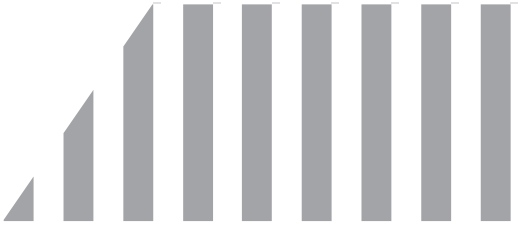
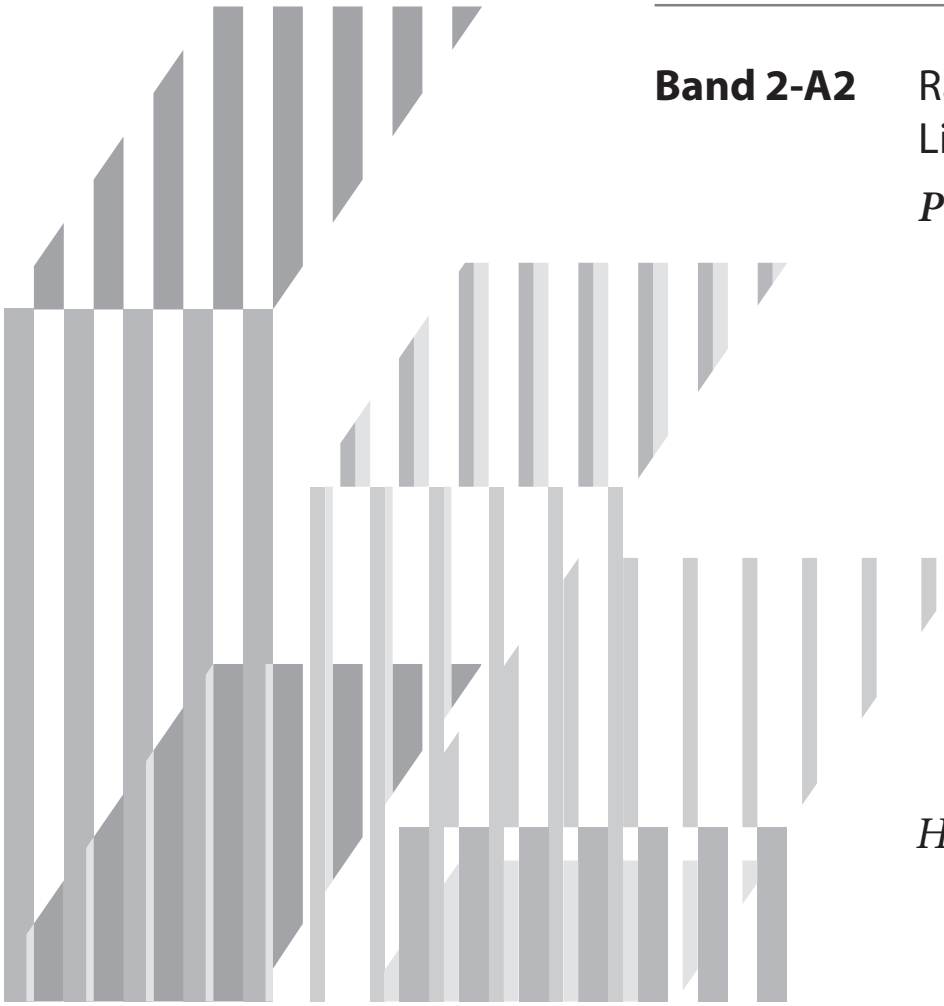
Band 2-A2



MODERNE STRUKTURIERTE PROGRAMMIERUNG

Objektorientiert und algorithmisch
ENTWERFEN | IMPLEMENTIEREN | TESTEN

2. Auflage



Band 2-A2 Ranggruppen
Link-Listen
Praxis

Horst van Bremen

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Gestaltung	Katharina Schmitt
Titel- und Einbandgrafik	Monika Sylvia Gomm † 1971
Englisch-Korrektorat	David Roseveare
Verlag	BoD · Books on Demand GmbH, Überseering 33, 22297 Hamburg, bod@bod.de
Druck	Libri Plureos GmbH, Friedensallee 273, 22763 Hamburg
Version / Datum	Version 2.1.1 vom 1.7.2026
Literaturverzeichnis	siehe Kapitel „11.1 Literaturverzeichnis“ auf Seite 441 f
Rechtliches	siehe Kapitel „11.2 Rechtliche Hinweise“ auf Seite 443 ff
ISBN des Ringbuchs	978-3-6963-6420-5

Über den Autor

Horst van Bremen
Diplom 1972 an der TU Darmstadt – Elektrotechnik und Informatik
39 Berufsjahre in der IT, davon 18 als Freiberufler

IT-Systemarchitekt
Datenbankexperte

Programmiererfahrung seit 1969
Strukturierte Programmierung nach Jackson seit 1974
Freier Autor seit 2011

Vorwort zur ersten A2-Ausgabe

Profis, die ihre Programme mit JSP entwerfen, betrachten die algorithmischen Implementierungen als mässig spannende Umsetzungsarbeiten, deren Hauptziel darin besteht, durch geschickte Modularisierung den Code übersichtlich und möglicherweise wiederverwendbar zu machen. Da eine JSP-Programmstruktur mit Operationen keinerlei Vorgaben für den formalen Programmaufbau macht, sind Erfahrung und Geschick für die Qualität des Ergebnisses maßgeblich. Diagramme der Modul-Hierarchien sind dabei hilfreich. Inhaltlich ist zu diesem Zeitpunkt ja alles bereits bei der Modellierung entschieden worden, so dass es über weite Strecken nur noch darum geht, den Entwurf getreulich in Code umzusetzen.

(Wenn Sie, geschätzte Leserinnen und Leser, aus dem vorhergehenden Abschnitt eine gewisse Langeweile herausgelesen haben, so liegen Sie damit richtig. JSP/MSP machen das Leben von Programmier(inne)n leichter, aber auch weniger spannend.)

Zugegeben, ein solches Kaliber wie EvaluateSportsDS in C erfordert schon deutlich mehr Energie als ein Brot-und-Butter-Programm, aber nach Zerlegung der Programmstruktur in ihre Komponenten stand vor allem die höchstmögliche Sorgfalt bei der Implementierung im Vordergrund. Der Einbau hinreichend vieler Trace-Anweisungen, der zunächst als lästiger Zusatzaufwand empfunden wird, garantiert in der Testphase die nötige Transparenz, um die Korrektheit der Programmaktionen verifizieren zu können. Diese Mühe sollte man sich also auf jeden Fall machen, aber dabei auch daran denken, dass dieser Code abschaltbar sein muss.

Ganz anders sieht es bei objektorientierten Implementierungen aus. Der Band 1-A2 demonstriert im Kapitel „OO-Vorbereitungen für EvaluateSportsDS“, welch weiter Weg von der Programmstruktur mit Operationen zur fertigen Planung der OO-Klassen zurückzulegen ist. Das könnten Leser(innen) durchaus als ein deutliches Manko der hier vorgestellten MSP-Lösungen werten – aber was wäre die Alternative? Gewiss lassen sich diese Lösungswege nicht auf alle möglichen Aufgabenstellungen der objektorientierten Programmierung übertragen, aber für einen guten Teil davon, vor allem den mit Aspekten der Sequentialität, wird hier eine – hoffentlich attraktive – Alternative zum freien Erfindertum vorgelegt.

Die nun folgenden Kapitel zeigen ausführlich, wie die anspruchsvollen Aufgaben, welche der Internationale Schulsportwettbewerb stellt, auf der Basis von JSP/MSP in die Praxis umgesetzt werden können. Für mich, den Autor, war dabei besonders bemerkenswert, wie harmonisch die OO-Implementierungen sich aus dem Entwurf ergaben. Es war nicht nötig, irgendwelche Tricks anzuwenden oder gar, bildlich gesprochen, dem Wolf in den Rachen zu greifen, um das Rotkäppchen wieder herauszuholen, sondern alles ergab sich folgerichtig. Davon profitierte beispielsweise die inkrementelle Implementierung von MailSportResults. Insofern steht zu hoffen, dass das viele Papier ausschließlich mit sinnvollen Anregungen für Leser(innen) aller Qualifikationsstufen oberhalb von A1 bedruckt worden ist…

Lemgo, den 14.2.2025

Übersichtsverzeichnis Band 2-A2

Inhaltsverzeichnis Band 2-A2

06/2015 Römischer Tempel „Maison Carrée“ in Nîmes, Hauptort des Départements Gard (Okzitanien)

EvaluateSportsDS in C

Die C-Implementierung von EvaluateSportsDS fand mit der Lightweight IDE statt. Erschrecken Sie bitte nicht: Das ist ziemlich viel Code, obwohl die Auswertung der Ergebnisse des Internationalen Schulsportwettbewerbs anfangs eine recht einfache Aufgabe zu sein schien. Um Ihnen die erforderliche Übersicht zu geben, bevor der Code dargestellt wird, folgen zunächst die bekannten Programmstrukturen mit Operationen, ergänzt um die Zuordnungen der Strukturblöcke zu den Modulen. Ein Aufruf-Hierarchie-Diagramm zeigt danach, wie die Module einzuordnen sind.

7.1 Ergänzte Programmstrukturen

7.1.1 Top-Programmstruktur

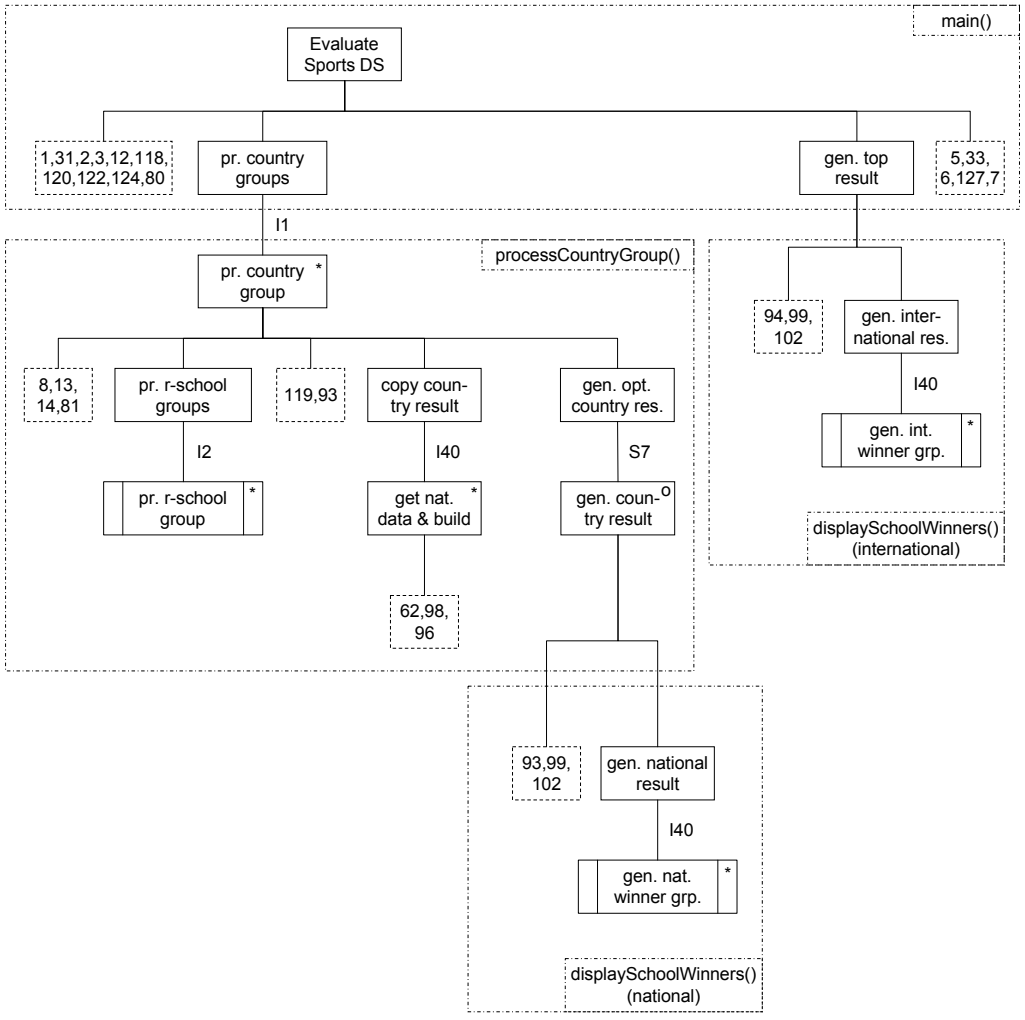


Abbildung 7.1: *Modulorientierte Darstellung des Top-Struktogramms*

Die ausgelagerten Struktogramme sind wieder durch das Symbol „vordefinierter Prozess“ gekennzeichnet. Die Namen der C-Module werden am oberen oder unteren Rand der gestrichelt gezeichneten Boxen genannt.

7.1.2 Process R-School Group

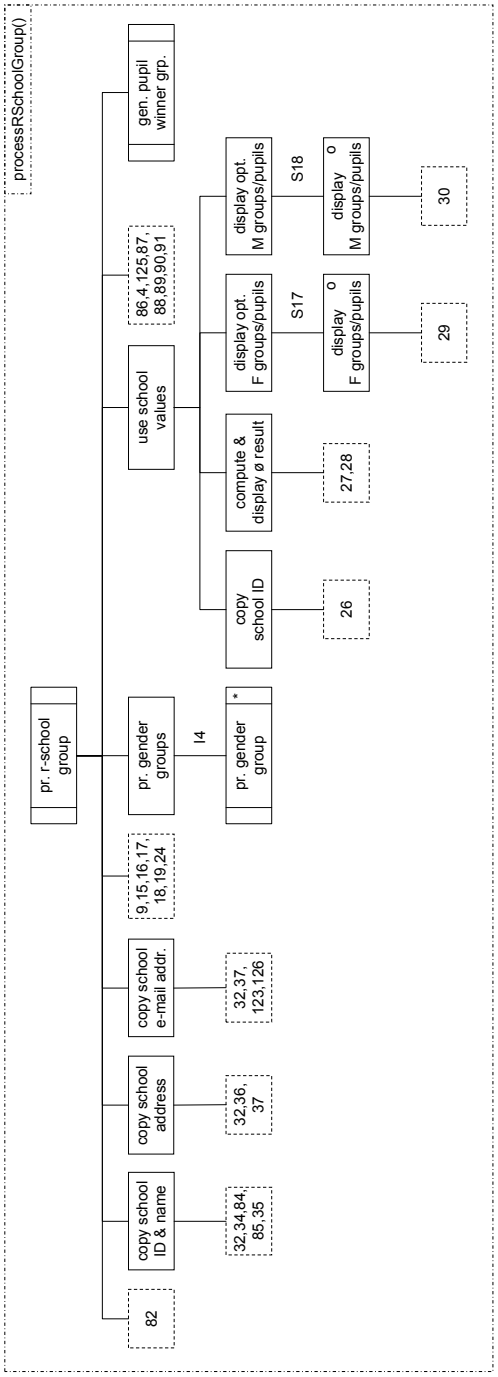


Abbildung 7.2: *Ausgelagertes Struktogramm „pr(ocess) r-school-group“ als Modul mit Auslagerungen*

Der Unterbaum von „process gender group“ folgt als „Abbildung 7.3: Ausgelagertes Struktogramm „pr(ocess) gender group“, aufgeteilt in zwei Module“. Die Gewinner-Listen-Auswertungen werden in der „Abbildung 7.4: Modulzuordnungen der ausgelagerten Struktogramme „gen(erate) pupil winner gr(ou)p“, „gen(erate) nat(ional) winner gr(ou)p“ und „gen(erate) int(ernational) winner gr(ou)p““ auf Seite 142 gezeigt.

7.1.3 Process Gender Group / Process Age Group

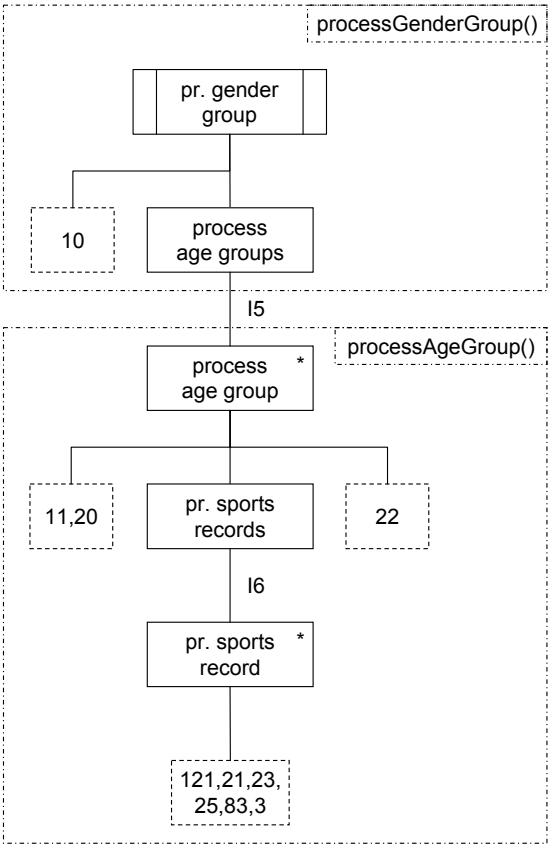


Abbildung 7.3: *Ausgelagertes Struktogramm „pr(ocess) gender group“, aufgeteilt in zwei Module*

7.1.4 Gewinner-Listen-Auswertungen

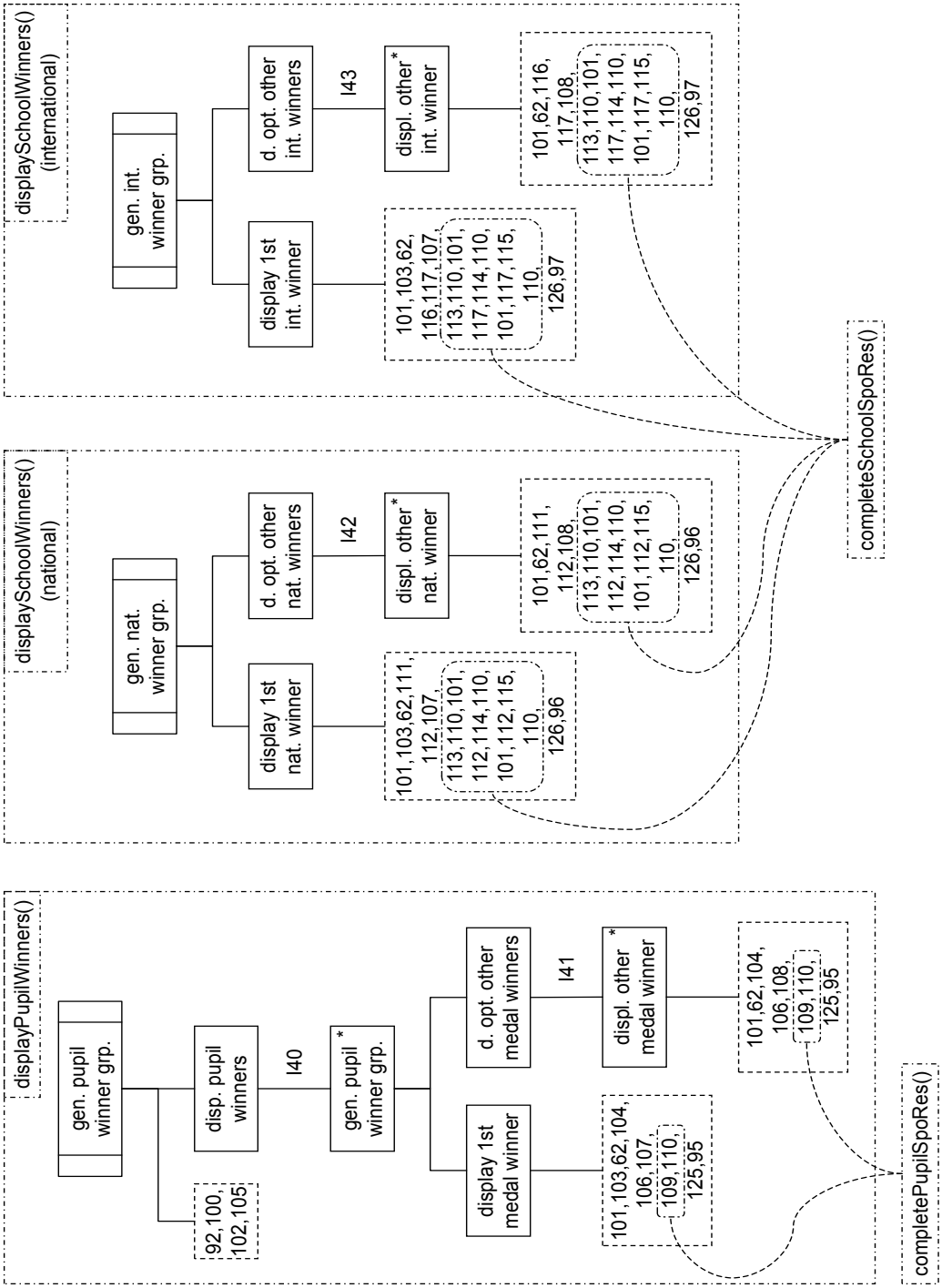


Abbildung 7.4: *Modulzuordnungen der ausgelagerten Struktogramme „gen(erate) pupil winner gr(ou)p", „gen(erate) nat(ional) winner gr(ou)p" und „gen(erate) int(ernational) winner gr(ou)p"*

Die beiden Struktogramme, aus denen `displaySchoolWinners()` hervorging, sind in der Top-Programmstruktur bereits begonnen worden. Die ausgelagerten Teile sind hier gezeigt. Gemeinsame Operationen wurden in die complete-Unterprogramme ausgelagert, um Redundanz zu vermeiden.

7.1.5 Build Linked List

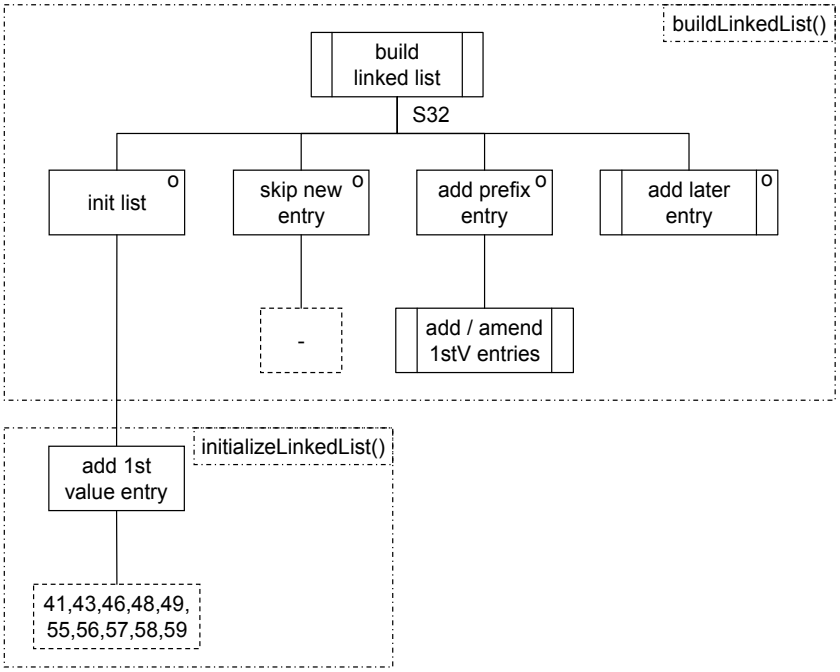


Abbildung 7.5: *Aufteilung des ausgelagerten Struktogramms „build linked list" in zwei Module und eine weitere Auslagerung*

Das Modul `buildLinkedList()` besteht aus mehreren C-Unterprogrammen. Die ausgelagerten Teile „add / amend 1stV(alue) entries" und „add later entry" verwenden beide die in der „Abbildung 7.6: Gleichsetzung des Struktogramms „compress linked list" mit dem Modul compressLinkedList()" auf Seite 144 gezeigte Kompressionsroutine, deren Struktogramm unverändert blieb.

7.1.6 Compress Linked List

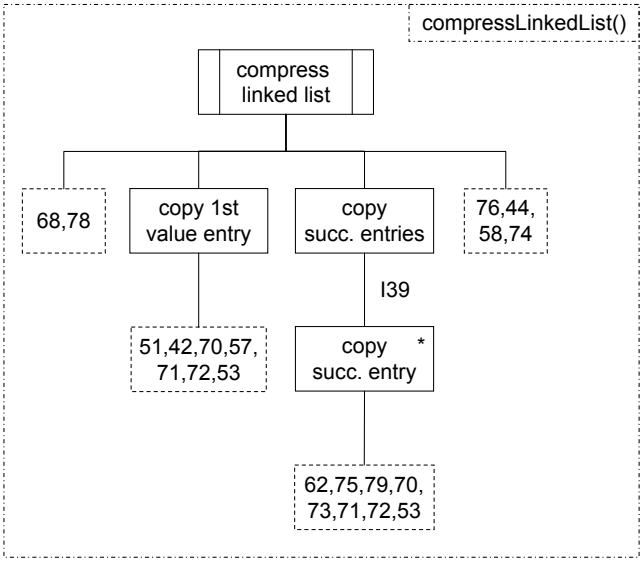


Abbildung 7.6: *Gleichsetzung des Struktogramms „compress linked list" mit dem Modul* `compressLinkedList()`

7.1.7 Add / Amend 1st Value Entries

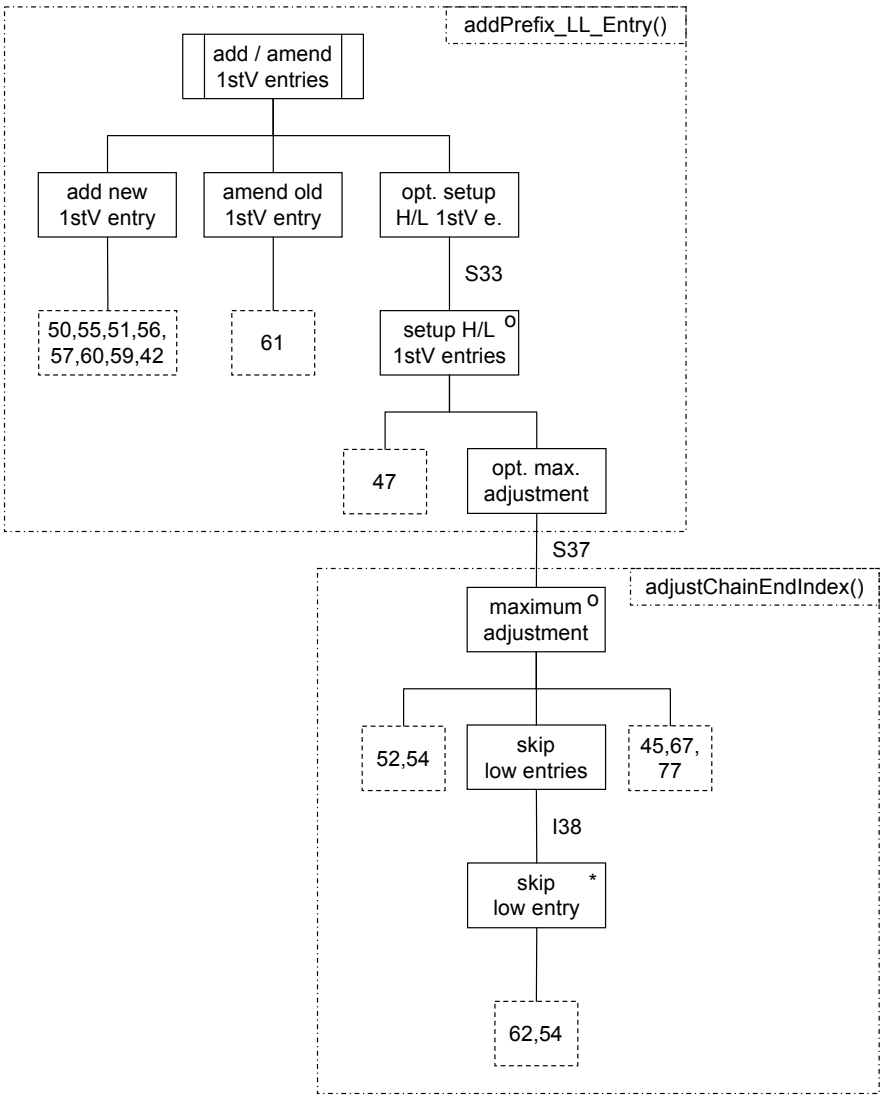


Abbildung 7.7: *Aufteilung des ausgelagerten Struktogramms „add / amend 1stV(alue) entries" in zwei Module, um „maximum adjustment" auszugliedern, das auch von „add later entry" referenziert wird*

7.1.8 Add Later Entry

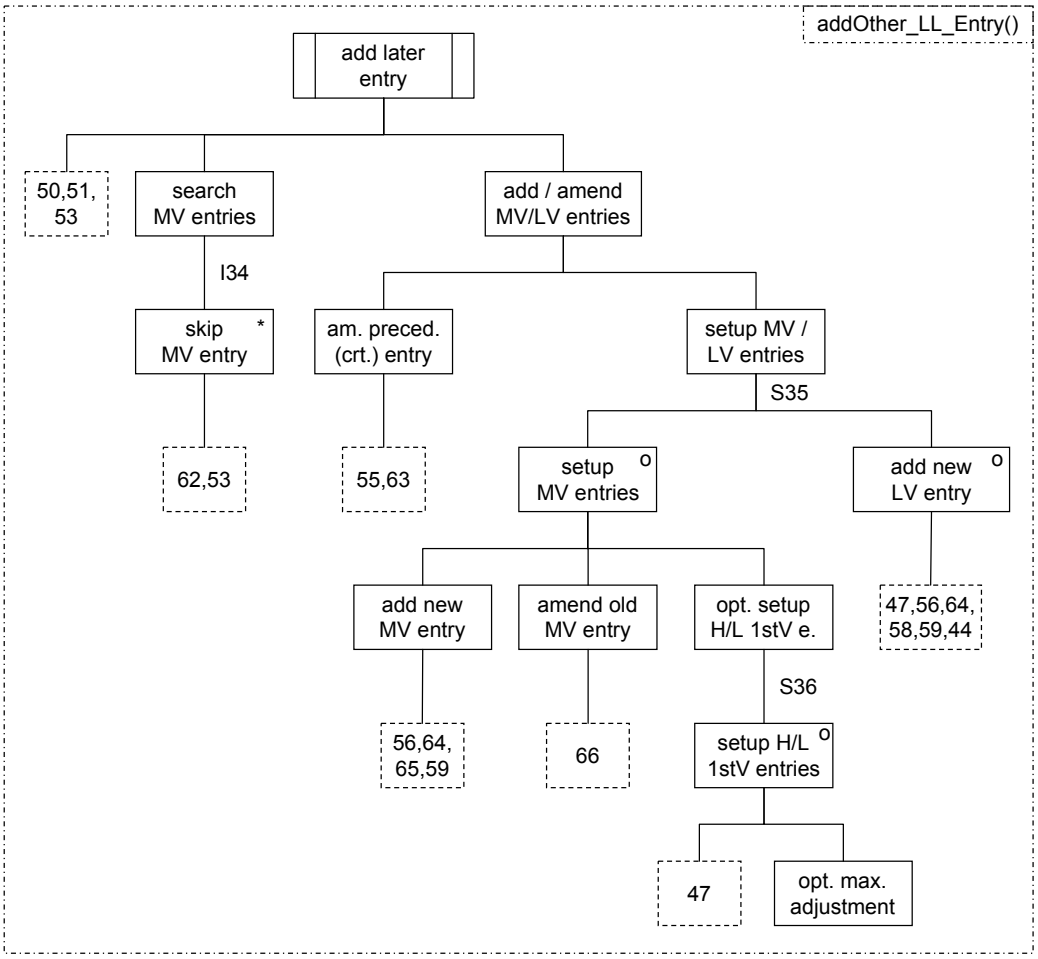


Abbildung 7.8: *Gleichsetzung des ausgelagerten Struktogramms „add later entry", das „maximum adjustment" ebenfalls verwendet, mit dem Modul* `addOther_LL_Entry()`

7.2 Aufruf-Hierarchie

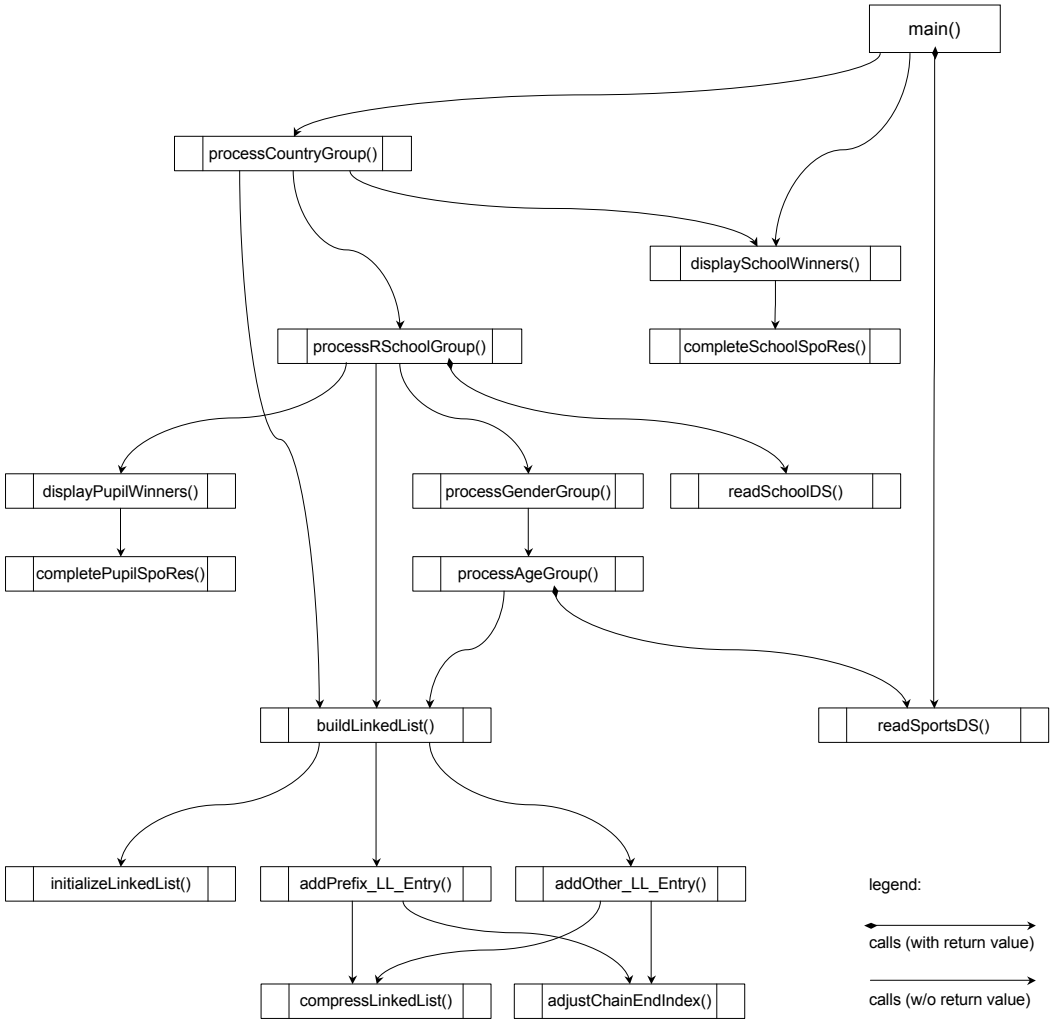


Abbildung 7.9: *Aufruf-Hierarchie der C-Variante von EvaluateSportsDS*

Hinweis: Die Hilfs-Unterprogramme `itoa()` und `substr()` werden hier nicht berücksichtigt, da zu ihnen zwar sehr viele Linien führen würden, aber die Grafik dadurch schwer lesbar und der Erkenntnisgewinn gering wären.

Diese Grafik ist weit übersichtlicher als die vorhergehenden Zuordnungen der Strukturblöcke zu den Modulen. Sie zeigt zum einen die Aufrufschachtelung der Rangverarbeitungen und zum anderen die Link-Listen-Funktionalität als separate UP-Gruppe mit dem einzigen Einsprungspunkt `buildLinkedList()`. Die beiden display-UPs greifen zwar auf die Link-Listen zu, haben aber keinen direkten Kontakt zur Logik von „build linked list". Das ist nur deshalb möglich, weil die Link-Listen jeweils das Eigentum der folgenden Komponenten sind: `main()` / `processCountryGroup()` / `processRSchoolGroup()`.

Die meisten Aufrufe liefern keine Funktionsresultate zurück. Die einzigen Ausnahmen sind die Lesemodule, aus denen jeweils die Strukturen mit den konkatenierten Schlüsseln rückgemeldet werden.

7.3 Moment mal!

… möchten Sie vielleicht an dieser Stelle ausrufen. Wie ist diese Aufrufhierarchie entstanden und mit welcher Legitimation werden hier die Boxen und die anderen, JSP-untypischen grafischen Elemente eingesetzt?

Diese Fragen sind keineswegs einfach zu beantworten. Lassen Sie uns mit den Symbolen beginnen und die schwierigere Frage nach der Entstehung der Aufrufhierarchie zurückstellen.

7.3.1 Symbole und Bezeichnungen der JSP-Methode

Die JSP-Methode verwendet konsequent nur ganz wenige grafische Elemente, von denen Sie die meisten schon kennengelernt haben, also zunächst die Strukturblöcke für Folge, Iteration und Auswahl, sowie die Verbindungslinien zwischen ihnen, welche die „Besteht aus"-Relation symbolisieren. Zur Kennzeichnung von Iteration und Auswahl sind die Buchstaben I und S, sowie die Zeichen * und o vorgesehen. Nummern dienen dazu, die Kontrollen voneinander zu unterscheiden. Sie können auch – sehr kurze – Texte dafür verwenden.

Zur Darstellung von Korrespondenzen dienen entweder – gestrichelte – Verbindungslinien, oder kleine Kreise an den Ecken der miteinander korrespondierenden Strukturblöcke, die identische Nummern enthalten.

Bei der Programmstruktur treten die Operationskästchen hinzu, in denen die Operationen in der vorgesehenen Ausführungsfolge aufgezählt werden. Das kann verbal oder mit Hilfe der Operationsnummern geschehen. Dagegen sind die Bezeichnungen der Strukturblöcke immer verbal. Da sie sehr kurz und prägnant zugleich sein müssen, sind Abkürzungen üblich. Die für diese Bezeichnungen geltenden Regeln sollen hier nicht wiederholt werden. Es liegt jedenfalls kein hoher Formalisierungsgrad vor.

Bei der Programminversion kommen noch kleine Pfeile hinzu, die auf Operationskästchen gerichtet werden, um Einsprung, Aussprung und Rückkehr an dieselbe Stelle im Code zu symbolisieren. Auch beim Backtracking werden Pfeile verwendet.

Ein weiteres Symbol dient in dieser Buchreihe dazu, ausgelagerte Struktogrammteile zu kennzeichnen. Es kommt aus der Flussdiagrammtechnik und bedeutet „vordefinierter Prozess". Die Anwendung dieses Symbols auf Datenstrukturen mag etwas befremdlich sein, folgt aber konsequent der JSP-Grafiksprache, die für abstrakte Datenelemente oder -gruppen und für abstrakte Programmelemente dasselbe Strukturblock-Rechteck verwendet.

7.3.2 Vergleich zu verwandten Symbolsprachen

Im Vergleich mit der Fülle von Symbolen, die im Rahmen der Objektorientierung verwendet werden – zum Beispiel in den UML®-Diagrammen – ist die JSP-Symbolsprache geradezu kümmerlich unterentwickelt. Ihr fehlt einerseits der Anschluss nach „oben", also an die Ebene des Systementwurfs, wo die Interaktionen der Komponenten eines Anwendungssystems untereinander und mit den externen Partnern modelliert werden. Letztere sind die umgebenden Partnersysteme und seine Benutzer, wenn eine Mensch-Maschine-Kommunikation vorgesehen ist. Hierfür besitzt die JSP-Methode überhaupt kein Konstrukt. Das von Michael A. Jackson später zusammen mit John Cameron entwickelte und propagierte Jackson System Development (JSD) konnte diese Lücke nicht schliessen und war letztlich erfolglos, weil es zu einseitig auf der Idee des Zustandsvektors aufbaute. Solchen Ensembles zustandsbeschreibender Daten werden wir bei der Programminversion begegnen. Sie stellen das Gedächtnis dar, in dem zwischen zwei Aufrufen einer invertierten Routine deren Zustandsinformation gehalten wird.

Fairerweise ist zur Verteidigung der JSP-Methode anzuführen, dass sie nie dazu gedacht war, dem Systementwurf zu dienen. Sie war und bleibt immer nur ein Weg, um Programmstrukturen aus Datenstrukturen herzuleiten und die eigentliche Implementierung intensiv vorzubereiten. Den Methodenbruch von der Ebene der strukturierten Analyse zur Ebene des Programmdesigns hat die JSP-Methode nicht zu verantworten. Sie ist vorher erfunden worden.

Auch nach „unten", also zur Implementierungsebene hin, fehlt der JSP-Methode eine zwingende Verbindung. Der Pseudocode ist allenfalls ein Hilfsmittel ohne gestaltende Kraft. Weder die monolithische Pseudocodierung noch die Aufsplitterung in zahlreiche flache, strukturblockbezogene Pseudocode-Sequenzen vermögen zu überzeugenden Ergebnissen zu führen. Es fehlt der Kitt, der die erforderliche Kohäsion zwischen zusammengehörenden Code-Abschnitten herstellen kann, die anschließend beispielsweise als Unterprogramme realisiert werden. Diese Kohäsion ist dagegen bei der objektorientierten Programmentwicklung reichlich vorhanden , nämlich in den Klassen selbst, ihren Abhängigkeiten voneinander (Vererbung, Relationen zwischen Klassen), sowie der engen Verschränkung der Methoden sowohl untereinander als auch mit den Klassen-Interfaces.

Der zusätzliche Modellierungsschritt beim Übergang von der Programmstruktur mit Operationen zum Klassendiagramm setzt genau hier an. Er liefert die Kohäsionsinformationen, die für die objektorientierte Implementierung zwingend erforderlich sind, will man nicht in Beliebigkeit enden. Der bei JSP fehlende innermethodische Anschluss der Modellierung an die Implementierung erweist sich in diesem Umfeld als Vorteil, weil er die OO-Implementierung keinen sachfremden Zwängen unterwirft.

7.3.3 Michael A. Jackson und seine „Codierknechte"

Der scheinbar einfachere Fall, freie Hand für irgendeine algorithmische Implementierung zu haben, erweist sich so auf den zweiten Blick als der tückischere. Nun kommt nämlich noch das empfindliche Programmierer-Ego ins Spiel, dem schon die Ergebnisse der JSP-basierten Modellierung suspekt sind. Nicht selten fühlen sich algorithmisch arbeitende „Künstler" durch die Vorgabe einer Programmstruktur mit Operationen und Kontrollen zu Codierknechten herabgewürdigt, deren einzige Aufgabe darin besteht, die Spezifikation 1:1 in Code umzusetzen. Hier liegt gleich ein mehrfaches Missverständnis vor:

- Programmieren und Codieren werden oft fälschlicherweise gleichgesetzt, denn das Programmieren umfasst auch das Modellieren eines Programms. Eine gute JSP-basierte Lösung zu finden, kann bei vertrackten Problemen durchaus schwieriger als das Codieren selbst sein, weil dieses erst dann beginnt (oder: beginnen sollte!), wenn die Lösung fertig erarbeitet ist.
- Das Fehlen von Leitplanken, die den dahinrasenden algorithmischen Programmierer oder die hurtige Programmiererin vor dem Absturz in einen Code-Sumpf bewahren könnten, mahnt nach unzähligen Fehlschlägen zu besonnenem, umsichtigem Vorgehen. Ein gut gebautes Programm entsteht anhand einer Programmstruktur mit erheblich höherer Erfolgsaussicht als ohne. Die richtige Gliederung des Codes in Hauptprogramm und Unterprogramme, die passenden Definitionen der Datenelemente und ihrer Aggregate, die geschickte Gestaltung der Aufrufschnittstellen, die Nutzung von naheliegenden Generalisierungen, das Einbinden vorhandener Software, die kundige Verwendung der jeweiligen Programmiersprache, performante Realisierung, Einrückungen, Kommentare und anderes mehr formen schließlich ein respektables Stück Software, das auch Jahre später noch fehlerfrei seinen Dienst tut und – hoffentlich – mit demselben hohen Anspruch gepflegt wird. Es gibt also auch bei einer präzisen JSP-Vorgabe noch viel Profilierungspotential!

7.3.4 Grafisches

Die Entlehnung des Symbols„vordefinierter Prozess“ als Kennzeichnung ausgelagerter Struktogrammteile kann beim Übergang von der JSP-Modellierung zur Implementierung Verwirrung stiften, weil es einen UP-Aufruf oder UP-Kopf zu bezeichnen scheint. Daher ist es wichtig, in Grafiken, die Zuordnungen von Strukturblöcken zu Modulen – Unterprogrammen – zeigen sollen, jede Vermischung zu vermeiden. Aus diesem Grund sind die Modulgrenzen durch strichpunktierte Linien gekennzeichnet. Die Modulnamen sind in den dadurch aufgespannten Rahmen oben oder unten rechts in einem eigenen kleinen strichpunktierten Kästchen angeordnet. Sie können bei Auslagerungen mehrfach auftreten.

Eine Ausnahme davon bilden die complete-Module, deren Operations-Inhalt zwar strichpunktiert umrahmt ist, deren Namen aber über Verbindungslinien damit verknüpft sind. Das geschieht allein aus grafischen Gründen, hat also nichts mit der Implementierung zu tun. Bei Handzeichnungen empfiehlt es sich, anstelle der Strichpunktierung verschiedenfarbige Linien zu verwenden

An mehreren Stellen ist eine Iteration auf zwei Module verteilt. Ein dafür typische Beispiel zeigt das Kapitel „7.1.3 Process Gender Group / Process Age Group“ auf Seite 141 ff. Die – invertierte – Iteration I5 und der darin enthaltene Aufruf sind wie folgt implementiert:

```
while (strncmp(old_rank_1_3_key, pS_RK_New->rank_1_3_key, 15) == 0)
{
   processAgeGroup(pF_SportsDS,
                   pS_SpoConD,
                   pS_RK_New,
                   pS_Pupil_LL,
                   pS_CntRes);
};
```

Der Schleifenkopf entspricht dem Strukturblock „process age groups“, der dem Modul `processGender-Group()` zugeordnet ist. „Process age group“ repräsentiert den Schleifenrumpf, der als `processAgeGroup()` implementiert wird. Die Grafik zeigt I5 zwischen den beiden Modulen, um den Aufruf in der Schleife hervorzuheben.

7.4 Entstehung der Aufrufhierarchie

Die Daten- und Programmmodellierung nach Jackson hat etwas Zwangsläufiges. Sind die Strukturen richtig verstanden worden, so werden zwei Analytiker(innen) dasselbe Problem hochwahrscheinlich mit ähnlichen Ergebnissen lösen. Ähnlich, aber nicht gleich, denn es werden sich mindestens die Bezeichnungen der Strukturblöcke, die Nummerierung der Kontrollen und der Operationen, sowie die Namen der bearbeiteten Datenelemente voneinander unterscheiden. Ausserdem kann sich die Granularität – der Detaillierungsgrad – von Operationen unterscheiden. Das ist aber unwichtig, solange sich keine wesentliche inhaltliche Differenz einschleicht, was sich jedoch durch kritisches Vergleichen erkennen und auflösen lässt.

Diese banal erscheinende Beobachtung ist in Wirklichkeit für den System- und Programmentwurf fundamental wichtig. Edward Yourdon beschreibt in [1] im Kapitel 19 „Building a Preliminary Behavioural Model“ unter 19.1 „The Classical Approach“ auf der Seite 360 das Sechs-Analytiker-Problem, das von einer an formalen Zuständigkeiten orientierten Arbeitsteilung bei der Analyse eines Gesamtsystems zu sechs Teilsystem-Modellierungen führt, wobei es je nach Anzahl der beteiligten Systemanalytiker auch mehr oder weniger sein könnten. Er stellt diesem konkurrenzgetriebenen Phänomen seine Modern Structured Analysis (MSA) entgegen, die personenunabhängige Ergebnisse zu liefern verspricht. JSP und MSA haben somit gemeinsam, eine

stringente Vorgehensweise anzubieten, von der zu erwarten ist, dass auch komplexe Probleme von verschiedenen Personen auf nahezu dieselbe Weise gelöst werden. Das ist für die wirtschaftliche Systementwicklung von hoher Bedeutung.

Diese Vorhersagbarkeit endet – nicht nur in der algorithmischen Programmierung – beim Übergang vom Modell zum Code. Ein- und dasselbe Modell kann auf zahlreiche verschiedene Weisen implementiert werden, wenn es nur umfangreich genug ist – und das ist im vorliegenden Fall gegeben. Daher besitzt jede Codestruktur ein Element des Willkürlichen und kann somit zum Objekt heftiger Diskussionen zwischen Entwicklern werden. Wer schon die Ergebnisse der JSP-Modellierung als Einengung seiner Programmierfreiheit auffasst, wird noch unwirscher reagieren, wenn jemand behauptet, auch für die Abbildung in den Code gebe es Gesetzmäßigkeiten.

So etwas Böses tun wir hier natürlich nicht… oder vielleicht doch? Die Struktogramme liefern uns jedenfalls zahlreiche Anhaltspunkte für sinnvolle Segmentierungen in C-Module:

- Aus der Top-Programmstruktur geht die `main()`-Routine hervor. Diese sollte nur die wesentlichsten Operationen und Aufrufe enthalten, die den Einstieg in das eigentliche Programm prägen. Umfangreiche Initialisierungen und Terminierungen sind daher Auslagerungskandidaten.
- Jede Iteration, mit der eine neue Ebene der Ranggruppenverarbeitung betreten wird, definiert eine mögliche Schnittstelle zwischen zwei Modulen. Mehrere Beispiele dafür finden sich in den obigen Grafiken. Verzichtet man darauf, sind unübersichtliche – geschachtelte – Schleifen unvermeidbar.
- Auslagerungen gemeinsamer Struktogrammteile implizieren, diese Teile als Module zu formulieren, die an den Auslagerungsstellen aufgerufen werden.
- Identische Code-Sequenzen machen den Code länger und können bei der Programmpflege auseinanderlaufen. Vergessene Änderungen sind gefährlich und Doppelarbeit ist ärgerlich. Daher lohnt sich die Auslagerung solcher Sequenzen in eigene Module, auch wenn diese keine hohe Festigkeit aufweisen, also wenig Zusammenhang zwischen den betreffenden Operationen besteht.
- Module beziehungsweise Unterprogramme sollen nicht zu lang werden, weil sonst die Übersichtlichkeit leidet. Daher sind vertikale Schnitte durch die Programmstruktur beispielsweise unterhalb von „build linked list“ geführt worden, nachdem eine erste Version sich als viel zu umfangreich erwiesen hatte.
- Module werden gegebenenfalls von vornherein als Quasi-Operationen gekapselt, wie beispielsweise „compress linked list“. Auch Eingabeoperationen sind häufig so komplex, dass sie in eigene Module gehören.
- Thematisch andergeartete Verarbeitungsabschnitte, etwa „gen(erate) pupil winner gr(ou)p“ im Struktogramm von „process r-school group“, eignen sich ebenfalls für die Auslagerung in eigene Module.

Exzessive Modularisierung hat andererseits Nachteile zur Folge, weil die Übersichtlichkeit verlorengeht und die Performance unter dem maschinellen Mehraufwand leidet, den jeder Unterprogrammaufruf erfordert.

Ein weiterer Problempunkt kann sein, dass Aufrufschnittstellen ungewöhnlich breit sind, also sehr viele Elemente respektive Strukturen übergeben werden müssen. Das geht auf jeden Fall zu Lasten der Performance. Zudem wird der eigentlich angestrebte Zweck, möglichst übersichtliche und treffend formulierte Module zu formen, dadurch verfehlt. Mit jedem übergebenen Parameter muss ja etwas getan werden. Daher sind solche Schnittstellen nur als Notlösungen für Fälle anzusehen, in denen physische Grenzen erreicht werden, beispielsweise die maximale Größe eines Lademoduls.

Wenn wir aber ehrlich zu uns sind, müssen wir hier bekennen, dass diese Modularisierungskriterien-Aufzählungsfleissarbeit nicht den hohen Erwartungen entspricht, die wir eigentlich daran stellen müssen. Zu vieles bleibt offen, manche Kriterien widersprechen einander sogar tendenziell, Performance kollidiert mit Kapselung, alles ist sehr kleinteilig. Es fehlt der große Wurf, also ein leitendes Prinzip. Solche Prinzipien sind in der Geschichte der Informatik zahlreich kreiert und in Wellen sich ausbreitend propagiert worden. Stets war

damit die Heilserwartung verbunden, endlich die Probleme in den Griff zu bekommen, die vor allem in großen Projekten und unter sich ständig wandelnden Anforderungen die Entwickler(innen) plagen.

Erst die strukturierte, dann die normierte, funktionale, generative, aspektorientierte, prozedurale, modulare, deklarative, objektorientierte, agile Programmierung – und viele andere Ansätze mehr – sollten jeweils *die* Lösung sein, und doch ist kein Land in Sicht. Die Informatik ist immer noch eine sehr junge Wissenschaft, die in den kommenden Jahrhunderten und Jahrtausenden gewiss zahlreiche weitere Paradigma, Prinzipien, Methoden, Verfahren, Sprachen etcetera in die Welt setzen wird. Das nützt uns aber jetzt leider gar nichts. Daher folgt hier ein ganz unwissenschaftlich erscheinender Vorschlag für die aktuelle Aufgabe, umfangreiche Struktogramme in gut geschnittene Module umzusetzen:

Das Prinzip der Eleganz

Hier ist natürlich nicht die modische **Eleganz** gemeint, auf welche dieser Begriff häufig reduziert wird. In Wikipedia lesen wir am Ende der Begriffserklärung[1] folgendes:

> »Meyers Großes Konversations-Lexikon definierte 1905 Eleganz als Zierlichkeit, Anmut sowie „in sprachlicher Hinsicht" als schon „bei den Römern die mit Klarheit verbundene Korrektheit der Rede, so daß der Ausdruck das Gedachte treu und wahr wiedergibt und zugleich grammatikalisch richtig, natürlich, angemessen und treffend ist." Im weiteren Sinne bezeichnet Eleganz „überhaupt dasjenige, was den Eindruck des Wohlgefälligen macht, besonders mit dem Nebenbegriff des Neuen und Modemäßigen; so namentlich in der Kleidung, in der häuslichen Einrichtung etc." Paul Valéry hat 1922 allgemeiner folgendermaßen formuliert: „Elegantia – Das bedeutet Freiheit und Ökonomie ins Sichtbare übertragen – Ungezwungenheit, Leichtigkeit in schwierigen Angelegenheiten. Finden, ohne den Anschein zu erwecken, gesucht zu haben – Wissen, ohne offenzulegen, daß man gelernt hat."«

Dieser Abschnitt ist natürlich nur im übertragenen Sinn dazu geeignet, das Phänomen „Eleganz" im Kontext der strukturierten Programmierung annähernd zu beschreiben. Damit ein Programm als elegant bezeichnet werden darf, muss es diversen Anforderungen in überzeugender Weise gerecht werden. Es muss Spass machen, es zu studieren und dabei zu verstehen, warum es gerade in dieser Form geschaffen wurde. Das Rohmaterial, also die zur Umsetzung der Operationen und Kontrollen vorhandenen Sprachmittel, ist seinerseits in vielen Sprachen nicht elegant. Die Kunst des Programmierens besteht an dieser Stelle darin, dieses Rohmaterial – immer streng an den per JSP erarbeiteten Vorgaben ausgerichtet – so zu gliedern, dass ein technisch und ästhetisch zugleich ansprechendes Programm entsteht.

Die geschickte Modularisierung ist dabei nur ein – wenngleich sehr wichtiges – Thema. Eine ähnlich große Rolle spielt die Wahl der geeigneten Namen für Datenelemente, Strukturen und Module. Alles, was die Lesbarkeit und Verständlichkeit erhöht, ohne überladen zu wirken, prägt ebenfalls den Gesamteindruck positiv. Die korrekte Umsetzung der Operationen und Kontrollen ist in diesem Kontext nahezu eine Selbstverständlichkeit, auch wenn wir Informatiker uns mit dem Schreiben fehlerfreien Codes immer noch sehr schwer tun und dieses Ziel oftmals nicht erreichen. Immerhin erhöht die JSP-Methode die Wahrscheinlichkeit für Zero Defect gegenüber konkurrierenden Ansätzen ganz wesentlich. In mehreren Jahrzehnten der Programmierpraxis des Autors hat es aus diesem Grund nie einen schwerwiegenden Fehler in einem eigenen Produktionsprogramm gegeben.

Ganz klar, es ist ein Anspruch, das eigene Werk als elegant zu bezeichnen. Es wird immer Leute geben, die das bestreiten und Kritik an der jeweiligen Realisierung üben. Das sollte Sie aber nicht daran hindern, diesen Anspruch für sich selbst zum leitenden Prinzip zu machen. Berechtigte Kritik kann dann als Ansporn zur eigenen Weiterentwicklung – anstatt als narzisstische Kränkung – aufgefasst werden. Wer hohe Erwartungen an das Ergebnis der eigenen Arbeit stellt und diese konsequent umsetzt, wird mit Sicherheit weit höhere Qualität

liefern, als jemand, der das Programmieren vorwiegend als einen von vielen Wegen zum Geldverdienen ansieht. Wenn Sie nach mehreren Jahren eines Ihrer Programme erneut studieren und mit Ihrem Werk immer noch vollauf einverstanden sind, dann haben Sie dieses Qualitätsziel erreicht.

So wurden letztlich die Aufrufhierarchie und der Zuschnitt der Module am Prinzip der Eleganz ausgerichtet. Nach jedem großen Implementierungsabschnitt fand eine kurze Code-Revision statt, um das Erreichte kritisch zu prüfen und unelegante Passagen zu identifizieren. Diese wurden anschliessend so lange modifiziert, bis der Gesamteindruck stimmte. Dieses iterative Vorgehen führt mit geringerem Aufwand zum Ziel als ein abschliessender Verschönerungsversuch, der unter Zeitdruck und zunehmender Lustlosigkeit zu leiden droht.

Apropos Schönheit: Wenn Ihnen jemand sagt, dass es nicht beabsichtigt sei, „in Schönheit zu sterben", dann sollten Sie hellwach werden: Ihre Qualitätskriterien und die Vorstellungen dieser Person kollidieren miteinander. Sie können dann entweder Abstriche machen und sich an die durchschnittliche Qualität des Teams anpassen, was Ihnen das Verbleiben im Projekt sichert, aber ein schales Gefühl hinterlässt, oder sich alsbald nach einer neuen Umgebung umsehen, in der deutlich mehr Wert auf gute Ergebnisse gelegt wird. Das ist sicherlich der mühsamere, aber letztlich beruflich weit befriedigendere Weg.

7.5 C-Code von EvaluateSportsDS

7.5.1 Globale Deklarationen

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Evaluate Sports Dataset Program

#define CR_INIT       {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
#define ESK_INIT      {" ", "    ", "  ", "         ", "  ", ""}
#define LL_HDR_INIT   {0, 0, 0, 0, 0, ' '}
#define RK_INIT       {" ", "    ", "  ", "         ", " ", "  ", "  ", "", "", "", ""}
#define SCD_INIT      {"   ", "  ", "        ", " ", "  ", "  ", "  ", "  ", "  ", "  ", ""}
#define SIDD_INIT     {"             ", "", "", ""}

#define FALSE         0
#define LL_LH_CEIL    ((LL_MAX + 1) / 2) - 1
#define LL_MAX        199
#define MAX_UNIQ_VAL  3
#define TRUE          1

#define TRACE_MODULE  0  // trace module names
#define TRACE_DETAIL  0  // trace logic details
#define TRACE_LL      0  // trace linked list output + LL functions
#define TRACE_INPUT   0  // trace input functions

#define ZERO                0
#define BLANK_LEN           1
#define COUNTRY_LEN         3
#define REGION_ID_LEN       2
```

```
#define SCHOOL_NO_LEN        8
#define SCHOOL_ID_LEN        (COUNTRY_LEN          + \
                              REGION_ID_LEN        + \
                              SCHOOL_NO_LEN)
#define LINE_NO_LEN          2
#define SCHOOL_LINE_KEY_LEN  (SCHOOL_ID_LEN        + \
                              LINE_NO_LEN)
#define GENDER_CODE_LEN      1
#define AGE_LEN              2
#define SPRINT_POINTS_LEN    2
#define JUMP_POINTS_LEN      2
#define THROW_POINTS_LEN     2
#define MAX_NAME_LEN         47
#define MAX_ADDR_LEN         64
#define MAX_SCHOOL_VAL_LEN   27

// pupil-related offset calculations (key data + nonkey attributes)
#define P_COUNTRY_OFFSET     ZERO
#define P_REGION_ID_OFFSET   (P_COUNTRY_OFFSET     + \
                              COUNTRY_LEN)
#define P_SCHOOL_NO_OFFSET   (P_REGION_ID_OFFSET   + \
                              REGION_ID_LEN)
#define GENDER_CODE_OFFSET   (P_SCHOOL_NO_OFFSET   + \
                              SCHOOL_NO_LEN        + \
                              BLANK_LEN)
#define AGE_OFFSET           (GENDER_CODE_OFFSET   + \
                              GENDER_CODE_LEN      + \
                              BLANK_LEN)
#define SPRINT_POINTS_OFFSET (AGE_OFFSET           + \
                              AGE_LEN              + \
                              BLANK_LEN)
#define JUMP_POINTS_OFFSET   (SPRINT_POINTS_OFFSET + \
                              SPRINT_POINTS_LEN    + \
                              BLANK_LEN)
#define THROW_POINTS_OFFSET  (JUMP_POINTS_OFFSET   + \
                              JUMP_POINTS_LEN      + \
                              BLANK_LEN)
#define PUPIL_NAME_OFFSET    (THROW_POINTS_OFFSET  + \
                              THROW_POINTS_LEN     + \
                              BLANK_LEN)

// school-related offset calculations (key data + nonkey attributes)
#define S_COUNTRY_OFFSET     ZERO
#define S_REGION_ID_OFFSET   (S_COUNTRY_OFFSET     + \
                              COUNTRY_LEN)
#define S_SCHOOL_NO_OFFSET   (S_REGION_ID_OFFSET   + \
                              REGION_ID_LEN)
```

```
#define LINE_NO_OFFSET       (S_SCHOOL_NO_OFFSET   + \
                              SCHOOL_NO_LEN)
#define SCHOOL_NAME_OFFSET   (LINE_NO_OFFSET       + \
                              LINE_NO_LEN          + \
                              BLANK_LEN)
#define SCHOOL_ADDR_OFFSET   (LINE_NO_OFFSET       + \
                              LINE_NO_LEN          + \
                              BLANK_LEN)
#define SCHOOL_VALUES_OFFSET (LINE_NO_OFFSET       + \
                              LINE_NO_LEN          + \
                              BLANK_LEN)

extern int errno;

struct S_C_D                // sports contest data structure
{
   char   country_code[4];
   char   region_id[3];
   char   school_number[9];
   char   gender_code[2];
   char   age[3];
   char   sprint_points[3];
   char   jump_points[3];
   char   throw_points[3];
   char   name[48];
};

struct R_K                  // sports contest rank key structure
{
   char   read_status[2];
   char   country_code[4];
   char   region_id[3];
   char   school_number[9];
   char   gender_code[2];
   char   age[3];
   char   rank_1_key[5];    // read_status & country_code
   char   rank_1_2_key[15]; // read_status thru school_number
   char   rank_1_3_key[16]; // read_status thru gender_code
   char   rank_1_4_key[18]; // read_status thru age
};

struct E_S_K                // extended school key structure
{
   char   read_status[2];
   char   country_code[4];
   char   region_id[3];
   char   school_number[9];
```

```
   char   line_no[3];
   char   e_school_key[15]; // read_status thru school_number
};

struct SID_D                // school ID & data structure
{
   char   school_ID[14];
   char   name[51];
   char   address[65];
   char   values[28];
};

struct L_L_E                // linked list entry structure
{
   int    entry_value;
   int    uplink;
   int    downlink;
   struct S_C_D  Pupil_Data;
   struct SID_D  School_Data;
};

struct L_L                  // linked list structure
{
   int    list_level;
   int    max_list_level;
   int    chain_top_index;
   int    chain_end_index;
   int    unique_values;
   char   contestant_type;
   struct L_L_E  LLE[LL_MAX];
};

struct C_R                  // counter / result structure
{
   int    countries;
   int    total_schools;
   int    schools;
   int    F_age_groups;
   int    M_age_groups;
   int    F_pupils;
   int    M_pupils;
   int    school_points;
   int    sports_recs;
   int    school_recs;
   int    result_recs;
};
```

```
void processCountryGroup( FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          FILE         * pF_SchoolDS,
                          struct L_L   * pS_I_School_LL,
                          struct C_R   * pS_CntRes,
                          FILE         * pF_SchoolRes);

void processRSchoolGroup( FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          FILE         * pF_SchoolDS,
                          struct L_L   * pS_N_School_LL,
                          struct C_R   * pS_CntRes,
                          FILE         * pF_SchoolRes);

void processGenderGroup(  FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          struct L_L   * pS_Pupil_LL,
                          struct C_R   * pS_CntRes);

void processAgeGroup(     FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          struct L_L   * pS_Pupil_LL,
                          struct C_R   * pS_CntRes);

void buildLinkedList(     int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList);

void initializeLinkedList(int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList);

void addPrefix_LL_Entry(  int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList);

void addOther_LL_Entry(   int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList);
```

```
void adjustChainEndIndex( struct L_L   * pS_LinkList);

void compressLinkedList(  struct L_L   * pS_LinkList);

void displayPupilWinners( FILE         * pF_SchoolRes,
                          struct C_R   * pS_CntRes,
                          struct L_L   * pS_LinkList);

void completePupilSpoRes( FILE         * pF_SchoolRes,
                          char           SpoRes[],
                          struct S_C_D * pS_Pupil_Data,
                          int            point_sum);

void displaySchoolWinners(_Bool          National,
                          FILE         * pF_SchoolRes,
                          struct C_R   * pS_CntRes,
                          struct L_L   * pS_LinkList);

void completeSchoolSpoRes(FILE         * pF_SchoolRes,
                          char           SpoRes[],
                          char           prefix[],
                          struct SID_D * pS_School_Data,
                          int          * pi_line_no);

#if TRACE_LL
void displayLinkedList(   struct L_L   * pS_LinkList);
#endif

struct R_K readSportsDS(  FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD);

struct E_S_K readSchoolDS(FILE         * pF_SchoolDS,
                          char           school_rec[]);

void itoa(int value, char result[], int digits);

void substr(char dest[], char src[], int offset, int len);
```

Anmerkungen

Dieser Code-Abschnitt umfasst die `#include`- und `#define`-Anweisungen, die `struct`-Definitionen und die Deklarationen der Unterprogramme, also der Prozeduren und Funktionen.

Die `#define`-Anweisungen sind wiederum in Initialisierungsangaben für `struct`-Deklarationen in den Modulen und in Einzelwertzuweisungen gegliedert. `#define` bewirkt lediglich, dass der C-Compiler die darauf folgende Bezeichnung – in Großbuchstaben – durch den rechts davon stehenden String ersetzt, wenn er bei der lexikalischen Code-Analyse darauf stößt. Das erscheint auf den ersten Blick nicht als beeindruckend, ist

aber extrem nützlich, um programmweit einheitliche Vorgaben zu machen, die sogar aufeinander aufbauen können, wie zum Beispiel folgende Zeilen zeigen:

```
#define LL_LH_CEIL    ((LL_MAX + 1) / 2) - 1
#define LL_MAX        199
```

Zahlreiche `#define`-Zeilen dienen dazu, Magic Numbers etwa bei `substr()`-Aufrufen zu vermeiden. Auch diese Definitionen bauen teilweise aufeinander auf.

Speziell bei verteilten Deklarationen von Strukturen, die auf denselben `struct`-Definitionen basieren, sind die zentral vorgegebenen Initialisierungsterme wertvoll. Nach der Änderung einer `struct`-Definition muss nur die zugehörige `INIT`-Vorgabe angepasst werden, anstatt an mehreren Stellen dieselben Änderungen vornehmen zu müssen.

Auch `struct`-Definitionen können geschachtelt werden, wie das Beispiel `L_L_E` / `L_L` zeigt. Dadurch wird – hier in Kombination mit dem `#define LL_MAX` – eine übersichtliche, flexible Darstellung der zentralen Link-List-Struktur erzielt. Im einzelnen erscheint zu den `struct`-Definitionen folgendes als erwähnenswert:

- `struct S_C_D`

  Diese Struktur ist dafür vorgesehen, die individuellen Daten zu jedem Schüler aufzunehmen. Die Blanks in den „sports contest records" sind hier nicht mehr vorhanden. Das Alter und die Punktzahlen sind als Strings definiert, weil dies die überwiegende Nutzungsform dieser Angaben ist. Nur an einer einzigen Stelle, nämlich beim Berechnen der jeweiligen Punktsumme, muss jede Punktzahl in ihre Integer-Form umgewandelt werden.
- `struct R_K`

  Die Rangschlüsselstruktur erscheint gegenüber den Gruppenschlüsselstrukturen in den Kapiteln „3.1 Globale Deklarationen" auf Seite 53 ff und „5.1 Globale Deklarationen" auf Seite 99 ff als wesentlich umfangreicher. Es sind nicht nur mehr Schlüsselbestandteile vorhanden, sondern auch je ein konkatenierter Rangschlüssel-String für jeden Rang. Auch hier gibt es einen speziellen Grund dafür, das Alter als Ziffern-String und nicht als Zahl zu behandeln: Je nach der physischen Implementierung der Zahlen auf dem ausführenden Computer kommt es auf dessen „Endianness" an, also darauf, ob eine Zahl im Big-Endian-Format oder im Little-Endian-Format abgespeichert wird. Das Middle-Endian-Format ist inzwischen ausser Gebrauch. Eine Zahl in einem konkatenierten String kann zwar beispielsweise in COBOL so deklariert werden, aber beim bitweisen Vergleich zweier solcher Rangschlüssel „von links nach rechts" hinge das Ergebnis von der Endianness der Maschine ab. Das aber wäre bei einer Portierung fatal.
- `struct E_S_K`

  Die „school records" besitzen keine Rangstruktur, sondern nur eine Gruppenstruktur mit je drei Sätzen. Es ist daher nur ein konkatenierter Schlüssel erforderlich, der allerdings aus relativ vielen Elementen besteht. Die `line_no` gehört nicht dazu. Sie identifiziert aber den jeweiligen Satz und wird deshalb in der Struktur mitgeführt.
- `struct SID_D`

  Die in dieser Struktur zusammengefassten Daten beschreiben die Schule als Contestant der nationalen und internationalen Link-Listen. Die Mail-Adresse wird hier nicht mehr benötigt.
- `struct L_L_E`

  Der im Unterkapitel „Definition der Link-Liste" des Band-1-A2-Kapitels „Link-Listen-Verarbeitung" gezeigte Aufbau der Link-Listen-Elemente macht keine Aussage über die Struktur der Teilnehmerdaten, da es implementierungsabhängig ist, ob diese als polymorph definiert werden dürfen, oder ob feste Strukturen gewählt werden müssen. C unterstützt keine Polymorphie; also ist je ein `struct` für Schülerdaten und für Schul-Daten anstelle einer gemeinsamen Contestant-Deklaration erforderlich.

- `struct L_L`
  Da jede Link-Liste entweder eine Schülerliste oder eine Schulliste ist, erfordert die Unterscheidung der Teilnehmerdaten das zusätzliche Datenelement `contestant_type` mit den Werten `'P'` (= Pupil) oder `'S'` (= School). Somit ist jeder Link-Liste von aussen anzusehen, zu welcher Kategorie sie gehört. Das erleichtert die Programmierung in den Link-List-Modulen und sorgt für mehr Sicherheit, als dies ständige Typübergaben per Parameter täten.
- `struct C_R`
  Dies ist der „Lumpensammler" für Zähler und Ergebniswerte. Die darin enthaltenen Variablen könnten zwar einzeln deklariert werden, aber dann müsste das Programm sie auch durch alle Modulschichten gezielt per Parameter übergeben. Das würde die Schnittstellen sehr verbreitern. Die (Re-)Initialisierungen auf den Rangstufen sorgen dafür, dass sich hier keine Schleppwerte ansammeln können.

Die Deklarationen der ausgabenspezifischen Unterprogramme zeigen deutlich, dass die `struct`-Definitionen jeweils eine übersichtliche Bündelung der vertikal gleich ausgerichteten Parameter ermöglichen. Die technischen Prozeduren `itoa` und `substr` fallen hier deutlich aus dem Rahmen. Sie wurden nötig, weil es im C99-Standard keine ähnlichen und eleganten Standardroutinen gibt.

7.5.2 main()

```
int main(int argc, char *argv[])
{
   FILE       * pF_SportsDS;
   FILE       * pF_SchoolDS;
   FILE       * pF_SchoolRes;

   struct R_K   RK_New      = RK_INIT;

   struct S_C_D SpoConD     = SCD_INIT;

   struct L_L   I_School_LL = LL_HDR_INIT;

   struct C_R   CntRes      = CR_INIT;       // op. 12,118,120,122,124

   printf("*** EvaluateSportsDS ***\n");

   switch(argc)
   {
   case 1:  printf("\n All file paths missing! \n");
            return 3;
   case 2:  printf("\t Path 1:  %s \n", argv[1]);
            printf("\n Path 2 (SchoolDS) & path 3 (SchoolRes) missing! \n");
            return 2;
   case 3:  for (int i = 1; i < argc; i++)
               { printf("\t Path %d:  %s \n", i-1, argv[i]); }
            printf("\n Path 3 (SchoolRes) missing! \n");
            return 1;
   case 4:  /* for (int i = 1; i < argc; i++)
```

```
               { printf("\t Path %d:  %s \n", i-1, argv[i]); } */
         break;
default: for (int i = 1; i < argc; i++)
            { printf("\t Path %d:  %s \n", i-1, argv[i]); }
         printf("\n Too many paths, or apostrophes missing! \n");
         return 4;
}

// op. 1: open sports dataset
pF_SportsDS = fopen(argv[1], "r");

if (pF_SportsDS == NULL)
{
   printf("SportsDS OPEN failed\n");
   printf("Error: %d (%s)\n", errno, strerror(errno));
   return 5;
};

// op. 31: open school dataset
pF_SchoolDS = fopen(argv[2], "r");

if (pF_SchoolDS == NULL)
{
   printf("SchoolDS OPEN failed\n");
   printf("Error: %d (%s)\n", errno, strerror(errno));
   return 6;
};

// op. 2: open output file
pF_SchoolRes = fopen(argv[3], "w");

if (pF_SchoolRes == NULL)
{
   printf("SchoolRes OPEN failed\n");
   printf("Error: %d (%s)\n", errno, strerror(errno));
   return 7;
};

// op. 3: read 1st sports contest record
RK_New = readSportsDS(pF_SportsDS, &SpoConD);

// input file empty?
if (strncmp(RK_New.read_status, "E", 1) == 0)
{
   printf("SportsDS empty\n");
   return 8;
};
```

```
   // op. 80
   I_School_LL.unique_values   = 0;
   I_School_LL.contestant_type = 'S';

   // I1: process country groups
   while (strncmp(RK_New.read_status, "D", 1) == 0)
   {
      processCountryGroup(pF_SportsDS,
                          &SpoConD,
                          &RK_New,
                          pF_SchoolDS,
                          &I_School_LL,
                          &CntRes,
                          pF_SchoolRes);
   }

   // generate international result
   displaySchoolWinners(FALSE,
                        pF_SchoolRes,
                        &CntRes,
                        &I_School_LL);

   // close input & output files
   fclose(pF_SportsDS);    // op. 5
   fclose(pF_SchoolDS);    // op. 33
   fclose(pF_SchoolRes);   // op. 6

   // op. 127
   printf("============================\n");
   printf("Evaluate Sports DS Statistics\n");
   printf("============================\n");
   printf("Countries                  %.2d\n", CntRes.countries);
   printf("Schools                   %.3d\n",  CntRes.total_schools);
   printf("Sports contest records %.6d\n",     CntRes.sports_recs);
   printf("School records           %.4d\n",   CntRes.school_recs);
   printf("Sports result records    %.4d\n",   CntRes.result_recs);
   printf("============================\n");
   printf("*** Successful completion ***\n");

   return 0;               // op. 7
}
```

Anmerkungen

- Angesichts der Tatsache, dass die `main()` ein recht komplexes Programm regiert, wirkt sie sehr einfach. Nach erstaunlich wenigen Daten-Deklarationen folgt die Parameterprüfung, die wegen der zwei

Eingabedateien etwas umfangreicher als bei SonnetAnalysis ausfällt. Dem Öffnen der Dateien, das bei Fehlschlag sofort zum Abbruch mit einer entsprechenden Fehlermeldung führt, folgt das Vorlesen. Der Sports Dataset darf natürlich nicht leer sein.

- Das Signal für `buildLinkedList()`, die internationale Schulliste zu initialisieren, wird durch die Zuweisung des `contestant_type` ergänzt. Diese internationale Link-Liste ist das Eigentum der `main()`. Da die Liste von den darunterliegenden Modulen aufgebaut wird, muss ihre Adresse dorthin übergeben werden.
- Mit dem ersten gelesenen Satz beginnt die Ranggruppenverarbeitung durch `processCountryGroup()`. Alle `FILE`-Pointer und die Adressen aller in der `main()` deklarierten Strukturen werden übergeben. Die Schleife endet mit dem EOF des Sports Datasets.
- Danach steht fest, welche Schulen im internationalen Vergleich die besten Durchschnittswerte erzielt haben. `displaySchoolWinners()` liest die Ergebnisse aus und vervollständigt damit die School Sports Results.
- Nun sind nur noch die Dateien zu schließen und die Statistik auszugeben.
- Die `main()` und das übrige Programm interessieren sich im Gegensatz zu SonnetAnalysis nicht dafür, ob die eingelesenen Sätze mit CR / LF, oder nur mit LF enden. CR-/LF-Zeichen und schleppende Blanks werden beim Einlesen entfernt beziehungsweise übergangen.

7.5.3 processCountryGroup()

```
/**
 * processCountryGroup() represents the top level of the group processing loops.
 * It is called once for each "new" country by the main().
 * processCountryGroup() calls processRSchoolGroup() repeatedly until
 * all region-school data groups inside the actual country are processed.
 * Then it offers the list of the national winner schools for the international
 * list by calling buildLinkedList() repeatedly.
 * Finally, it outputs the national list if and only if there are more than 3
 * participating schools in the country.
 */
void processCountryGroup( FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          FILE         * pF_SchoolDS,
                          struct L_L   * pS_I_School_LL,
                          struct C_R   * pS_CntRes,
                          FILE         * pF_SchoolRes)
{
   char old_rank_1_key[5];

   int  crt_i     = 0;
   int  entries   = 0;
   int  nxt_i     = 0;

   const
   int  MAX_ENTRY = LL_LH_CEIL;
```

```
   struct L_L N_School_LL = LL_HDR_INIT;

#if TRACE_MODULE
   printf("\n processCountryGroup()\n");
#endif

   // op. 8,13,14
   strcpy(old_rank_1_key, pS_RK_New->rank_1_key);
   pS_CntRes->countries++;
   pS_CntRes->schools = 0;

   // op. 81
   N_School_LL.unique_values   = 0;
   N_School_LL.contestant_type = 'S';

   // I2: process r-school groups
   while (strncmp(old_rank_1_key, pS_RK_New->rank_1_key, 4) == 0)
   {
      processRSchoolGroup(pF_SportsDS,
                          pS_SpoConD,
                          pS_RK_New,
                          pF_SchoolDS,
                          &N_School_LL,
                          pS_CntRes,
                          pF_SchoolRes);
   };

#if TRACE_LL
   displayLinkedList(&N_School_LL);
// printf("\t Exit 100\n");
// exit(100);
#endif

   // op. 119,93
   pS_CntRes->total_schools += pS_CntRes->schools;
   nxt_i = N_School_LL.chain_top_index;

   // I40: copy country result
   while (nxt_i > -1 &&
          entries < MAX_ENTRY)
   {
      // op. 62,98,96
      crt_i = nxt_i;
      entries++;

      buildLinkedList(N_School_LL.LLE[crt_i].entry_value,
                      NULL,
```

```
                        &N_School_LL.LLE[crt_i].School_Data,
                        pS_I_School_LL);

      nxt_i = N_School_LL.LLE[crt_i].downlink;
   };

   // S7: 4 or more participating schools in country?
   if (pS_CntRes->schools >= 4)
   {
      // generate country result
      displaySchoolWinners(TRUE,
                           pF_SchoolRes,
                           pS_CntRes,
                           &N_School_LL);
   };
}
```


Anmerkungen

- Eingangs deklariert `processCountryGroup()` die nationale Schulliste für das aktuelle Land und sichert den konkatenierten Rang-1-Schlüssel. Die Einzel-Strings in der übergebenen `struct RK_New`, aus denen der Rang-1-Schlüssel zusammengesetzt ist, werden nicht kopiert, weil das nicht erforderlich ist.
- Analog zur `main()` setzt `processCountryGroup()` das Initialisierungssignal und den Typ für die Schulliste, bevor in der gruppenwechselgesteuerten Schleife die nächsttiefere Rangstufenverarbeitung aufgerufen wird.
- Nach Verarbeitung aller Regions-Schul-Daten für das Land liegt die nationale Schul-Gewinnerliste vor. Da alle nationalen Gewinner am internationalen Vergleich teilnehmen, liest `processCountryGroup()` die Liste aus und füttert mit deren Inhalten die internationale Schulliste, indem für jeden Eintrag `buildLinkedList()` aufgerufen wird. Nur wenn für ein Land mehr als 4 Schulen teilgenommen haben, wird per `displaySchoolWinners()` die nationale Rangliste der Schulen in die Datei School Sports Results ausgegeben.
- Die zusätzliche Abfrage in der Ausleseschleife, ob `entries < MAX_ENTRY` gilt, verhindert eine unendliche Schleife für den Fall, dass die Link-Struktur beschädigt ist. Sie könnte beispielsweise einen Downlink enthalten, der auf einen bereits durchlaufenen Eintrag rückverweist. `LL_LH_CEIL` ist nicht nur der höchste Index der unteren Link-Listen-Hälfte, sondern auch der höchste für `entries` zulässige Wert.

7.5.4 processRSchoolGroup()

```
/**
 * processRSchoolGroup() processes the data for a given region-school group.
 * The region key is used only for making the school key unique; therefore no
 * extra region processing loop is required. Each region-school group consists
 * of one or two gender groups, i.e., first (or only) the female group,
 * then (or only) the male group.
 * The school data uses the same region-extended school key and is sorted in the
 * same (ascending) order. The program expects complete synchronity of the
 * corresponding extended school keys. Each new group of school data (3 records)
```

```
 * is checked for keeping this promise; otherwise the program ends abnormally.
 * Subsequently each group of school data is written to the output file (line #
 * 01, 02 and 03), repeating the school ID in line 01.
 * Then the gender group loop follows.
 * After processing all sports contest records (pupil data) for the actual school,
 * the statistic values are available, including the cumulated number of points.
 * The module computes the average value for the school and stores the integer &
 * fractional result. Then it builds the statistics record (line # 04) and writes
 * it. Each school can be a national (and perhaps an international) winner, based
 * on its average point value. Therefore its data is offered to buildLinkedList().
 * Finally, the module displays the list of pupil winners, sorted in descending
 * point order, by calling displayPupilWinners(). There, completePupilSpoRes() is
 * used to finalize the winning pupil's records and writing them to the output
 * file (from line # 05 on).
 */
void processRSchoolGroup( FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          FILE         * pF_SchoolDS,
                          struct L_L   * pS_N_School_LL,
                          struct C_R   * pS_CntRes,
                          FILE         * pF_SchoolRes)
{
   char old_rank_1_2_key[15] = {' '};
   char school_address[65]   = {' '};
   char school_ID[14]        = {' '};
   char school_rec[BUFSIZ]   = {' '};
   char school_name[51]      = {' '};
   char school_values[28]    = {' '};
   char result[10]           = {' '};
   char RR01[81]             = {' '};
   char RR04[44]             = {' '};

   int  avg_school_pt_int    = 0;
   int  avg_school_pt_frac   = 0;
   long avg_school_pt_100    = 0;

   struct E_S_K ESK_New      = ESK_INIT;

   struct L_L   Pupil_LL     = LL_HDR_INIT;

   struct SID_D School_Data  = SIDD_INIT;

#if TRACE_MODULE
   printf("\n processRSchoolGroup()\n");
#endif
```

```
// op. 82
Pupil_LL.unique_values   = 0;
Pupil_LL.contestant_type = 'P';

// op. 32: read school record, check for key match
ESK_New = readSchoolDS(pF_SchoolDS, school_rec);

if (strncmp(ESK_New.e_school_key, pS_RK_New->rank_1_2_key, 14) != 0)
{
   printf("\n\t School key and sports contest key do not match! \n");
   printf("\t e_school_key: %s \n", ESK_New.e_school_key);
   printf("\t rank_1_2_key: %s \n", pS_RK_New->rank_1_2_key);
   exit(1);
};

// op. 34,84,85,35
substr(RR01, school_rec, 0, (SCHOOL_LINE_KEY_LEN+1));
substr(school_ID, school_rec, 0, SCHOOL_ID_LEN);
strcat(RR01, school_ID);
strcat(RR01, " ");
substr(school_name, school_rec, SCHOOL_NAME_OFFSET, MAX_NAME_LEN);
strcat(RR01, school_name);
strcat(RR01, "\n");

fputs(RR01, pF_SchoolRes);

// op. 32,36,37
ESK_New = readSchoolDS(pF_SchoolDS, school_rec);

substr(school_address, school_rec, SCHOOL_ADDR_OFFSET, MAX_ADDR_LEN);

fputs(school_rec, pF_SchoolRes);

// op. 32,37 for e-mail address record
ESK_New = readSchoolDS(pF_SchoolDS, school_rec);

fputs(school_rec, pF_SchoolRes);

// op. 123, 126
pS_CntRes->school_recs += 3;
pS_CntRes->result_recs += 3;

// op. 9,15-19,24
strcpy(old_rank_1_2_key, pS_RK_New->rank_1_2_key);

pS_CntRes->schools++;
pS_CntRes->F_age_groups  = 0;
```

```
   pS_CntRes->M_age_groups  = 0;
   pS_CntRes->F_pupils      = 0;
   pS_CntRes->M_pupils      = 0;
   pS_CntRes->school_points = 0;

   // I4: process gender groups
   while (strncmp(old_rank_1_2_key, pS_RK_New->rank_1_2_key, 14) == 0)
   {
      processGenderGroup(pF_SportsDS,
                         pS_SpoConD,
                         pS_RK_New,
                         &Pupil_LL,
                         pS_CntRes);
   };

   // use school values

   // copy school ID (op. 26 replaced)
   strcpy(RR04, school_ID);
   strcat(RR04, "04");

   // compute & display average result
   // op. 27: no data type for decimal value nn.nn available
   // scale the result by 100, then compute integer & fraction parts
   avg_school_pt_100  = (100 * pS_CntRes->school_points) /
                        (pS_CntRes->F_pupils + pS_CntRes->M_pupils);
   avg_school_pt_int  = avg_school_pt_100 / 100;
   avg_school_pt_frac = avg_school_pt_100 % 100;

#if TRACE_DETAIL
   printf("\n\t == school results ==\n");
   printf("\t schools: %d \n",       pS_CntRes->schools);
   printf("\t F_age_groups: %d \n",  pS_CntRes->F_age_groups);
   printf("\t M_age_groups: %d \n",  pS_CntRes->M_age_groups);
   printf("\t F_pupils: %d \n",      pS_CntRes->F_pupils);
   printf("\t M_pupils: %d \n",      pS_CntRes->M_pupils);
   printf("\t school_points: %d \n", pS_CntRes->school_points);
   printf("\t avg_s_p_100: %ld \n",  avg_school_pt_100);
   printf("\t avg_s_p_int: %d \n",   avg_school_pt_int);
   printf("\t avg_s_p_frac: %d \n",  avg_school_pt_frac);
#endif

   // op. 28
   strcat(RR04, " Ø ");
   itoa(avg_school_pt_int, result, 2);
   strcat(RR04, result);
   strcat(RR04, ",");
```

```
   itoa(avg_school_pt_frac, result, 2);
   strcat(RR04, result);

   if (pS_CntRes->F_age_groups > 0)   // S17
   {
      // op. 29
      strcat(RR04, " F ");
      itoa(pS_CntRes->F_age_groups, result, 0);
      strcat(RR04, result);
      strcat(RR04, "/");
      itoa(pS_CntRes->F_pupils, result, 0);
      strcat(RR04, result);
   };

   if (pS_CntRes->M_age_groups > 0)   // S18
   {
      // op. 30
      strcat(RR04, " M ");
      itoa(pS_CntRes->M_age_groups, result, 0);
      strcat(RR04, result);
      strcat(RR04, "/");
      itoa(pS_CntRes->M_pupils, result, 0);
      strcat(RR04, result);
   };

   // op. 86,4
   substr(school_values, RR04, SCHOOL_VALUES_OFFSET, MAX_SCHOOL_VAL_LEN);
   strcat(RR04, "\n");

#if TRACE_DETAIL
   printf("\t RR04= %s\n", RR04);
#endif

   fputs(RR04, pF_SchoolRes);

   // op. 125,87 - 91
   pS_CntRes->result_recs++;

   strcpy(School_Data.school_ID, school_ID);
   strcpy(School_Data.name, school_name);
   strcpy(School_Data.address, school_address);
   strcpy(School_Data.values, school_values);

   buildLinkedList(avg_school_pt_100,
                   NULL,
                   &School_Data,
                   pS_N_School_LL);
```

```
   displayPupilWinners(pF_SchoolRes,
                       pS_CntRes,
                       &Pupil_LL);
}
```

Anmerkungen

Die Verarbeitung der Rangstufe Region-Schule ist der Ort, an dem

- die Schul-Daten für die Ausgabezeilen 01, 02 und 03 eingelesen und sowohl passend ausgegeben als auch zwischengespeichert werden, wobei die Synchronität der erweiterten Schul-Schlüssel geprüft wird (und es bei Differenzen zum Abbruch kommt),
- der Rang-1-2-Schlüssel als `old_rank_1_2_key` gesichert wird,
- die Zählung der weiblichen / männlichen Gruppen beziehungsweise Teilnehmer startet und endet,
- die Verarbeitung der nächsttieferen Rangstufe durch `processGenderGroup()` in einer Schleife angefordert wird, die bei EOF oder beim Wechsel des Landes, der Regions-ID oder der Schulnummer endet,
- die Verdichtung der sportlichen Ergebnisse zum Durchschnittswert für die Schule stattfindet,
- die Ausgabezeile 04 mit den schulischen Ergebnissen erzeugt wird,
- die Schul-Stammdaten – ohne E-Mail-Adresse – und die Schulergebnisse an die nationale Schulliste per `buildLinkedList()` übergeben werden, und
- die Gewinnerliste der Schüler durch `displayPupilWinners()` – und darin `completePupilSpoRes()` – aufgebaut und ausgegeben wird.

Diese Vielzahl an Aufgaben macht den Code ziemlich lang. Eingangs werden entsprechend viele lokale Datenelemente deklariert. `processRSchoolGroup()` ist der Eigentümer der Schüler-Link-Liste, für die es das Initialisierungssignal setzt und den Typ vorgibt.

Der Extended School Key `ESK_New` dient lediglich dazu, festzustellen, ob der Rangstufenschlüssel zum School Dataset-Gruppenschlüssel paarig ist. Andernfalls würde ein Chaos eintreten. Bei unpaarigen Schlüsseln wird daher der Lauf mit einer Fehlermeldung abgebrochen.

Das Modul schneidet mit der programmeigenen `substr()`-Prozedur Teile der Schul-Namens- und -Adressdaten aus den „school records" heraus und montiert diese später mit den Standardroutinen `strcpy()` / `strcat()` zu `RR01` / `RR04` sammen. Da die Schulstammdaten-Sätze keine starre Länge haben, werden störende CR- oder LF-Zeichen am Satzende, sowie schleppende Blanks schon beim Einlesen entfernt.

Die Sprache C besitzt keinen Datentyp mit einer festen Anzahl von – hier: zwei – Nachkommastellen. Daher ermittelt `processRSchoolGroup()` den Schul-Durchschnitt etwas unkonventionell, indem die Gesamtpunktzahl der Schüler zuerst mit 100 multipliziert und dann durch die Anzahl der Schüler geteilt wird. Durchschnitts-Stellen nach dem Hundertstel werden dadurch abgeschnitten. Danach bestimmt das Modul die Vorkommastellen und die Nachkommastellen getrennt. `avg_school_pt_100` wird zusammen mit den Schul-Teilnehmerdaten an `buildLinkedList()` übergeben. Die programmeigene `itoa()`-Prozedur setzt die Vor- und Nachkommapartikel in Ziffern für `RR04` um.

Der C-Compiler der Lightweight-IDE gibt übrigens bei der Zuweisung

```
strcat(RR04, " Ø ");
```

die Warnung „illegal character encoding" aus, aber das Zeichen wird dennoch korrekt verwendet.

7.5.5 processGenderGroup()

```
/**
 * processGenderGroup() is a simple intermediate group loop. It calls
 * processAgeGroup() repeatedly until all age groups have been processed.
 */
void processGenderGroup(  FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          struct L_L   * pS_Pupil_LL,
                          struct C_R   * pS_CntRes)
{
   char old_rank_1_3_key[16];

#if TRACE_MODULE
   printf("\n processGenderGroup()\n");
#endif

   // op. 10
   strcpy(old_rank_1_3_key, pS_RK_New->rank_1_3_key);

   // I5: process age groups
   while (strncmp(old_rank_1_3_key, pS_RK_New->rank_1_3_key, 15) == 0)
   {
      processAgeGroup(pF_SportsDS,
                      pS_SpoConD,
                      pS_RK_New,
                      pS_Pupil_LL,
                      pS_CntRes);
   };
}
```

Anmerkungen

Krasser kann der Unterschied zwischen zwei Ranggruppenverarbeitungen kaum ausfallen, wie der zwischen `processRSchoolGroup()` und `processGenderGroup()`. Der epischen Länge der Rang-1-2-Gruppenverarbeitung folgt die lakonische Kürze der Rang-1-3-Gruppenverarbeitung. Hier wird lediglich der Rang-1-3-Schlüssel gesichert, der zusätzlich `'F'` oder `'M'` am Ende enthält, und dann die unterste Ranggruppenverarbeitung in einer Schleife aufgerufen, die bei Gruppenwechsel oder EOF endet. Es gibt hier auch nichts zu zählen, weil die Zählergebnisse bezüglich der Gruppenanzahlen und -größen in `processAgeGroup()` anfallen.

7.5.6 processAgeGroup()

```
/**
 * processAgeGroup() processes all sports contest records (pupil data)
 * in a given age group. It counts the pupils in the group and sums up the
 * individual points in the three sport activities. Each pupil can be a medal
 * candidate. Therefore processAgeGroup() offers its data to buildLinkedList().
 * Finally, depending on the gender group code, the gender statistics are
 * updated.
 */
void processAgeGroup(     FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD,
                          struct R_K   * pS_RK_New,
                          struct L_L   * pS_Pupil_LL,
                          struct C_R   * pS_CntRes)
{
   char old_rank_1_4_key[18];

   int  pupils    = 0;   // op. 20
   int  point_sum = 0;

#if TRACE_MODULE
   printf("\n processAgeGroup()\n");
#endif

   // op. 11
   strcpy(old_rank_1_4_key, pS_RK_New->rank_1_4_key);

   // I6: process sports records
   while (strncmp(old_rank_1_4_key, pS_RK_New->rank_1_4_key, 17) == 0)
   {
      // op. 121,21,23,25
      pS_CntRes->sports_recs++;
      pupils++;
      point_sum                 = atoi(pS_SpoConD->sprint_points)
                                + atoi(pS_SpoConD->jump_points)
                                + atoi(pS_SpoConD->throw_points);
      pS_CntRes->school_points += point_sum;

      // op. 83: is this pupil a medal candiate?
      buildLinkedList(point_sum,
                      pS_SpoConD,
                      NULL,
                      pS_Pupil_LL);

      // op. 3: read next sports contest record
      * pS_RK_New = readSportsDS(pF_SportsDS, pS_SpoConD);
```

```
   };

   // op. 22
   if (old_rank_1_4_key[14] == 'F')
   {
      pS_CntRes->F_age_groups++;
      pS_CntRes->F_pupils += pupils;
   }
   else
   {
      pS_CntRes->M_age_groups++;
      pS_CntRes->M_pupils += pupils;
   };
}
```


Anmerkungen

Auf der untersten Rangstufe sichert `processAgeGroup()` zunächst seinen konkatenierten Rang-1-4-Schlüssel und tritt dann in die Schleife ein, die bei Gruppenwechsel oder EOF endet. Der erste Satz, den die `main()` eingelesen hat, wird hier auch zuerst verarbeitet. Bei späteren Aufrufen ist zuerst derjenige Satz dran, der den vorhergehenden Gruppenwechsel ausgelöst hat. Das Modul wandelt die Ziffernstrings der drei Sportergebnisse mit `atoi()` in Zahlen um und bestimmt die Punktsumme. Diese wird sowohl zur Schul-Punktsumme addiert als auch zusammen mit der Adresse der Struktur `SpoConD`, welche die übrigen Teilnehmerdaten enthält, an `buildLinkedList()` übergeben. Es folgen das Nachlesen und die Gruppenwechsel-Prüfung.

Am Ende jeder Altergruppe saldiert das Modul die Schüleranzahlen je nach Geschlecht und erhöht die jeweilige Altersgruppen-Anzahl, operiert also auf der Gender-Ebene. Das funktioniert natürlich nur dann, wenn es – wie hier – wenige bekannte Codes gibt, welche die nächsthöhere Rangstufe differenzieren. Andernfalls müssten die Additionen „eine Etage höher" stattfinden (siehe Unterkapitel „Ablösen der Operation 22" des Band-1-A2-Kapitels „OO-Vorbereitungen für EvaluateSportsDS").

7.5.7 readSportsDS()

```
/**
 * readSportsDS() obtains a sports contest record and preprocesses it.
 * The raw data is converted into elements of the structure SpoConD, and all
 * (pre-initialized) rank keys are built. At end-of-file, the read status is set
 * to 'E' in all rank keys.
 * If a result record contains CR / LF characters or trailing blanks, the index
 * "top" is decreased.
 * There is no check of the data quality. The function assumes correct input
 * data and the ascending key order. This is legitimate because all sports contest
 * data must pass a thorough examination before being accepted for the contest.
 * The function returns the rank key structure. The SpoConD(ata) is returned via
 * the pointer reference pS_SpoConD.
 */
struct R_K readSportsDS(  FILE         * pF_SportsDS,
                          struct S_C_D * pS_SpoConD)
```

```
{
   char * ps_SportsDS;
   char   sports_rec[BUFSIZ] = {' '};

   struct R_K RK_Crt         = RK_INIT;

#if TRACE_INPUT
   printf("\n readSportsDS()\n");
#endif

   ps_SportsDS = fgets(sports_rec, BUFSIZ, pF_SportsDS);

   if (ps_SportsDS == NULL || strlen(sports_rec) <= PUPIL_NAME_OFFSET)
      { strcpy(RK_Crt.read_status, "E"); }
   else
   {
#if TRACE_INPUT
      printf("\t sports_rec = %s", sports_rec);
#endif

      int top = strlen(sports_rec);

      for (int i = top - 1; i > (PUPIL_NAME_OFFSET+1); i--)
      {
         if ((sports_rec[i] == ' ')   ||
             (sports_rec[i] == '\r')  ||
             (sports_rec[i] == '\n'))
            { top--; }
         else
            { break; }
      }

      substr(pS_SpoConD->country_code,  sports_rec, P_COUNTRY_OFFSET,
                                                    COUNTRY_LEN);
      substr(pS_SpoConD->region_id,     sports_rec, P_REGION_ID_OFFSET,
                                                    REGION_ID_LEN);
      substr(pS_SpoConD->school_number, sports_rec, P_SCHOOL_NO_OFFSET,
                                                    SCHOOL_NO_LEN);
      substr(pS_SpoConD->gender_code,   sports_rec, GENDER_CODE_OFFSET,
                                                    GENDER_CODE_LEN);
      substr(pS_SpoConD->age,           sports_rec, AGE_OFFSET, AGE_LEN);
      substr(pS_SpoConD->sprint_points, sports_rec, SPRINT_POINTS_OFFSET,
                                                    SPRINT_POINTS_LEN);
      substr(pS_SpoConD->jump_points,   sports_rec, JUMP_POINTS_OFFSET,
                                                    JUMP_POINTS_LEN);
      substr(pS_SpoConD->throw_points,  sports_rec, THROW_POINTS_OFFSET,
                                                    THROW_POINTS_LEN);
```

```
      substr(pS_SpoConD->name,          sports_rec, PUPIL_NAME_OFFSET, top);

      strcpy(RK_Crt.read_status, "D");
      strcpy(RK_Crt.country_code,  pS_SpoConD->country_code);
      strcpy(RK_Crt.region_id,     pS_SpoConD->region_id);
      strcpy(RK_Crt.school_number, pS_SpoConD->school_number);
      strcpy(RK_Crt.gender_code,   pS_SpoConD->gender_code);
      strcpy(RK_Crt.age,           pS_SpoConD->age);

   };

   strcpy(RK_Crt.rank_1_key,   RK_Crt.read_status);
   strcat(RK_Crt.rank_1_key,   RK_Crt.country_code);

   strcpy(RK_Crt.rank_1_2_key, RK_Crt.rank_1_key);
   strcat(RK_Crt.rank_1_2_key, RK_Crt.region_id);
   strcat(RK_Crt.rank_1_2_key, RK_Crt.school_number);

   strcpy(RK_Crt.rank_1_3_key, RK_Crt.rank_1_2_key);
   strcat(RK_Crt.rank_1_3_key, RK_Crt.gender_code);

   strcpy(RK_Crt.rank_1_4_key, RK_Crt.rank_1_3_key);
   strcat(RK_Crt.rank_1_4_key, RK_Crt.age);

#if TRACE_INPUT
   printf("\n\t Sports Contest record\n");
   printf("\t country_code: %s \n",  pS_SpoConD->country_code);
   printf("\t region_id: %s \n",     pS_SpoConD->region_id);
   printf("\t school_number: %s \n", pS_SpoConD->school_number);
   printf("\t gender_code: %s \n",   pS_SpoConD->gender_code);
   printf("\t age: %s \n",           pS_SpoConD->age);
   printf("\t name: %s \n",          pS_SpoConD->name);
   printf("\t sprint_points: %s \n", pS_SpoConD->sprint_points);
   printf("\t jump_points: %s \n",   pS_SpoConD->jump_points);
   printf("\t throw_points: %s \n",  pS_SpoConD->throw_points);
   printf("\t rank_1_key: %s \n",    RK_Crt.rank_1_key);
   printf("\t rank_1_2_key: %s \n",  RK_Crt.rank_1_2_key);
   printf("\t rank_1_3_key: %s \n",  RK_Crt.rank_1_3_key);
   printf("\t rank_1_4_key: %s \n",  RK_Crt.rank_1_4_key);
#endif

   return RK_Crt;
}
```

Anmerkungen

Die Leseroutine für die „sports contest records“ ist gegenüber dem vergleichbaren Modul `readTextin()` im Band-2-A1-Kapitel „Sonnet_Analysis in C“ wesentlich aufwendiger. Aus den eingelesenen Sätzen werden etliche Strings extrahiert und zunächst in die Struktur `SpoConD` übertragen. Daraus kopiert das Modul die Einzel-Strings für den `struct RK_Crt` und erzeugt dann die aufeinander aufbauenden Rangschlüssel. Die Satz-Nutzdaten werden als Parameter und `RK_Crt` als Funktionsergebnis rückgemeldet.

`readSportsDS()` geht davon aus, dass sämtliche erwarteten Satzbestandteile vollständig zur Verfügung stehen, da nicht geprüft wird, ob die Satzlänge größer als 28 ist, also auch ein Namensstring vorliegt. CR-/LF-Zeichen und schleppende Blanks werden übergangen. `substr()` setzt auf dem angegebenen Offset auf und schneidet maximal so viele Zeichen aus, wie angegeben. Bei einem Satz, der zu kurz ist, würde `substr()` daher *nach* dessen Terminierungs-Null Speicherinhalte auswerten, die vom letzten vollständigen Satz stammen. Da das Programm von geprüften Eingaben ausgehen darf, ist diese Gefahr nicht relevant.

7.5.8 readSchoolDS()

```
/**
 * readSchoolDS() obtains one school data record and extracts its key elements
 * (i.e., country code, region ID, school #) plus the line #.
 * It builds the (pre-initialized) rank key structure ESK_Crt.
 * If a school record contains CR / LF characters or trailing blanks, it is
 * truncated.
 * At end-of-file, the read status is set to 'E' in ESK_Crt.
 * There is no check of the data quality. The module assumes correct input
 * data and the ascending key & line # order. This is legitimate because all
 * school data must pass a thorough examination before being accepted for the
 * contest.
 * The function returns ESK_Crt. The school record is returned via the call
 * interface as a character array.
 */
struct E_S_K readSchoolDS(FILE         * pF_SchoolDS,
                          char           school_rec[])
{
   char * ps_SchoolDS;

   struct E_S_K ESK_Crt = ESK_INIT;

#if TRACE_INPUT
   printf("\n readSchoolDS()\n");
#endif

   ps_SchoolDS = fgets(school_rec, BUFSIZ, pF_SchoolDS);

   if (ps_SchoolDS == NULL)
      { strcpy(ESK_Crt.read_status, "E"); }
   else
   {
```

```
#if TRACE_INPUT
   printf("\t school_rec = %s", school_rec);
#endif

      int top = strlen(school_rec);

      for (int i = top - 1; i > (SCHOOL_NAME_OFFSET+1); i--)
      {
         if ((school_rec[i] == ' ')   ||
             (school_rec[i] == '\r')  ||
             (school_rec[i] == '\n'))
            { school_rec[i] = '\0'; }
         else
            { break; }
      }

      strcpy(ESK_Crt.read_status,   "D");
      substr(ESK_Crt.country_code,  school_rec, S_COUNTRY_OFFSET,
                                                COUNTRY_LEN);
      substr(ESK_Crt.region_id,     school_rec, S_REGION_ID_OFFSET,
                                                REGION_ID_LEN);
      substr(ESK_Crt.school_number, school_rec, S_SCHOOL_NO_OFFSET,
                                                SCHOOL_NO_LEN);
      substr(ESK_Crt.line_no,       school_rec, LINE_NO_OFFSET,
                                                LINE_NO_LEN);
   };

   strcpy(ESK_Crt.e_school_key, ESK_Crt.read_status);
   strcat(ESK_Crt.e_school_key, ESK_Crt.country_code);
   strcat(ESK_Crt.e_school_key, ESK_Crt.region_id);
   strcat(ESK_Crt.e_school_key, ESK_Crt.school_number);

#if TRACE_INPUT
   printf("\n\t School record\n");
   printf("\t country_code : %s \n", ESK_Crt.country_code);
   printf("\t region_id    : %s \n", ESK_Crt.region_id);
   printf("\t school_number: %s \n", ESK_Crt.school_number);
   printf("\t line_no      : %s \n", ESK_Crt.line_no);
   printf("\t e_school_key : %s \n", ESK_Crt.e_school_key);
#endif

   return ESK_Crt;
}
```

Anmerkungen

`readSchoolDS()` funktioniert ähnlich wie `readSportsDS()`. Da die Satzinhalte durch `processRSchoolGroup()` ausgewertet werden, baut `readSchoolDS()` lediglich den erweiterten Schul-Schlüssel, den `struct ESK_Crt`, auf. Die Schul-Stammdaten werden als Parameter und der Schlüssel als Funktionsergebnis rückgemeldet.

CR-/LF-Zeichen und schleppende Blanks werden durch je eine Terminierungs-Null ersetzt. Auch hier gibt es keine Prüfung, ob jeder Satz lang genug ist, denn das wird vorausgesetzt.

7.5.9 buildLinkedList()

```
/**
 * buildLinkedList() represents the top of the structure block tree "build
 * linked list". It determines the current status of the linked list and
 * decides what to do with the offered new data.
 */
void buildLinkedList(     int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList)
{
#if TRACE_LL
   printf("\n buildLinkedList(), new_value = %d\n", new_value);
#endif

   // S32 (part 1): linked list empty?
   if (pS_LinkList->unique_values == 0)
   {
      initializeLinkedList(new_value,
                           pS_Pupil_Data,
                           pS_School_Data,
                           pS_LinkList);
      return;
   }

   // at least one linked list entry exists
#if TRACE_LL
   printf("\t last entry_value = %d\n",
          pS_LinkList->LLE[pS_LinkList->chain_end_index].entry_value );
#endif

   // S32 (part 2): is this a new value below the current bronze level?
   if (pS_LinkList->unique_values >= MAX_UNIQ_VAL &&
       new_value < pS_LinkList->LLE[pS_LinkList->chain_end_index].entry_value)
   {
      // skip new entry
      return;
```

```
   };

#if TRACE_LL
   printf("\t 1st entry_value = %d\n",
          pS_LinkList->LLE[pS_LinkList->chain_top_index].entry_value);
#endif

   // S32 (part 3): is this a new value at or above the current gold level?
   if (new_value >= pS_LinkList->LLE[pS_LinkList->chain_top_index].entry_value)
   {
      addPrefix_LL_Entry(new_value,
                         pS_Pupil_Data,
                         pS_School_Data,
                         pS_LinkList);
   }
   else
   {
      addOther_LL_Entry( new_value,
                         pS_Pupil_Data,
                         pS_School_Data,
                         pS_LinkList);
   };

#if TRACE_LL
   displayLinkedList(pS_LinkList);
// printf("\t Exit 130\n");
// exit(110);
#endif
}
```

Anmerkungen

`buildLinkedList()` ist in dieser Implementierung nur noch ein Verteiler, der die Kontrolle S32 vornimmt, um zu entscheiden, ob

- die betreffende Link-Liste mit dem ersten Eintrag initialisiert werden soll, dessen `new_value` und Teilnehmerdaten soeben übergeben wurden, oder
- schon mehr als drei verschiedene Punktsummen oder Durchschnittswerte vorliegen und `new_value` kleiner als der aktuelle Bronze-Wert ist ⇨ kein Medaillen-Kandidat, verwerfen
- `new_value` den aktuellen Gold-Wert erreicht oder übertrifft ⇨ voranstellen, eventuell die Eintragskette danach kürzen
- `new_value` unter Gold und größer oder gleich Bronze ist ⇨ einketten und die Eintragskette danach kürzen, falls nötig.

Wenn es darum geht, eine Schülerliste aufzubauen, enthält `new_value` die jeweilige Punktsumme. Der `struct`-Pointer `pS_Pupil_Data` zeigt dann auf die `SpoConD`-Struktur mit den Schüler-Daten. Der zweite Teilnehmerdaten-Pointer enthält `NULL`.

Im Fall der (inter)nationalen Schullisten ist `new_value` = Schul-Durchschnitt * 100. Die Teilnehmerdaten werden durch den `struct`-Pointer `pS_School_Data` adressiert. Dann ist der erste Teilnehmerdaten-Pointer gleich `NULL`.

`buildLinkedList()` reicht `new_value` und die übergebenen Adressen an seine Unterprogramme weiter.

7.5.10 initializeLinkedList()

```
/**
 * initializeLinkedList() builds the first element of a new linked list
 * and initializes all other linked list entries.
 */
void initializeLinkedList(int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList)
{
   const
   int  LL_LHTI = LL_LH_CEIL;    // lower half top index

   int  new_i   = 0;             // op. 55

   struct S_C_D Pupil_Init   = SCD_INIT;
   struct SID_D School_Init  = SIDD_INIT;

#if TRACE_LL
   printf("\n initializeLinkedList()\n");
#endif

   // op. 41,43,46,48,49,55(see init),56,57,58
   pS_LinkList->chain_top_index        = 0;
   pS_LinkList->chain_end_index        = 0;
   pS_LinkList->unique_values          = 1;
   pS_LinkList->list_level             = 0;
   pS_LinkList->max_list_level         = LL_LHTI;
   pS_LinkList->LLE[new_i].entry_value = new_value;
   pS_LinkList->LLE[new_i].uplink      = -1;
   pS_LinkList->LLE[new_i].downlink    = -1;

   // op. 59
   if (pS_LinkList->contestant_type == 'P')
   {
      pS_LinkList->LLE[new_i].Pupil_Data  = * pS_Pupil_Data;
      pS_LinkList->LLE[new_i].School_Data = School_Init;
   }
   else
   {
      pS_LinkList->LLE[new_i].School_Data = * pS_School_Data;
```

```
      pS_LinkList->LLE[new_i].Pupil_Data  = Pupil_Init;
   };

   // initialize all other LL entries
   for (int i = 1; i < LL_MAX; i++)
   {
   pS_LinkList->LLE[i].entry_value = 0;
   pS_LinkList->LLE[i].uplink      = 0;
   pS_LinkList->LLE[i].downlink    = 0;
   pS_LinkList->LLE[i].Pupil_Data  = Pupil_Init;
   pS_LinkList->LLE[i].School_Data = School_Init;
   };

#if TRACE_LL
   displayLinkedList(pS_LinkList);
#endif
}
```


Anmerkungen

`initializeLinkedList()` hat eine sehr einfache Aufgabe. Zusätzlich zum Anlegen des ersten Eintrags, bei dem die jeweils komplementären Teilnehmerdaten initialisiert werden, belegt dieses Modul den Rest der Link-Liste mit Nullen beziehungsweise mit Struktur-Init-Werten vor. Verbliebene Inhalte früherer Rankings werden dadurch unschädlich gemacht, und die beiden add-Module brauchen keine weiteren Initialisierungen mehr vorzunehmen.

7.5.11 addPrefix_LL_Entry()

```
/**
 * addPrefix_LL_Entry() places a new linked list entry in front of all existing
 * entries. First, it checks whether there is sufficient space available
 * for the new entry. If the list is full (although only a few entries are
 * relevant), the list is compressed by moving these entries into the other half
 * of the list space, using compressLinkedList().
 * Then the new entry is connected to the existing chain. If the value (points
 * or point sum) in this entry is higher than the previous "gold" value, the
 * former "bronze" entry / entries (if any) must be cut off the chain, using
 * adjustChainEndIndex().
 */
void addPrefix_LL_Entry(  int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList)
{
   int  crt_i = 0;
   int  new_i = 0;
```

```
#if TRACE_LL
   printf("\n addPrefix_LL_Entry(): new_value = %d\n", new_value);
#endif

   // op. 50
   if (pS_LinkList->list_level < pS_LinkList->max_list_level)
      { pS_LinkList->list_level++; }
   else
      { compressLinkedList(pS_LinkList); };

   // op. 55,51,56,57,60
   new_i = pS_LinkList->list_level;
   crt_i = pS_LinkList->chain_top_index;

   pS_LinkList->LLE[new_i].entry_value = new_value;
   pS_LinkList->LLE[new_i].uplink      = -1;
   pS_LinkList->LLE[new_i].downlink    = crt_i;

   // op. 59
   if (pS_LinkList->contestant_type == 'P')
      { pS_LinkList->LLE[new_i].Pupil_Data  = * pS_Pupil_Data; }
   else
      { pS_LinkList->LLE[new_i].School_Data = * pS_School_Data; };

   pS_LinkList->chain_top_index        = new_i;   // op. 42

   // op. 61 : Amend old 1st value entry
   pS_LinkList->LLE[crt_i].uplink      = new_i;

   // S33: is this a new gold level entry?
   if (new_value > pS_LinkList->LLE[crt_i].entry_value)
   {
      // yes: op. 47
      pS_LinkList->unique_values++;

      // S37: optional maximum adjustment?
      if (pS_LinkList->unique_values > MAX_UNIQ_VAL)
         { adjustChainEndIndex(pS_LinkList); };
   };
}
```

Anmerkungen

`addPrefix_LL_Entry()` prüft eingangs, ob in der aktuellen Link-Listen-Hälfte überhaupt noch Platz für einen neuen Eintrag ist. Falls ja, wird dieser zur Belegung vorgesehen, falls nein, muss durch Umspeichern der relevanten Einträge in die andere Hälfte erst Platz geschaffen werden.

Der neue Eintrag wird aufgebaut und mit dem vorherigen Top-Eintrag verknüpft. Ist der `new_value` größer als der Wert im bisher führenden Gold-Eintrag, so wird die Zahl der unterschiedlichen Werte erhöht. Übersteigt diese das durch

```
#define MAX_UNIQ_VAL  03
```

festgelegte Limit, so liegen vier verschiedene Punktsummen respektive Durchschnitte vor. Der vierte Platz bekommt keine Medaille mehr. Durch Aufruf von `adjustChainEndIndex()` trennt das Modul dann die bisherigen Bronze-Link-Listen-Einträge von der Kette ab. Gewöhnlich ist das ein einziger Eintrag, aber es wurden testweise auch schon zahlreiche punktgleiche Einträge abgehängt.

Der `MAX_UNIQ_VAL`-Wert darf nicht größer als die Anzahl der Medaillensymbole in der Liste G_S_B sein, weil es sonst bei der Auswertung der Link-Listen und der Zuweisung der G_S_B-Zeichen zu einem Indexüberlauf kommt.

7.5.12 addOther_LL_Entry()

```
/**
 * addOther_LL_Entry() adds a linked list entry on the 2nd or 3rd rank.
 * First, it checks whether there is sufficient space available for the new
 * entry. If the list is full (although only a few entries are relevant), the
 * list is compressed by moving these entries into the other half of the list
 * space, using compressLinkedList().
 * The place for the new entry is searched top-down. It may be a new  "silver"
 * or "bronze" entry, or it is a duplicate entry (with the same value as in an
 * existing entry). The new entry is placed always in front of the old entry
 * with the same or lower value; otherwise it is put at the end of the chain.
 * If there are more than three different values in the chain, the former
 * "bronze" entry / entries must be cut off, using adjustChainEndIndex().
 */
void addOther_LL_Entry(   int            new_value,
                          struct S_C_D * pS_Pupil_Data,
                          struct SID_D * pS_School_Data,
                          struct L_L   * pS_LinkList)
{
   int  crt_i = 0;
   int  new_i = 0;
   int  nxt_i = 0;

#if TRACE_LL
   printf("\n addOther_LL_Entry(): new_value = %d\n", new_value);
#endif
```

```
   // op. 50
   if (pS_LinkList->list_level < pS_LinkList->max_list_level)
      { pS_LinkList->list_level++; }
   else
      { compressLinkedList(pS_LinkList); };

   // op. 51,53
   crt_i = pS_LinkList->chain_top_index;
   nxt_i = pS_LinkList->LLE[crt_i].downlink;

   // I34: search middle value entries
   while (nxt_i > -1 &&
          new_value < pS_LinkList->LLE[nxt_i].entry_value)
   {
      // op. 62,53
      crt_i = nxt_i;
      nxt_i = pS_LinkList->LLE[crt_i].downlink;
   };

#if TRACE_LL
   printf("\t crt_i = %d, nxt_i = %d\n", crt_i, nxt_i);
#endif

   // add / amend middle value or last value entries

   // op. 55,63
   new_i = pS_LinkList->list_level;
   pS_LinkList->LLE[crt_i].downlink = new_i;

   // S35: setup middle value entries?
   if (nxt_i > -1)
   {
      // op. 56,64,65
      pS_LinkList->LLE[new_i].entry_value = new_value;
      pS_LinkList->LLE[new_i].uplink      = crt_i;
      pS_LinkList->LLE[new_i].downlink    = nxt_i;

      // op. 59
      if (pS_LinkList->contestant_type == 'P')
         { pS_LinkList->LLE[new_i].Pupil_Data  = * pS_Pupil_Data; }
      else
         { pS_LinkList->LLE[new_i].School_Data = * pS_School_Data; };

      // op. 66: amend old middle value entry
      pS_LinkList->LLE[nxt_i].uplink      = new_i;

      // S36: is this a new silver / bronze level entry?
```

```
      if (new_value > pS_LinkList->LLE[nxt_i].entry_value)
      {
         // yes: op. 47
         pS_LinkList->unique_values++;

         // S37: optional maximum adjustment?
         if (pS_LinkList->unique_values > MAX_UNIQ_VAL)
            { adjustChainEndIndex(pS_LinkList); };
      };
   }
   else
   {
      // add new last value entry
      // op. 47,56,64,58
      pS_LinkList->unique_values++;
      pS_LinkList->LLE[new_i].entry_value = new_value;
      pS_LinkList->LLE[new_i].uplink      = crt_i;
      pS_LinkList->LLE[new_i].downlink    = -1;

      // op. 59
      if (pS_LinkList->contestant_type == 'P')
         { pS_LinkList->LLE[new_i].Pupil_Data  = * pS_Pupil_Data; }
      else
         { pS_LinkList->LLE[new_i].School_Data = * pS_School_Data; };

      pS_LinkList->chain_end_index = new_i; // op. 44
   };
}
```


Anmerkungen

`addOther_LL_Entry()` funktioniert ähnlich wie `addPrefix_LL_Entry()`, muss aber nach der Platzreservierung erst nach demjenigen Eintrag in der Link-Liste suchen, dem der neue Eintrag vor- oder nachgestellt werden soll:

- Zum Voranstellen wird die bisherige Kette an der Einfügestelle aufgetrennt und es werden neue Uplink-/Downlink-Verknüpfungen vorgenommen. Liegt eine neue Punktsumme respektive ein neuer Durchschnitt vor und sind schon mindestens drei verschiedene Werte registriert worden, müssen die bisherigen Bronze-Einträge mittels `adjustChainEndIndex()` abgehängt werden.
- Nur der zweite oder der dritte eindeutige `new_value` unterhalb „Gold" oder „Silber" führt zum Anhängen eines „last value entry". Das prüft bereits `buildLinkedList()`. Trifft die Suche in der Link-Liste auf den Downlink -1 und ist der `new_value` auch dann noch zu niedrig für das Einketten, so fügt `addOther_LL_Entry()` den neuen Eintrag ohne weitere Prüfung an die Kette an.

7.5.13 adjustChainEndIndex()

```
/**
 * adjustChainEndIndex() cuts the former "bronze" entries off the linked list by
 * a) searching the chain backwards until an entry is reached which contains
 *    another value than the chain end's entry value,
 * b) setting the chain end index to the found entry's index, and
 * c) setting the downlink index in this entry to -1 (i.e. end of chain).
 */
void adjustChainEndIndex( struct L_L   * pS_LinkList)
{
#if TRACE_LL
   printf("\n adjustChainEndIndex()\n");
#endif

   // op. 52,54
   int crt_i = pS_LinkList->chain_end_index;
   int nxt_i = pS_LinkList->LLE[crt_i].uplink;

   // I38: skip low entries
   while (pS_LinkList->LLE[nxt_i].entry_value ==
          pS_LinkList->LLE[crt_i].entry_value)
   {
      // op. 62,54
      crt_i = nxt_i;
      nxt_i = pS_LinkList->LLE[crt_i].uplink;
   };

   // op. 45,67,77
   pS_LinkList->chain_end_index     = nxt_i;
   pS_LinkList->LLE[nxt_i].downlink = -1;
   pS_LinkList->unique_values       = MAX_UNIQ_VAL;
}
```

Anmerkungen

`adjustChainEndIndex()` verfolgt ausgehend vom letzten Bronze-Eintrag die Kette aufwärts, bis ein Eintrag mit einem anderen – höheren – `entry_value` erreicht ist. Dieser wird zum letzten Bronze-Eintrag gemacht und der `chain_end_index` darauf gerichtet.

7.5.14 compressLinkedList()

```
/**
 * compressLinkedList() performs the garbage collection each time when one half
 * of the list space is full. The index of the first unused entry is equal
 * to the start index of the opposite (empty) half space. This is the new chain
 * top index.
 * compressLinkedList() positions on the actual chain top entry and copies it to
 * the entry at the new chain top index. The uplink index is set to -1 (i.e.
 * begin of chain) and the new downlink index points to the following new entry.
 * Then all other active entries in the chain are copied accordingly, setting
 * the uplink / downlink indexes to the previous / next entry.
 * The copy loop ends when all "active" entries are copied or when too many
 * entries are about to be copied. The second case is possible if
 * a) there are too many entries with duplicate values, or
 * b) the linked list contains an "index loop" along the downlink chain.
 * The program ends abnormally in the second case.
 * In the first (normal) case, the list level and the chain end index are stored,
 * and the downlink in the last entry is explicitly set to -1 (because it
 * contains the index of the next entry so far).
 * Finally the module resets the lower or the upper half of the list space which
 * was about to overflow before the call to compressLinkedList().
 */
void compressLinkedList(  struct L_L   * pS_LinkList)
{
   const
   int  LL_LHTI = LL_LH_CEIL;    // lower half top index
   const
   int  LL_UHBI = LL_LHTI + 1;   // upper half bottom index

   int  crt_i   = 0;
   int  entries = 1;             // op. 78
   int  i       = 0;
   int  max     = 0;
   int  new_i   = 0;
   int  nxt_i   = 0;

   struct S_C_D Pupil_Init  = SCD_INIT;
   struct SID_D School_Init = SIDD_INIT;

#if TRACE_LL
   printf("\n compressLinkedList()\n");
   displayLinkedList(pS_LinkList);
#endif

   // op. 68,78
   if (pS_LinkList->chain_top_index <= LL_LHTI)
```

```
   { new_i = LL_UHBI; }
else
   { new_i = 0; };

// copy 1st value entry
// op. 51,42,70,57,71
crt_i = pS_LinkList->chain_top_index;

pS_LinkList->chain_top_index        = new_i;
pS_LinkList->LLE[new_i].entry_value =
pS_LinkList->LLE[crt_i].entry_value;
pS_LinkList->LLE[new_i].uplink      = -1;
pS_LinkList->LLE[new_i].downlink    = new_i + 1;

// op. 72
if (pS_LinkList->contestant_type == 'P')
{
   pS_LinkList->LLE[new_i].Pupil_Data  =
   pS_LinkList->LLE[crt_i].Pupil_Data;
}
else
{
   pS_LinkList->LLE[new_i].School_Data =
   pS_LinkList->LLE[crt_i].School_Data;
};

nxt_i = pS_LinkList->LLE[crt_i].downlink;   // op. 53

// I39: copy all other entries until chain end is reached
// or storage overflow is imminent
while (nxt_i > -1 &&
       entries < LL_LHTI)
{
   // op. 62,75,79,70,73,71
   crt_i = nxt_i;
   new_i++;
   entries++;

   pS_LinkList->LLE[new_i].entry_value =
   pS_LinkList->LLE[crt_i].entry_value;
   pS_LinkList->LLE[new_i].uplink      = new_i - 1;
   pS_LinkList->LLE[new_i].downlink    = new_i + 1;

   // op. 72
   if (pS_LinkList->contestant_type == 'P')
   {
      pS_LinkList->LLE[new_i].Pupil_Data    =
```

```
            pS_LinkList->LLE[crt_i].Pupil_Data;
         }
         else
         {
            pS_LinkList->LLE[new_i].School_Data =
            pS_LinkList->LLE[crt_i].School_Data;
         };

         nxt_i = pS_LinkList->LLE[crt_i].downlink;   // op. 53
      };

#if TRACE_LL
   displayLinkedList(pS_LinkList);
#endif

   if (entries >= LL_LHTI)
   {
      printf("\n ++++ Compress: Storage overflow imminent!\n");
      exit(2);
   };

   // op. 76,44,58
   pS_LinkList->list_level             = new_i + 1;
   pS_LinkList->chain_end_index        = new_i;
   pS_LinkList->LLE[new_i].downlink    = -1;

   // op. 74: reset full half of the linked list array
   max = pS_LinkList->max_list_level;

   if (new_i > LL_UHBI)
   {
      // reset lower half
      i                           = 0;
      pS_LinkList->max_list_level = LL_MAX;
   }
   else
   {
      // reset upper half
      i                           = LL_UHBI;
      pS_LinkList->max_list_level = LL_LHTI;
   }

   for (; i < max; i++)
   {
      pS_LinkList->LLE[i].entry_value = 0;
      pS_LinkList->LLE[i].uplink      = 0;
      pS_LinkList->LLE[i].downlink    = 0;
```

```
      pS_LinkList->LLE[i].Pupil_Data  = Pupil_Init;
      pS_LinkList->LLE[i].School_Data = School_Init;
   };

#if TRACE_LL
   printf("\t compress result\n");
   displayLinkedList(pS_LinkList);
#endif

   return;
}
```

Anmerkungen

`compressLinkedList()` ist deutlich umfangreicher, als sein Struktogramm erwarten lässt. Das liegt nicht nur an den Teilnehmerdaten, sondern auch an der zusätzlichen Re-Initialisierung der freigewordenen Link-Listen-Hälfte.

Der Zähler `entries` dient auch hier wieder dazu, im Fall einer beschädigten Link-List-Struktur eine unendliche Schleife zu verhindern. Ausserdem ermöglicht er es, einen drohenden Link-Listen-Überlauf wegen zu vieler Duplicates – Punktsummen / Durchschnitte – zu erkennen und den Lauf dann mit einer Fehlermeldung zu beenden. Beim Test mit zahlreichen absichtlich gleichen Punktsummen konnte diese Situation zuverlässig erkannt und ein unkontrollierter Absturz des Programms verhindert werden.

Es gilt `LL_LHTI = LL_LH_CEIL` analog zu `MAX_ENTRY = LL_LH_CEIL` in `processCountryGroup()`. Daher sind die `entries`-Abfragen äquivalent.

7.5.15 displayPupilWinners()

```
/**
 * displayPupilWinners() generates - together with completePupilSpoRes() -
 * the records for the winning pupils of a school.
 * The first "gold" winner gets the "G" flag in the output. If there are more
 * "gold" winners, their records are added without flagging them. The same
 * happens to the "silver" and "bronze" winners.
 * Therefore the module consists of an outer loop for the first medal winners in
 * a group of duplicates. The latter are processed in an inner loop (which is
 * seldom used because duplicates are rare).
 * If the linked list contains an "index loop" along the downlink chain, the
 * module does not terminate the program abnormally. It generates simply too
 * many winner records.
 */
void displayPupilWinners( FILE         * pF_SchoolRes,
                          struct C_R   * pS_CntRes,
                          struct L_L   * pS_LinkList)
{
   char * psa_G_S_B[3]  = {"G", "S", "B"};
   char   result[10]    = {' '};
   char   school_ID[14] = {' '};
```

```
   char   SpoRes[81]    = {' '};

   int    crt_i         = 0;
   int    entries       = 0;
   int    line_no       = 4;                // op. 100
   int    medal_level   = -1;               // op. 102
   int    nxt_i         = 0;

   const
   int    MAX_ENTRY     = LL_LH_CEIL;

#if TRACE_MODULE
   printf("\n displayPupilWinners()\n");
#endif

   nxt_i = pS_LinkList->chain_top_index;    // op. 92

   // op. 105
   strcpy(school_ID, pS_LinkList->LLE[nxt_i].Pupil_Data.country_code);
   strcat(school_ID, pS_LinkList->LLE[nxt_i].Pupil_Data.region_id);
   strcat(school_ID, pS_LinkList->LLE[nxt_i].Pupil_Data.school_number);

   // I40: display pupil winners
   while (nxt_i > -1 &&
          entries < MAX_ENTRY)
   {
      // display 1st medal winner
      // op. 101,103,62,106,107
      line_no++;
      medal_level++;
      crt_i = nxt_i;
      entries++;

      strcpy(SpoRes, school_ID);
      itoa(line_no, result, 2);
      strcat(SpoRes, result);
      strcat(SpoRes, psa_G_S_B[medal_level]);

      completePupilSpoRes(pF_SchoolRes,
                          SpoRes,
                          &pS_LinkList->LLE[crt_i].Pupil_Data,   // op. 104
                          pS_LinkList->LLE[crt_i].entry_value);

      // op. 125,95
      pS_CntRes->result_recs++;
      nxt_i = pS_LinkList->LLE[crt_i].downlink;
```

```
      // I41: display optional other medal winners
      while (nxt_i > -1                              &&
             pS_LinkList->LLE[nxt_i].entry_value ==
             pS_LinkList->LLE[crt_i].entry_value     &&
             entries < MAX_ENTRY)
      {
         // display winner of same medal
         // op. 101,62,106,108
         line_no++;
         crt_i = nxt_i;
         entries++;

         strcpy(SpoRes, school_ID);
         itoa(line_no, result, 2);
         strcat(SpoRes, result);
         strcat(SpoRes, " ");

         completePupilSpoRes(pF_SchoolRes,
                             SpoRes,
                             &pS_LinkList->LLE[crt_i].Pupil_Data,   // op. 104
                             pS_LinkList->LLE[crt_i].entry_value);

         // op. 125,95
         pS_CntRes->result_recs++;
         nxt_i = pS_LinkList->LLE[crt_i].downlink;
      };
   };
}
```

Anmerkungen

`displayPupilWinners()` hat die Aufgabe, die Liste der Medaillengewinner einer Schule aufzubauen und in die Datei School Sports Results auszugeben. Dabei sind eventuelle Punktsummen-Duplikate zu beachten, wie im Unterkapitel „Schreiben und Lesen der Link-Listen" des Band-1-A2-Kapitels „Ranggruppenverarbeitung II" erläutert. Dies ist die Ursache der geschachtelten Schleifenstruktur, die sich aus Sicherheitsgründen ebenfalls zusätzlich der Abfrage `entries < MAX_ENTRY` bedient.

Die Liste G_S_B ist als Array aus drei Strings – zu je einem Zeichen – aufgebaut, damit der Buchstabe, der die jeweilige Medaille symbolisiert, mittels `strcat` an den entstehenden Ausgabestring angehängt werden kann. Hierfür ist ein String-Pointer-Array (`psa_G_S_B[3]`) deklariert.

Die allen Gewinner-Zeilen gemeinsamen Operationen sind in das Modul `completePupilSpoRes()` ausgelagert, dem `displayPupilWinners()` den Ausgabe-Datei-Pointer, den angefangenen Ausgabesatz, die Daten zum jeweiligen Schüler und dessen Punktsumme übergibt. Dieses – nachfolgend wiedergegebene – triviale Modul wird nicht kommentiert.

7.5.16 completePupilSpoRes()

```
/**
 * completePupilSpoRes() adds the detail data to the output record for
 * a winning pupil, and writes the record.
 */
void completePupilSpoRes( FILE         * pF_SchoolRes,
                          char           SpoRes[],
                          struct S_C_D * pS_Pupil_Data,
                          int            point_sum)
{
   char result[10]    = {' '};

#if TRACE_MODULE
   printf("\n completePupilSpoRes()\n");
#endif

   // op. 109
   strcat(SpoRes, pS_Pupil_Data->gender_code);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_Pupil_Data->age);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_Pupil_Data->sprint_points);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_Pupil_Data->jump_points);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_Pupil_Data->throw_points);
   strcat(SpoRes, " ");
   itoa(point_sum, result, 2);
   strcat(SpoRes, result);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_Pupil_Data->name);
   strcat(SpoRes, "\n");

#if TRACE_DETAIL
   printf("\t pupil result = %s", SpoRes);
#endif

   fputs(SpoRes, pF_SchoolRes);   // op. 110
}
```

7.5.17 displaySchoolWinners()

```
/**
 * displaySchoolWinners() generates - together with completeSchoolSpoRes() -
 * the records for the winning schools in either the national or the
 * international linked list.
 * The first "gold" winner gets the "G" flag in the output. If there are more
 * "gold" winners, their records are added without flagging them. The same
 * happens to the "silver" and "bronze" winners.
 * Therefore the module consists of an outer loop for the first medal winners in
 * a group of duplicates. The latter are processed in an inner loop (which is
 * seldom used because duplicates are very rare or nonexistent).
 * The main difference between the records for national or international winning
 * schools is the prefix with the country code or "AAA". The region ID is always
 * blank, and the school # is all zeroes.
 * Please note that every school entry contains the complete school data whereas
 * the output records show this data in three records per school.
 * If the linked list contains an "index loop" along the downlink chain, the
 * module does not terminate the program abnormally. It generates simply too
 * many winner records.
 */
void displaySchoolWinners(_Bool          National,
                          FILE         * pF_SchoolRes,
                          struct C_R   * pS_CntRes,
                          struct L_L   * pS_LinkList)
{
   char   prefix[14]    = {' '};
   char * psa_G_S_B[3]  = {"G", "S", "B"};
   char   result[10]    = {' '};
   char   SpoRes[81]    = {' '};

   int    crt_i         = 0;
   int    entries       = 0;
   int    line_no       = 0;                // op. 99
   int    medal_level   = -1;               // op. 102
   int    nxt_i         = 0;

   const
   int    MAX_ENTRY     = LL_LH_CEIL;

#if TRACE_MODULE
   printf("\n displaySchoolWinners()\n");
#endif

   nxt_i = pS_LinkList->chain_top_index;    // op. 93

   if (National)
```

```
{
   // op. 112 w.o. line #
   substr(prefix, pS_LinkList->LLE[nxt_i].School_Data.school_ID,
          S_COUNTRY_OFFSET, COUNTRY_LEN);
   strcat(prefix, "  00000000");
}
else
{
   // op. 117 w.o. line #
   strcpy(prefix, "AAA  00000000");
};

// I40: generate winner groups
while (nxt_i > -1 &&
       entries < MAX_ENTRY)
{
   // display 1st medal winner
   // op. 101,103,62
   line_no++;
   medal_level++;
   crt_i = nxt_i;
   entries++;

   // op. 112/117: add line #
   strcpy(SpoRes, prefix);
   itoa(line_no, result, 2);
   strcat(SpoRes, result);
   strcat(SpoRes, psa_G_S_B[medal_level]);                    // op. 107

   completeSchoolSpoRes(pF_SchoolRes,
                        SpoRes,
                        prefix,
                        &pS_LinkList->LLE[crt_i].School_Data,   // op. 111/116
                        &line_no);

   // op. 126,96/97
   pS_CntRes->result_recs += 3;
   nxt_i = pS_LinkList->LLE[crt_i].downlink;

   // I42/I43: display optional other (inter-)national winners
   while (nxt_i > -1                              &&
          pS_LinkList->LLE[nxt_i].entry_value ==
          pS_LinkList->LLE[crt_i].entry_value     &&
          entries < MAX_ENTRY)
   {
      // display winner of same medal
      // op. 101,62,108
```

```
        line_no++;
        crt_i = nxt_i;
        entries++;

        // op. 112/117: add line #
        strcpy(SpoRes, prefix);
        itoa(line_no, result, 2);
        strcat(SpoRes, result);
        strcat(SpoRes, " ");   // op. 108

        completeSchoolSpoRes(pF_SchoolRes,
                             SpoRes,
                             prefix,
                             &pS_LinkList->LLE[crt_i].School_Data,// op. 111/116
                             &line_no);

        // op. 126,96/97
        pS_CntRes->result_recs += 3;
        nxt_i = pS_LinkList->LLE[crt_i].downlink;
      };
   };
}
```

Anmerkungen

`displaySchoolWinners()` funktioniert ähnlich wie `displayPupilWinners()`. Allerdings unterscheiden sich die Ausgaben der nationalen von den internationalen Gewinnerlisten durch das Voranstellen des jeweiligen Land-Codes oder das Voranstellen von `'AAA'` (Op. 112 oder Op. 117). Der boole'sche Parameter `National` wird entsprechend auf `TRUE` beziehungsweise `FALSE` gesetzt. Die aktuellen Parameterwerte sind keine C-Pseudokonstanten, sondern `#define`-Strings:

```
#define FALSE         0
#define TRUE          1
```

Pro Schule sind drei Sätze auszugeben und fortlaufend zu numerieren. `displaySchoolWinners()` baut den Anfang des ersten Satzes auf und übergibt die restliche Arbeit an `completeSchoolSpoRes()`. Dieses meldet die zuletzt erreichte Zeilennummer – als Call-by-Reference-Parameter – zurück. `completeSchoolSpoRes()` ist so einfach, dass es nicht kommentiert wird. Die danach folgende Prozedur `displayLinkedList()` muss nur dann übersetzt werden, wenn ein Link-Listen-Trace angefordert wird. Sie enthält ebenfalls keinen kommentierungswürdigen Code.

7.5.18 completeSchoolSpoRes()

```
/**
 * completeSchoolSpoRes() builds and completes three records for a winning
 * school, based on the prefix which is generated by displaySchoolWinners().
 * It adds the detail data for each record type to the prefix and writes the
 * records.
 */
void completeSchoolSpoRes(FILE         * pF_SchoolRes,
                          char           SpoRes[],
                          char           prefix[],
                          struct SID_D * pS_School_Data,
                          int          * pi_line_no)
{
   char   result[10]    = {' '};

#if TRACE_MODULE
   printf("\n completeSchoolSpoRes()\n");
#endif

   // op. 113
   strcat(SpoRes, pS_School_Data->school_ID);
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_School_Data->name);
   strcat(SpoRes, "\n");

#if TRACE_DETAIL
   printf("\t School result a = %s", SpoRes);
#endif

   fputs(SpoRes, pF_SchoolRes);   // op. 110

   (* pi_line_no)++;              // op. 101

   // op. 112/117: add line #
   strcpy(SpoRes, prefix);
   itoa(* pi_line_no, result, 2);
   strcat(SpoRes, result);
   // op. 114
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_School_Data->address);
   strcat(SpoRes, "\n");

#if TRACE_DETAIL
   printf("\t School result b = %s", SpoRes);
#endif
```

```
   fputs(SpoRes, pF_SchoolRes);   // op. 110

   (* pi_line_no)++;              // op. 101

   // op. 112/117: add line #
   strcpy(SpoRes, prefix);
   itoa(* pi_line_no, result, 2);
   strcat(SpoRes, result);
   // op. 115
   strcat(SpoRes, " ");
   strcat(SpoRes, pS_School_Data->values);
   strcat(SpoRes, "\n");

#if TRACE_DETAIL
   printf("\t School result c = %s", SpoRes);
#endif

   fputs(SpoRes, pF_SchoolRes);   // op. 110
}
```

7.5.19 displayLinkedList()

```
/**
 * displayLinkedList() is a test aid. It was used during most of the
 * development phase in order to verify the linked list's functionality. When
 * the program runs in the production environment, displayLinkedList() is not
 * needed and can be suppressed by not compiling it.
 */
#if TRACE_LL
void displayLinkedList(   struct L_L   * pS_LinkList)
{
   char result[10]  = {' '};
   char SpoRes[68]  = {' '};

   int  crt_i       = 0;
   int  entries     = 0;
   int  nxt_i       = 0;

   const
   int    MAX_ENTRY = LL_LH_CEIL;

   printf("\n displayLinkedList()\n");

   printf("\t chain_top_index = %d\n", pS_LinkList->chain_top_index);
   printf("\t chain_end_index = %d\n", pS_LinkList->chain_end_index);
   printf("\t unique_values   = %d\n", pS_LinkList->unique_values);
   printf("\t max_list_level  = %d\n", pS_LinkList->max_list_level);
```

```
nxt_i = pS_LinkList->chain_top_index;

while (nxt_i > -1 &&
       entries < MAX_ENTRY)
{
   crt_i = nxt_i;
   entries++;

   printf("\t entry_index = %d\n", crt_i);
   printf("\t entry_value = %d\n", pS_LinkList->LLE[crt_i].entry_value);
   printf("\t uplink      = %d\n", pS_LinkList->LLE[crt_i].uplink);
   printf("\t downlink    = %d\n", pS_LinkList->LLE[crt_i].downlink);

   if (pS_LinkList->contestant_type == 'P')
   {
      strcpy(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.country_code);
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.region_id);
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.school_number);
      strcat(SpoRes, "nn ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.gender_code);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.age);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.sprint_points);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.jump_points);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.throw_points);
      strcat(SpoRes, " ");
      itoa(pS_LinkList->LLE[crt_i].entry_value, result, 2);
      strcat(SpoRes, result);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].Pupil_Data.name);
      printf("\t pupil result = %s\n", SpoRes);
   }
   else
   {
      strcpy(SpoRes, pS_LinkList->LLE[crt_i].School_Data.school_ID);
      strcat(SpoRes, " ");
      strcat(SpoRes, pS_LinkList->LLE[crt_i].School_Data.name);
      printf("\t school ID & name = %s\n", SpoRes);
      printf("\t School address = %s\n",
             pS_LinkList->LLE[crt_i].School_Data.address);
      printf("\t School result = %s\n",
             pS_LinkList->LLE[crt_i].School_Data.values);
   };
```

```
      nxt_i = pS_LinkList->LLE[crt_i].downlink;
   };
}
#endif
```

7.5.20 itoa() und substr()

Die Prozedur `itoa()` ist bereits aus ProcessWordList im Band 2-A1 bekannt. `substr()` wurde bereits in Example1 im selben Band definiert. In beiden Modulen wurden die `exit()`-Nummern geändert.

Endnoten

1 [Wikipedia-Seitentitel: Eleganz
Datum der letzten Bearbeitung: 3. Dezember 2021, 12:03 UTC
Versions-ID der Seite: 217842505
Permanentlink: https://de.wikipedia.org/w/index.php?title=Eleganz&oldid=217842505
Datum des Abrufs: 15. Juni 2022, 15:15 UTC]