StormC1.1

COLLABORATORS

	TITLE : StormC1.1	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 29, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 StormC1.1 **1**

1.1 StormC.guide . . . 1
1.2 StormC.guide/STC_Sort . . . 2
1.3 StormC.guide/STC_Sections . . . 2
1.4 StormC.guide/STC_Project . . . 3
1.5 StormC.guide/STC_Options . . . 3
1.6 StormC.guide/STC_Objectcodes . . . 4
1.7 StormC.guide/STC_Incbin . . . 4
1.8 StormC.guide/STC_SAVEDS . . . 5
1.9 StormC.guide/STC_Inline . . . 6
1.10 StormC.guide/STC_Except . . . 6
1.11 StormC.guide/STC_Proto . . . 6
1.12 StormC.guide/STC_Codefold . . . 7
1.13 StormC.guide/STC_Editor . . . 7
1.14 StormC.guide/STC_Librarian . . . 9
1.15 StormC.guide/STC_Debugger . . . 9

Chapter 1

StormC1.1

1.1 StormC.guide

StormC V1.1 Erweiterungen

Software und Dokumentation

Inhaltsverzeichnis

Projektverwaltung	Sortieren von und innerhalb von Sektionen
Projektverwaltung	Neue Sektionen und Steuerungsmöglichkeiten
Tooltypes	GOLDED, SAVEMEM, QUIET
Optioneneinsteller	Getrennter Optioneneinsteller
Objektdateienablage	Codebedingte Objektdateienablage
INCBIN	Anbinden von Binärdateien
__saveds	Das Schlüsselwort __saveds
Inlining	Verhindern von Inline-Funktionen
Exceptions	Ignorieren von Exceptionspezifizieren
Rapid Prototyping	Ersetzung von nichtvorhandenen Funktionen
Codefolding	Optimierung beim Linken
Editor	Neue Funktionalität
StormLibrarian	Einfache Erzeugung von Linker-Bibliotheken
Debugger	Der in allen Teilen überarbeitete Debugger

1.2 StormC.guide/STC_Sort

Sortieren in der Projektverwaltung

Einzelne Einträge in Projektsektionen oder auch ganze Sektionen
können in der Reihenfolge schnell geändert (verschoben) werden.

Wie Sie wissen bietet die StormShell beim Hinzufügen von
Bibliotheken wie auch beim Hinzufügen von Dateien die Möglich-
keit, mehrere Dateien im ASL-Dateidialog auszuwählen. Die aus-
gewählten Dateien werden dann immer in alphabetischer Reihenfolge
in das Projekt aufgenommen. Dies kann besonders bei Bibliotheken
eine ärgerliche Folge haben.

Der ein oder andere hat sicher schon mit Linker-Fehlermeldungen
kämpfen müssen, nur weil er aus Unachtsamkeit vergessen hat, die
Bibliothek "Storm.Lib" vor der "Amiga.Lib" im Projekt einzutragen.

Bei der gleichzeitigen Auswahl von "Amiga.Lib" und "Storm.Lib" werden
die beiden in der alphabetischen Reihenfolge in das Projekt einsortiert
und müssen zuerst umsortiert werden, bevor gelinkt werden kann.

In der Version 1.1 der StormShell stehen dazu die beiden neuen
Menüpunkte "Nach oben verschieben" und "Nach unten verschieben" zur
Verfügung. Durch Anwahl eines der genannten Menüpunkte wird der aktuell
ausgewählte Projekteintrag oder die ganze Sektion nach oben oder
nach unten verschoben.

1.3 StormC.guide/STC_Sections

Neue Projektsektionen

In der neuen Version der Projektverwaltung wurde eine eigene
Sektion für StormWIZARD-Dateien ermöglicht. Die Integration von
StormWIZARD ist dabei genau so perfekt, wie die Einbindung
aller anderen Programme. Beim Hinzufügen von durch StormWIZARD
erzeugte Dateien mit der Endung ".wizard" wird die neue Sektion
"StormWIZARD Schnittstelle" angelegt.

Bei Doppelklick auf den Eintrag wird StormWIZARD automatisch gestartet
und die gewählte Datei geladen. Die Integration aller Entwicklungs-
komponenten garantiert eine optimale Strukturierung und Organisation
Ihrer Projekte.

Erweiterte Tastaturfunktionalität

Die Unterstützung durch zusätzliche Tastaturqualifier in der
Projektverwaltung war bisher nur sehr stiefmütterlich behandelt.
Bei Doppelklick auf den Programmnamen im Projekt und gleichzeitigem
Gedrückthalten der <ALT>-Taste wird das Programm automatisch im
Debug-Modus gestartet.

Hinzu kam nun die Unterstützung von Datatypes. Bei Doppelklick auf
einen Eintrag in der Projektverwaltung und Gedrückthalten der

<ALT>-Taste wird Multiview gestartet und die Datei geladen. Ist nun
ein entsprechender Datatyp in Ihrem System installiert, wird die
Datei angezeigt.

Ein kleines Beispiel verdeutlicht, wie sinnvoll diese Erweiterung ist:
Sie arbeiten an Ihrem Projekt und möchten auch eine AmigaGuide-
Dokumentation erstellen. Am einfachsten fügen Sie hierfür einen neuen
Eintrag in die Guide-Sektion ein. Bislang konnten Sie den Eintrag
zwar doppelklicken, um den Text dann mit dem Editor zu bearbeiten,
wollten Sie aber das Endergebnis sehen, mußte die Originaldatei
gesucht und doppeltgeklickt oder sogar Multiview manuell gestartet
werden.

Mit der neuen Projektverwaltung genügt es, die <ALT>-Taste gedrückt-
zuhalten, und den Eintrag in der AmigaGuide-Sektion doppelt zu
klicken. Daraufhin wird Multiview gestartet und Ihr AmigaGuide-
Dokument in der Anwenderfassung angezeigt. Eine Arbeitserleichterung,
um die wir von vielen Anwendern gebeten wurden!

1.4 StormC.guide/STC_Project

QUIET

Das Merkmal QUIET unterdrückt die Ausgabe der Startgrafik. Auch das kleine
Fenster, das bei zu geringer Farbanzahl alternativ erscheint, wird nicht
angezeigt.

SAVEMEM

Bei Speicherknappheit bietet der Einsatz dieses Merkmals die Möglichkeit das
System nur teilweise zu laden, so daß Speicher eingespart wird. SAVEMEM
verhindert das Laden von StormC, StormLINK, StormASM und StormRUN.

Erst bei Bedarf werden die jeweiligen Programmteile geladen und belegen
Speicher. Solange genügend Speicher frei ist, bleiben die Programme
jedoch im Speicher.

GOLDED

Anstelle von StormED wird alternativ der populäre Editor GoldED aus der
Schublade "GOLDED:" geladen. Da GoldED in der gleichen Weise wie StormED
genutzt wird und nicht wie üblich als Schaltzentrale für den Compiler
fungiert, muß der Editor speicherresident geladen werden.

Stellen Sie bitte im GoldED unter "Konfig/Diverses..." die Einstellungen
für "Diverses" auf "resident" und speichern die Einstellung.

1.5 StormC.guide/STC_Options

Neuer Optioneneinsteller mit erweiterten Optionen

Der neue Optioneneinsteller wurde aufgeräumt und in seiner Bedienung

wesentlich handlicher gestaltet. Die mit StormWIZARD verfügbaren
Karteikartenreiter verschaffen einen Ueberblick über alle Einstell-
möglichkeiten.

Der Einsteller wurde zudem in vier Gruppen gegliedert:

1. Projektumgebung

2. Compileroptionen

3. Linkeroptionen

4. Debuggeroptionen

1.6 StormC.guide/STC_Objectcodes

Codebedingte Objektdateien-Ablage

Viele Anwender der Version 1.0 sind begeistert von unserer Projekt-
verwaltung und der Möglichkeit, alles zum Projekt gehörige
strukturiert und überschaubar verwalten zu können. Selbstverständlich
gibt es auch hierfür viele Erweiterungsmöglichkeiten, die uns genannt
wurden.

Eine wichtige Erweiterung ist die Verwaltung verschiedener Versionen
Ihres Projektes. Gemeint sind z.B. Versionen Ihres Programmes
für verschiedene Prozessortypen. Bislang mußte das komplette Projekt
neu kompiliert werden, was bei umfangreichen Projekten unter Umständen
sehr lange dauern kann.

Die Version 1.1 bietet Ihnen die Möglichkeit, in jedem Projekt eine
Schublade anzugeben, in die die Objektdateien gespeichert werden
sollen. Für verschiedene Versionen genügt es nun, den einzelnen
Projekten unterschiedliche Schubladen für die Objektdateien anzugeben.
Beispielsweise "68K-Code", "020-Code","060-Code", usw...

Da in den Schubladen immer die Objektdateien mit den entsprechenden
Optionen enthalten sind, werden immer nur die Module neu kompiliert,
deren Quellcode tatsächlich geändert wurde!

Den entsprechenden Einsteller hierzu finden Sie im Optioneneinsteller
für die Projektumgebung.

1.7 StormC.guide/STC_Incbin

Anbinden von Binärdateien

Die neuen Linkereigenschaft, Binärdateien anbinden zu können, fand
man bisher eigentlich nur bei Assembler-Entwicklungssystemen. In
StormC 1.1 wurde sie zur StormWIZARD-Unterstützung integriert, kann
aber auch für andere Zwecke genutzt werden.

Bei Aktivierung der Option "Binde WIZARD-Datei" wird die in der
Projektverwaltung enthaltene Wizard-Datei an das Programm gelinkt.
Damit die Daten korrekt initialisiert werden können, muss selbst-
verständlich ein Symbolname für den Datenhunk angegeben werden. Die
Initalisierungs-Routine für die Wizard-Daten kann dann die an das
Programm angehängte Datei initialisieren.

1.8 StormC.guide/STC_SAVEDS

Die Nutzung von __saveds

Wenn Sie eine Funktion mit dem Schlüsselwort __saveds definieren
generiert der Compiler zusätzlichen Code am Anfang der Funktion der
die Adresse der LinkerDatabase in das Register a4 lädt.

Beispiel:

```
    int __saveds function(int a, int b);
```

Das Laden des Registers a4 mit dem Wert der LinkerDatabase ist dann
sinnvoll, wenn die betreffende Funktion von anderen Programmen genutzt
werden soll. Zum Beispiel in Shared Libraries, bei Hook-Funktionen oder
in Boopsi-Dispatchern.

Das Register a4 wird mit dem Wert des Symbols _LinkerDB wie folgt
initialisiert:

```
    LEA    _LinkerDB, a4
```

Falls die Compileroption "Near Data (a6)" eingeschaltet
ist, wird der Wert aus _LinkerDB selbstverständlich in das Register
a6 statt nach a4 geladen.

Das Symbol _LinkerDB wird von Linker mit der Adresse des ersten Elements
des NearDatahunks geladen.

Residentfähige Programme können nicht auf absolute Adressen verweisen.
Das Schlüsselwort __saveds kann daher nicht verwendet werden, wenn die
Linkeroption "residentfähiges Programm" eingeschaltet ist.

Die Angabe des Schlüsselwortes __saveds bei der Funktionsdefinition
entspricht dem Aufruf der Funktion "GetBasReg()" oder "geta4()". Bedeutet
allerdings keinen Unterfunktionsaufruf, spart daher Programmcode und
wird schneller abgearbeitet.

Das __saveds Schlüsselwort muß bei Funktionsdefinitionen enthalten sein,
kann aber in der Prototypdeklaration ignoriert werden. Wird __saveds im
Prototyp nicht aber bei der Definition einer Funktion angegeben, meldet
der Compiler einen Fehler. Bei Angabe des Schlüsselwortes vor Funktions-
pointern oder Funktionsparametern wird eine Fehlermeldung ausgegeben.

1.9 StormC.guide/STC_Inline

Verhindern von Inline-Funktionen

Inline-Funktionen werden wie Makros direkt an der Stelle des Aufrufs im Quelltext eingefügt. Der Compiler spart damit den Aufruf der Funktionen und kann daher extrem gut optimieren. Durch die Benutzung der Inline-Funktionen wird das Programm schneller und wenn man sich auf kurze Inline-Funktionen beschränkt (z.B. das Auslesen des Wertes eines Attributs einer Klasse) auch kürzer.

In der Testphase können Inline-Funktion jedoch sehr störend sein, da es für den Debugger unmöglich ist, den Quelltext zu Inline-Funktionen anzuzeigen.

Mit dem Schalter kann der Compiler angewiesen werden, die Inline-Funktionen wie normale Funktionen aufzurufen. Der Debugger kann dann auch den Quelltext dieser Inline-Funktionen anzeigen und im Einzelschrittmodus abarbeiten.
Dabei stört es nicht, dass Inline-Funktionen meistens in den Header-Dateien der Module zu finden sind.

1.10 StormC.guide/STC_Except

Ignorieren von Exceptionspezifizierern

Für die Testphase oder Sicherheitsfanatiker bietet C++ die Möglichkeit eine Funktion (auch Memberfunktionen einer Klasse) mit sogenannten Exceptionspecifier auszustatten: Mit Hilfe des Schlüsselworts <throw> hinter dem Funktionsprototyp wird eine Liste von Exceptions angegeben, die durch diese Funktion höchstens ausgeworfen werden. Alle anderen Expcetions werden als unerwartete Exception behandelt.

Dafür muss der Compiler um jede Funktion mit Excpetionspecifier einen "try-Block" setzen, der alle erlaubten Exception auffängt und wieder auswirft, aber alle anderen Exceptions abfängt und die Funktion unexpected() aufruft. Dieser zustäzliche "try- Block" macht das Programm länger und etwas langsamer.

Es empfiehlt sich deshalb nach ausreichender Testphase den Compiler anzuweisen diese Exceptionspecifier zu ignorieren. Das Programm wird dadurch unter Umständen drastisch kürzer.

1.11 StormC.guide/STC_Proto

Ersetzung von nichtvorhandenen Funktionen

Diese Option wurde zur Unterstützung des sogenannten Rapid Prototyping implementiert. Generieren Sie einfach eine Funktion mit dem Namen "_stub", schalten die genannte Option ein und programmieren drauflos. In der Stub-Funktion sollte vielleicht eine Debugausgabe gemacht oder

ein Dialog mit entsprechendem Hinweis: "Funktion noch nicht implementiert" ausgegeben werden.

1.12 StormC.guide/STC_Codefold

Codefolding (Fold Common Code)

Der StormLink ist in der Lage, einige Optimierungen, die ein Compiler macht, auch im fertigen Programm durchzuführen. Dazu gehört auch das Entfernen von mehrfach vorhandenem Code.

Wenn der StormLink zwei Routinen entdeckt, die identisch sind, so daß es keinen Unterschied macht, ob das Programm die erste oder die zweite aufruft, ersetzt der StormLink die zweite Routine durch den Code der ersten. Wer behauptet, daß er ja selber programmiert und sowas nicht macht, bitteschön. Aber nicht wundern, wenn bei der Benutzung von Templates das Programm immer länger und länger wird. Da diese Option einiges an Laufzeit kostet, ist ihre Benutzung auch nur dann zu empfehlen, wenn der Optimizer des Compilers auf der höchsten Stufe steht.

1.13 StormC.guide/STC_Editor

StormEd Farbeinsteller

Vielfach gewünscht, ist es nun in der aktuellen Version einfach möglich, die farbliche Textdarstellung mit der Maus zu verändern. Die einfache Einstellmöglichkeit löst damit die bislang komplizierte Angabe der Vorder- und Hintergrundfarbe in der jeweiligen Settings-Datei ab.

Die globale Editoreinstellungen wurde dafür um den Einsteller "Farben" (Karteikartenreiter) erweitert.

Mit dem Wechselsymbol direkt unter den Karteikartenreitern wird die "Syntaxart" gewählt. Es können Eingestellungen für diese Arten vorgenommen werden:

    Preprozessor -> Dictionary/Preprocessor.dic

    Beispiel: #include

    C/C++ Symbole -> Dictionary/C Symbols.dic

    Beispiel: void

    C/C++ Bibliotheksfunktionen -> Dictionary/C Library.dic

    Beispiel: printf

AmigaOS Typen -> Dictionary/Amiga Types.dic

Beispiel: ULONG

AmigaOS Funktionen -> Dictionary/Amiga Functions.dic

Beispiel: DisplayBeep()

ANSI-C Kommentare

Beispiel: /* ANSI-C Kommentar */

C++ Kommentare

Beispiel: // C++ Kommentar

String Konstanten

Beispiel: "Hello World"

Zeichen Konstanten

Beispiel: 'x'

Zahlen Konstanten

Beispiel: 42

Und selbstverständlich für die drei Benutzer-Bibliotheken!

Mit den Symbolen in der Gruppe "Farbe" kann die zwischen der Einstellung
für die Vordergrund- und für die Hintergrundfarbe gewählt werden.

Mit dem Wechselsymbol darunter wählt man für die jeweilige Art die
"Standard Textfarbe" oder die "Standard Hintergrundfarbe", wie sie
im Palette-Voreinsteller der Workbench eingestellt wurden. Will
man eine eigene Farbeinstellung vornehmen muß das Wechselsymbol die
Auswahl "Eigene Einstellung" enthalten. Dann wird das darunter angeordnete
Palettefeld zugänglich und man kann eine eigene Farbe auswählen.

In der Gruppe "Stil" kann unabhängig von den Farbeinstellungen ein
Textstil für die jeweilige Art gewählt werden.

In der Gruppe "Beispiel" wird ein Codefragment angezeigt, anhand dessen
die gemachten Einstellungen begutachtet werden können.

Klammern-Check und Blockeinrückung

Die automatische Überprüfung der Klammernstruktur ist sehr hilfreich
bei tief geschachtelten Konstruktionen. All zu leicht wird eine
Klammer vergessen und man erhält eine Fehlermeldung, die nicht
unbedingt auf eine fehlende Klammer schließen lässt. Auch bei der
Programmierung von Installer-Skripten ist diese Funktion sehr
hilfreich.

Markierte Textbereiche können ab sofort nach links oder rechts
verschoben werden. Halten Sie dazu die <Ctrl>-Taste gedrückt und
drücken die <Cursor links> bzw. <Cursor rechts>-Taste.

Je nach Tabulatoreinstellung wird dann der markierte Textblock
verschoben.

1.14 StormC.guide/STC_Librarian

Einfache Erzeugung von Linker-Bibliotheken

Nicht nur die Erzeugung von Shared-Libraries ist eine wichtige
Bedingung für ein echtes Entwicklungssystem, sondern auch die
Möglichkeit, Funktionsgruppen in einer Linkerbibliothek zusammen-
fassen zu können. Bei der Erzeugung von Bibliotheken wie beispiels-
weise der Amiga.Lib oder der Storm.Lib muß bereits beim Kompilieren
der Module darauf geachtet werden, dass jede Funktion am Besten in
einem eigenen Hunk abgelegt wird. Der Compiler bietet dazu die Option
"Eigener Hunk für jede Funktion". Nach dem Kompilieren müssen die
einzelnen Module zu einer Library "gejoint" werden, was lästige
Tipparbeit bedeutet.

Unser neuer StormLibrarian hilft Ihnen dabei und bietet die Möglich-
keit Objektmodule zu Library-Projekten zusammenzufassen. Haben Sie
Ihr Projekt erstellt, genügt ein einfacher Klick auf "Erzeugen" und
die Bibliothek wird unter dem angegebenen Namen erzeugt. Bei
Veränderung der Objektmodule muß nun lediglich das Projekt in den
StormLibrarian geladen und die Bibliothek neu erzeugt werden - fertig!

1.15 StormC.guide/STC_Debugger

Neues überarbeitetes Kapitel!

7 Debugger

Nach dem ersten erfolgreichen Kompilieren eines Programms
beginnt die Testphase. Natürlich könnte man das Programm
einfach aus einem CLI (bzw. einer Shell) heraus starten,
aber daraus ergeben sich einige, für den Amiga typische
Probleme:
1* Wenn das Programm gravierende Fehler erzeugt und das
   ist in C oder C++ jederzeit möglich, kann der Computer
   abstürzen, denn die CPU kennt sogenannte Exceptions,
   also Ausnahmezustände der Soft- oder Hardware. Diese
   werden durch das Betriebssystem nur sehr allgemein

behandelt und erlauben häufig keine vernünftige Weiterarbeit, ohne einen Neustart des Systems.

2* Wenn das Programm in eine Endlosschleife läuft, hilft zumeist nur ein Neustart.

3* Selbst wenn nach einer Exception der Computer nicht neugestartet wird, oder wenn das Programm mit der Funktion "exit()" oder "abort()" beendet wird, bleiben die Betriebssystemressourcen größtenteils belegt. Während reservierter Speicher nur ärgerlich ist, können eine offengebliebene Datei oder ein offengebliebenes Fenster die Weiterarbeit massiv behindern oder auch zu einem Absturz des Betriebssystems führen.

Die genannten Fehler lassen sich nicht auf die Quelltext-position im Programm zur ückführen.

Allgemeines zur Runshell

Aus diesen vier Gründen besitzt StormC eine "Runshell". Diese startet das Programm ebenfalls wie aus dem CLI, enthält jedoch umfangreiche Möglichkeiten, das Programm zu kontrollieren und zu steuern.

Ein besonderes Merkmal ist das "Resource-Tracking", das beispielsweise Problem 3 löst. Beim Abbuch oder Ende eines Programms werden alle Ressourcen angezeigt, die durch das Programm reserviert, aber nicht freigegeben wurden. Die Ressourcen werden dann durch die "Runshell" freigegeben. Die Quelltextstelle, an der die jeweilige Ressource reserviert wurde, kann dabei auf einfachste Weise angesprungen werden.

Angeschlossen an diese "Runshell" ist der "Source-Level-Debugger". Mit diesem kann man ein Programm schrittweise abarbeiten, um Fehler zu lokalisieren, oder Variablenwerte zu kontrollieren. Aber auch, um direkt Einfluß auf das Programm zu nehmen und jederzeit eine Problemstelle (im härtesten Fall ein Absturz) im Quelltext zu lokalisieren.

Ebenso ist ein "harter Abbruch" des Programms möglich (z.B. in Endlosschleifen), ohne das Programm in einem speziellen Modus kompilieren zu müssen.

Überhaupt ist dieses eine Maxime in der Testphase unter StormC: Das Programm ist in der Testphase identisch zum endgültigen Produkt. Um den "Source-Level-Debugger" verwenden zu können, muß zwar mit einer entsprechenden Option kompiliert werden, diese hat aber keinerlei Einfluß auf das erzeugte Programm. Damit wird vermieden, daß ein Programm im Normallauf einen Fehler zeigt, den es im Debugger nicht hat, wie es bei anderen Systemen immer wieder vorkommt.

Eine weitere Besonderheit ist die Möglichkeit, jederzeit aus dem normalen Programmlauf in den Debugger wechseln zu können oder wieder aus dem Debugger in den normalen Programmlauf zurückzukehren, sofern das Programm mit der Debug-Option kompiliert wurde.

Ein Beispiel zum Resource-Tracking

Öffnen Sie das Projekt "IllResource.¶" in der Schublade "StormC:Examples/IllResource". Dieses kleine Programm reserviert einige Ressourcen, gibt sie aber nur zum Teil wieder frei.

Starten Sie dieses Programm z.B. mit der Funktionstaste <F9>. Daraufhin wird das Programm bei Bedarf neu kompiliert und dann gestartet.

Weitere Möglichkeiten des Programmstarts sind: der Druck auf den Knopf mit dem laufenden Männchen in der Toolbar, die Auswahl des Menüpunktes "Starten" aus dem Menü "Kompilieren", der Doppelklick auf den Programmeintrag im Projektfenster (dabei wird das Program nie neu kompiliert) oder direkt nach dem Kompilieren aus dem Fehlerfenster mit dem Knopf "Starten".

Nach dem Kompilieren und Start des Programms wird das Kontrollfenster der "RunShell" geöffnet. Dies ist Ihr "Steuerpult" für das Programm.

Die Statuszeile zeigt Ihnen kurze Hilfetexte zu jedem Element des Kontrollfensters und außerdem den Status Ihres Programms, wenn Sie sich auf keinem Oberflächen-element befinden.

Das Programm "IllResource" zeigt Ihnen jetzt an, daß es wartet (Funktion "Wait()" der "exec.library") und zusätzlich die Maske der Signalflags, auf die gewartet wird. Momentan steht hier ein "0x00001000", das entspricht einem <Ctrl>+<C>.

Die Symbole in der Gruppe "Debugger" werden momentan nicht benötigt, denn zuerst soll das Programm normal arbeiten. Mit dem Auswahlsymbol im Rahmen läßt sich der Debugger einschalten, die darin angebrachten Symbole stehen für "Gehe", "Gehe einen Schritt", "Gehe einen Schritt, aber bearbeite Funktionsaufrufe schnell", "Gehe bis zum Ende der Funktion" und "Zeige die zuletzt ermittelte Programm-position". Diese Funktionen werden im weiteren Text bei der Debuggereinführung näher erläutert.

Programm kurzzeitig anhalten

Der Knopf "Pause" erlaubt, das Programm jederzeit anzuhalten, z.B. wenn Sie den Inhalt eines Fensters näher betrachten wollen, der sonst zu schnell wieder überschrieben würde. Dieser Knopf wird wie ein Druckschalter bedient, zuerst wird der Knopf "eingedrückt", dann bleibt er in diesem Zustand. Das Programm hält solange an, bis Sie den Knopf

wieder drücken und er "herausspringt". Das Programm läuft daraufhin weiter. Der Menüpunkt "Pause" im Menü "Debugger" hält das Programm ebenfalls an.

Programm hart abbrechen

Daneben liegt der Knopf "Kill". Diese martialische Aktion erlaubt, das Programm jederzeit zu beenden. Der Menüpunkt "Kill" im Menü "Debugger" hat die gleiche Wirkung.

Taskpriorität ändern

Der Gruppe der Prioritäteneinsteller zeigt die Priorität des Programms an und erlaubt, diese zu ändern. Obwohl man die Priorität über den ganzen Bereich von -128 bis 127 einstellen kann, sollte man sie im Bereich von -20 bis 20 lassen, es könnte sonst zu Schwierigkeiten mit anderen Programmen kommen, oder sogar zur Unbenutzbarkeit des Systems.

Der beim Programmstart eingestellte Wert (im Normalfall -1) ist insofern vernünftig, daß er um eins kleiner ist als die Priorität des Debuggers und somit immer ein zügiges Arbeiten im Debugger möglich ist.

Hinweis
   Insbesondere durch zu hohe Prioritätseinstellungen kann das System "lahmgelegt" werden. Bitte seien Sie vorsichtig mit der Ver nderung dieser Einstellungen.

Signale verschicken

Die Gruppe der Signalsymbole enthält vier Knöpfe, um die vier Unterbrechungssignale von AmigaDOS zu setzen:

   <Ctrl>+<C>, <Ctrl>+<D>, <Ctrl>+<E>, <Ctrl>+<F>.

Damit ist es sehr einfach möglich, in der Testphase zusätzliche "Wait()" Anweisungen einzukompilieren und das Programm damit zusätzlich zu steuern. Das Programm "IllResource" wartet eben gerade auf das Signal <Ctrl>+<C>.

Klicken Sie jetzt auf das Symbol "Ctrl-C" in der Gruppe der Signalsymbole.

Nun hat das Programm einige Ressourcen reserviert, aber dann nicht wieder freigegeben, das Programm ist beendet.

Obwohl das Resource-Tracking immer funktioniert und Ressourcen durch die "RunShell" freigegeben werden, kann die Quelltextstelle nur angezeigt werden, wenn das Programm mit Debugdateien kompiliert wurde.

Die Protokolliste zeigt einige Informationen bezüglich Ressourcen, die reserviert und nicht freigegeben wurden. Das betrifft einige Speicheranforderungen, Bibliotheken und eine Datei.

Doppelklicken Sie auf eine Zeile in der Protokoll-Liste und die Stelle im Quelltext, an welcher diese Ressource reserviert wurde, wird Ihnen angezeigt. Falls der Quelltext des Programms nicht sowieso schon im Editor angezeigt war, wird ein neues Editorfenster ge ffnet und der Quelltext geladen.

In einem "echten" Programm könnten Sie nun die passende Freigabe programmieren. An welcher Stelle das passieren müßte, hängt natürlich von ihrem Programm ab.

Um die Quelltextstelle zu einer fehlerhaften Reservierung anzeigen zu können, muß eine Bedingung erfüllt sein:

* Die entsprechende Betriebssystemfunktion muß direkt im Quelltext aufgerufen werden. Damit sind alle Aufrufe über Bibliotheken nicht brauchbar, die Betriebssystemfunktionen sollten also nicht über die "amiga.lib" und die darin enthaltenen Stub-Funktionen aufgerufen werden, sondern immer über die passenden "#pragma amicall" und "#pragma tagcall"-Aufrufe.

Geben Sie nun die Ressourcen mit dem Menüpunkt "Ressourcen freigeben" aus dem Menü "Debugger" frei, oder schließen Sie das Kontrollfenster. In beiden Fällen werden Sie noch gefragt ob zuvor die Ressourcen freigegeben werden sollen.

Verwendung des Debuggers

Laden Sie das Projekt "StormC:Examples/DebuggerTutorial/DebuggerTutorial.¶". Das Programm besteht aus zwei Modulen mit einer Headerdatei, gerade genug, um auch nicht ganz triviale Fälle beim Debuggen zu zeigen.

Starten Sie das Programm jetzt normal und machen Sie sich damit vertraut. Zuerst erscheint ein Eingabemenü mit drei Punkten:

* Eingabe einer Adresse,
* Ausgabe aller gespeicherten Adressen und
* Ende des Programms.

Wenn Sie eine neue Adresse eingeben wollen, so geben Sie 1 (<Return> nicht vergessen) ein. Dann geben Sie Name, Vorname, Straße und Wohnort einer Person ein. Wiederholen Sie das zwei- oder dreimal und schauen Sie sich dann durch Menüauswahl 2 die Adressen an. Beenden Sie danach das Programm.

Da man normalerweise nur eigene Programme debuggt sollten Sie sich jetzt noch den Quelltext anschauen, um die Funktionsnamen kennenzulernen.

Das Hauptprogramm steht im Modul "main.c". Es zeigt das Hauptmenü an und wertet die Menüauswahl aus. Je nach Eingabe wird eine Funktion aus dem Modul "address.c" aufgerufen.

Das Modul "address.c" enthält zuerst einige statische Funktionen, die für die anderen Funktionen benötigt werden. Dann folgen alle Funktionen zur Adressverwaltung.

Die Headerdatei "address.h" enthält die Datenstruktur "struct Address"
und die Prototypen aller Funktionen, die etwas mit den Adressen
anstellen, also insbesondere Ein- und Ausgabe. Diese Funktionen sind
im Modul "address.c" implementiert und werden durch das Modul
"main.c" aufgerufen.

Wenn Sie soweit Bescheid wissen, starten Sie das Programm erneut, allerdings
diesmal im Debugger, also mit <F10> oder dem passenden Piktogramm
aus der Toolleiste.

Weitere Möglichkeiten, ein Programm zu debuggen sind:

* der Menüpunkt "Debuggen" im Menü "Kompilieren",
* der Doppelklick auf den Programmnamen im Projektfenster mit
  gleichzeitig gedrückter <Alt>-Taste oder direkt nach dem
  Kompilieren aus dem Fehlerfenster mit Klick auf den Knopf
  "Debuggen".

Wieder wird zuerst das Kontrollfenster geöffnet, allerdings diesmal
mit eingeschalteter Debugger-Kontrolle. Das Programm läuft
automatisch bis zur ersten Stelle im Programm, für die auch
Quelltext existiert. Wenn Sie den Quelltext des Moduls "main.c"
nicht schon geladen haben, so geschieht das jetzt. Das Editorfenster
sieht dabei etwas anders aus als sonst: Links neben dem Text befindet
sich jetzt eine Spalte mit Unterbrechungspunkten.

Mit Unterbrechungspunkt wird jede Stelle im Quelltext bezeichnet,
an der das Programm anhalten kann, Breakpoint heißen die Stellen an
denen ein Programm wieder anhält, wenn es mit "Gehe" gestartet wurde
und mit voller CPU Geschwindigkeit abläuft.

In dieser Spalte können Sie mit der Maus einen Unterbrechungspunkt
setzen oder löschen. Außerdem zeigt diese Spalte, wie weit ein
einzelner Schritt im Programm führt, denn häufig führt ein Schritt
in C über mehrere Textzeilen hinweg (z.B. über Kommentare, Variablen-
deklarationen, lange Ausdr cke etc).

Die Position des Programms wird im Editor immer durch einen weißen Balken
angezeigt, der horizontal die ganze Zeile markiert.

Das Variablenfenster

Als nächstes wird noch das Fenster der Variablen ge öffnet.
Darin werden immer in drei Seiten alle Variablen angezeigt,
die das Programm zur Zeit ansprechen kann.

In der obersten Zeile des Fensters wird wieder eine kurze Hilfe
ausgegeben, die Ihnen sagt, welche Aufgabe die verschiedenen
Symbolknöpfe im Fenster haben.

Unter dieser Zeile befindet sich eine ganze Leiste von Knöpfen.
Links liegen 3 Kn öpfe mit denen Sie eine der 3 Variablenseiten
auswählen k nnen.

* Die 1. Seite enthält alle Funktionsparameter und lokale
  Variablen der aktuellen Funktion sowie die globalen Variablen
  des Moduls, das diese Funktion enthält.

* Die 2. Seite enthält alle globalen Variablen aller Module.
* Die 3. Seite enthält alle Variablen, die Sie überwachen wollen. Das können lokale Variablen jeder Funktion sein oder auch Teile einer Struktur, die Sie inspiziert haben.

Wählen Sie die erste Variable der aktuellen Variablen aus (address), indem Sie den Namen der Variablen anklicken. Daraufhin können Sie auch die weiteren Elemente im Fenster erreichen. Neben den Knöpfen zur Seitenauswahl liegen die Knöpfe zur Aktion auf der ausgewählten Variablen.

Durch Druck auf den 1. Knopf bekommen Sie die Quelltextstelle angezeigt, an der die Variable definiert ist, indem der Textcursor auf die Stelle plaziert wird.

Der 2. Knopf ist für C++ Programmierer interessant. Er zeigt eine Liste aller Member-Funktionen des Typs der Variablen. Im Beispiel existieren jedoch keine Member-Funktionen, so daß die Liste leer dargestellt wird.

Der 3. Knopf inspiziert die Variable. Dazu wird in der Variablenliste die Variable "aus der Nähe" betrachtet, z.B. wird jedes Feld einer Struktur einzeln aufgeführt. Der Doppelklick auf eine Variable hat die gleichen Auswirkung wie das Anwählen von Knopf 3.

Der 4. Knopf erlaubt wieder die Rückkehr aus der vorherigen Inspektion, der Inhalt der Variablenliste entspricht danach wieder dem Zustand vor der Inspektion.

Der 5. Knopf öffnet den Hexeditor und setzt den Cursor an die Adresse der Variablen.

Der 6. Knopf überträgt die Variable in die Seite der überwachten Variablen. Damit können Sie sich eine Liste von Variablen zusammenstellen, die Sie immer sehen wollen, auch wenn das Programm sich gerade nicht in dem dazugehörigen Modul oder der Funktion befindet.

Der 7. Knopf ist nur auf der Seite der überwachten Variablen aktiv und entfernt eine überwachte Variable aus dieser Liste.

Das Texteingabefeld "Cast" erlaubt ein einfaches Casting des Typs. In diesem Feld können Sie jeden Typ angeben, der im Modul irgendwo definiert wird. Achten Sie jedoch darauf, den Typ so zu schreiben, wie er in der Variablenliste auch ausgegeben wird. Wird der Typ gefunden, so wird der Wert der Variablen entsprechend des neuen Typs interpretiert. Zum Beispiel kann es sinnvoll sein aus einem ULONG ein LONG zu machen, um den Wert mit Vorzeichen angezeigt zu bekommen. Die häufigste Anwendung ist jedoch den Zieltyp eines Pointers zu ändern, z.B. aus "Message *" ein "IntuiMessage *". Bei der nächsten Inspektion dieser Variablen wird dan natürlich der neue Typ benutzt.

Der alte Typ wird immer noch in der Variablenliste angezeigt, der neue Typ nur im Texteingabefeld "Cast". Wollen Sie zum alten Typ zurückkehren, brauchen Sie nur das Texteingabefeld zu löschen und <Return> drücken.

Haben Sie sich einmal mit einem Typ vertippt, so blinkt der Bildschirm

und die alte Eingabe bleibt unverändert stehen.

Wertänderung

Wenn Sie eine Zahlvariable ausgewählt haben (hier zum Beispiel die dritte Variable illegal), können Sie im Texteingabefeld "Wert" auch den Wert der Variablen direkt ändern, ohne den Hexeditor bemühen zu m ssen.

Haben sie eine Variable ausgewählt deren Typ ein Pointer auf ein UBYTE oder BYTE ist (gemeinhin auch "String" genannt), so können Sie den Strings im Texteingabefeld "Wert" ändern. Achten Sie jedoch darauf, nicht mehr Zeichen einzugeben, als Speicherbereich für diesen String zur Verfügung steht.

Variablensortierung

Die Variablenliste wird zu Anfang unsortiert dargestellt. Sie können mit dem Cyclegadget "Sort" diese Sortierung umstellen. Die alphabetische Sortierung benutzt den Namen der Variablen zur Sortierung, damit lassen sich bei vielen Variablen (z.B. der Liste aller globalen Variablen) schnell bestimmte Variablen finden. Die Sortierung "Letzte Änderung" stellt immer die Variable an den Anfang der Liste, die zuletzt geändert wurde. Diese Sortierung wird nicht nur bei der Auswahl dieses Sortiermodus durchgeführt, sondern nach jedem Programmschritt. Dadurch sehen Sie immer, welche Variable zuletzt geändert wurde. Der Debugger erkennt dabei jede Änderung, also auch Änderungen einzelner Strukturfelder oder Variablen, die über den Umweg eines Pointers verändert wurden.

Die Sortierung können Sie für die 3 Variablenseiten getrennt einstellen. Arbeiten Sie nun einige Befehle des Programms ab, indem Sie das mittlere Symbol in der Gruppe der Debuggersymbole klicken. An Ihrem Editorfenster sehen Sie, wie das Programm eine "printf" Funktion nach der anderen abarbeitet, die Ausgabe geschieht passend auf dem Konsolefenster. Die Statuszeile im Kontrollfeld zeigt Ihnen nach jedem Schritt "Breakpoint erreicht" an. Während Sie einen Schritt gehen und dieser Schritt länger dauert z.B. bei einer Eingabe von der Konsole, können Sie auch die Meldung "Programm wird fortgesetzt" lesen.

Irgendwann erreichen Sie die Funktion "gets(s)" in Zeile 25. Wenn Sie diese Funktion abarbeiten, müssen Sie natürlich eine passende Eingabe im Konsolefenster machen.

Bitte geben Sie <1> und <Return> ein. In der switch-Anweisung laufen Sie nun in den ersten Zweig hinein. In Zeile 29 steht der Funktionsaufruf der Funktion "readAddressmask()". Diese Funktion steht im Modul "address.c". Wählen Sie nun nicht den Knopf zum schnellen Bearbeiten von Funktionen weiter, sondern wählen Sie den zweiten Knopf von links, um das Programm in einzelnen Schritten abzuarbeiten. Dadurch "betreten" Sie die Funktion "readAddressmask()".

Der Quelltext des Moduls "address.c" wird geladen und die Quelltext-position wieder angezeigt. Gehen Sie nun Schritt für Schritt durch die Funktion. Je nachdem, ob Sie den 2. oder 3. Knopf in der Grupper der Debugger-Symbole im Kontrollfenster benutzen, werden die weiteren Unterprogramme schnell abgearbeitet oder Sie durchlaufen auch diese

schrittweise.

Machen Sie bei Bedarf Eingaben für Name, Vorname, Straße und Wohnort im Konsollfenster. Wählen Sie im Variablenfenster als Sortierung für die aktuellen Variablen "Letzte Änderung" und sehen Sie, wie immer die Variable des zuletzt eingegebenen Elements an oberste Stelle rückt.

Bevor Sie die Funktion beendet haben, können Sie den 4. Knopf von links im Debuggerrahmen ausprobieren. Dieser läßt das Programm relativ schnell weiterlaufen, bis das Ende der Funktion erreicht ist und Sie wieder ins Hauptprogramm gelangen.

Dort sollten Sie auf Zeile 30 stehen und die Variable "address" sollte einen sinnvollen Wert anzeigen. Diese können Sie nun näher betrachten. Wählen Sie die Variable im Fenster der aktuellen Variablen aus und drücken Sie den Knopf zum Inspizieren.

Die Variablenliste wird ersetzt durch die Elemente der Struktur "Address". Sie können die eingegebenen Strings auch jetzt noch ändern. Wählen Sie dazu z.B. das Feld "name" aus und ändern Sie den Wert dann im Texteingabefeld "Wert". In diesem Fall sollten Sie einen höchstens gleich langen String angeben, denn das Programm hat die Strings im dynamischen Speicher mit "malloc()" reserviert und nur soviel Platz wie nötig reserviert.

Setzen Sie nun die Fehlersuche fort und probieren Sie ein paar Möglichkeiten aus, z.B. das Überwachen von Variablen und Struktur-feldern. Wenn Sie sich einigermaßen "wohl fühlen" im Debugger, beenden Sie das Programm, wahlweise korrekt oder mit "Kill".