Free Pascal :
Users' manual

Users' manual for Free Pascal, version 0.99.10
1.2
February 1999

Michaël Van Canneyt
Florian Klämpfl

Contents

Chapter 1

Introduction

1.1 About this document

This is the user's manual for Free Pascal. It describes the installation and use of the Free Pascal compiler on the different supported platforms. It does not attempt to give an exhaustive list of all supported commands, nor a definition of the Pascal language. Look at the Reference guide for these things. For a description of the possibilities and the inner workings of the compiler, see the Programmer's guide. In the appendices of this document you will find lists of reserved words and compiler error messages (with descriptions).

This document describes the compiler as it is/functions at the time of writing. Since the compiler is under continuous development, some of the things described here may be outdated. In case of doubt, consult the README files, distributed with the compiler. The README files are, in case of conflict with this manual, authoritative.

1.2 About the compiler

Free Pascal is a 32-bit compiler for the i386 and m68k processors[1]. Currently, it supports 6 operating systems:

- DOS
- LINUX
- AMIGA (version 0.99.5 only)
- WINDOWSNT
- OS/2 (using the EMX package, so it also works on DOS/Windows)

and work is in progress to port it to other platforms (notably, FREEBSD).

Free Pascal is designed to be, as much as possible, source compatible with Turbo Pascal 7.0 and Delphi 4 (although this goal is not yet attained), but it also enhances these languages with elements like function overloading. And, unlike these ancestors, it supports multiple platforms.

[1] Work is being done on a port to ALPHA Architecture

It also differs from them in the sense that you cannot use compiled units from one system for the other.

Also, at the time of writing, there is no Integrated Development Environment (IDE) available for Free Pascal. This gap will, hopefully, be filled in the future.

Free Pascal consists of three parts :

1. The compiler program itself.
2. The Run-Time Library (RTL).
3. Utility programs and units.

Of these you only need the first two, in order to be able to use the compiler. In this document, we describe the use of the compiler. The RTL is described in the Reference guide.

1.3 Getting more information.

If the documentation doesn't give an answer to your questions, you can obtain more information on the Internet, on the following addresses:

- http://tfdec1.fys.kuleuven.ac.be/~michael/fpc/fpc.html is the main site. It contains also useful mail addresses and links to other places. It also contains the instructions for inscribing to the *mailing-list*.
- http://www.brain.uni-freiburg.de/~klaus/fpc/fpc.html is a mirror of the main Free Pascal information site.

Both places can be used to download the Free Pascal distribution, although you can probably find them on other places also.

Finally, if you think something should be added to this manual (entirely possible), please do not hesitate and contact me at michael@tfdec1.fys.kuleuven.ac.be .

Let's get on with something useful.

Chapter 2

Installing the compiler

2.1 Before Installation : Requirements

System requirements

The compiler needs at least the following hardware:

1. An I386 or higher processor. A coprocessor is not required, although it will slow down your program's performance if you do floating point calculations.
2. 2 Mb of free memory. Under DOS, if you use DPMI memory management, such as under Windows, you will need at least 16 Mb.
3. At least 500 Kb. free disk space.

Software requirements

Under DOS

The DOS distribution contains all the files you need to run the compiler and compile pascal programs.

Under Linux

Under LINUX you need to have the following programs installed :

1. GNU as, the GNU assembler.
2. GNU ld, the GNU linker.
3. Optionally (but highly recommended) : GNU make. For easy recompiling of the compiler and Run-Time Library, this is needed.

Other than that, Free Pascal should run on almost any I386 LINUX system.

2.2 Installing the compiler.

The installation of Free Pascal is easy, but is platform-dependent. We discuss the process for each platform separately.

Figure 2.1: The DOS install program screen.

Installing under DOS

Mandatory installation steps.

First, you must get the latest distribution files of Free Pascal. They come as zip files, which you must unzip first, or you can download the compiler as a series of separate files. This is especially useful if you have a slow connection, but it is also nice if you want to install only some pats of the compiler distribution. The distribution zip file contains an installation program INSTALL.EXE. You must run this program to install the compiler.

The screen of the installation program looks like figure 2.1.

The program allows you to select:

- What components you wish to install. e.g do you want the sources or not, do you want docs or not. Items that you didn't download when downloading as separate files, will not be enabled, i.e. you can't select them.
- Where you want to install (the default location is `C:\PP`).

In order to run Free Pascal from any directory on your system, you must extend your path variable to contain the `C:\PP\BIN` directory. Usually this is done in the AUTOEXEC.BAT file. It should look something like this :

```
SET PATH=%PATH%;C:\PP\BIN
```

(Again, assuming that you installed in the default location).

If you want to use the graphic drivers you must modify the environment variable `GO32`. Instructions for doing this can be found in the documentation of the Graph unit, at the `InitGraph` procedure.

Optional Installation: The coprocessor emulation

For people who have an older CPU type, without math coprocessor (i387) it is necessary to install a coprocessor emulation, since Free Pascal uses the coprocessor

to do all floating point operations.

The installation of the coprocessor emulation is handled by the installation program (INSTALL.EXE). However,

Installing under Linux

Mandatory installation steps.

The LINUX distribution of Free Pascal comes in three forms:

- a tar.gz version, also available as seperate files.
- a .rpm (Red Hat Package Manager) version, and
- a .deb (debian) version.

All of these packages contain a `ELF` version of the compiler binaries and units. the older `aout` binaries are no longer distributed, although you still can use the comiler on an `aout` system if you recompile it.

If you use the .rpm format, installation is limited to

```
rpm -i fpc-pascal-XXX.rpm
```

(`XXX` is the version number of the .rpm file)

If you use debian, installation is limited to

```
dpkg -i fpc-XXX.deb
```

Here again, `XXX` is the version number of the .deb file.

You need root access to install these packages. The .tar file allows you to do an installation if you don't have root permissions.

When downloading the `.tar` file, or the separate files, installation is more interactive.

In case you downloaded the .tar file, you should first untar the file, in some directory where you have write permission, using the following command:

```
tar -xvf fpc.tar
```

We supposed here that you downloaded the file fpc.tar somewhere from the Internet. (The real filename will have some version number in it, which we omit here for clarity.)

When the file is untarred, you will be left with more archive files, and an install program: an installation shell script.

If you downloaded the files as separate files, you should at least download the install.sh script, and the libraries (in libs.tar.gz).

To install Free Pascal, all that you need to do now is give the following command:

```
./install.sh
```

And then you must answer some questions. They're very simple, they're mainly concerned with 2 things :

1. Places where you can install different things.

2. Deciding if you want to install certain components (such as sources and demo programs).

The script will automatically detect which components are present and can be installed. It will only offer to install what has been found. because of this feature, you must keep the original names when downloading, since the script expects this.

If you run the installation script as the `root` user, you can just accept all installation defaults. If you don't run as `root`, you must take care to supply the installation program with directory names where you have write permission, as it will attempt to create the directories you specify. In principle, you can install it wherever you want, though.

At the end of installation, the installation program will generate a configuration file for the Free Pascal compiler which reflects the settings that you chose. It will install this file in the /etc directory, (if you are not installing as `root`, this will fail), and in the directory where you installed the libraries.

If you want the Free Pascal compiler to use this configuration file, it must be present in /etc, or you can set the environment variable `PPC_CONFIG_PATH`. Under csh, you can do this by adding a

```
setenv PPC_CONFIG_PATH /usr/lib/ppc/0.99.1
```

line to your .login file in your home directory. (see also the next section)

2.3 Optional configuration steps

You may wish to set some environment variables. The Free Pascal compiler recognizes the following variables :

- `PPC_EXEC_PATH` contains the directory where 'as' and 'ld' are. (default /usr/bin)
- `PPC_GCCLIB_PATH` contains the directory where libgcc.a is (no default). This if for LINUX only.
- `PPC_CONFIG_PATH` specifies an alternate path to find ppc386.cfg (default under LINUX is /etc)
- `PPC_ERROR_FILE` specifies the path and name of the error-definition file. (default /usr/lib/fpc/errorE.msg)

These locations are, however, set in the sample configuration file which is built at the end of the installation process, except for the `PPC_CONFIG_PATH` variable, which you must set if you didn't install things in the default places.

finally

Also distributed in Free Pascal is a README file. It contains the latest instructions for installing Free Pascal, and should always be read first.

2.4 Testing the compiler

After the installation is completed and the environment variables are set as described above, your first program can be compiled.

Included in the Free Pascal distribution are some demonstration programs, showing what the compiler can do. You can test if the compiler functions correctly by trying to compile these programs.

The compiler is called

- PPC386.EXE under DOS, and
- ppc386 under LINUX

To compile a program (e.g `demo\hello.pp`) simply type :

```
ppc386 hello
```

at the command prompt. If you don't have a configuration file, then you may need to tell the compiler where it can find the units, for instance as follows:

```
ppc386 -Upc:\pp\rtl\dos\go32v2 hello
```

under DOS, and under LINUX you could type

```
ppc386 -Up/usr/lib/fpc/0.99.7/linuxunits hello
```

This is, of course, assuming that you installed under `C:\PP` or /usr/lib/fpc/0.99.7, respectively.

If you got no error messages, the compiler has generated an executable called hello (no extension) under LINUX, and a file hello.exe under DOS.

To execute the program, simply type :

```
hello
```

If all went well, you should see the following friendly greeting:

```
Hello world
```

In the DOS case, this friendly greeting may be preceded by some ugly message from the GO32 extender program. This unfriendly behavior can be switched off by setting the GO32 environment variable.

Chapter 3

Compiler usage

Here we describe the essentials to compile a program and a unit. We also describe how to make a stand-alone executable of the compiled program under DOS. For more advanced uses of the compiler, see the section on configuring the compiler, and the Programmer's guide.

The examples in this section suppose that you have a ppc386.cfg which is set up correctly, and which contains at least the path setting for the RTL units. In principle this file is generated by the installation program. You may have to check that it is in the correct place (see section 5.2 for more information on this).

3.1 File searching

Before you start compiling a program or a series of units, it is important to know where the compiler looks for its source files and other files. In this section we discuss this, and we indicate how to influence this.

Remark: The use of slashes (/) and backslashes (\) as directory separators is irrelevant, the compiler will convert to whatever character is used on the current operating system. Examples will be given using slashes, since this avoids problems on LINUX.

Command line files

The file that you specify on the command line, such as in

```
ppc386 foo.pp
```

will be looked for ONLY in the current directory. If you specify a directory in the filename, then the compiler will look in that directory:

```
ppc386 subdir/foo.pp
```

will look for foo.pp in the subdirectory subdir of the current directory.

Under LINUX, the name of this file is case sensitive, under other operating systems (DOS, WINDOWSNT, OS/2) this is not the case.

Unit files

When you compile a unit or program that needs other units, the compiler will look for compiled versions of these units in the following way:

1. It will look in the current directory.
2. It will look in the directory where the compiler binary is. (not under LINUX)
3. It will look in all the directories specified in the unit search path.

You can add a directory to the unit search path with the `-Up` or `-Fu` options (See 5.1, See 5.1). Every occurrence of one of those options will append a directory to the unit search path.

On LINUX, the compiler will first convert the filename of a unit to all-lowercase. This is necessary, since Pascal is case-independent, and the statements `Uses Unit1;` or `uses unit1;` should have the same effect. Also, unit names that are longer than 8 characters will first be looked for with their full length. If the unit is not found with this name, the name will be truncated to 8 characters, and the compiler will look again in the same directories, but with the truncated name.

For instance, suppose that the file foo.pp needs the unit bar. Then the command

```
ppc386 -Up.. -Upunits foo.pp
```

will tell the compiler to look for the unit bar in the following places:

1. In the current directory.
2. In the directory where the compile binary is (not under LINUX).
3. In the parent directory of the current directory.
4. In the subdirectory units of the current directory

If the compiler finds the unit it needs, it will look for the source file of this unit in the same directory where it found the unit. If it finds the source of the unit, then it will compare the file times. If the source file was modified more recent than the unit file, the compiler will attempt to recompile the unit with this source file.

If the compiler doesn't find a compiled version of the unit, or when the `-B` option is specified, then the compiler will look in the same manner for the unit source file, and attempt to recompile it.

It is recommended to set the unit search path in the configuration file ppc386.cfg. If you do this, you don't need to specify the unit search path on the command-line every time you want to compile something.

3.2 Include files

If you include files in your source with the {`$I filename`} directive, the compiler will look for it in the following places:

1. It will look in the path specified in the incude file name.
2. It will look in the directory where the current source file is.

3. it will look in all directories specified in the include file search path.

You can add files to the include file search path with the `-I` (See 5.1) option.
As an example, consider the following include statement in a file units/foo.pp:

```
{$i ../bar.inc}
```

Then the following command :

```
ppc386 -Iincfiles units/foo.pp
```

will cause the compiler to look in the following directories for bar.inc:

1. the parent directory of the current directory
2. the units subdirectory of the current directory
3. the incfiles directory of the current directory.

3.3 Object files

When you link to object files (using the {`$L file.o`} directive, the compiler will look for this file in the same way as it looks for include files:

1. It will look in the path specified in the object file name.
2. It will look in the directory where the current source file is.
3. it will look in all directories specified in the object file search path.

You can add files to the object file search path with the `-Fo` (See 5.1) option.

Configuration file

Unless you specify the `-n` (See 5.1) option, the compiler will look for a configuration file ppc386.cfg in the following places:

- Under LINUX
  1. The current directory.
  2. In your home directory, it looks for .ppc386.cfg.
  3. The directory specified in the environment variable `PPC_CONFIG_PATH`, and if it's not set under /etc.
- Under all other OSes:
  1. The current directory.
  2. If it is set, the directory specified in the environment variable. `PPC_CONFIG_PATH`.
  3. The directory where the compiler is.

3.4 Compiling a program

Compiling a program is very simple. Assuming that you have a program source in the file prog.pp, you can compile this with the following command:

```
ppc386 [options] prog.pp
```

The square brackets `[ ]` indicate that what is between them is optional.

If your program file has the .pp or .pas extension, you can omit this on the command line, e.g. in the previous example you could have typed:

```
ppc386 [options] prog
```

If all went well, the compiler will produce an executable, or, for version 1 of the DOS extender, a file which can be converted to an executable.

Unless you are using DOS and version 1 of the DOS extender, the file you obtained is the executable. You can execute it straight away, you don't need to do anything else. Under version 1 of the DOS extender, additional processing is required. See section 3.6 on how to create an executable in this case.

You will notice that there is also another file in your directory, with extensions .o. This contains the object file for your program. If you compiled a program, you can delete the object file (.o), but not if you compiled a unit. Then the object file contains the code of the unit, and will be linked in any program that uses the unit you compiled, so you shouldn't remove it.

3.5 Compiling a unit

Compiling a unit is not essentially different from compiling a program. The difference is mainly that the linker isn't called in this case.

To compile a unit in the file foo.pp, just type :

```
ppc386  foo
```

Recall the remark about file extensions in the previous section.

When all went well, you will be left with 2 (two) unit files:

1. foo.ppu This is the file describing the unit you just compiled.
2. foo.o This file contains the actual code of the unit. This file will eventually end up in the executables.

Both files are needed if you plan to use the unit for some programs. So don't delete them. If you want to distribute the unit, you must provide both the .ppu and .o file. One is useless without the other.

Remark: Under LINUX, a unit source file *must* have a lowercase filename. Since Pascal is case independent, you can specify the names of units in the `uses` clause in either case. To get a unique filename, the Free Pascal compiler changes the name of the unit to all lowercase when looking for unit files.

The compiler produces lowercase files, so your unit will be found, even if your source file has uppercase letters in it. Only when the compiler tries to recompile the unit, it will not find your source because of the uppercase letters.

3.6 Creating an executable for GO32V1 and PMODE/DJ targets

The GO32V1 platform is officially no longer supported, so this section is of interest only to people who wish to make go32V1 binaries anyway.

GO32V1

When compiling under DOS, GO32V2 is the default target. However, if you use go32V1 (using the `-TGO32V1` switch), the compilation process leaves you with a file which you cannot execute right away. There are 2 things you can do when compiling has finished.

The first thing is to use the DOS extender from D.J. Delorie to execute your program :

```
go32 prog
```

This is fine for testing, but if you want to use a program regularly, it would be easier if you could just type the program name, i.e.

```
prog
```

This can be accomplished by making a DOS executable of your compiled program.

There two ways to create a DOS executable (under DOS only):

1. if the GO32.EXE is already installed on the computers where the program should run, you must only copy a program called STUB.EXE at the begin of the AOUT file. This is accomplished with the AOUT2EXE.EXE program. which comes with the compiler:

   ```
   AOUT2EXE PROG
   ```

   and you get a DOS executable which loads the GO32.EXE automatically. the GO32.EXE executable must be in current directory or be in a directory in the `PATH` variable.

2. The second way to create a DOS executable is to put GO32.EXE at the beginning of the AOUT file. To do this, at the command prompt, type :

   ```
   COPY /B GO32.EXE+PROG PROG.EXE
   ```

   (assuming Free Pascal created a file called PROG, of course.) This becomes then a stand-alone executable for DOS, which doesn't need the GO32.EXE on the machine where it should run.

PMODE/DJ

You can also use the PMODE/DJ extender to run your Free Pascal applications. To make an executable which works with the PMODE extender, you can simply create an GO32V2 executable (the default), and then convert it to a PMODE executable with the following two extra commands:

1. First, strip the GO32V2 header of the executable:

   ```
   EXE2COFF PROG.EXE
   ```

   (we suppose that PROG.EXE is the program generated by the compilation process.

2. Secondly, add the PMODE stub:

   ```
   COPY /B PMODSTUB.EXE+PROG PROG.EXE
   ```

   If the PMODSTUB.EXE file isn't in your local directory, you need to supply the whole path to it.

That's it. No additional steps are needed to create a PMODE extender executable.

Be aware, though, that the PMODE extender doesn't support virtual memory, so if you're short on memory, you may run unto trouble. Also, officially there is not support for the PMODE/DJ extender. It just happens that the compiler and some of the programs it generates, run under this extender too.

3.7 Reducing the size of your program

When you created your program, it is possible to reduce its size. This is possible, because the compiler leaves a lot of information in the program which, strictly speaking, isn't required for the execution of it. The surplus of information can be removed with a small program called strip. It comes with the GO32 development environment under DOS, and is standard on LINUX machines where you can do development. The usage is simple. Just type

```
strip prog
```

On the command line, and the strip program will remove all unnecessary information from your program. This can lead to size reductions of up to 30 %.

remark: in the WIN32 *32 version, strip is called stripw*

You can use the `-Xs` switch to let the compiler do this stripping automatically at program compile time (the switch has no effect when compiling units).

Another technique to reduce the size of a program is to use smartlinking. Normally, units (including the system unit) are linked in as a whole. It is however possible to compile units such that the can be smartlinked. This means that only the functions and procedures are linked in your program, leaving out any unnecessary code. This technique is described in full in the programmers guide.

Chapter 4

Compiling problems

4.1 General problems

- **IO-error -2 at ...** : Under LINUX you can get this message at compiler startup. It means typically that the compiler doesn't find the error definitions file. You can correct this mistake with the `-Fr` option under LINUX. (See 5.1)
- **Error : File not found : xxx** or **Error: couldn't compile unit xxx**: This typically happens when your unit path isn't set correctly. Remember that the compiler looks for units only in the current directory, and in the directory where the compiler itself is. If you want it to look somewhere else too, you must explicitly tell it to do so using the `-Up` option (See 5.1). Or you must set op a configuration file.

4.2 Problems you may encounter under DOS

- **No space in environment**.
  An error message like this can occur, if you call `SET_PP.BAT` in the AUTOEXEC.BAT.
  To solve this problem, you must extend your environment memory. To do this, search a line in the CONFIG.SYS like

  ```
  SHELL=C:\DOS\COMMAND.COM
  ```

  and change it to the following:

  ```
  SHELL=C:\DOS\COMMAND.COM /E:1024
  ```

  You may just need to specify a higher value, if this parameter is already set.
- **Coprocessor missing**
  If the compiler writes a message that there is no coprocessor, install the coprocessor emulation.
- **Not enough DPMI memory**
  If you want to use the compiler with `DPMI` you must have at least 7-8 MB free `DPMI` memory, but 16 Mb is a more realistic amount.

Chapter 5

Compiler configuration

The output of the compiler can be controlled in many ways. This can be done essentially in two distinct ways:

- Using command-line options.
- Using the configuration file: ppc386.cfg.

The compiler first reads the configuration file. Only then the command line options are checked. This creates the possibility to set some basic options in the configuration file, and at the same time you can still set some specific options when compiling some unit or program. First we list the command line options, and then we explain how to specify the command line options in the configuration file. When reading this, keep in mind that the options are case sensitive. While this is customary for LINUX, it isn't under DOS.

5.1 Using the command-line options

The available options for version 0.99.10 of the compiler are listed by category (see appendix A for a listing as generated by the compiler):

General options

-h if you specify this option, the compiler outputs a list of all options, and exits after that.

-? idem as `-h`, waiting after every screenfull for the enter key.

-i This option tells the compiler to print the copyright information. You can give it an option, as `-ixxx` where xxx can be one of the following:

D : Returns the compiler date.

V : Returns the compiler version.

SO : Returns the compiler OS.

SP : Returns the compiler processor.

TO : Returns the target OS.

TP : Returns the target Processor.

-l This option tells the compiler to print the Free Pascal logo on standard output. It also gives you the Free Pascal version number.

-n Tells the compiler not to read the configuration file.

Options for getting feedback

-vxxx Be verbose. `xxx` is a combination of the following :

- `e` : Tells the compiler to show only errors. This option is on by default.
- `i` : Tells the compiler to show some general information.
- `w` : Tells the compiler to issue warnings.
- `n` : Tells the compiler to issue notes.
- `h` : Tells the compiler to issue hints.
- `l` : Tells the compiler to show the line numbers as it processes a file. Numbers are shown per 100.
- `u` : Tells the compiler to print the names of the files it opens.
- `t` : Tells the compiler to print the names of the files it tries to open.
- `p` : Tells the compiler to print the names of procedures and functions as it is processing them.
- `c` : Tells the compiler to warn you when it processes a conditional.
- `m` : Tells the compiler to write which macros are defined.
- `d` : Tells the compiler to write other debugging info.
- `a` : Tells the compiler to write all possible info. (this is the same as specifying all options)
- `0` : Tells the compiler to write no messages. This is useful when you want to override the default setting in the configuration file.
- `b` : Tells the compiler to show all procedure declarations if an overloaded function error occurs.
- `x` : Tells the compiler to output some executable info (for Win32 platform only).
- `r` : Rhide/GCC compatibility mode: formats the errors differently, so they are understood by RHIDE.

Options concerning files and directories

-exxx `xxx` specifies the directory where the compiler can find the executables `as` (the assembler) and `ld` (the compiler).

-FD same as `-e`.

-Fexxx This option tells the compiler to write errors, etc. to the file in `xxx`.

-Fgxxx (LINUX only, obsolete) `xxx` specifies the path where the compiler can find the GNU C library. This is superseded by the `-Fl` option.

-Fixxx adds `xxx` to the path where the compiler searches for its include files.

-Flxxx Adds `xxx` to the library searching path, and is passed to the linker.

-FLxxx (LINUX only) Tells the compiler to use xxx as the dynamic linker. Default this is /lib/ld-linux.so.2, or lib/ld-linux.so.1, depending on which one is found.

-Foxxx Adds xxx to the object file path. This path is used when looking for files that need to be linked in.

-Frxxx xxx specifies the file which contain the compiler messages. Default the compiler ahs built-in messages. Specifying this option will override the default messages.

-Fuxxx Idem as `-Up`: Add xxx to the object path.

-FUxxx Tells the compiler to write units in directory `xxx` instead of the current directory.

-Ixxx Add xxx to the include file search path. This path is used when looking for include files.

-P uses pipes instead of files when assembling. This may speed up the compiler on OS/2 and LINUX. Only with assemblers (such as GNU as) that support piping..

-Upxxx Tells the compiler to add xxx to the path where to find units.
By default, the compiler only searches for units in the current directory and the directory where the compiler itself resides. This option tells the compiler also to look in the directory xxx.

Options controlling the kind of output.

for more information on these options, see also Programmer's guide

-a Tells the compiler not to delete the assembler file. This also counts for the (possibly) generated batch script.

-al Tells the compiler to include the sourcecode lines in the assembler file as comments. This feature is still experimental, and should be used with caution.

-Axxx specifies what kind of assembler should be generated . Here `xxx` is one of the following :

- **o** : A unix .o (object) file, using GNU as.
- **nasmcoff** : a coff file using the nasm assembler.
- **nasmelf** : a ELF32 file (LINUX only) using the nasm assembler.
- **nasmonj** : a obj file using the nasm assembler.
- **masm** : An obj file using the Microsoft masm assembler.
- **tasm** : An obj file using the Borland tasm assembler.

-CD Create dynamic library.

-Chxxx Reserves `xxx` bytes heap. `xxx` should be between 1024 and 67107840.

-Ci Generate Input/Output checking code.

-Cn Omit the linking stage.

-Co Generate Integer overflow checking code.

-Cr Generate Range checking code.

-Csxxx Set stack size to `xxx`.

-CS Create static library.

-Ct generate stack checking code.

-Cx Use smartlinking when compiling and linking units.

-dxxx Define the symbol name `xxx`. This can be used to conditionally compile parts of your code.

-E Same as `-Cn`.

-g Generate debugging information for debugging with `gdb`

-gg idem as `-g`.

-gd generate debugging info for `dbx`.

-gh use the heaptrc unit (see Unit reference).

-Oxxx optimize the compiler's output; `xxx` can have one of the following values :

g optimize for size, try to generate smaller code.

G optimize for time, try to generate faster code (default).

r keep certain variables in registers (experimental, use with caution).

u uncertain optimizations

1 Level 1 optimizations (quick optimizations).

2 Level 2 optimizations (`-O1` plus some slower optimizations).

3 Level 3 optimizations (`-O2` plus `-Ou`).

Pn Specify processor: `n` can be one of

1 optimize for 386/486

2 optimize for Pentium/PentiumMMX (tm)

3 optimizations for PentiumPro/PII/Cyrix 6x86/K6 (tm)

The exact effect of these effects can be found in the Programmer's guide.

-oxxx Tells the compiler to use `xxx` as the name of the output file (executable). Only with programs.

-pg Generate profiler code for `gprof`.

-s Tells the compiler not to call the assembler and linker. Instead, the compiler writes a script, `PPAS.BAT` under DOS, or `ppas.sh` under LINUX, which can then be executed to produce an executable.

-Txxx Specifies the target operating system. `xxx` can be one of the following:

- **GO32V1** : DOS and version 1 of the DJ DELORIE extender (no longer maintained).
- **GO32V2** : DOS and version 2 of the DJ DELORIE extender.
- **LINUX** : LINUX.
- **OS2** : OS/2 (2.x) (this is still under development).
- **WIN32** : Windows 32 bit.

-uxxx undefine the symbol `xxx`. This is the opposite of the `-d` option.

-uxxx Undefine symbol `xxx`.

-Xx executable options. This tells the compiler what kind of executable should be generated. the parameter `x` can be one of the following:

- **c** : (LINUX only) Link with the C library. You should only use this when you start to port Free Pascal to another operating system.
- **D** : Link with dynamic libraries (defines the `FPC_LINK_DYNAMIC` symbol)
- **s** : Strip the symbols from the executable.
- **S** : Link with static libraries (defines the `FPC_LINK_STATIC` symbol)

Options concerning the sources (language options)

for more information on these options, see also Programmer's guide

-Rxxx Specifies what assembler you use in your `asm` assembler code blocks. Here `xxx` is one of the following:

att `asm` blocks contain AT&T assembler.

intel `asm` blocks contain Intel assembler.

direct `asm` blocks should be copied as-is in the assembler file.

-S2 Switch on Delphi 2 extensions.

-Sc Support C-style operators, i.e. `*=, +=, /= and -=`.

-Sd tells the compiler to dispose asmlists. This uses less memory, but is slower.

-Se The compiler stops after the first error. Normally, the compiler tries to continue compiling after an error, until 50 errors are reached, or a fatal error is reachd, and then it stops. With this switch, the compiler will stop after the first error.

-Sg Support the `label` and `goto` commands.

-Si Support `C++` style INLINE.

-Sm Support C-style macros.

-So Try to be Borland TP 7.0 compatible (no function overloading etc.).

-Sp Try to be `gpc` (GNU pascal compiler) compatible.

-Ss The name of constructors must be `init`, and the name of destructors should be `done`.

-St Allow the `static` keyword in objects.

-Un Do not check the unit name. Normally, the unit name is the same as the filename. This option allows both to be different.

-Us Compile a system unit. This option causes the compiler to define only some very basic types.

5.2 Using the configuration file

Using the configuration file ppc386.cfg is an alternative to command line options. When a configuration file is found, it is read, and the lines in it are treated like you typed them on the command line. They are treated before the options that you type on the command line.

You can specify comments in the configuration file with the # sign. Everything from the # on will be ignored.

The compiler looks for the ppc386.cfg file in the following places :

- Under LINUX
  1. The current directory.
  2. In your home directory, it looks for .ppc386.cfg.
  3. The directory specified in the environment variable PPC_CONFIG_PATH, and if it's not set under /etc.
- Under all other OSes:
  1. The current directory.
  2. If it is set, the directory specified in the environment variable. PPC_CONFIG_PATH.
  3. The directory where the compiler is.

When the compiler has finished reading the configuration file, it continues to treat the command line options.

One of the command-line options allows you to specify a second configuration file: Specifying @foo on the command line will open file foo, and read further options from there. When the compiler has finished reading this file, it continues to process the command line.

The configuration file allows some kind of preprocessing. It understands the following directives, which you should place on the first column of a line :

#IFDEF

#IFNDEF

#ELSE

#ENDIF

#DEFINE

#UNDEF

#WRITE

#INCLUDE

#SECTION

They work the same way as their {$...} counterparts in Pascal.

What follows is a description of the different directives.

#IFDEF

Syntax:

```
#IFDEF name
```

Lines following `#IFDEF` are skipped read if the keyword `name` following it is not defined.

They are read until the keywords `#ELSE` or `#ENDIF` are encountered, after which normal processing is resumed.

Example :

```
#IFDEF VER0_99_5
-Up/usr/lib/fpc/0.99.5/linuxunits
#ENDIF
```

In the above example, /usr/lib/fpc/0.99.5/linuxunits will be added to the path if you're compiling with version 0.99.5 of the compiler.

#IFNDEF

Syntax:

```
#IFNDEF name
```

Lines following `#IFDEF` are skipped read if the keyword `name` following it is defined.

They are read until the keywords `#ELSE` or `#ENDIF` are encountered, after which normal processing is resumed.

Example :

```
#IFNDEF VER0_99_5
-Up/usr/lib/fpc/0.99.6/linuxunits
#ENDIF
```

In the above example, /usr/lib/fpc/0.99.6/linuxunits will be added to the path if you're NOT compiling with version 0.99.5 of the compiler.

#ELSE

Syntax:

```
#ELSE
```

`#ELSE` can be specified after a `#IFDEF` or `#IFNDEF` directive as an alternative. Lines following `#ELSE` are skipped read if the preceding `#IFDEF` `#IFNDEF` was accepted.

They are skipped until the keyword `#ENDIF` is encountered, after which normal processing is resumed.

Example :

```
#IFDEF VER0_99_5
-Up/usr/lib/fpc/0.99.6/linuxunits
```

```
#ELSE
-Up/usr/lib/fpc/0.99.5/linuxunits
#ENDIF
```

In the above example, /usr/lib/fpc/0.99.5/linuxunits will be added to the path if you're compiling with version 0.99.5 of the compiler, otherwise /usr/lib/fpc/0.99.6/linuxunits will be added to the path.

#ENDIF

Syntax:

```
#ENDIF
```

`#ENDIF` marks the end of a block that started with `#IF(N)DEF`, possibly with an `#ELSE` between it.

#DEFINE

Syntax:

```
#DEFINE name
```

`#DEFINE` defines a new keyword. This has the same effect as a `-dname` command-line option.

#UNDEF

Syntax:

```
#UNDEF name
```

`#UNDEF` un-defines a keyword if it existed. This has the same effect as a `-uname` command-line option.

#WRITE

Syntax:

```
#WRITE Message Text
```

`#WRITE` writes `Message Text` to the screen. This can be useful to display warnings if certain options are set.

Example:

```
#IFDEF DEBUG
#WRITE Setting debugging ON...
-g
#ENDIF
```

if `DEBUG` is defined, this will produce a line

```
Setting debugging ON...
```

and will then switch on debugging information in the compiler.

#INCLUDE

Syntax:

```
#INCLUDE filename
```

`#INCLUDE` instructs the compiler to read the contents of filename before continuing to process options in the current file.

This can be useful if you want to have a particular configuration file for a project (or, under LINUX, in your home directory), but still want to have the global options that are set in a global configuration file.

Example:

```
#IFDEF LINUX
  #INCLUDE /etc/ppc386.cfg
#ELSE
  #IFDEF GO32V2
    #INCLUDE c:\pp\bin\ppc386.cfg
  #ENDIF
#ENDIF
```

This will include /etc/ppc386.cfg if you're on a linux machine, and will include `c:\pp\bin\ppc386.cfg` on a dos machine.

#SECTION

Syntax:

```
#SECTION name
```

The `#SECTION` directive acts as a `#IFDEF` directive, only it doesn't require an `#ENDIF` directive. the special name `COMMON` always exists, i.e. lines following `#SECTION COMMON` are always read.

Chapter 6

Porting Turbo Pascal Code

Free Pascal was designed to resemble Turbo Pascal as closely as possible. There are, of course, restrictions. Some of these are due to the fact that Free Pascal is a 32-bit compiler. Other restrictions result from the fact that Free Pascal works on more than one operating system.

In general we can say that if you keep your program code close to ANSI Pascal, you will have no problems porting from Turbo Pascal, or even Delphi, to Free Pascal. To a large extent, the constructs defined by Turbo Pascal are supported. This is even more so if you use the `-So` or `-S2` switches.

In the following sections we will list the Turbo Pascal constructs which are not supported in Free Pascal, and we will list in what ways Free Pascal extends the Turbo Pascal language.

6.1 Things that will not work

Here we give a list of things which are defined/allowed in Turbo Pascal, but which are not supported by Free Pascal. Where possible, we indicate the reason.

1. Parameter lists of previously defined functions and procedures must match exactly. The reason for this is the function overloading mechanism of Free Pascal. (however, the `-So` switch solves this. See 5.1)
2. `(* ... *)` as comment delimiters are not allowed in versions older than 0.9.1. This can easily be remedied with a grown-up editor.
3. The `MEM, MEMW, MEML` and `PORT` variables for memory and port access are not available in the system unit. This is due to the operating system. Under DOS, the extender unit (GO32.PPU) implements the mem constuct. under LINUX, the ports unit implements such a construct.
4. `PROTECTED, PUBLIC, PUBLISHED, TRY, FINALLY, EXCEPT, RAISE` are reserved words. This means you cannot create procedures or variables with the same name. While they are not reserved words in Turbo Pascal, they are in Delphi. Using the `-So` switch will solve this problem if you want to compile Turbo Pascal code that uses these words.
5. The reserved words `FAR, NEAR` are ignored. This is because Free Pascal is a 32 bit compiler, so they're obsolete.

6. `INTERRUPT` only will work on a DOS machine.
7. Boolean expressions are only evaluated until their result is completely determined. The rest of the expression will be ignored.
8. By default the compiler uses `AT&T` assembler syntax. This is mainly because Free Pascal uses GNU `as`. However other assembler forms are available, Programmer's guide.
9. Turbo Vision is not available.
10. The 'overlay' unit is not available. It also isn't necessary, since Free Pascal is a 32 bit compiler, so program size shouldn't be a point.
11. There are more reserved words. (see appendix B for a list of all reserved words.)
12. The command-line parameters of the compiler are different.
13. Compiler switches and directives are mostly the same, but some extra exist.
14. Units are not binary compatible.

6.2 Things which are extra

Here we give a list of things which are possible in Free Pascal, but which didn't exist in Turbo Pascal or Delphi.

1. There are more reserved words. (see appendix B for a list of all reserved words.)
2. Functions can also return complex types, such as records and arrays.
3. You can handle function results in the function itself, as a variable. Example

```
function a : longint;

begin
   a:=12;
   while a>4 do
     begin
        {...}
     end;
end;
```

   The example above would work with TP, but the compiler would assume that the `a>4` is a recursive call. To do a recursive call in this you must append `()` behind the function name:

```
function a : longint;

begin
   a:=12;
   { this is the recursive call }
   if a()>4 then
     begin
```

```
        {...}
     end;
end;
```

4. There is partial support of Delphi constructs. (see the Programmer's guide for more information on this).

5. The `exit` call accepts a return value for functions.

```
function a : longint;

begin
   a:=12;
   if a>4 then
     begin
        exit(a*67); {function result upon exit is a*67 }
     end;
end;
```

6. Free Pascal supports function overloading. That is, you can define many functions with the same name, but with different arguments. For example:

```
procedure DoSomething (a : longint);
begin
{...}
end;

procedure DoSomething (a : real);
begin
{...}
end;
```

   You can then call procedure `DoSomething` with an argument of type `Longint` or `Real`.
   This feature has the consequence that a previously declared function must always be defined with the header completely the same:

```
procedure x (v : longint); forward;

{...}

procedure x;{ This will overload the previously declared x}
begin
{...}
end;
```

   This construction will generate a compiler error, because the compiler didn't find a definition of `procedure x (v :  longint);`. Instead you should define your procedure x as:

```
procedure x (v : longint);
{ This correctly defines the previously declared x}
begin
{...}
end;
```

(The See 5.1 switch disables overloading. When you use it, the above will compile, as in Turbo Pascal.

7. Operator overloading. Free Pascal allows to overload operators, i.e. you can define e.g. the '+' operator for matrices.
8. On FAT16 and FAT32 systems, long file names are supported.

6.3 Turbo Pascal compatibility mode

When you compile a program with the `-So` switch, the compiler will attempt to mimic the Turbo Pascal compiler in the following ways:

- Assigning a procedural variable doesn't require a @ operator. One of the differences between Turbo Pascal and Free Pascal is that the latter requires you to specify an address operator when assigning a value to a procedural variable. In Turbo Pascal compatibility mode, this is not required.
- Procedure overloading is disabled. This means that function header and implementation can be different (i.e. the function iplementation doesn't need to repeat the function header).
- Forward defined procedures don't need the full parameter list when they are defined. Due to the procedure overloading feature of Free Pascal, you must always specify the parameter list of a function when you define it, even when it was declared earlier with `Forward`. In Turbo Pascal compatibility mode, there is no function overloading, hence you can omit the parameter list:

```
Procedure a (L : Longint); Forward;

...

Procedure a ; { No need to repeat the (L : Longint) }

begin
 ...
end;
```

- recursive function calls are handled dfferently. Consider the following example :

```
Function expr : Longint;

begin
  ...
  Expr:=L:
  Writeln (Expr);
  ...
end;
```

  In Turbo Pascal compatibility mode, the function will be called recursively when the `writeln` statement is processed. In Free Pascal, the function result will be printed. In order to call the function recusively under Free Pascal, you need to implement it as follows :

```
Function expr : Longint;

begin
  ...
  Expr:=L:
  Writeln (Expr());
  ...
end;
```

- Any text after the final `End.` statement is ignored. Normally, this text is processed too.
- You cannot assign procedural variables to void pointers.
- The @ operator is typed when applied on procedures.
- You cannot nest comments.

Chapter 7

Utilities and units that come with Free Pascal

Besides the compiler and the Run-Time Library, Free Pascal comes with some utility programs and units. Here we list these programs and units.

7.1 Supplied programs

ppudump program

ppudump is a program which shows the contents of a Free Pascal unit. It is distributed with the compiler you can just issue the following command

```
ppudump [options] foo.ppu
```

to display the contents of the foo.ppu unit. You can specify multiple files on the command line.

The options can be used to change the verbosity of the display. By default, all available information is displayed. You can set the verbosity level using the `-Vxxx` option. Here, `xxx` is a combination of the following letters:

h: show header info.

i: show interface information.

m: show implementation information.

d: show only (interface) definitions.

s: show only (interface) symbols.

b: show browser info.

a: show everything (default if no -V option is present).

Demo programs

Also distributed with Free Pascal comes a series of demonstration programs. These programs have no other purpose than demonstrating the capabilities of Free Pascal. They are located in the demo directory of the sources.

Documentation Example programs

All example programs of the documentation are available. Check out the directories that end on ex in the documentation sources. There you will find all example sources.

ppumove program

ppumove is a program to make shared or static libraries from multiple units. It can be compared with the tpumove program that comes with Turbo Pascal.

It should be distributed in binary form along with the compiler.

It's usage is very simple:

```
ppumove [options] unit1.ppu unit2.ppu ... unitn.ppu
```

Where `options` is a combination of

-b: If specified, ppumve will generate a batch file that will contain the external linking and archiving commands that must be executed. The name of this batch file is pmove.sh on LINUX, and pmove.bat otherwise.

-d xxx: If specified, the output files will put in the directory xxx

-e xxx: Sets the extension of the moved unit files to xxx. By default, this is .ppl. You don't have to specify the dot.

-o xxx: sets the name of the output file, i.e. the name of the file containing all the units. This parameter is mandatory when you use multiple files. On LINUX, ppumove will prepend this name with lib if it isn't already there, and will add an extension appropriate to the type of library.

-q: Causes ppumove to operate silently.

-s: Tells ppumove to make a static library instead of a dynamic one; By default a dynamic library is made on LINUX.

-w: Tells ppumove that it is working under WINDOWSNT. This will change the names of te linker and archiving program to ldw and arw, respectively.

-h or -?: will display a short help.

The action of the ppumve program is as follows: It takes each of the unit files, and modifies it so that the compile will know that it should look for the unit code in the library. The new unit files will have an extension .ppu, this can be changed with the `-e` option. It will then put together all the object files of the units into one library, static or dynamic, depending on the presence of the `-s` option.

The name of this library must be set with the `-o` option. If needed, the prefix lib will be prepended under LINUX.. The extension will be set to .a for static libraries, for shared libraries the extensions are `.so` on linux, and `.dll` under WINDOWSNT and OS/2.

As an example, the following command

```
./ppumove -o both -e ppl ppu.ppu timer.ppu
```

under linux, will generate the following output:

```
PPU-Mover Version 0.99.7
Copyright (c) 1998 by the Free Pascal Development Team

Processing ppu.ppu... Done.
Processing timer.ppu... Done.
Linking timer.o ppu.o
Done.
```

And it will produce the following files:

1. libboth.so : The shared library containing the code from ppu.o and timer.o. Under WINDOWSNT, this file would be called both.dll.
2. timer.ppl : The unit file that tells the Free Pascal compiler to look for the timer code in the library.
3. ppu.ppl : The unit file that tells the Free Pascal compiler to look for the timer code in the library.

You could then use or distribute the files libboth.so, timer.ppl and ppu.ppl.

7.2 Supplied units

Here we list the units that come with the Free Pascal distribution. Since there is a difference in the supplied units per operating system, we list them separately per system. They are documented in the Unit reference.

Under DOS

strings This unit provides basic string handling routines for the `pchar` type, comparable to similar routines in standard `C` libraries.

objects This unit provides basic routines for handling objects.

dos This unit provides basic routines for accessing the operating system DOS. It provides almost the same functionality as the Turbo Pascal unit.

printer This unit provides all you need for rudimentary access to the printer.

getopts This unit gives you the GNU `getopts` command-line arguments handling mechanism. It also supports long options.

crt This unit provides basic screen handling routines. It provides the same functionality as the Turbo Pascal `CRT` unit.

graph This unit provides basic graphics handling, with routines to draw lines on the screen, display texts etc. It provides the same functions as the Turbo Pascal unit.

go32 This unit provides access to possibilities of the `GO32` DOS extender.

emu387 This unit provides support for the coprocessor emulator.

mmx This unit provides support for `mmx` extensions in your code.

Under Linux

strings This unit provides basic string handling routines for the `PChar` type, comparable to similar routines in standard `C` libraries.

objects This unit provides basic routines for handling objects.

crt This unit provides basic screen handling routines. It provides the same functionality Turbo Pascal `CRT` unit. It works on any terminal which supports the `vt100` escape sequences.

dos This unit provides an emulation of the same unit under DOS. It is intended primarily for easy porting of Pascal programs from DOS to LINUX. For good performance, however, it is recommended to use the `linux` unit.

linux This unit provides access to the LINUX operating system. It provides most file and I/O handling routines that you may need. It implements most of the standard `C` library constructs that you will find on a Unix system. If you do a lot of disk/file operations, the use of this unit is recommended over the one you use under Dos.

printer This unit provides an interface to the standard Unix printing mechanism.

getopts This unit gives you the GNU `getopts` command-line arguments handling mechanism. It also supports long options.

mmx This unit provides support for `mmx` extensions in your code.

sockets This unit gives you access to sockets and TCP/IP programming.

graph Is an implementation of Borlands `graph` unit, which works on the Linux console. It's implementation is fairly complete, the only non-functional things are the fillpatterns and line styles. It uses the libvga and libvgagl graphics libraries, so you need these installed for this unit to work. Also, programs using this library need to be run as root, or setuid root, and hence are a potential security risk.

ports This implements the various `port[]` constructs. These are provided for compatibility only, and it is not recommended to use them extensively. Programs using this construct must be run as ruit or setuid root, and are a serious security risk on your system.

Chapter 8

Debugging your Programs

Free Pascal supports debug information for the GNU debugger `gdb`. This chapter describes shortly how to use this feature. It doesn't attempt to describe completely the GNU debugger, however. For more information on the workings of the GNU debugger, see the `gdb` users' guide.

Free Pascal also suports `gprof`, the GNU profiler, see section 8.4 for more information on profiling.

8.1 Compiling your program with debugger support

First of all, you must be sure that the compiler is compiled with debugging support. Unfortunately, there is no way to check this at run time, except by trying to compile a program with debugging support.

To compile a program with debugging support, just specify the `-g` option on the command-line, as follows:

```
ppc386 -g hello.pp
```

This will generate debugging information in the executable from your program. You will notice that the size of the executable increases substantially because of this[1].

Note that the above will only generate debug information `f`or the code that has been generated when compiling `hello.pp`. This means that if you used some units (the system unit, for instance) which were not compiled with debugging support, no debugging support will be available for the code in these units.

There are 2 solutions for this problem.

1. Recompile all units manually with the `-g` option.
2. Specify the 'build' option (`-B`) when compiling with debugging support. This will recompile all units, and insert debugging information in each of the units.

The second option may have undesirable side effects. It may be that some units aren't found, or compile incorrectly due to missing conditionals, etc..

If all went well, the executable now contains the necessary information with which you can debug it using GNU `gdb`.

[1] A good reason not to include debug information in an executable you plan to distribute.

8.2 Using gdb to debug your program

To use gdb to debug your program, you can start the debugger, and give it as an option the *full* name of your program:

```
gdb hello
```

Or, under DOS:

```
gdb hello.exe
```

This starts the debugger, and the debugger immediately loads your program into memory, but it does not run the program yet. Instead, you are presented with the following (more or less) message, followed by the gdb prompt '(gdb)':

```
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.15.1 (i486-slackware-linux),
Copyright 1995 Free Software Foundation, Inc...
(gdb)
```

To start the program you can use the `run` command. You can optionally specify command-line parameters, which will then be fed to your program, for example:

```
(gdb) run -option -anotheroption needed_argument
```

If your program runs without problems, gdb will inform you of this, and return the exit code of your program. If the exit code was zero, then the message 'Program exited normally'.

If something went wrong (a segmentation fault or so), gdb will stop the execution of your program, and inform you of this with an appropriate message. You can then use the other gdb commands to see what happened. Alternatively, you can instruct gdb to stop at a certain point in your program, with the `break` command.

Here is a short list of gdb commands, which you are likely to need when debugging your program:

quit Exits the debugger.

kill Stops a running program.

help Gives help on all gdb commands.

file Loads a new program into the debugger.

directory Add a new directory to the search path for source files.
Remark: My copy of gdb needs '.' to be added explicitly to the search path, otherwise it doesn't find the sources.

list Lists the program sources per 10 lines. As an option you can specify a line number or function name.

break Sets a breakpoint at a specified line or function

awatch Sets a watch-point for an expression. A watch-point stops execution of your program whenever the value of an expression is either read or written.

for more information, see the gdb users' guide, or use the 'help' function in gdb.

The appendix F contains a sample init file for gdb, which produces good results when debugging Free Pascal programs.

It is also possible to use RHIDE, a text-based IDE that uses gdb. There is a version of RHIDE available that can work together with FPC.

8.3 Caveats when debugging with gdb

There are some peculiarities of Free Pascal which you should be aware of when using gdb. We list the main ones here:

1. Free Pascal generates information for GDB in uppercare letters. This is a consequence of the fact that pascal is a case insensitive language. So, when referring to a variable or function, you need to make it's name all uppercase.

   As an example, of you want to watch the value of a loop variable count, you should type

   ```
   watch COUNT
   ```

   Or if you want stop when a certain function (e.g MyFunction) is called, type

   ```
   break MYFUNCTION
   ```

2. Line numbers may be off by a little. This is a bug in Free Pascal and will be fixed as soon as possible.

3. gdb does not know sets.

4. gdb doesn't know strings. Strings are represented in gdb as records with a length field and an array of char contaning the string.

   You can also use the following user function to print strings:

   ```
   define pst
   set $pos=&$arg0
   set $strlen = {byte}$pos
   print {char}&$arg0.st@($strlen+1)
   end

   document pst
     Print out a Pascal string
   end
   ```

   If you insert it in your gdb.ini file, you can look at a string with this function. There is a sample gdb.ini in appendix F.

5. Objects are difficult to handle, mainly because gdb is oriented towards C and C++. The workaround implemented in Free Pascal is that object methods are represented as functions, with an extra parameter this (all lowercase !) The name of this function is a concatenation of the object type and the function name, separated by two underscore characters.

   For example, the method TPoint.Draw would be converted to TPOINT__DRAW, and could be stopped at with

```
break TPOINT__DRAW
```

6. Global overloaded functions confuse `gdb` because they have the same name. Thus you cannot set a breakpoint at an overloaded function, unless you know it's line number, in which case you can set a breakpoint at the starting linenumber of the function.

8.4 Support for `gprof`, the gnu profiler

You can compile your programs with profiling support. for this, you just have to use the compiler switch `-pg`. The compiler wil insert the necessary stuff for profiling.

When you have done this, you can run your program as you normally would run it.

```
yourexe
```

Where yourexe is the name of your executable.

When your program finishes a file called gmon.out is generated. Then you can start the profiler to see the output. You can better redirect the output to a file, becuase it could be quite a lot:

```
gprof yourexe > profile.log
```

Hint: you can use the –flat option to reduce the amount of output of gprof. It will then only output the information about the timings

For more information on the GNU profiler `gprof`, see its manual.

Chapter 9

CGI programming in Free Pascal

In these days of heavy WWW traffic on the Internet, CGI scripts have become an important topic in computer programming. While CGI programming can be done with almost any tool you wish, most languages aren't designed for it. Perl may be a notable exception, but perl is an interpreted language, the executable is quite big, and hence puts a big load on the server machine.

Because of its simple, almost intuitive, string handling and its easy syntax, Pascal is very well suited for CGI programming. Pascal allows you to quickly produce some results, while giving you all the tools you need for more complex programming. The basic RTL routines in principle are enough to get the job done, but you can create, with relatively little effort, some units which can be used as a base for more complex CGI programming.

That's why, in this chapter, we will discuss the basics of CGI in Free Pascal. In the subsequent, we will assume that the server for which the programs are created, are based upon the NCSA `httpd` WWW server, as the examples will be based upon the NCSA method of CGI programming[1]. They have been tested with the apache server on LINUX, and the xitami server on WINDOWSNT.

The two example programs in this chapter have been tested on the command line and worked, under the condition that no spaces were present in the name and value pairs provided to them.

There is however, a faster and generally better uncgi unit available, you can find it on the contributed units page of the Free Pascal web site. It uses techniques discussed here, but in a generally more efficient way, and it also provides some extra functionality, not discussed here.

9.1 Getting your data

Your CGI program must react on data the user has filled in on the form which your web-server gave him. The Web server takes the response on the form, and feeds it to the CGI script.

There are essentially two ways of feeding the data to the CGI script. We will discuss both.

[1] ... and it's the only WWW-server I have to my disposition at the moment.

Data coming through standard input.

The first method of getting your data is through standard input. This method is invoked when the form uses a form submission method of `POST`. The web browser sets three environment variables `REQUEST_METHOD`, `CONTENT_TYPE` and `CONTENT_LENGTH`. It feeds then the results of the different fields through standard input to the CGI script. All the Pascal program has to do is :

- Check the value of the `REQUEST_METHOD` environment variable. The `getenv` function will retrieve this value this for you.
- Check the value of the `CONTENT_TYPE` environment variable.
- Read `CONTENT_LENGTH` characters from standard input. `read (c)` with `c` of type `char` will take care of that.

if you know that the request method will always be `POST`, and the `CONTENT_TYPE` will be correct, then you can skip the first two steps. The third step can be done easier: read characters until you reach the end-of-file marker of standard input.

The following example shows how this can be achieved:

```
program cgi_post;

uses dos;

const max_data = 1000;

type datarec = record
  name,value : string;
  end;

var data : array[1..max_data] of datarec;
    i,nrdata : longint;
    c  : char;
    literal,aname : boolean;

begin
writeln ('Content-type: text/html');
writeln;
if getenv('REQUEST_METHOD')<>'POST' then
   begin
   writeln ('This script should be referenced with a METHOD of POST');
   write ('If you don''t understand this, see this ');
   write ('< A HREF="http://www.ncsa.uiuc.edu/SDG/Softare/Mosaic');
   writeln ('/Docs/fill-out-forms/overview.html">forms overview</A>.');
   halt(1);
   end;
if getenv('CONTENT_TYPE')<>'application/x-www-form-urlencoded' then
   begin
   writeln ('This script can only be used to decode form results');
   halt(1)
   end;
nrdata:=1;
aname:=true;
```

```
while not eof(input) do
  begin
  literal:=false;
  read(c);
  if c='\' then
     begin
     literal:=true;
     read(c);
     end;
  if literal or ((c<>'=') and (c<>'&')) then
     with data[nrdata] do
        if aname then name:=name+c else value:=value+c
  else
     begin
     if c='&' then
         begin
         inc (nrdata);
         aname:=true;
         end
      else
         aname:=false;
      end
  end;
writeln ('<H1>Form Results :</H1>');
writeln ('You submitted the following name/value pairs :');
writeln ('<UL>');
for i:=1 to nrdata do writeln ('<LI> ',data[i].name,' = ',data[i].value);
writeln ('</UL>');
end.
```

While this program isn't shorter than the C program provided as an example at NCSA, it doesn't need any other units. everythig is done using standard Pascal procedures[2].

Note that this program has a limitation: the length of names and values is limited to 255 characters. This is due to the fact that strings in Pascal have a maximal length of 255. It is of course easy to redefine the `datarec` record in such a way that longer values are allowed. In case you have to read the contents of a `TEXTAREA` form element, this may be needed.

Data passed through an environment variable

If your form uses the `GET` method of passing it's data, the CGI script needs to read the `QUERY_STRING` environment variable to get it's data. Since this variable can, and probably will, be more than 255 characters long, you will not be able to use normal string methods, present in pascal. Free Pascal implements the `pchar` type, which is a pointer to a null-terminated array of characters. And, fortunately, Free Pascal has a strings unit, which eases the use of the `pchar` type.

The following example illustrates what to do in case of a method of `GET`

```
program cgi_get;
```

[2] actually, this program will give faulty results, since spaces in the input are converted to plus signs by the web browser. The program doesn't check for this, but that is easy to change. The main concern here is to give the working principle.

```
uses strings,linux;

const max_data = 1000;

type datarec = record
  name,value : string;
  end;

var data : array[1..max_data] of datarec;
    i,nrdata : longint;
    p  : PChar;
    literal,aname : boolean;

begin
Writeln ('Content-type: text/html');
Writeln;
if StrComp(GetEnv('REQUEST_METHOD'),'POST')<>0 then
   begin
   Writeln ('This script should be referenced with a METHOD of GET');
   write ('If you don''t understand this, see this ');
   write ('< A HREF="http://www.ncsa.uiuc.edu/SDG/Softare/Mosaic');
   Writeln ('/Docs/fill-out-forms/overview.html">forms overview</A>.');
   halt(1);
   end;
p:=GetEnv('QUERY_STRING');
nrdata:=1;
aname:=true;
while p^<>#0  do
  begin
  literal:=false;
  if p^='\' then
     begin
     literal:=true;
     inc(longint(p));
     end;
  if ((p^<>'=') and (p^<>'&')) or literal then
     with data[nrdata] do
        if aname then name:=name+p^ else value:=value+p^
  else
     begin
     if p^='&' then
         begin
         inc (nrdata);
         aname:=true;
         end
      else
         aname:=false;
      end;
  inc(longint(p));
  end;
Writeln ('<H1>Form Results :</H1>');
Writeln ('You submitted the following name/value pairs :');
Writeln ('<UL>');
```

```
for i:=1 to nrdata do writeln ('<LI> ',data[i].name,' = ',data[i].value);
Writeln ('</UL>');
end.
```

Although it may not be written in the most elegant way, this program does the same thing as the previous one. It also suffers from the same drawback, namely the limited length of the `value` field of the `datarec`.

This drawback can be remedied by redefining `datarec` as follows:

```
type datarec = record;
      name,value : pchar;
     end;
```

and assigning at run time enough space to keep the contents of the value field. This can be done with a

```
 getmem (data[nrdata].value,needed_number_of_bytes);
```

call. After that you can do a

```
strlcopy (data[nrdata].value,p,needed_number_of_bytes);
```

to copy the data into place.

You may have noticed the following unorthodox call :

```
inc(longint(p));
```

Free Pascal doesn't give you pointer arithmetic as in C. However, `longints` and `pointers` have the same length (namely 4 bytes). Doing a type-cast to a `longint` allows you to do arithmetic on the `pointer`.

Note however, that this is a non-portable call. This may work on the I386 processor, but not on a ALPHA processor (where a pointer is 8 bytes long). This will be remedied in future releases of Free Pascal.

9.2 Producing output

The previous section concentrated mostly on getting input from the web server. To send the reply to the server, you don't need to do anything special.You just print your data on standard output, and the Web-server will intercept this, and send your output to the WWW-client waiting for it.

You can print anything you want, the only thing you must take care of is that you supply a `Contents-type` line, followed by an empty line, as follows:

```
Writeln ('Content-type: text/html');
Writeln;
{ ...start output of the form... }
```

And that's all there is to it !

9.3 I'm under Windows, what now ?

Under Windows the system of writing CGI scripts can be totally different. If you use Free Pascal under Windows then you also should be able to do CGI programming, but the above instructions may not work. They are known to work for the xitami server, however.

If some kind soul is willing to write a section on CGI programming under Windows for other servers, I'd be willing to include it here.

Appendix A

Alphabetical listing of command-line options

The following is alphabetical listing of all command-line options, as generated by the compiler:

```
ppc386 [options] <inputfile> [options]
put + after a boolean switch option to enable it, - to disable it
  -a     the compiler doesn't delete the generated assembler file
           -al        list sourcecode lines in assembler file
  -B     build all modules
  -C     code generation options
           -CD        create dynamic library
           -Ch<n>     <n> bytes heap (between 1023 and 67107840)
           -Ci        IO-checking
           -Cn        omit linking stage
           -Co        check overflow of integer operations
           -Cr        range checking
           -Cs<n>     set stack size to <n>
           -Ct        stack checking
           -CS        create static library
           -Cx        use smartlinking
  -d<x>  defines the symbol <x>
  -e<x>  set path to executable
  -E     same as -Cn
  -F     set file names and paths
           -FD<x>     sets the directory where to search for compiler utilities
           -Fe<x>     redirect error output to <x>
           -FE<x>     set exe/unit output path to <x>
           -Fg<x>     same as -Fl
           -Fi<x>     adds <x> to include path
           -Fl<x>     adds <x> to library path
           -FL<x>     uses <x> as dynamic linker
           -Fo<x>     adds <x> to object path
           -Fr<x>     load error message file <x>
           -Fu<x>     adds <x> to unit path
           -FU<x>     set unit output path to <x>, overrides -FE
  -g     generate debugger information
           -gg        use gsym
```

```
          -gd        use dbx
          -gh        use heap trace unit
  -i      information
          -iD        return compiler date
          -iV        return compiler version
          -iSO       return source OS
          -iSP       return source processor
          -iTO       return target OS
          -iTP       return target processor
  -I<x>   adds <x> to include path
  -k<x>   Pass <x> to the linker
  -l      write logo
  -n      don't read the default config file
  -o<x>   change the name of the executable produced to <x>
  -pg     generate profile code for gprof
  -P      use pipes instead of creating temporary assembler files
  -S      syntax options
          -S2        switch some Delphi 2 extensions on
          -Sc        supports operators like C (*=,+=,/= and -=)
          -Sd        tries to be Delphi compatible
          -Se        compiler stops after the first error
          -Sg        allow LABEL and GOTO
          -Si        support C++ stlyed INLINE
          -Sm        support macros like C (global)
          -So        tries to be TP/BP 7.0 compatible
          -Sp        tries to be gpc compatible
          -Ss        constructor name must be init (destructor must be done)
          -St        allow static keyword in objects
  -s      don't call assembler and linker (only with -a)
  -u<x>   undefines the symbol <x>
  -U      unit options
          -Un        don't check the unit name
          -Up<x>     same as -Fu<x>
          -Us        compile a system unit
  -v<x>   Be verbose. <x> is a combination of the following letters :
          e : Show errors (default)       d : Show debug info
          w : Show warnings               u : Show unit info
          n : Show notes                  t : Show tried/used files
          h : Show hints                  m : Show defined macros
          i : Show general info           p : Show compiled procedures
          l : Show linenumbers            c : Show conditionals
          a : Show everything             0 : Show nothing (except errors)
          b : Show all procedure          r : Rhide/GCC compatibility mode
              declarations if an error    x : Executable info (Win32 only)
              occurs
  -X      executable options
          -Xc        link with the c library
          -XD        link with dynamic libraries (defines FPC_LINK_DYNAMIC)
          -Xs        strip all symbols from executable
          -XS        link with static libraries (defines FPC_LINK_STATIC)

Processor specific options:
  -A<x>   output format
          -Ao        coff file using GNU AS
```

```
          -Anasmcoff coff file using Nasm
          -Anasmelf  elf32 (linux) file using Nasm
          -Anasmobj  obj file using Nasm
          -Amasm     obj using Masm (Mircosoft)
          -Atasm     obj using Tasm (Borland)
-R<x>  assembler reading style
          -Ratt      read AT&T style assembler
          -Rintel    read Intel style assembler
          -Rdirect   copy assembler text directly to assembler file
-O<x>  optimizations
          -Og        generate smaller code
          -OG        generate faster code (default)
          -Or        keep certain variables in registers (still BUGGY!!!)
          -Ou        enable uncertain optimizations (see docs)
          -O1        level 1 optimizations (quick optimizations)
          -O2        level 2 optimizations (-O1 + slower optimizations)
          -O3        level 3 optimizations (same as -O2u)
          -Op        target processor
                    -Op1  set target processor to 386/486
                    -Op2  set target processor to Pentium/PentiumMMX (tm)
                    -Op3  set target processor to PPro/PII/c6x86/K6 (tm)
-T<x>  Target operating system
          -TGO32V1   version 1 of DJ Delorie DOS extender
          -TGO32V2   version 2 of DJ Delorie DOS extender
          -TLINUX    Linux
          -TOS2      OS/2 2.x
          -TWin32    Windows 32 Bit

-?     shows this help
-h     shows this help without waiting
```

Appendix B

Alphabetical list of reserved words

```
absolute
abstract
and
array
as
asm
assembler
begin
break
case
cdecl
class
const
constructor
continue
destructor
dispose
div
do
downto
else
end
except
exit
export
exports
external
fail
false
far
file
finally
for
forward
function
goto
if
implementation
in
index
inherited
initialization
inline
interface
interrupt
is
label
library
mod
name
near
new
nil
not
object
of
on
operator
or
otherwise
packed
popstack
private
procedure
program
property
protected
public
raise
record
repeat
self
set
shl
shr
stdcall
string
then
to
true
try
type
unit
until
uses
var
virtual
while
with
xor
```

Appendix C

Compiler messages

This appendix is meant to list all the compiler messages. The list of messages is generated from he compiler source itself, and should be faitly complete. At this point, only assembler errors are not in the list.

C.1 General compiler messages

This section gives the compiler messages which are not fatal, but which display useful information. The number of such messages can be controlled with the various verbosity level `-v` switches.

Compiler: arg1 When the `-vu` switch is used, this line tells you what compiler is used.

Source OS: arg1 When the `-vd` switch is used, this line tells you what the source operating system is.

Info: Target OS: arg1 When the `-vd` switch is used, this line tells you what the target operating system is.

Using executable path: arg1 When the `-vu` switch is used, this line tells you where the compiler looks for it's binaries.

Using unit path: arg1 When the `-vu` switch is used, this line tells you where the compiler looks for compiled units. You can set this path with the `-Fu` or `-Up` options.

Using include path: arg1 When the `-vu` switch is used, this line tells you where the compiler looks for it's include files (files used in {`$I xxx`} statements). You can set this path with the `-I` option.

Using library path: arg1 When the `-vu` switch is used, this line tells you where the compiler looks for the libraries. You can set this path with the `-Fl` option.

Using object path: arg1 When the `-vu` switch is used, this line tells you where the compiler looks for object files you link in (files used in {`$L xxx`} statements). You can set this path with the `-Fo` option.

Info: arg1 Lines compiled, arg2 sec When the `-vi` switch is used, the compiler reports the number of lines compiled, and the time it took to compile them (real time, not program time).

Fatal: No memory left The compiler doesn't have enough memory to compile your program. There are several remedies for this:

- If you're using the build option of the compiler, try compiling the different units manually.
- If you're compiling a huge program, split it up in units, and compile these separately.
- If the previous two don't work, recompile the compiler with a bigger heap (you can use the `-Ch` option for this, See 5.1)

C.2 Scanner messages.

This section lists the messages that the scanner emits. The scanner takes care of the lexical structure of the pascal file, i.e. it tries to find reserved words, strings, etc. It also takes care of directives and conditional compiling handling.

Fatal: Unexpected end of file this typically happens in on of the following cases :

- The source file ends befor then final `end.` statement. This happens mostly when the `begin` and `end` statements aren't balanced;
- An include file ends in the middle of a statement.
- A comment wasn't closed.

Fatal: String exceeds line You forgot probably to include the closing ' in a string, so it occupies multiple lines.

Fatal: illegal character An illegal character was encountered in the input file.

Fatal: Syntax error arg1 expected This indicates that the compiler expected a different token than the one you typed. It can occur almost everywhere where you make a mistake against the pascal language.

Start reading includefile arg1 When you provide the `-vu` switch, the compiler tells you when it starts reading an included file.

Warning: Comment level arg1 found When the `-vw` switch is used, then the compiler warns you if it finds nested comments. Nested comments are not allowed in Turbo Pascal and can be a possible source of errors.

Note: argF directive (FAR) ignored The `FAR` directive is a 16-bit construction which is recorgnised but ignored by the compiler, since it produces 32 bit code.

Note: Stack check is global under linux Stack checking with the `-Cs` switch is ignored under LINUX, since LINUX does this for you. Only displayed when `-vn` is used.

Note: Ignored compiler switch arg1 With `-vn` on, the compiler warns if it ignores a switch

Warning: Illegal compiler switch arg1 You included a compiler switch (i.e. {`$...` }) which the compiler doesn't know.

Warning: This compiler switch has a global effect When `-vw` is used, the compiler warns if a switch is global.

Error: Illegal char constant This happens when you specify a character with its ASCII code, as in `#96`, but the number is either illegal, or out of range. The range is 1-255.

Fatal: Can't open file arg1 Free Pascal cannot find the program or unit source file you specified on the command line.

Fatal: Can't open include file arg1 Free Pascal cannot find the source file you specified in a {`$include` } stateent.

Error: Too many argENDIFs or argELSEs Your {`$IFDEF ..`} and {$ENDIF} statements aren't balanced.

Warning: Records fields can be aligned to 1,2,4 or 16 bytes only You are specifying the {`$PACKRECORDS n`} with an illegal value for `n`. Only 1,2,4 or 16 are valid in this case.

Warning: Enumerated can be saved in 1,2 or 4 bytes only You are specifying the {`$PACKENUM n`} with an illegal value for `n`. Only 1,2 or 4 are valid in this case.

Error: arg1 expected for arg2 defined in line arg3 Your conditional compilation statements are unbalanced.

Error: Syntax error while parsing a conditional compiling expression There is an error in the expression following the {`$if` } compiler directive.

Error: Evaluating a conditional compiling expression There is an error in the expression following the {`$if` } compiler directive.

Warning: Macro contents is cut after char 255 to evalute expression The contents of macros canno be longer than 255 characters. This is a safety in the compiler, to prevent buffer overflows. This is shown as a wraning, i.e. when the `-vw` switch is used.

Error: ENDIF without IF(N)DEF Your {`$IFDEF ..`} and {$ENDIF} statements aren't balanced.

Fatal: User defined: arg1 A user defined fatal error occurred. see also the Programmer's guide

Error: User defined: arg1 A user defined error occurred. see also the Programmer's guide

Warning: User defined: arg1 A user defined warning occurred. see also the Programmer's guide

Note: User defined: arg1 A user defined note was encountered. see also the Programmer's guide

Hint: User defined: arg1 A user defined hint was encountered. see also the Programmer's guide

Info: User defined: arg1 User defined information was encountered. see also the Programmer's guide

Error: Keyword redefined as macro has no effect You cannot redefine keywords with macros.

Fatal: Macro buffer overflow while reading or expanding a macro Your macro or it's result was too long for the compiler.

Warning: Extension of macros exceeds a deep of 16. When expanding a macro macros have been nested to a level of 16. The compiler will expand no further, since this may be a sign that recursion is used.

Error: compiler switches aren't allowed in (* ... *) styled comments Compiler switches should always be between { } comment delimiters.

Handling switch "arg1" When you set debugging info on (`-vd`) the compiler tells you when it is evaluating conditional compile statements.

ENDIF arg1 found When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

IFDEF arg1 found, arg2 When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

IFOPT arg1 found, arg2 When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

IF arg1 found, arg2 When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

IFNDEF arg1 found, arg2 When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

ELSE arg1 found, arg2 When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements.

Skipping until... When you turn on conditional messages(`-vc`), the compiler tells you where it encounters conditional statements, and whether it is skipping or compiling parts.

Info: Press ¡return¿ to continue When the `-vi` switch is used, the compiler stops compilation and waits for the `Enter` key to be pressed when it encounters a {`$STOP`} directive.

Warning: Unsupported switch arg1 When warings are turned on (`-vw`) the compiler warns you about unsupported switches. This means that the switch is used in Delphi or Turbo Pascal, but not in Free Pascal

Warning: Illegal compiler directive arg1 When warings are turned on (`-vw`) the compiler warns you about unrecognised switches. For a list of recognised switches, Programmer's guide

Back in arg1 When debugging messages are on (`-vd`) the compiler tells you when it has finished reading an include file.

Warning: Unsupported assembler style specified arg1 When you specify an assembler mode with the {`$ASMMODE xxx`} the compiler didn't recognize the mode you specified.

C.3 Parser messages

This section lists all parser messages. The parser takes care of the semantics of you language, i.e. it determines if your pascal constructs are correct.

Error: Parser - Syntax Error An error against the Turbo Pascal language was encountered. This happens typically when an illegal character is found in the sources file.

Warning: Procedure type FAR ignored This is a warning. `FAR` is a construct for 8 or 16 bit programs. Since the compile generates 32 bit programs, it ignores this directive.

Warning: Procedure type NEAR ignored This is a warning. `NEAR` is a construct for 8 or 16 bit programs. Since the compile generates 32 bit programs, it ignores this directive.

Error: No DLL File specified No longer in use.

Error: Constructor name must be INIT You are declaring a constructor with a name which isn't `init`, and the `-Ss` switch is in effect. See the `-Ss` switch (See 5.1).

Error: Destructor name must be DONE You are declaring a constructor with a name which isn't `done`, and the `-Ss` switch is in effect. See the `-Ss` switch (See 5.1).

Error: Illegal open parameter You are trying to use the wrong type for an open parameter.

Error: Procedure type INLINE not supported You tried to compile a program with C++ style inlining, and forgot to specify the `-Si` option (See 5.1). The compiler doesn't support C++ styled inlining by default.

Warning: Private methods shouldn't be VIRTUAL You declared a method in the private part of a object (class) as `virtual`. This is not allowed. Private methods cannot be overridden anyway.

Warning: Constructor should be public Constructors must be in the 'public' part of an object (class) declaration.

Warning: Destructor should be public Destructors must be in the 'public' part of an object (class) declaration.

Note: Class should have one destructor only You can declare only one destructor for a class.

Error: Local class definitions are not allowed Classes must be defined globally. They cannot be defined inside a procedure or function

Error: Anonym class definitions are not allowed An invalid object (class) declaration was encountered, i.e. an object or class without methods that isn't derived from another object or class. For example:

```
Type o = object
         a : longint;
         end;
```

will trigger this error.

Error: Illegal parameter list You are calling a function with parameters that are of a different type than the declared parameters of the function.

Error: Wrong parameter type specified There is an error in the parameter list of the function or procedure. The compiler cannot determine the error more accurate than this.

Error: Wrong amount of parameters specified There is an error in the parameter list of the function or procedure, the number of parameters is not correct.

Error: overloaded identifier isn't a function identifier The compiler encountered a symbol with the same name as an overloaded function, but it isn't a function it can overload.

Error: overloaded functions have the same parameter list You're declaring overloaded functions, but with the same parameter list. Overloaded function must have at least 1 different parameter in their declaration.

Error: function header doesn't match the forward declaration arg1 You declared the function in the `interface` part, or with the `forward` directive, but define it with a different parameter list.

Note: only values can be jumped over in enumeration types Free Pascal allows enumeration constructions as in C. Given the following declaration two declarations:

```
type a = (A_A,A_B,A_E=:6,A_UAS:=200);
type a = (A_A,A_B,A_E=:6,A_UAS:=4);
```

The second declaration would produce an error. `A_UAS` needs to have a value higher than `A_E`, i.e. at least 7.

Note: Interface and implementation names are different ! This note warns you if the implementation and interface names of a functions are different, but they have the same mangled name. This is important when using overloaded functions.

Error: function nesting ¿ 31 You can nest function definitions only 31 times.

Error: range check error while evaluating constants The constants are out of their allowed range.

Error: duplicate case label You are specifying the same label 2 times in a `case` statement.

Error: typed constants of classes are not allowed You cannot declare a constant of type class or object.

Error: functions variables of overloaded functions are not allowed You are trying to assign an overloaded function to a procedural variable. This isn't allowed.

Error: string length must be a value from 1 to 255 The length of a string in Pascal is limited to 255 characters. You are trying to declare a string with length greater than 255. (This is not true for `Longstrings` and `AnsiStrings`.

Warning: use extended syntax of DISPOSE and NEW to generate instances of objects If you have a pointer `a` to a class type, then the statement `new(a)` will not initialize the class (i.e. the constructor isn't called), although space will be allocated. you should issue the `new(a,init)` statement. This will allocate space, and call the constructor of the class.

Error: class identifier expected This happens when the compiler scans a procedure declaration that contains a dot, i.e., a object or class method, but the type in front of the dot is not a known type.

Error: method identifier expected This identifier is not a method. This happens when the compiler scans a procedure declaration that contains a dot, i.e., a object or class method, but the procedure name is not a procedure of this type.

Error: function header doesn't match any method of this class This identifier is not a method. This happens when the compiler scans a procedure declaration that contains a dot, i.e., a object or class method, but the procedure name is not a procedure of this type.

procedure/function arg1 When using the `-vp` switch, the compiler tells you when it starts processing a procedure or function implementation.

Error: Illegal floating point constant The compiler expects a floating point expression, and gets something else.

Error: FAIL can be used in constructors only You are using the `FAIL` instruction outside a constructor method.

Error: Destructors can't have parameters You are declaring a destructor with a parameter list. Destructor methods cannot have parameters.

Error: Only class methods can be referred with class references This error occurs in a situation like the following:

```
Type :
   Tclass = Class of Tobject;

Var C : TClass;

begin
...
C.free
```

`Free` is not a class method and hence cannot be called with a class reference.

Error: Only class methods can be accessed in class methods This is related to the previous error. You cannot call a method of an object from a inside a class method. The following code would produce this error:

```
class procedure tobject.x;

begin
  free
```

Because free is a normal method of a class it cannot be called from a class method.

Error: Constant and CASE types do not match One of the labels is not of the same type as the case variable.

Error: The symbol can't be exported from a library You can only export procedures and functions when you write a library. You cannot export variables or constants.

Warning: A inherited method is hidden by arg1 A method that is declared `virtual` in a parent class, should be overridden in the descendent class with the `override` directive. If you don't specify the `override` directive, you will hide the parent method; you will not override it.

Error: There is no method in an ancestor class to be overridden: arg1 You try to `override` a virtual method of a parent class that doesn't exist.

Error: No member is provided to access property You specified no `read` directive for a property.

Error: Illegal symbol for property access There is an error in the `read` or `write` directives for an array property. When you declare an array property, you can only access it with procedures and functions. The following code woud cause such an error.

```
tmyobject = class
  i : integer;
  property x [i : integer]: integer read I write i;
```

Error: Cannot access a protected field of an object here Fields that are declared in a `protected` section of an object or class declaration cannot be accessed outside the module wher the object is defined, or outside descendent object methods.

Error: Cannot access a private field of an object here Fields that are declared in a `private` section of an object or class declaration cannot be accessed outside the module where the class is defined.

Warning: overloaded of virtual method must be virtual: arg1 If you declare overloaded methods in a class, then they should either all be virtual, or none. You cannot mix them.

Error: overloaded methods which are virtual must have the same return type: arg1 If you declare virtual overloaded methods in a class definition, they must have the same return type.

Error: EXPORT declared functions can't be nested You cannot declare a function or procedure within a function or procedure that was declared as an export procedure.

Error: methods can't be EXPORTed You cannot declare a procedure that is a method for an object as `export`ed. That is, your methods cannot be called from a C program.

Error: call by var parameters have to match exactly When calling a function declared with `var` parameters, the variables in the function call must be of exactly the same type. There is no automatic type conversion.

Error: Class isn't a parent class of the current class When calling inherited methods, you are trying to call a method of a strange class. You can only call an inherited method of a parent class.

Error: SELF is only allowed in methods You are trying to use the `self` parameter outside an object's method. Only methods get passed the `self` parameters.

Error: methods can be only in other methods called direct with type identifier of the class A construction like `sometype.somemethod` is only allowed in a method.

Error: Illegal use of ':' You are using the format : (colon) 2 times on an expression that is not a real expression.

Error: range check error in set constructor or duplicate set element The declaration of a set contains an error. Either one of the elements is outside the range of the set type, either two of the elements are in fact the same.

Error: Pointer to object expected You specified an illegal type in a `New` statement. The extended synax of `New` needs an object as a parameter.

Error: Expression must be constructor call When using the extended syntax of `new`, you must specify the constructor method of the object you are trying to create. The procedure you specified is not a constructor.

Error: Expression must be destructor call When using the extended syntax of `dispose`, you must specify the destructor method of the object you are trying to dispose of. The procedure you specified is not a destructor.

Error: Illegal order of record elements When declaring a constant record, you specified the fields in the wrong order.

Error: Expression type must be class or record type A `with` statement needs an argument that is of the type `record` or `class`. You are using `with` on an expression that is not of this type.

Error: Functions with void return value can't return any value In Free Pascal, you can specify a return value for a function when using the `exit` statement. This error occurs when you try to do this with a procedure. Procedures cannot return a value.

Error: constructors and destructors must be methods You're declaring a procedure as destructor or constructor, when the procedure isn't a class method.

Error: Operator is not overloaded You're trying to use an overloaded operator when it isn't overloaded for this type.

Error: Re-raise isn't possible there You are trying to raise an exception where it isn't allowed. You can only raise exceptions in an `except` block.

Error: The extended syntax of new or dispose isn't allowed for a class You cannot generate an instance of a class with the extended syntax of `new`. The constructor must be used for that. For the same reason, you cannot call `Dispose` to de-allocate an instance of a class, the destructor must be used for that.

Error: Assembler incompatible with function return value You're trying to implement a `assembler` function, but the return type of the function doesn't allow that.

Error: Procedure overloading is switched off When using the `-So` switch, procedure overloading is switched off. Turbo Pascal does not support function overloading.

Error: It is not possible to overload this operator (overload = instead) You are trying to overload an operator which cannot be overloaded. The following operators can be overloaded :

```
+, -, *, /, =, >, <, <=, >=, is, as, in, **, :=
```

Error: Comparative operator must return a boolean value When overloading the `=` operator, the function must return a boolean value.

Error: Only virtual methods can be abstract You are declaring a method as abstract, when it isn't declared to be virtual.

Fatal: Use of unsupported feature! You're trying to force the compiler into doing something it cannot do yet.

Error: The mix of CLASSES and OBJECTS isn't allowed You cannot derive `objects` and `classes` intertwined . That is, a class cannot have an object as parent and vice versa.

Warning: Unknown procedure directive had to be ignored: arg1 The procedure direcive you secified is unknown. Recognised procedure directives are `cdecl`, `stdcall`, `popstack`, `pascal` `register`, `export`.

Error: absolute can only be associated to ONE variable You cannot specify more than one variable before the `absolute` directive. Thus, the following construct will provide this error:

```
Var Z : Longint;
    X,Y : Longint absolute Z;
```

absolute can only be associated a var or const The address of a `absolute` directive can only point to a variable or constant. Therefore, the following code will produce this error:

```
Procedure X;

var p : longint absolute x;
```

Error: absolute can only be associated a var or const The address of a `absolute` directive can only point to a variable or constant. Therefore, the following code will produce this error:

```
Procedure X;

var p : longint absolute x;
```

Error: Abtract methods shouldn't have any definition (with function body) Abstract methods can only be declared, you cannot implement them. They should be overridden by a descendant class.

Error: This overloaded function can't be local (must be exported) You are defining a overloaded function in the implementation part of a unit, but there is no corresponding declaration in the interface part of the unit.

Warning: Virtual methods are used without a constructor in arg1 If you declare objects or classes that contain virtual methods, you need to have a constructor and destructor to initialize them. The compiler encountered an object or class with virtual methods that doesn't have a constructor/destructor pair.

Macro defined: arg1 When `-vm` is used, the compiler tells you when it defines macros.

Macro undefined: arg1 When `-vm` is used, the compiler tells you when it undefines macros.

Macro arg1 set to arg2 When `-vm` is used, the compiler tells you what values macros get.

Info: Compiling arg1 When you turn on information messages (`-vi`), the compiler tells you what units it is recompiling.

Compiling arg1 for the second time When you request debug messages (`-vd`) the compiler tells you what units it recompiles for the second time.

Error: Array properties aren't allowed at this point You cannot use array properties at that point.a

Error: No property found to override You want to overrride a property of a parent class, when there is, in fact, no such property in the parent class.

Error: Only one default property is allowed, found inherited default property in class arg1 You specified a property as `Default`, but a parent class already has a default property, and a class can have only one default property.

Error: The default property must be an array property Only array properties of classes can be made `default` properties.

Error: Virtual constructors are only supported in class object model You cannot have virtual constructors in objects. You can only have them in classes.

Error: No default property available You try to access a default property of a class, but this class (or one of it's ancestors) doesn't have a default property.

Error: The class can't have a published section, use the argM+ switch If you want a `published` section in a class definition, you must use the {`$M+`} switch, whch turns on generation of type information.

Error: Forward declaration of class arg1 must be resolved here to use the class as ancestor To be able to use an object as an ancestor object, it must be defined first. This error occurs in the following situation:

```
Type ParentClas = Class;
     ChildClass = Class(ParentClass)
       ...
     end;
```

Where `ParentClass` is declared but not defined.

Error: Local operators not supported You cannot overload locally, i.e. inside procedures or function definitions.

Error: Procedure directive arg1 not allowed in interface section This procedure directive is not allowed in the `interface` section of a unit. You can only use it in the `implementation` section.

Error: Procedure directive arg1 not allowed in implementation section This procedure directive is not defined in the `implementation` section of a unit. You can only use it in the `interface` section.

Error: Procedure directive arg1 not allowed in procvar declaration This procedure directive cannot be part of a procedural of function type declaration.

Error: Function is already declared Public/Forward arg1 You will get this error if a function is defined as `forward` twice. Or it is once in the `interface` section, and once as a `forward` declaration in the `implmentation` section.

Error: Can't use both EXPORT and EXTERNAL These two procedure directives are mutually exclusive

Error: NAME keyword expected The definition of an external variable needs a `name` clause.

Warning: arg1 not yet supported inside inline procedure/function Inline procedures don't support this declaration.

Warning: Inlining disabled Inlining of procedures is disabled.

Info: Writing Browser log arg1 When information messages are on, the compiler warns you when it writes the browser log (generated with the {`$Y+` } switch).

Hint: may be pointer dereference is missing The compiler thinks that a pointer may need a dereference.

Fatal: Selected assembler reader not supported The selected assembler reader (with {`$ASMMODE xxx`} is not supported. The compiler can be compiled with or without support for a particular assembler reader.

Error: Procedure directive arg1 has conflicts with other directives You specified a procedure directive that conflicts with other directives. for instance `cdecl` and `pascal` are mutually exclusive.

Error: Calling convention doesn't match forward This error happens when you declare a function or procedure with e.g. `cdecl;` but omit this directive in the implementation, or vice versa. The calling convention is part of the function declaration, and must be repeated in the function definition.

Error: Register calling (fastcall) not supported The `register` calling convention, i.e., arguments are passed in registers instead of on the stack is not supported. Arguments are always passed on the stack.

Error: Property can't have a default value Set properties or indexed properties cannot have a default value.

Error: The default value of a property must be constant The value of a `default` declared property must be knwon at compile time. The value you specified is only known at run time. This happens .e.g. if you specify a variable name as a default value.

Error: Symbol can't be published, can be only a class Only class type variables can be in a `published` section of a class if they are not declared as a property.

Error: That kind of property can't be published Properties in a `published` section cannot be array properties. they must be moved to public sections. Properties in a `published` section must be an ordinal type, a real type, strings or sets.

C.4 Type checking errors

This section lists all errors that can occur when type checking is performed.

Error: Type mismatch This can happen in many cases:

- The variable you're assigning to is of a different type than the expression in the assignment.
- You are calling a function or procedure with parameters that are incompatible with the parameters in the function or procedure definition.

Error: Integer expression expected The compiler expects an expression of type integer, but gets a different type.

Error: Ordinal expression expected The expression must be of ordinal type, i.e., maximum a `Longint`. This happens, for instance, when you specify a second argument to `Inc` or `Dec` that doesn't evaluate to an ordinal value.

Error: Type identifier expected The identifier is not a type, or you forgot to supply a type identifier.

Error: Variable identifier expected This happens when you pass a constant to a `Inc` var or `Dec` proceure. You can only pass variables as arguments to these functions.

Error: pointer type expected The variable or expression isn't of the type `pointer`. This happens when you pass a variable that isn't a pointer to `New` or `Dispose`.

Error: class type expected The variable of expression isn't of the type `class`. This happens typically when

1. The parent class in a class declaration isn't a class.
2. An exception handler (`On`) contains a type identifier that isn't a class.

Error: Variable or type indentifier expected The argument to the `High` or `Low` function is not a variable nor a type identifier.

Error: Can't evaluate constant expression No longer in use.

Error: Set elements are not compatible You are trying to take the union of two set, when the set element types are not the same. The base type of a set must be the same when taking the union

Warning: Automatic type conversion from floating type to COMP which is an integer type An implicit type conversion from a real type to a `comp` is encountered. Since `Comp` is a 64 bit integer type, this may indicate an error.

Warning: Using / will give a floating point result When using the '/' operator in Free Pascal the result will be of type real, when used with integers.

Hint: use DIV instead to get an integer result When hints are on, then an integer division will procuce this message.

Error: string types doesn't match, because of argV+ mode When compiling in {`$V+` } mode, the string you pass as a parameter should be of the exact same type as the declared parameter of the procedure.

Error: succ or pred on enums with assignments not possible When you declared an enumeration type which has assignments in it, as in C, like in the following:

```
Tenum = (a,b,e:=5);
```

you cannot use the `Succ` or `Pred` functions on them.

Error: Can't read or write variables of this type You are trying to `read` or `write` a variable from or to a file of type text, which doesn't support that. Only integer types, booleans, reals, pchars and strings can be read from/written to a text file.

Error: Type conflict between set elements There is at least one set element which is of the wrong type, i.e. not of the set type.

Error: Integer or real expression expected The first argument to `str` must a real or integer type.

C.5 Symbol handling

This section lists all the messages that concern the handling of symbols. This means all things that have to do with procedure and variable names.

Error: Identifier not found arg1 The compiler doesn't know this symbol. Usually happens when you misspel the name of a variable or procedure, or when you forgot to declare a variable.

Fatal: Internal Error in SymTableStack() An internal error occurred in the compiler; If you encounter such an error, please contact the developers and try to provide an exact description of the circumstances in which the error occurs.

Error: Duplicate identifier arg1 The identifier was already declared in the current scope.

Error: Unknown identifier arg1 The identifier encountered hasn't been declared, or is used outside the scope where it's defined.

Error: Forward declaration not solved: arg1 This can happen in two cases:

- This happens when you declare a function (in the `interface` part, or with a `forward` directive, but do not implement it.
- You reference a type which isn't declared in the current `type` block.

Fatal: Identifier type already defined as type You are trying to redefine a type.

Error: Error in type defenition There is an error in your definition of a new array type:

One of the range delimiters in an array declaration is erroneous. For example, `Array [1..1.25]` will trigger this error.

Error: Type identifier not defined The compiler encountered an unknown type.

Error: Only static variables can be used in static methods or outside methods A static method of an object can only access static variables.

Error: Invalid call to tvarsym.mangledname() An internal error occurred in the compiler; If you encounter such an error, please contact the developers and try to provide an exact description of the circumstances in which the error occurs.

Fatal: record or class type expected The variable or expression isn't of the type `record` or `class`.

Error: Instances of classes or objects with an abtsract method are not allowed You are trying to generate an instance of a class which has an abstract method that wasn't overridden.

Error: Label not defined arg1 A label was declared, but not defined.

Error: Illegal label declaration

Error: GOTO und LABEL are not supported (use command line switch -Sg)

Error: Label not found A `goto label` was encountered, but the label isn't declared.

GOTO and LABEL are not supported (use command line switch -Sg) You must compile a program which has `label`s and `goto` statements with the `-Sg` switch. By default, `label` and `goto` aren't supported.

Error: identifier isn't a label The identifier specified after the `goto` isn't of type label.

Error: label already defined You are defining a label twice. You can define a label only once.

Error: illegal type declaration of set elements The declaration of a set contains an invalid type definition.

Error: Forward class definition not resolved arg1 You declared a class, but you didn't implement it.

Hint: Parameter not used arg1 This is a warning. The identifier was declared (locally or globally) but wasn't used (locally or globally).

Warning: Local variable not used arg1 You have declared, but not used a variable in a procedure or function implementation.

Error: Set type expected The variable or expression isn't of type `set`. This happens in an `in` statement.

Warning: Function result does not seem to be set You can get this warning if the compiler thinks that a function return value is not set. This will not be displayed for assembler procedures, or procedures that contain assembler blocks.

Error: Unknown field identifier The field doesn't exist in the record definition.

Warning: Local variable arg1 does not seem to be initialized This message is displayed if the compiler thinks that a variable will be used (i.e. appears in the right-hand-side of an expression) when it wasn't initialized first (i.e. appeared in the right-hand side of an assigment)

Error: identifier idents no member arg1 When using the extended syntax of `new`, you must specify the constructor method of the class you are trying to create. The procedure you specified does not exist.

Found declaration: arg1 You get this when you use the `-vb` switch. In case an overloaded procedure is not found, then all candidate overloaded procedures are listed, with their parameter lists.

C.6 Code generator messages

This section lists all messages that can be displayed if the code generator encounters an error condition.

Error: BREAK not allowed You're trying to use `break` outside a loop construction.

Error: CONTINUE not allowed You're trying to use `continue` outside a loop construction.

Error: Expression too complicated - FPU stack overflow Your expression is too long for the compiler. You should try dividing the construct over multiple assignments.

Error: Illegal expression This can occur under many circumstances. Mostly when trying to evaluate constant expressions.

Error: Invalid integer You made an exression which isn't an integer, and the compiler expects the result to be an integer.

Error: Illegal qualifier One of the following is happening :

- You're trying to access a field of a variable that is not a record.
- You're indexing a variable that is not an array.
- You're dereferencing a variable that is not a pointer.

Error: High range limit ¡ low range limit You are declaring a subrange, and the lower limit is higher than the high limit of the range.

Error: Illegal counter variable The type of a `for` loop variable must be an ordinal type. Loop variables cannot be reals or strings.

Error: Can't determine which overloaded function to call You're calling overloaded functions with a parameter that doesn't correspond to any of the declared function parameter lists. e.g. when you have declared a function with parameters `word` and `longint`, and then you call it with a parameter which is of type `integer`.

Error: Parameter list size exceeds 65535 bytes The I386 processor limits the parameter list to 65535 bytes (the `RET` instruction causes this)

Error: Illegal type conversion When doing a type-cast, you must take care that the sizes of the variable and the destination type are the same.

Error: File types must be var parameters You cannot specify files as value parameters, i.e. they must always be declared `var` parameters.

Error: The use of a far pointer isn't allowed there Free Pascal doesn't support far pointers, so you cannot take the address of an expression which has a far reference as a result. The `mem` construct has a far reference as a result, so the following code will produce this error:

```
var p : pointer;
...
p:=@mem[a000:000];
```

Error: illegal call by reference parameters You are trying to pass a constant or an expression to a procedure that requires a `var` parameter. Only variables can be passed as a `var` parameter.

Error: EXPORT declared functions can't be called No longer in use.

Warning: Possible illegal call of constructor or destructor (doesn't match to this context) No longer in use.

Note: Inefficient code You construction seems dubious to the compiler.

Warning: unreachable code You specified a loop which will never be executed. Example:

```
while false do
  begin
  {.. code ...}
  end;
```

Error: procedure call with stackframe ESP/SP The compiler enocountered a procedure or function call inside a procedure that uses a `ESP/SP` stackframe. Normally, when a call is done the procedure needs a `EBP` stackframe.

Error: Abstract methods can't be called directly You cannot call an abstract method directy, instead you must call a overriding child method, because an abstract method isn't implemented.

Fatal: Internal Error in getfloatreg(), allocation failure An internal error occurred in the compiler; If you encounter such an error, please contact the developers and try to provide an exact description of the circumstances in which the error occurs.

Fatal: Unknown float type The compiler cannot determine the kind of float that occurs in an expression.

Fatal: SecondVecn() base defined twice An internal error occurred in the compiler; If you encounter such an error, please contact the developers and try to provide an exact description of the circumstances in which the error occurs.

Fatal: Extended cg68k not supported The varextended type is not supported on the m68k platform.

Fatal: 32-bit unsigned not supported in MC68000 mode The cardinal is not supported on the m68k platform.

Fatal: Internal Error in secondinline() An internal error occurred in the compiler; If you encounter such an error, please contact the developers and try to provide an exact description of the circumstances in which the error occurs.

Register arg1 weight arg2 arg3 Debugging message. Shown when the compiler considers a variable for keeping in the registers.

Error: Stack limit excedeed in local routine Your code requires a too big stack. Some operating systems pose limits on the stack size. You should use less variables or try ro put large variables on the heap.

Stack frame is omited Some procedure/functions do not need a complete stackframe, so it is omitted. This message will be displayed when the -vd switch is used.

Error: Unable to inline object methods You cannot have inlined object methods.

Error: Unable to inline procvar calls A procedure with a procedural variable call cannot be inlined.

Error: No code for inline procedure stored The compiler couldn't store code for the inline procedure.

C.7 Unit loading messages.

This section lists all messages that can occur when the compiler is loading a unit from disk into memory. Many of these mesages are informational messages.

PPU Loading arg1 When the `-vu` switch is used, the compiler tells you what units it loads.

PPU Time: arg1 When you use the `-vd` flag, the unit time is shown.

PPU File too short When you use the `-vd` flag, the unit time is shown.

PPU Invalid Header (no PPU at the begin) A unit file contains as the first three bytes the ascii codes of `PPU`

PPU Invalid Version arg1 This unit file was compiled with a different version of the compiler, and cannot be read.

PPU Flags: arg1 When you use the `-vd` flag, the unit flags are shown.

PPU Crc: arg1 When you use the `-vd` flag, the unit CRC check is shown.

PPU Source: arg1 When you use the `-vt` flag, the unit CRC check is shown.

objectfile and assemblerfile are older than ppufile When you use the `-vd` flag, the compiler warns if the assembler of object file of the unit are older than the unit file itself.

objectfile is older than assemblerfile When you use the `-vd` flag, the compiler warns if the assembler file of the unit is older than the object file of the unit.

Unitsearch: arg1 When you use the `-vt`, the compiler tells ou where it tries to find unt files.

Writing arg1 When you specify the `-vu` switch, the compiler will tell you where it writes the unit file.

Fatal: Can't Write PPU-File An err

Fatal: reading PPU-File This means that the unit file was corrupted, and contains invalid information. Recompilation will be necessary.

Fatal: Invalid PPU-File entry: arg1 The unit the compiler is trying to read is corrupted, or generated with a newer version of the compiler.

Fatal: PPU Dbx count problem There is an inconsistency in the debugging information of the unit.

Error: Illegal unit name: arg1 The name of the unit doesn't match the file name.

Fatal: Too much units Free Pascal has a limit of 1024 units in a program. You can change this behavior by changing the `maxunits` constant in the files.pas file of the compiler, and recompiling the compiler.

Fatal: Circular unit reference Two units are using each other in the interface part. This is only allowed in the `implementation` part. At least one unit must contain the other one in the `implementation` section.

Fatal: Can't compile unit arg1, no sources available A unit was found that needs to be recompiled, but no sources are available.

Warning: Compiling the system unit requires the -Us switch When recompiling the system unit (it needs special treatment), the `-Us` must be specified.

Fatal: There were arg1 errors compiling module, stopping When the compiler encounters a fatal error or too many errors in a module then it stops with this message.

C.8 Command-line handling errors

This section lists errors that occur when the compiler is processing the command line or handling the configuration files.

Warning: Only one source file supported You can specify only one source file on the command line. The first one will be compiled, others will be ignored. This may indicate that you forgot a `'-'` sign.

Warning: DEF file can be created only for OS/2 This option can only be specified when you're compiling for OS/2

Error: nested response files are not supported you cannot nest response files with the `@file` command-line option.

Fatal: No source file name in command line The compiler expects a source file name on the command line.

Error: Illegal parameter: arg1 You specified an unknown option.

Hint: -? writes help pages When an unknown option is given, this message is diplayed.

Fatal: Too many config files nested You can only nest up to 16 config files.

Fatal: Unable to open file arg1 The option file cannot be found.

Note: Reading further options from arg1 Displayed when you have notes turned on, and the compiler switches to another options file.

Warning: Target is already set to: arg1 Displayed if more than one `-T` option is specified.

Warning: Shared libs not supported on DOS platform, reverting to static If you specify `-CD` for the DOS platform, this message is displayed. The compiler supports only static libraries under DOS

Fatal: too many IF(N)DEFs the `#IF(N)DEF` statements in the options file are not balanced with the `#ENDIF` statements.

Fatal: too many ENDIFs the `#IF(N)DEF` statements in the options file are not balanced with the `#ENDIF` statements.

Fatal: open conditional at the end of the file the `#IF(N)DEF` statements in the options file are not balanced with the `#ENDIF` statements.

Warning: Debug information generation is not supported by this executable It is possible to have a compiler executable that doesn't support the generation of debugging info. If you use such an executable with the `-g` switch, this warning will be displayed.

Hint: Try recompiling with -dGDB It is possible to have a compiler executable that doesn't support the generation of debugging info. If you use such an executable with the `-g` switch, this warning will be displayed.

C.9 Assembler reader errors.

This section lists the errors that are generated by the inline assembler reader. They are *not* the messages of the assembler itself.

General assembler errors

Divide by zero in asm evaluator This fatal error is reported when a constant assembler expressions does a division by zero.

Evaluator stack overflow, Evaluator stack underflow These fatal errors are reported when a constant assembler expression is too big to evaluate by the constant parser. Try reducing the number of terms.

Invalid numeric format in asm evaluator This fatal error is reported when a non-numeric value is detected by the constant parser. Normally this error should never occur.

Invalid Operator in asm evaluator This fatal error is reported when a mathematical operator is detected by the constant parser. Normally this error should never occur.

Unknown error in asm evaluator This fatal error is reported when an internal error is detected by the constant parser. Normally this error should never occur.

Invalid numeric value This warning is emitted when a conversion from octal,binary or hexadecimal to decimal is outside of the supported range.

Escape sequence ignored This error is emitted when a non ANSI C escape sequence is detected in a C string.

Asm syntax error - Prefix not found This occurs when trying to use a non-valid prefix instruction

Asm syntax error - Trying to add more than one prefix This occurs when you try to add more than one prefix instruction

Asm syntax error - Opcode not found You have tried to use an unsupported or unknown opcode

Constant value out of bounds This error is reported when the constant parser determines that the value you are using is out of bounds, either with the opcode or with the constant declaration used.

Non-label pattern contains @ This only applied to the m68k and Intel styled assembler, this is reported when you try to use a non-label identifier with a '@' prefix.

Internal error in Findtype()

Internal Error in ConcatOpcode()

Internal Errror converting binary

Internal Errror converting hexadecimal

Internal Errror converting octal

Internal Error in BuildScaling()

Internal Error in BuildConstant()

internal error in BuildReference()

internal error in HandleExtend()

Internal error in ConcatLabeledInstr() These errors should never occur, if they do then you have found a new bug in the assembler parsers. Please contact one of the developers.

Opcode not in table, operands not checked This warning only occurs when compiling the system unit, or related files. No checking is performed on the operands of the opcodes.

@CODE and @DATA not supported This Turbo Pascal construct is not supported.

SEG and OFFSET not supported This Turbo Pascal construct is not supported.

Modulo not supported Modulo constant operation is not supported.

Floating point binary representation ignored

Floating point hexadecimal representation ignored

Floating point octal representation ignored These warnings occur when a floating point constant are declared in a base other then decimal. No conversion can be done on these formats. You should use a decimal representation instead.

Identifier supposed external This warning occurs when a symbol is not found in the symolb table, it is therefore considered external.

Functions with void return value can't return any value in asm code Only routines with a return value can have a return value set.

Error in binary constant

Error in octal constant

Error in hexadecimal constant

Error in integer constant These errors are reported when you tried using an invalid constant expression, or that the value is out of range.

Invalid labeled opcode

Asm syntax error - error in reference

Invalid Opcode

Invalid combination of opcode and operands

Invalid size in reference

Invalid middle sized operand

Invalid three operand opcode

Assembler syntax error

Invalid operand type You tried using an invalid combination of opcode and operands, check the syntax and if you are sure it is correct, please contact one of the developers.

Unknown identifier The identifier you are trying to access does not exist, or is not within the current scope.

Trying to define an index register more than once

Trying to define a segment register twice

Trying to define a base register twice You are trying to define an index/segment register more then once.

Invalid field specifier The record or object field you are trying to access does not exist, or is incorrect.

Invalid scaling factor

Invalid scaling value

Scaling value only allowed with index Allowed scaling values are 1,2,4 or 8.

Cannot use SELF outside a method You are trying to access the SELF identifier for objects outside a method.

Invalid combination of prefix and opcode This opcode cannot be prefixed by this instruction

Invalid combination of override and opcode This opcode cannot be overriden by this combination

Too many operands on line At most three operand instructions exist on the m68k, and i386, you are probably trying to use an invalid syntax for this opcode.

Duplicate local symbol You are trying to redefine a local symbol, such as a local label.

Unknown label identifer

Undefined local symbol

local symbol not found inside asm statement This label does not seem to have been defined in the current scope

Assemble node syntax error

Not a directive or local symbol The assembler statement is invalid, or you are not using a recognized directive.

I386 specific errors

repeat prefix and a segment override on <= i386 ... A problem with interrupts and a prefix instruction may occur and may cause false results on 386 and earlier computers.

Fwait can cause emulation problems with emu387 This warning is reported when using the FWAIT instruction, it can cause emulation problems on systems which use the em387.dxe emulator.

You need GNU as version ¿= 2.81 to compile this MMX code MMX assembler code can only be compiled using GAS v2.8.1 or later.

NEAR ignored

FAR ignored `NEAR` and `FAR` are ignored in the intel assemblers, but are still accepted for compatiblity with the 16-bit code model.

Invalid size for MOVSX/MOVZX

16-bit base in 32-bit segment

16-bit index in 32-bit segment 16-bit addressing is not supported, you must use 32-bit addressing.

Constant reference not allowed It is not allowed to try to address a constant memory address in protected mode.

Segment overrides not supported Intel style (eg: rep ds stosb) segment overrides are not support by the assembler parser.

Expressions of the form [sreg:reg... are currently not supported] To access a memory operand in a different segment, you should use the sreg:[reg...] snytax instead of [sreg:reg...]

Size suffix and destination register do not match In intel AT&T syntax, you are using a register size which does not concord with the operand size specified.

Invalid assembler syntax. No ref with brackets

Trying to use a negative index register

Local symbols not allowed as references

Invalid operand in bracket expression

Invalid symbol name:

Invalid Reference syntax

Invalid string as opcode operand:

Null label references are not allowed

Using a defined name as a local label

Invalid constant symbol

Invalid constant expression

/ at beginning of line not allowed

NOR not supported

Invalid floating point register name

Invalid floating point constant:

Asm syntax error - Should start with bracket

Asm syntax error - register:

Asm syntax error - in opcode operand

Invalid String expression

Constant expression out of bounds

Invalid or missing opcode

Invalid real constant expression

Parenthesis are not allowed

Invalid Reference

Cannot use __SELF outside a method

Cannot use __OLDEBP outside a nested procedure

Invalid segment override expression

Strings not allowed as constants

Switching sections is not allowed in an assembler block

Invalid global definition

Line separator expected

Invalid local common definition

Invalid global common definition

assembler code not returned to text

invalid opcode size

Invalid character: ¡

Invalid character: ¿

Unsupported opcode

Invalid suffix for intel assembler

Extended not supported in this mode

Comp not supported in this mode

Invalid Operand:

Override operator not supported

m68k specific errors.

Increment and Decrement mode not allowed together You are trying to use dec/inc mode together.

Invalid Register list in movem/fmovem The register list is invalid, normally a range of registers should be separated by - and individual registers should be separated by a slash.

Invalid Register list for opcode

68020+ mode required to assemble

Appendix D

Run time errors

The Free Pascal Run-time library generates the following errors at run-time [1]:

1 Invalid function number You tried to call a DOS function which doesn't exist.

2 File not found You can get this error when you tried to do an operation on a file which doesn't exist.

3 Path not found You can get this error when you tried to do an operation on a file which doesn't exist, or when you try to change to, or remove a directory that doesn't exist, or try to make a subdirectory of a subdirectory that doesn't exist.

4 Too many open files When attempting to open a file for reading or writing, you can get this error when your program has too many open files.

5 File access denied You don't have access to the specified file.

6 Invalid file handle If this happens, the file variable you are using is trashed; it indicates that your memory is corrupted.

12 Invalid file access code This will happen if you do a reset or rewrite of a file when `FileMode` is invalid.

15 Invalid drive number The number given to the Getdir function specifies a non-existent disk.

16 Cannot remove current directory You get this if you try to remove the current diirectory.

17 Cannot rename across drives You cannot rename a file such that it would end up on another disk or partition.

100 Disk read error DOS only. An error occurred when reading from disk. Typically when you try to read past the end of a file.

101 Disk write error DOS only. Reported when the disk is full, and you're trying to write to it.

102 File not assigned This is reported by Reset, Rewrite, Append, Rename and Erase, if you call them with an unassigne function as a parameter.

[1] The LINUX port will generate only a subset of these.

103 File not open Reported by the following functions : Close , Read, Write, Seek, EOf, FilePos, FileSize, Flush, BlockRead, and BlockWrite if the file isn't open.

104 File not open for input Reported by Read, BlockRead, Eof, Eoln, SeekEof or SeekEoln if the file isn't opened with Reset.

105 File not open for output Reported by write if a text file isn't opened with Rewrite.

106 Invalid numeric format Reported when a non-numerice value is read from a text file, when a numeric value was expected.

150 Disk is write-protected (Critical error, DOS only.)

151 Bad drive request struct length (Critical error, DOS only.)

152 Drive not ready (Critical error, DOS only.)

154 CRC error in data (Critical error, DOS only.)

156 Disk seek error (Critical error, DOS only.)

157 Unknown media type (Critical error, DOS only.)

158 Sector Not Found (Critical error, DOS only.)

159 Printer out of paper (Critical error, DOS only.)

160 Device write fault (Critical error, DOS only.)

161 Device read fault (Critical error, DOS only.)

162 Hardware failure (Critical error, DOS only.)

200 Division by zero You are dividing a number by zero.

201 Range check error If you compiled your program with range checking on, then you can get this error in the following cases:

1. An array was accessed with an index outside its declared range.
2. You're trying to assign a value to a variable outside its range (for instance a enumerated type).

202 Stack overflow error The stack has grown beyond itss maximum size. This error can easily occur if you have recursive functions.

203 Heap overflow error The heap has grown beyond its boundaries, ad you are rying to get more memory. Please note that Free Pascal provides a growing heap, i.e. the heap will try to allocate more memory if needed. However, if the heap has reached the maximum size allowed by the operating system or hardware, then you will get this error.

204 Invalid pointer operation This you will get if you call dispose or Freemem with an invalid pointer (notably, `Nil`)

205 Floating point overflow You are trying to use or produce too large real numbers.

206 Floating point underflow You are trying to use or produce too small real numbers.

207 Invalid floating point operation Can occur if you try to calculate the square root or logarithm of a negative number.

210 Object not initialized When compiled with range checking on, a program will report this error if you call a virtal method without having initialized the VMT.

211 Call to abstract method Your program tried to execute an abstract virtual method. Abstract methods should be overridden, and the overriding method should be called.

212 Stream registration error This occurs when an invalid type is registered in the objects unit.

213 Collection index out of range You are trying to access a collection item with an invalid index. (objects unit)

214 Collection overflow error The collection has reached its maximal size, and you are trying to add another element. (objects unit)

216 General Protection fault You are trying to access memory outside your appointed memory.

217 Unhandled expetion occurred An exception occurred, and there was no exception handler present. The sysutils unit installs a default exception handler which catches all excpetions and exits gracefully.

Appendix E

The Floating Point Coprocessor emulator

In this appendix we note some caveats when using the floating point emulator on GO32V2 systems. Under GO32V1 systems, all is as described in the installation section.

Q: I don't have an 80387. How do I compile and run floating point programs under GO32V2?

Q: What shall I install on a target machine which lacks hardware floating-point support?

A : Programs which use floating point computations and could be run on machines without an 80387 should be allowed to dynamically load the emu387.dxe file at run-time if needed. To do this you must link the `emu387` unit to your exectuable program, for example:

```
Program MyFloat;

Uses emu387;

var
 r: real;
Begin
 r:=1.0;
 WriteLn(r);
end.
```

`Emu387` takes care of loading the dynamic emulation point library.

You should always add emulation when you distribute floating-point programs.

A few users reported that the emulation won't work for them unless they explicitly tell `DJGPP` there is no `x87` hardware, like this:

```
 set 387=N
 set emu387=c:/djgpp/bin/emu387.dxe
```

There is an alternative FP emulator called WMEMU. It mimics a real coprocessor more closely.

WARNING: We strongly suggest that you use WMEMU as FPU emulator, since emu387.dxe does not emulate all the instructions which are used by the Run-Time Libary such as `FWAIT`.

Q: I have an 80387 emulator installed in my AUTOEXEC.BAT, but DJGPP-compiled floating point programs still doesn't work. Why?

A : DJGPP switches the CPU to protected mode, and the information needed to emulate the 80387 is different. Not to mention that the exceptions never get to the real-mode handler. You must use emulators which are designed for DJGPP. Apart of emu387 and WMEMU, the only other emulator known to work with DJGPP is Q87 from QuickWare. Q87 is shareware and is available from the QuickWare Web site.

Q: I run DJGPP in an OS/2 *DOS box, and I'm told that* OS/2 *will install its own emulator library if the CPU has no FPU, and will transparently execute FPU instructions. So why won't DJGPP run floating-point code under* OS/2 *on my machine?*

A : OS/2 installs an emulator for native OS/2 images, but does not provide FPU emulation for DOS sessions.

Appendix F

A sample **gdb.ini** file

Here you have a sample gdb.ini file listing, which gives better results when using gdb. Under LINUX you should put this in a .gdbinit file in your home directory or the current directory..

```
set print demangle off
set gnutarget auto
set verbose on
set complaints 1000
dir ./rtl/dosv2
set language c++
set print vtbl on
set print object on
set print sym on
set print pretty on
disp /i $eip

define pst
set $pos=&$arg0
set $strlen = {byte}$pos
print {char}&$arg0.st@($strlen+1)
end

document pst
  Print out a pascal string
end
```