Cyclone

Marcel Timmermans and german translation Tiemann

COLLABORATORS			
	TITLE : Cyclone		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Marcel Timmermans and german translation Tiemann	August 5, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Cyclone

1.1 Cyclone Modula-2 Compiler

```
                          Cyclone Modula-2 Compiler
  Ein Modula-2 Compiler für den Amiga
        von Marcel Timmermans


Einleitung

Systemvoraussetzungen

Installation

Cyclone~Erweiterungen~zu~Modula-2
                            Die Programme


Der~Compiler~(cmc)

Der~Linker~(cml)

Das~Make-Programm~(cmm)

Der~Fehlerlister~(cmerr)

Das~Cache-System~(cmcache)

Das~Projektprogramm~(cmpm)

Der~Library-Linker~(~CmLibLink)

Der~FD-Konvertierer~(FDTOM2
                            Verschiedenes

           Zu~erledigen
           Programmgeschichte

Fehlerreports

FAQ
```

1.2 CopyRight Informationen

COPYRIGHTS

Der Cyclone Modula-2 Compiler, die Programme
Cmc
,
Cml
,
Cmm
,

Cmerr
,
Cmpm
,
CmCache
,
CmLibLink
und
FdToM2
beinhaltent,
ist urheberrechtlich geschützt (Copyright © 1994-96) durch

Marcel~Timmermans
, alle Rechte vorbehalten.

Dieses Programm ist GIFTWARE, wenn Sie dieses Programm mögen, dann sollten Sie dem Autor ein Geschenk ("Gift") zukommen lassen, das Ihrer Meinung nach seinen Bemühungen um Cyclone entspricht.

Dieses Programm kann frei verteilt werden, solange alle Dateien ohne Modifikation zusammengehalten werden.

Dieses Programm darf *NICHT* auf einer BBS bereitgestellt werden, die urheberrechtliche Ansprüche auf das bereitgestellte Material macht.

Kommerzieller Vertrieb des Cyclone Modula-2 Compilers oder der beigefügten Dateien ist ohne schriftliche Erlaubnis des
Autors
NICHT
erlaubt.

1.3 Informationen über den Autor

Der Autor,

Meine Anschrift ist:

```
  Marcel Timmermans
  Aagje DekenStraat 22 (* kurz: A.Dekenstr 22 *)
  6836 RM Arnhem
  Holland / Europe.
```

Oder benutzen Sie meine E-Mail-Adresse:

```
  Internet: mtimmerm@worldaccess.nl
            http://www.worldaccess.nl/~mtimmerm
```

1.4 "

Der
Autor
kann nicht haftbar gemacht werden für die korrekte Funktionsweise der Anleitung und/oder der Programme, die sie beschreibt. Jeglicher direkt oder indirekt enstandene Schaden, der durch sachgemäße oder unsachgemäße Nutzung der Anleitung oder der Programme entsteht, liegt in der alleinigen Verantwort des Benutzers.

Diese Software darf nicht in Situationen eingesetzt werden, on denen diese Nichthaftbarmachung außer Kraft gesetzt würde.

1.5 "

Herzlichen Dank für Ihr Interesse an diesem Modula-2 Compiler.

Modula-2 ist eine höhere Programmiersprache, die von Prof. Dr. Niklaus Wirth von der ETH Zürich/Schweiz entwickelt wurde. Dieser Modula-2 Compiler wurde auf die speziellen Anforderungen des AmigaOS hin erweitert, so daß er ein ideales System zur Erstellung kleiner und großer Programme für den Amiga ist. Cmc ist ein objektorientierter single-pass Compiler, der in MC68000-Assembler programmiert wurde und MC68000-Code erzeugt. Objektdateien werden vom Compiler im Standard-AmigaDOS-Format erzeugt. Die Programme vom Cyclone Modula-2 Compiler, kurz Cyclone, können unter KS 1.2, 1.3, 2.x, 3.x und höher betrieben werden.

Einige Besonderheiten

- Schnell
- Ausnahmeverarbeitung (TRY EXCEPT/FINALLY END Blöcke)
- Inline-Assembler
- Objektorientierte Erweiterungen wie Vererbung, Methoden
- Inlining möglich
- Leichte Handhabung von Threads
- Standard Amiga-Objektdateiformat (blink kompatibel)

- Optimierender Linker
- Programme sind resident und reentrant
- Bedingte Compilierung
- Library-Linker
- Cache-System (um Symboldateien im Speicher zu halten, was die Compilation beschleunigt)
- Einfache Benutzung von tag-Argumenten mittels: [...,...]

1.6 Installation von Cyclone

Um die Installation so einfach wie möglich zu machen, habe ich ein Installationsskript für den AT-Installer geschrieben. Bitte benutzen Sie dieses zur Installation des Cyclone Modula-2 Compilers.

1.7 "

Obwohl Cyclone mittels Diskettenlaufwerken betrieben werden kann, ist eine Installation auf Festplatte vorzuziehen. Ansonsten wird ein Amiga-Computer mit mindestens 1MB Speicher benötigt, abhängig von der Größe des zu compilierenden Moduls. Cyclone wurde unter KS 2.x und 3.x getestet, sollte aber unter allen Kickstart-Versionen funktionieren.

Wenn Sie Probleme unter anderen Kickstart-Versionen haben, wenden Sie sich bitte an
Autor
.

1.8 Der Linker (cml)

Cml ist der zu Cyclone gehörige Linker. Normalerweise wird er Shell/CLI gesteuert benutzt, aber cml kann auch aus Skriptdateien von Texteditoren aus aufgerufen werden. Cml erwartet als Standarddateierweiterung
.obj
.

Cml benötigt mindestens 20000 Bytes Stack und kann resident gehalten werden, um das Linken zu beschleunigen.

Linker Ausgabe

Der Linker durchsucht die Haupt
objektdatei
und sucht Module, die vom Hauptmodul benötigt werden. Dies ist möglich, da im Hauptmodul Prozeduren im folgenden Format aufgeführt sind: <modulefilename>.<procedurename>

Der Linker sucht nur in Pfaden, die in der Datei 'am2:path' oder in einer lokalen datei 'Pfad' verzeichnet sind.

```
Beispiel:

>cml sparks

cml sparks Cyclone Modula-2 Linker V1.05e Beta, 12.11.95, (c) Marcel
Timmermans.

 - sparks.obj
 - am2:modules/obj/ModulaLib.obj
 - am2:modules/obj/ExecL.obj
 - am2:modules/obj/GraphicsL.obj
 - am2:modules/obj/IntuitionL.obj
 - am2:modules/obj/Random.obj
 - am2:modules/obj/MathIEEESingBas.obj
 + sparks
 CODE:3188 DATA:0 VAR:132 PROGRAM: 3224 >

Linker-Optionen

Der Linker versteht einige Kommandozeilenoptionen.  Diese können mit + (an)
und - (aus) in der Kommandozeile gesetzt werden. Diese Schalter können in
jeglicher Kombination erscheinen, dabei wird aber zwischen Groß- und
Kleinschreibung unterschieden, -d ist also nicht gleich -D!

  Linker Argumente
  ----------------------------

  Option   Standard  Beschreibung
  ------   --------  ------------------------------------------------------
   d        Aus (-)   Debug
```

1.9 Das Make-Programm (cmm)

```
Cmm ist das Make-Programm von Cyclone. Cmm kann Shell/CLI-gesteuert benutzt
werden oder aus einem Editorskript gestartet werden. Cmm erwartet als
Standarddateierweiterung .mod, das müssen Sie also nicht eintippen.

Auf Grund rekursiver Prozeduren braucht cmm 20000 Bytes Stack. Cmm kann
resident gehalten werden.

Stack setzten durch Eingabe >stack 20000

Cml resident machen durch Eingabe von >resident cml

Beispiel:

>Cmm sparks

Cyclone Modula-2 Make V1.00e, 04.03.95,(c) Marcel Timmermans.
 - sparks.mod
 - am2:modules/txt/Random.def
 - am2:modules/txt/Random.mod
 - am2:modules/txt/MathIEEESingBas.def
 - am2:modules/txt/IntuitionL.def
```

```
 - am2:modules/txt/Reg.def
 - am2:modules/txt/IntuitionD.def
 - am2:modules/txt/InputEvent.def
 - am2:modules/txt/UtilityD.def
 - am2:modules/txt/ExecD.def
 - am2:modules/txt/Timer.def
 - am2:modules/txt/GraphicsD.def
 - am2:modules/txt/Hardware.def
 - am2:modules/txt/GraphicsL.def
 - am2:modules/txt/ExecL.def
 + sparks.mtx Missing objectfile of sparks am2:cmc  sparks.mod  MC68000
Cyclone Modula-2 Compiler V0.45e, 05.03.95,(c) Marcel Timmermans.
 - sparks.mod
 - am2:modules/sym/ModulaLib.sym
 - am2:modules/sym/ExecL.sym
 - am2:modules/sym/GraphicsD.sym
 - am2:modules/sym/GraphicsL.sym
 - am2:modules/sym/IntuitionD.sym
 - am2:modules/sym/IntuitionL.sym
 - am2:modules/sym/Random.sym
 + sparks.obj

   Optimizing ...  54 Bytes optimized
   Symbol debug information available
   CODE: 1720 CONST: 0 BBS: 16

cml sparks Cyclone Modula-2 Linker V1.05e Beta, 12.11.95, (c) Marcel
Timmermans.

 - sparks.obj
 - am2:modules/obj/ModulaLib.obj
 - am2:modules/obj/ExecL.obj
 - am2:modules/obj/GraphicsL.obj
 - am2:modules/obj/IntuitionL.obj
 - am2:modules/obj/Random.obj
 - am2:modules/obj/MathIEEESingBas.obj
 + sparks
 CODE:3188 DATA:0 VAR:132 PROGRAM: 3224


Cmm produziert eine Abhängigkeitsdatei namens <dateiname>.mtx. Dadurch
müssen nicht alle abhängigen Dateien immer wieder durchsucht werden.
Achtung! Manchmal ist dies nicht wünschenswert, wenn z.B. noch nicht alle
Module importiert wurden. Daher kann die Benutzung der Abhängigkeits- datei
durch die Option '-d' vermieden werden. Nach Löschen von sparks.obj sieht
der nächste Versuch so aus:

Cyclone Modula-2 Make V1.00e, 04.03.95,(c) Marcel Timmermans.
 - sparks.mtx Missing objectfile of sparks am2:cmc  sparks.mod  MC68000
Cyclone Modula-2 Compiler V0.45e, 05.03.95,(c) Marcel Timmermans.
 - sparks.mod
 - am2:modules/sym/ModulaLib.sym
 - am2:modules/sym/ExecL.sym
 - am2:modules/sym/GraphicsD.sym
 - am2:modules/sym/GraphicsL.sym
 - am2:modules/sym/IntuitionD.sym
 - am2:modules/sym/IntuitionL.sym
```

```
 - am2:modules/sym/Random.sym
 + sparks.obj

   Optimizing ...  54 Bytes optimized
   Symbol debug information available
   CODE: 1720 CONST: 0 BBS: 16 cml sparks Cyclone Modula-2 Linker V1.05e
Beta, 12.11.95, (c) Marcel Timmermans.
 - sparks.obj
 - am2:modules/obj/ModulaLib.obj
 - am2:modules/obj/ExecL.obj
 - am2:modules/obj/GraphicsL.obj
 - am2:modules/obj/IntuitionL.obj
 - am2:modules/obj/Random.obj
 - am2:modules/obj/MathIEEESingBas.obj
 + sparks
 CODE:3188 DATA:0 VAR:132 PROGRAM: 3224
```

Cmm Optionen

Cmm versteht einige Kommandozeilenoptionen. Diese können mit + (an) und - (aus) in der Kommandozeile gesetzt werden. Diese Schalter können in jeglicher Kombination erscheinen, dabei wird aber zwischen Groß- und Kleinschreibung unterschieden, -v ist also nicht gleich -V!

Cmm Argumente

Option	Standard	Beschreibung
x	An(+)	suche extern. Cmm wird in allen Pfaden suchen, die in der 'path' Datei enthalten sind.
d	An(+)	benutze Abhängigkeitsdatei
a	Aus(-)	compiliere alle gefundenen Module.
b	An(+)	starte den Linker
v	An(+)	ausführliche Ausgaben
c	Aus	Hiernach können die Compiler-Optionen gesetzt werden. z.B. Cmm +c[compiler options]
l	Aus	Hiernach können die Linker-Optionen gesetzt werden. z.B. Cmm +l[linker options]

Mögliche Probleme

- Make hielt an, Fehler passiert?
  Möglich wenn der Linker oder Compiler nicht gestartet werden konnte.

- Make erstellte keine korrekte Abhängigkeitsdatei?
  Stellen Sie sicher, daß die Uhr richtig geht. Stellen Sie zwischendurch keine andere Zeit ein.
  Sollte Ihr Amiga über keine eingebaute Uhr verfügen, dann stellen Sie die Zeit bitte während des Bootens richtig ein.

Läuft Ihre Uhr instabil, dann beseitigen Sie dies Problem mit dem Programm DateInspector.

Cmm kann durch mehrfach vorhandene Kopien eines Moduls verwirrt werden.

1.10 Der Fehlerlister (cmerr)

Cmerr ist der Fehlerlister von Cyclone. Starten Sie cmerr Shell/CLI-gesteuert oder aus einem Editorskript heraus, Cmerr erwartet als Standarderweiterung .mod. Um die Fehler eines Definitionsmoduls aufzulisten, benutzen Sie die Endung .def.

Was tut cmerr?

Cmerr listet die vom Compiler berichteten Fehler auf. Sofern der Compiler einen Fehler findet, erstellt er eine .defE oder modE datei, abhängig von der compilierten Quelldatei. Diese Fehlerdatei enthält die Fehlernummer und Position des Fehlers.

Fehlerformat

Dateiposition - 4 Bytes Fehlernummer - 2 bytes . . . <Endheader> - LONGINT(-1)

Kommandozeilenoption

Cmerr hat eine Kommandozeilenoption, damit kann man die zu einer Fehlernummer zugehörige Zeichenkette ermitteln.

Beispiel:

```
>cmerr +n100

Cyclone Modula-2 Errorlister V1.00e, 22.03.95,(c) Marcel Timmermans.

operand type incompatible with +

>
```

Wie benutzt man cmerr?

Geben Sie einfach ein: cmerr <errorfile>[.def]

```
Example >cmerr errtest Cyclone Modula-2 Errorlister V1.00e, 22.03.95,(c)
Marcel Timmermans.
 - errtest.mod
 - errtest.modE -----
   6: j:=1;
      ^
      61: undefined identifier

>
```

6 markiert die Zeile und 61 ist die fehlernummer. Alle Fehler sind in

normalen ASCII-Text in der Datei am2:cmc_errors aufgeführt.

1.11 Das Cache-System (cmcache)

Mit Cmcache können Sie das Cache-System starten und kontrollieren. ↩
Nach dem
Start von Cmcache wird eine Semaphore erzeugt, diese bleibt aktiv bis sie
wiederum mit Cmcache beseitigt wird.

Was ist das nun genau?

Nun, normalerweise werden von einem Modul benötigte
Symboldateien
oder

Objektdateien
immer wieder von einem Laufwerk geladen. Zur Beschleunigung
können Symbol- und Objektdateien aber im Speicher behalten werden. Sobald
nun eine Objektdatei benötigt wird, wird sie aus dem Speicher statt von
einem Laufwerk geholt, was die Leistung deutlich erhöht.

Derzeit macht nur cmc davon Gebrauch. Wenn der Compiler eine Symboldatei
benutzt, dann zeigt er dies durch ein '=' statt eines '-' an.

Selbstredend braucht dies Speicher. Deshalb kann über eine Option der
Speicherverbrauch von Cmcache eingestellt werden, der Standard beträgt
300000 Bytes. Momentan ist das Cache-Verhalten noch nicht optimal, alte
selten benutzte Dateien werden nicht durch neue ersetzt, wenn der
Cache-Speicher voll ist. Wenn eine Symboldatei einmal geladen ist, dann kann
sie nur durch komplette Leerung des Caches freigegeben werden!

Warnung:

Wenn der Compiler eine neue Objekt- oder Symboldatei erstellt, dann wird die
alte Version gegebenenfalls aus dem Cache entfernt. Der komplette Pfad
einer Datei wird im Speicher gehalten. Trotzdem werden Dateien aus
verschiedenen Verzeichnissen gleich behandelt. So wird eine Datei namens
dh1:work/files.sym und eine Datei am2:modules/files.sym aus dem Cache
entfernt, wenn der Compiler eine neue Symboldatei erzeugt: files.sym.

Wie startet man cmcache?

Tippen Sie einfach cmcache und Enter und das cache-System wird installiert.

Beispiel

```
>cmcache

  Cyclone Modula Cache Utility V1.00e,
  04.03.95, (c) Marcel Timmermans.
  Installed CmCache!  >
```

Was passiert bei nochmaliger Installation?

Beispiel

```
>Cmcache
  Cyclone Modula Cache Utility V1.00e,
  04.03.95, (c) Marcel Timmermans.
  CmCache is already installed
  300000 bytes maximum memory
  0 bytes used memory  >
```

Cmcache Optionen

Einige Kommandozeilenoptionen können benutzt werden. Passen Sie auf, da die Optionen anders gehandhabt werden als in den restlichen Programmen des Cylone-Systems!

Cmcache Argumente

Option	Beschreibung
l	Liste alle Objekt-/Symboldateien, die im Cache vorhanden sind
f	Entferne alle Dateien aus dem Cache
q	Entferne das Cache-System und alle darin vorhandenen Dateien
u	Setze die maximal von cmcache benutzte Speichermenge Beispiel Cmcache u500000 Das cache-System belegt nun maximal 500000 Bytes Speicher.

1.12 Das Projektprogramm

Cmpm dient zum Erstellen von Projekten in Cyclone. Normalerweise wird Cmpm von der Shell gestartet.

Um eine geordnete Struktur der Festplatte während der Programmentwicklung zu behalten, kann man ein Projektverzeichnis, z.B. 'compiler', erstellen. Mit Cmpm kann man dies Verzeichnis mit den nötigen Unterverzeichnissen anlegen.

Beispiel

```
>cmpm compiler Cyclone Modula-2 Projectmaker, 1.00, 04.03.95, © Marcel
Timmermans
 + (compiler)
 + (compiler/bin)
 + (compiler/obj)
 + (compiler/sym)
 + (compiler/ref)
 + (compiler/txt)
```

Cmpm hat nun das Verzeichnis compiler und die Unterverzeichnisse darin erstellt:

- bin
- obj

```
- sym
- ref
- txt

So werden alle vom Compiler oder Linker erstellten Dateien in die
richtigen Verzeichnisse einsortiert.

sym  <- beinhaltet die Symboldateien (.sym)
ref  <- beinhaltet die Referenzdateien für den Quell-Code-Debugger
          (in Arbeit)
txt  <- hier können die Quelldateien aufbewahrt werden ( .def | .mod )
obj  <- beinhaltet die Objektdateien (.obj)
bin  <- die Binärdateien enden hier (keine Erweiterung)
```

Cmpm Optionen

Cmpm versteht einige Kommandozeilenoptionen. Diese können über + (an) oder - (aus) in der Kommandozeile gesetzt werden. Die Optionen können in jeder Reihenfolge gesetzt werden, es wird aber nicht zwischen Groß- und Kleinschreibung unterschieden, -b bedeutet also daselbe wie -B!.

Cmpm Argumente

Option	Standard	Beschreibung
b	An (+)	Erzeuge ein bin Verzeichnis
o	An (+)	Erzeuge ein obj Verzeichnis
r	An (+)	Erzeuge ein ref Verzeichnis
s	An (+)	Erzeuge ein sym Verzeichnis
t	An (+)	Erzeuge ein txt Verzeichnis
m	Aus(-)	Scheibe die Dateien aus dem aktuellen Verzeichnis in die korrekten Unterverzeichnisse im Projektverzeichnis.

1.13 Ausnahmeverarbeitung (Exception-Handling)

Die Ausnahmeverarbeitung hat nicht mit Prozessorausnahmen zu tun!

Ein Ausnahme-Handler (Exception-Handler) ist eine Prozedur, die aufgerufen wird, wenn ein Laufzeitfehler auftritt, z.B. bei Speichermangel.

Der Block zwischen TRY und EXCEPT wird wie normal ausgeführt und wenn keine Ausnahme auftritt, wird der Block zwischen EXCEPT und END übersprungen. Wenn eine Ausnahme aufgerufen wird, entweder im TRY Teil oder in irgendeiner Prozedur, die von dort aus aufgerufen wird, wird der Ausnahme- Handler aufgerufen.

Eine Ausnahme kann durch die Prozedur Raise erzeugt werden, die von ModulaLib importiert werden kann. Als Argument erwartet Raise eine Ausnahmefehlernummer. Diese Fehlernummer kann im Ausnahme-Handler abgefragt werden. Die Fehlernummer wird in der Variablen ExceptNr aufbewahrt, welche von ModulaLib importiert werden kann.

z.B.

```
 MODULE Exception;

 FROM ModulaLib IMPORT ExceptNr,Raise;
 FROM SYSTEM    IMPORT ADDRESS,ADR;
 FROM ExecL     IMPORT AllocMem;
 FROM ExecD     IMPORT MemReqSet,MemReqs;
 FROM InOut     IMPORT WriteString;

 CONST
  NOMEM=1;


 PROCEDURE TrySome;
 VAR
  p:ADDRESS;
 BEGIN
  TRY
   p:=AllocMem(100000,MemReqSet{public});
   IF p=NIL THEN Raise(NOMEM); END;
  EXCEPT
   CASE ExceptNr DO
    | 1: WriteString('Sorry too little memory!\n");
   ELSE
    Raise(ExceptNr);
   END;
  END
 END TrySome;

 BEGIN
  TrySome;
 CLOSE
 (* Dies ist der Standard Ausnahme-Handler. Hier darf (noch) nicht Raise()
  * aufgerufen werde, die ergäbe eine Endlosschleife!!
  *)
 END Exception;

 Aber es gibt noch mehr! Wenn die EXCEPT Anweisung durch ein FINALLY
 ersetzt wird, dann wird er Block zwischen FINALLY und END auf jeden
 Fall abgearbeitet!!

 Die Ausnahmeverarbeitung ist Stack-basiert. Wenn man also Raise() in einem
 EXCEPT END Block anwendet, dann wird der vorherige Ausnahme-Handler
 aufgerufen. Sofern keiner mehr vorhanden ist, wird CLOSE aufgerufen.
 Ist überhaupt kein Exception-Handler vorhanden, dann hat Raise(x) die
 gleiche Wirkung wie Exit(0).
```

1.14 Cyclones FAQ

```
Cyclone FAQ

Diese FAQ-Liste (Frequently Asked Questions, häufig gestellte Fragen) wurde
angefertigt, nachdem ich einige Fragen mehrmals gestellt bekam.
```

F. Manchmal bekomme ich beim Linken die Fehlermeldung:
   Symbol [module].somenumber not found?!

A. Meistens wurden die Module nicht in der richtigen Reihenfolge
   compiliert. Ein Make-Durchlauf kann hier helfen!
   Es kann auch an einer fehlerhaften Uhr liegen. Man besorge sich
   DateInspector. Schlimmenstfalls lösche man alle #?.obj Dateien,
   leere den cache und starte make (wodurch alles neu compiliert wird).

F. Kann Cyclone-Objectcode mit anderen Sprachen gelinkt werden?

A. Ja! Cyclone benutzt das Standard-Amiga-Objektformat! Sie müssen nur
   die Aufrufkonventionen beachten, da Cyclone hier anders als
   beispielsweise C arbeitet.
    F. Kann ich amiga-lib hinzulinken?

A. Momentan unterstützt der Linker keine externen Objektdateien. Dies wird
   bald implementiert.

F. Program X wurde mit C und Cyclone geschrieben, und die Cyclone-Version
   ist viel schneller/langsamer.

A. Es ist nicht leicht, beide fair zu vergleichen. Manchmal hat ein
   Compiler spezielle optimierte Routinen (Laufzeitprüfung, Stringverar-
   beitung...). Aber zum Erreichen maximaler Geschwindigkeit kann der
   eingebaute Assembler benutzt werden.

Q. Was ist das Problem mit dem Fehler Key mismatch?!

A. Bei jeder Compilation legt der Compiler einen sogenannten Modulschlüssel
   an. Dieser Schlüssel ist einmalig und wird benutzt, um zwischen
   verschiedenen Versionen des gleichen Moduls zu unterscheiden. Dieser
   Schlüssel existiert für die Symbol- und die Objektdatei. Für ein
   Implementationsmodul wird der Schlüssel des zugehörigen Definitions-
   moduls übernommen. Die Schlüssel importierter Module werden außerdem
   in erzeugten Symbol- und Objektdateien gespeichert. Jede Nichtüber-
   einstimmung der Schlüssel eines importierten Moduls führt so zu
   einer Fehlermeldung beim Compilieren oder Linken. Während des Linkens
   bekommt man eine Fehlernummer wie "cannot find symbol module.somenumber"

   Zur korrekten Funktion wird eine stabile Uhr benötigt!!

1.15 CmLibLinker

CmLiblink (Cyclone's Library linker)

Mit Hilfe dieses Programms kann eine residente Library (Bibliothek) erstellt
werdem, so wie sie beispielsweise in libs: zu finden sind.

Um eine solche Library zu erstellen, benötigt man einige Informationen über
die Arbeitsweise einer Library. Diese entnimmt man den RKM (libraries)
manuals.

Im Verzeichnis am2:liblinker/demolib ist ein entsprechendes Beispiel zu finden!

CmLiblink Optionen

Die folgenden Optionen können mit CmLibLink benutzt werden:

CmLibLink Argumente

Option	Standard	Beschreibung
d	Aus(+)	Debugcode
r	0	Revision
v	0	Version
s	SIZEOF(Library)	Library-Größe
l	none	eine Datei, die die exportierten Prozeduren vermerkt

Format einer Prozedurliste-Datei (wird mit Option +l gebraucht)

MODULENAME.PROCDURENAME

Alle exportierten Prozeduren können in die Liste separat mit einem Return aufgenommen werden. Kein Argumente in der Liste verwenden!

z.B.

Um PROCEDURE Times(x,y:INTEGER) aus dem Modul sample zu exportieren:

sample.Times

Normalerweise würde ein Linker ModulaLib.obj einbinden. Für eine Library gibt es aber eine spezielle Version von ModulaLib. Diese steht in: am2:liblinker/modulalib/

Nach Modifikationen muß diese neu compiliert werden und in am2:modules/obj/ kopiert werden.

Diese Seite ist noch in Bearbeitung!! :-)

1.16 FdToM2

FDToM2

Dieses kleine Programm wandelt FD-Dateien in Modula-2 Definitions-/ Implemenationsdateien um.

FDToM2 Argumente

```
  ------------------------

  Option   Standard  Beschreibung
  ------   ------- ------------------------------------------------------
   i        Aus(-)   erzeuge ein Implementationsmodul
   m        0        minimale Library-Version
   t        fdname   der Name dieses Moduls
   l        none     der Bibliotheksname

z.B.

 fdtom2 -i +lamigaguide.library +m40 +tAmigaguide amigaguide_lib.fd

 Dies erzeugt ein Definitionsfile namens:
  AmigaGuideL.def

Sofern eine Definitionsdatei ohne Implementationsdatei erzeugt wird, wird
'L' and den Dateinamen angehängt!
```

1.17 Der Compiler (cmc)

```
Starten~von~cmc

Kommandozeilenoptionen

Einfache~Typen

Schlüsselwörter

Interne~Prozeduren

Erweiterungen~von~Cyclone

Der~Inline-Assembler

Erste~Programmerstellung

Wichtig~zu~wissen

Modul~SYSTEM

Modul~ModulaLib~(The~Runtime~System)

Modul~NoGuru

Modul~Break~(Control-C)

Compiler~Direktiven

Automatische~Optimierungen

Compiler~Limitierungen

Fehler~während~des~Compilierens
```

Laufzeitfehler

1.18 cmc von der Shell starten

Gewöhnlich wird cmc von der Shell aus gestartet, kann aber ↩
natürlich auch
aus einem Skript aus einem Editor heraus aufgerufen werden. Cmc ist
reentrant, mit "resident am2:amc" kann es resident gehalten werden.

Format: cmc [Optionen] <Quelldatei>

Wenn cmc ohne Argumente gestartet wird, dann fährt der Compiler im
interaktiven Modus hoch. In diesem Modus können Optionen oder ein Dateiname
übergeben werden. Ein '?' wird eine Kurzbeschreibung anzeigen. Für eine
detaillierte Beschreibung aller Optionen siehe
Kommandozeilenoptionen
.

Cmc kann jederzeit durch Drücken von CTRL-C gestoppt werden.

1.19 Kommandozeilenoptionen

Kommandozeilenoptionen

In cmc können verschiedene Optionen gesetzt werden. Dies geht mit + (an) und
- (aus) in der Kommandozeile. Einige Kommandos können auch im Quell- text
gesetzt werden, siehe
compiler

directives
für mehr Informationen. Die
Optionen können in jeder Reihenfolge übergeben werden, jedoch wird zwischen
Groß- und Kleinschreibung unterschieden, -g ist also nicht gleich -G!

Compiler Argumente

Option	Standard	Beschreibung
g	An (+)	Optimierer.
n	An (+)	Füge Code zur Nil-Überprüfung ein.
r	An (+)	Füge Code zur Bereichsprüfung ein (Range).
o	An (+)	Füge Code zur Überlaufsprüfung ein (Overflow).
s	An (+)	Füge Code zur Überprüfung eines Stack-Überlaufs ein.
f	An (+)	Füge Code zur Rückgabeüberprüfung ein (Return).
c	An (+)	Füge Code zur Fallüberprüfung ein (Case).
d	An (+)	Füge symbolische Debuginformatione ein.
v	An (+)	Gib mehr Information aus (verbose).
l	An (+)	Setzt lokale Variablen auf "Null" (EntryClear)
a	An (+)	Ausrichtung aller Variablen auf Langwörter (Aligning).

```
t        Aus(-)   Test Option.
z        -        Hier kann die Variable für bedingte Compilierung
                  gesetzt werden,. z.B. (+zEnglish)
1        Aus      Erzeuge Code für 68010 Prozessor
2        Aus      Erzeuge Code für 68020 Prozessor
3        Aus      Erzeuge Code für 68030 Prozessor
4        Aus      Erzeuge Code für 68040 Prozessor
```

Zur Coderezeugung für verschiedene Prozessoren kann immer nur eine der Optionen gesetzt sein. Bei Mehrfachsetzung ist nur die letzte aktiv. Es existiert kein Unterschied zwischen -2 und +2.

Es sollte kein Überprüfungscode in Threads, Libraries oder Hooks benutzt werden.

1.20 Einfache Typen

```
 Datentypen und ihre Darstellung

 CHAR           1 Byte
 BOOLEAN        1 Byte
 SHORTINT       1 Byte Zweierkomplement
 SHORTCARD      1 Byte
 INTEGER        2 Bytes Zweierkomplement
 CARDINAL       2 Bytes
 LONGCARD       4 Bytes
 LONGINT        4 Bytes Zweierkomplement

 REAL           4 Bytes
 LONGREAL       8 Bytes

 SHORTSET       1 Byte  (0..7)
 BITSET         2 Bytes (0..15)
 LONGSET        4 Bytes (0..31)

 POINTERS       4 Bytes
 PROCEDUREs     4 Bytes
```

Anmerkung: Der Typ REAL benutzt die mathieeesingbas.library und mathieeesingtrans.library, die beide erst ab AmigaOS V2.0 oder höher zur Verfügung stehen. Sollten diese auch mit älteren AmigaOS-Versionen benötigt werde, dann werde ich möglicherweise Ersatz dafür schreiben. Der Typ LONGREAL benutzt die mathieeedoubbas.library und die mathieeedoubtrans.library, welche bereits ab AmigaOS V1.2 zur Verfügung stehen. Der Typ BOOLEAN ist nur 1 Byte groß. Dies ist nicht in allen Programmiersprachen so! FALSE ist gleich Null, alles andere ist TRUE!!!

1.21 Schlüsselwörter

```
                Standard Schlüsselwörter

Die folgenden Schlüsselwörter werden vom Compiler benutzt und dürfen NICHT
für Variablen, Typen, Prozedeuren etc. verwendet werden.

BY, DO, IF, IN, OF, OR, TO, AND, DIV, END, FOR, MOD, NOT, SET, VAR, CASE,
CODE, ELSE, EXIT, FROM, LOOP, THEN, TYPE, WITH, ARRAY, BEGIN, CONST, ELSIF,
UNTIL, WHILE, EXPORT, IMPORT, MODULE,
                CLOSE
                , RECORD, REPEAT, RETURN,
FORWARD, BPOINTER, POINTER, PROCEDURE, QUALIFIED, DEFINITION,
                INLINE
                IMPLEMENTATION,
                COMPDATE,~COMPTIME
                ,
                CONSTRUCTOR,~DESTRUCTOR,~CLASS
                ,
                TRY
                ,

                EXCEPT
                ,
                FINALLY
                ,
                IGNORE
                Beachten Sie, daß Cyclone Groß-/Kleinschreibung unterscheidet,  ↩
                   diese
Schlüsselwörter müssen groß geschrieben werden. Wenn Sie einen entsprech-
enden Editor haben (z.B. GoldED), dann können Sie AutoCase anstellen, um die
korrekte Schreibweise zu gewährleisten. Außerdem wird der Quellcode dadurch
deutlich besser lesbar.
```

1.22 Interne Prozeduren

```
Interne Prozeduren des Compilers

 PROCEDURE ABS(x):LONGINT;
 PROCEDURE CAP(ch:CHAR):CHAR;
 PROCEDURE CHR(ch:CHAR):CHAR;
 PROCEDURE DEC(x:ScalarTyp{;d:ScalarTyp}):ScalarTyp;
 PROCEDURE EXCL(x:SetTyp;o:ElemSetTyp):SetTyp;
 PROCEDURE FLOAT(x:ScalarTyp):REAL;
 PROCEDURE HALT;
 PROCEDURE INC(x:ScalarTyp{;d:ScalarTyp}):ScalarTyp;
 PROCEDURE INCL(x:SetTyp;o:ElemSetTyp):SetTyp;
 PROCEDURE ODD(x:ScalarTyp):ScalarTyp;
 PROCEDURE HIGH(s:ARRAY OF CHAR):LONGINT;
 PROCEDURE MAX(x:ScalarTyp):LONGINT;
 PROCEDURE MIN(x:ScalarTyp):LONGINT;
 PROCEDURE ORD(x:ScalarTyp):LONGINT;
 PROCEDURE SIZE(x:AnyType):LONGINT;
 PROCEDURE TRUNC(r:RealTyp):LONGINT;
 PROCEDURE VAL(NewAnytyp,OldAnytyp):NewAnyTyp;
```

```
PROCEDURE NEW(classtyp)
PROCEDURE DISPOSE(classtyp)

Neben den normalen Modula-2 Prozeduren gibt es zwei weitere,
NEW und DISPOSE. Diese werden für objektorientierte Programmierung
gebraucht.



NEW(classtyp)
-------------

Wenn ein neues Objekt  deklariert wurde, dann muß es initialisiert werden
und neuer Speicher wird benötigt.

z.B. NEW(objectvar);

Besorgt den Speicher für das Objekt, initialisiert den Typdeskriptor und
ruft ggf. den Konstruktor auf.

Anmerkung: Der Speicher für das Objekt wird bei Programmende automatisch
fregegeben!!


DEALLOCATE(classtyp)
--------------------

Diese Prozedur gibt den Speicher des Objekts wieder frei.
```

1.23 OO-programmieren

```
Die objektorientierte Zukunft

Cmc hat ein paar extra Konstrukte, um die Sprache objektorientiert zu
machen. Der Objekttyp CLASS ist ein Zeiger auf ein Objekt.

Ein Objekt kann wie im folgenden definiert werden.

TYPE
 TList = CLASS(TObject)
           prev,next:TList;
           CONSTRUCTOR Init;
           PROCEDURE Add;
           DESTRUCTOR Done;
          END

CONSTRUCTOR TList.Init BEGIN (* Weiterer Code *) END TList.Init

PROCEDURE TList.Add BEGIN (* Weiterer Code *) END TList.Add

DESTRUCTOR TList.Done BEGIN (* Weiterer Code *) END TList.Done

VAR
 List:TList;
```

```
BEGIN
 NEW(List);
 List^.Add;
 WITH TList^  (*** NOCH NICHT MÖGLICH
  Add;
 END;          **********************)
 DISPOSE(List); END x.
```

Der Konstruktor wird bei Initialisierung eines neuen Objekts durch NEW wie oben aufgerufen. Danch kann jede andere Prozedur wie oben aufgerufen werden. Schließen (disposing) eine Objekts geschieht mit der Prozedur DISPOSE.

Im Moment bietet der OO-Teil wenig Möglichkeiten zum "Verstecken" der Datenimplementation, und die WITH Anweisung kann mit Klassen noch nicht genutzt werden.

Natürlich sind (virtuelle) Methoden, Vererbung und Polymorphie nutzbar.

Beispiele sind in den verschiedenen Modulen im Verzeichnis OOModules zu finden und im Thread-Beispiel im Demo-Verzeichnis.

1.24 Laufzeitfehler

Laufzeitfehler

Wenn das Modul
NoGuru
importiert wird, dann wird ein Trap-Handler installiert, um Laufzeitfehler abzufangen und durch Meldungen anzuzeigen.

Der Compiler erzeugt in einigen Situationen speziellen Code um verschiedene Fehlerursachen wie NIL-Zeiger, Bereichsüberschreitung etc. zu entdecken.

Sofern NoGuru nicht importiert wird, wird ein Laufzeitfehler in einem Guru enden.

Die folgenden Laufzeitfehler können auftreten:

- Bus error (68000 Exception) (Busfehler)
- Address error (68000 Exception) (Adressfehler)
- Illegal instruction (68000 Exception) (Illegale Instruktion)
- Zero divide (68000 Exception) (Division durch Null)
- Rangecheck error (CHK) (68000 Exception) (Bereichsüberprüfung)
- Overflow error (TPAPV) (68000 Exception) (Überlauf)
- Privilege error (68000 Exception) (Privilegeverletzung)
- Trace (68000 Exception)
- Line 1010 emulator (68000 Exception)
- Line 1111 emulator (68000 Exception)
- illegal CASE-index (Trap 0) (Unerlaubter CASE-Index)
- Pointer is NIL (Trap 1) (Zeiger ist NIL)
- Overflow (Trap 2) (Überlauf)
- Stack Overflow! (Trap 3) (Stack-Überlauf)
- Return Failure! (Trap 4) (Rückgabefehler)

```
 - Range error! (Trap 5) (Bereichsfehler)

NoGuru wird in Libraries, Hooks oder Programmen mit mehreren Tasks/Threads
wenig Wirkung zeigen.
```

1.25 Compiler-Beschränkungen

```
 Compiler-Beschränkungen

Der Compiler enthält einige Beschränkungen. Einige sind Hardware-bedingt,
andere durch die Machart des Compilers.
 - Die Codegröße eines Moduls ist auf 32000 Bytes beschränkt (code+
   data+bbs).
 - Die Länge einer Zeichenkette ist auf 1000 Zeichen beschränkt. Mit
   Concatenate kann man Zeichenkette bis zu 32000 Zeichen erzeugen.
   s='this is a test'+' with a concatenate string'
 - Der Modulname darf maximal 30 Zeichen lang sein.
 - De Anzahl von EXIT-Ebenen in LOOP ist maximal 16.
 - Die tiefste Verschachtelung von LOOP ist 4.
 - Die tiefste Verschachtelung von WITH ist 3.
 - Die maximale Anzahl globaler Module ist 64.
 - Die maximale Größe lokaler Variablen ist 32KB
 - Eine Menge (Set) kann maximal 32 Element beinhalten.
 - Eine CASE-Anweisung kann maximal 127 Fälle unterscheiden

Diese Limitierungen werden Sie wahrscheinlich nicht erreichen. Falls doch,
dann sollten Sie Ihren Programmierstil auf Strukturiertheit hin überprüfen.


Anmwerkung: Dei Codegröße eines Moduls ist auf 32000 Byte beschränkt. Eine
komplette Applikation kann selbstredend größer sein. Der Linker erzeugt
automatisch sogenannte ALV's (Auto Link Vectors).
```

1.26 Die Laufzeit-Library (Bibliothek) für Modula-2 (ModulaLib)

```
 Die Modula-2 Library (ModulaLib)

Diese Library enthält den Start- und Abschlußcode für erzeugte ausführbare
Programme. Desweiteren enthält sie Prozeduren zur Laufzeitunterstützung, wie
mulu, muls, div, etc., die vom Compiler und Unterstützungscode benötigt
werden. Der Quellcode von ModulaLib liegt vor, seien Sie aber bei Änderungen
an diesem Modul sehr vorsichtig. Der Compiler erwartet einige Dinge in
dieser Library, unbedachte Änderungen können dazu führen, daß das Modul
nicht mehr compiliert werden kann.


 ModulaLib Variablen
 --------------

 wbStarted ist eine BOOLEAN und gibt an, ob das Program von der
 Workbench (true) oder nicht (false) gestartet wurde.
```

```
 thisTask  zeigt auf den Haupt-Task des Programms.

 kickVersion beinhaltet die Versionsnummer der Exec-Library.

 returnVal kann auf einen bestimmten Rückgabewert gesetzt werden.
 Normalerweise geschieht dies mit ModulaLib.Exit(returncode).

 dosCmdBuf enthält den Argumentenpuffer (A0 nach dem Start).

 dosCmdLen enthält die Länge des Arguments (D0 nach dem Start)

 wbenchMsg enthält die Workbench-Startup-Message.

 ExceptNr enthält die letzte Ausnahmenummer, so wie sie mit
 Raise(nr) gesetzt wurde.

 ExceptStck, saveA7 and saveA5 werden für den Exception-Handler
 benötigt. Nicht manipulieren!


 ModulaLib Prozeduren
 --------------------

(******* Private procedures; DO NOT USE *********)
 PROCEDURE easystartup;
 PROCEDURE StackChk(space{0}:LONGINT);
 PROCEDURE LoadA4;
 PROCEDURE Mulu32(x{0},y{1}:LONGINT):LONGINT;
 PROCEDURE Muls32(x{0},y{1}:LONGINT):LONGINT;
 PROCEDURE Divu32(x{0},y{1}:LONGINT):LONGINT;
 PROCEDURE Divs32(x{0},y{1}:LONGINT):LONGINT;
 PROCEDURE ModDiv(x{0},y{1}:LONGINT):LONGINT;
 PROCEDURE New(VAR adr:ADDRESS;size:LONGINT);
 PROCEDURE Dispose(VAR adr:ADDRESS);
 PROCEDURE SFix(x{0}:REAL):LONGINT;
 PROCEDURE StoredA4;

 Die oben aufgeführten Prozeduren sind privat und sollten nicht aus dem
 Definitionsfile entfernt werden.


(* Runtime support procedures *)

 PROCEDURE Assert(cc: BOOLEAN; Msg:ADDRESS);

 Diese Prozedur testet einen Ausdruck ("cc") und wird davon abhängig
 einen Requester zur Terminierung des Programms anbieten. Der Benutzer
 hat die Wahl, das Programm entweder zu terminieren oder weiterlaufen zu
 lassen.

 z.B. Assert(FALSE,ADR("some text")); Ruft den Requester mit der Meldung
 'some text' auf.

 PROCEDURE TerminateRequester(Msg:ADDRESS);

 Öffnet einen Requester mit bestimmbarer Meldung ("Msg") und terminiert das
```

```
Programm.

PROCEDURE Terminate;

Terminiert das Programm.

PROCEDURE TermOpenLib(Msg{9}:ADDRESS);

Öffnet einen Requester mit der Meldung 'Error opening library '+Msg und
terminiert das Programm.

PROCEDURE Exit(returnCode{0}:LONGINT);

Beendet das Programm mit einem Rückgabewert.

PROCEDURE Halt;

Ist gleichbedeutend mit Assert(FALSE,ADR("HALT"));

PROCEDURE Raise(i{0}:LONGINT);

Diese Prozedur wird benötigt, um eine Ausnahme zu erzeugen.
```

1.27 NoGuru

```
                Dieses Modul stellt einen Trap-Handler für Laufzeitfehler zur ↩
                   Verfügung.
Bei Import von NoGuru wird dieser automatisch installiert. Im Falle eines
Laufzeitfehlers wird dieser so abgefangen und sauber abgearbeitet, d.h.
ein Requester mit entsprechender Fehlermeldung erscheint und das Programm
terminiert sauber. Natürlich kann auch ein eigener Traphandler geschrieben
werden.
               ModulaLib
                wird bei Programmende immer den
von Ihnen installierten durch den voreingestellten ersetzten.

In einem Programm mit mehreren Task/Threads zeigt NoGuru nur beim
Haupt-Task Wirkung.
```

1.28 Fehlermeldungen

(Die deutschen Fehlermeldungen liegen gesondert vor, bei Unstimmigkeiten können Sie sie hier mit den englischen Originalen vergleichen)

Fehl.	Beschreibung
1	syntax error
2	colon expected
3	semi-colon expected
4	comma expected
5	missing right parenthesis

```
 6 | missing left parenthesis
 7 | missing right bracket
 8 | missing left bracket
 9 | missing right brace
10 | missing left brace
11 | equal sign expected
12 | assignment expected
13 | ellipsis expected
14 | "*)" expected
15 | TO expected
16 | DO expected
17 | UNTIL expected
18 | THEN expected
19 | OF expected
20 | period expected
21 | END expected
22 | statement start with illegal symbol
23 | BEGIN or END expected
24 | statement part is not allowed in DEFINITION MODULE
25 | -
26 | illegal use of code procedure
27 | illegal use of forward procedure
28 | identifiers in heading and end do not match
29 | procedure call of a function
30 | type conversion not possible
31 | MODULE expected
32 | IMPORT expected
33 | number too large
34 | factor starts with illegal symbol
35 | unexpected end of file
36 | file error
37 | error opening object file
38 | error opening symb?l file
39 | number not allowed
40 | Invalid number
41 | didn't found procedure in floating point library
42 | only a constant expression is allowed
43 | incompatible type of label or of subrange bound
44 | illegal type of case expression
45 | Lower bound greater than upper bound
46 | duplicate case value
47 | too many cases
48 | illegal base type
49 | invalid type name
50 | maximum constants in enumeration type is 255
51 | identifier, (, or [ expected
52 | index type of array must be a enumeration or a range
53 | maximum array size exceeded
54 | unexpected symbols
55 | illegal symbol
56 | record type expected
57 | expecting some parameters
58 | more parameters than expected
59 | object should be a module
60 | identifier expected
61 | undefined identifier
62 | identifier already declared
```

```
 63 | illegal use of type
 64 | -
 65 | W or L as extension size expected
 66 | error operand format
 67 | size error
 68 | addressing mode not supported
 69 | mode error
 70 | expression error
 71 | record error
 72 | var error
 73 | label already exist
 74 | references already exist
 75 | branch out of range
 76 | undefined label
 77 | register overflow
 78 | S, B, W or L as extension size expected
 79 | unknown identifier
 80 | constant value expected
 81 | mode not supported
 82 | maximum address size exceeded
 83 | -
 84 | illegal index type
 85 | constant out of range
 86 | field name is not available in this record
 87 | variable is not a pointer
 88 | not possible to make ?ype negative
 89 | AND, OR and NOT only possible with boolean type
 90 | set element out of range
 91 | basetype of set incompatible
 92 | type mismatch
 93 | maximum of nested WITH exceeded
 94 | overflow failure
 95 | incompatible operand types
 96 | operand type incompatible with DIV
 97 | operand type incompatible with MOD
 98 | operand type incompatible with AND
 99 | operand type incompatible with OR
100 | operand type incompatible with +
101 | operand type incompatible with -
102 | operand type incompatible with /
103 | operand type incompatible with *
104 | operand type incompatible with relation
105 | illegal parameter type of standard procedure
106 | register parameter expected (0-15)
107 | type as parameter expected
108 | exceeding size of this TAG variable
109 | Assignment to a non variable
110 | error procedure assignment
111 | incompatible assignment
112 | illegal assignment
113 | boolean type of expression expected
114 | invalid call, object is not procedure
115 | parameter types not identical
116 | illegal type of case expression
117 | illegal type of control variable
118 | identifier after FOR has an illegal type
119 | incompatible operands
```

```
120 | step in FOR statement cannot be zero
121 | too many exit statements
122 | exit not inside LOOP statement
123 | more parameters in definition than in implementation.
124 | mismatch between return types
125 | function in definition and a procedure in implementation.
126 | procedure in definition has parameters and in implementation not
127 | error in export list
128 | there are procedures without bodies
129 | maximum length of modulename exceeded (30 chars)
130 | division by zero
131 | internal compiler error; unknown standard function
132 | string is too long
133 | illegal symbol format
134 | symbol keys do not match
135 | 32KB modulcode exceeded
136 | maximum structures in symbol file exceeded
137 | maximum size constant data exceeded
138 | maximum global modules exceeded (64)
139 | internal compiler error, registers not freed
140 | expression not allowed or illegal
141 | expression not addressable or loadable
142 | fieldindex to big
143 | expected dynamic array parameter
144 | illegal range
145 | longreal type expected
146 | CAST can only make a type transfer with same sizes
147 | Not allowed to pass a constant to a var parameter
148 | illegal character
149 | only by first element and only integer constant
150 | register already used
151 | LONGREALS have to be in D0..D6
152 | negative bound not allowed
153 | upper bound may not greater than 31
154 | complex type not allowed in registers
155 | function may not be an ARRAY or RECORD type
156 | function size to big!
157 | Opaque error
158 | internal compiler error, D0 and/or D1 are in use
159 | internal compiler error, this should never happen
160 | Global register-vars not allowed
161 | expected a simple type number
162 | didn't found the procedure in ModulaLib
163 | did not find the ModulaLib
164 | Illegal character after "\"
165 | internal error
166 | invalid option only +,- or =
167 | this option is at this point not changeable
168 | these options can only be set once
169 | undefined option
170 | real type expected
171 | xref expected identifier
172 | label or memmonic expected
173 | use EVEN, odd address!
174 | address register expected!
175 | absolute expression expected
176 | no operand allowed
```

```
177 | expected one operand
178 | expected two operands
179 | size bigger as 8 Bit
180 | size bigger as 16 bit
181 | size bigger as 3 bit (1-8)
182 | illegal trap vector (0-15)
183 | data register expected (D0-D7)
184 | var outside range 0..MAX(INTEGER)
185 | mode must be d16(PC)
186 | not possible to use register variable
187 | you may only use string constants
188 | syntax failure module name
189 | array to big
190 | index from array to big
191 | references on a unknown type
192 | empty string is not allowed
193 | string only with DC.B allowed
194 | total local variable size to large, must be <32KB!
195 | parameter out of range
196 | expression too complex
197 | source and destination size must be the same!
198 | can only convert strings with size <= 4 bytes
199 | size bigger as destination
200 | CAST cannot transfer from or to longreal
201 | type expected
202 | You may only use code procedures in a definition file without an
      object file
203 | You may not use vars in a definition file without an object file
204 | relocation not supported
205 | not a valid class
206 | there are undefined methods
207 | unknown method/var
208 | cannot load 'Self' param
209 | method matches not inherited method
      (PROCEDURE/CONSTRUCTOR/DESTRUCTOR)
210 | illegal use of CONSTRUCTOR/DESTRUCTOR
211 | CONSTRUCTOR/DESTRUCTOR may not have any parameters and/or function
212 | size is not equal to four
213 | Typedescriptor overflow
214 | INLINE procedure in definition and not in implementation module
215 | didn't found the procedure in AutoLib
216 | didn't found the module AutoLib
217 | option already defined
218 | conditional compiling to deep nested
219 | conditional option ENDIF expected
220 | conditional ENDIF without IF
221 | error opening inline reference file
222 | function may not be used in a library!
223 | incomplete exception handler
224 | POINTER TO undefined identifier
225 | to many elements
226 | to little elements
227 | illegal inline format
```

1.29 Automatische Optimierungen

Automatische Optimierungen

Constant folding (Konstantenauswertung)

Wenn die Operanden eines Operators Konstanten sind, wird dieser von cmc bereits während des Compilierens ausgewertet.

Constant merging (Konstantenverschmelzung)

Wenn die selbe Zeichenkette mehr als einmal benutzt wird, dann wird diese nur einmal im Datensegment des Programms abgelegt.

Wenn sie also z.B. mehrmals ADR("Ready") benutzen, dann wird nur eine Kopie des Textes angelegt.

Inlining

Wenn Prozeduren mit INLINE markiert werden, dann wird die Prozedur direkt in den Code eingefügt, wird also nicht angesprungen. Dies beschleunigt das Programm! Andererseits wird daduch auch die Programmgröße erhöht. So werden also am besten nur kleine Prozeduren mit INLINE markiert.

Peephole-Optimierungen

Es würde den Rahmen der Anleitung sprengen, alle lokalen Kleinst-optimierungen anzugeben.

Andere Optimierungen

Einige andere simple Optimierungen des Compilers:

- Verzweigungen werden ggf. von "lang" auf "kurz" umgestellt
- unbenutzte NOPs werden entfernt
- Unbenutzte LINK A5,#0 werden entfernt

Optimierendes Linken

Der Linker von Cyclone (cml) entfernt unbenutzten Code automatisch, sobald ein ausführbares Programm erstellt wird. Diese Entfernung bezieht sich jeweils auf komplette Prozeduren.

1.30 Compiler Direktiven

Compiler Direktiven

Eine Compiler Direktive ist ein Kommentar mit einer speziellen Syntax.

Diese Direktiven müssen direkt am Anfang eines Kommentars beginnen; eine Direktive startet mit "(*$" und endet mit "*)". Mehrere Direktiven können in einem Kommentar eingebettet werden. Sie können allerdings nicht

C++~Kommentarstil
(//) für diese Direktiven einsetzten.

Es gibt zwei Arten von Direktiven:

- Schalterdirektiven
- Bedingte Direktiven

Schalterdirektiven können ein- und ausgeschaltet werden mit + oder - und mit = zurückgesetzt werden. Diese Direktiven haben immer eine höhere Priorität als die Optionen der Kommandozeile.

Schalterdirektiven

Option	Stack	Standard	Beschreibung
StackChk	ja	*	Füge Code für Stack-Überprüfung ein.
NilChk	j	*	Füge Code für NIL-Test ein
RangeChk	j	*	Füge Code für Bereichsprüfung ein (Range).
OverflowChk	j	*	Füge Code für Überlaufsprüfung ein (Overflow)
CaseChk	j	*	Füge Fallprüfung ein (CASE)
ReturnChk	j	*	Füge Code für Rückgabetest ein (RETURN)
LoadA4	j	Aus	Register A4 mit mit globalem Datenzeiger neu laden
SaveA4	nein	Aus	Speichere A4 auf dem Stack
EntryExitCode	n	An (+)	Mit dieser Option kann der Standard-ein- und -austrittscode einer Prozedur entfernt werden. Falls ausgeschaltet, muß dieser selbst erstellt werden! (LINK, UNLINK, RTS etc)
CopyDyn	n	An (+)	Normalerweise kopiert der Compiler einen Open-Array-Werteparameter in den Stack und benutzt diesen in der Prozedur. So wirken sich Änderungen an dem Parameter nur lokal aus! Dies kann ausgeschaltet werden, sofern nichts am Parameter geändert wird, was ein bißchen Code spart.
CaseTab	j	Aus(-)	Mit dieser Methode wechselt man zwischen Tabellengenerierung und eine einfachen Methode bei der Compilierung von CASE. Bei langen CASE-Ausdrücken erzeugt CaseTab+ kürzeren Code, sonst ist die einfache Methode oftmals besser.
Implementation	n	An (+)	Diese Option kann nur am Anfang einer Definitionsdatei gesetzt werden. Ist

			sie ausgestellt, dann erwartet der Compiler keine Implementationsdatei. Siehe auch Modul ASCII.def
ReloadA4	n	Aus	Aktiviert, wird A4 am Anfang jeder Prozedur neu geladen.
LibCode	n	Aus	Muß bei Erzeugung von Library-Code aktiviert sein.
AutoLib	n	An (+)	Aktiviert, öffnet ein Definitions-modul ggf. eine Library automatisch. Sonst muß die Library per Hand geöffnet werden und der BasePointer in der entsprechenden Variablen ge-speichert werden.
EntryClear	n	An (+)	Bei Aktivierung werden alle lokalen Variablen auf "Null" gesetzt.
Align	n	An (+)	Bei Aktivierung werden alle Variablen auf Langwörter ausgerichtet.
SaveScratch	y	An (+)	Aktiviert, sollen die Register A0,A1, D0 und D1 bei Aufruf einer Prozedur auf dem Stack gesichert werden. Sie werden gesichert, wenn sie benutzt werden (z.B. als Registerparameter). Sofern sie sicher sind, daß die Regis-ter in der aufzurufenden Prozedur nicht zerstört werden, kann diese Op-tion ausgeschaltet werden. Das spart einigen Code!

Hier ein Beispiel zur Nutzung einer solchen Direktive:

```
(*$NilChk- RangeChk+ ReturnChk= *)
```

Bedingte Direktivenes

```
SET    Bezeichner        Setzt einen Bezeichner auf TRUE
CLEAR  Bezeichner        Setzt einen Bezeichner auf FALSE
NOT    Bezeichner        invertiert Bezeichner
IF     Bezeichner        \
ELSE                      > Schlüsselwörter IF [NOT] Bezchn. [ELSE] END
END                      /
```

Hier ein Beispiel:

```
MODULE Directives;
(*$ SET English *)

IMPORT io:InOut;

BEGIN
 (*$ IF English *)
    io.WriteString('English');
 (*$ ELSE *)
    io.WriteString('No English');
```

```
   (*$ END  *)
  END Directives.
```

1.31 Der Inline-Assembler

```
                Der Inline-Assembler

Ein 68000-Assebler wird bereitgestellt. Sobald ASSEMBLE von
                SYSTEM
                importiert wurde, kann Inline-Assembler-Code zwischen den ↩
                   Schlüsselwörten
ASSEMBLE und END geklammert eingegeben werden. Assembler-Code kann formlos
eingegeben werden, Kommentare werden im Modula-2 Stil eingegeben. Auf alle
Modula-2 Variablen kann im Assembler zugegriffen werden. Relokation kann
(noch) nicht benutzt werden.

Bedenken Sie, dies ist ein Modula-2 Compiler, kein Assembler! Der Inline-
Assembler ist nur für kurze zeitkritische bzw. Low-Level-Routinen gedacht.


Schlüsselwörter


Folgende Instruktionen werden derzeit unterstützt:.

A0, A1, A2, A3, A4, A5, A6, A7, D0, D1, D2, D3, D4, D5, D6, D7, DC, MP, OR,
SB, SP, SR, PC, CCR, USP, EVEN, XREF, ABCD, ADD, ADDA, ADDI, ADDQ, ADDX,
AND, ANDI, ASL, ASR, BCC, BCHG, BCLR, BCS, BEQ, BGE, BGT, BHI, BLE, BLS,
BLT, BMI, BNE, BPL, BRA, BSET, BSR, BTST, BVC, BVS, CHK, CLR, CMP, CMPA,
CMPI, CMPM, DBCC, DBCS, DBEQ, DBF, DBGE, DBGT, DBHI, DBLE, DBLS, DBLT, DBMI,
DBNE, DBPL, DBRA, DBT, DBVC, DBVS, DIVSI, DIVS, DIVU, EOR, EORI, EXG, EXT,
JMP, JSR, LEA, LINK, LSL, LSR, MOVE, MOVEA, MOVEM, MOVEP, MOVEQ, MULS,
MULSI, MULU, NBCD, NEG, NEGX, NOP, NOT, OR, ORI, PEA, RESET, ROL, ROR, ROXL,
ROXR, RTE, RTR, RTS, SBCD, SCC, SCS, SEQ, SF, SGE, SGT, SHI, SLE, SLS, SLT,
SMI, SNE, SPL, ST, STOP, SUB, SUBA, SUBI, SUBQ, SUBX, SVC, SVS, SWAP, TAS,
TRAP, TRAPV, TST, UNLK


Wie kann ich diesen Assembler benutzen?


Das ist nicht schwer. Zuerst importiert man ASSEMBLE aus
                SYSTEM
                , danach kann
der Assembler beliebig genutzt werden.

Beispiele:

PROCEDURE StrLength(s:ARRAY OF CHAR):INTEGER; (*$ EntryExitCode- *)   (*
Keinen Ein-/Austrittscode erzeugen, das machen wir selbst! *)
 BEGIN
 ASSEMBLE(
    MOVE.L  (A7)+,A0 (* return address *)
    MOVE.L  (A7)+,A1 (* s *)
    MOVE.L  (A7)+,D0 (* HIGH(s) *)
```

```
    MOVE.L  D0,D1
  l:
    TST.B   (A1)+
    DBEQ    D1,l
    SUB.W   D1,D0     (* len:=HIGH(s)-D1 *)
    JMP     (A0)
  END);   END StrLength;

PROCEDURE SetAVar; VAR i:INTEGER; BEGIN
 ASSEMBLE(
    MOVE.W #1,i(A5)
    (* Ein-(Austrittscode wird vom Compiler erzeugt
       (EntryExitCode ist an !!!) *)
 END) END SetAVar;
```

Wie man sehen kann, werden Kommentare nicht mit einem Semikolon (;) begonnen, sondern mit (* und abgeschlossen mit *) wie in Modula-2.

1.32 Inlining

Inlining

Cyclone kann Inline-Code erzeugen. Wenn eine Funktion mit INLINE markiert wird, dann erzeugt der Compiler Code an der Stelle, wo die Funktion aufgerufen wird, anstelle einer Verzweigung. Dies kann den Programmcode etwas vergrößern, kann ihn aber auch schneller machen. Inlining kann auch die Compiliergeschwindigkeit erhöhen.

Wenn Inline-Funktionen in einem Implementationsmodul benutzt werden, dann müssen exportierte Funktionen sowohl im Definitionsmodul als auch im Implementationsmodul mit INLINE markiert werden.

z.B.

```
PROCEDURE Times(x{2},y{3}:LONGINT):LONGINT; INLINE;
```

Diese Prozedur wird bei folgendem Aufruf inline erzeugt

```
BEGIN
 i:=Times(3,4); END x.
```

Geschickterweise sollte man Inlining nur für kleine Prozeduren verwenden.

Selbverständlich kann man nicht die Adresse eine Inline-Prozedure bestimmen oder diese rekursiv aufrufen.

1.33 Objectformat

Objectformat

Momentan erzeugt der Compiler nur Code im Small-Modell (Large wird sobald wie möglich impementiert). Jedes Modul hat seine eigenen Code- und Daten-

segmente, der Linker bindet sie dann alle zusammen.

1.34 SYSTEM Module

Das Modul SYSTEM enthält die folgenden Definitionen:

```
TYPES
 BYTE
 WORD
 ADDRESS
 SHORTSET
 BITSET
 LONGSET

 PROCEDURE ADR(VAR a:AnyTyp):ADDRESS;
 PROCEDURE ASSEMBLE({Assembler command(s)} END);
 PROCEDURE CAST(NewAnytyp,OldAnytyp):NewAnyTyp;
 PROCEDURE REG(rconst:[0..15]):LONGINT;
 PROCEDURE SETREG(rconst:[0..15],t:typesize4);
 PROCEDURE SHIFT(x:ScalarTyp,n:LONGINT):ScalarType;
 PROCEDURE TAG(VAR x:Anytyp; val {,val}:ADDRESS):ADDRESS;
 PROCEDURE TSIZE(t:Anytyp):LONGINT;
 PROCEDURE RAWDATA(const:word, {const});

ADR
 Die 'ADR' Funktion bestimmt die Laufzeitadresse einer Variablen oder
 Stringkontanten. Der Typ der Rückgabe ist ADDRESS. Es kann stattdessen
 auch das Symbol @ benutzt werden.

ASSEMBLE
 Aktiviert den
                eingebauten~Assembler
                .

CAST
 Mit CAST kann man einen Typ ohne Konversion in einen neuen Typ ändern.
 Als einzige Beschränkung bleibt bestehen, daß Originaltyp und neuer
 Typ die gleiche Größe haben müssen.

REG
 Gibt den Wert eines Registers zurück.
 0..7  -> D0..D7
 8..15 -> A0..A7

SETREG
 Setzt den neuen Wert (t) in das Register rconst.
 0..7  -> D0..D7
 8..15 -> A0..A7

SHIFT
 Diese Prozeduer verschiebt Argument x um n Bits. Ein positiver Wert von n
 bewirkt eine Verschiebung nach links, ein negativer nach rechts.
  TAG
 Der erste Parameter eines Tags bestimmt den Speicherbereich, der für die
 Langwörter benötigt wird, die in den nächsten Parametern folgen.
```

```
 f.e.  VAR TagBuffer:ARRAY[0..10] OF LONGINT;
           TagAdr:ADDRESS;

 TagAdr:=TAG(TagBuffer,waTitle,ADR('TagDemo'),waWidth,500,waHight,200,tagEnd
));

 TagAdr Memory  0    ADR('TagDemo')
                4    500
                8    200
                12   tagEnd

 Die TAG-Funktion ist nur aus Kompatibilitätsgründen mit anderen Compilern
 vorhanden. Normalerweise wird besser die Cyclone "[ stuff ]" Notation
 benutzt.

 Zum Beispiel

 DoMethod(myobj, [sometag, someval, anothertag, anotherval, tagEnd]);


                Hier
                 gibt es mehr Informationen.


TSIZE
 Nur aus Kompatibilitätsgründen implementiert. Besser SIZE benutzen.

RAWDATA
 Hiermit können rohe Daten in eine Objektdatei integriert werden.
```

1.35 Ein einfaches Programm erstellen

```
In diesem Kapitel durchlaufen wir alle Schritte, um aus einem kleinen
Programmtext ein ausführbares Programm zu machen.

1) Schreiben Sie mit Ihrem bevorzugtem Texteditor ein Cyclone Module-2
   Programm, z.B.

   MODULE HelloWorld;

   FROM InOut IMPORT WriteString, WriteLn;

   BEGIN
    WriteString("Hello World!"); WriteLn;
   END HelloWorld.


   Speichern Sie dieses Programm in eine Datei, die mit .mod endet,
   beispielweise Helloworld.mod!

2) Zum compilieren kann man cmm (das make Programm) oder cmc benutzen
   (den Compiler und später den Linker). Ich werde beides demonstrieren.

   Zuerst mit dem Make Programm.
```

```
shell> cmm helloworld
Cyclone Modula-2 Make V1.06e Beta,
29.06.96,(c) Marcel Timmermans.
- helloworld.mod
- am2:modules/txt/InOut.def
- am2:modules/txt/InOut.mod
- am2:modules/txt/Workbench.def
- am2:modules/txt/GraphicsD.def
- am2:modules/txt/Hardware.def
- am2:modules/txt/ExecD.def
- am2:modules/txt/UtilityD.def
- am2:modules/txt/IntuitionD.def
- am2:modules/txt/KeyMapD.def
- am2:modules/txt/Timer.def
- am2:modules/txt/InputEvent.def
- am2:modules/txt/DosD.def
- am2:modules/txt/Reg.def
- am2:modules/txt/DosL.def
- am2:modules/txt/ModulaLib.def
- am2:modules/txt/ModulaLib.mod
- am2:modules/txt/ExecL.def
- am2:modules/txt/Ascii.def
- am2:modules/txt/String.def
- am2:modules/txt/String.mod
- am2:modules/txt/Convert.def
- am2:modules/txt/Convert.mod
+ helloworld.mtx

cmc  helloworld.mod
Cyclone Modula-2 Compiler V0.72e Beta,
14.07.96,(c) Marcel Timmermans.
- cmc.opt (+g-snrofcd)
- helloworld.mod
- am2:modules/sym/ModulaLib.sym
- am2:modules/sym/InOut.sym
+ HelloWorld.obj
  Optimizing ...  18 Bytes optimized
  CODE: 48 CONST: 0 BBS: 4 DATA: 0

cml  helloworld
Cyclone Modula-2 Linker V1.09e Beta,
20.06.96, (c) Marcel Timmermans.
- helloworld.obj
- am2:modules/obj/ModulaLib.obj
- am2:modules/obj/ExecL.obj
- am2:modules/obj/InOut.obj
- am2:modules/obj/Convert.obj
- am2:modules/obj/String.obj
- am2:modules/obj/DosL.obj
+ helloworld
  CODE:1708 DATA:0 VAR:136 PROGRAM: 1744
  Purebit set. Program is reentrant.
Shell>
```

Tippen Sie helloworld in der Shell, um das Prgramm zu starten:

```
Shell>Helloworld
Hello World!
Shell>
```

Das Programm kan auch mit dem Compiler und dem Linker direkt erstellt werden.

```
Shell>cmc helloworld.mod
Cyclone Modula-2 Compiler V0.72e Beta,
14.07.96,(c) Marcel Timmermans.
- cmc.opt (+g-snrofcd)
- helloworld.mod
- am2:modules/sym/ModulaLib.sym
- am2:modules/sym/InOut.sym
+ HelloWorld.obj
  Optimizing ...  18 Bytes optimized
  CODE: 48 CONST: 0 BBS: 4 DATA: 0

Shell>cml helloworld
Cyclone Modula-2 Linker V1.09e Beta,
20.06.96, (c) Marcel Timmermans.
- helloworld.obj
- am2:modules/obj/ModulaLib.obj
- am2:modules/obj/ExecL.obj
- am2:modules/obj/InOut.obj
- am2:modules/obj/Convert.obj
- am2:modules/obj/String.obj
- am2:modules/obj/DosL.obj
+ helloworld
  CODE:1708 DATA:0 VAR:136 PROGRAM: 1744
  Purebit set. Program is reentrant.
Shell>
```

Das Programm kann wiederum wie oben beschrieben gestartet werden. Komplexe Programme können leichter mit cmm erstellt werden, weil dieses Programm die Module in der richtigen Reihenfolge compiliert und linkt. Für einfache Programme kann dies schneller geschehen, indem man Compiler und Linker direkt aufruft.

Die Compiler- und Linkerausgabe ist etwas anders als oben dargestellt. Normalerweise werden die Module nicht auf mehreren, sondern auf einer Zeile angezeigt.

1.36 Erweiterungen des Compilers

THIS PAGE IS UNDER CONSTRUCTION :)

Hier finden Sie compiler spezifisch auskunft!

Register

Handhabung von Registern

```
               Listen
                                    Statische Listen [..,..]
   Threads                    Wie Threads benutzt werden (English)

               Inlining
                                 Benutzung von Inlining

               OO-Programmierung
                         OO-Programmierung

               Exception-handling
                        Benutzung von Exception-Handlern

               Addressierbare~Konstanten
                Wie werden adressierbare Konstanten benutzt

               Escape-Zeichen
                           Escape-Zeichen in Zeichenketten und -konstanten

               CLOSE
                                   CLOSE-Schlüsselwort

               C++~Kommentare
                           C++ Kommentar (//)
```

1.37 Register

```
                Register

Frei benutzbare Register sind D2..D7 und A0..A3. Die Register A4,A5,A6 und
A7 sind für speziellen Gebrauch reserviert!

Register A4

Dieses Register zeigt auf den kleinen globalen Datenbereich. Nach Ändern
dieses Registers sind viele Operation nicht mehr möglich!! Wann immer A4
geändert wird, muß diese Register mit (
               SaveA4
               ) auf den Stack gelegt und
später wieder zurüchgeschrieben werden. Es kann auch
               LoadA4
                zum
Wiederrücksetzen des Registers benutzt werden.

Bitte Vorsicht beim Ändern diese Registers, dies kann zu seltsamen Fehlern
führen.


Register A5

Dies ist das LINK Register und die Basis der lokalen Variablen und Parameter
einer Prozedur. Dieses Register wird immer bei Start einer Prozedur geladen
(sofern EntryExitCode angestellt ist). Das bedeutet, daß das Register  bei
Benutzung von Assembler niemals geändert werden soll!
```

Register A6

Dieses Register zeigt auf eine Library, das geladen wird, wenn von Library-Prozeduren/Funktionen Gebrauch gemacht wird.

Register A7

Dieses Register wird als Stackpointer eingesetzt.

Registervariablen

Registervariablen können in lokalen Prozeduren eingesetzt werden. Dies geschieht mit folgender Syntax:

```
VAR r{3}:INTEGER;
```

Dies bedeutet, daß bei Benutzung von 'r' immer auf D3 zugegriffen wird.

Registerparameter

Es ist ebenso möglich, Parameter direkt in Register zu verlagern. Eigentlich wurde dies Möglichkeit für Library-Prozeduren geschaffen, kann sich aber auch sonst als sinnvoll erweisen.

Beispiel.

```
 PROCEDURE Swap(x{0},y{1});
```

Dies bedeutet, daß die Argument x und y in den Registern D0 und D1 übergeben werden. Dies ist nur mit einfachen Typen möglich. Strukturierte Typen, wie Records oder Arrays sind nicht möglich. Sie müssen Zeiger auf diese übergeben.

1.38 Listen

Statische Listen ([..,..])

Listen sind ähnlich konstanten Array von Langwörtern. Listen werden mittels [und] geschrieben und mit Kommas unterteilt. Beispiel:

```
[1,2,3,@i,5]
```

ist ein Ausdruck, der als Rückgabewert eine Adresse auf eine bereits initialisierte Liste hat.

Speicherformat

```
     Offset
     ---

      0      1
```

```
      4      2
      8      3
      12     ADDRESS OF i
      16     5
```

Listen sind am nützlichsten beim Schreiben von taglists. Siehe auch das
'Rom Kernal Reference für mehr Informationen über tags. mit dieser Funktion
können einfach Taglists gefüllt werden. Listen können auch verschachtelt
werden. Beispiel:

```
[1,2,3,[10,11,12],ADR("test")]
```

Speicherformat

```
     Offset
     ---

      0     1
      4     2
      8     3
      12    Adresse der Liste [10,11,12] 24
      16    Adresse des konstanten Textes 'test'
      20    0
      24    10
      28    11
      32    12
```

1.39 Addressierbare Konstanten

Addressierbare Konstanten

Sie können die ADR Funktion in einem CONST Definitionsblock benutzen, z.B.

```
...

TYPE
 PenTyp   = ARRAY[0..3] OF INTEGER;
 LabelRec = RECORD
              adr:ADDRESS;
              pens:PenTyp;
              attr:ud.TagItemPtr;
            END;
 LabelTyp = ARRAY[0..2] OF LabelRec;


CONST

 Labels =LabelTyp{
            LabelRec{ADR("Display"),PenTyp{-1,..},NIL},
            LabelRec{ADR("Edit"),PenTyp{-1,..},NIL}
            LabelRec{ADR("File"),PenTyp{-1,..},NIL}};
```

So kann ein Datenblock erstellt werden. Momentan kann nur Text als Argument

für ADR benutzt werden.

1.40 Beschreibung der Schlüsselwörter

COMPDATE & COMPTIME

Bei Benutzung des Schlüsselwortes COMPDATE setzt der Compiler automatisch das aktuelle Datum als Null-terminierte Zeichenkette ein. Ebenso wird für COMPTIME die Zeit eingesetzt.

```
f.e.
  CONST
  CurrCompDateTxt = 'Current compilation date is : '+COMPDATE;
  CurrCompTimeTxt = 'Current compilation time is : '+COMPTIME;
```

Am 12 März 1994 um 12:01:01 Uhr wird die konstante Zeichenkette nach der Compilation so ausehen:

```
  CurrCompDateTxt = 'Current compilation date is : 12-03-1994';
  CurrCompTimeTxt = 'Current compilation time is : 12:01:01';
```

ANMERKUNG: Dieser Text ist nur in der Objektdatei sichtbar. So bleibt die Quelldatei intakt.

IGNORE

Mit dem Schlüsselwort IGNORE können Funktionsresultate ignoriert werden, z.B. soll das Resultat von LoadFile(name:ARRAY OF CHAR):BOOLEAN; ignoriert werden, dann kann die Funktion wie folgt aufgerufen werden:

```
IGNORE LoadFile('myfile');
```

Dies würde einigen Code sparen im Gegensatz zu den Konstruktionen IF LoadFile('myfile' THEN END; oder ok:=LoadFile('myfile');

Benutzen sie IGNORE vorsichtig. Ignorieren Sie niemals Fehlercodes, die später Probleme bereiten könnten.

1.41 Ausgabedateien

Objektdatei (*.obj)

Eine Objektdatei entält Maschinencode, Daten und Informationen für die Relokation. Das Objektformat ist BLink-Kompatibel.

Symboldatei (*.sym|*.ref)

Eine Symboldatei enthält Informationen über Konstanten, Typen, Variablen und

Prozeduren, die von Implementationsmodul exportiert werden und das vom Compiler gebraucht wird, wenn das Modul von einem anderen Modul importiert wird. Die Dateien mit der Endung .ref werden vom Source-Level-Debugger benötigt, der noch in Arbeit ist.

1.42 Fehler berichten

Fehler berichten

Die Programme von Cyclone wurden mit großer Aufmerksamkeit in Bezug auf ihre Zuverlässigkeit hin programmiert. Nichtsdestotrotz können sie noch Fehler enthalten.

Wenn Sie einen finden, wenn Sie Kommentare oder Vorschläge haben, dann schreiben Sie bitte an
mich
eine E-Mail (in Englisch!), die den Fehler klar beschreibt.

Bitte fügen Sie ein Beispiel bei, das den Fehler demonstriert. Auch Angaben über Ihr System (KS, Prozessor, Speicher) sind sinnvoll.

1.43 Escape-Zeichen

Über die Benutzung von Escape-Zeichen in Zeichenketten und -konstanten
Es gibt eine Menge nützlicher Zeichen, die nicht in einfacher Weise in einen Quelltext geschrieben werden können. Deshalb erlaubt Cyclone die Benutzung von Escape-zeichen in Zeichenketten und -konstanten. Ein Escape-Zeichen besteht aus einem "\", gefolgt von einem oder mehreren anderen Zeichen. Das "\" Zeichen signalisiert dem Compiler, daß die folgenden Zeichen eine spezielle Bedeutung haben. Diese Bedeutungen sind:

```
\o     : füge eine Null ein, CHR(0)
\b     : füge Backspace ein, CHR(08H)
\e     : füge Escape ein, CHR(1BH)
\t     : füge Tabulator ein, CHR(09H)
\n     : füge Newline ein, CHR(0AH)
\f     : füge Form-Feed ein, CHR(0CH)
\r     : Füge Carriage Return CHR(0DH)
\[     : füge CSI (Control Sequence Introducer), CHR(9BH)
\xnn   : füge Zeichen mit ASCII Wert nn hexadezimal ein
\nnn   : füge Zeichen mit ASCII Wert nnn oktal ein
```

Jedes andere Zeichen bleibt unverändert, es gelten also auch folgende Definitionen:

```
\   ein einfacher Backslash
\'   Ein einfacher Apostroph
\"   Ein Anführungszeichen, auch in einer Zeichenkettenkonstenten
```

Beispiel:

```
 CONST
     EscText = "This is an\n\tescaped string constant\n";
```

1.44 Modul Break (Control-C)

Modul Break

Dieses Modul erlaubt die einfache Installation eines Break-Handlers, der auf die Dos-Signale Ctrl-C, Ctrl-D, Ctrl-E und Ctrl-F reagiert. Bei Import dieses Moduls muß die Prozedur InstallExeception aufgerufen werden, um den BreakHandler zu installieren. Mit der Prozedur TestBreak() kann jedes der Signale abgefragt werden. Wenn ein Signal entdeckt wurde, springt das Programm in den CLOSE-Teil des Programms und startet die Abschlußphase. Voreingestellt ist die Reaktion auf Ctrl-C. Geändert werden kann dies über das SET actualBreak.

Mit der Prozedur RemoveExeception kann der Break-Handler wieder entfernt werden. Das Modul Break tut dies aber auch automatisch im CLOSE-Teil.

Bei Benutzung des Threads-Modul funktioniert das Modul Break nicht.

1.45 Wissenswertes

TRUE und FALSE

Der einer BOOLEAN zugewiesene Wert bei TRUE ist = -1 (FFH), 1 Byte groß, für FALSE ist dies = 0 (0H), ebenfalls 1 Byte groß.

Dies ist in TagListen sehr wichtig, da dies anders als in der Sprache C ist! In C ist false ebenfalls 0, während true meistens als 1 definiert ist.

Daher wurden die Konstanten LTRUE und LFALSE in ExecD definiert. In TagListen sollten dies Konstanten Benutzt werden.

Register A4

Dieses Register zeigt auf den kleinen globalen Datenbereich. Nach Ändern dieses Registers sind viele Operation nicht mehr möglich!! Wann immer A4 geändert wird, muß diese Register mit (
SaveA4
) auf den Stack gelegt und
später wieder zurückgeschrieben werden. Es kann auch
LoadA4
zum
Wiederrücksetzen des Registers benutzt werden.

Bitte Vorsicht beim Ändern diese Registers, dies kann zu seltsamen Fehlern führen!(Siehe auch

```
                Register
                )


"/" und DIV

Gegeben sei folgender Ausdruck: x:=x/y; x,y sind INTEGERs, dann wird der
"/"-Operator genau wie der Operator DIV benutzt. Man könnte natürlich auch:
x:=x DIV y; schreiben. Bei Gleitkommazahlen muß zur Division immer der
Operator "/" benutzt werden.
```

1.46 CLOSE Anweisung

```
Bei Benutzung der Anweisung 'CLOSE' kann ein Anweisungsblock definiert
werden, der bei Beendigung des Programms ausgeführt wird, sei dieses Ende
erwartungsgemäß oder durch einen Fehler gekommen.

Hier können also Dinge erledigt werden, die bei Programmende oder -fehler
nötig sind, z.B. Speicherfreigabe, Ressourcen freigeben, Dateien und
Bibliotheken schließen etc.

Wenn ein Programm endet, dann werden alle CLOSE Teile aller importierten
Module in der richtigen Reihgenfolge geschlossen. Achtung: Ein CLOSE Teil
kann mehr als einmal aufgerufen werden! Deshalb muß der Code sehr umsichtig
programmiert sein, spezielle Abfragen vor Ressourcen-Freigabe sind
unvermeidbar.
```

1.47 Bekannte Fehler

```
CMM

 - Wenn ein Quellcode nicht zur Verfügung steht, dann versucht make nicht,
   das Modul, welches die Symboldatei zum fehlenden Quellcode importiert,
   neu zu compilieren!!
```

1.48 C++ Kommentar

```
Neben dem üblichen Modula-2 Kommentar "(* ... *)" kann auch der C++/BCPL
Kommentarstil "//"benutzt werden.

z.B.

 MODULE Example;
 // Dieses Programm benutzt den C++ Kommentarstil

 // jede Kommentarzeile muß mit '//' kommentiert werden.
 // Optionen können mit diesem Kommentarstil nicht gesetzt werden

 (*$ RangeChk- *) // ^^^^ Options
```

```
BEGIN
// Heute mal nichts machen
END Example.
```

1.49 Danke

Die folgenden Personen verdienen besonderen Dank von mir, da sie einen signifikanten Anteil an der Entwicklung des Cyclone Modula-2 Compilers hatten (keine besondere Reihenfolge)

* Robert Ennals - für umfangreiches Testen, das Schreiben einiger Module (Threads, MemPoll etc.), Durchsicht der Dokumentation und noch mehr
* Stefan Tiemann - für das Schreiben einiger Systemmodule, umfangreiches Testen und Testprogramme und die Übersetzung der Dokumentation ins Deutsche
* Wouter van Oortmersen - für seine hilfreichen Diskussionen
* Soenke Tesch - für umfangreiches Testen und Testprogramme
* Stefan Schulz - für das Codegeneratormodul zu MuiBuilder und umfangreiches Testen
* Stephan Splitthoff - für umfangreiches Testen
* Herbert Klackl - für umfangreiches Testen
* Dr. Maybe - für sehr hilfreiche Gesprächsstunden
* Ralph Babel - für sein GROSSARTIGES Buch "The Amiga Guru Book"
* Commodore - für die Erschaffung dieser wunderbaren Maschine!!! AMIGA ↩ FOREVER!

Weitere Leute, die essentiell an der Entwicklung von Cyclone beteiligt waren:

* Rhett R. Rodewald
* Robert Barton
* Tim Corringham
* S Sinclair
* Andrew P Scheller

und nicht zuletzt meine Freundin für die vielen Abende, die ich sie allein ließ. Danke Marie-An!

Und an all die Leute, die ich vergaß!