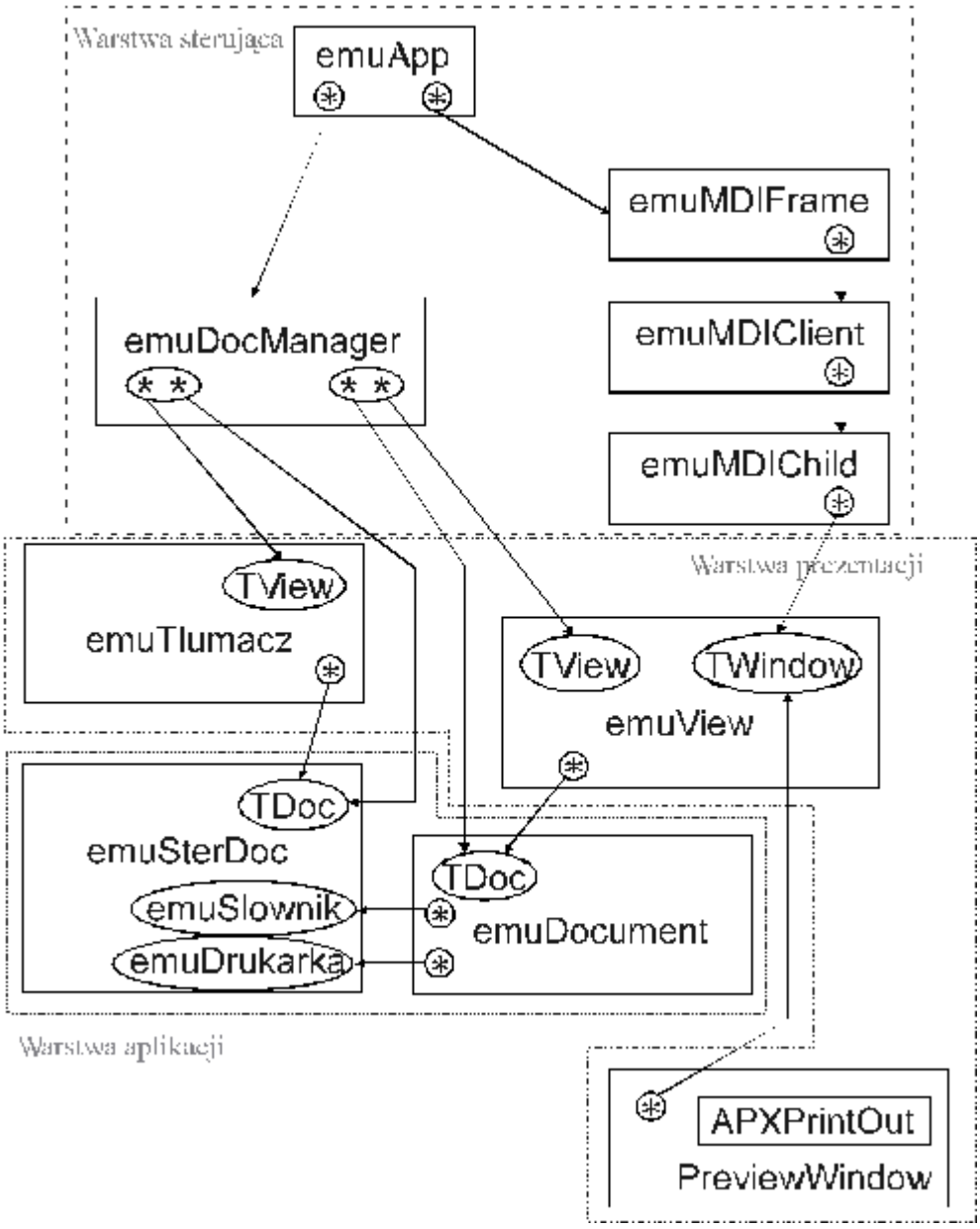


Fig. F Hierarchy of main application classes

1.1.1.3. Document Control Class in class

emuDocManager: public TDocManager (emdcmngr.cpp, emdcmngr.h) - an object managing the list of current, registered templates. The *DEFINE_DOC_TEMPLATE_CLASS* macro creates a template associating document objects with views, based on which *TDocManager* handles standard *File* menu commands , passing them to appropriate documents. It is also responsible for displaying user interface elements for selecting a file and a view.

1.1.2. Presentation Layer

1.1.2.1. *Printer Driver Data Presentation Class*

class emuTlucza: public TView (emutlmcz.cpp, emutlmcz.h) - an object that mediates access to data contained in the document. *emuTlucza* allows you to obtain a pointer to the dictionary class (*emuSłownik*) created in the document object of the printer driver (*emuSterDoc*). As the only view in the application, *emuTlucza* does not have a window associated with it and is specially handled in the main object of the application. For each document object of the print file (*emuDocument*), *emuApp* sets a pointer to *emuTlucza (only one emuTlucza* object can exist at a time).

1.1.2.2. *File to print data presentation classes*

class emuView: public TWindowView (emuview.cpp, emuview.h) - a document view object of the print file inheriting properties of two OWL classes . As a descendant *of TView* it supports access to *emuDocument* data . Inheriting from *TWindow* is a graphical representation of the document in the form of a visible window, the pointer of which is passed to the *emuMDIChild object* . In response to requests to display the next page, *emuView* passes the *emuDocument* object a display context for drawing an image of the print page. In order for each response of the view to the *WM_PAINT* message not to trigger the entire mechanism of translation, interpretation and creation of a drawing from the print file, the metafile context is used, stored in the form of an object of the *TMetaFilePict* class .

class APXPrintOut: public TPrintout (apxprint.cpp, apxprint.h) - represents a physical document sent to the printer. An object of this class is responsible for drawing the page on a physical printer or in a print preview window.

class PreviewWindow: public TDecoratedFrame (apxprev.cpp, apxprev.h) - creates a print preview frame and supports displaying one or two pages at a time in a *TLayoutWindow object. It is also responsible for printing on a physical printer as a result of selecting the application's Print* command.

1.1.2.3. Other classes

class emuEditView : public TEditView (emuedtvw.cpp, emuedtvw.h) - view class associated with any file type (directly *TFileDocument*); allows basic editing, text printing and searching operations.

class TBmpViewWindow (emuabout.cpp, emuabout.h) - used to present the initial program vignette as a bitmap.

class emuAboutDlg: public TDialog (emabtdlg.cpp, emabtdlg.h) - using the

class emuAboutDlg: public TDialog (emuabtdlg.cpp, emuabtdlg.h) - using the *ProjectRCVersion* helper class , which provides project information, displays a dialog box about the program.

1.1.3. Application Layer

1.1.3.1. Printer Driver Document Classes

class emuSterDoc: public TFileDocument (emustrdc.cpp, emustrdc.h) - represents a document data object and provides a way to interpret it for the view. It contains a number of methods for handling a physical file on disk, including handling streams. Each document can be associated with several views, so it supports communication with them by creating a list of current views and sending them messages about any changes. Documents and views have a list of properties (*Property*), based on which the application decides how to process data. *emuSterDoc* , among its properties, has pointers to the objects *emuDictionary* and *emuPrinter* created by it when opening a printer driver document .

class emuDictionary: public ContainerType (emuslwnk.cpp, emuslwnk.h) - this object derives from the container dictionary class created with the macro *typedef TDictionaryAsHashTable<AssociationType> ContainerType;* is a set of associations, along with methods for adding and searching for them. Association (*typedef TIIAssociation<emuMyClass, GraphicalObject> AssociationType;*) is a pair of pointers to objects *emuMyClass* and *GraphicalObject* . *The emuDocument* class object , finding the control code, performs a graphic operation derived from *GraphicalObject* . During the creation of the dictionary by *emuSterDoc* , only those graphic objects are dynamically created and added that are necessary for the interpretation of a given device.

class emuMyClass (emumycla.cpp, emumycla.h) - a single printer or plotter code, along with a comparison operator to find it.

class GraphicalObject (emugrobj.cpp, emugrobj.h) - base class for graphical operations in the device context, from which objects corresponding to individual control codes are virtually derived. An object of this class does not perform any operation, but for diagnostic purposes prints the name of the operation to be performed.

class GraphObjHPGL: public virtual GraphicalObject (emughpgl.cpp, emughpgl.h) - base class for plotter-specific graphical operations. The classes for specific graphical operations are derived from it: *class HPGL_0xXXXX: public virtual GraphObjHPGL* , where *0xXXXX* is a unique internal application number for a given operation.

class GraphObjPCL: public virtual GraphicalObject (emugopcl.cpp, emugopcl.h) - base class for graphic operations specific to inkjet and laser printers. From it derive classes of specific graphic operations *class PCL_0xXXXX: public virtual GraphObjPCL* , where

specific graphic operations *class TGL_0xXXXX: public virtual GraphObjTGL* , where *0xXXXX* is a unique internal application number for a given operation.

class GraphObjIBMPro: public virtual GraphicalObject (file emugopro.cpp, emugopro.h) - base class for graphic operations specific to dot matrix printers. From it derive classes of specific graphic operations *class IBMPro_0xXXXX: public virtual GraphObjIBMPro* , where *0xXXXX* is a unique internal application number for a given operation.

class emuPrinter (emudrkrk.cpp, emudrkrk.h) - a class containing the initial settings of the emulated device, also used when creating a graphic image.

1.1.3.2. Print File Document Classes

class emuDocument: public TFileDocument (emudcmnt.cpp, emudcmnt.h) - handles operations related to the physical print file. On request, *emuView* parses the file, looking for control codes in *the emuDictionary* and performing graphical operations with the virtual *Draw()* function of an object derived from *GraphicalObject* .

class emuPage (emustron.cpp, emustron.h) - class supporting creation of drawing of a specific page. *emuDocument* uses a set of pages *(typedef TIArrayAsVector<emuPage> emuPages;*) to remember the position of the print file stream and printer settings (*emuPrinter*) for each page.

1.2. Expansion

The program provides an excellent framework for extension with new functions and modules. Using the OWL Document/View technique , other types of views and documents can be created for existing or new objects. Also, modules that interpret print files can be easily extended. The comments in the source files and the helper classes that were not built into the final version of the program will be helpful in extending the program.

1.2.1. Adding an object to a view in

To add a view object, simply define a new class derived from *TView* , which will use the methods contained in the selected document object. If special data forms are used in the view for presentation, they must be defined in the document object. The view object can be associated with a document that has several other presentation methods. In order for the program to support them, appropriate templates must be defined for *emuDocManager* , e.g.:

```
DEFINE_DOC_TEMPLATE_CLASS(emuSterDoc, TSerListView, DocType7);
DocType7 __dvt7("User Viewer", "*.emu", 0, "EMU", dtAutoDelete);
```

where: *emuSterDoc* is an existing document class, and *TSterListView* is the controller view class. The view can be in the form of a window shown on the screen or in any other form of

interpretation of the data contained in the document.

1.2.2. Adding an object to a document in

Each document object is responsible for loading data and correctly feeding it to views. If a document has several views and one of them allows data changes, it is also necessary to handle messages that pass information about this fact to other views. Communication between views and documents can take place via messages, property lists (*Prosperities*), or direct pointer references. More information about document cooperation with views can be found in the Borland C++ 4.52 compiler documentation .

1.2.3. Expansion of graphic functions

In the EmuLator program, the following function is used during the interpretation of the print file:

```
int emuDocument::Draw( TDC& strDC, emuPage* page )
```

where: *strDC* is the device context in which the image is to be created; *page* is a pointer to an object containing the current page settings and a copy of the printer settings (*emuPrinter*) from the previously created page, or (in the case of the first page) the device settings read by *emuSterDoc* . If control code is found in the document stream, the emuDictionary object *returns* a pointer to the association from which a pointer to the appropriate graphical object is obtained. The operation is performed by calling a virtual function of an object derived from *GraphicalObject* :

```
if (found)
{
if (object = found->Value())
object -> Draw(strDC, stream, page);
```

The function *void GraphicalObject::Draw(TDC& dc, TInStream* is, emuStrona* str)* to perform the task is called with the following parameters:

TDC& dc - reference to the device context on which the graphic operation should be performed;

TInStream is* - a pointer to a print file stream from which further data can be retrieved to perform the operation;

emuPage str* - pointer to an object where you can find the current settings of the emulated device (*emuPrinter *prints*) and, in case of an error or end of page, you should set one of the values of *the status* variable :

```
enum {
Error=0
```

```
Error=0,
EndPage,
EndFile,
Next,
};
```

To extend the program with new graphic functions, you need to create an object derived virtually, directly or indirectly (e.g., like *GraphObjHPGL* objects) from the *GraphicalObject* class , which performs the operation using some of the above parameters. Then you need to choose a unique application internal number for the graphic operation (from the range 0x0000 to 0xFFFF) and place the object addition operation in emuslwnk.cpp:

```
int
emuDictionary::AddItem(const string& mc, const unsigned long int & mv,
const string& mvs)
{
int result=0;
switch (mv) {
case 0xXXXX: {
AssociationType assoc( new emuMyClass(mc),
new GraphicalObject(mvs) );
result=Add(assoc);
break;
}
```

This function is called by *emuSterDoc* when reading a device driver file with the following parameters:

mc - control code of the emulated device;

mv - internal application code for graphics operations;

mvs - operation name used for diagnostic purposes, taken from the driver file.

1.2.4. Extending the properties of emulated devices

If parameters are needed to perform a graphic operation, which cannot be found in the *emuPrinter* definition (emudrkrk.h file), they should be added to this class. The value initialization takes place in *bool emuSterDoc::GetPrinter(TInStream* is)* where a procedure for reading parameters should be added, e.g.:

```
if (key.contains("PAGE SIZE")) // Page size
{
row += SkipAll(*is);
long x = GetLong(*is);
```

```
row++;
if (!is->good()) goto error;
row += SkipAll(*is);
long y = GetLong(*is);
row++;
if (!is->good()) goto error;
printer->SetRStr(x*(metric ? mm_pkt : cal_pkt),
y*(metric ? mm_pkt : inch_pkt));
}
else
```

key is a keyword encountered in the printer driver file using the following convention:

```
!Page Size
```

Then, any parameters are read from the text stream *is with* the appropriate function for them. The *SkipAll(*is)* function skips all characters in the stream until it encounters the ! character at the beginning of the line. The variable *line* indicates the line number where the error occurs. *The x*(metric ? mm_pkt : cal_pkt)* construct allows you to set the parameters in predefined units (currently mm or inch).

1.2.5. Helper Classes

When running the project, you can use the debug classes included in the source. To do this, you need to enable debugger code inclusion in the project options, and change the comments in the emuapp.cpp file, activating the templates for these views. These are the following classes:

class TDumpView : public TListBox, public TView (dumpview.cpp, dumpview.h) - allows association with any type of document, and shows exactly its content with hexadecimal codes of individual bytes in the file;

class TInfoView : public TWindowView (infoview.cpp, infoview.h) - allows you to see the list of properties (*Prosperities*) of any document;

class TSterListView : public TListBox, public TView (strlisvw.cpp, strlisvw.h) - a view specially created for the needs of the *emuSterDoc* class document , showing the contents of the dictionary initialized with the printer driver file.

Program support

Instructions for use

The EmuLator program requires installation on the computer's hard drive before it can be used. Its installation package is located on HD 3.5" floppy disks. The installation process is typical

used. Its installation package is located on HD 3.5" floppy disks. The installation process is typical for the Windows environment .

After installation, the EmuLator program has a group in the Program Manager window called Printer and Plotter EmuLator . The main application program icon is the icon called EmuLator .

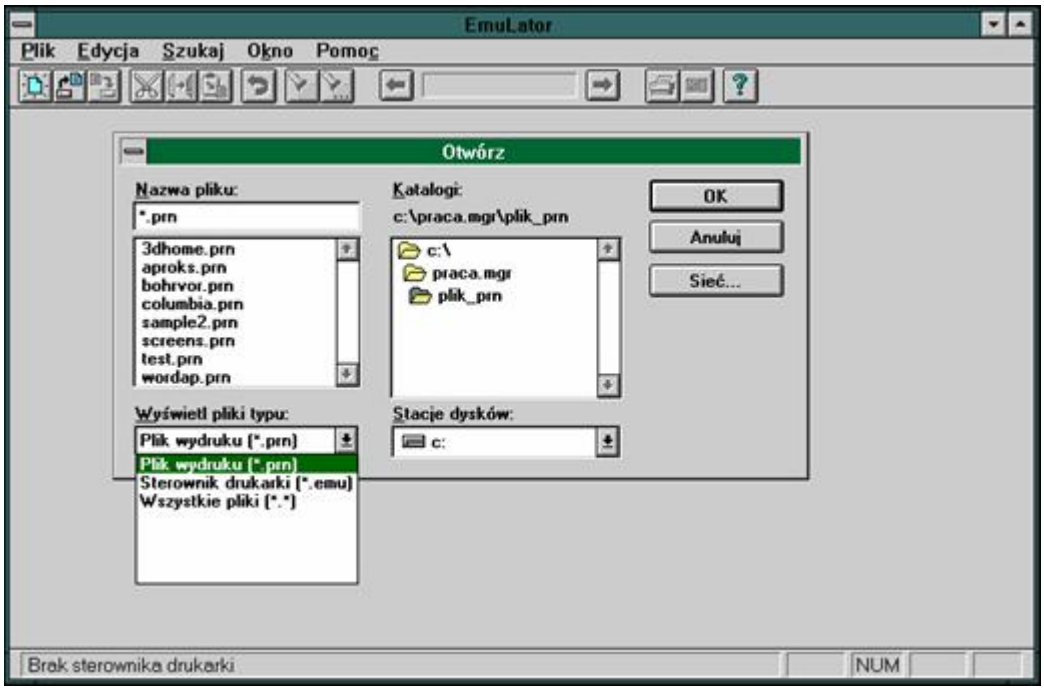


Fig. G. The main application window

After starting the application, an application window titled EmuLator appears on the screen (Fig. 7).

Program operation operations are usually limited to a few basic ones:

Opening a document

Opening a document containing a printer printout must be preceded by loading the device driver for which the file was created. Loading the driver is done by selecting the *Open* option from the *File menu* or clicking the appropriate icon on the toolbar. After expanding the Open dialog box , set the file type to *Printer Driver* with the *.emu extension and enter *the* name of the driver corresponding to the given printer in the File Name dialog box . The name of the device for which the driver was loaded will appear in the status line at the bottom of the window (Fig. 9).

the driver was loaded will appear in the status line at the bottom of the window (Fig. 9).

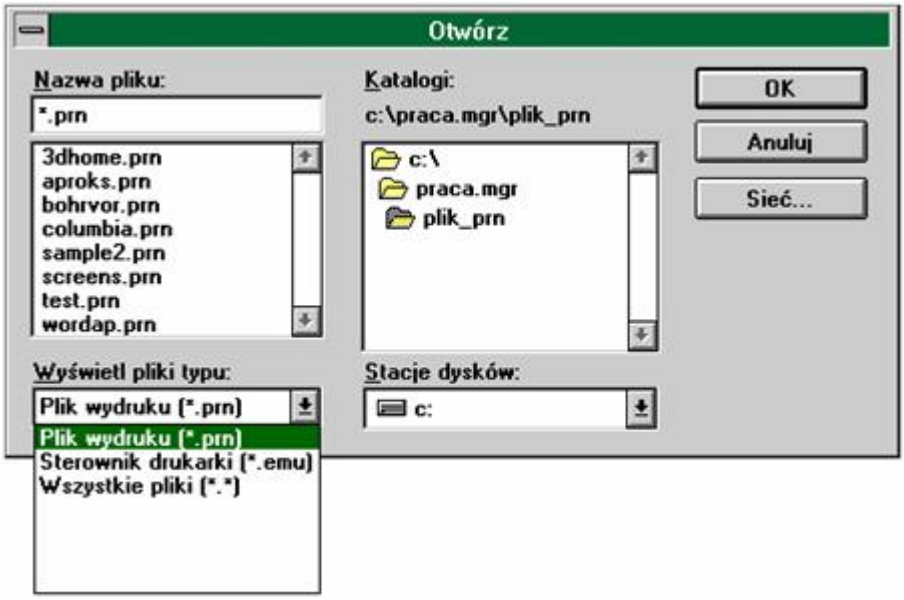


Fig. H Window open

If the command code is incorrectly saved in the controller, when attempting to load it, the program will display information about the line number with the incorrectly entered code.

Fig. I Status line with loaded driver.

The program is prepared to open a print file. The procedure is similar to the one above, with the difference that as the type of file to be opened, *Print Files* with the extension *. prn should be selected . When trying to reopen a document that has already been loaded, the program will inform the user about this and will not allow him to do this. However, it is possible to add other views to the same document; the *Add View* option from the *Window* menu . The contents of the print file will be displayed in the MDI window with the title of the open document. The next (previous) page of the document (if it exists) is displayed after selecting the *Page* option in the *Edit* menu or the arrow button in the toolbar (Fig. 10).

Closing the document

To close an open document, select *Close* from the *File* menu . When you change the device for which the document is displayed, all views (documents) are closed automatically.

Print preview

The document can be displayed as it will appear on the page after printing by selecting the *Print* Preview option from the *File* menu or the appropriate icon from the toolbar. The new image

of the

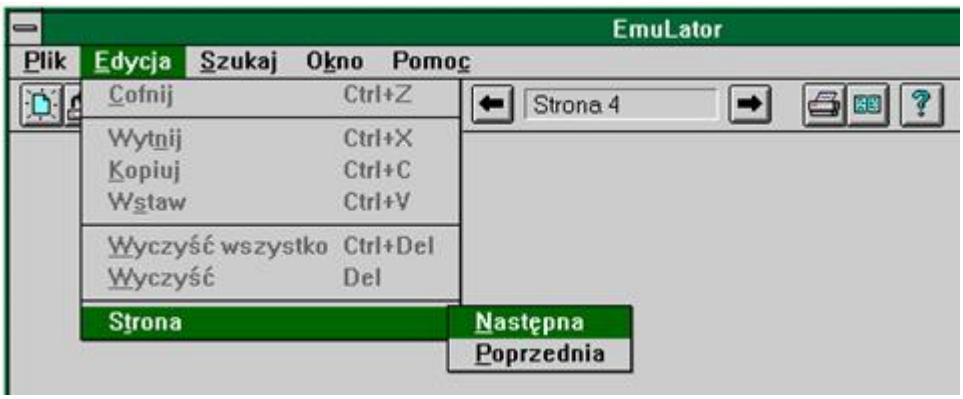
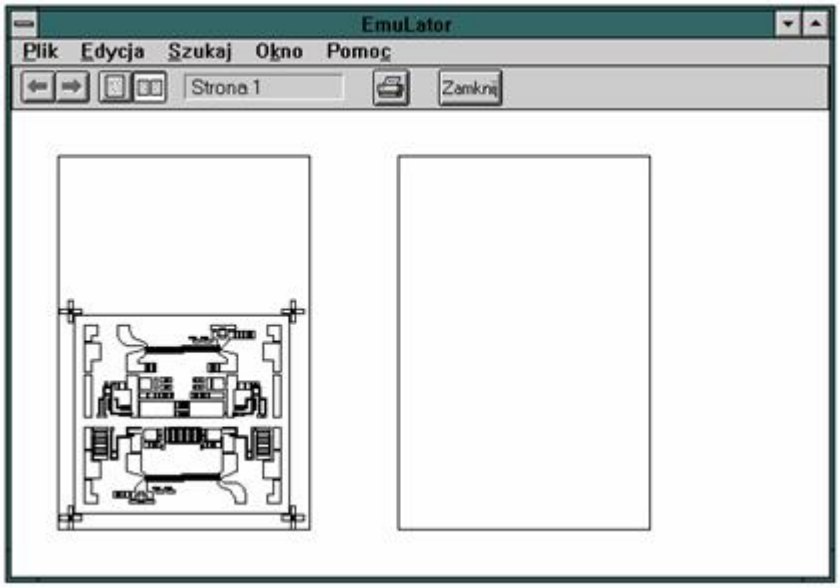


Fig. J Browsing document pages

document window shows one or two pages of the print file. The next (previous) pages can be viewed by clicking the arrow icons on the toolbar (Fig. 11).

Print document

After reviewing the document, you can print it on a device connected to the workstation. Printing is done by selecting the *Print* option from *the File* menu or by clicking the appropriate toolbar icon. Printing will take place in the current printer settings, which can be changed in the *Printer Settings* option in the *File* menu .



Editing the printer driver

Fig. K Print preview

Printer drivers can be modified to suit the device being used. They must be loaded in the *Open* dialog box with the *All files (* *) file type selected*. Editing operations are enabled by *the*

Open dialog box with the *All files (*.*)* file type selected. Editing operations are enabled by the *Edit, Search, Window* menu options . The driver cannot be loaded while being modified.

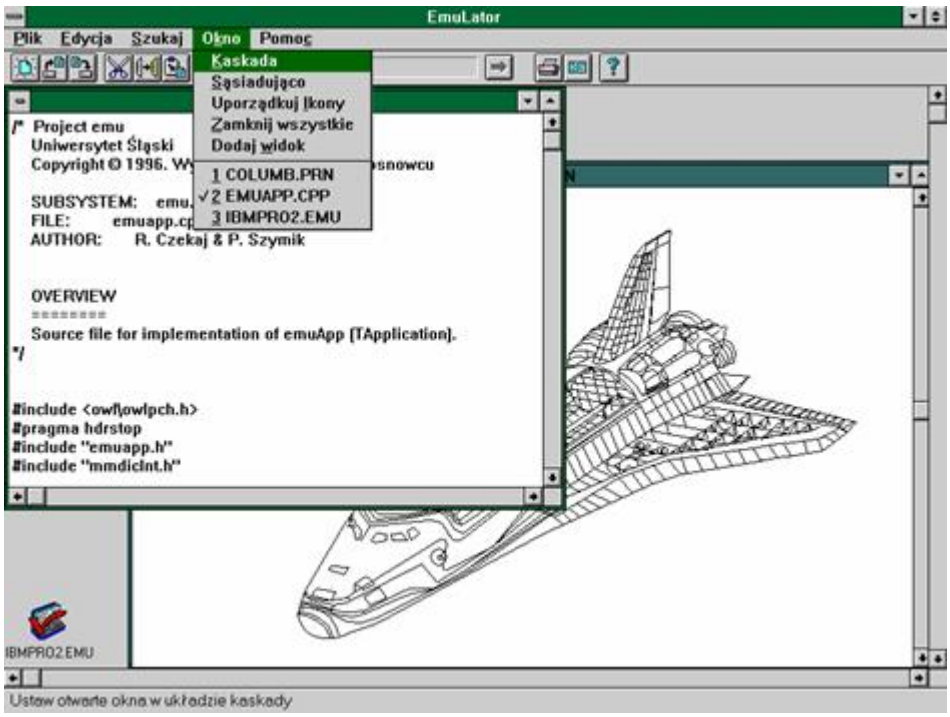


Fig. L Editing text files

Getting help

The help system for the EmuLator application can be invoked from the *Help* menu . It allows you to obtain help based on a search keyword or a search topic. When you press the Shift+F1 key combination, the mouse pointer changes shape, allowing you to point to the control element about which you need information.

Driver Description

The printer driver file format was created for the EmuLator application using a simple notation convention. Unlike other programs, EmuLator uses drivers that are not compiled programs. They are saved in a text file format that does not require any additional operations. The program has a built-in check for notation correctness and when an error is recognized, the user will be informed on which line it occurred. The number of printer control codes that can be saved is basically unlimited, as is the number of possible initial settings for a printer or plotter.

Each line starting with the characters ! , # , $, % is treated as a comment by the driver code reader. Each line starting with the character ! is read by the printer setup initialization module, and each line must start with a comment character or contain valid data. From the moment the comment character is encountered, the program ignores subsequent characters in the stream until it encounters the end-of-line character.

The first line of the driver is required, the second one without comment denotes the name of

the emulated device. It is given in the program status line.

```
EMU Driver File
$ Plotter driver file:
HP-GL 2 Plotter
$ for EmuLator program
```

The command description is copied by the stream handler as a whole line.

The following fragment contains data for one printer control code.

```
# Example of how to describe a given plotter code:
2 2 bytes of code to read
0x2b 0xff device control code
command description code word description
0xffff EmuLator program code for the operation (hex from 0x0 to
# 0xffff)
```

Numbers can be entered freely according to the conventions of the C language:

```
22 - DEC
0x22 - HEX
022 - OCT
```

separated by a space, e.g. 0x56 0x53

The commands are divided into groups. The division is introduced only to increase the clarity of the controller's recording.

```
#####################################################################
# HP-GL2 LANGUAGE CODES COMMAND #
# GROUP COMMAND IN "0A" #
#####################################################################
```

The actual description of the command code follows in the next lines of the driver file. The first parameter indicates the number of bytes in the driver code to read. The next line contains the actual device control codes. After the description line, there is a unique number of the graphic operation of the EmuLator application . In the case below, it is a single-byte code 0D meaning carriage return, located in group 0A, and marked with number 11.

```
1
0x0D
carriage return
0x0A11
```

The description of one printer control code must be separated from the next one by a comment character, e.g.:

```
1
0x0A
LF line feed
0x0A01
#
2
0x56 0x53
drawing speed selection
0x131B
```

The list of all available graphic operations, with their corresponding numbers, is located in the printer driver directory, in the *file emu_nrop.txt* .

After describing all the necessary printer or plotter control codes, the initial settings of the emulated device are described. The first item of a single block describing one of the settings is a keyword, the next ones are its values.

```
#!!!!!!!!!!!!!!!!!!! PRINTER INITIAL SETTINGS!!!!!!!!!!!!!!!
# First line is the type of settings, second is the value
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Device Type
!Device
#!0 # unknown type
#!1 # dot matrix printer
#!2 # inkjet printer
#!3 # laser printer
!10 # plotter
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# The unit used hereinafter
# mm
# inch inch
!Unit
!mm
#
```

Lines that are not commented out initialize the values to defaults. They are changed when control code that modifies a given value appears in the print file.

Summary

The EmuLator application was created with the intention of providing a universal solution to the problem of interpreting files in printouts. It is located in the installation version on diskettes attached to the descriptive part of the work. The program contains a set of basic graphic functions that allow for interpreting files in the printouts of the IBM ProPrinter dot matrix printer and the HPGL plotter.

The print file may contain simple text data and a graphic image. The program analyzes the print file and after finding the control code, it calls the corresponding graphic function, presenting the effect of the operation on the computer monitor. Searching the file for control codes consists in comparing characters taken from the print file stream with known codes saved in the controller's text file. After recognizing the control code, the next bytes of data are treated as arguments to the graphic operation call. An unrecognized sequence of characters is treated as simple ASCII text shown on the screen. Files saved in text format are also presented in this way.

The program can be easily adapted to emulate other types of devices by editing text files in the printer driver in terms of built-in graphic functions. It is possible to quickly create other objects performing other printer or plotter functions. However, this involves a complete recompilation of the program.

In the described form, the program can be adapted to read any unknown print file format, as well as other files containing any data that can be displayed on the monitor screen (e.g. vector and bitmap graphics, text editor, etc.).

It must be admitted that the universality of the created program did not positively affect the speed of analysis. Control codes are searched for starting from the longest word found in the dictionary. This method allows for unification of support for different control formats, i.e. it is possible to read both control languages in text format (e.g. PostScript) as well as control sequences starting with the ESC character (e.g. PCL and ESC/P). However, for files containing mostly ASCII characters, the speed of analysis decreases with the increase in the length of the control word. It is possible to modify the program by replacing the Borland container class (from which *the emuSłownik* is derived) with a faster data structure with a more efficient search method. Another solution is to use different file analysis methods for each format. However, this requires the programmer to write a separate module for each of them, similar to traditional compiled drivers and import filters.

It is possible to improve the program by enriching it with mechanisms for exchanging data using the Clipboard, or even OLE 2. By adding appropriate functions to classes derived from

GraphicalObject , it may even be possible to control and improve individual objects, e.g. scaling for bitmaps, any transformations for vector drawings, or changing the font type or size.

The emulator allows you to print viewed files after interpreting them. Printing data is possible on any printer available in the Windows environment. Due to the hardware independence of the GDI device context , the program allows you to perform an implicit conversion of any format known to it, in a way that allows it to send the results to any device available in the Windows system.

Taking into account the above considerations and observations resulting from the construction and operation of the EmuLator application, the designers believe that the goal and assumptions of this work have been fully realized.

Literature

[1] Barkakati N.: *Graphics and Animation in Windows.* Warsaw, Intersoftland 1994, (transl. from English).

[2] Barteczko K.: *Object-oriented programming; A practical introduction to object-oriented programming in C++.* Warsaw, LUPUS 1993

[3] Dro d ewicz P.: *Programming for Windows in C.* Warsaw, Lynx-SFT 1994.

[4] Faison T.: *Borland C++ 4.5 object-oriented programming.* Warszawa, Oficyna Wydawnicza READ ME 1996, (translation from English).

[5] Klein M.: *A Guide to DLL Libraries and Memory Management.* Warsaw, Intersoftland 1994, (transl. from English).

[6] Levine J.: *Graphical File Programming in C/C++.* Warszawa, Translator 1994, (transl. from English).

[7] Marciniak A.: *PCL language.* Poznan, NAKOM 1992.

[8] McCoy BC: *Summary - printers FAQ Home Page.* Usenet, comp.periphs.printers Frequently Asked Question (FAQ) Letter, 06/08/1996

[9] Osiak S.: *PostScript step by step.* Warsaw, Agencja Wydawnicza M&M 1991.

[10] Smith NE: *Laser printers.* Warsaw, ZNI MIKOM 1995, (transl. from English).

[11] Sowi ski R.: *The plotter's star.* CADmania No. 5 (11), November 1995

[12] Wacławek R.: *Program support for laser printers.* Warsaw, Komputerowa Oficyna Wydawnicza HELP 1992.

[13] Wac awek R.: *Windows from the kitchen.* Warsaw, Komputerowa Oficyna Wydawnicza HELP 1993.

[14] Zalewski A.: *Programming in C and C++ languages using Borland C++ package.* Poznań, NAKOM 1995.

[15] *Dot matrix printer D-100MPC.* B onie, Zak ady Mechaniczno-Precyzyjne Mera-B onie 1991.

[16] *LC-20 printer user manual.* Warsaw, Intersoftland 1991.

Abstract

The discussed work presents technical aspects of the operation of modern printers and plotters and methods of controlling these devices using print control languages. The described computer program was created as a result of analysis of individual formats in control languages.

The EmuLator application enables emulation of the selected printer and plotter, and allows for graphical presentation of the file in the printout on the computer screen. The program was built using the Windows 3.11 graphical environment and the Borland C++ programming tool package in version 4.52.

The EmuLator program contains a set of basic graphic functions that allow for the interpretation of files in the printout of the IBM ProPrinter dot matrix printer and the HPGL plotter . Within the scope of these functions, each user can easily prepare an application for analyzing files in the printout for other devices by editing text files in the printer driver.

Detailed documentation and comments in the program source files allow the program to be extended with new graphic functions for other devices. The object-oriented properties of the C++ language allow the application to be enriched with functions that were not foreseen by the creators.

The EmuLator program, together with the help system, is available on floppy disks in the installation version.

Table of contents