Eigenheiten meiner Programmierung:

Effiziente Programmierung war eigentlich immer für meine Arbeit notwendig. Egal ob ein Arbeitgeber erwartet hat, auf alten Leasingrückläufern Wunder zu vollbringen, oder es darum ging auf einem schmalbrüstigen Raspberry schnelle Grafikfunktionen auszuführen.

Über Jahrzehnte habe ich mir eine ganze Menge effizienter Tools geschaffen, die ich in der Unit aroclasses.pas vorhalte, die mir meine Arbeit enorm erleichtert. Die Hauptgebiete sind Dateien bearbeiten auf sehr niedrigem Level, effiziente Arbeit mit Zeichenketten und Aufgaben vereinfachen, die öfter anfallen. Da alle meine Projekte auf diese Tools zurückgreifen, müssen unter Uses aroclasses.pas und aro_types.pas eingebunden werden!
Einige wenige Funktionen sind in Assembler geschrieben, was natürlich abhängig vom Prozessor und Betriebssystem ist. Ich verwende derzeit nur LINUX 64 – Bit und die dazu passenden Register. Wer etwas anderes verwendet, kann gern selbst Hand anlegen oder muss eben noch auf diese Funktionen verzichten.

Beispiele :

function GetCPUTick: QWord;

Wenn man eine Optimierung vorgenommen hat, ist es natürlich wichtig, was es wirklich gebracht hat. Aber verlässliche Zeitmessungen sind durch das Betriebssystem eigentlich nicht möglich. Und sehr kurze Zeiten erst recht nicht, da GetTickCount nur von Zeit zu Zeit aktualisiert wird. Aber die meisten CPU haben ein Register, welches mit jedem CPU – Takt um 1 erhöht wird. GetCPUTick gibt den Wert dieser Registers zurück. Damit kann man feststellen wie viel Takte zwei Versionen im direkten Vergleich benötigen. Bitte beachten, GetCPUTick benötigt selbst einige Takte und das Register läuft irgend wann über und beginnt wieder bei 0!

Meine Projekte benötigten alle eine Ini - Datei, Sprachdateien, Bilder und vieles mehr. Schon seit Delphi erwarte ich diese Dateien in vorbestimmten Ordnern. Und diese Ordner erwarte ich im gleichen Verzeichnis wie die ausführbare Datei. Damit halte ich alles zusammen, kann den ganzen Ordner als ZIP exportieren und es gibt nie Unklarheiten, wo die Dateien gesucht werden müssen. Dazu benötigt das Programm natürlich den exakten Pfad, in dem die Programmdatei gestartet wurde. Ich erzeuge deshalb, durch das Einbinden der Unit aroclasses, sofort beim Programmstart eine Zeichenkette mit dem vollständigen Pfad, auf den ich jeder Zeit zugreifen kann.

Pascal verwendet Variablen vom Typ String, was für den Programmierer sicher einfach ist. Aber kein Betriebssystem kann damit umgehen. Jedes Betriebssystem erwartet ausschließlich nullterminierte Zeichenketten. Strings müssen also erst in nullterminierte Zeichenketten gewandelt werden. Außerdem werden Strings längst nicht mehr so verwendet wie einst in Turbopascal.
Delphi und Lazarus verwalten diese Variablentype dynamisch, weil der Programmeier sich so keine Gedanken mehr darüber machen muss, wie lang der String am Ende mal sein wird.
Aber jedes mal, wenn an den String weitere Zeichen angehängt werden sollen und der reservierte Speicher dafür nicht einfach vergrößert werden kann, muss neuer Platz reserviert werden, der Inhalt muss in den neuen Bereich kopiert werden, der alte Speicher wieder freigegeben werden und die Adresse des eigentlichen Speicherbereiches wird aktualisiert. Zeitraubende Aktionen, von denen weder der Programmierer noch der Nutzer etwas mitbekommen, die sich allerdings am Ende addieren.
Normale nullterminierte Zeichenketten benötigen unnötige Zeit, um jedes mal die gesamte Zeichenkette nach der abschließenden Null zu durchsuchen, was auch Zeitverschwendung bedeutet. Und die speziellen Funktionen für nullterminierte Zeichenketten sind einfach nur schlimm.

Ein Beispiel :

```
Var     PC : PAnsiChar;
        SS : Array[0..255] of AnsiChar;
Begin
  PC := StrCopy(ss, 'ABC');
```

In PC kann man die Adresse von ss als Ergebnis erhalten, was absolut keinen Sinn ergibt!
Erstens kann man ss sofort an jede Funktion als PansiChar übergeben und zweitens bleibt immer noch PC := @ss fals man die Adresse benötigt.

Warum soll ich also mit solchen unzulänglichen Lösungen leben? Nur weil das irgend jemand irgend wann mal so veröffentlicht hat, oder weil alle anderen das so tun ?

Also habe ich das Ganze wieder vom Kopf wieder auf die Füße gestellt und meine eigenen Funktionen geschrieben. Meine Funktionen geben nicht die unnötige Anfangsadresse zurück, sondern die Adresse der abschließenden Null! Ein kleiner Unterschied mit großen Auswirkungen. Denn das ist genau die Position an die z.B. sofort weitere Zeichen angehängt werden können, ohne erst die gesamte Kette nach der Position der abschließenden Null durchsuchen zu müssen.

In Linux fallen Pfadnamen in der Regel wesentlich länger aus als in WINDOWS. Wenn man nun eine lange Zeichenkette mit Pfad und Dateinamen hat und sucht die Position der Dateierweiterung, kann StrRScan verwenden. Die Funktion durchsucht von Beginn an jedes Zeichen, was unnötige Zeit braucht. Wenn ich einen Pointer auf den Terminator habe, und sicher bin, das das Suchergebnis sich fast am Ende befindet, suche ich natürlich nur wenige Zeichen vom Ende her rückwärts.

Wenn man Assembler beherrscht, sind solche Funktionen einfach und schnell mal geschrieben, bringen aber etwas bei jedem Aufruf.

Deshalb arbeite ich auch fast ausschließlich nur noch mit meinen eigenen optimierten Funktionen für nullterminierte Zeichenketten.

In meinen Projekten müssen sehr oft ein – oder zweistellige Zahlen angezeigt werden. Man kann natürlich jedes mal IntToStr() aufrufen. Ich nicht !

In aro_types.pas habe ich :

```
const
        ZiffAr : Array[0..256] of PAnsiChar =( '0', '1', '2', '3', '4', '5', '6', '7', '8','9',…….
```

deklariert und verwende

```
        strcopy(ss, Ziffar[Wert]); oder  Edit2.Text := Ziffar[Wert];
```

für kleine Werte bis 256, ohne jedes Mal erst den Integer in eine Zeichenkette wandeln zu müssen.

Und wenn der Wert meistens unter 257 liegt, aber gelegentlich auch mal über 256 liegen kann, verwende ich eine Funktion, welche unter 257 ZiffAr[] zurückgibt und darüber IntToStr() benutzt.

Viele kleine Schritte, die alle unnötige Rechenzeiten einsparen, welche sich am Ende aber addieren und sich damit ganz deutlich bemerkbar machen.

Als die Richtlinie aufgestellt wurde, wie der Quelltext formatiert werden sollte, waren PC – Monitore fast quadratisch. Es gab sogar Monitore die man so drehen konnte, das das Bild hochkant dargestellt wurde. Da machte die Empfehlung noch Sinn. Heute werden neue Monitore fast ausschließlich im 16:9 Format hergestellt. Das Resultat : Nur noch die halbe Seitenbreite enthält wirklich sinnvolle Informationen. Durch die vielen Zeilen, mit wenig Informationen sieht man damit nur wenig Quelltext auf einen Blick und muss ständig scrollen. Ich wende die Richtlinie deshalb in meinen Arbeiten schon lange nicht mehr an, da sie für mich keinen Sinn mehr macht. Damit kann ich einen größeren Quelltextbereich überblicken.
Wem das nicht gefällt kann das einfach mit Ctrl-D auf den alten Stand bringen.