CP/M-80-Besonderheiten

Kommandozeilen-Parameter

Das P Kommando erlaubt Ihnen einen oder mehrere Parameter einzugeben, die an Ihr Programm übergeben werden, wenn es im Memory-Modus läuft. Dies geschieht genauso wie bei der Eingabe von Parametern in der DOS Kommandozeile. Diese Parameter können durch die Funktionen ParamCount und ParamStr angesprochen werden.

Chain und Execute

Chain(FilVar)
Execute(FilVar)

Daten können vom laufenden Programm zum chained (verketteten) Programm entweder durch gemeinsame globale (shared global variables) oder durch absolute Variablen (absolute address variables) übertragen werden. Um sich vor Überlappungen zu schützen, sollten gemeinsame globale Variablen am Anfang beider Programmen deklariert werden, wobei die Reihenfolge in beiden Deklarierungen die gleiche sein muß. Weiterhin müssen beide Programme auf den gleichen Speicherplatz compiliert werden.

Beispiel: Program MAIN.COM und Chain CHRCOUNT.CHN:

```
program Main;
var
  Txt: String[80]; { global }
  CntPrg: file;
begin
  Write('Enter any text: '); Beadln(Txt);
  Assign(CntPrg, 'ChrCount.chn');
  Chain(CntPrg);
end.

program ChrCount;
var
  Txt: String[80]; { global }
  NoOfChar,
  NoOfUpc,
  I: Integer;
begin
  NoOfUpc := 0;
  NoOfChar := Length(Txt);
  for I := 1 to length(Txt) do
    if Txt[I] in ['A'..'Z'] then NoOfUpc:=Succ(NoOfUpc);
  Write('No of characters in entry: ',NoOfChar);
  Writeln('.No of upper case characters: ',NoOfUpc,'.');
```

```
end.
```

Wenn Sie wollen, daß ein TURBO Programm feststellt, ob es durch eXecute oder direkt von einer DOS Kommandozeile aufgerufen wurde, sollten Sie eine absolute Variable an der **Adresse $80** verwenden. Dort befindet sich das Byte, das die Länge der CP/M Kommandozeile enthält. Wenn ein Programm von CP/M aufgerufen wurde, enhält dieses Bit einen Wert zwischen 0 und 127, bei Aufruf durch eXecute setzt das aufgerufene Programm die Variable auf einen Wert höher als 127. Wenn Sie die Variable in dem aufgerufenen Programm prüfen, bedeutet ein Wert zwischen 0 und 127, daß das Programm von CP/M, ein höherer Wert, daß es von einem anderen Turbo Program aufgerufen wurde.

Overlays

OvrDrive(Laufwerk);

Laufwerk bezeichnet (0 = angemeldetes Laufwerk, 1 A:, 2 = B:, usw.). Bei nachfolgenden Aufrufen von Overlay-Dateien werden die Dateien auf dem angegebenen Laufwerk erwartet. Wenn eine Overlay-Datei auf einem Laufwerk geöffnet wurde, werden weitere Aufrufe derselben Datei auf demselben Laufwerk gesucht.

```
program OvrTest;
overlay procedure ProcA;
begin
  Writeln('Overlay A');
end;
overlay procedure ProcB;
hegin
Writeln('Overlay FV);
end;

procedure Dummy;
begin
  { Dummy-Prozedur, um die Overlays in zwei Gruppen zu teilen }
end;

overlay procedure ProcC;
begin
  WriteleOverlay C');
end;

begin
  OvrDrive(2);  { Laufwerk B: }
  ProcA;   { erste Overlay-Datei mit ProcA und ProcB }
  OvrDrive(0);  { selbes Laufwerk }
  ProcC;   { zweite Overlay-Datei mit ProcC }
  OvrDrive(2);  { Laufwerk B: }
  ProcB;   { erste Overlay-Datei mit ProcA und ProcB }
end.
```

Absolute Variablen

```
Var
  IOByte: Byte absolute $0003;
  CmdLine: string[127] absolute $80;

  Str: string[32];
  StrLen: Byte absolute Str; { dieselbe Adr wie Str }
```

Vordefinierte Arrays Mem[] und Ports[]

TURBO Pascal bietet zwei vordefinierte Arrays vom Typ Byte, Mem und Port, die als direkter Zugang zum CPU-Speicher und zu den Daten-Ports benutzt werden können.

```
Mem[WsCursor]:= 2;
IOByte:= Mem[3];
```

Funktionen/Prozeduren

Addr(name) Speicheradresse des ersten Bytes des Objekts MemAvail ermittelt freien Platz auf dem Heap. Ord(ptr) gibt die in einem Zeigerargument enthaltene Adresse als ganze Zahl aus. Ptr(addr) wandelt sein ganzzahliges Argument in einen Zeiger um Bdos(Func[,ParamDE]) Fkt. C BdosHL(Func[,ParamHL]) Bios(Func[,ParamBC]) Fkt. 0-WBOOT, 1-CONST usw.

Benutzergeschriebene I/O Treiber:

Variable enthält die Adresse der ConStrPtr function ConSt:boolean { wird durch KeyPressed gerufen } ConInPtr procedure ConIn:Char ConOutPtr procedure ConOut(Ch:Char) LstOutPtr procedure LstOut(Ch:Char) AuxOutPtr procedure AuxOut(Ch:Char) AuxInPtr function AuxIn:Char UsrOutPtr procedure UsrOut(Ch:Char) UsrInPtr function Usrin:Char

Externe Unterprogramme

procedure DiskReset; external $EC00; function IOStatus: boolean; external $D123;

Parameter können an externe Unterprogramme übergehen werden. Die Syntax ist genau dieselbe, wie bei normalen Prozedur- und Funktionsaufrufen: procedure Plot(X,Y: Integer); external $F003; procedure QuickSort(var List: PartNo); external $1C00;

Parameter Parameter werden an Prozeduren und Funktionen mittels des Z-80 Stack übergeben. Normalerweise braucht dies den Programmierer nicht zu interessieren, da der von TURBO Pascal erzeugte Maschinencode automatisch Parameter vor einem Aufruf auf einen Stack schiebt (PUSH) und sie bei Beginn eines Unterprogramms holt (POP). Falls ein Programmierer externe Unterprogramme verwenden will, müssen diese selbst die Parameter vom Stack holen (POP). Am Eingang zu einer external Unterroutine enthält der oberste Teil des Stack immer die Rückkehradresse (ein Wort). Die

Parameter, falls vorhanden, liegen unter dieser Rückkehradresse, d.h. auf höheren Adressen des Steck. Deshalb muß, um an die Parameter zu kommen, die Unterroutine zuerst die Rückkehradresse holen (POP), dann alle Parameter und die Rückkehradresse schließlich wieder abspeichern, indem sie sie zurück auf den Steck schiebt (PUSH). 22.19.1 Variabien-Parameter Bei einem Variablen VAR-Parameter wird ein Wort auf den Stack übertragen, indem die absolute Speicheradresse des ersten besetzten Bytes des aktuellen Parameters angegeben wird. 22.19.2 Wert-Parameter Bei einem Wertparameter hängen die auf den Stack übertragenen Daten vom Typ des Parameters ab, wie in den folgenden Abschnitten beschrieben. 22.19.2.1 Skalare Inleger, Boolean, Char und deklarierte skalare Typen werden als ein Wort auf •den Stack übertragen. Wenn die Variable bei der Speicherung nur ein Byte belegt, ist das signifikanteste Byte des Parameters 0. Normalerweise wird ein Wort vom Stack durch eine Instruktion wie POP HL geholt. 22.19.2.2 Reelle Zahlen Eine reelle Zahl wird mit 6 Bytes auf den Stack übertragen. Wenn diese Bytes mit der Instruktionssequenz POP HL POP DE POP BC geholt werden, dann enthält L den Exponenten, H das fünfte (am wenigsten signifikante) Byte der Mantisse, E das vierte, D das dritte, C das zweite und B das erste (signifikanteste) Byte. 22.19.2.3 Strings Wenn ein String an der obersten Adresse des Stack ist, enthält das Byte, auf das durch SP gezeigt wird, die Länge des Strings. Die Bytes an den Adressen SR+ 1 bis SP+n (wobei n die Länge des String ist) enthalten den String, wobei das erste Zeichen an der niedrigsten Adresse gespeichert wird. Die folgenden Maschinencode-Instruktionen können benutzt werden, um den String an der Spitze des Stack zu holen (POP) und in StrBuf zu speichern. LD DE,StrBuf LD HL,0 LD B,H ADD HL,SP LD C,(HL) INC BC LDIR LD SP,HL 22.19.2.4 Mengen Eine Menge belegt immer 32 Bytes im Stack (die Mengenkompression wird nur beim Laden und Speichern von Mengen verwendet). Die folgende Maschinecode- Instruktion kann benutzt werden, um die Menge an der Spitze des Steck zu holen (POP) und in SetBuf zu speichern. LD DE,SetBuf LD HL,0 ADD IlL,SP LD BC,32 LDIR LD SP,HL Damit wird das am wenigsten signifikante Byte der Menge an der niedrigsten Adresse in SetBuf gespeichert. 22.19.2.5 Zeiger Ein Zeiger wird als ein Wort auf den Steck übertragen, das die Adresse der dynamischen Variable enthält. Der Wert NIL entspricht dem Wort Null. 22.19.2.6 Arrays und Records Auch wenn sie als Wert-Parameter verwendet werden, werden Array- und Recordparameter nicht wirklich auf den Steck geschoben (PUSH). Stattdessen wird ein Wort übertragen, das die Adresse des ersten Bytes des Parameters enthält. Es liegt dann in der Verantwortlichkeit der Unterroutine, das Wort zu holen (POP) und es als die Ursprungsadresse in einer Blockkopieroperation zu benutzen. 22.20 Ergebnisse von Funktionen Vom Benutzer geschriebene external Funktionen müssen ihre Ergebnisse exakt in folgender Weise spezifizieren. Werte eines Skalartyps, außer reelle Zahlen, müssen in das HL Registerpaar ausgegeben werden. Wenn der Typ des Ergebnisses nur ein Byte braucht, dann muß er in das L Register ausgegeben werden, und H muß Null sein. Reelle Zahlen müssen in die Registerpaare BC, DE und HL ausgegeben werden. B, C, D, E und H müssen die Mantisse (signifikantestes Byte in B) und L muß den Exponenten enthalten. Strings und Mengen müssen an die oberste Adresse des Steck ausgegeben werden in Formaten, wie sie auf Seite 284 beschrieben sind. Zeigerwerte müssen in das HL Registerpaar ausgegeben werden.

In-line Maschinencode (Assembler)

TURBO Pascal enthält inline Anweisungen, die einen sehr bequemen Weg bieten, Maschinencode (Assembler) direkt in den Programmtext einzufügen. Eine in/ine-Anweisung besteht aus dem reservierten Wort inline, gefolgt von einer oder mehreren Konstanten, Variablenbezeichnern oder Kommandozählerreferenzen, die durch Schrägstriche getrennt und in Klammern eingeschlossen sind. Ein Codeelement ist aus einem oder mehreren Datenelementen aufgebaut, die durch (+) oder (-) Zeichen getrennt sind. Ein Datenelement ist entweder eine integer Konstante, ein Variablenbezeichner, ein Prozedurbezeichner, ein Funktionsbezeichner oder ein Kommandozählerwert. Ein Kommandozählerwert wird als (*) Stern geschrieben. Die Zeichen < und >

können verwendet werden, um die automatische Größenwahl zu überschreiben. Wenn das Codeelement mit dem Zeichen < beginnt, wird nur das wenigst signifikante Byte des Werts codiert, auch wenn es sich um eine 16-Bit Wert handelt. Wenn das Codeelement mit dem Zeichen > beginnt, wird immer ein Wort codiert, auch wenn das niedrigste Byte 0 ist.

Beispiele:

```
inline(10/$2345/count+1/sort-*+2);

inline(<$1234/>$44); { erzeugt drei Bytes Code: $34, $44 und $00 }
```

Inline-Anweisungen können innerhalb des Anweisungsteils eines Blocks jederzeit mit anderen Anweisungen gemischt werden, und sie können alle Register der CPU benutzen. Beachten Sie, daß der Inhalt eines Stackzeiger-Registers (SP) am Ein- und Ausgang einer Inline- Routine der gleiche sein muß.

Interrupt-Handhabung

Das TURBO Pascal Laufzeit-Paket und der vom Compiler erzeugte Code sind beide voll interruptfähig. Interruptroutinen müssen alle benutzten Register zurückgeben. Falls benötigt, können Interruptroutinen in Pascal geschrieben werden. Solche Prozeduren sollten immer mit dem A Compilerbefehl (1$A +I) compiliert werden. Sie dürfen keine Parameter enthalten und müssen selbst sicherstellen, daß alle benutzten Register aufbewahrt werden. Dies geschieht, indem eine inline Anweisung mit den notwendigen PUSH-Instruktionen ganz zu Beginnn der Prozedur plaziert wird und eine weitere inline Anweisung mit den entsprechenden POP-Instruktionen ganz am Ende. Die letzte Instruktion der inline Anweisung sollte eine EI Instruktion ($FB) sein, um weitere Interrupts zu ermöglichen. Wenn daisy chained (verkettete) Interrupts benutzt werden, kann die inline Anweisung auch eine RETI Instruktion ($ED,S4D) spezifizieren, die die durch den Compiler erzeugte RET Instruktion überschreibt. Die generellen Regeln für den Registergebrauch sind, daß ganzzahlige Operationen nur die AF, BC, DE und HL Register verwenden können; andere Operationen können IX und IV und reelle Operationen wechselnde Register verwenden. Eine Interruptprozedur sollte keine I/O Operation mit Standardprozeduren oder -funktionen von TURBO Pascal benutzen, da diese Routinen nicht rückspringend sind. Dies gilt auch für BDOS-Aufrufe (und gelegentlich für BIOSAufrufe, abhängig von der spezifischen CP/M Implementation). Der Programmierer kann mit den Instruktionen DI oder EI einer inline-Anwe sung jederzeit im Programm Interrupts außer Kraft setzen oder erlauben. Wenn Modus 0 (IM 0) oder Modus 1 (IM 1) Interrupts benutzt werden, liegt es in der Verantwortlichkeit des Programmierers, die Restart-Stellen in der Basis- Seite zu initialisieren (Beachten Sie, daß RST 0 nicht verwendet werden kann, da CP/M die Stellen 0 bis 7 benutzt). Wenn Modus 2 (IM 2) Interrupts verwendet werden, sollte der Programmierer eine initialisierte Sprungtabelle (ein Array von integer Zahlen) an einer absoluten Adresse erzeugen und das Register I durch eine inline Anweisung am Anfang des Programms initialisieren.