

The GNU Pascal Manual

Jan-Jaap van der Heijden,
Peter Gerwinski,
Frank Heckenbach,
Berend de Boer,
Dominik Freche,
Eike Lange,
Peter N Lewis,

and others

Last updated Jan 2005

for version 20050331 (GCC 2.8.1, 2.95.x, 3.2.x, 3.3.x or 3.4.x)

Copyright © 1988-2005 Free Software Foundation, Inc.

For GPC 20050331 (GCC 2.8.1, 2.95.x, 3.2.x, 3.3.x or 3.4.x)

Published by the Free Software Foundation
59 Temple Place - Suite 330
Boston, MA 02111-1307, USA

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the sections entitled "GNU General Public License", "The GNU Project", "The GNU Manifesto" and "Funding for Free Software" are included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to make and distribute translations of this manual into another language, under the above conditions for modified versions, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Short Contents

Table of Contents

GNU Pascal	
------------------	--

6.2.3.3	Integer Types with Specified Size	65
6.2.3.4	Integer Types and Compatibility	66
6.2.3.5	Summary of Integer Types	

False.....

LongCard	350
LongestBool	351
LongestCard	352
LongestInt	352
LongestReal	353
LongestWord	353
LongInt	354
LongReal	355
LongWord	355
Low	356
LT	357
LTPad	357
Mark	328
Max	328
MaxChar	328
MaxInt	359
MaxReal	359
MedBool	360
MedCard	360
MedInt	361
MedReal	362
MedWord	

PChar	384
Pi	385
PObjectType	385
Pointer	386
Polar	387
Pos	387
Position	

xor 451

**9 Pascal keywords and operators supported by
GNU Pascal. 453**

**10 Where to get support for GNU Pascal; how to
report bugs. 459**

10.1 The GPC Mailing List 459

10.2 The GPC Mailing List Archives 460

10.3 Newsgroups relevant to GPC 460

10.4 Where to get individual support for GPC 460

10.5 If the compiler crashes 460

10.6 Howto o report GPC bugs 461

.....

Appendix B	GNU LESSER GENERAL PUBLIC	
LICENSE		497

GNU Pascal

This manual documents how to run, install and maintain the GNU Pascal Compiler (GPC),

- Global 'goto'. (Yes, 'goto' has its place when it is not restricted to the current routine.) [Example (parserdemo.pas)]
- Automatically set discriminants of variant records in 'New'. [Example (variantdemo.pas)]
- Sets of arbitrary size. [Example (bigsetsdemo.pas)]
- From Extended Pascal:
 - Strings of arbitrary length. [Example (stringschemademo.pas)]
 - 'ReadStr' and 'WriteStr'. Read from and write to strings with the full comfort of 'ReadLn'. [Example (r)1w(s)-1(tt)1(igd)1(e)-1m(o.p)1(as)-1())]TJ/F6010.909Tf-045.645-14.745T
 - ([E)1(x)am)-1(p)1(le)]TJ352.4012-11.955Td[((b)1(igs)-1(etsdem)

- From Borland Delphi:
 - ‘abstract’ object types and methods
 - ‘is’ and ‘as’ operators to test object type membership
 - Comments with ‘//’
 - Empty parameter lists with ‘J050J051’
 - Assertions
 - A ‘SetLength’ procedure for strings makes it unnecessary to use dirty tricks like assignments to the “zeroth character”.
 - ‘Initialize’ and ‘Finalize’ for low-level handling of variables.
 - ‘initialization’ and ‘finalization’ for units.
- From Pascal-SC (PXSCJ051):
 - User-definable operators. Add your vectors with ‘+’.
- Carefully designed GNU extensions help you to make your real-world programs portable
 - 64-bit signed and unsigned integer types.
 - Special types guarantee compatibility to other GNU languages such as GNU C.

- trimming string relations as functions ('EQPad' etc.) (fjf873.pas)
- new options '-W[no-]i n t e r f a c e - f i l e - n a m e'
- 'SeekEOF' and ' ' new opth515691(s)-27(erator1(567)pr(of(2:)-ce2:)-dj1)-1(.t1)-444(-1(568)and1(

-

3.2.4 What additional libraries should I have?

You will need certain additional libraries when you compile some of the units. These can be found in the directory

3.3.3 What do I download?

As discussed in [Section 3.2.2 \[Components\]](#), [page 17](#), other than GPC itself, you need an assembler, linker and friends, a C library and possibly a debugger. The site <http://www.delorie.com/djgpp/> recommends [\[h.h26\(,\)-827\(oh26\(follo\(linw,\)-69g\(oh26\(fila\)-623\(ds\(oh26\(ry](#)


```
{ 'Byte' is 'unsigned char' in C,
  'ShortCard' is 'unsigned short' in C,
  'MedCard' is 'unsigned long' in C,
  'Word' is 'unsigned' in C,
  etc. (all these types are built-in). }
```

```
type
  TDpmiVersionRet = record
    Major      : Byte;
    Minor      : Byte;
    Flags      : ShortCard;
    CPU        : Byte;
    Master_PIC : Byte;
    Slave_PIC  : Byte;
  end;
```

```
type
  TDpmiFreeMemInfo = record
    LargestAvailableFreeBlockInBytes,
    MaximumUnlockedPageAllocationInPages,
    MaximumLockedPageAllocationInPages,
    LinearAddressSpaceSizeInPages,
    TotalNumberOfUnlockedPages,
    TotalNumberOfFreePages,
    TotalNumberOfPhysicalPages,
    FreeLinearAddressSpaceInPages,
    SizeOfPagingFilePartInPages,
    Reserved1,
    Reserved2,
    Reserved3: MedCard;
  end;
```

```
function DpmiGetVersion (var Version: TDpmiVersionRet): Integer;
  external name '__dpmi_get_version';
```

```
function DpmiGetFreeMemoryInformation
  (var MemInfo: TDpmiFreeMemInfo): Integer;
  external name '__dpmi_get_free_memory_information';
```

```
var
  Version: TDpmiVersionRet;
  MemInfo: TDpmiFreeMemInfo;
```

```
begin
  if DpmiGetVersion (Version) = 0 then
    begin
      WriteLn ('CPU type:      ', Version.CPU, '86');
      WriteLn ('DPMI major:    ', Version.Major);
      WriteLn ('DPMI minor:    ', Version.Minor);
    end
  else
```


3.6 Miscellaneous

4 How to download, compile and install GNU Pascal.

This chapter covers:

- Downloading GPC sources or binaries
- Installation instructions for a GPC binary distribution
- Compilation of the source distribution on a Unix system
- Compilation notes for specific platforms
- Building and installing a cross-compiler
- Crossbuilding a compiler

4.1 Where and what to download

You can download the source code of the current GNU Pascal release from

CRT	Dos, MS-Windows	PDCurses (3)
GMP	any	gmp
RegEx	any	rx
(debugging)	Unix, MS-Windows	ElectricFence (4)

Notes:

(1) ncurses version 5.0 or newer is strongly recommended because older versions contain a bug that severely affects CRT programs.

CygWin

CygWin is an environment which implements a POSIX layer under MS Windows, giving

```
[gcc]
COMPI LER_PATH=%/>; COMPI LER_PATH%DJDI R%/bi n
LI BRARY_PATH=%/>; LI BRARY_PATH%DJDI R%/I i b
```

```
[gpc]
COMPI LER_PATH=%/>; COMPI LER_PATH%DJDI R%/bi n
LI BRARY_PATH=%/>; LI BRARY_PATH%DJDI R%/I i b
```

If you are using the DJGPP version of GPC but do not have a 'DJGPP' directory, please download and install DJGPP (see [Section 4.1 \[Download\]](#), page 27).

Binary distributions include 'I i bgcc. a' and 'specs', files that are normally part of GCC. If you have GCC installed, they will be replaced unless you manually install the archive.

4.3 Compiling GPC

The preferred way to distribute GNU software is distribution of the source code. However, it can be a non-trivial exercise to build GNU Pascal on some non-Unix systems, so we also provide ready-to-run binaries for a number of platforms. (See [Section 4.2 \[Binary Distributions\]](#), page 29 for how to install a binary distribution.)

GPC is based on the GNU Compiler Collection, GNU CC or GCC. You will need the GCC sources to build it. It must be the same version as the one GPC is implemented with – 2. 8. 1, 2. 95. x, 3. 2. x, 3. 3. x or 3. 4. x as of this writing. Although you need GCC to build the GNU Pascal compiler, you don't need GCC to compile Pascal programs once GNU Pascal is installed. (However, using certain libraries will require compiling C wrappers, so it is a good idea to install the C compiler as well.)

Because GNU Pascal shares its back-end with GCC, it should run on any system supported by GCC. A full n0-a1(d)]TJIs. tF5110.909TfTdgC1rtledlby GC10.909scndlbe fundli

GPC . . . #endif', so they should not interfere when you build a C compiler from this source tree.

Note 2: The '--enable-languages=pascal' option means that we only want to build the Pascal compiler and not, for instance, the C++ compiler.

5 Command Line Options supported by G'U Pascal.

`--progress-messages`

Output source file names and line numbers while compiling.

`--no-progress-messages`

Do not output source file names and line numbers while compiling (default).

`--progress-bar`

Output number of processed lines while compiling.

`--no-progress-bar`

Do not output number of processed lines while compiling (default).

`--automake-gpc`

Set the Pascal compiler invoked by 6-17.361(m)270ic5210.909Tf-57.6-17.335Td[`--no-automake-`

--ignore-function-results

Do not complain when a function is called like a procedure.

--no-ignore-function-results

Complain when a function is called like a procedure (default).

--pointer-arithmetic

Enable pointer arithmetic.

--no-pointer-arithmetic

Disable pointer arithmetic (default).

--cstrings-as-strings

Treat CStrings as strings.

--no-cstrings-as-strings

Do not treat CStrings as strings (default).

-Wabsolute

Warn about variables at absolute addresses and 'out variables' variable with non-(v)

--short-circuit

- no-exact-compare-strings
Blank-pad strings for comparisons.
- double-quoted-strings
Allow strings enclosed in

- stack-checking
Enable stack checking (50same as '\$\$+}' (51.
- no-stack-checking
Disable stack checking (50same as '\$\$-' (50default t(51
- read-base-specifier
In read statements, allow input base specifier 'n#' (50default(51.
- no-read-base-specifier
In read statements, do not allow input base specifier 'n#' (50default in ISO 7185 Pascal(51.
- read-hex
In read statements, allow hexadecimal input with '\$' (50default(51.
- no-read-hex
In read statements, do not allow hexadecimal input with '\$' (50default in ISO 7185 Pascal(51.
- read-white-space
In read statements, require whitespace after numbers.
- no-read-white-space
In read statements, do not require whitespace after numbers (50default(51.
- write-clip-strings
In write statements, truncate strings exceeding their field width ('Write (SomeLongString : 3(51.
- no-write-clip-strings
Do not truncate strings exceeding their field width(0.360.909Tf367(a6-F5210.dt)-49J/F511ite-clie-

--typed-address

Make the result of the address operator typed (same as '{\$T+}', default).

--no-typed-address

-Winterface-file-name

Warn when a unit/module interface differs from the file name.

-Wno-interface-file-name

Do not warn when a unit/module interface differs from the file name (default).

--methods-always-virtual

Make all methods virtual (default when '--methods-always-virtual

--mac-pascal

Support (some features of) traditional Macintosh Pascal compilers.

--gnu-pascal

- static On systems that support dynamic linking, this prevents linking with the shared libraries, i.e. forces static linking. On other systems, this option has no effect.
- D*macro*[=*def*]
 Define the macro and conditional symbol *macro* as *def* (or as '1' if *def* is omitted).
- b *machine*
 The argument *machine* specifies the target machine for compilation. This is useful when you have installed GNU Pascal as a cross-compiler.
- v

end.

The programal


```
const  
    FiveFoo    = 5;
```


reserved word `value` can be replaced by `'='`, however `value` is not allowed in ISO-Pascal and Borland Pascal, and the replacement by `'='` is not allowed in Extended Pascal.

Type declaration example

type

```
var
  var_identifier: type_identifier value constant_expression;
  ...
  var_identifier: type_identifier value constant_expression;
or
var
  var_identifier: type_definition value constant_expression;
  ...
  var_identifier: type_definition value constant_expression;
```

6.1.6.2 The Function

`function` *function_identifier*: *function_result_*


```

case ordinal_expression of
  selector: statement;
  ...
  selector: statement;
otherwise { ''else'' instead of ''otherwise'' allowed }
  statement;
  ...
  statement;
end

```

or, as part of the invariant record type definition:

```

type
  foo = record
    field_declarations
  case bar: variant_type of
    selector: (field_declarations);
    selector: (field_declarations);
    ...
  end;

```

or, without a variant selector field,

```

type
  foo = record
    field_declarations
  case variant_type of
    selector: (field_declarations);
    selector: (field_declarations);
    ...
  end;

```

The case statement com(art)-es the value of *ordinal_expression* to each *selector*


```

{$Ldemomod3.pas} {explicitly link module}

{Manually do the "import" from DemoMod3}
type
  FooType= Integer;

procedure SetFoo(f: FooType); external name 'SetFoo';
function  GetFoo: FooType;      external name 'GetFoo';

begin
  SetFoo(999);
  WriteLn(GetFoo)
end.

```

Module initialization and finalization:

The `to begin` do module initialization and `to end` do module finalization constructs now

[\[MedInt\], page 361](#)

signed 32-bit integer type, '-2147483648..2147483647',
compatible to 'long int' in GNU C.

[\[MedCard\], page 360](#)

unsigned 32-bit integer type, '0..4294967295',
compatible to 'unsigned long int'

6.2.4 Built-in Real (Floating Point) Types

Ord applied to *name_identifier*

See also

[Section 6.2.11.7 \[Pointer Types\], page 76](#), [\[nil\], page 370](#)


```
program StringDemo (Output);

type
  SType = String (10);
  SPtr  = ^String;

var
  Str : SType;
  Str2: String (100000);
  Str3: String (20) value 'string expression';
  DStr: ^String;
  ZStr: SPtr;
  Len : Integer value 256;
  Ch  : Char value 'R';
```



```
begin
  Foo := Bar;  { Modify address which foo is holding }
  Foo^ := 5;   { Access data foo is pointing to }
end.
```


See also

[Section 6.8 \[OOP\], page 86](#)

6.2.11.10 Initial values to type denoters

```
    RestrictedRecord = restricted UnrestrictedRecord;

var
    r1: UnrestrictedRecord;
    r2: RestrictedRecord;
    i: restricted Integer;
    k: Integer;

function AccessRestricted (p: UnrestrictedRecord): RestrictedRecord;
var URes: UnrestrictedRecord;
begin
    { The parameter is treated as unrestricted, even though the actual
      parameter may be restricted }
    URes.a := p.a;
    { It is allowed to assign a function result }
    AccessRestricted := URes;
end;

begin
    r1.a := 354;

    { Assigning a restricted function result to a restricted variable }
```


-

6.3 Operators


```
begin
  s[8 .. 12] := 'folks';
  WriteLn (s) { yields 'Hello, folks!' }
end.
```


end;

Use 'FooParent. bar (z)' if you want to be sure that *this*


```

    P, N: 1 .. 100;
begin
    Rewrite (F);
    P := 42;
    N := 17;
    SeekWrite (F, P);
    Write (F, N)
end.

```

The following direct access routines may be applied to a direct access file:

```

SeekRead (F, N); { Open file in inspection mode, seek to record N }
SeekWrite (F, N); { Open file in generation mode, seek to record N }
SeekUpdate (F, N); { Open file in update mode, seek to record N }
Update (F); { Writes F^, position not changed. F^ kept. }
p := Position (F); { Yield the current record number }
p := LastPosition (F); { Yield the last record number in file }

```

If the file is open for inspection or update, Get may be applied. If the file is open for generation or update,

6.10.3 Accessing Command Line Arguments

GPC supports access to the command line arguments with the BP compatible `ParamStr` and `ParamCount` functions.

- `ParamStr[0]` is the program name,
- `ParamStr[1] .. ParamStr[ParamCount]` are the a309536.09TemStr[P2y6oThe program b.09Tlow acc.

$S2 > S1$ Comparison between two sets. Returns boolean result. True if True

6.11 Interfacing with Other Languages

The standardized GNU compiler back-end makes it relatively easy to share libraries between GNU Pascal and other GNU compilers. On Unix-like platforms (*not* on Dos-like platforms), the GNU compiler back-end usually complies to the standards defined for that system, so communication with other compilers should be easy, too.

- By default, GPC capitalizes the first letter (only) of each identifier, so 'procedure FooBAR' must be imported as 'extern voi d Foobar()' froo428.C.51

for Mac OS X:

`'/usr/share/locale' or '/sw/share/locale'`

for Linux, *BSD:

`'/usr/share/locale' or '/usr/local/share/locale'`

See also

[undefined \[Gettext\], page](#)


```
export
  GPC = all;
  GPC_CP = (ERead { @@ not really, but an empty export doesn't work
    } );
  GPC_EP = (ERead { @@ not really, but an empty export doesn't work
    } );
  GPC_BP = (MaxLongInt, ExitCode, ErrorAddr, FileMode, Pos);
  GPC_Delphi = (MaxLongInt, Int64, InitProc, EConvertError,
```



```

    CInteger; external name '_p_CStringChMod';
function CStringChOwn (FileName: CString; Owner: CInteger; Group:
    CInteger): CInteger; external name '_p_CStringChOwn';
function CStringUTime (FileName: CString; AccessTime: UnixTimeType;
    ModificationTime: UnixTimeType): CInteger; external
    name '_p_CStringUTime';

{ Constants for SeekHandle }
const
    SeekAbsolute = 0;
    SeekRelative = 1;
    SeekFileEnd  = 2;

{ Seek to a position on a file handle. }
function SeekHandle (Handle: CInteger; Offset: FileSizeType;
    W25({})]TJ0cCInteger; 025(CIntegerType;)]TJ11.4ernal name '_p_CStringCle .455-12.453

```

```

{ System routines }

{ Sets the process group of Process (or the current one if Process
  is 0) to ProcessGroup (or its PID if ProcessGroup is 0). Returns
  True if successful. }
function SetProcessGroup (Process: CInteger; ProcessGroup:
  CInteger): Boolean; external name '_p_SetProcessGroup';

{ Sets the process group of a terminal given by CInteger: (or its 0). Returns
  True if successful. }
function Set3rocessGroup (PrTd[(CIntocess:)-525(CInteger;)Handl eocessGroup:
  CInteger): external name '_p_SetProcessGroup';

```



```
function CStringEnd      (Src: CString): CString; attribute
```

```

function NewString      (const s: String) = Result: PString;
  attribute (name = '_p_NewString'); external;
procedure DisposeString (p: PString); external name '_p_Dispose';

procedure SetString     (var s: String; Buffer: PChar; Count:
  Integer); attribute (name = '_p_SetString'); external;
function StringOfChar   (ch: Char; Count: Integer) = s: TString;
  attribute (name = '_p_StringOfChar'); external;

procedure TrimLeft      (var s: String); attribute (name
  = '_p_TrimLeft'); external;
procedure TrimRight     (var s: String); attribute (name
  = '_p_TrimRight'); external;
procedure TrimBoth      (var s: String); attribute (name
  = '_p_TrimBoth'); external;
function TrimLeftStr    (const s: String) = Result: TString;
  attribute (name = '_p_TrimLeftStr'); external;
function TrimRightStr   (const s: String) = Result: TString;
  attribute (name = '_p_TrimRightStr'); external;
function TrimBothStr    (const s: String) = Result: TString;
  attribute (name = '_p_TrimBothStr'); external;
function LTrim          (const s: String) = Result: TString;
  external name '_p_TrimLeftStr';

function GetStringCapacity (const s: String): Integer; attribute
  (name = '_p_GetStringCapacity'); external;

{ A shortcut for a common use of WriteStr
function Integer2String (i: Integer) = s: Str64; attribute (name
  = '_p_Integer2String'); external;

{ Convert integer n to string in base Base.
function Integer2StringBase (n: LongestInt; Base:
  TInteger2StringBase): TString; attribute (name
  = '_p_Integer2StringBase'); external;

{ Convert integer n to string in base Base, with sign, optionally in
  uppercase representation with printed base, padded with

  specified Width.
function Integer2StringBaseExt (n: LongestInt; Base:
  TInteger2StringBase; Width: TInteger2StringWidth; Upper: Boolean;
  PrintBase: Boolean): TString; attribute (name
  = '_p_Integer2StringBaseExt'); external;

{ String handling routines (higher level), from string2.pas
type
  PChars0 = ^TChars0;
  TChars0 = array [0 .. div SizeOf(Char) - 1] of Char;

```



```
ExitCode: Integer; attribute (name = '_p_ExitCode'); external;  
{ Contains the address of the code where a runtime occurred, nil
```



```
{ Non-POSIX signals }  
SigTrap  : Integer; attribute (const); external name '_p_SIGTRAP';  
SigIOT   : Integer; attribute (const); external name '_p_SIGIOT';  
SigEMT   : Integer; attribute (const); external name '_p_SIGEMT';  
SigBus   : Integer; attribute (const); external name '_p_SIGBUS';  
SigSys   : Integer; attribute (const); external name '_p_SIGSYS';  
SigStkFlt: Integer; attribute (const); external  
name '_p_SIGSTKFLT';
```


Afterwards, the following optional modifiers may follow. Their meaning is locale-dependent, and many systems and locales just ignore them.

'E' Use the locale's alternate representation for date and time. In a Japanese locale, for example, '%Ex' might yield a date format based on the Japanese Emperors' reigns.

'O' Use the locale's alternate numeric symbols25(for)-525(exanumbers.)-525(Thi s)]

number (see 'V') belongs to the previous or next year, that year is used instead.

- 'h' The abbreviated month name according to the current locale. This is the same as 'b'.
- 'H' The hour as a decimal number, using a 24-hour clock ('00' .. '23').
1000' .. '23').

DirSeparator:	the separator of the directories within a full file name
DirSeparators:	a set of all possible directory and drive name separators
ExtSeparator:	the separator of a file name extension
DirRoot:	the name of the root directory
DirSelf:	the name of a directory in itself
DirParent:	the name of the parent directory
MaskNoStdDir:	a file name mask that matches all names except the standard directories DirSelf and DirParent
NullDeviceName:	the full file name of the null device
TtyDeviceName:	the full file name of the current Tty
ConsoleDeviceName:	the full file name of the system console. On Dos systems, this is the same as the Tty, but on systems that allow remote login, this is a different thing and may reach a completely


```

{ Remove all trailing DirSeparators from s, if there are any, as
  long as removing them doesn't change the meaning (i.e., they don't
  denote the root directory. }
function RemoveDirSeparator (const s: String) = Result: TString;
  attribute (na la25(don't)]TJ0-12.453Td[(denote)-525(the)-525(root)-525(di recn_p_ing,
  fucurrenion Removeondenot0Son Removeo

```



```
(iocritical, name = '_p_Execute'); external;  
function ExecuteNoTerminal (const CmdLine: String): Integer;  
  attribute (iocritical, name = '_p_ExecuteNoTerminal'); external;  
  
{ File handling routines, from files.pas }  
  
type  
  TextFile = Text;  
  TOpenMode = (fo_None, fo_Reset, fo_Rewrite, fo_Append,  
    fo_SeekRead, fo_SeekWrite, fo_SeekUpdate);  
  PAnyFile = ^AnyFile;
```



```
const
  NoChange = -1; { can be passed to ChOwn for Owner and/or Group to
    not change that value }

procedure CloseFile (var f: GPC_FDR); attribute (name
  = '_p_CloseFile'); external;
procedure ChMod (var f: GPC_FDR; Mode: Integer); attribute
  (iocritical, name = '_p_ChMod'); external;
procedure ChOwn (var f: GPC_FDR; Owner, Group: Integer); attribute
  (iocritical, name = '_p_ChOwn'); external;
```

```
end;
```



```

function IOSelectRead (const Files: array [m .. n: Natural] of
  PAnyFile; MicroSeconds: MicroSecondTimeType): Integer; attribute
  (name = '_p_IOSelectRead'); external;

{ Bind a filename to an external file }
procedure AssignFile (var t: AnyFile; const FileName: String);
  attribute (name = '_p_AssignFile'); external;
procedure AssignBinary (var t: Text; const FileName: String);
  attribute (name = '_p_AssignBinary'); external;
procedure AssignHandle (var t: AnyFile; Handle: Integer; CloseFlag:
  Boolean); attribute (name = '_p_AssignHandle'); external;

{ ge-521oure0tnBooi mtil e }

```

```
= '_p_OptionArgument'); external;  
UnknownOptionCharacter: Char; attribute (name  
= '_p_UnknownOptionCharacter'); external;  
GetOptErrorFlag      : Boolean; attribute (name  
= '_p_GetOptErrorFlag'); external;  
  
{ Parses command line arguments for options and returns the next  
one.
```

The special argument '--' forces an end of option-scanning

The following sections describe all units included with GPC (besides the 'GPC' module which

- A few features cannot be implemented in a portable way and are only available on some systems:

Sound, NoSound 1) -----.

"chmod u+s 'which SVGATextMo2e'", as root once, but only if you really want each user to be allowed to change the text mode.

- 6) Only on local consoles.
 - 7) Some terminals only. Most xterms etc. support it as well as other terminals that support an "alternate screen" in the smcup/rmcup terminal capabilities.
 - 8) Only with PDCurses, not with ncurses. Changing the number of screen *columns* doesn't work in a full-screen session.
- When CRT is initialized (automatically or explicitly; see the comments for CRTInit), the screen is cleared, and at the end of the program, the cursor is placed at the bottom of the screen (curses behaviour).
 - All the other things (including most details like color and function key constants) are compatible with BP's CRT unit, and there are many extensions that BP's unit does not have.

the resulting executable to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the executable file might be covered by the GNU General Public License.

Please also note the license of the curses library used. }

```
{ $gnu-pascal , l - }  
{ $if __GPC_RELEASE__ < 20030722 }
```



```
procedure ClrScr; external name 'crt_ClrScr';  
procedure ClrEOL; external name 'crt_ClrEOL';  
procedure InsLine; external name 'crt_e';  
procedure DelLine; external name 'crt_DelLine';  
procedure TextColor (Color: TTextAttr);  
procedure TextBackground (Color: TTextAttr);  
procedure LowVideo;
```

{ Changes the input and output file and the terminal description CRT


```
WindowMax: TWindowXY absolute WindMax;
```

```
{ The attribute set by NormVideo }
```

```
NormAttr: TTextAttr = 7; attribute (name = 'crt_NormAttr');
```

updates may occur more frequently (even if the update level is set to `UpdateNever`). About the default value, see the comments for `CRTInit`.

`UpdateNever` : never (unless explicitly requested with `CRTUpdate`)
`UpdateWaitInput`: before Delay and CRT input, unless typeahead is detected
`UpdateInput` : before Delay and CRT input
`UpdateRegularly`: before Delay and CRT input and otherwise in regular intervals without causing too much refresh. This uses a timer on some systems (currently, Unix with ncurses). This was created

```

function AltGrKey (ch: Char): TKey; external name 'crt_AltGrKey';
function ExtraKey (ch: Char): TKey; external name 'crt_ExtraKey';

{Char)eckCharifCharkCharisCharaCharpseudoCharkey generated byCharaChardeadlyCharsignal
function IsDeadlySignal (k: TKey): Boolean;

{CharProduceCharaCharbeepCharorCharaCharscreenCharflash }
procedureCharBeep; external name 'crt_Beep';
procedureCharFlash; external name 'crt_Flash';

{CharGetCharsize ofCharcurrentCharwindow (calculated usingCharGetWindow) }
function GetXMax: Integer;
function GetYMax: Integer;

{CharGet/gotoCharanCharabsolute position }
function W;
function W;
procedureCharFlash;

{CharGet/gotoTurnscreenCharolloGetWindow}

```



```
TTextAttr); external name 'crt_ReadChar';  
  
{ Change only text attributes, leave characters. Truncated at the  
  right margin. }  
procedure ChangeTextAttr (x, y, Count: Integer; NewAttr: TTextAttr);  
  
{ Fill current window }
```

```

function GetActivePanel: TPanel; external
  name 'crt_GetActivePanel';
procedure PanelNew (x1, y1, x2, y2: CInteger;
  BindToBackground: Boolean); external name 'crt_PanelNew';
procedure PanelDelete (Panel: TPanel); external
  name 'crt_PanelDelete';
procedure PanelBindToBackground (Panel: TPanel; BindToBackground:
  Boolean); external name 'crt_PanelBindToBackground';
function PanelIsBoundToBackground (Panel: TPanel): Boolean;
  external name 'crt_PanelIsBoundToBackground';
procedure PanelActivate (Panel: TPanel); external
  name 'crt_PanelActivate';
procedure PanelHide (Panel: TPanel); external
  name 'crt_PanelHide';
procedure PanelShow (Panel: TPanel); external
  name 'crt_PanelShow';
function PanelHidden (Panel: TPanel): Boolean;
  external name 'crt_PanelHidden';
procedure PanelTop (Panel: TPanel); external
  name 'crt_PanelTop';
procedure PanelBottom (Panel: TPanel); external
  name 'crt_PanelBottom';
procedure PanelMoveAbove (Panel, Above: TPanel); external
  name 'crt_PanelMoveAbove';
procedure PanelMoveBelow (Panel, Below: TPanel); external
  name 'crt_PanelMoveBelow';
function PanelAbove (Panel: TPanel): TPanel; external
  name 'crt_PanelAbove';
function PanelBelow (Panel: TPanel): TPanel; external
  name 'crt_PanelBelow';

{ TPCRT compatibility }

{ Write a string at the given position without moving the cursor.
  Truncated at the right margin. }
procedure WriteString (const s: String; y, x: Integer);

{ Write a string at the given position with the given attribute
  without moving the cursor. Truncated at the right margin. }
procedure FastWriteWindow (const s: String; y, x: Integer; Attr:
  TTextAttr);

{ Write a string at the given absolute position with the given
  attribute without moving the cursor. Truncated at the right
  margin. }
procedure FastWrite (const s: String; y, x: Integer; Attr:
  TTextAttr);

{ WinCRT compatibility }

```

```
const
  cw_UseDefault t = Integer ($8000);

var
  { Ignored }
```



```
DosError_IOError      = 29;  
DosError_ReadFault    = 30;  
  
type
```



```
    GetDate (not the RTS date/time routines), and only for this
    process, not for child processes or even the parent process or
    system-wide. }
procedure SetDate (Year, Month, Day: Word);
procedure SetTime (Hour, Minute, Second, Sec100: Word);
{$endif}
```

6.15.3 Overcome some differences between Dos and Unix

The following listing contains the interface of the DosUnix unit.

This unit is there to overcome some of those differences between Dos and Unix systems that are not automatically hidden by GPC and the Run Time System. Currently features translation of bash style input/output redirections ('


```

    the ISO-8859-1 (Latin1) character set }
function OEM2Latin1 (ch: Char): Char;
function OEM2Latin1Str (const s: String) = r: TString;

{ Translates a character from the ISO-8859-1 (Latin1) character set
  to the "OEM" characters used under Dos }
function Latin12OEM (ch: Char): Char;
function Latin12OEMStr (const s: String) = r: TString;

```

6.15.4 Higher level file and directory handling

The following listing contains the interface of the FileUtils unit.

```

be2.453Td[fulreeeSomfreththna34(of)ity07Td[(/cUth50cLicen525(thMERCHANTABILITYYeeeeSor

```

```
unit FileUtils;  
  
interface  
  
uses GPC;  
  
type  
  TStringProc = procedure (const s: String);  
  
{ Finds all files matching the given Mask in the given Directory and  
  all subdirectories of it. The matching is done using all wildcards
```


Author: Frank Heckenbach <frank@pascal.gnu.de>

This file is part of GNU Pascal.

```

{ Force the use of 64-bit limbs for all 64-bit MIPS CPUs if ABI
  permits. }
{$define _LONG_LONG_LIMB}
{$endif}

type
  {$ifdef _SHORT_LIMB}
    mp_limb_t      = CCardinal;
    mp_limb_signPS
  {$elif defined (_LONG_LONG_LIMB)}
    mp_limb_t      = LongCard;
    mp_limb_signPS
  {$else}
    mp_limb_t      = MedCard;
    mp_limb_signPS
  {$endif}

mp_ptr

  {$if defined (_CRAY) and not defined (_CRAYMPP)}
    mp_size_t      = Ce
    mp_exp_t
  {$else}
    mp_size_t      = Mede
    mp_exp_t
  {$endif}

mpz
  mp_alloc,
  mp_size: Tdhi (use)Neger;
  mp_d: Td2100(mp_ptr)]TJ-11.454-12.453Td[(end;)]TJ0-24.907Td[(mpz)array_ptr

en4;

```



```

    mpz_t); external name '__gmpz_sqrt';
procedure mpz_sqrtrem      (var Dest, DestR: mpz_t; protected
    var Src: mpz_t); external name '__gmpz_sqrtrem';
function  mpz_perfect_square_p (protected var Src: mpz_t): CInteger;
    external name '__gmpz_perfect_square_p';

function  mpz_probab_prime_p  (protected var Src: mpz_t;
    Repetitions: CInteger): CInteger; external
    name '__gmpz_probab_prime_p';
procedure mpz_gcd          (var Dest: mpz_t; protected var Src1,
    Src2: mpz_t); external name '__gmpz_gcd';
function  mpz_gcd_ui       (var Dest: mpz_t; protected var Src1:
    mpz_t; Src2: MedCard): MedCard; external name '__gmpz_gcd_ui';
procedure mpz_gcdext       (var Dest, DestA, DestB: mpz_t;
    protected var SrcA, SrcB: mpz_t); external name '__gmpz_gcdext';
function  mpz_invert       (var Dest: mpz_t; protected var Src,
    Modulus: mpz_t): CInteger; external name '__gmpz_invert';
function  mpz_jacobi       (protected var Src1, Src2: mpz_t):
    CInteger; external name '__gmpz_jacobi';

function  mpz_cmp          (protected var Src1, Src2: mpz_t):
    CInteger; external name '__gmpz_cmp';
function  mpz_cmp_ui       (protected var Src1: mpz_t; Src2:
    MedCard): CInteger; external name '__gmpz_cmp_ui';
function  mpz_cmp_si       (protected var Src1: mpz_t; Src2:
    MedInt): CInteger; external name '__gmpz_cmp_si';
function  mpz_sgn          (protected var Src: mpz_t): CInteger;
    attribute (inline);

procedure mpz_and          (var Dest: mpz_t; protected var Src1,
    Src2: mpz_t); external name '__gmpz_and';
procedure mpz_ior          (var Dest: mpz_t; protected var Src1,
    Src2: mpz_t); external name '__gmpz_ior';
procedure mpz_com          (var Dest: mpz_t; protected var Src:
    mpz_t); external name '__gmpz_com';
function  mpz_popcount     (protected var Src: mpz_t): MedCard;
    external name '__gmpz_popcount';
function  mpz_hamdist      (protected var Src1, Src2: mpz_t):
    MedCard; external name '__gmpz_hamdist';
function  mpz_scan0        (protected var Src: mpz_t;
    StartingBit: MedCard): MedCard; external name '__gmpz_scan0';
function  mpz_scan1        (protected var Src: mpz_t;
    StartingBit: MedCard): MedCard; external name '__gmpz_scan1';
procedure mpz_setbit       (var Dest: mpz_t; BitIndex: MedCard);
    external name '__gmpz_setbit';
procedure mpz_clrbit       (var Dest: mpz_t; BitIndex: MedCard);
    external name '__gmpz_clrbit';

procedure mpz_random       (var Dest: mpz_t; MaxSize:
    mp_size_t); external name '__gmpz_random';
procedure mpz_random2      (var Dest: mpz_t; MaxSize:

```

```

    mp_size_t); external name '__gmpz_random2';
function mpz_sizeinbase      (protected var Src: mpz_t; Base:
    CInteger): SizeType; external name '__gmpz_sizeinbase';

{ Rational (i.e. Q) routines }

procedure mpq_canonicalize    (var Dest: mpq_t); external
    name '__gmpq_canonicalize';

procedure mpq_init            (var Dest: mpq_t); external
    name '__gmpq_init';
procedure mpq_clear           (var Dest: mpq_t); external
    name '__gmpq_clear';
procedure mpq_set              (var Dest: mpq_t; protected var Src:
    mpq_t); external name '__gmpq_set';
procedure mpq_set_z            (var Dest: mpq_t; protected var Src:
    mpz_t); external name '__gmpq_set_z';
procedure mpq_set_ui           (var Dest: mpq_t; Nom, Den: MedCard);
    external name '__gmpq_set_ui';

```



```

    mpq_t); external name '__gmpq_get_num';
procedure mpq_get_den      (var Dest: mpz_t; protected var Src:
    mpq_t); external name '__gmpq_get_den';

{ Floating point (i.e. R) routines }

procedure mpf_set_default_prec (Precision: MedCard); external
    name '__gmpf_set_default_prec';
procedure mpf_init          (var Dest: mpf_t); external
    name '__gmpf_init';
procedure mpf_init2        (var Dest: mpf_t; Precision:
    MedCard); external name '__gmpf_init2';
procedure mpf_clear        (var Dest: mpf_t); external
    name '__gmpf_clear';
procedure mpf_set_prec     (var Dest: mpf_t; Precision:
    MedCard); external name '__gmpf_set_prec';
function  mpf_get_prec     (protected var Src: mpf_t): MedCard;
    external name '__gmpf_get_prec';
procedure mpf_set_prec_raw (var Dest: mpf_t; Precision:
    MedCard); external name '__gmpf_set_prec_raw';

procedure mpf_set          (var Dest: mpf_t; protected var Src:
    mpf_t); external name '__gmpf_set';
procedure mpf_set_ui       (var Dest: mpf_t; Src: MedCard);
    external name '__gmpf_set_ui';
procedure mpf_set_si       (var Dest: mpf_t; Src: MedInt);
    external name '__gmpf_set_si';
procedure mpf_set_d        (var Dest: mpf_t; Src: Real);
    external name '__gmpf_set_d';
procedure mpf_set_z        (var Dest: mpf_t; protected var Src:
    mpz_t); external name '__gmpf_set_z';
procedure mpf_set_q        (var Dest: mpf_t; protected var Src:
    mpq_t); external name '__gmpf_set_q';
function  mpf_set_str      (var Dest: mpf_t; Src: CString; Base:
    CInteger): CInteger; external name '__gmpf_set_str';

procedure mpf_init_set     (var Dest: mpf_t; protected var Src:
    mpf_t); external name '__gmpf_init_set';
procedure mpf_init_set_ui  (var Dest: mpf_t; Src: MedCard);
    external name '__gmpf_init_set_ui';
procedure mpf_init_set_si  (var Dest: mpf_t; Src: MedInt);
    external name '__gmpf_init_set_si';
procedure mpf_init_set_d   (var Dest: mpf_t; Src: Real);
    external name '__gmpf_init_set_d';
function  mpf_init_set_str (var Dest: mpf_t; Src: CString; Base:
    CInteger): CInteger; external name '__gmpf_init_set_str';

function  mpf_get_d        (protected var Src: mpf_t): Real;
    external name '__gmpf_get_d';

```

```

mp_exp_t; Base: CInteger;
                                NumberOfDigits: SizeType; protected
var Src: mpf_t); CString; external name '__gmpf_get_str';

procedure mpf_add                (var Dest: mpf_t; protected var Src1,
    Src2: mpf_t); external name '__gmpf_add';
procedure mpf_add_ui             (var Dest: mpf_t; protected var Src1:
    mpf_t; Src2: MedCard); external name '__gmpf_add_ui';
procedure mpf_sub                (var Dest: mpf_t; protected var Src1,
    Src2: mpf_t); external name '__gmpf_sub';
procedure mpf_ui_sub             (var Dest: mpf_t; Src1: MedCard;
    protected var Src2: mpf_t); external name '__gmpf_ui_sub';
procedure mpf_sub_ui             (var Dest: mpf_t; protected var Src1:
    mpf_t; Src2: MedCard); external name '__5(mpf_ui_sub';)]TJ-11.455-12.453Td[(procedu
    Src2: mpf_t); external name '_mului_sub';
procedure                        (var Dest: mpf_t; protected var Src1:
    mpf_t; Src2: MedCard); external name '_mulpf_ui_sub';
procedure                        (var Dest: mpf_t; protected var Src1,
    Src2: mpf_t); external name '_divpf_sub';
procedure mdivpf_sub            (var Dest: mpf_t; Src1: MedCard;
    protected var Src2: mpf_t); external name '__gmdivpf_sub';
procedure                        (var Dest: mpf_t; protected var Src1:
    mpf_t; Src2: MedCard); external name '_divpf_ui_sub';
procedure                        (var Dest: mpf_t; protected varSrc1:
    external name '_5qrtui_sub';
procedure                        (var Dest: mpf_t; Src: MedCardSrc1:
procedure                        (var Dest: mpf_t; protected varSrc1:
    external name '_negpf_add';
procedure                        (var Dest: mpf_t; protected varSrc1:
    external name '_5bsui_sub';
procedure                        (var Dest: mpf_t; protected var Src1:
    mpf_t; Src2: MedCard); external name '_mulp2d[(pf_add';)]TJ-11.455-12.454Td[(procedu
    mpf_t; Src2: MedCard); external name '_divp2d[(et_str';)]TJ-11.455-24.90function
    NuBrOfDigits: MedCa_t): CInt); external name '_equi_sub';
procedure                        (var Dest: mpf_t; protected var Src1,
    Src2: mpf_t); external name '_reldiffpf_add';

```



```

    HAVE_GMP4} protected var {$endif} m: mpz_t); external
    name '__gmp_randinit_lc';

    HAVE_GMP4} protected var {$endif}m: mp; varm: name '__gmp_randinit_lc_2_lc';
    name '__gmp_rsec_lc';
    procedure gmp_rsec_ui cal 150525((var)-525(State:)-525(gmp_rs5(St_t; var)-5Sec:c';)]J-11.45

```

```
    name ' __gmpz_swap' ;  
function mpz_tdiv_ui      (protected var Src1: mpz_t; Src2:
```



```

{$endif}

module GPCUtil;

export GPCUtil = all
(
  { Return the current working directory }
  GetCurrentDirectory => ThisDirectory,

  { Does a directory exist? }
  DirectoryExists => IsDirectory,

  { Does file name s exist? }
  FileExists => ExistFile,

  { Return just the directory path of Path. Returns
    DirSelf + DirSeparator if Path contains no directory. }
  DirFromPath => JustPathName,

  { Return just the file name part without extension of Path.
    Empty if Path contains no file name. }
  NameFromPath => JustFileName,

  { Return just the extension of Path. Empty if Path contains
    no extension. }
  ExtFromPath => JustExtension,

  { Return the full pathname of Path }
  FExpand => FullPathName,

  { Return a ofDoes }

  { Return a ofDoes }

  { Return a ofDoes }

  {part a }

  {part a }

```



```

function Int2PChar (i: Integer): PChar;

{ Convert Integer to string }
function Int2Str (i: Integer) = s: TString;

{ Convert string to Integer }
function Str2Int (const s: String; var i: Integer): Boolean;
    attribute (ignorable);

{ Convert string to LongInt }
function Str2Long (const s: String; var i: LongInt): Boolean;
    attribute (ignorable);

{ Convert string to Double }
function Str2Real (const s: String; var i: Double): Boolean;
    attribute (ignorable);

{ Return a string right-padded to length Len with ch }
function PadCh (const s: String; ch: Char; Len: Integer) = Padded:
    TString;

{ Return a string right-padded to length Len with spaces }
function Pad (const s: String; Len: Integer): TString;

{ Return a string left-padded to length Len with ch }
function LeftPadCh (const s: String; ch: Char; Len: Integer) = Padded:
    TString;

{ Return a string left-padded to length Len with blanks }
function LeftPad (const s: String; Len: Integer): TString;

{ Uniform access to big memory blocks for GPC and BP. Of course, for

```


MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License

This unit provides the powerful mechanism of national language support by accessing '.mo' files and locale information.

It includes:

locales (not xlocales) and libintl.


```
function SetLocale (Category: Integer; const Locale: String):  
    TString; attribute (ignorable);  
  
{ Set and/or return the current locale. Same as above, but returns  
  a CString. }  
function SetLocaleC (Category: Integer; const Locale: String):  
    CString; attribute (ignorable);
```



```
function OvrGetBuf: LongInt;  
procedure OvrSetRetry (Size: LongInt);  
function OvrGetRetry: LongInt;  
procedure OvrClearBuf;
```


failure.

On systems that don't support waiting for a particular child, PID is ignored. On systems like MSDOS that don't really multitask PWait is just a mechanism to provide a consistent interface for the caller. }

```
function PExecute (ProgramName: CString; Arguments: PCStrings; var  
    ErrMsg: String; Flags: Integer): Integer; attribute (ignorable);  
function PWait st2aD51: Integer; Integer; Flags: Integer)r
```

reasons why the executable file might be covered by the GNU
General Public License. }

```
{ $gnu-pascal, I - }
{ $if __GPC_RELEASE__ < 20030303 }
{ $error This unit requires GPC release 20030303 or newer. }
{ $endif }
{ $ifndef __i386__ }
{ $error The Ports unit is only for the IA32 platform }
{ $endif }
```

unit Ports;

interface

```
{ Port access functions }
function InPortB (PortNumber: ShortWord): Byte;
function InPortW (PortNumber: ShortWord): ShortWord;
procedure OutPortB (PortNumber: ShortWord; aValue: Byte);
procedure OutPortW (PortNumber, aValue: ShortWord);
```

```
  prsseorts; coverede curase isrivfilgres
  getting access thePortsforrogramse.
}
```

```
{ $ifdef
```

```
  nameW
```

```
function OPLW Level: ttributeW nameW
function is ttributeW nameW
```

```
Public525=Wfunction OPLW Level: ttribuexJ0n5(Public525=W)-525'dl'0-12.453Td[(function)-1050e is
```


Author: Frank Heckenbach <frank@pascal.gnu.de>

This file is part of GNU Pascal.

GNU Pascal is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your

```
{ Optional command line parameters for the printer program.  
  Ignored when nil. }  
PrinterArguments: PPStrings = nil;  
  
{ How to deal with the printer spooler after the printer pipe is  
  closed, cf. the Pipes unit:er}
```

are similar to wild cards used in file name matching, but much more powerful.

There are two kinds of regular expressions supported by this unit, basic and extended regular expressions. The difference between them is not functionality, but only syntax. The following is a short overview of regular expressions. For a more thorough explanation see the literature, or the documentation of the rx library, or man pages of programs like grep(1) and sed(1).

Basic	Extended	Meaning
'.'	'.'	matches any single character
'Baad-z]'	'Baad-z]'	matches either 'a', 'e', or any character from 'i' to 'z'
'B^aad-z]'	'B^aad-z]'	matches any character but 'a', 'e', or 'i' .. 'z'
		To include in such a list the the characters ']', '^', or '-', put them first, anywhere but first, or first or last, resp.
'B[:al num:]]'	'B[:al num:]]'	matches any alphanumeric character
'B^[:di gi t:]]'	'B^[:di gi t:]]'	matches anything but a digit
'Ba[:space:]]'	'Ba[:space:]]'	matches the letter 'a' or a space character (space, tab)
...		(there are more classes available)
'\w'	'\w'	= B[:al num:]]
'\W'	'\W'	= B^[:al num:]]
'^'	'^'	matches the empty string at the beginn[(beginn[eTd[(25(a)-525ne' z')] TJ-183. 27-12. 45
'?34W'	'^'	matches or he


```
function FindSubExpressionReferences (var RegEx: RegExType; const  
    s: String; OnlySub: Boolean): Integer;  
  
{ Replaces (sub)expression references in ReplaceStr by the actualType; const
```



```

    would be longer than the capacity of Dest. }
function StrReadWord (const s: String; var i: Integer; var Dest:
    String): Boolean; attribute (ignorable);

{ Check that a certain string is contained in s (after possible
  space characters). }
function StrReadConst (const s: String; var i: Integer; const
    Expe1.4d: String) = Res: Boolean; attribute (ignorable);

{ A simpler to use version of StrReadConst that expects a ', '. }
function StrReadComma (const s: String; var i: Integer) = Res:
    Boolean; attribute (ignorable);

{ Read an integer number from a string. }
function StrReadInt (const s: String; var i: Integer; var Dest:
    Integer): Boolean; attribute (ignorable);

{ Read a real number from a string. }
function StrReadReal (const s: String; var i: Integer; var Dest:
    Real): Boolean; attribute (ignorable);

{ Read a Boolean value, represented by a single character
  from CharactersTrue or CharactersFalse (cf. Char2Boolean), from a
  string. }
function StrReadBoolean (const s: String; var i: Integer; var Dest:
    Boolean): Boolean; attribute (ignorable);

{ Read an enumerated value, i.e., one of the entries of IDs, from a
  string, and stores the ordinal value, i.e., the index in IDs
  (always zero-based) in Dest. }
function StrReadEnum (const s: String; var i: Integer; var Dest:
    Integer; const IDs: array of PString): Boolean; attribute
    (ignorable);

{ String hash table }

const
    DefaultHashSize = 1403;

type

```


taken care of so they work.

If `'__BP_RANDOM__'` is defined (`'-D__BP_RANDOM__'`), this unit will provide an exactly BP compatible pseudo random number generator. In particular, the range for integer randoms will be truncated to


```
function SSeg: PtrWord;
function SPtr: PtrWord;
```

```
{ Routines to handle BP's 6 byte 'Real' type which is formatted like
  this:
```

```
47                                     0
-|-----|-----|-----|-----|-----|-----|
47 ite                                6 iasted
```

```
is formas
The irsas The is
tois is The
```

```
GP,d T(is)-525(type)-525(is)-525represented The type which is
to BP's typ,s Thre(foeo)-525bhe
```

```
The functiose 'ReaToBP' Real '
T(is)-525(type)-525hans GP'se 'Real' typ.s Tat,e
```

```
wmas to The The
```

```
{ The
```

```
{is is BP's
```

```
functio{'ReaToBP' R1.s
functios TheThe
```

```
    HeapErrorNil      = 1;  
    HeapErrorRetry    = 2;  
  
var
```



```

    General Public License. }

{$gnu-pascal, I-}
{$if __GPC_RELEASE__ < 20030412}
{$error This unit requires GPC release 20030412 or newer.}
{$endif}

unit TFDD;

interface

uses GPC;

{ Write to multiple files. Everything written to Dest after calling
  this procedure will be written to both File1 and File2. Can be
  chained. }
procedure MultiFileWrite (var Dest, File1, File2: AnyFile);

```

6.15.19 Trap runtime errors

The following listing contains the source code of the Trap unit.

possible after a trapped error (perhaps by telling the user to
 save the data, then `ter1`
`save -D137` `redata, theth12.453Td[(save)-5JET1ctterror tnhes4-24136ve`


```
{gnu-pascal , I -}
```


7 A QuickStart Guide from Borland Pascal to GNU Pascal.

- declare a variable as `'String`

7.2.5 Real type

GPC does not support BP's 6-byte 'Real' type. It supports 'Single', 'Double' and 'Extended' which, at least on the GNU IA32 GNU and some GNU other processors, are GNU compatible to BP.

For BP's 6-byte 'Real' type, GPC's 'System' unit provides an emulation, called 'BPreal', as well as conversion routines to GPC's 'Real' type (which is the same as 'Double'), called 'RealToBPreal' and 'BPrealToReal'. You'll probably only need them when reading or writing binary files containing values of the BP 6-byte real type. There are no operators (e.g., '+') available for 'BPreal', but since GPC supports operator overloading, you could define them yourself (e.g., convert to 'Real', do the operation, and convert back). Needless to say that this is very inefficient and should not be done for any serious computations. Better convert your

File Device Drivers

Especially for strings, there are ready-made procedures like 'ReadStringBigEndian' or


```
C:\GNU-PAS> GPC HELLO.PAS -O HELLO.EXE
```

GPC is a very quiet compiler and doesn't print anything on the screen unless you request it

Foo

end.

For records and arrays, GPC supports both BP style and Extended Pascal style initializers.


```
begin
  a := Succ (a, 587md
  WriteLI (y8Too:=)e-{o:=58-egin
```


end.

- Borland's real (floating point) types are supported except for the 6-byte software Real type (but the 'System' unit provides conversion routines for it). GNU Pascals's 'Real' type has 8 bytes on the IA32 and is the same as 'Double

7.20 Special Parameters

- Untyped reference parameters can be denoted by
 `procedure Foo (var x);`

- Inline: GNU Pascal allows “inline” Pascal procedures and functions, while Borland Pascal

stop here: Certainly the API of the OS is **not** ideal for your program's needs. Just create more

8 The Alphabetical GPC Language Reference

```
WriteLn (Abs (42));           { 42 }  
WriteLn (Abs (-42));          { 42 }  
WriteLn (Abs (-12.1) : 0 : 1); { 12.1 }  
i1 := Cmplx (1, 1);           { 1 + i }  
WriteLn (Abs (i1) : 0 : 3)     { 1.414, i.e. Sqrt (2) }  
end.
```

See also

abstract

Not yet implemented.

Synopsis

See also

[Section 6.3 \[Operators\], page 82.](#)

AlignOf**Synopsis**

```
function AlignOf (var x): Integer;
```

Description

See also

[Chapter 9 \[Keywords\], page 453.](#)

and**Synopsis**

```

    operator and (operand1, operand2: Boolean) = Result: Boolean;
or
    operator and (operand1, operand2: integer _type) = Result: integer _type;
or
    procedure and (var operand1: integer _type; operand2: integer _type);

```

Description

In GNU Pascal, 'and' has three buildmean31.1(gs)-1(:)]TJ-108.316-14.856Td[(1.)-660(Logical)-407("and)1(")

tree operator in GPC: If the first operand is 'False', the

`c := 1`

`and_then`

Synopsis

```
operator and_then (operand1, operand2: Boolean) = Result: Boolean;
```

Desc2799ipt2799io2799en

Desc2799ipt2799io2799en {line 91, file /usr/share/doc/gnu-pascal/2799io2799en/Desc2799ipt2799io2799en.tex, line 91, file /usr/share/doc/gnu-pascal/2799io2799en/Desc2799ipt2799io2799en.tex}

with different block size or a typed file of a type with size not equal to one). This is the only way to reliably read/write a certain amount of data from/to an 'AnyFile'.

'AnyFile' pointers cannot be allocated with 'New' (because it would be unspecified which kind of file to create).

Conforming to

'AnyFile' is a GNU Pascal extension.

Example

```

program AnyFileDemo;

procedure Test (var f: AnyFile);
var v: ^AnyFile;
begin
  { Generic file operations are allowed for 'AnyFile' }
  Rewrite (f);

  { 'AnyFile' can also be accessed via pointers }
  v := @f;
  Close (v^);
end;

var
  t: Text;
  f: file;
  g: file of Integer;

begin
  { Any kind of file variable can be passed as 'AnyFile' }
  Test (t);
  Test (f);
  Test (g)
end.
```

See also

[\[Text\]](#), [page 429](#), [\[file\]](#), [page 322](#).

Append

Synopsis

```

procedure Append (var F: any_
```

```

; [FileName: String;]
```

Like 'Rewrite', 'Reset' and 'Extend' do, 'Reset' accepts an optional second parameter for the name of the file in the filesystem and a third parameter for the block size of the file. The third parameter is allowed only (and by default also required) for untyped files. For details, see [\[Rewrite\]](#), page 405.

Conforming to

'Append', including the 'BlockSize' parameter, is a Borland Pascal extension. ISO 10206 Extended Pascal has [\[Extend\]](#), page 318 instead. The 'FileName' parameter is a GNU Pascal extension.

Example

```
program AppendDemo;
var
  Sample: Text;
begin
  Assign (Sample, 'sample.txt');
  Rewrite (Sample);
  WriteLn (Sample, 'Hello, World!'); { 'sample.txt' now has one line }
  Close (Sample);

  { ... }

  Append (Sample);
  WriteLn (Sample, 'Hello again!'); { 'sample.txt' now has two lines }
  Close (Sample)
end.
```

See also

[\[Assign\]](#), page 272, [\[Reset\]](#), page 402

array

Synopsis

In type definitions:

array [*index_type*] of *element_type*

or

array [*index_type*, ..., *index_type*] of *element_type*

In parameter list declarations:

Description *element_type*

The reserved word 'array' is used to define an array type.

Conforming to conforming to open arrays

Array types are defined in ISO 7185 Pascal.

Example

```
program ArrayDemo;
type
  IntArray = array [1 .. 20] of Integer;
  WeekDayChars = array [(Mon, Tue, Wed, Thu, Fri, Sat, Sun)] of Char;
  Foo = array [0 .. 9, 'a' .. 'z', (Baz, Glork1, Fred)] of Real;
  TwoDimIntArray = array [1 .. 10] of IntArray;
  { is equivalent to: }
  TwoDimIntArray2 = array [1 .. 10, 1 .. 20] of Integer;

procedure PrintChars (F: array of Char);
var
  i: Integer;
begin
  for i := Low (F) to High (F) do
    WriteLn (F[i])
  end;

var
  Wal do: WeekDayChars;

begin
  Wal do := 'Hi World';
  PrintChars (Wal do)
end.
```

See also

[Chapter 9 \[Keywords\], page 453](#), [Section 6.2.11.2 \[Array Types\], page 70](#), [\[High\], page 333](#)
[\[Low\], page 356](#)

Example

See also

[\[Reset\]](#), [page 402](#), [\[Rewrite\]](#), [page 405](#), [\[Update\]](#), [page 441](#), [\[Extend\]](#), [page 318](#), [\[Append\]](#), [page .66](#).

Assigned

(Under construction.)

Synopsis

```
function Assigned (p: Pointer): Boolean;  
or  
function Assigned (p: procedural_type): Boolean;
```

Description

The 'Assigned' function returns 'True

attribute

(Under construction.)

Synopsis

declaration

Conforming to

'begin' is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example


```

                                in this case }
Special ,                      { Binding points to an existing
                                special file (device, pipe, socket,
                                etc.); 'Existing' is False in
                                this case }
SymLink: Boolean;              { Binding points to a symbolic link }
Size: FileSizeType;           { Size of file, or -1 if unknown }
AccessTime,                    { Time of last access }
Modifi cati ol o-5253-5212. 45325(or)-55(Time)-525(of)-525(l ast)-ChangeTd[5(l ink)Uni x

```


Description

'Byte' is an unsigned integer type which is one "unit" wide. On most platforms one unit has

See also

[Section 6.2.9 \[Boolean \(Intrinsic\)\], page 69](#), [\[Boolean\], page 280](#), [\[True\], page 434](#)

Example

```
program CardDemo;
var
  Foo: set of 1 .. 100;
begin
  Foo := [1, 2, 3, 5, 1, 1, 1, 2, 2, 2, 3, 3, 5, 5]; { four elements }
  WriteLn ('foo consists of', Card (Foo), ' elements')
end.
```

See also

[\[set\]](#), [page 412](#)

Cardinal

Synopsis

```
type
  Cardinal { built-in type }
```

Description

'Cardinal' is the "natural" unsigned integer type in GNU Pascal. On some platforms it is 32

case

Synopsis

```
case expression of  
  selector: statement;  
  ...  
  selector: statement;  
end;
```

or, with alternative statement sequence:

```
case expression of  
  selector: statement;  
  ...  
  selector: statement;  
otherwise { ''else'' instead of ''otherwise'' is allowed }  
  statement;  
  ...  
  statement;  
end;
```

or, as part of the invariant record type definition:

```
foo = record  
  field_  
  ...  
end;
```



```
    a := True;  
    if Ord (a) = 1 then WriteLn ('Ord (True) = 1')  
end.
```

See also

Description

The built-in type 'Char' holds elements of the operating system's character set '50usually ASCII'51. The 'Char' type is a special case of ordinal type. Conversion bion b(on)8:-4na(of)36geb(oner2.909

```
    else  
      WriteLn (' Okay. ' )  
end.
```

See also

[\[MkDir\]](#), page 364[JET100198.184.222651.174cm0001k1001-184.222-651.174cmBT/F5110.909Tf184.22265](#)

['Chr'eed' Paceln\[p333\(P\)98\(as\)-1\(cal\)-333\(v\)5](#)

I

Example

See also

Description

'CompilerAssert' checks the given 'Condition'

Description

Conforming to

'Complex' is an ISO 10206 Extended Pascal extension. **Example**

```
program ComplexDemo;
var
  a: Complex;
begin
  a := Cmplx (42, 3);
  WriteLn (Re (a), ' + ', Im (a), ' i')
end. See also Concat (Under construction.)
```

Synopsis

```
function Concat (S1, S2: String): String;
or
function Concat (S1, S2, S3: String): String;
or
... Description
```

Conforming to

'Concat' is a UCSD Pascal extension. **Example** **See also** Conjugate

Synopsis

```
function Conjugate (z: Complex): Complex;
```

Description

'Conjugate' computes the complex conjugate of the complex number 'z'

Conforming to

'Conjugate' is a GNU Pascal extension.

Example

```
program ConjugateDemo;
var
  z: Complex;
begin
  z := Cmplx (2, 3); { z is 2 + i * 3 }
  WriteLn ('z = ', Re (z) : 0 : 5, ' + i * ', Im (z) : 0 : 5);
  z := Conjugate (z); { z conjugate is 2 - i * 3 }
  WriteLn ('z conjugate = ', Re (z) : 0 : 5, ' + i * ', Im (z) : 0 : 5)
end.
```

See also

Example

```

program ConstDemo;

type
  Rec = record
    x: Integer;
    y: Integer;
  end;

const
  a = 5;
  constr: Rec = (10, 12);

procedure doIt (const r: Rec; const s: String);
begin
  WriteLn (r.x);
  WriteLn (r.y);
  WriteLn (s);
end;

var
  variabler: Rec;

begin
  variabler.x := 16;
  variabler.y := 7;
  doIt (variabler, 'Should be 16 and 7');
  doIt (constr, 'Should be 10 and 12');
end.

```

See also

[Chapter 9 \[Keywords\], page 453](#), [\[var\], page 444](#), [\[protected\], page 391](#), [Section 6.1.6.4 \[Subroutine Parameter List Declaration\], page 53](#).

constructor

(Under construction.) ;–)

Synopsis

Description

Object constructor.

Conforming to

‘constructor’ is an Object Pascal and a Borland Pascal extension.

Example

Description

,

Example

Example

```
program CurrentRoutineNameDemo;

procedure FooBar;
begin
  WriteLn (CurrentRoutineName) { 'FooBar' }
end;

begin
  WriteLn (CurrentRoutineName); { 'main program' }
  FooBar
end.
```

See also

CWord

Synopsis

```
type
  CWord = CCardinal;
```

Description

'CCardinal' is an unsigned integer type. On some platforms it is 32 bits wide and thus has a range of '0 .. 4294967295'. Its range of '

Example

Set [\[TimeStamp\]](#), page 431.

See also

[\[TimeStamp\]](#), page 431, [\[GetTimeStamp\]](#), page 331, [\[Time\]](#), page 430, Section 6.10.8 [\[Date And Time Routines\]](#), page 99.

Dec

Synopsis

For ordinal types:

```
procedure Dec (var x: ordinal_type);
```

or

```
procedure Dec (var x: ordinal_type; Amount: and_integer_type);
```

For pointer types:

```
procedure Dec (var p: any_pointer_type);
```

or

```
procedure Dec (var p: any_pointer_type; Amount: and_integer_type);
```

Description

For ordinal types, 'Dec' decreases the value of 'x' by one or by 'amount'

See also

[\[Inc\]](#), page 338

Description

'Double' is a synonym for the 'Real' data type and supported for compatibility with other compilers.

Conforming to

'Double' is a Borland Pascal extension.

Example

```
program DoubleDemo;
var
  A: Double; { There is nothing special with 'Double'. }
  B: Real;
begin
  A := Pi;
  A := B
end.
```

See also

downto

Synopsis

```
for variable := value1 downto value2 do
  statement
```

Description

The 'downto' reserv(d)8die

else

Synopsis

As part of the `if ... then ... else` statement:

```
if Boolean_expression then
  statement1
else
  statement2
```

or, as part of the `case ... else` statement:

```
case expression of
  selector: statement;
  ...
  selector: statement
else { ''otherwise'' instead of ''else'' is allowed }
  statement;
  ...
  statement
end
```

Description

'else' is part of the 'if ... then ... else' statement which provides a possibility to execute statements alternatively. In the case statement, 'else' starts a series of statements which is

See also

[Chapter 9 \[Keywords\]](#), page 453, [Section 6.1.7.2 \[begin end Compound Statement\]](#), page 56,

begin


```
to begin do
  a := 42;

end.
```

See also

[Chapter 9 \[Keywords\], page 453](#), [Section 6.1.8.1 \[Modules\], page 60](#).

exports

finalization

(Under construction.)

Synopsis

Description

Unit finalization.

It is equivalent to Extended Pascal's 'to end do'.

Conforming to

'fi nal i zati on' is a Borland Delphi extension.

Example

See also

[Chapter 9 \[Keywords\]](#), [page 453](#), [\[initialization\]](#), [page 340](#), [\[to end do\]](#), [page 433](#).

Finalize

(Under construction.)

Synopsis

```
procedure Fi nal i ze (var Aynthing);
```

Description

'Fi nal i ze' does all necessary clean-ups for the parameter. This is normally done automatically when a variable goes out of scope, so you need to call 'Fi nal i ze' only in special situations, e.g. when you deallocate a dynamic variable with 'FreeMem' rather than 'Di spose'.

Using a variable after applyi1(i)1(c)068.7260T67.939009Tf3.030Td[(F4-2d[(D(-E.09101iC-.817Td[(d[(Fi

Flush

(Under construction.)

Synopsis

```
procedure Flush (var F: any_
```

Example

```
program ForDemo;
var
  CharSet: set of Char;
  c: Char;
  n: Integer;
  Fac: array [0 .. 10] of Integer;
  PInt: ^Integer;
begin
  CharSet := ['g', 'p', 'c'];
  for c in CharSet do
    WriteLn (c); { prints 'c', 'g', 'p' in three lines }
  Fac[0] := 1;
  for n := 1 to 10 do { computes the factorial of n for n = 0 .. 10 }
    Fac[n] := Fac[n - 1] * n;
  {$X+}
  { prints n! for n = 0 .. 10 }
  for PInt := @Fac[0] to @Fac[10] do
    WriteLn (PInt - @Fac[0], '! = ', PInt^);
end.
```

See also

[Chapter 9 \[Keywords\], page 453](#), [Section 6.2.11.6 \[Set Types\], page 76](#), [Section 6.6 \[Pointer Arithmetics\], page 84](#)

FormatString

Description

Declaration of a routine whose definition follows below.

Conforming to

'forward

See also

[Section 6.2.4 \[Real Types\], page 68](#)

function

(Under construction.)

Synopsis

Example

The Borland-compatibility unit 'Graph' from the 'BPcompat' package supports a 'GetImage' and a 'PutImage'

implementation

(Under construction.)

Synopsis

Description

Module or unit implemenon part.

Conforming to

‘i mplementati on’ is 4n Extended Pascal 4nd a UCSD Pascal extension.

Example

See also

[Chapter 9 \[Keywords\]](#), page 453.

import

Synopsis

See also

[\[Dec\]](#), [page 304](#), [\[Pred\]](#), [page 388](#), [\[Succ\]](#), [page 428](#), [Section 6.6 \[Pointer Arithmetics\]](#), [page 84](#).

Include

Synopsis

```
Include (set_variable
```


IOResult

(Under construction.)

Synopsis

```
function IOResult: Integer;
```

Description

Conforming to

'IOResult' is a UCSD Pascal extension.

Example

See also

is

Synopsis

Description

Object type membership test.

Conforming to

'is' is an O-33bject Pascal and a Borland Delphi extension.

Example

See also

[Chapter 9 \[Keywords\]](#), [page 453](#), [Td\[\('\)-333\(is\)-33\(a\)1\(l\)11242\(an\)7f28.799-22.753Td\[\(function\)-61\(l\)11s](#)

Leave

Synopsis

```
Leave { simple statement }
```

Description

With 'Leave' you can exit a loop or a `while` or `do` loop instantly. It can only be used within

Conforming to

'Leave

Conforming to

'Length

Ln

Synopsis

```
function Ln (x: Real): Real;  
or  
function Ln (z: Complex): Complex;
```

Description

Ln is the natural logarithm function.

Example

```
program LongestIntDemo;  
var  
  a: LongestInt;  
begin  
  a := 42;  
  WriteLn (a)  
end.
```

See also

[Section 6.2.3 \[Integer Types\], page 64](#), [Section 6.2.11.1 \[Subrange Types\], page 70](#).

LongestReal

(Under construction.)

Synopsis

```
type  
  LongestReal = LongReal; { might get bigger than LongReal someday }
```

Description

Conforming to

‘LongestReal’ is a GNU Pascal extension.

Example

```
program LongestReal Demo;  
var  
  a: LongestReal;  
begin  
  a := 42;  
  WriteLn (a)  
end.
```

See also

LongestWord

Synopsis

```
type  
  LongestWord = LongestCard;
```

Description

'LongestWord

```
    a := 42;  
    WriteLn (a)  
end.
```

See also

[Section 6.2.3 \[Integer Types\], page 64](#), [Section 6.2.11.1 \[Subrange Types\], page 70](#).

Mark

(Under construction.)

Synopsis

```
procedure Mark (var P: Pointer);
```

Description

Conforming to

‘Mark’ is a UCSD Pascal extension.

Example

See also

Max

(Under construction.)

Synopsis

```
function Max (x1, x2: ordinal_or_real_type): same_type;
```

Description

Conforming to

‘Max’ is a GNU Pascal extension.

Example

See also

MaxChar

(Under construction.)

Synopsis

Description

The value of `Max` is the largest value of `MaxChar`

Conforming to

'MaxChar' is an ISO 10206 Extended Pascal extension.

Example

See also

MaxInt

(Under construction.)

Synopsis

Description

The MaxInt

Example

```
program MedCardDemo;  
var  
  a: MedCard;  
begin  
  a := 42;  
  WriteLn (a)  
end.
```

See also

[Section 6.2.3 \[Integer Types\], page 64](#), [Section 6.2.11.1 \[Subrange Types\], page 70](#).

MedInt

Synopsis

```
type  
  MedInt { built-in type }
```

Description

MedReal

See also


```
    WriteLn ( ' Foo' )  
end;  
  
begin  
  p := Foo;  { Works, despite the 'near'. }  
  p  
end.
```

See also

[Chapter 9 \[Keywords\], page 453](#)

```
function New (variant_record_pointer_type;  
             tag_fields): same
```

Conforming to

'ni l' is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example

```
program Ni l Demo;
const
  NodeNum = 10;
type
  PNode(=) -523(v)55(ar)1(i an)28(eU4e1(al) -333ndecord5-12. 4534v)55(ar)1(i anKey: ) -333n
  type
```

Description

Boolean or bitwise negation operator.

Conforming to

‘not’ is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example

See also

[Chapter 9 \[Keywords\], page 453.](#)

Null

Synopsis

```
var
  Null: Void absolute 0;
```

Description

‘Null’ is a predefined variable at address ‘ni l’. ‘Null’ can be passed as a “void” argument to a procedure, function or operator expecting a “var” parameter. *Note:* Make sure they 110atoui-333(all)-1(e-3

Odd

Synopsis

```
function Odd (i: Integer): Boolean;
```


Ord

Synopsis

```
function Ord (ordinal_value): Integer;
```

Example

```
program OrElseDemo;  
var  
  a: Integer;  
begin  
  ReadLn (a);  
  if (a = 0) or else (100 div a > 42) then { This is safe. }  
    WriteLn ('100 div a > 42')  
end.
```

See also

[Chapter 9 \[Keywords\]](#), [page 453](#), [\[or](#)

Example

```

program Or_ElseDemo;
var
  a: Integer;
begin
  ReadLn (a);
  if (a = 0) or_else (100 div a > 42) then { This is safe. }
    WriteLn (' 100 div a > 42')
end.

```

See also

[Chapter 9 \[Keywords\], page 453](#), [\[or else\], page 377](#), [\[or\], page 375](#), [\[and_then\], page 264](#).

otherwise

Synopsis

Default 'case' branch as part of the `case` `case`
`else'e`

Example

See also

packed

Synopsis

Description

‘packed

```
S := 'Hello!'; { blank padded }  
WriteLn (S);  
  
T := 'GNU Pascal'; { GPC extension: this also works. }  
WriteLn (T)  
end.  
'DateRec' has 24 bits = 3 bytes in total; 'Dates' has 3000 bytes.
```

See also

[Chapter 9 \[Keywords\]](#), [page 453](#), [\[Pack\]](#), [page 380](#),

Conforming to

‘Posi ti on’ is an ISO 10206 Extended Pascal extension.

Example

See also

pow

(Under construction.)

Synopsis

```
operator pow (base: Real; exponent: Integer) = power: Real;  
or  
operator pow (base: Complex; exponent: Integer) = power: Complex;
```

Description

Exponentiation operator with integer exponent.

Conforming to

‘pow’ is an ISO 10206 Extended Pascal extension.

Example

See also

[Chapter 9 \[Keywords\], page 453.](#)

Pred

Synopsis

```
function Pred (i: ordinal_type): ordinal_
```

Description

Returns the predecessor of the *ordinal_type* value 'i', or, if the second argument 'j' is given, its 'j'th predecessor. For integer values 'i', this is 'i - 1' (or 'i - j'). (No, 'Pred' does *not* work faster than plain subtraction. Both are optimized the same way, often to a single machine instruction.)

If extended syntax is on, the argument may also be a pointer value. In this case, the address is decremented by the size of the variable pointed to, or, if 'j' is given, by 'j' times the size of

Description

GPC currently accepts but ignores the 'private' directive in object type declarations.

Conforming to

'private' is a Borland Pascal extension.

Example

See also

[Chapter 9 \[Keywords\]](#), [page 453](#), [\[protected\]](#), [page 391](#), [\[public\]](#), [page 394](#), [\[published\]](#), [page 395](#).

procedure

(Under construction.)

Synopsis

Description

Procedure declaration.

Conforming to

'procedure' is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example

See also

Example

See also

[Chapter9\[Keywords\],page 453](#) .

property

Notyetimplemented.

Synopsis

Description

Object properties.

Conforming to

‘property’is anObjectPascalanda BorlandDelphiextension.

Example

See also

[Chapter9\[Keywords\],page 453](#) .

protected

(Underconstruction.)

Synopsis

Description

The ExtendedPascalmeaning of ‘ protected’is supported byGPC.

PtrCard

(Under construction.)

Synopsis

```
type
  PtrCard = Cardinal attribute (Size = BitSizeOf (Pointer));
```

Description

An unsigned integer type of the same size as a pointer.

Conforming to

‘PtrCard’ is a GNU Pascal extension.

Example

```
program PtrCardDemo;
var
  a: PtrCard;
  p: Pointer;
begin
  GetMem (p, 10);
  a := PtrCard (p);
  Inc (a);
  p := Pointer (a)
end.
```

See also

PtrDi Type

(Under construction.)

Synopsis

```
type
  PtrDi ffType { built-in type }
```

Description

‘PtrDi ffType

Example

```

program PtrDiffTypeDemo;
var
  a: array [1 .. 10] of Integer;
  d: PtrDiffType;
  p, q: ^Integer;
begin
  p := @a[1];
  q := @a[4];
  {$X+}
  d := q - p
end.

```

See also

PtrInt

(Under const-43o-525(:=lon.525n)11F6713.091Tf-28.794-24.121Td[(See)-3ynopsisF5210.909Tf28.799-24.6
 arrayffTer;0-12.453Td[(p,)-525(arrayfoir;)r-11.455-12.453Td[4begin
 p,25(q:)-510np,:= q2nanp,:= a[1]foir;anend.

also

PtrWord

(Under construction.)

Synopsis

```
type
  PtrWord = PtrCard;
```

Description

published

Re

Synopsis

```
function Re (z: Complex): Real;
```

Description

'Re' extracts the real part of the complex number 'z'.

Conforming to

'Re' is an ISO 10206 Extended Pascal extension.

Example

```
program ReDemo;
var
  z: Complex;
begin
  z := Cmplx (1, 2);
  WriteLn (Re (z) : 0 : 5)
end.
```

See also

[\[Cmplx\]](#), [page 292](#), [\[Im\]](#), [page 335](#), [\[Arg\]](#), [page 269](#)

ReadLn

(Under construction.)

Synopsis

```
procedure ReadLn (var F: Text; variables);  
or  
procedure ReadLn (variables);
```

Description

Conforming to

‘ReadLn’ is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example

See also

ReadStr

(Under construction.)

Synopsis

```
procedure ReadStr (const S: String; variables);
```

Description

Conforming to

‘ReadStr’ is an ISO 10206 Extended Pascal extension.

Example

See also

Real

(Under construction.)

Synopsis

```
type
```


Sai626323n

repeat

Synopsis

repeat

Description

'Reset' opens an existing file for reading. The file pointer is positioned at the beginning of the file.

Like 'Rewrite', 'Append' and 'Extend' do, 'Reset' accepts an optional second parameter for the name of the file in the filesystem and a third parameter for the block size of the file. The third parameter is allowed only (and by default also required) for untyped files. For details, see [\[Rewrite\]](#), page 405.

Conforming to

'Reset' is defined in ISO 7185 Pascal. The 'BlockSize' parameter is a Borland Pascal extension. The 'FileName' parameter is a GNU Pascal extension.

Example

```
program ResetDemo;
var
  Sample: Text;
  s: String (42);
begin
  Rewrite (Sample); { Open an internal file for writing }
  WriteLn (Sample, 'Hello, World!');
  Reset (Sample); { Open it again for reading }
  ReadLn (Sample, s);
  WriteLn (s);
  Close (Sample)
end.
```

See also

[\[Assign\]](#), page 272, [\[Rewrite\]](#), page 405,

Conforming to

‘Return’ is a GNU Pascal extension.

Example

See also

ReturnAddress

(Under construction.)


```
WriteLn (Int (12.345) : 1 : 5); { 12.00000 }
WriteLn (Round (12.345) : 1); { 12 }
WriteLn (Trunc (12.345) : 1); { 12 }

WriteLn (Frac (-12.345) : 1 : 5); { -0.34500 }
WriteLn (Int (-12.345) : 1 : 5); { -12.00000 }
WriteLn (Round (-12.345) : 1); { -12 }
WriteLn (Trunc (-12.345) : 1); { -12 }

WriteLn (Frac (12.543) : 1 : 5); { 0.54300 }
WriteLn (Int (12.543) : 1 : 5); { 12.00000 }
WriteLn (Round (12.543) : 1); { 13 }
WriteLn (Trunc (12.543) : 1); { 12 }

WriteLn (Frac (-12.543) : 1 : 5); { -0.54300 }
WriteLn (Int (-12.543) : 1 : 5); { -12.00000 }
WriteLn (Round (-12.543) : 1); { -13 }
WriteLn (Trunc (-12.543) : 1); { -12 }
end.
```

See also

[Section 6.2.4 \[Real Types\], page 68](#)

Description

Conforming to

‘Seek’ is a UCSD Pascal extension.

Example

See also

SeekEOF

Conforming to

‘SeekWrite’ is an ISO 10206 Extended Pascal extension.

Example

See also

segment

Not yet implemented.

Synopsis

Description

Segment specification.

Conforming to

‘segment’ is a UCSD Pascal extension.

Example

See also

[Chapter 9 \[Keywords\], page 453.](#)

Self

(Under construction.)

Synopsis

Description

Conforming to

‘Self’

set

Synopsis

In type definitions:

```
set of ordinal_type { built-in type class }
```

Description

A set contains zero or more elements from an ordinal type, e.g. Char, a subrange of Char,

See also

[\[Length\]](#), [page 347](#), [\[String\]](#), [page 427](#)

```
{ This is somewhat fragmentary code. }  
function GetObject (var InputFile: File) = Result: BasePtr;  
const  
  VMTable: array [1 .. 2] of PObjectType =  
    (TypeOf (BaseObj), TypeOf (ChildObj));  
var  
  Size: }
```

2. Use as a "procedure": 'operand1' is shifted left by 'operand2'; the result is stored in 'operand1'.

Conforming to

'shl' is a Borland Pascal extension.

Use of 'shl' as a "procedure" is a GNU Pascal extension.

Example

```
program ShortIntDemo;  
var  
  a: ShortInt;  
begin  
  a := 42;
```


Example

```
program ShrDemo;
var
  a: Integer;
begin
  a := 1024 shr 4; { yields 64 }
  a := -127 shr 4; { yields -8 }
```


Example

```
program SmallIntDemo;  
var  
  a: SmallInt;  
begin  
  a := 42;  
  WriteLn (a)  
end.
```

SqRt

Synopsis

```
function SqRt (x:
```


foargetr

S(anda(dInput))TJ/F5110.909Tf74.41540Td['r)-333isec
Example

See also

StandarOuSt

(Under conscruction.1(r(o))TJ/F6713.091Tf-14.944-7.9374Td[(ynopsise)]TJ0-34.91Td[D(e)-1sn)1(ar)-

foargetr

S(anda(OuSnput))TJ/F5110.909Tf80.18140Td['r)-333isec
Exacle

See also

Str1

ynopsise

Description

The 'StdErr' variable is connected to the standard error file handle. To report errors, you should prefer 'WriteLn (StdErr, ' everything wrong')' over 'WriteLn (' everything wrong')'.

Conforming to

'StdErr' is a GNU Pascal extension.

Example

```
program StdErrDemo;
var
  Denominator: Integer;
begin
  ReadLn (Denominator);
  if Denominator = 0 then
    WriteLn (StdErr, ParamStr (0), ': division by zero')
  else
    WriteLn ('1 / ', Denominator, ' = ', 1 / Denominator)
end.
```

See also

[\[StandardError\]](#), [pag001-424](#), [\[Output\]](#), [pag001-380](#)

See also

[\[WriteStr\]](#), page 450.

String

(Under construction.)

Synopsis

Description

Conforming to

‘String’ is an Extended Pascal and a UCSD Pascal extension.

Example

001-536.882-783.838499T6t(9T186.88Tr5)-1(nfor)-1(m4(St)1(r)-1(i)1(ng))J/F51

Example

```
program TextDemo;
var
  t: Text;
begin
  Rewrite (t, 'hello.txt');
  WriteLn (t, 'Hello, world!')
end.
```

See also

[\[file\]](#), [page 322](#), [\[AnyFile\]](#), [page 265](#).

then

(Under construction.)

Synopsis

Description

Part of an 'i f' statement or part of the 'and then' operator.

Conforming to

'then' is defined in ISO 7185 Pascal and supported by all known Pascal variants.

Example

```
t: Text;
begin
```


Trim

(Under construction.)

Synopsis

```
function Trim (S: String): String;
```

Description

Description

Conforming to

‘Truncate’ is a Borland Pascal extension.

Example

See also

type

Synopsis

As a type declaration:

```
type  
    type_identifier = type_definition;
```

or with initialization:

```
type  
    type_identifier = type_definition value constant_expression;
```

Description

The reserved word ‘type’ starts the declaration of a *type identifier* which is defined by *type_definition*. For further description see [Section 6.1.4 \[Type Declaration\], page 50](#), [Section 6.1.4 \[Type Declaration\], page 50](#),

5type of

(Under construction.)

```
constructOr Foo.Init;  
begin  
end;  
  
var  
  MyFoo: FooPtr;  
  
begin  
  MyFoo := New (BarPtr, Init);
```

Conforming to

'uni t' is a UCSD Pascal extension.

Example

See also

[Chapter 9 \[Keywords\]](#), page 453

Example

```
program VarDemo;
```


See also

while

Synopsis

```
while boolean_expression do  
  statement
```

Description

The ‘while’ statement declares a loop. For26634Tt 2(s)-crihapio’ s

Example

See also

WriteLn

(Under construction.)

Synopsis

```
procedure WriteLn (var F: Text; values_and_format_specifications);  
or procedure WriteLn (values  
                     _and_format
```


property (OP, BD) (see

10.2 The GPC Mailing List Archives

Perhaps your problem was already discussed on the list. There is a searchable archive of the

11 The GNU Pascal To-Do List.

This is the To-Do list for the GNU Pascal Compiler.

11.2.1 Planned features: Strings

- const/var 'AnyString' parameters and pointers (records internally, cf. gpc.pas) (GetCapacity; only for var parameters)
- 'SetLength

- default parameters (cf. OOE section C.4; Delphi 4?) (iniparm[12].pas) <E183vio-000lyH-00@f12.mail.ru> multithread
variable number of arguments <32F9CFE7.5CB@lmemw.ericsson.se> ?? ??? '--wi p

- 20041012: initializers of variant records don't work (fjf259.pas), (peter6.pas) <C1256791.0021F002.00@synln01.synstar.de>
- 20041012: initializers of packed arrays don't work (emil5.pas)
- 20041007: the '

- 20031004: do not allow disposing of 'nil' pointers in standard Pascal modes (fjf917*.pas)
- 20031001: arithmetic expressions don't work as lower array/subrange bounds (fjf204.pas, fjf248.pas, fjf293.pas, fjf336.pas, fjf346a.pas, fjf622.pas)
-

- Recognize Pascal strings (to avoid looking for comments and directives within strings) enclosed in single (like Standard Pascal) or double quotes (like C).
- Option handling, sharing tables in 'gpc-options.h' with the compiler:
 - Default option settings
 - Options can imply other options (e.g., 'borland-pascal' -> 'no-macros' etc.)
 - Short compiler directives
 - Short directive 'W' (warnings) is disabled in 'borland-pascal' and 'delphi' because it has another meaning there
- Compiler directives ('

- Delphi's 'external [*libname*] [name *name*]' construct where *libname* and *name* can be string expressions. Since name

12.4 Tree Nodes

```
|  
|--- (decl arati on)  
|
```

represented as a 'PLUS_EXPR' tree node whose 'TREE_OPERAND[0]' is the 'VAR_DECL' node 'Foo', and whose 'TREE_OPERAND[1]' is the 'VAR_DECL' node '

12.6 GPI files – GNU Pascal Interfaces

That's it. Now you should be able to "read" GPI files using GPC's '`--debug-gpi`' option.

Appendix A GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place – Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

GPL Preamble

The licenses for most

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the

you accompany it with the complete corresponding machine-readable source code, which

- c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d. If distribution of the work is made by offering access to copy from a designated place,

Appendix C DEMO COPYING

This is the common copying notice for all files found in 'demos/' and 'docdemos/' (unless stated otherwise in the file itself). They are distributed under the GNU General Public License with a notable exception:

Copyright (C) 1997-2005 Free Software Foundation, Inc.

Authors: See notice in the demo program. If not listed there, these are the authors of the GNU Pascal Compiler.

Appendix D Contributors to GNU Pascal.

Jukka Virtanen

Francisco Javier Fernandez Serrador

translated the GPC documentation into Spanish.

Maurice Lombardi

maintains the DJGPP port of GPC, improved the numerical routines for real and complex numbers and improved and extended the GMP real routines.

Emil Jerabek

improved the numerical routines for real and complex numbers.

Adriaan van Os

helped with the port of GPC to Mac OS X and set up a [web site](#) with sources, binaries, patches and building instructions for this platform, and helped with Mac Pascal dialect support in GPC, since January 2003.

Neil Santos

James A. Morrison

helps porting GPC tsGCC-4.x backends since February 2005.

Russell Whitaker

updated and maintained the GNU Pascal FAQ. (see [Chapter 3 \[FAQ\]](#), [page 15](#))

Matthias Klose

integrated GPC into EGCS and Debian GNU/Linux in May 1998, improved the installation process, etc.

Peter N Lewis

added support for Mac Pascal dialect specific features and improved the documentation.

OrlandsLlanes

maintains the GPC 4.1.0 (1998) and 4.1.1 (1999) versions.

maintains the GPC 4.1.0 (1998) and 4.1.1 (1999) versions.

- Alain Lichnewsky ported GNU CC to the Mips cpu.
- Devon Bowen, Dale Wiles and Kevin Zachmann ported GNU CC to the Tahoe.
- Jonathan Stone wrote the machine description for the Pyramid computer.
- Gary Miller ported GNU CC to Charles River Data Systems machines.
- Richard Kenner of the New York University Ultracomputer Research Laboratory wrote the machine descriptions for the AMD 29000, the DEC Alpha, the IBM RT PC, and the IBM RS/6000 as well as the support for instruction attributes. He also made changes to better support RISC processors including changes to common subexpression elimination, strength

Appendix E Resources For Use With GPC.

GRX

GRX is a graphics library for C and GNU Pascal, including a mostly BP compatible 'Graph' unit. It is available from

<http://www.gnu.de/software/grx/>

Although GRX originated on DJGPP, a DOS programming platform, it is portable to Linux

[http://www.moorecad.com/standardpascal/i so7185. ps](http://www.moorecad.com/standardpascal/i%207185.ps)
(ISO 7185 Pascal)

[http://www.moorecad.com/standardpascal/i so10206. ps](http://www.moorecad.com/standardpascal/i%2010206.ps)
(ISO 10206 Extended Pascal)

Note: These documents are a bit hard to navigate (e.g., in ghostview) because they are missing the so called "document structuring comments" (DSC). The GPC source distribution contains a little script 'ps2dsc' to add the DSC again and make the documents easier to navigate.

Appendix F The GNU Project.

GNU Pascal is part of the GNU project which was founded by Richard Stallman in 1984. The aim of the GNU project is to provide a complete operating system with editors, compilers etc. as *Free Software*.

People often confuse *Free Software* with *public domain software* or have other wrong infor-

Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just hand-holding and repair service.

rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

“Won't programmers starve?”

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it does a good job; but its defenders are wrong in assuming it always works this way. If the runners forget why the reward is offered and become intent on winning, no matter how, they may find other strategies – such as, attacking other runners. If the runners get into a fist fight, they will all finish late.

Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them ("For every ten yards you run, you can fire one shoty(e)os g57hotyF sr

But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing – often, chosen because he hopes to use the r1(s)-3ults w(s)-hen it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.

The consequenc1(s)-3:

- The computer-using community supports software development.
- This community decides what level of support is needed.
- Users who care which projects their share is spent on can choose this for themselves.

In the long run, making programs free is a step toward the post-scarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

We have already greatly reduced the amount of work that the w(s)-hole society must do for its

Appendix F: The GNU Project.

Index-GPC

★

★	97, 98
★★	97

-

-	
---------	--

CWord	302
Cycle	303

D

Data Types	68
Data Types, Definition	64
Database	514
Date	99, 303
DBM	514
debugging	102
Dec	97, 304

Loops, Loop Control Statements	60
Low	356
LT	357
LTPad	357

M

Machine-dependencies in Types	80
magic, internals	483
Mailing List	459

Slice access	83
SmallInt	422

