DELF · A NATURAL SORT · PHASE 5

A Natural Sort, Idiomatically

From right-but-slow
to well-modelled and fast:
2249 ms to 148 ms,
Explorer’s order exactly.

```
// the three kinds, in Explorer’s order
TToken =
  | Symbol(Mark: char)
  | Digits(Text: string)
  | Letter(Ch  : char);

// 'file10.txt' is f, i, l, e, 10, ., t, x, t
function NaturalKey(const Filename: string): TArray<TToken>;
begin
  Exit( Filename.Runs(IsDigit).AndThen(RunTokens) );
end;
```

8 Nov 2026 · poc/examples/08-natural-sort/5-idiom.delf

THE BAR · ADUG FORUM, JANUARY 2024

Six lines, and Windows does the thinking

```
function StrCmpLogicalW(psz1, psz2: PWideChar): Integer;
  stdcall; external 'shlwapi.dll';

Comparer := TComparer<string>.Construct(
  function(const Left, Right: string): Integer
  begin
    Result := StrCmpLogicalW(PWideChar(Left.ToLower),
                             PWideChar(Right.ToLower));
  end);

TArray.Sort<string>(StringArray, Comparer);
```

Good

Short, correct, Explorer’s own order.

On 20,000 real names

94 ms. 27 ms without ‘ToLower’.
The call ignores case already.

But

The code doesn’t say **what** a Natural Sort is.
Windows only. The code lives in a DLL.

The user: “the win version is cheating and isn’t multi-platform .. but can be a comparison”

THE BAR · YYPBD/YYPBD-DELPHI-STRINGSORT

Written by hand: fast, and not quite right

```
while True do
begin
  Inc( Pos1 );
  Inc( Pos2 );
  …
  if (Str1[Pos1] ⩾ '0') and (Str1[Pos1] ⩽ '9') and
     (Str2[Pos2] ⩾ '0') and (Str2[Pos2] ⩽ '9') then
  begin
    EndPos1 := Pos1;
    repeat
      Inc(EndPos1);
    until not ((Str1[EndPos1] ⩾ '0') and (Str1[EndPos1] ⩽ '9'));
    …
    while True do
    begin
      if EndPos1 - Pos1 = EndPos2 - Pos2 then
        …
      else if EndPos1 - Pos1 > EndPos2 - Pos2 then
      begin
        if Str1[Pos1] = '0' then
          …
```

What it is

An index into each string, digit tests by hand;
196 lines with its TStringList and TListView wrappers.

On 20,000 real names

161 ms. But 886 neighbours that Explorer would put the other way round.

src/lib/Sort.StringCompare.pas, NaturalOrderCompareString

WHERE DELF STARTED · PHASE 4 · 4-KINDS.DELF

Explorer’s rule, stated, and slow

```
// the kinds of token, in Explorer's order
TKind  = (Symbol, Digit, Letter);
TToken =
  | Number(Digits: string)
  | Single(C     : char);

function KindOf(const T: TToken): TKind;
begin
  Exit( match T of
          | Number(_) : Digit
          | Single(C) : if C.IsLetter then Letter else Symbol
        end );
end;

// Tokens('file10.txt') is f, i, l, e, 10, ., t, x, t
function NaturalCompare(const A, B: string): Integer;
begin
  const X = Tokens(A);
  const Y = Tokens(B);
  Exit( X.ZipWith(Y, CompareTokens).Find(λC → C <> 0)
        .OrDefault(CompareValue(X.Length, Y.Length)) );
end;
```

Right

Three kinds, declared in Explorer’s order: Ord ranks them. Explorer’s order exactly.

2249 ms

Both words cut into tokens inside every comparison. A character asked “is it a letter?” every time.

Phase 4 of “A natural sort, four times”

THE AIM

“I want to write good looking, well-modelled code – in our idiom – that also performs pretty well.”

The model and how it reads come first. Speed is the check that the idiom doesn’t cost too much.

NINE WAYS, TIMED

Same names, same sort, very different costs

Approach	ms	Explorer's order
StrCmpLogicalW, Windows	27	the oracle
the user's 2024 version (ToLower, then Windows)	94	exact
yypbd's hand-written walk	161	886 swapped
Delf phase 4, as written	2,249	exact
phase 4, helpers marked inline	2,243	exact
phase 4, tokens cut once per name	344	exact
keys compared by a lazy Seq pipeline	693	exact
keys compared by a loop	150	exact
Delf phase 5, the idiom	148	exact

dcc64 37.0 · 20,000 distinct real file names · best of 3 · tools/natsort_bench.py

WHAT DECIDED THE SPEED · 1

Where the work happens, not how it’s called

2249 ms

Phase 4 tokenised the words compared inside **every comparison**: about 600,000 times for 20,000 words.

344 ms

The same code with each word cut **once**, its tokens kept as its key. Nothing else changed.

6.5× faster, still exactly Explorer’s order. A word **is** its key: compute it once.

WHAT DECIDED THE SPEED · 2

What didn’t: inline, and two extra strings

inline: 2249 → 2243 ms

Six small helpers marked inline. No change.
The time was in allocating tokens and arrays, not in calls.

ToLower: 27 → 94 ms

StrCmpLogicalW already ignores case.
Two ToLower calls make two new strings per comparison, 3.5× the time, for no visible change.

Measure before believing: “inlining would help” was a guess, and wrong.

WHAT DECIDED THE SPEED · 3

Comparing two keys: laziness isn’t free

Comparing two keys, cut once	ms
lazy: AsSeq(X).Zip(Y).Map(CompareTokens).Filter(≠ 0).First	693
eager: ZipWith(X, Y, CompareTokens), then Find the first non-zero	344
a loop that stops at the first difference	150

Eager builds an array

A whole array of results, to read its first non-zero.

Lazy builds a chain

Six closures and a tuple per pair, afresh on every comparison.

Laziness pays for one chain over many elements: experiment 64 ran 10 million in 53 ms.

THE LESSON

Put the loop in the library

The source states the rule; the library runs it at the loop's speed.
Two new routines, provisional names:

`X.CompareBy(Y, Compare)`

First difference decides, else the shorter first.
C++: lexicographical_compare_three_way.

`Xs.SortBy(Key, Compare)`

Each key made once, sorted by key, stable.

`Xs: a sequence of X, X, X, …`

A token says what it is when it’s made

```
// the three kinds, in Explorer’s order
TToken =
  | Symbol(Mark: char)
  | Digits(Text: string)
  | Letter(Ch  : char);

// 'file10.txt' is f, i, l, e, 10, ., t, x, t
function NaturalKey(const Filename: string): TArray<TToken>;
begin
  Exit( Filename.Runs(IsDigit).AndThen(RunTokens) );
end;
```

A name is its key

Read once into tokens; the sort works on keys.

Classified once

Phase 4 asked “is it a letter?” at every comparison; now the token knows.

Phase 4 had TToken = Number | Single, with a separate TKind enum worked out at comparison time.

Explorer’s rule in one match; the sort, one line

```
function CompareTokens(const A, B: TToken): Integer;
begin
  Exit( match (A, B) of
    | (Digits(X), Digits(Y)) : CompareDigits(X, Y)
    | (Letter(C), Letter(D)) : CompareValue(Ord(C.ToUpper), Ord(D.ToUpper))
    | (Symbol(C), Symbol(D)) : CompareValue(SymbolRank(C), SymbolRank(D))
    | _                      : CompareValue(KindRank(A), KindRank(B))
  end );
end;

// keys compared first difference wins; each key made once
Names.SortBy(NaturalKey, λ(x, y) → x.CompareBy(y, CompareTokens))
```

148 ms on 20,000 real names, Explorer’s order exactly, and no call to Windows in the rule.

The file writes the last line as a named CompareKeys for now: the next slide says why.

AGAINST THE ORACLE

The rule in sight, at a known price

148 ms

Delf phase 5, the idiom: the rule as one match, each key made once. Exactly Explorer's order.

27 ms

StrCmpLogicalW, Windows' own: the oracle. It **is** File Explorer's algorithm. And hidden code. *(So why the @#$%# are you so slow, Explorer??)*

5.5× the oracle, from 83× in phase 4.

With **ToLower** on both filenames: **94 ms**. But StrCmpLogicalW already ignores case.

PS: FilePilot. Tiny download. Super fast Explorer app. You're welcome.
https://filepilot.tech/

dcc64 37.0 · 20,000 distinct real file names · best of 3 · tools/natsort_bench.py

WHAT IT SHOWED ABOUT THE LANGUAGE · 1

Your own higher-order function can’t take a const routine

```
// a file’s own: (T → K) goes out as TFunc<T, K>
function SortBy<T, K>(…; const Key: (T → K); …)

// a routine with a const parameter
function NaturalKey(const Name: string): …

Names.SortBy(NaturalKey, …)
// dcc: E2010 Incompatible types
```

Delphi’s rule

A routine passed as a value must match const exactly. Our library carries every operation twice, TFunc and TConstFunc.

A file can’t

In Delf both overloads are (T -> K): one signature twice.

Fixed 9 Nov: delfc forwards at the call site, in the output only (next).

THE FIX · 9 NOV

A function type says which inputs it may change

```
// no mode: it leaves its input alone
(string → Integer)
  → TConstFunc<string, Integer>

// a mode only where it changes meaning
(var Integer → ())
  → TProc_v<Integer>
((Integer, out Integer) → ())
  → TProc_co<Integer, Integer>
```

```
function Double(X: Integer): Integer;

// Apply wants (Integer → Integer)
Apply(Double, 6)

// out: forwarded, once, where it's passed
Apply(function(const A: Integer): Integer
  begin Result := Double(A); end, 6)
```

A routine whose modes differ is **forwarded**: one indirect call, about 7% on the cheapest comparison. Never a copy.

SPEC §6 · tests func_modes, func_modes_errors · Delf.Funcs: the moded types, one to four parameters

WHICH FITS WHERE

Values one way, functions the other

A value into a parameter

```
procedure Bump(var N: Integer);
const C = 5;
Bump(C) ×
```

A const value can’t go into a var slot: the function may intend to change it.

A function into a slot

```
procedure Show(const N: Integer);
// Touch wants (var Integer → ())
Touch(Show) ✓
```

A function from a const fits a slot for a function from a var: it won’t take the opportunity to change its input.

routine’s parameter ↓ · slot’s →	no mode (const)	var	out
by value or const	✓	✓ a hint: it won’t write	✗ it never assigns
var	✗ may be given a constant	✓	✗ may not assign
out	✗ may be given a constant	✗ today · sound in principle: open	✓

The two rules: the user, 9 Nov · Delphi itself: exact match only, E2010 both ways (experiments 65, 67)

THE NAMES FOR IT

Subtyping, and the arrow that flips it

Contravariant: what it takes

`(Animal → Owner) fits (Cat → Owner)`

`(const T → R) fits (var T → R)`

A function that asks less of its input fits where more is offered. The order flips.

Covariant: what it gives

`(Owner → Cat) fits (Owner → Animal)`

A function that gives more than promised fits. The order stays.

C# Func<in T, out R> · Kotlin in / out · Scala Function1[-T, +R]

Delphi: exact match only. **Delf: contravariant in modes, by forwarding**; in types, not yet.

THE CHECK · 9 NOV

Each name against the next, in one line

```
// before: a counter and an index
var Disagree := 0;
for var i := 0 to High(Ours) - 1 do
  if Explorer(Ours[i], Ours[i + 1]) > 0 then
    Inc(Disagree);
```

```
// after: what is counted, said once
const Disagree =
  Ours.Pairwise(Explorer)
      .Count(_>0);
```

`Xs.Pairwise(F)`

F of each element and the next. C++'s adjacent_difference with the operation passed in: none built in.

`(_>0)  (_=True)  (_-_)`

Sections: _ beside an operator makes it a function, holes left to right, written tight. (_-_) is each less the next.

SPEC §6 · Delf.Arrays · tests rotate_pairwise, sections · same count as the loop on 20,000 names

WHAT IT SHOWED ABOUT THE LANGUAGE · 2

Three smaller things writing it found

Exit([T])

One record in brackets went out as a set literal: dcc's E2001. Result := [T] works. The bracket should take the result's type.

A shared field name

C in both Symbol and Letter is legal, but the output names them SymbolC and LetterC. delfc says so; Mark and Ch read better anyway.

A helper hides a routine

Run[1].IsDigit calls Char's helper, not the file's IsDigit. delfc warns and says which one Delphi calls.

Two caught by delfc as the draft was written; one by dcc, now in the Honesty Box.

A FIRST DRAFT · STILL ROUGH

What the next version should remove

KindRank

Repeats the declared order by hand, 0, 1, 2. The cases already say it.

CompareKeys

A named function only because of the const limit; it should be a lambda in the sort.

Little helpers

IsDigit, Kept, OneChar exist only to be passed as values.

And two names to choose: CompareBy and SortBy, against the C++ algorithm maps.

WHAT THE PROCESS SHOWED

The well-modelled version was also the fast one

Oracle first

Explorer’s own function checked every version, on real names.

Measure, don’t guess

Inlining and laziness were both expected to help. Neither did.

Let code find gaps

Writing the idiom found a library gap and a language limit.

poc/examples/08-natural-sort · README, “The bar” · slides/phase5-outline.md