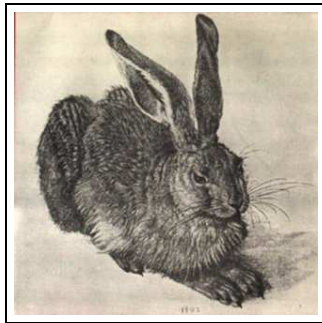


Rabbit



Franz Brandel
Institut für Höhere Studien

Version 0.a

Inhaltsverzeichnis

Einführung	xi
I Syntax	1
1 Lexikalische Elemente	3
1.1 Kommentare	3
1.2 Variablen	3
1.3 Funktionsbezeichner	3
1.4 Reservierte Wörter	4
1.5 Operatoren	4
2 Datentypen	5
3 Deklarationen	7
3.1 Variablendeklaration	7
3.2 Funktionsdeklaration	7
4 Ausdrücke	9
5 Statements	11
5.1 Block	11
5.2 FOR - Statement	11
5.3 IF - Statement	11
5.4 Funktionsaufruf	13
5.5 RETURN - Statement	13
5.6 Anweisung	13
II Library Interface	15
6 Vorwort	17
7 Die Standard Lib	19
7.1 print	19
7.2 inputi	19
7.3 inputf	20
7.4 inputs	20

8	Rabbit und Curses	21
8.1	addstr();	21
8.2	box();	21
8.3	clear();	22
8.4	endwin();	22
8.5	initscr();	22
8.6	move();	23
8.7	newwin	23
8.8	set_colors	23
8.9	wprintw();	24
9	Die Unix Lib	25
9.1	open	25
9.2	close	26
9.3	write	26
10	Die Bargaining Lib	27
10.1	game_label	27
10.2	game_window	27
10.3	game_clear	28
10.4	game_print	29
10.5	game_printfile	29
10.6	game_input	30
10.7	game_inittimer	31
10.8	game_goto	31
10.9	game_waitfor	32
10.10	game_sync	32
10.11	game_ifthenelse	33
10.12	game_start	33
10.13	game_end	34
10.14	game_serverwait	34
10.15	game_intervar	35
10.16	game_setvar	35
10.17	game_evalinter	36
10.18	game_random	36
10.19	game_comment	37

Verzeichnis der Beispiele

1.1	Einfaches Beispiel	3
3.1	Variablendeklarationen	7
3.2	Funktionsaufrufe	8
4.1	Booleansche Werte 1	9
4.2	Booleansche Werte 2	10
4.3	Parameterübergabe	10
5.1	FOR - Statement	13
5.2	IF - Statement	13
5.3	Funktionsaufrufe	14
5.4	RETURN Statement	14
5.5	Anweisungen	14
7.1	print	19
7.2	inputi	19
7.3	inputf	20
7.4	inputs	20
8.1	addstr	21
8.2	box	22
8.3	clear	22
8.4	endwin	22
8.5	initscr	23
8.6	move	23
8.7	newwin	23
8.8	newwin	24
8.9	wprintw	24
10.1	game_label	27
10.2	game_window	28
10.3	game_clear	28
10.4	game_print	29
10.5	game_printfile	30
10.6	game_input	30
10.7	game_inittimer	31
10.8	game_goto	32
10.9	game_waitfor	32
10.10	game_sync	33
10.11	game_ifthenelse	33
10.12	game_start	34
10.13	game_end	34
10.14	game_serverwait	34
10.15	game_setvar	35

10.16game_setvar	36
10.17game_evalinter	36
10.18game_random	37
10.19game_comment	37

Tabellenverzeichnis

1.1	Operatoren	4
2.1	Datentypen	5
9.1	Access Parameter	25
10.1	game_label	27
10.2	game_window	28
10.3	game_clear	28
10.4	game_print	29
10.5	game_printfile	29
10.6	game_input	30
10.7	game_timer	31
10.8	game_goto	31
10.9	game_ifthenelse	33
10.10	game_start	33
10.11	game_intervar	35
10.12	game_intervar	35
10.13	game_evalinter	36
10.14	game_random	37
10.15	game_comment	37

Abbildungsverzeichnis

5.1	FOR - Statement	12
5.2	IF - Statement	12

Einführung

Dieser Text ist das Referenzmanual des *Rabbit*. Der Name *Rabbit* leitet sich von den Wörtern *rapid* und *prototyping* ab. Ziel bei der Konzeption dieser Sprache war es, eine einfache Syntax zu entwerfen, die auch von Anwender geschätzt wird, deren hauptsächliche Tätigkeit nicht im Bereich Programmierung liegt. Ein weiteres Ziel lag in der Schaffung der Möglichkeit selbst entwickelte oder fremde Libraries mit einem Minimum an Programmieraufwand in *Rabbit* zu integrieren. Teil II dieses Textes geht darauf ausführlich ein. Beispiele für solche Libraries sind die SV-Lib (IHS), *curses*, etc. *Rabbit* ist eine prozedurale Sprache, mit drei skalaren Datentypen.

Ausführung

Ein *Rabbit*-Programm wird durch den gleichnamigen Interpreter ausgeführt.

```
rabbit <Rabbit-Programm>
```

Wird kein Programm angegeben wird, wird versucht ein Programm mit Namen `rabbit.default` auszuführen, wenn es das nicht gibt, ein Fehler ausgegeben und *Rabbit* beendet.

Die Rabbit-Syntax wird eingeteilt in:

- Strukturelementen(Syntax)
- Librarycalls

Dieser Einteilung folgt auch die Gliederung dieses Textes.

Konventionen

Alle lexikalischen Elemente, wie auch Codebeispiele werden *verbatim* dargestellt. Text zwischen spitzen Klammern bedeutet, das hier etwas eingesetzt werden *muß*.

```
<hier_den_Namen_einsetzen>
```

Text in eckigen Klammer bedeutet, daß hier etwas eingesetzt werden *kann*.

```
[Wenn_Sie_wollen_setzen_Sie_Ihren_Namen_ein]
```


Teil I
Syntax

Kapitel 1

Lexikalische Elemente

1.1 Kommentare

Das # - Zeichen bildet den Kommentar in Rabbit. Alles, was dahinter steht wird ignoriert.

Beispiel 1.1 Einfaches Beispiel

```
# Code Fragment:  
  
# Das ist ein Kommentar  
print ("Hello");      # wird exekutiert  
# Dieses nicht: print ("World!");  
#  
...
```

1.2 Variablenbezeichner

Alle Variablen werden mit GROSSBUCHSTABEN bezeichnet. Zulässige Variablen-
namen sind zB:

VAR	V	VARIABLE
STR1	STR	I

1.3 Funktionsbezeichner

Funktionsbezeichner werden mit Kleinbuchstaben und dem Zeichen `_` bezeichnet
zB:

```
print  
curses_addstr
```

1.4 Reservierte Wörter

Folgende Wörter sind Bestandteil der Grammatik, und dürfen daher nicht als Variablen- oder Funktionsbezeichner verwendet werden:

FUNC	RETURN
FOR	IF
ELSE	DECLARE
ENDIF	ENDFOR
ENDFUNC	INT
FLOAT	STRING
AND	NOT
OR	

1.5 Operatoren

Tabelle 1.1 Operatoren

:=	Zuweisung	=	Gleichheit
<>	Ungleichheit	<	kleiner als
>	größer als	<=	kleiner oder gleich
>=	größer oder gleich	+	Addition
-	Subtraktion	*	Multiplikation
/	Division	(Klammer auf
)	Klammer zu	NOT	logische Negation
OR	logisches "oder"	AND	logisches "und"

Kapitel 2

Datentypen

Rabbit unterstützt drei skalare Datentypen.

Tabelle 2.1 Datentypen

Type	Bezeichner	Beispiel
Integer	INT	-2, -1, 0, 1, 2, ...
Gleitkomma	FLOAT	-1.45, 0.0, 789, 4.23 ...
Zeichenkette	STRING	"String", "Hansi", "Fred", ...

Kapitel 3

Deklarationen

Eine Deklaration legt die Zuordnung eines Namens mit einem Objekt fest. Objekte sind:

- Variablen
- Funktionen

3.1 Variablendeklaration

Variablen können an verschiedenen Stellen in einem Programm definiert werden. Sie haben erst Gültigkeit wenn sie deklariert worden sind. Sie werden mittels des Schlüsselwortes DECLARE deklariert.

Format

```
DECLARE <name> INT | FLOAT | STRING ;
```

Beispiel 3.1 Variablendeklarationen

```
# Codefragment
#
DECLARE GANZZAHL INT      ; # Ab hier existiert die
                          # Integervariable "GANZZAHL"
DECLARE ZEICHEN  STRING ; # Der String "ZEICHEN"
DECLARE GLEIT    FLOAT   ; # Der Float "GLEIT"
...

```

3.2 Funktionsdeklaration

Funktionen werden mit dem Schlüsselwort FUNC deklariert. Optional kann der Rückgabewert der Funktion vorangestellt werden.

Format

```
[INT|FLOAT|STRING] FUNC <funcname>([[parameter],...])
# Code
....
ENDFUNC
```

Beispiel

Das folgende Beispielprogramm enthält eine Funktion, die keine Übergabeparameter erwartet und keine Werte zurückliefert (main), und eine Funktion square, die sowohl einen Wert nimmt, wie auch retour liefert.

Beispiel 3.2 Funktionsaufrufe

```
# Ein korrektes Programm
# (c) Institute for Advanced Studies Vienna
# Franz Brandel
# -----
FUNC main ( )          # Liefert keinen Wert,
                       # nimmt keine Argumente

    DECLARE Z FLOAT;

    Z := square ( 4.0 );
                       # Dazu braucht man
                       # keinen Computer!

ENDFUNC

# -----

FLOAT FUNC square(ZAHL FLOAT)
                       # Nimmt Float und liefert
                       # wieder einen
    RETURN ZAHL * ZAHL ;
ENDFUNC
```

Kapitel 4

Ausdrücke

Ausdrücke werden aus Konstanten, Variablen, Operatoren und Funktionsaufrufen gebildet. Ein Ausdruck evaluiert immer zu einem skalaren Datentyp.

Konstanten

Integerkonstanten kommen aus der Menge der ganzen Zahlen und werden daher dargestellt:

```
-2 -1 0 1 2 ...
```

Float-Konstanten müssen mit einem Dezimalpunkt versehen werden, um wirklich identifizierbar zu sein:

```
-2.0 -1.0 0.0 1.0 2.0 -233.654 ...
```

Strings müssen in Anführungsstrichen gesetzt werden:

```
"Rabbit" "Institute for Advanced Studies - Vienna"
```

Booleansche Werte

Ein Boolescher Wert evaluiert zu 1 wenn wahr, und zu 0 wenn falsch.

Im folgenden Beispiel seien `VARFLOAT` und `VARINT` Float- bzw. Integervariablen.

Beispiel 4.1 Booleansche Werte 1

```
3*4+6*90.09
VARFLOAT * 6
VARINT
```

Die folgenden Ausdrücke evaluieren zu `-5` bzw `0`:

Beispiel 4.2 Booleansche Werte 2

```
-5*(1=1)
7*NOT(1=1)
```

Typkonversion

Zur Typkonversion gelten folgende Regeln:

Regel 4.1 *Wenn zwei numerische Typen mit einem Operator verknüpft werden, und einer oder beide ein `FLOAT` ist(sind), ist das Result ein `FLOAT` sonst ein `INT`. Wenn ein `STRING` mit einem numerischen Typen verknüpft wird, wird ein Fehler ausgegeben. Zulässig ist eine Addition von Strings, die in einer Verkettung der Strings besteht.*

!! FEHLERALARM !!

In der gegenwärtigen Version ist es sehr wichtig, daß der des Übergabeparameters einer Funktion, mit der Deklaration des Arguments übereinstimmt.

Beispiel 4.3 Parameterübergabe

```
# (c) Institute for Advanced Studies - Vienna
# Franz Brandel
# -----
FUNC main ()
  DECLARE Z FLOAT;
  Z := add ( 2, 3 ) ;      # Liefert 0 !!!
  Z := add ( 2.0, 3.0 ) ; # Liefert 5
ENDFUNC
# -----
FLOAT FUNC add ( A FLOAT, B FLOAT )
  RETURN A + B ;
ENDFUNC
#
```

Kapitel 5

Statements

Es stehen Statements zu Verfügung:

1. Block
2. FOR- Statement
3. IF- Statement
4. Funktionsaufruf
5. RETURN- Statement
6. Anweisung

5.1 Block

Ein Block ist eine Sequenz von einem oder mehreren Statements.

5.2 FOR - Statement

Das For - Statement wird verwendet um Programmteile, in Abhängigkeit von einem Ausdruck, wiederholt auszuführen.

Format

```
FOR ( [statement1] <expression> ; [statement2] )  
  <BLOCK>  
ENDFOR
```

5.3 IF - Statement

Das IF - Statement stellt eine Programmverzweigung in Abhängigkeit des Ausdruckes dar. Wenn der Ausdruck wahr ist, wird Statement 1 ausgeführt, sonst Statement 2, sofern vorhanden.

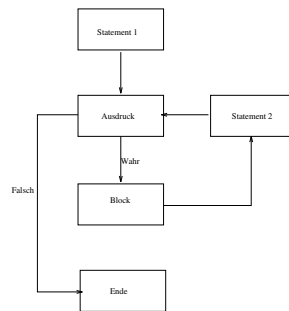


Abbildung 5.1: FOR - Statement

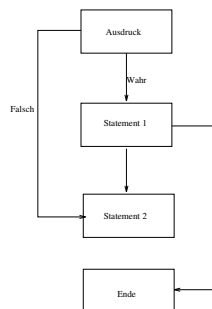


Abbildung 5.2: IF - Statement

Beispiel 5.1 FOR - Statement

```

# (c) Institute for Advanced Studies - Vienna
# Franz Brandel
# Zu I wird 1 addiert bis I = 11 (sic!) ist
# Jeder Wert von 1 bis 10 wird ausgegeben
#
FOR ( I := 1; I <= 10; I := I + 1 )
    print ( I );
ENDFOR
#

```

Beispiel 5.2 IF - Statement

```

# (c) Institute for Advanced Studies - Vienna
# Franz Brandel
# Beispiel eines IF - Statements

FUNC main ()

    DECLARE I INTEGER;

    I = inputi(); # Erwartet eine Eingabe

    IF ( I > 0 )
        print ( I, "ist groesser als 0" );
    ELSE
        print ( I, "ist kleiner oder gleich 0");
    ENDIF

ENDFUNC

```

5.4 Funktionsaufruf

Ein Funktionsaufruf besteht aus dem Namen der Funktion und ihren Parametern in Klammern. Wenn keine Parameter übergeben werden sollen, müssen leere Klammern hinten angefügt werden.

5.5 RETURN - Statement

Das RETURN-Statement dient der Rückgabe eines Wertes von einer Funktion. *Achtung!* Es ist auf die Übereinstimmung der Typen zu achten.

5.6 Anweisung

Mit dem Operator := weist man einer Variablen einen Wert zu. Auf der rechten Seite muß ein Ausdruck stehen.

Beispiel 5.3 Funktionsaufrufe

```
# Ein komplettes Programm
# Institute for Advanced Studies - Vienna
# Franz Brandel
FUNC main ()
    druck_string ( "Hello!");
    print_nonsense ();
ENDFUNC
# -----
FUNC druck_string ( S STRING )
    print ( S );
ENDFUNC
# -----
FUNC print_nonsens ()
    print ("Nonsense!");
ENDFUNC
```

Beispiel 5.4 RETURN Statement

```
# Codefragment
#
INT FUNC give_number ()
    DECLARE RET INT;
    RET = input ();
    RETURN RET;
ENDFUNC
```

Beispiel 5.5 Anweisungen

```
DECLARE I INT;
DECLARE S STRING;
DECLARE F FLOAT;
I := 12*89;
S := "Hello World";
F := getfloat();
```

Teil II

Library Interface

Kapitel 6

Vorwort

Nach dem Studium der letzten Kapitel erkennt man sehr schnell, daß Rabbit allein jegliche Möglichkeit fehlt um zum Beispiel das Ergebnis einer Berechnung auf dem Schirm auszugeben auszugeben. Selbstverständlich gibt es diese Möglichkeit. Diese und viele andere Funktionen sind in Zusatzlibraries implementiert. Funktionen aus einer Library werden genau so aufgerufen, wie selbst geschriebene Unterprogramme.

```
libfunc ( [arg1], ... )
```

oder als Funktionen

```
Var1 = libfunc ( [arg1], ... )
```

Verfügbare Libraries

Standard Lib Sie enthält rudimentäre Funktionalität zur Schirmaus- und eingabe.

Curses Curses ist eine der ältesten und am meisten verbreiteten Libraries in der Unix - Welt. Sie bietet eine (fast) standardisierte Möglichkeit zur Terminalprogrammierung. Es empfiehlt sich curses zu verwenden um Fenster-technik zu realisieren.

SV - lib Diese Library kann verwendet werden, um Beschäftigungsverlaufsdaten darzustellen und Statistiken darüber zu bilden.

Unix Sie enthält ein Interface zu den wichtigsten Unix - Systemcalls und Libraryfunktionen, die im Zusammenhang mit anderen Libs hoffentlich von Nutzen sind.

Kapitel 7

Die Standard Lib

In ihrer Kürze kann sie dennoch wertvolle Dienste leisten.

7.1 print

Druckt das oder die angegebenen Argumente am Schirm aus. **Achtung!** Es ist eine variable Anzahl von Argumenten möglich. Print erkennt bei jedem Argument den Typ automatisch und stellt ihn entsprechend dar.

Synopsis

```
print ( [arg1,... ] );
```

Beispiel 7.1 print

```
print ( 133, 67.0, -5, "Happy Hour", COUNTER );
```

7.2 inputi

Wartet auf eine Eingabe über die Tastatur und gibt diese als INTEGER zurück.

Synopsis

```
INTEGER inputi ();
```

Beispiel 7.2 inputi

```
DECLARE I INTEGER;
```

```
I = inputi ();
```

7.3 **inputf**

Wartet auf eine Eingabe über die Tastatur und gibt diese als FLOAT zurück.

*

Synopsis

```
    FLOAT inputf ( );
```

Beispiel 7.3 *inputf*

```
    DECLARE F FLOAT;
```

```
    F = inputf ( );
```

7.4 **inputs**

Wartet auf eine Eingabe über die Tastatur und gibt diese als STRING zurück.

Synopsis

```
    STRING inputs ( );
```

Beispiel 7.4 *inputs*

```
    DECLARE S STRING;
```

```
    S = inputs ( );
```

Kapitel 8

Rabbit und Curses

Die Curses - Library ist seit vielen Jahren integraler Bestandteil des Unix - Betriebssystems. Eine genaue Spezifikation findet sich in [1]. Bei der Integration von Curses in Rabbit wurde darauf geachtet, daß die Prozeduraufrufe, wie auch die Parametrisierung möglichst gut übereinstimmen.

8.1 addstr();

Gibt einen String am Schirm aus.

Synopsis

```
addstr( S STRING );
```

Beispiel 8.1 addstr

```
DECLARE ST STRING;
```

```
ST = "Hallo";
```

```
addstr ( ST );
```

```
addstr ( "I am a Rabbit!" );
```

8.2 box();

Diese Funktion setzt einen Rahmen um ein Fenster. Der erste Character des Strings vert wird vertikal, der erste Character des Strings hor wird horizontal als Rahmen gesetzt.

Synopsis

```
box ( vert STRING, hor STRING );
```

Beispiel 8.2 `box`

```
box ( HANDLE , "|", "-" );
```

Erzeugt folgenden Fensterrahmen.

```
-----  
|           |  
|           |  
|           |  
-----
```

8.3 `clear()`;

Schirm löschen.

Synopsis

```
clear();
```

Beispiel 8.3 `clear`

```
clear();
```

8.4 `endwin()`;

Curses *muß* mit diesem Befehl beendet werden.

Synopsis

```
endwin();
```

Beispiel 8.4 `endwin`

```
endwin();
```

8.5 `initscr()`;

Curses initialisieren. Diese Funktion muß die *erste* Funktion aller verwendeten Curses - Calls sein.

Synopsis

```
initscr();
```

Beispiel 8.5 initscr

```
initscr();
```

8.6 move();

Mit dieser Funktion wird die Position für die nächste Schirmein- und ausgabe festgelegt.

Synopsis

```
move( Y INTEGER, X INTEGER);
```

Beispiel 8.6 move

```
move( 14, 7);
```

```
# Setzt den Curser auf Zeile 14 Spalte 7.
```

8.7 newwin

Mit newwin erzeugen Sie ein neues Fenster am Schirm. Als Rückgabewert erhalten Sie ein "Windowhandle". Es dient zur Identifikation des Window in weiteren Funktionen. *Verändern Sie dieses Handle nie!*

Synopsis

```
STRING newwin( lines INTEGER,  
               cols INTEGER,  
               y INTEGER,  
               x INTEGER );
```

Beispiel 8.7 newwin

```
DECLARE HANDLE STRING;
```

```
HANDLE = newwin ( 10, 10, 5, 40 );  
box ( HANDLE, "-", "|" );
```

8.8 set_colors

Mit set_colors setzt man die Farbe der Schirmausgabe um.

Synopsis

```
set_colors( foreground STRING,
            background STRING );
```

Farbe	Code
blau	BLUE
rot	RED
grün	GREEN
gelb (braun)	YELLOW
schwarz	BLACK
weiß	WHITE
magenta	MAGENTA
cyan	CYAN

Tabelle 8.1: Verfügbare Farben

Beispiel 8.8 `newwin`

```
set_colors ( "BLUE", "WHITE" );
           # Schreibt ab jetzt blau auf weisz
```

8.9 `wprintw()`;

Druckt Werte im spezifizierten Fenster aus.

Synopsis

```
wprintw ( win STRING, val1 STRING |
          INTEGER |
          FLOAT, ... );
```

Beispiel 8.9 `wprintw`

```
wprintw ( WINHANDLE, 123.0, 89, "HALLO" );
```

Kapitel 9

Die Unix Lib

Verfügbare Libs

open Öffnen von Dateien

close Schließen von Filedescriptor

write Schreiben auf Filedescriptor

read Lesen von Filedescriptor

connect Initiieren von Interprozesskommunikationskanälen (IPC)

accept Annehmen von IPC - Kanälen

feof Testen ob Dateiende erreicht

9.1 open

Öffnen einer Datei. Der Rückgabewert stellt ein Handle dar, das der weiteren Identifikation der Datei dient. Verändern Sie dieses Handle nie! Der `access` - Parameter steuert die Zugriffsmethode auf die Datei.

Tabelle 9.1 Access Parameter

"r"	Lesen
"w"	Schreiben, ein bestehendes File wird überschrieben
"a"	Anfügen an ein bestehenes oder neues File
"r+"	Lesen und Schreiben von/auf einem bestehenden File
"w+"	Erzeugen eines neuen File zum Schreiben/Lesen
"a+"	Anfügen, erzeugen eines Files zum Schreiben/Lesen

Synopsis

```
INTEGER open ( path STRING, access STRING );
```

Beispiel

```
DECLARE HANDLE INTEGER;

HANDLE = open ( "/users/fb/etc/phonebook", "r" );
# fb's privates Telefonbuch oeffnen
```

9.2 close

Die mögliche Anzahl offener Filedescriptoren hängt vom Betriebssystem ab, auf dem Rabbit läuft. Somit sollte man darauf achten, daß nur die Dateien offen sind, die auch wirklich gebraucht werden. POSIX schreibt als Minimum 16 vor, in der Regel kann aber mit 256 rechnen.

Synopsis

```
close ( HANDLE integer );
```

Beispiel

```
DECLARE HANDLE INTEGER;

HANDLE = open ( "/users/fb/etc/phonebook", "r" );
# fb's privates Telefonbuch oeffnen
# Lesen/schreiben von/auf Telefonbuch.
close (HANDLE);
# und wieder schliessen
```

9.3 write

Dies ist eine Funktion mit variabler Zahl Argumenten. Es können beliebige Anzahl und beliebiger Typ, auch gemischt, angegeben werden.

Synopsis

```
write(HANDLE INTEGER, WERT INTEGER|STRING|FLOAT, ... );
```

Beispiel

```
DECLARE HANDLE INTEGER;

HANDLE = open ( "/users/fb/etc/phonebook", "r" );
# fb's privates Telefonbuch oeffnen
write( HANDLE, "Hallo Welt!", 1234, 89.99 );
# Schreibt zuerst den String, dann den Integer
# und dann den Float
close (HANDLE);
# und wieder schliessen
```

Kapitel 10

Die Bargaining Lib

Vorwort

Die Bargain Library

10.1 game_label

Definiert ein label. Dies ist eine mögliche Sprungstelle für ein goto.

Tabelle 10.1 game_label

"labelname"	Name des labels
-------------	-----------------

Synopsis

```
game_label ( <NAME> STRING ) ;
```

Beispiel 10.1 game_label

```
# Beispiel eines game_label library Aufrufes  
# (c) Institut für here Studien - Wien
```

```
game_label ( "mylabel" ) ;
```

Siehe auch: goto

10.2 game_window

Definiert ein curses window und speichert dieses unter einer ID ab. **Zeilen- und Spaltennummern beginnen mit 0.**

Tabelle 10.2 game_window

“id”	ID unter der das Window wieder ansprechbar ist
“y”	Zeile der linken oberen Ecke
“x”	Spalte der linken oberen Ecke
“lines”	Anzahl Zeilen
“cols”	Anzahl Spalten

Synopsis

```
game_window( <ID> INT, <Y> INT, <X> INT, <LINES> INT, <COLS> );
```

Beispiel 10.2 game_window

```
# Beispiel eines game_window library Aufrufes
# (c) Institut fr here Studien - Wien

game_window ( 1, 0,0, 24, 80 ) ;
# Erzeugt ein Window von der linken oberen Ecke mit
# 80 Spalten und 24 Zeilen.
```

Siehe auch: clear, print, printfile, input

10.3 game_clear

Löscht den Inhalt des angegebenen Fensters, nicht das Fenster selbst.

Tabelle 10.3 game_clear

“id”	ID des Fensters, dessen Inhalt gelöscht werden soll.
------	--

Synopsis

```
game_clear ( <ID> INT );
```

Beispiel 10.3 game_clear

```
# Beispiel eines game_clear lib - Aufrufes
# (c) Institut fr here Studien - Wien

game_window ( 1, 0,0, 24, 80 ) ;
game_clear ( 1 );
```

Siehe auch window, print, printfile, input

10.4 game_print

Gibt einen Text im angegebenen Window, an der angegebenen Position aus.

Tabelle 10.4 game_print

"id"	Window ID
"y"	Zeile
"x"	Spalte
"text"	auszugebender Text

Synopsis

```
game_print ( <ID> INT, <Y> INT, <X> INT, <TEXT> STRING );
```

Beispiel 10.4 game_print

```
# Beispiel eines game_print lib - Aufrufes
# (c) Institut für here Studien - Wien
```

```
game_window ( 1, 0,0, 24, 80 ) ;
game_clear ( 1 );
game_print ( <1, 0, 0, "Hello World!" );
```

Siehe auch `window`, `clear`, `printfile`, `input`

10.5 game_printfile

Gibt den Inhalt einer Datei im angegebenen Window, an der angegebenen Position aus.

Tabelle 10.5 game_printfile

"id"	Window ID
"y"	Zeile
"x"	Spalte
"text"	Datei mit Text

Synopsis

```
game_print ( <ID> INT, <Y> INT, <X> INT, <FILENAME> STRING );
```

Siehe auch `window`, `clear`, `print`, `input`

Beispiel 10.5 game_printfile

```
# Beispiel eines game_printfile lib - Aufrufes
# (c) Institut fr hhere Studien - Wien

game_window ( 1, 0,0, 24, 80 ) ;
game_clear ( 1 );
game_printfile ( 1, 0, 0, "myfile" );
```

10.6 game_input

Diese Funktion erwartet eine Eingabe am Client und liefert das Ergebnis an den Server zurück, das Ergebnis für die Verwendung im Zusammenhang mit `game_evalinter` abspeichert. Der Maximalwert zwingt den Client Werte im Bereich 1 - $jmax_j$ einzugeben. Der Vergleichswert wird mit abgespeichert. Es dient der späteren automatischen Auswertung. Außerdem wird der Inhalt des Textfiles angezeigt, indem zweckmäßigerweise die Frage formuliert sein kann.

Tabelle 10.6 game_input

“windowid”	Window in der der Cursor positioniert werden soll
“y”	Zeile
“x”	Spalte
“max”	Maximalwert
“comp”	Vergleichswert
“file”	Textfile

Synopsis

```
game_input ( <ID> INT,
             <Y> INT, <X> INT, <MAX> INT,
             <COMP> STRING, <FILE> STRING );
```

Beispiel 10.6 game_input

```
# Beispiel eines game_input lib - Aufrufes
# (c) Institut fr hhere Studien - Wien

game_window ( 1, 0,0, 24, 80 ) ;
game_clear ( 1 );
game_input ( 1, 0, 40, 5, "542", "5834678 modulo 4312");
```

Siehe auch `window`, `clear`, `print`, `printfile`, `evalinter`

10.7 game_inittimer

Startet den Timer auf dem Client. Nach Ablauf der angegebenen Zeit wird der Programmcounter auf das angegebene Label gesetzt. Um einen Timer zu deaktivieren braucht man ihn nur auf den Wert Null setzen.

Tabelle 10.7 game_timer

"secs"	Anzahl Sekunden
"label"	Nach Ablauf der Zeit wird bei diesem Label fortgesetzt

Synopsis

```
game_timer ( <SECS> INT, <LABEL> STRING );
```

Beispiel 10.7 game_inittimer

```
# Beispiel eines game_timer lib - Aufrufes
# (c) Institut fr hhere Studien - Wien

game_window ( 1, 0,0, 24, 80 ) ;
game_clear ( 1 );

# 5 Sekunden mssen reichen :-
game_inittimer ( 5, "TOOLATE" );
game_input ( 1, 0, 40, 5, "var1", "542", "5834678 modulo 4312");

game_goto ( "OK" );

game_label("TOOLATE");
game_print ( 1, 1, 0, "Die Uhr ist abgelaufen." );
game_end() ;

game_label("OK");
# Timer wieder deaktivieren
game_inittimer ( 0, "TOOLATE" );
game_print ( 1, 1, 0, "Nicht schlecht!" );
```

Siehe auch `game_goto`, `game_label`

10.8 game_goto

Plaziert den Programmcounter auf das angegeben Label.

Tabelle 10.8 game_goto

"label"	Name des Labels.
---------	------------------

Synopsis

```
game_goto ( <LABEL> STRING );
```

Beispiel 10.8 `game_goto`

Siehe Beispiel `\verb+game_timer+`

Siehe auch `game_timer`, `game_label`

10.9 `game_waitfor`

Diese Funktion hält den Client so lange auf, bis er die Taste Enter gedrückt hat.

Synopsis

```
game_wait();
```

Beispiel 10.9 `game_waitfor`

```
# Beispiel eines game_wait lib - Aufrufes
```

```
# (c) Institut für hhere Studien - Wien
```

```
game_window ( 1, 0,0, 24, 80 ) ;
game_clear(1);
game_print ( 1, 0, 0, "Wenn Sie diesen Text nicht mehr"
game_print ( 1, 1, 0, "sehen knnen, drcken sie <ENTER>" );
game_waitfor();
game_clear(1);
```

10.10 `game_sync`

Mit `game_sync` können Sie den Programmfluß am Server solange aufhalten, bis alle Clients an diese Stelle im Programm gelangt sind. Sobald der letzte dieses Statement erreicht hat, werden die Programmcounter aller Clients auf den nächsten Befehl gesetzt.

Synopsis

```
game_sync();
```

Beispiel 10.10 game_sync

```
# Beispiel eines game_sync lib - Aufrufes
# (c) Institut fr hhere Studien - Wien
```

```
game_sync ();
```

10.11 game_ifthenelse

Bedingte Verzweigung im Ablauf. Der Ausdruck wird ausgewertet, wenn sie den Wert "wahr" erhält wird zum Label "then" gesprungen sonst zum Label "else", wobei dieses Label auch ein leerer String sein kann, dann wird einfach beim nächsten Statement fortgesetzt.

Tabelle 10.9 game_ifthenelse

"ausdruck"	arithmetischer Ausdruck
"then"	Label im Falle von TRUE
"else"	Label im Falle von FALSE

Synopsis

```
game_if( <AUSDRUCK> STRING, <THEN> STRING, <ELSE> STRING );
```

Beispiel 10.11 game_ifthenelse

```
# Beispiel eines einen game_if lib - Aufruf
# (c) Institut fr hhere Studien - Wien
```

```
game_ifthenelse ("F1-20 > !_F1-20", "Sieg", "Step2");
game_ifthenelse ("F1-20 <= !_F1-20", "TieSieg", "Verlierer");
```

Siehe auch

10.12 game_start

Startet die Bargainsession entsprechend ihrer zuvor erfolgten Definition.

Tabelle 10.10 game_start

"NUMCLIENTS"	Zahl der erwarteten Clients
--------------	-----------------------------

Synopsis

```
game_start ( NUMCLIENTS INTEGER) ;
```

Beispiel 10.12 *game_start*

```
# Beispiel eines game_start library Aufrufes
# (c) Institut fr hhere Studien - Wien
# startet eine Session mit 2 Clients
game_start ( 2 ) ;
```

Siehe auch: `goto`

10.13 *game_end*

Dieser Befehl beendet die Bargain - Session und gibt seitens des Client die Kontrolle zurück an die den Client aufrufende Software (üblicherweise eine Shell) zurück.

Synopsis

```
game_end();
```

Beispiel 10.13 *game_end*

```
# Beispiel eines game_end lib - Aufrufes
# (c) Institut fr hhere Studien - Wien

game_end ();
```

10.14 *game_serverwait*

`Server_wait` synchronisiert alle Clients an dieser Stelle und wartet dann auf die Eingabe von ENTER des Operators. Anschließend setzt der Scheduler wieder mit der Abarbeitung des Programmes fort.

Synopsis

```
game_serverwait();
```

Beispiel 10.14 *game_serverwait*

```
# Beispiel eines game_serverwait lib - Aufrufes
# (c) Institut fr hhere Studien - Wien

game_serverwait ();
```

10.15 game_intervar

Mit `inter_var` hat man die Möglichkeit während des Ablaufes der Bargain - Session, Variablen (um-) zu setzen. **Achtung!** Jeder Client hat sein eigenes Set von Variablen, daher muß die Variable auch für jeden Client eingegeben werden. Dies kann für eine große Anzahl von Clients mühsam werden. Daher empfiehlt der Entwickler die eingebauten Calculusmöglichkeiten zu nutzen.

Tabelle 10.11 game_intervar

“var”	Name der Variable
-------	-------------------

Synopsis

```
game_intervar ( <VAR> STRING ) ;
```

Beispiel 10.15 game_setvar

```
# Beispiel eines game_intervar library Aufrufes
# (c) Institut für here Studien - Wien
```

```
game_intervar ( "hy" ) ;
```

Siehe auch: `setvar`

10.16 game_setvar

`setvar` bietet die Möglichkeit während des Ablaufes der Bargain - Session, Variablen innerhalb der eingebauten Calculusmöglichkeiten zu setzen. **Achtung!** Diese Variablen haben nur innerhalb der Bargain - Session Geltung und beziehen sich nicht auf Variablen, die in Rabbit definiert wurden.

Tabelle 10.12 game_intervar

“var”	Name der Variable
“ausdruck”	Ausdruck

Synopsis

```
game_setvar ( <VAR> STRING, <AUSDRUCK> STRING ) ;
```

Siehe auch: `setvar`

Beispiel 10.16 *game_setvar*

```
# Beispiel eines game_setvar library Aufrufes
# (c) Institut für here Studien - Wien
```

```
game_setvar ( "hy", "23 * wurzel" );
```

10.17 *game_evalinter*

evalinter wertet die Ergebnisse einer Befragung (siehe *game_input*) aus. Diese Funktion durchläuft alle *game_input* Statements zwischen den beiden Labels und vergleicht die gegebenen Antworten mit den im *game_input* - Statement gegebenen richtigen Antworten und inkrementiert für jede richtige Antwort den Wert der Variable um eins. Als Wert der Variable erhält man am Ende die Anzahl der richtigen Antworten zwischen den beiden Labels.

Tabelle 10.13 *game_evalinter*

"var"	Name der Variable
"label1"	1. Label
"label2"	2. Label

Synopsis

```
game_evalinter ( <VAR> STRING, <LABEL1> STRING, <LABEL2> STRING ) ;
```

Beispiel 10.17 *game_evalinter*

```
# Beispiel eines game_evalinter library Aufrufes
# (c) Institut für here Studien - Wien
```

```
game_window ( 1, 0,0, 24, 80 ) ;

game_label( "beginfragen");
game_input ( 1, 0, 40, 5, "var1", "542", "frage1.txt");
game_input ( 1, 0, 40, 5, "var1", "542", "frage2.txt");
game_label( "endfragen");

game_evalinter("ergebnis", "beginfragen", "endfragen");
```

Siehe auch: *setvar*

10.18 *game_random*

Eine ganzzahlige Zufallszahl zwischen den angegebenen Bereichen wird berechnet und in der Variable abgespeichert.

Tabelle 10.14 game_random

"var"	Name der Variable
"low"	untere Grenze
"high"	obere Frenze

Synopsis

```
game_random(<VAR> STRING,<LOW> STRING,<HIGH> STRING);
```

Beispiel 10.18 game_random

```
# Beispiel eines game_random library Aufrufes
# (c) Institut fr hhere Studien - Wien

# Simuliert einen Wrfel
game_label( "zufall", "1", "6");
```

Siehe auch: setvar

10.19 game_comment

Gibt einen Kommentar am Server aus. Diese Funktioniert dient dazu dem Operator Information über den Ablauf der Session zu liefern.

Tabelle 10.15 game_comment

"text"	Text
--------	------

Synopsis

```
game_comment ( <TEXT> STRING ) ;
```

Beispiel 10.19 game_comment

```
# Beispiel eines game_comment library Aufrufes
# (c) Institut fr hhere Studien - Wien

game_comment( "zufall = $zufall$");
```

Siehe auch: random

Literaturverzeichnis

- [1] Unix System Laboratories, *SYSTEM V INTERFACE DEFINITION VOL 3*, Unix Press, Addison-Wesley, 1991.

