ARAMIS

Quick Basic, PDS Basic nach C/ C++ Konverter

1 Einleitung

ARAMIS ist ein Konverter, der automatisch Basic-Code in C oder C++-Code umsetzt. Hierbei werden vom Konverter nahezu alle Basic Befehle, Variablendefinitionen und Sprachkonstrukte nach C oder C++ gelesen und interpretiert.

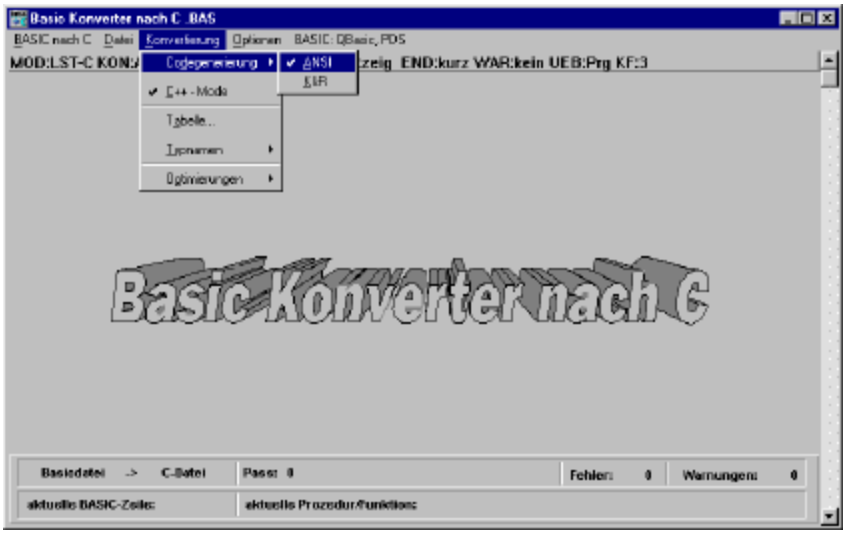
Der durch ARAMIS generierte C/C++-Code kann mit gängigen Compilern (Microsoft, Borland, Symantec, etc.) kompiliert werden. Die hierzu notwendigen Dateien sind enthalten. Zur Lauffähigkeit der so kompilierten Quellen sind weitere Bibliotheken/Klassenbibliotheken notwendig, die direkt bei CiceroSoft geordert werden können. Hier erhalten Sie auch kompetente Beratung für die Umsetzung von Basic Projekten. Ebenso übernimmt CiceroSoft die Umsetzung von Projekten auf Basis von ARAMIS bzw. dessen großem Bruder THALES.

2 Bedienung

2.1 Installation

Legen Sie die mit ARAMIS 1/x beschriftete Diskette ein und starten Sie das Programm setup.exe. Das Setup Programm gibt die notwendigen Anweisungen, um ARAMIS in ein von Ihnen bezeichnetes Verzeichnis auf Ihrem Computer zu installieren.

2.2 Die Oberfläche



Die Oberfläche besitzt ein Pulldown-Menü mit den Titeln „Datei“, „Konvertierung“, „Optionen“ und der Anzeige, welcher Basic Dialekt übersetzt werden kann.

Unter diesen Titeln finden Sie die folgenden Einträge:

Datei:

- *Basic Programm konvertieren*

  Läßt eine Basicdatei über Fileselektor auswählen. Diese wird dann konvertiert und unter gleichem Namen mit der jeweiligen Endung .c bzw. .cpp (C++ Mode) gesichert.

- *Exit*

Beendet das Programm

Konvertierung:

- *Codegenerierung*
  Stellt ein, ob Kernighan und Ritchie Typdefinitionen, Parameterübergaben (alte Schreibweise) oder der neuere ANSI Standard vom Konverter geschrieben wird. Standardwert: ANSI
- *C++-Mode*
  Schaltet den C++ Mode des Konverters ein
- *Tabelle*
  Es können 10 Ausdrücke eingegeben werden, die der Konverter direkt nach der Definition der rechten Seite ersetzt.
- *Typnamen*
  Schreibt die Namen der Variablentypen (z.B. char, int, etc.) nach der ANSI Norm (wie z.B. int) oder nach der portablen Definition wie in portab.h festgelegt. Standardwert: ANSI
- *Optimierungen*
  Es lassen sich formale Optimierungen des Quellcodes erzeugen, wenn der Konverter feststellt, daß z.B. geschweifte Klammern unterdrückt werden können (z.B. es folgt nach if() nur ein einziger Ausdruck) oder es lassen sich über mehrere Zeilen hinweg logische Bedingungen zu einzeiligen Ausdrücken kombinieren.
  z.B. kann:

  if bed = TRUE THEN a% = 1 ELSE a% = 2

  in C/C++ zu

  a = (bed == TRUE) ? 1: 2).

  formal optimiert werden.

Optionen:

- *Umwandlung zeigen*
  Bewirkt, daß nach einer Konvertierung generell der erzeugte C/C++ Code und der Basiccode in der Referenzanzeige des Konverters angezeigt wird. Für die Anzeige sind die folgenden 3 Bedingungen notwendig: 1. Referenzierung ist eingeschaltet. 2. Unter Speicher wurde angegeben, daß der gesamte Code im Speicher gehalten wird. 3. Der Basiccode arbeitet nicht mit #INCLUDE Anweisungen.
- *Referenzierung*
  Schaltet die innere Verwaltung einzelner Basic und C-Zeilen ein (braucht mehr Speicherplatz und kann daher abgeschaltet werden)
- *Endungen*
  Variablenendungen können in einer kurzen oder langen Schreibweise generiert werden (z.B. als Endung : I für Integer oder in der langen Schreibweise: _int)
- *Statistiken*
  Wenn eine Konvertierung mit Erfolg stattgefunden hat, können hier kurze Übersichten über Variablen, Felder oder Funktionen erhalten werden
- *Klammern*
  Formatierung von geschweiften Klammern, die Bedingungen, Schleifen oder Funktionen (allgemein Blöcke in C/C++) zusammenfassen, können hier ausgewählt werden.
- *Speicher*
  Der Konverter kann größere Dateien beim Konvertieren auf die Platte auslagern oder auch das Konvertierungsergebnis im Speicher halten. Dies wird in der Dialogbox festgelegt.

Das letzte Element der Menü-Bar zeigt den Basicdialekt, der im Moment vom Konverter bearbeitet wird (z.B. Qbasic, PDS Basic, SNI Basic, HP Basic, GFA Basic)

2.3 Konvertierung

Wurde eine Basicquelle ausgewählt, läuft die Übersetzung des Basic Quellcodes automatisch. Der Konverter zeigt dabei in der Statuszeile alle wichtigen Informationen an, wie Name des Programmes, Funktionen, in denen er gerade arbeitet, Zeilennummern und Status von Fehlern bzw. Warnungen. Treten beim Übersetzungsvorgang Probleme auf, so meldet der Konverter dies in Alertboxen.

2.3.1 Einflußnahme auf die Konvertierung

Neben den formalen Anweisungen, die im Menü eingestellt werden können, läßt sich die Konvertierung auch durch Eingriffe im Quellcode selbst steuern:

1. Ausblendung von Blöcken des Basic Listings (REM $RA - REM $RE). Trifft ARAMIS auf diese Ausdrücke, werden alle Basiczeilen, die zwischen den Blöcken liegen, überlesen. Die Blöcke dürfen auch geschachtelt vorkommen.
2. Einblendung von C oder C++ Code direkt in das Basic Listing (REM BNC-C)
3. Einblendung von Basic Code direkt in das Basic Listing, der nur vom Konverter berücksichtigt wird (REM BNC-BAS).
4. Einblendung von C oder C++ Code an beliebigen Stellen von H Dateien (REM BNC-H[nr]:, nr: Zahl von 1 bis 5)).
5. Variablenumdefinition von Auto Variablen zu Static Variablen (REM BNC-VAR:STATIC).
6. Generierung von einzelnen Modulen (nur in THALES verfügbar) (REM BNC-MODUL:modulname).

2.4 Wahl des Zielcodes

Unter dem Menüpunkt C++ Mode kann der zu erzeugende Code grundsätzlich gewählt werden. Es gibt zwei verschiedene Möglichkeiten: Entweder generiert der Konverter Ansi C Code, oder - bei Einschaltung des C++ Modes - reinen C++ Code, soweit dieser durch das Basic möglich wird. Der C++ Code ist wegen der dem Basic näheren Schreibweise – besonders bei Strings – besser lesbar als der ältere Ansi C Code. Bei diesem werden viele Operationen durch Funktionsaufrufe durchgeführt (vgl. Anhang: Beispielkonvertierungen)

2.5 Der Vorgang der Konvertierung

Während der Konvertierung arbeitet ARAMIS in mehreren Phasen (Pässen), um ein möglichst optimales Ergebnis für die spätere Verwendung und Weiterentwicklung zu erzielen. Die Pässe werden in der Statusleiste angezeigt.

Der Konverter liest die einzelnen Basicbefehle, Variablen und Funktionen und setzt sie in C oder C++ Code um. Eine Vorarbeit in der Basicquelle ist hierfür nicht notwendig.

2.6 Das Ergebnis eines Konvertierungslaufes

Betrachten Sie bitte hierzu auch die Beispielkonvertierungen im Anhang.

2.6.1 Variablen

Variablen werden während der Umsetzung zu speziellen Variablentypen nach C/C++ umgesetzt mit dem Ziel, daß sie analog zu Basic in C/C++ verwendbar und auf alle Betriebssysteme portabel sind. So wird zum Beispiel aus dem Datentyp „String“ der neue Typ „B_String“.

2.6.2 Funktionen/Subs, Markenprozeduren

Funktionen werden vom Namen her gleich dem Original geschrieben; um sie jedoch unverwechselbar zu Types oder Variablen zu machen wird ihnen ein Präfix vorangestellt:

F_ für Funktionen

P_ für Prozeduren, Markenprozeduren und SUBs

2.6.3 Felder

Felder (Arrays) werden ebenso wie Variablen automatisch von ARAMIS erkannt und nach C/C++ umgesetzt. Wegen der höheren Komplexität ist das Konvertierungsergebnis in der Regel nicht mehr so gut lesbar wie die analoge Umsetzung von Variablen. Mit dem Ergebnis der Konvertierung werden jedoch in voller Konsistenz alle Eigenheiten des Basic bei Feldern auch unter Ansi C realisiert (vgl. Anhang).

2.7 Erzeugte Dateien

Während der Konvertierung legt ARAMIS neben der Erzeugung der C bzw. CPP Dateien, in denen der direkt konvertierte Basic nach C/C++ Code steht, alle globalen Daten in verschiedenen Dateien ab.

Diese Dateien tragen immer den Namen der Zieldatei mit den Endungen

„.h“ für die Deklaration von Variablen, Feldern, Funktionen etc.

„.typ“ für die Deklaration von Strukturen, die aus den TYPE Variablen generiert werden

3 Die konvertierten Quellen mit einem C oder C++-Compiler kompilieren

Nach erfolgreicher Konvertierung des Basic-Codes, kann der C/C++-Code kompiliert werden.

Obwohl ARAMIS als Werkzeug zur automatisierten Umstellung von Basicprogrammen konzipiert ist, kann an dieser Stelle unter Umständen ein manueller Eingriff notwendig werden.
Das Ausmaß solcher Korrekturen hängt aber sehr stark vom Basic-Code ab, und kann in den meisten Fällen durch Modifikationen im Basic verhindert werden.

Zu beachten sind daher folgende Dinge schon vor der Konvertierung:

- Basic- Code muß fehlerfrei Interpreter/Compiler durchlaufen
- Konsequente Programmierung, z.B. Variablen immer eindeutig bezeichnen (Typendungen %, $, ...)
- Keine Verwendung von Schlüsselwörtern als Variable
- Alle verwendeten Strukturen, Funktionen und Subs müssen bekannt sein

Bei der Kompilierung sind die verschiedenen Headerdateien sowohl des Quell-Codes als auch der Bibliothek zur Verfügung zu stellen.

Siehe hierzu auch das Beispielprogramm DEMO.BAS im Anhang (mit Microsoft DevStudio 5.0 - 32-Bit - kompiliert).

4 Konvertierungen größeren Umfangs

Falls Sie größere Umsetzungen von Basicprojekten planen, so kann Ihnen ARAMIS zwar den grundsätzlichen Weg weisen, jedoch aufgrund der notwendigen Einschränkungen (Maximallänge der BAS-Datei 32 K, keine Bibliotheken) als „kleines Werkzeug zum Minimalpreis“ nur bedingt komfortabel helfen.

CiceroSoft GmbH verwendet zur Umsetzung die Werkzeuge **THALES** für PDS, Qbasic, Quick Basic, SNI Basic, HP Basic und **HOMER** für die Umsetzung von Visaul Basic Projekten. Diese Konverter sind ganz und gar Profiwerkzeuge mit dem notwendigen Anspruch an Bedienung und Steuerung. Daher kann es nicht verwundern, daß diese Werkzeuge nicht zum gleichen Preis wie ARAMIS angeboten werden können.

Sollten Sie Interesse an diesen Profi-Tools oder an einer Dienstleistung unseres Hauses haben, so freuen wir uns über Ihre Rückmeldung:

Bitte schicken Sie uns hierzu ein Fax an eine der Nummern (06803) 9940-40 oder 41, bzw. senden Sie uns eine Email unter der Adresse cicero@cicerosoft.com. Wir werden uns dann schnellstmöglich mit Ihnen in Verbindung setzen.

ANHANG

An dieser Stelle soll das Ergebnis eines Konverterlaufes an Hand des Beispielprogrammes DEMO.BAS beleuchtet werden.
Das Konvertat wurde ohne manuelle Korrekturen mit 0 Errors unter MSDEV 5.0 als 32-Bit-Anwendung kompiliert.
Gezeigt werden in dem Programm der Umgang mit grundsätzlichen Basic-Elementen wie Strings, Types, Arrays, globalen Variablen, Schleifen und Dateihandling.
Die Dateien DEMO.C und DEMO.CPP zeigen dann jeweils das entsprechende Konvertat.

Allgemein

ARAMIS erzeugt aus der Datei DEMO.BAS die folgenden Dateien (C-Version)

- DEMO.C
- DEMO.H
- DEMO.TYP
- VCOMMON.C
- VCOMMON.H

DEMO.C
Konvertat des eigentlichen Basicprogrammes, bestehend aus einem MAIN-Teil und den Funktionen.
Im Programmkopf wird folgende Datei eingebunden:

DEMO.H
Einbindung der Headerdatei ARAMIS.H für die Bibliothek,
Listung aller vorkommenden Defines (Basic: CONST)
Einbindung der Dateien DEMO.TYP, VCOMMON.H und PROJEKT.H. Letztere wird vom Anwender bereitgestellt.
Deklaration aller Funktionen.

DEMO.TYP
Enthält die UserTypes, die in der C-Version als Strukturen vorliegen und in der C++-Version als Klassen mit einem Konstruktor zur Initialisierung der Klassenelemente.

VCOMMON.C
Stellt die globalen Variablen bereit; dies geschieht über die Einbindung von VCOMMON.H in Zusammenhang mit einer EXTERN-Definition.
Weiterhin wird ein Stack bereitgestellt, der zur Realisierung von GOSUB über setjmp/longjmp notwendig ist.

VCOMMON.H
Deklaration und Definition der globalen Variablen.

Konkrete Hinweise zum Konvertat finden Sie in den nachfolgenden Listings, wobei im C++-Listing nur noch spezielle Honweise auf C++ Eigenschaften erfolgen.

BASIC-LISTING

```
DECLARE SUB demoFile ()
DECLARE SUB demoString ()
DECLARE FUNCTION demoArray% (intVar%)
DECLARE SUB demoVar (intVar%)

TYPE typ
intVar          AS INTEGER
stringVar       AS STRING * 6
END TYPE

CONST PI = 3.141592
COMMON SHARED globalVar%

globalVar% = 0
demoFile
demoString
globalVar% = demoArray(2)
demoVar (1)

REM ****************** FILE I/O ************************
SUB demoFile
DIM userTyp AS typ

userTyp.intVar = 1
userTyp.stringVar = "USER1 "

OPEN "datei1.txt" FOR RANDOM AS #1 LEN = LEN(userTyp)
PUT #1, 1, userTyp
CLOSE #1
OPEN "datei1.txt" FOR RANDOM AS #1 LEN = LEN(userTyp)
GET #1, 1, userTyp
OPEN "datei2.txt" FOR OUTPUT AS #2
PRINT #2, userTyp.intVar, ", ", userTyp.stringVar
CLOSE
END SUB

REM ****************** STRINGS ************************
SUB demoString
DIM userTyp AS typ
DIM fixString AS STRING * 6
DIM quellString AS STRING
DIM zielString$

quellString = "links mitte rechts"
fixString = quellString
userTyp.stringVar = RIGHT$(quellString, 6)

zielString$ = fixString + MID$(quellString, 7, 6) + userTyp.stringVar

IF (zielString$ <> quellString) THEN
 PRINT "Fehler bei String-Konkatenation"
ELSE
 PRINT "QuellString = ", quellString
 PRINT "ZielString = ", zielString$
END IF

END SUB
```

```
REM ****************** FUNKTION, ARRAYS ,SCHLEIFE und SELECT CASE ***
FUNCTION demoArray% (intVar%)
DIM userTyp AS typ
DIM intArray(5) AS INTEGER

FOR i% = 1 TO 5
 intArray(i%) = i%
NEXT i%


SELECT CASE intVar%
CASE 1
      PRINT intArray(1)
CASE 2 TO 5
      PRINT intArray(intVar%)
CASE ELSE
      PRINT "Index out of Range"
END SELECT

demoArray% = intArray(intVar%)
END FUNCTION

REM ****************** LOKALE/GLOBALE VARIABLEN *****************
SUB demoVar (intVar%)
DIM localVar%
localVar% = 2 * PI
globalVar% = localVar% * intVar%
END SUB
```

C-LISTING

```
/* BASIC -> C Uebersetzung vom Wed Aug 20 15:29:06 1997
 *
 * Modul: Z:\MS\ARAMIS\SAMPLES\C\DEMO0.C
 *
 */

#include "DEMO.H"


void main(void)
     {
     Feld   intarrayI_f = NULLFELD;
            /* Initialisierung */

Initialisierungsroutinen der Bibliothek
     B_initstring(DSTR,MAXSTRING,M_AXSP);
     B_initfeld(21);
     B_init(2);

Konstanten in Basic erscheinen hier als Defines
#undef PI
#define PI  (3.141592)

     // COMMON  SHARED globalvar%;
Kennzeichnung von globalen Variablen durch führendes GV
```

```
Aufruf der Funktionen
     P_demofile();
     GVglobalvarI=0;
     P_demostring();

REFHI erledigt die Weitergabe einer Konstanten per Referenz
     GVglobalvarI=F_demoarray(REFHI(2));
     P_demovar(REFHI(1));
Programmende
     B_exit(0);
}

void   P_demofile(void)
     {
     typ    usertyp = {0};

     usertyp.intvar=1;
     B_typecpy(usertyp.stringvar ,sizeof(usertyp.stringvar ),"USER1 ");
     B_open("RANDOM",1,"datei1.txt",sizeof(usertyp));
     B_putk(1, 1,sizeof(usertyp),&(usertyp));
     B_close(1);
     B_open("RANDOM",1,"datei1.txt",sizeof(usertyp));
     B_getk(1, 1,sizeof(usertyp),&(usertyp));
     B_open("OUTPUT",2,"datei2.txt",-1);
     B_fprintf(2,"%d\t, \t%.*s\n",
          usertyp.intvar,sizeof(usertyp.stringvar),(usertyp.stringvar));
     B_closeall();
     }
     /*  ***************** STRINGS ********************** */

void   P_demostring(void)
     {
     typ    usertyp = {0};
Initialisierung der Strings
     B_String fixstringS = B_ldrealloc(MAXSTRING);
     B_String quellstringS = B_ldrealloc(MAXSTRING);
     B_String zielstringS = B_ldrealloc(MAXSTRING);


     quellstringS=B_nsprintf(quellstringS ,"links mitte rechts",18);
     fixstringS=B_strcpy(fixstringS ,quellstringS);
Typstrings bedürfen immer einer Sonderbehandlung, da keine terminierende
Null existiert
     B_typecpy(usertyp.stringvar ,sizeof(usertyp.stringvar
                                        ),B_right(quellstringS, 6));
     zielstringS =B_sprintf(zielstringS
     ,"%.*s%s%.*s",sizeof(fixstringS),(fixstringS),(B_mid(quellstringS,7,
                    6)),sizeof(usertyp.stringvar),(usertyp.stringvar));
Stringvergleich
     if((long)(B_strcmp(6,zielstringS,quellstringS)))
           {
           B_printf(-1,-1,"Fehler bei String-Konkatenation\n");
           }
     else
           {
           B_printf(-1,-1,"QuellString = \t%s\n",(quellstringS));
           B_printf(-1,-1,"ZielString = \t%s\n",(zielstringS));
           }
Freigabe des Stringspeichers
     B_dsfree(zielstringS);
     B_dsfree(quellstringS);
     B_dsfree(fixstringS);
     }
```

```
/*  ****************** FUNKTION, ARRAYS ,SCHLEIFE und SELECT CASE *** */
B_Int  F_demoarray(B_Int  huge *intvarI)
     {
     B_Int  demoarrayI = 0;
     typ    usertyp = {0};
     Feld   intarrayI_f = NULLFELD;
     B_Int  iI = 0;


     /* DIM intarray(5) */
Speicherzuweisung für das Array
     intarrayI_f.p.I = (B_Int  huge *)
                              B_dim(DIM_INT,&intarrayI_f,0,1,(long)5);
     for(iI=1;iI<=5;(iI)++)
           {
           intarrayI_f.p.I[iI]=iI;
           }
     switch( *intvarI)
           {
           case 1:
                 B_printf(-1,-1,"%d\n",intarrayI_f.p.I[1]);
                 break;
           default:
Abhandlung von CASE 2 TO 5; in dieser Art werden auch String-Argumente
behandelt
                 if(( *intvarI) >= 2  && ( *intvarI) <= 5))
                 {
                 B_printf(-1,-1,"%d\n",intarrayI_f.p.I [*intvarI]);
                 }
                 else
                 {
                 B_printf(-1,-1,"Index out of Range\n");
                 }
                 break;
                 }
Bildung des Rückgabewertes
     demoarrayI=intarrayI_f.p.I[ *intvarI];
Speicherfreigabe
     B_free(&intarrayI_f);
     return(demoarrayI);
     }
     /*  ****************** LOKALE/GLOBALE VARIABLEN ***************** */

void   P_demovar(B_Int  huge *intvarI)
     {
     B_Int  localvarI = 0;

Zugriff auf Konstante PI
     localvarI=2*PI;
Multiplikation mit dereferenziertem Parameter
     GVglobalvarI=localvarI* *intvarI;
     }
```

C++-LISTING

```
/* BASIC -> C Uebersetzung vom Wed Aug 20 15:49:06 1997
 *
 * Modul: Z:\MS\ARAMIS\SAMPLES\CPP\DEMO0.CPP
 *
 */

#include "DEMO.H"


void   main(void)
       {
       B_ARRAY(B_Int) intarrayI_f(1,1L,(long)5);
              /* Initialisierung */
       B_initstring(DSTR,MAXSTRING,M_AXSP);
       B_initfeld(21);
       B_init(2);

#undef PI
#define PI   (3.141592)

       // COMMON  SHARED globalvar%;
       GVglobalvarI=0;
       P_demofile();
       P_demostring();
       GVglobalvarI=F_demoarray(REFHI(2));
       P_demovar(REFHI(1));
              // ****************** FILE I/O ***********************
       B_exit(0);
}

void    P_demofile(void)
       {
       typ    usertyp;

       usertyp.intvar=1;
       usertyp.stringvar=BS("USER1 ");
       B_open(BS("RANDOM"),1,BS("datei1.txt"),sizeof(usertyp));
       B_putk(1, 1,sizeof(usertyp),&(usertyp));
       B_close(1);
       B_open(BS("RANDOM"),1,BS("datei1.txt"),sizeof(usertyp));
       B_getk(1, 1,sizeof(usertyp),&(usertyp));
       B_open(BS("OUTPUT"),2,BS("datei2.txt"),-1);
       B_fprintf(2,"%d\t, \t%.*s\n",
        usertyp.intvar,B_strlen(usertyp.stringvar),B_strptr(usertyp.stringvar));
       B_closeall();
       }
       // ****************** STRINGS ***********************

void    P_demostring(void)
       {
   Initialisierung der Typen erfolgt per Konstruktor
       typ    usertyp;
   Initialisierung der Strings erfolgt per Konstruktor
      B_String FS(fixstringS,6);
       B_String quellstringS;
       B_String zielstringS;


   String-Zuweisungen, Vergleiche und Konkatenationen „BASIC-LIKE“;
   Jedoch ist casting mit BS notwendig
       quellstringS=BS("links mitte rechts");
       fixstringS=quellstringS;
```

```
        zielstringS=fixstringS+B_mid(quellstringS,7, 6)+usertyp.stringvar;
        if((long)(zielstringS != quellstringS))
                {
        usertyp.stringvar=B_right(quellstringS, 6);
                B_printf(-1,-1,"Fehler bei String-Konkatenation\n");
                }
        else
                {
                B_printf(-1,-1,"QuellString = \t%s\n",B_strptr(quellstringS));
                B_printf(-1,-1,"ZielString = \t%s\n",B_strptr(zielstringS));
                }
        }
        // ****************** FUNKTION, ARRAYS ,SCHLEIFE und SELECT CASE ***

    B_Int  F_demoarray(B_Int  &intvarI)
        {
        B_Int  demoarrayI = 0;
        typ    usertyp;
    Initialisierung des Arrays, ebenfalls per Konstruktor
        B_ARRAY(B_Int) intarrayI_f(1,1L,(long)5);
        B_Int  iI = 0;



        for(iI=1;iI<=5;(iI)++)
                {
    Zugriff auf Array-Elemente per Methode gA
                intarrayI_f.gA(1,iI)=iI;
                }
        switch(intvarI)
                {
                case 1:
                        B_printf(-1,-1,"%d\n",intarrayI_f.gA(1,1));
                        break;
                default:
                        if(((intvarI) >= 2  && (intvarI) <= 5))
                                {
                                B_printf(-1,-1,"%d\n",intarrayI_f.gA(1,intvarI));
                                }
                        else
                                {
                                B_printf(-1,-1,"Index out of Range\n");
                                }
                        break;
                        }
        demoarrayI=intarrayI_f.gA(1,intvarI);
        return(demoarrayI);
        }
        // ****************** LOKALE/GLOBALE VARIABLEN *****************

    void   P_demovar(B_Int  &intvarI)
        {
        B_Int  localvarI = 0;


        localvarI=2*PI;
    Dereferenzierungen nicht mehr nötig wie im C-Beispiel
        GVglobalvarI=localvarI*intvarI;
        }
```