Migration von Fortran 95 Libraries (DLLs) nach C++

Das Problem / die Aufgabe

Wir erhielten den Fortran 95 Quellcode samt Projektdateien zur Nutzung in einer Entwicklungsumgebung und sollten diesen syntax- und funktionsäquivalent in C++ Code umsetzen. Der Fortran 95 Quellcode umfaßte mehrere tausend Zeilen und enthielt SUBROUTINEs, FUNCTIONs und MODULEs, die zu diversen Dynamic Link Libraries (DLLs) zusammengebunden wurden. Die DLLs waren Bestandteil eines umfangfreichen Programmsystems mit graphischer Benutzeroberfläche, das Bilddaten verarbeitete. Das Programmsystem war in C++ programmiert und rief die Fortran-Routinen in der DLL.
Bei der Sichtung des Fortran 95 Quellcode wurde schnell klar, daß zur Umsetzung nach C++ keine automatischen Konvertierer wie bspw. FOR_C oder F2C helfen würden, da diese lediglich mit Fortran 77 Code operieren können. Neben einigen kleineren Problemen, wollen wir hier die wesentlichen aufzeigen:

▼ Problem: in C++ gibt es weder eine direkte feldweise Verarbeitung noch die der maskierten

Die mit Fortran 90 eingeführten feldweise operierenden Befehle gibt es in C++ nicht. Und davon wurde in dem uns vorliegenden Fortran 95 Quellcode extensiv Gebrauch gemacht. Einfaches Beispiel:

```
REAL(4) rMatrix1(n,m), rMatrix2(n,m)
rMatrix2 = rMatrix2   ! alle Werte werden kopiert
```

Leider bietet C++ keine feldweise operierenden Befehle, die bspw. obigen Fortran 90/95 Quellcode sofort bspw. wie folgt umzusetzen gestatten:

```
float rMatrix1[m][n], rMatrix2[m][n];
rMatrix2 = rMatrix2;  // wird von C++ nicht verarbeitet
```

Um die Werte von einem Feld zum anderen zu übertragen, wäre ein Schleifenkonstrukt vonnöten. Man mag einwenden, daß dies kein Problem ist, womit man für einfache Befehle nicht unrecht hätte. Jedoch bestand der Fortran 95 Quellcode aus zahlreichen und um einiges komplexeren Anweisungen, wie bspw. die folgenden zeigen:

```
COMPLEX(8) ct(-o2:o1)
REAL(8) rf(-n:n), t, p(0:n)
REAL(4), ALLOCATABLE :: c(:,:,:)
LOGICAL(1), ALLOCATABLE :: laMask(:)
ct(-o2:(2*n-o2)) = DCMPLX(rf,0.)
! Das komplexe Feld ct(:,:) erhält für den angegebenen Bereich
! (-o2:(2*n-o2)) die durch die DCMPLX Funktion erzeugten Werte
! mit auf dem Feld rf(:,:) stammenden Werten im Realteil
t = (SUM(DBLE(p(0:n1)) + SUM(DBLE(c(i,:,k)),MASK=laMask)) / m
! Es wird über Elemente der Felder p(:) und c(:,:,:) summiert,
! wobei im ersten Therm ein Bereich (0:n1) und im zweiten ein
! Bereich mit Maskierung angegeben wird. Es werden dort also nur
! die Elemente c(i,j,k) addiert, für die laMask(j)=.TRUE. ist.
```

Hier werden Feldbereiche herausgegriffen, sog. intrinsische Funktionen von Fortran verwandt, die selbst wieder feldweise operieren, und es werden Maskierungen eingesetzt, die wie feldweise operierende IF-Bedingungen wirken. Diese Möglichkeiten bietet C++ nicht von Haus aus.

▼ Problem: C++ verfügt nicht über Fortrans Feldindizierungsmöglichkeiten und indiziert "versetzt" sowie "transponiert"

Die obigen Beispiele zeigen auch gleich zwei weitere Probleme bei der Konvertierung von Fortran nach C++: der Umgang mit Feldindices. Zum einen gibt es in C++ keine Feldbereiche (ein Feld wird beginnend mit 0 indiziert) und zum anderen sind die Indices mehrdimensionaler Felder in C++ vertauscht. In C++ werden die Feldelemente im Speicher anders abgelegt als in Fortran, nämlich "transponiert": In Fortran ist der erste Feldindex derjenige, der aufeinanderfolgende Elemente im Speicher adressiert, in C++ ist es der letzte. Dieses Problem wird akut, sobald Aufrufe ein und derselben Routine mit Übergabe von mehrdimensionalen Feldern sowohl von C++ ("C++ ruft Fortran Routine SUB") als auch von Fortran stattfinden. ("Fortran ruft Routine SUB").
Die Umsetzung der Feldindizierung nach C++ fordert zudem die Beachtung, daß die Feldindizierung in C++ bei 0 beginnt und in Fortran bei 1, sofern bei der Felddeklaration nichts anderes angegeben wurde. Da eine automatisierte Quellcode-Konvertierung nicht möglich war, ergab sich angesichts der Gefahr von Fehlern bei der manuellen Umsetzung ("bei den Fortran Feldindices bei der Umsetzung nach C++ immer 1 abziehen"), die Notwendigkeit zu einer Alternative, die das Fehlerrisiko minimierte.

▼ Problem: Fortran POINTER sind keine C++ pointer

Mit Fortran 90 wurde eine neuer Typ namens POINTER eingeführt, dessen Funktionalität mit dem C++ pointer nur bedingte Gemeinsamkeiten hat. Das wird an einem einfachen Beispiel deutlich:

```
REAL rMat(n)
REAL, POINTER :: pMat(:)
pMat => rMat(2:n:2)
! pMat stellt nun eine Auswahl der Elemente von rMat dar (jedes
! Zweite), wobei eine Änderung eines Elements j in pMat(j) die
! gleichzeitige Änderung des entsprechenden Elements in rMat(2*j)
! zur Folge hat - und umgekehrt.
```

Der Fortran POINTER stellt somit eher einen Alias oder eine "umkehrbare" Zugriffsfunktion dar. Dafür gibt es in C++ kein Äquivalent. Auch der Fortran POINTER wurde oft in den zu portierenden Quellen genutzt.

Die Lösung

Um die sich aus der unterschiedlichen Verarbeitung von Feldern sowie den oben aufgeführten in C++ fehlenden Möglichkeiten im Umgang mit Feldern ergebenden Probleme zu lösen, bedienten wir uns zum einen des C++ Präprozessors und programmierten Makros, und zum anderen der objektorientierten Möglichkeiten von C++. Wir führten ein neues Objekt ein, das **mArray**, das wir mit den notwendigen Methoden ausstatteten und für das wir die üblichen Operatoren erweiterten (operator overloading). Beispiel:

```
#include "qtF2CMatrixUse.h"
// und weitere Typ Definitionen (bspw. für INT4 und REAL4)
void calc( INT4 n, REAL4* arrX, REAL4* arrY ) {
mArray<REAL4> mArrX = mArray<REAL4>(1,n,
```

```
                              MARRAY_TYPES::MARRAY_TYPES_FORTRAN);
mArray<REAL4> mArrY = mArray<REAL4>(1,n,
                              ARRAY_TYPES::MARRAY_TYPES_FORTRAN);
// Werte aus dem Fortran Feld ins mArray übertragen
mArrX.put(arrX,n);
mArrY.put(arrY,n);
...
mArrX = mArrY;       // feldweise Verarbeitung wie in Fortran
```

Damit erreichten wir das Ziel der syntaxäquivalenten Programmierung in C++ und waren in der Lage, auch kompakte Fortran Befehle wie bspw.

```
t = ( SUM(p(k:k+n)) + SUM(p(n-k:nH-k)) ) / m
```

in C++ einzeilig umzusetzen, und ohne über die Indexbehandlung weiter nachdenken zu müssen.

```
t = ( sum((p[SI((k,k+n,1)]) + SUM(p[SI(n-k,nH-k,1)]) ) / m
// Die Funktion SI vermittelt die zu verwendenden Indexbereiche.
```

Als problematisch erwies sich noch der Nachweis der funktionalen Äquivalenz. Gleichheit kann wegen numerischer Ungenauigkeiten bei Gleitkommaberechnungen nicht gefordert werden. Da die Programme mit Datenmengen in der Größenordnung von einigen Megabyte bis zu 2 Gigabyte operierten, war eine manuelle Überprüfung ausgeschlossen. Auch ein Vergleich der Ergebnisdateien mittels entsprechender Werkzeuge (bspw. WinMerge oder WinDiff) war aufgrund numerischer Ungenauigkeiten kein gangbarer Weg. Zudem mußten wir aufgrund fehlender Testdaten solche generieren. Da von dem übergeordneten Programmsystem die Fortran DLLs gerufen wurden, erweiterten wir die gerufenen FUNCTIONs und SUBROUTINEs um Protokollroutinen (eingebettet in Compiler-Metadirektiven, um ihre Wirkung auch abschalten zu können), die die Werte der Aufrufparameter und der in den MODULEs definierten und betroffenen globalen Variablen und Felder in Dateien schrieben. Diese Dateien konnten wir dann in eigens geschriebenen Testprogrammen einlesen, darin die nach C++ konvertierten Funktionen rufen, und deren Ergebnisse mit den zuvor erhaltenen vergleichen und nachweisen, daß die Ergebnisse mit hinreichender Genauigkeit übereinstimmten. Zudem konnten wir bei der Umsetzung des Fortran Quellcodes nach C++ einige Programmierfehler korrigieren.

Kunde:	*k.A. (wg. Geheimhaltungsvereinbarung)*
Projektzeitraum:	*2012*
Verwendete Software:	*Intel Visual Fortran v12.1, Microsoft Visual Studio 2010 Professional, Microsoft Visual C++*
Stichworte:	*Fortran 95, C++, DLL, mehrdimensionale Felder (arrays), Windows, FOR_C*

QT software GmbH
Konstanzer Str. 10
D-10707 Berlin
Germany
Telefon 030/9290087-0
Telefax 030/9290087-2
eMail info@qtsoftware.de
http://www.qtsoftware.de
Shop http://www.qtsoftware.com