The "programs.sty" style file*

Miguel Alabau
LaBRI, Université Bordeaux I (France)
e-mail: Miguel.Alabau@labri.u-bordeaux.fr

June 13, 2005

Abstract

This style file contains a set of definitions that allow a fairly easy pretty-printing of programs. In particular, text alignement is obtained by simply typing `space` characters. Emphasized characters, mathematical symbols and commands are directly taken into account.

Contents

1 Introduction

The LaTeX `verbatim` environment allows for easy typesetting of text. However it is sometimes convenient to type programs that involve some mathematics, some emphasized text or some boldfaced keywords. LaTeX provides the `tabbing` environment for freely typesetting programs. But a cumbersome aspect of this environment is the way tabs are specified: their presence makes the text to be obscured. The file *programs.sty* provides different environments and commands for typesetting programs. Spaces are interpreted as in the `verbatim` environment, avoiding the user to type `\=` and `\>` control characters. Accents, mathematical symbols, emphasized and boldface fonts can be used. Another useful feature is the capability to number lines and to put labels on lines (and, of course, to refer to them).

For instance, you may type something like

```
\begin{programf}
function sqrt(x: integer): integer;
         (* sqrt(x)  = $\sqrt{x}$ *)
```

*This file has version number v1.0 dated 95/04/01. The documentation was last revised on 96/01/31.

```
function pow(x,y: real): real;
         (* pow(x,y) = $x^y$ *)
\end{programf}
```

which leads to the following output

```
1  function sqrt(x: integer): integer;
2           (* sqrt(x)  = √x *)
3  function pow(x,y: real): real;
4           (* pow(x,y) = x^y *)
5
```

It is also possible to typeset the same program in a smaller font, enclosed within two horizontal lines, and with the lines unnumbered.

```
\programsurround
\begin{programt}*
function sqrt(x: integer): integer;
         (* sqrt(x)  = $\sqrt{x}$ *)
function pow(x,y: real): real;
         (* pow(x,y) = $x^y$ *)
\end{programt}
```

yielding to the following output:

```
function sqrt(x: integer): integer;
        (* sqrt(x)  = √x *)
function pow(x,y: real): real;
        (* pow(x,y) = x^y *)
```

A set of other options is provided, together with two file inclusion capabilities.

2 User's Manual

In this section, we describe the environments and commands provided by this style file (section 2.1). We indicate also three sets of control commands:

- Global commands, i.e. global switches for the commands/environments of section 2.1 (section 2.2).
- Commands whose scope is the next program only (section 2.3).
- Commands that are used *within* a program environment (section 2.4).

The user is provided with two *meta-commands* that allow to define new program environments for some other fonts, or to redefine existing program environments.

This section terminates by indicating how to proceed for extracting the different archives from the file *programs.dtx*.

2.1 Environments for typesetting programs

The following environments are provided, every one of them corresponding to one of the LaTeX predefined font sizes:

environments	sizes
program	normalsize
programl	large
programL	Large
programs	small
programf	footnotesize
programsc	scriptsize
programt	tiny

These environments are to be used like the `verbatim` environment. However they work differently, since the usual LaTeX escapes are allowable from within the environment. For instance, math mode as well as emphasized characters may be used.

By default, lines are numbered. If someone wants to type an unnumbered text, it is necessary to put a * just after the beginning of the environment. For instance:

```
\begin{programL}*
```

```
    <unnumbered text>
\end{programL}
```

Program indentation obey to the variable `\ProgramIndent` (see section 2.2). However, it is possible, for one given environment, not to obey to the global indentation of programs. This is done by indicating another indentation between square braces just after entering the environment. For instance, an unnumbered program indented 2cm from the left margin of the text is:

```
\begin{programL}[2cm]*
    <unnumbered text>
\end{programL}
```

There is also a set of inclusion commands similar to the `\verbatimfile` (verbatim inclusion of a file) and `\verbatimlisting` (verbatim inclusion of a file, with numbered lines) commands of the "*verbatimfiles.sty*" by Chris Rowley. Of course, the files input by these commands are subject to the same permisive syntax as for the environments above (math syntax, emphasized text, *etc.*).

program inclusion commands		
unnumbered programs	numbered programs	sizes
\fprogram	\lprogram	normalsize
\fprograml	\lprograml	large
\fprogramL	\lProgramL	Large
\fprograms	\lprograms	small
\fprogramf	\lprogramf	footnotesize
\fprogramsc	\lprogramsc	scriptsize
\fprogramt	\lprogramt	tiny

We describe in section 2.5 how to define new program environments.

2.2 Global commands

`\ProgramIndent` This command serves to control the default indentation of the programs. It is used as described below:

```
\ProgramIndent{1cm}
```

and has the effect to make all the programs to be indented by default one centimeter from the left margin, unless this value is changed by another `\ProgramIndent` command. Default is no indentation at all.

`\programindent` This macro redefines the macro `\ProgramIndent`. It is present here for compatibility with previous versions of the *programs.sty* style.

`\LeftMarginNumberLine` `\RightMarginNumberLine` `\BothMarginsNumberLine` `\InBodyLeftNumberLine` These four commands are self-explanatory. They allow the user to specify that line numbers must be put in either the left or the right margin, or in both margins, or that lines must appear inside the body of the text on the left of the program. These options may be put anywhere in the text, in the preamble as well as in the body. The effect of one of these commands stands until it is changed by another one of them. Of course, different commands may be put in several parts of the text, if the user wants its programs to be numbered differently. The default is for the lines to appear in the left margin of the text (`\LeftMarginNumberLine`).

`\BothMarginNumberLine` This macro redefines the macro `\BothMarginsNumberLine`. It is present here for

compatibility with previous versions of the *programs.sty* style.

`\ttProgram` `\rmProgram` `\emProgram` Text of programs are usually typed with a teletype font (like in the `verbatim` environment). The user has the ability to change this default font to one of the three predefined fonts: teletype, roman, italicized roman.

`\ProgramDefaultFont` The command `\ProgramDefaultFont` serves to reset the printing to the default font.

2.3 Commands to be used before a program environment

`\ProgramSurround` Programs are usually typeset as they are. However a user can specify that the next program to be printed will be surrounded by two horizontal lines, as long as the width of the text. This is done by putting this command in the body of the text before the program appears.

`\programsurround` This macro redefines the macro `\ProgramSurround`. It is present here for compatibility with previous versions of the *programs.sty* style.

`\SetProgramCounter` By default, program lines are counted from 1. It is possible to change the value of the first line number of the next program by issuing the following command before the program is included:

```
\SetProgramCounter{6}
```

In this example, the lines of the next program will start from 6.

`\setprogramcounter` This macro redefines the macro `\SetProgramCounter`. It is present here for compatibility with previous versions of the *programs.sty* style.

`\NoResetProgramCounter` If the user desires that the number of the first line of the next program is equal to the number of the last line of the last previous program, he must issue the command `\NoResetProgramCounter` before the next program. This command has no effect if issued before the first program.

`\noresetprogramcounter` This macro redefines the macro `\NoResetProgramCounter`. It is present here for compatibility with previous versions of the *programs.sty* style.

2.4 Commands to be used inside a program environment

`\UnnumLine` This command is to be used only *within* programs. It must appear at the end of a line and has the effect not to number the following line. It serves when the user wants to keep only one unique line number for long statements that span accross several lines.

`\unnumline` This macro redefines the macro `\UnnumLine`. It is present here for compatibility with previous versions of the *programs.sty* style.

2.5 Meta-Commands: how to define new program environments

`\NewProgram` `\RenewProgram` The `\NewProgram` command serves to define a new program environment. The `\RenewProgram` command is to be used for redefining already defined program environments. These commands must be used as below:

`\NewProgram`{*name*}{*font_name*}
`\RenewProgram`{*name*}{*font_name*}

The command `\NewProgram` defines one environment and two commands. Let us assume that the user issues the following command:

```
\NewProgram{LittleProg}{smallsize}
```

then an environment called `LittleProg` will be generated for direct typesetting of programs, and two commands will be created: `fLittleProg` and `lLittleProg` for inclusion of unnumbered (resp. numbered) text.

`\newprogram` `\renewprogram` These two macros are old names present here for compatibility with previous versions of the *programs.sty* style. `\newprogram` redefines `\NewProgram`, and `\renewprogram` redefines `\RenewProgram`.

2.6 The Index File

In order for the processing of this file to be complete, an index format file is required. Let us assume that it is named `programs.ist`, then the following command must be run and then another compilation of the current file:

```
⟨index⟩
⟨index⟩%% ------------------------------------------------------------
⟨index⟩%% Assuming this file is named "programs.ist" (after being
⟨index⟩%% generated from "programs.dtx" by running "latex docstrip"),
⟨index⟩%% the following command will produce a well formated index:
⟨index⟩%%
⟨index⟩%%                  makeindex -s programs.ist programs.idx
⟨index⟩%% ------------------------------------------------------------
⟨index⟩
```

Another possibility is to set the environment variable `INDEXSTYLE` to a directory name where the ".ist" files (index format files) may be found.

A possible index file is given below[1]:

```
⟨index⟩actual '='
⟨index⟩quote '!'
⟨index⟩level '>'
⟨index⟩preamble
⟨index⟩"\n \\begin{theindex} \n \\makeatletter\\scan@allowedfalse\n"
⟨index⟩postamble
⟨index⟩"\n\n \\end{theindex}\n"
⟨index⟩item_x1   "\\efill \n \\subitem "
⟨index⟩item_x2   "\\efill \n \\subsubitem "
⟨index⟩delim_0   "\\pfill "
⟨index⟩delim_1   "\\pfill "
⟨index⟩delim_2   "\\pfill "
⟨index⟩% The next lines will produce some warnings when
⟨index⟩% running Makeindex as they try to cover two different
⟨index⟩% versions of the program:
⟨index⟩lethead_prefix   "{\\bf\\hfil "
⟨index⟩lethead_suffix   "\\hfil}\\nopagebreak\n"
⟨index⟩lethead_flag       1
⟨index⟩heading_prefix   "{\\bf\\hfil "
⟨index⟩heading_suffix   "\\hfil}\\nopagebreak\n"
⟨index⟩headings_flag       1
```

2.7 The Driver File

There is also a driver file, called *programs.drv*, that is included in the distribution. It is devoted to control the latex compilation of the documentation. Its code is given below.

```
⟨*driver⟩
```

[1] It can be generated by invoquing the compilation of "docstrip" with the "index" option.

```
\newif\ifnoprogsfile
\openin1 programs.sty
\ifeof1 \noprogsfiletrue\else\noprogsfilefalse\fi\closein1
\ifnoprogsfile
    \typeout{*****************************************************}
    \typeout{To get a more complete documentation, you should}
    \typeout{copy the current file into 'programs.sty'}
    \typeout{*****************************************************}
\fi
\ifnoprogsfile
    \documentclass{ltxdoc}
\else
    \documentclass{ltxdoc}
    \usepackage{programs}
\fi
\MakePercentIgnore%
%
\setlength{\textwidth}{31pc}%
\setlength{\textheight}{54pc}%
\setlength{\parindent}{0pt}%
\setlength{\parskip}{2pt plus 1pt minus 1pt}%
\setlength{\oddsidemargin}{8pc}%
\setlength{\marginparwidth}{8pc}%
\setlength{\topmargin}{-2.5pc}%
\setlength{\headsep}{20pt}%
\setlength{\columnsep}{1.5pc}%
\setlength{\columnwidth}{18.75pc}%
%%
\setcounter{IndexColumns}{2}%
\EnableCrossrefs%
\RecordChanges
\CodelineIndex
%\OldMakeindex     % use if your MakeIndex is pre-v2.9%
\begin{document}%
    \DocInput{programs.dtx}
\end{document}
```
⟨/driver⟩

2.8 Extracting the documents included in the file programs.dtx

There are three documents included in the *programs.dtx* file: the style file (*programs.sty*), the index style file for printing a cross-referenced document (*programs.ist*), and the driver file for printing the document: *programs.drv*.

For file extraction it is necessary to use the `docstrip` utility, which is part of the `doc` distribution [3]. Normally, a file `docstrip.tex` should exist on the LaTeX style files directory. Extraction is performed by typing:

```
latex docstrip
```

This is an interactive program, and the dialogue for generating the style file should be:

```
*************************************************************
* This program converts documented macro-files into fast *
* loadable files by stripping off (nearly) all comments! *
*************************************************************
```

```
**************************************************
* First type the extension of your input file(s):  *
\infileext=doc
**************************************************

**************************************************
* Now type the extension of your output file(s)  : *
\outfileext=sty
**************************************************

**************************************************
* Now type the name(s) of option(s) to include   : *
\Options=style
**************************************************

**************************************************
* Finally give the list of input file(s) without   *
* extension seperated by commas if necessary     : *
\filelist=programs
**************************************************
```

For generating the index file it suffices to rerun the `docstrip` utility and to answer "ist/index" instead of "sty/style" int the above steps 2 and 3, and in another run to answer "drv/driver".

The three files may be produced in a single pass, by simply latexing the file *programs.ins* which goes along with the file *programs.dtx*.

Generation of the documentation is then simply performed as follows:

```
latex programs.drv
latex programs.drv
latex programs.drv
makeindex  -s programs.ist programs.idx
latex programs.drv
```

69 ⟨∗style⟩

3 Description of Macros

`\AlreadyDefined@@Programs` This macro can be tested by any style file to know if the file "programs.sty" has been input. But it allows a modular programming style similar to the one used with the C header files. Hence, the first time the "programs.sty" style file is included all of its body will be included; the second time, the body will not be included.

```
70 \expandafter\ifx\csname AlreadyDefined@@Programs\endcsname\relax%
71 \expandafter\def\csname AlreadyDefined@@Programs\endcsname{}%
72 \else\endinput\fi
```

3.1 Controlling program indentation

`\ProgramIndent` `\@@programindent` `\@@programindent` is the amount of program indentation for the left margin of the text. Initially, it is set to `\z@` :

```
73 ⟨style⟩%% CONTROLLING PROGRAM INDENTATION
74 \newdimen\@@programindent
75 \@@programindent=\z@
```

The \ProgramIndent has the only effect to set the variable of \@@programindent to the value indicated by its parameter:

```
\def\ProgramIndent#1{\@@programindent=#1}
```

3.2 Surrounding programs by rules

\ProgramSurround
\if@@surround

By default, a program is printed as is, but it is possible to indicate that it is going to be enclosed within two \hrule:

```
⟨style⟩%% SURROUNDING PROGRAMS BY RULES
\newif\if@@surround\@@surroundfalse
\def\ProgramSurround{\@@surroundtrue}
```

\@@progline
\@@noprogline

These two macros define the shape of the surrounding lines. The definition of \@@progline is such that the surrounding lines lengths are always equal to the width of the current line (even if it is changed from one program to another).

```
\def\@@progline{\def\@@prgln{\rule{\linewidth}{0.1mm}}\@@prgln}
\def\@@noprogline{\rule{0pt}{0pt}}
```

3.3 Line numbering

\@@defaultindent

The purpose of this macro is to keep space enough for printing the line numbers of the programs. I have defined its length for make it easy printing long programs (thousands of lines).

```
⟨style⟩%% LINE NUMBERING
\newlength{\@@defaultindent}
\settowidth{\@@defaultindent}{{\tt{}12345}}
```

\if@@resetlineno
\if@@unnumline
\if@@CurrentProgIsUnnumbered

These three conditions serve to indicate the printing status of the current program. More precisely, \if@@resetlineno is a boolean flag to specify if line numbering must be reset for the next program. It defaults to true. \if@@unnumline is a boolean flag to specify that the next line to be printed is not to be numbered. It defaults to false (i.e. every line is numbered, by default). \if@@CurrentProgIsUnnumbered is a global flag for the program, that indicates if the program being printed is numbered or not. It defaults to false (i.e. programs are numbered, by default).

```
\newif\if@@resetlineno \@@resetlinenotrue \newif\if@@unnumline
\@@unnumlinefalse
\newif\if@@CurrentProgIsUnnumbered \@@CurrentProgIsUnnumberedfalse
```

\NoResetProgramCounter

This macro is provided to the user to specify that the first line number of the next program must be equal to the last line number of the previous program. More precisely, lines for the next program will be numbered from \@@lineno + 1.

```
\def\NoResetProgramCounter{\@@resetlinenofalse}
```

\UnnumLine

As said in section 2.4, this macro must appear at the end of a program line. Its effect is to set on the boolean flag \@@unnumlinetrue to prevent the macro \@@xnewprog from numbering the next line of the program. The "\ " that appears ahead of the macro serves to make the command valid even if issued on an empty line.

```
\def\UnnumLine{\ \@@unnumlinetrue}
```

@@lineno
\SetProgramCounter

This is the definition of a counter for the program lines. Once the macro \SetProgramCounter called, its effect is to make lines starting from the value indicated as param #1. Of course, if the user issues a \SetProgramCounter command,

it is implicitly assumed that he wants the lines to be numbered. That is why the condition `\if@@resetlineno` is set to false.

```
\newcounter{@@lineno}\setcounter{@@lineno}{1}
\def\SetProgramCounter#1{\setcounter{@@lineno}{#1}\@@resetlinenofalse}
```

`@@dummylineno` This little trick is an internal line counter for the unnumbered programs. It is necessary for making it possible to put labels on lines in unnumbered programs, and refer to them. Internal numbering of unnumbered programs always begins at 1.

```
\newcounter{@@dummylineno}\setcounter{@@dummylineno}{1}
```

`\LeftMarginNumberLine` `\RightMarginNumberLine` `\BothMarginsNumberLine` `\InBodyLeftNumberLine` `\@@PlaceOfNumbers` The first four commands are provided to the user for indicating line number placement. They have the only effect to change the value of `\@@PlaceOfNumbers` which is an internal value whose purpose is to define where the line numbers are to appear on the text. It is used by the macro `\@@xnewprog`.

```
\def\LeftMarginNumberLine{\let\@@PlaceOfNumbers\@@LeftMarginNumberLine}
\def\RightMarginNumberLine{\let\@@PlaceOfNumbers\@@RightMarginNumberLine}
\def\BothMarginsNumberLine{\let\@@PlaceOfNumbers\@@BothMarginsNumberLine}
\def\InBodyLeftNumberLine{\let\@@PlaceOfNumbers\@@InBodyLeftNumberLine}
\def\@@LeftMarginNumberLine{0} \def\@@RightMarginNumberLine{1}
\def\@@BothMarginsNumberLine{2}
\def\@@InBodyLeftNumberLine{3}
```

For more readability, a

```
\LeftMarginNumberLine
```

command is issued, in order to initialize `\@@PlaceOfNumbers`.

3.4 Program default fonts

`\@@DefaultProgramFont` Text of programs is usually typed with a teletype font (like in the `verbatim` environment). Default font printing is controlled by this counter. Its value is used in the macro `\@@astyped` described elsewhere in this document.

```
⟨style⟩%% PROGRAM DEFAULT FONTS
\def\@@DefaultProgramFont{0}
```

`\ttProgram` `\rmProgram` `\emProgram` `\ProgramDefaultFont` These commands allow the user to change the default font of the programs. This is performed by redefining the running macros `\@@astyped` and `\@@program`.

```
\def\ttProgram{\def\@@DefaultProgramFont{0}\def@@astyped\def@@program}
\def\rmProgram{\def\@@DefaultProgramFont{1}\def@@astyped\def@@program}
\def\emProgram{\def\@@DefaultProgramFont{2}\def@@astyped\def@@program}
\def\ProgramDefaultFont{\ttProgram}
```

3.5 The real environment

`\@@vobeyspaces` `\@@xobeysp` We first begin by redefining the space character that will be used in the `@@astyped` environment. It is important to let a space after the occurrence of `\let` below, since at this point space characters are become active. If `\@@xobeysp` had been issued on a different line, a risk would have existed to have space redefined to empty space.

```
⟨style⟩%% THE REAL ENVIRONMENT
{\catcode`\ =\active\gdef\@@vobeyspaces{\catcode`\ \active\let \@@xobeysp}}
\def\@@xobeysp{\leavevmode\penalty10000\ }
```

\def@@astyped
\@@astyped
\end@@astyped

Then, we define the `@@astyped` environment by the means of its two macros `\@@astyped` and `\end@@astyped`. It is very strongly related to the `astyped` environment [**?**]. However, rather than directly using the `astyped` environment, I have prefered to make the *programs.sty* style file independent.

`\def@@astyped` causes the `@@astyped` environment to be defined. This is because we want a different `@@astyped` environment to be defined for every new `program` environment, because fonts may have changed, hence spacing may differ from one environment to another one.

```
\def\def@@astyped{%
    \def\@@astyped{%
        \partopsep\z@%
        \topsep\z@%
        \trivlist \item[]%
            \leftskip\@totalleftmargin%
            \rightskip\z@%
            \parindent\z@%
            \parfillskip\@flushglue%
            \parskip\z@%
            \@tempswafalse%
            \def\par{\if@tempswa\hbox{}\fi\@tempswatrue\@@par}%
            \obeylines%
            \ifcase\@@DefaultProgramFont \tt\or \rm\or \em\else \tt\fi
            \catcode``=13 \@noligs%
            \let\do\@makeother \do\ \do\^^K\do\^^A%
            \frenchspacing\@@vobeyspaces%
            \noindent\hspace{\parindent}%
            \if@@surround\@@progline\else\@@noprogline\fi%
            \nopagebreak%
            }
    \def\end@@astyped{%
            \nopagebreak%
            \noindent\hspace{\parindent}%
            \if@@surround\@@progline\else\@@noprogline\fi%
        \endtrivlist%
        }
}
```

3.6 Meta-commands for defining new program environments

\NewProgram
\RenewProgram
\@@newprog

The command `\NewProgram` (resp. `\RenewProgram`) can be used to define (resp. redefine) new program environments. The first parameter is the name of a program environment to be created, and the second one is the name of a size for the police (e.g. *smallsize*, *tiny*, *etc.*). See section 2.5 for an example.

I have defined `\RenewProgram` same as `\NewProgram` because I am too lazzy, but it should test if the environment to be redefined has been previously defined.

```
⟨style⟩%% META-COMMANDS FOR DEFINING NEW PROGRAM ENVIRONMENTS
\def\NewProgram#1#2{\@@newprog{#1}{#2}}
\def\RenewProgram#1#2{\@@newprog{#1}{#2}}
\def\@@newprog#1#2{%
    \@namedef{#1}{%
        \begingroup\def\@@tempa{\@nameuse{#2}}%
        \def\@@tempb{\baselinestretch}\def\baselinestretch{1}%
        \@ifundefined{@@tempa}{\normalsize}{\@@tempa}%
        \def@@astyped\@@astyped%
        \@ifnextchar[{\@@xnewprog}{\@@xnewprog[\@@programindent]}%
    }%
```

```
    \@namedef{end#1}{%
        \everypar{}%
```

The little trick below is necessary because `\@@lineno` is incremented by 1 at the beginning of every `program` environment (see `\@@xnewprog` below). Hence, when `\NoResetProgramCounter` is used, the line numbers of the last line of the previous program and the first line of the new program would be the same. The condition below avoids this drawback.

```
        \if@@CurrentProgIsUnnumbered \relax%
        \else%
                \addtocounter{@@lineno}{1}%
        \fi%
        %
        \end@@astyped%
        \let\baselinestretch=\@@tempb\endgroup%
        \global\@@resetlinenotrue%
        \global\ProgramDefaultFont%
        \global\@@surroundfalse%
    }%
```

At last, if actual value of parameter `#1` is `FOO`, we define two file inclusion commands: `\fFOO` and `lFOO` for inclusion of unnumbered and numbered programs (see section 2.1).

```
    \@namedef{f#1}##1{\@nameuse{#1}*\par\input##1\@nameuse{end#1}}%
    \@namedef{l#1}##1{\@nameuse{#1}\par\input##1\@nameuse{end#1}}%
}
```

`\@@numlinelength` `\@@xnewprog`

The macro `\@@xnewprog` performs the printing of the lines.

```
\newlength{\@@numlinelength}
\def\@@xnewprog[#1]{%
```

If the first character is the symbol `*` then no line numbers are printed.

```
        \@ifstar{%
          \@@CurrentProgIsUnnumberedtrue
            \setcounter{@@dummylineno}{0}%
            \leavevmode%
            \everypar{%
                \refstepcounter{@@dummylineno}%
                \@@unnumlinefalse%
                \noindent\hspace{#1}}%
        }%
```

Otherwise, this is the normal case:

```
        {%
          \@@CurrentProgIsUnnumberedfalse
          \if@@resetlineno%
                  \setcounter{@@lineno}{0}%
          \else%
                  \addtocounter{@@lineno}{-1}%
          \fi%
          \leavevmode%
          \everypar{%
              \if@@unnumline%
```

I decided to make a default indentation on the left side of the unnumbered program if the user has requested a numbering on the left side of the page for the numbered programs. This is to keep an homogeneous layout.

```
                  \ifx \@@PlaceOfNumbers\@@InBodyLeftNumberLine%
                      \hspace{\@@defaultindent}%
```

```
                    \rule{0pt}{0pt}%
                \fi
```

Otherwise, for numbered programs, we begin by incrementing the line counter and making it possible a reference to the line number to be done (see the *latex.tex*[2] file for explanations on `\refstepcounter`).

```
            \else%
                \refstepcounter{@@lineno}%
```

Then, we look at the placement of the line numbers, which is controlled by the variable `\@@PlaceOfNumbers`:

```
                \ifx \@@PlaceOfNumbers\@@LeftMarginNumberLine%
                    \llap{{\rm\the@@lineno\ \ }}%
                \else \ifx \@@PlaceOfNumbers\@@RightMarginNumberLine%
                    \noindent\hspace{\columnwidth}%
                    \rlap{{\rm\ \ \the@@lineno}}%
                    \noindent\hspace{-\columnwidth}%
                \else \ifx \@@PlaceOfNumbers\@@BothMarginsNumberLine%
                    \noindent\hspace{\columnwidth}%
                    \rlap{{\rm\ \ \the@@lineno}}%
                    \noindent\hspace{-\columnwidth}%
                    \llap{{\rm\the@@lineno\ \ }}%
                \else \ifx \@@PlaceOfNumbers\@@InBodyLeftNumberLine%
                    \hspace{\@@defaultindent}%
                    \rule{0pt}{0pt}%
                    \llap{{\rm\the@@lineno\ \ }}%
                \else
```

Otherwise (default case), numbers are printed on the left margin of the page:

```
                     \llap{{\rm\the@@lineno\ \ }}%
                \fi\fi\fi\fi
```

Then we reset the boolean flag `\@@unnumlinefalse` in order to make the next line to be numbered (of course, this is useful only if the program is numbered), and we indent the program according to what was requested by the user.

```
            \fi\@@unnumlinefalse%
            \noindent\hspace{#1}%
        }%
    }%
}
```

3.7 Predefined environments and commands

`\def@@program` `\ProgramDefaultFont`

This command serves to define the environments and commands described in section 2.1. It is invoked by the `\ProgramDefaultFont` command.

```
⟨style⟩%% PREDEFINED ENVIRONMENTS AND COMMANDS
\def\def@@program{%
  \NewProgram{program}{normalsize}
  \NewProgram{programl}{large}
  \NewProgram{programL}{Large}
  \NewProgram{programs}{small}
  \NewProgram{programf}{footnotesize}
  \NewProgram{programsc}{scriptsize}
  \NewProgram{programt}{tiny}
}
```

[2] This file is part of the LaTeX distribution.

Then we terminate by instructing LaTeX to switch to the default font for typing programs (which, in the current implementation is `\tt` in order to have a behaviour consistent with the `verbatim` environment).

```
\ProgramDefaultFont
```

3.8 Old macro names present here for compatibility reasons

`\newprogram`
`\renewprogram`
`\noresetprogramcounter`
`\programindent`
`\programsurround`
`\setprogramcounter`
`\unnumline`
`\BothMarginNumberLine`

These macro names are simple redefinitions of macros defined elsewhere in this document style. They are present here because they had been defined in previous versions of this style.

```
⟨style⟩%% OLD MACRO NAMES PRESENT HERE FOR COMPATIBILITY REASONS
\let\newprogram=\NewProgram \let\renewprogram=\RenewProgram
\let\noresetprogramcounter=\NoResetProgramCounter
\let\programindent=\ProgramIndent
\let\programsurround=\ProgramSurround
\let\setprogramcounter=\SetProgramCounter \let\unnumline=\UnnumLine
\let\BothMarginNumberLine=\BothMarginsNumberLine

⟨/style⟩
```

References

[1] D.E. Knuth. Computers & Typesetting (The TeXbook). Addison-Wesley, Vol. A, 1986.

[2] L. Lamport. *LaTeX: a Document Preparation System*. Addison-Wesley Publishing Company, 1986.

[3] F. Mittelbach. The `doc`-option. *TUGboat*, Vol. 10(2), *pp.* 245–273, July 1989.

[4] F. Mittelbach, D. Duchier and J. Braams. `docstrip.dtx` . The file is part of the DOC package.