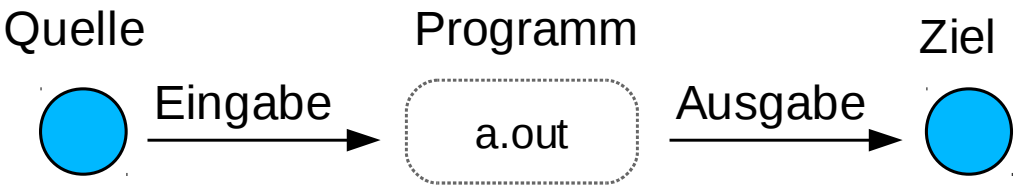
Ein-/Ausgabe

Worum geht's?



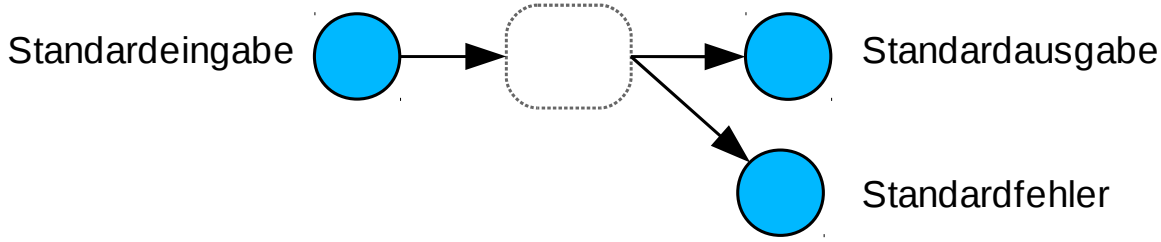
vordefinierte Quellen und Ziele

Eingabe

- Standardeingabe

Ausgabe

- Standardausgabe
- Fehlerausgabe



... und wie können Quelle und Ziel angegeben werden ?

Überblick

ANSI und POSIX

ANSI

- C Standardbibliothek
- stdio.h

POSIX

- Erweiterungen / Ergänzungen
- unistd.h

glib (und GIO)

glib

- glib.h

gio

- gio/gio.h

ANSI - Quellen und Ziele

Datentyp

- "versteckte" Struktur

```
FILE*
```

vordefinierte Quellen und Ziele

- stdin Standardeingabe
- stdout Standardausgabe
- stderr Standardfehler

```
extern FILE* stdin
extern FILE* stdout
extern FILE* stderr
```

"neue" Quellen und Ziele

- aus / nach Datei
  ... auch Gerätedatei

```
FILE*  fopen   (const char *filename, const char *mode)
int    fclose  (FILE *stream)
FILE*  freopen (const char *filename, const char *mode, FILE *stream)
```

wichtige modi: `"r"` lesen `"w"` schreiben `"r+"` lesen und schreiben `"b"` binär

... aber! z.B. keine Gerätedatei für Netzwerk-Interfaces

ANSI – Eingabe und Ausgabe

Eingabe

```
int  getchar (void)
int  fgetc   (FILE *stream)
int  ungetc  (int c, FILE *stream)

char *gets   (char *s)
char *fgets (char *s, int size, FILE *stream)

size_t *fread (void *ptr, size_t size, size_t n, FILE *fp)
```

Ausgabe

```
int putchar (int c)
int fputc   (int c, FILE *stream)

int puts  (const char *s)
int fputs (const char *s, int size, FILE *stream)
```

Programmieraufgabe I

Es soll ein cat ähnliches Programm geschrieben werden.

Im Vergleich zum Original ein wenig vereinfacht soll es folgende Aufrufsyntax verstehen:

cat [DATEI] ...

Folgende Anforderungen sollen dabei erfüllt werden:

Jede als Parameter übergebene Datei soll dabei eingelesen und auf der Standardausgabe ausgegeben werden.

Nicht vorhandene Dateien sollen dabei stillschweigend ignoriert werden.

Ohne Parameter aufgerufen soll das Programm stattdessen aus der Standardeingabe lesen. Die Ausgabe soll auch in diesem Fall auf der Standardausgabe erfolgen.

... einige Lösungsvorschläge

```c
#include <stdio.h>

void cat (FILE *fp) {
    int c = fgetc (fp);
    while (c != EOF) {
        putchar (c);
        c = fgetc (fp);
    }
}

int main (int argc, char *argv[]) {
    if (argc == 1)
        cat (stdin);
    else {
        int i;
        for (i=1; i<argc; ++i) {
            FILE *fp = fopen (argv[i], "rb");
            if (fp != NULL)
                cat (fp);
        }
    }
    return 0;
}
```

```c
#include <stdio.h>

void cat (void) {
    int c;
    while ((c = getchar ()) != EOF)
        putchar (c);
}

int main (int argc, char *argv[]) {
    if (argc == 1)
        cat ();
    else
        while (*argv++)
            if (freopen (*argv, "rb", stdin))
                cat ();
    return 0;
}
```

```c
#include <stdio.h>

int main (int argc, char *argv[]) {
    int c;
    while (*argv++)
        if (argc == 1 || freopen (*argv, "rb", stdin))
            while ((c = getchar ()) != EOF)
                putchar (c);
    return 0;
}
```

ANSI – "Zeichentypen"

Zeichenklassen

```
int isascii (int c)                                    <ctype.h>

int iscntrl (int c)

int isalpha (int c)
int islower (int c)
int isupper (int c)

int isdigit (int c)
int isxdigit (int c)
int isalnum (int c)

int ispunct (int c)

int isprint (int c)
int isgraph (int c)
int isspace (int c)
int isblank (int c)
```

ANSI – Formatierte Ein-/Ausgabe

Eingabe

```
int scanf  (const char *format, ...)
int fscanf (FILE *stream, const char *format, ...)
int sscanf (const char *string, const char *format, ...)
```

Ausgabe

```
int printf   (const char *format, ...)
int fprintf  (FILE *stream, const char *format, ...)
int sprintf  (char *s, const char *format, ...)

int snprintf (char *s, size_t size, const char *format, ...)
```

Formatstring

- Feldbreite
- Ausrichtung
- Genauigkeit (für Gleitkommazahlen)
- Länge / Größe (hh, h, l, ll)
- Konvertierung

... im Zweifelsfalle besser noch einmal nachlesen (z.B. man printf)
hier geht sonst schnell etwas schief !

Programmieraufgabe II

Es soll ein wc ähnliches Programm geschrieben werden.

Auch hier soll die Aufrufsyntax im Vergleich zum Original ein wenig verinfacht werden.
Das Programm soll nämlich ganz ohne Parameter aufgerufen werden:

wc

Folgende Anforderungen sollen dabei erfüllt werden:

Das Programm liest seine Eingabe von der Standardeingabe

Die Ausgabe soll in der Form: '*c* character, *w* words, *l* lines' erfolgen

Als ein "Wort" soll dabei jeweils die längste zusammenhängende Folge von Eingabezeichen gelten, die ausschließlich aus alphabetischen Zeichen besteht.

Zeilenumbrüche, Tabulatoren und Leerzeichen sollen als Zeichen mitgezält werden.

Lösungsvorschlag

... leider genauso einfach wie falsch ...

```c
#include <stdio.h>
#include <ctype.h>

int main (int argc, char *argv[]) {
    int n_chars = 0;
    int n_lines = 0;
    int n_words = 0;
   int in_word = 0;
   int c;
    while ((c = getchar ()) != EOF) {
        ++n_chars;
        if (c == '\n')
            ++n_lines;
        if (!in_word && isalpha (c))
            ++n_words;
        in_word = isalpha (c);
    }
   printf ("%d characters, %d words, %d lines\n", n_chars, n_words, n_lines);
}
```

Können 'ä's 'ö's und 'ü's richtig gezählt werden?

- Fragen wir einfach Mal nach: 55 characters, 10 words, 1 lines

Hoppla!

Zeichen und Zeichencodierung

Datentyp char
8 – bit
somit maximal 2^8 = 256 Zeichen

ASCII (American Standard Code for Information Interchange)
7 – bit je Zeichen
damit 128 Zeichen - inklusive Steuerzeichen

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

(prominente Vertreter hervorgehoben)

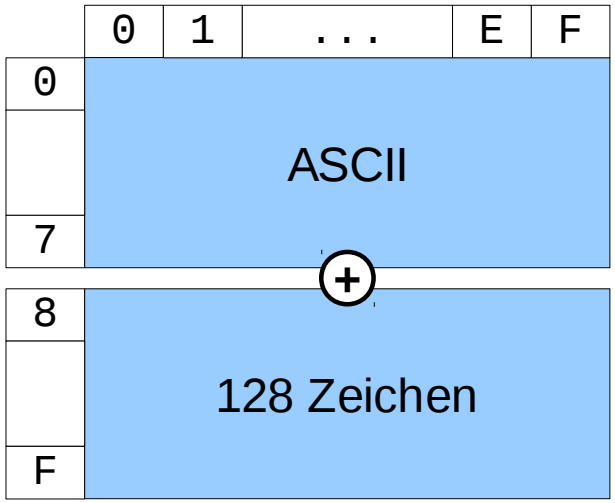
... auch über Dokumentation abrubar! man ascii

Zeichencodierungen II

ISO 8859-x
8 – bit

ASCII (höchstwertiges bit = 0)
+ 128 Zeichen (höchstwertiges bit = 1)

in Westeuropa
ISO 8859-1 (Latin-1)

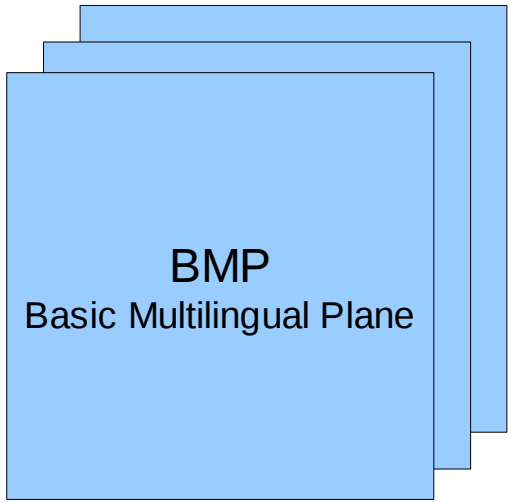


UNICODE
17 Ebenen mit jeweils 2^{16} = 65.536 Zeichen
(warum 17 ?)

damit 1.114.112 Zeichen darstellbar

aber! nicht mehr in einem byte!

... somit Multibyte Darstellung nötig



UNICODE

Codierungen

UTF-32 / UCS-4

- 32 bit je Zeichen
- alle Zeichen mit gleicher Länge codiert
- aber! benötigt viel Platz!

UTF-8

- Zeichen mit unterschiedlicher Länge codiert (bis zu 6 byte je Zeichen)
- benötigt weniger Platz
- aber! nur sequentieller Zugriff möglich!

UTF-16

- ursprünglich: 16 bit je Zeichen
- um alle Zeichen darstellen zu können: jetzt codierung mit einem/zwei 16-bit Zeichen surrogate pair (high surrogate + low surrogate)
- Codepoints U+D800 ... U+DFFF für UTF-16 Codierung reserviert

Reparaturversuche ...

Datentyp wchar_t

Codierung

- nicht standardisiert
- GNU – UCS-32

Funktionen

- zusätzliches 'w' im Namen
  ... sonst ändert sich nichts
- getwchar
- putwchar
  ...
- iswalpha
- iswdigit
  ...

... und Fehlschläge

Problem

- fragen wir Mal erneut:

```
$ wc
können 'ä's 'ö's und 'ü's richtig gezählt werden?
1 characters, 1 words, 0 lines
```

... scheint auch nicht so ganz korrekt zu sein!

... und Lösung

- ein letzter Schritt ist noch zu unternehmen
  die locale steht noch auf C !

  setlocale (LC_ALL, "")

```
$ wc
können 'ä's 'ö's und 'ü's richtig gezählt werden?
50 characters, 11 words, 1 lines
```

... sieht schon ein wenig besser aus!

ANSI – weitere Themen

Pufferung

- setbuf
- setvbuf
- fflush

Positionierung

- fseek
- ftell
- rewind
- fgetpos
- fsetpos

Dateioperationen

- remove
- rename

POSIX - Quellen und Ziele

Datentyp

- "Filedeskriptor"

```
int
```

vordefinierte Quellen und Ziele

- STDIN_FILENO Standareingabe
- STDOUT_FILENO Standardausgabe
- STDERR_FILENO Standardfehler

```
#define STDIN_FILENO   0
#define STDOUT_FILENO  1
#define STDERR_FILENO  2
```

"neue" Quellen und Ziele

```
int  open  (const char *filename, int flags)[1]
int  open  (const char *filename, int flags, mode_t mode)[1]
int  creat (const char *filename, mode_t mode)
```

prominente flags

nur lesen	O_RDONLY
nur schreiben	O_WRONLY
lesen/schreiben	O_RDWR
erstellen	O_CREAT
anhängen	O_APPEND

einige modi

lesen	S_IRUSR	S_IRGRP	S_IROTH
schreiben	S_IWUSR	S_IWGRP	S_IWOTH
ausführen	S_IXUSR	S_IXGRP	S_IXOTH
alle	S_IRWXU	S_IRWXG	S_IRWXO

[1)] ist ein wenig gelogen - korrekt ist: `int open (const char *, int, ...)`

POSIX – Eingabe und Ausgabe

neue Funktionen

- Blockweise lesen/schreiben

```
ssize_t  read  (int fd, void* buffer, size_t count)
ssize_t  write  (int fd, void* buffer, size_t count)
```

mit bekannten Funtionen arbeiten

- Konvertierung: fd → fp

```
FILE*  fdopen  (int fd, const char *mode)
```

- Zeilenweise lesen

```
ssize_t  getline (char** line, size_t *n, FILE* stream)
```

Was hat's gebracht?

weitere Quellen und Ziele

können nicht nur auf geöffnete Datei verweisen

Pipes

- pipe ()
- popen ()
- pclose ()

Fifos (named pipe)

- mkfifo () mknod ()
- mkfifoat () mknodat ()

Sockets

- socket ()
- bind ()
- listen ()
- accept ()
- connect ()

Beispiel

Sockets

Typen

- verbindungsorientiert SOCK_STREAM
- verbindungslos SOCK_DGRAM
- low-level SOCK_RAW

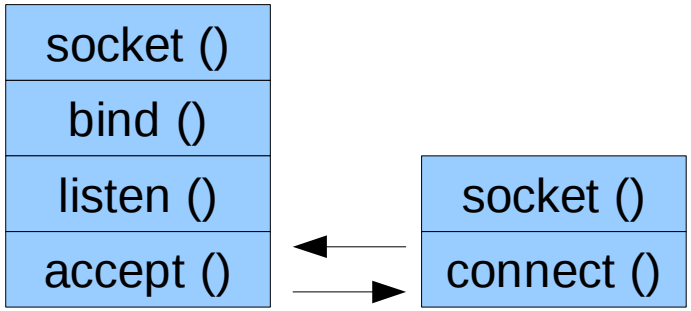
Adressfamilien

- Unix Domain Sockets AF_UNIX
- (TCP)/IP V4 AF_INET
- (TCP)/IP V6 AF_INET6

Funktionen

- erstellen socket ()
- Adresse "binden" bind ()
- passiv öffnen listen ()
accept ()
- aktiv öffnen connect ()

Funktionsweise



Implementierung ...

```
#ifndef COMMON_H
#define COMMON_H

#include <stdio.h>
#include <assert.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 7777
#endif
```

```
#include "common.h"
void server_loop (int client) {
  char buffer[80];
  int n;
 do {
    n = read (client, buffer,
              sizeof (buffer));
    write (STDOUT_FILENO, buffer, n
);
 } while (n > 0);
}
```

```
int main (int argc, char *argv[]) {
  int error;
 int server = socket (AF_INET, SOCK_STREAM, 0);
  assert (server != -1);
 struct sockaddr_in address;
  address.sin_family = AF_INET;
  address.sin_port = htons (PORT);
  inet_aton ("127.0.0.1", &address.sin_addr);
 error = bind (server,
               (struct sockaddr*) &address,
               sizeof (address));
  assert (!error);
 error = listen (server, 1);
  assert (!error);
 while (1) {
    int client;
    client = accept (server, NULL, NULL);
   if (client != -1)
    server_loop (client);
  }
 return 0;
}
```

... Implementierung

```
#include "common.h"

void client_loop (int client) {
    char buffer[80]; int n;
   do {
        n = read (STDIN_FILENO, buffer, sizeof (buffer));
        write (client, buffer, n);
    } while (n > 0);
}

int main (int argc, char *argv[]) {
   int client = socket (AF_INET, SOCK_STREAM, 0);
    assert (client != -1);

   struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_port = htons (PORT);
    inet_aton ("127.0.0.1", &address.sin_addr);

   int error = connect (client, (struct sockaddr*) &address, sizeof (address));
    assert (error != -1);

   client_loop (client);
    close (client);
    return 0;
}
```

Programmieraufgabe III

Es soll ein Programm words geschrieben werden, das eine von der Standardeingabe gelesene Eingabe in einzelne Wörter zerlegt. Diese Wörter sollen dann Zeilenweise ausgegeben werden.

Auch hier soll die Aufrufsyntax denkbar einfach sein:

```
words
```

Folgende Anforderungen sollen dabei erfüllt werden:

Als ein "Wort" soll wie in der Aufgabe zuvor die jeweils die längste zusammenhängende Folge von Eingabezeichen gelten, die ausschließlich aus alphabetischen Zeichen besteht.

Neben dem Wort selbst soll auch die Zeile, auf der es gefunden wurde ausgegeben werden.

Lösungsvorschlag

```
#define _GNU_SOURCE
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define LINE_LENGTH 80
char *next_word (char **string) {
   while (! isalpha (**string)) {
        if (**string == '\0')
            return NULL;
        ++*string;
    }
    char *start = *string;
    while (isalpha (**string))
        ++*string;
   size_t n = *string - start;
    char *word = malloc (n +1);
    assert (word != NULL);
   strncpy (word, start, n);
    word[n] = '\0';
    return word;
}
```

```
int main (int argc, char *argv[]) {

    size_t n = LINE_LENGTH;
    char *line = malloc (n);

    int n_line = 0;
    char *word;

    while ((getline (&line, &n, stdin))
            != -1) {
        char *start = line;
        while ((word = next_word (&start))) {
            printf ("%d: %s\n", n_line, word);
            free (word);
        }
        ++n_line;
    }
    return 0;
}
```

gar nicht Mal so schlecht, aber ...

.... das kennen wir ja schon

```
$ ./version-1
eins zwei drei
0: eins
0: zwei
0: drei
vier fünf
1: vier
1: f
1: nf
```

diesmal nehmen wir die glib zu Hilfe

zweiter Versuch

```
#include <glib.h>
#include <locale.h>

gchar *next_word (gchar **string) {
  gunichar c = g_utf8_get_char (*string);

  while (! g_unichar_isalpha (c)) {
    if (c == '\0')
      return NULL;
      *string = g_utf8_next_char (*string);
      c = g_utf8_get_char (*string);
  }
  gchar *start = *string;
  while (g_unichar_isalpha (c)) {
    *string = g_utf8_next_char (*string);
    c = g_utf8_get_char (*string);
  }

  gsize n = *string - start;
  gchar *word = g_strndup (start, n);

  return word;
}
```

```
int main (int argc, char *argv[]) {

  GIOChannel *in = g_io_channel_unix_new (0);

  gsize n;
  gchar *line;

  int n_line = 0;
  char *word;

  setlocale (LC_ALL, "");

  while (g_io_channel_read_line (in,
                  &line, &n, NULL, NULL)
          == G_IO_STATUS_NORMAL)
  {
    gchar *start = line;
    while ((word = next_word (&start))) {
      g_print ("%d: %s\n", n_line, word);
      g_free (word);
    }
    ++n_line;
  }
  return 0;
}
```

glib

verwendete Datentypen

```
GIOChannel
gunichar
```

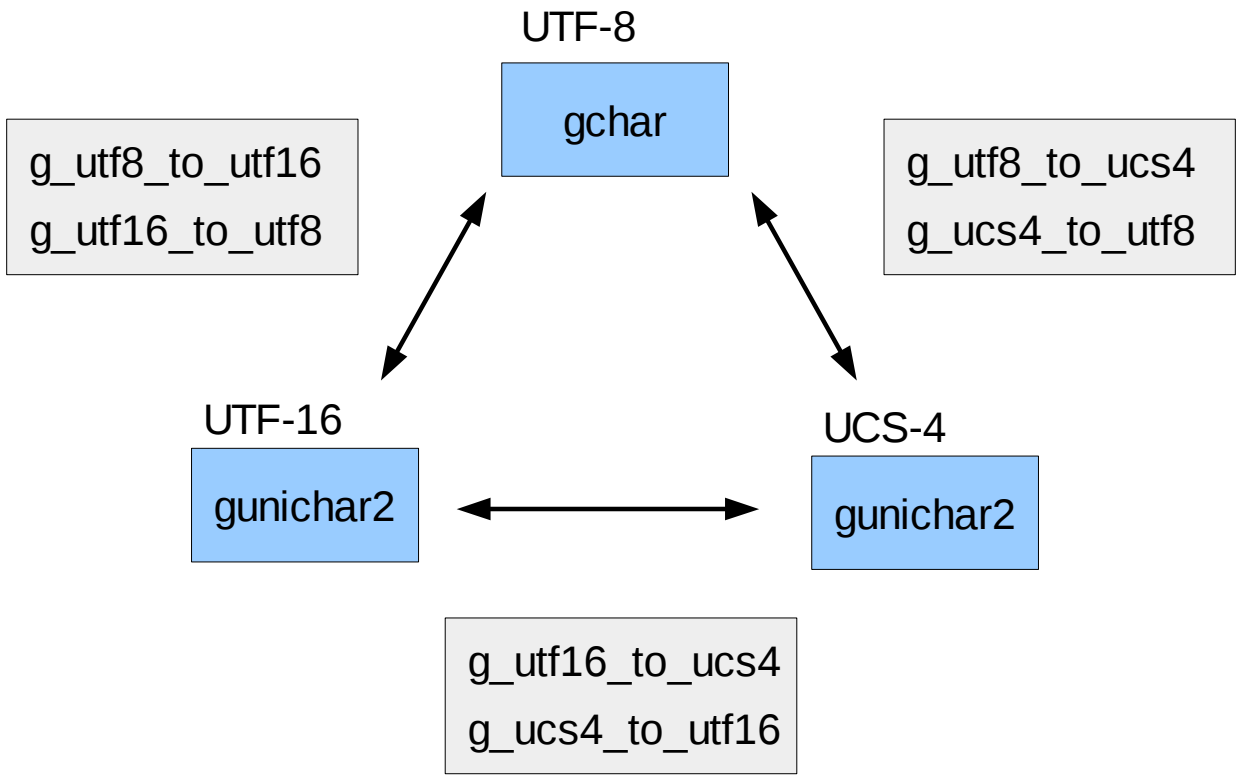
verwendete Funktionen

```
GIOChannel g_io_channel_unix_new   (int fd)
GIOStatus  g_io_channel_read_line  (GIOChannel *channel, gchar **line, gsize *n,
                                    gsize *terminator, GError **error )

gunichar   g_utf8_get_char     (gchar *string)
#define    g_utf8_next_char    (p)
gboolean   g_unichar_isalpha   (gunichar c)

gchar      *gstrndup (const gchar *string, gsize n)
```

Zeichencodierungen



glib – nützliche Datentypen

GHashTable

Datentyp

```
typedef struct _GHashTable GHashTable;
```

Erstellen

```
GHashTable *g_hash_table_new (GHashFunc hash_func, GEqualFunc key_equal_func);
```

... zwei Funktionen müssen beim Generieren angegeben werden:

```
typedef guint    (*GHashFunc) (gconstpointer key)
typedef gboolean (*GEqualFunc) (gconstpointer a, gconstpointer b)
```

vordefinierte Funktionen

```
GHashFunc
NULL = g_direct_hash() g_int_hash() g_int64_hash() g_double_hash() g_str_hash()

GEqualFunc
NULL = g_direct_equal()  ...  g_str_equal()
```

GHashTable - Operationen

```
guint g_hash_table_size (GHashTable  *h)


void     g_hash_table_insert (GHashTable *h, gpointer key, gpointer value)
gboolean g_hash_table_remove (GHashTable *h, gpointer key)
gpointer g_hash_table_lookup (GHashTable *h, gconstpointer key)


gpointer g_hash_table_foreach (GHashTable *h, GHFunc func, gpointer user_data)

typedef void (*GHFunc) (gpointer key, gpointer value, gpointer user_data)


GList *g_hash_table_get_keys   (GHashTable *h)
GList *g_hash_table_get_values (GHashTable *h)
```

Programmieraufgabe IV

Es soll ein Programm count_words erstellt werden, das die Vorkommen einzelner Worte in einer beliebigen Eingabe bestimmt.

Der Aufruf soll wie (fast) immer ohne Parameter erfolgen:

```
count_words
```

Folgende Anforderungen sollen dabei erfüllt werden:

Als ein "Wort" soll wie immer die jeweils die längste zusammenhängende Folge von Eingabezeichen gelten, die ausschließlich aus alphabetischen Zeichen besteht.

Die Ausgabe braucht (noch) nicht sortiert zu erfolgen.

Jedes Wort soll gefolgt von seiner Häufigkeit auf einer eigenen Zeile ausgegeben werden.

Lösungsvorschlag

```
#include <glib.h>
...

void insert (GHashTable *table,
             gchar *word, gint line) {

  gint *value = g_hash_table_lookup (table, word);

  if (value == NULL) {
      value = g_new (int, 1);
      *value = 1;
      g_hash_table_insert (table, word, value);
  }

  else {
    *value = *value + 1;
    g_free (word);  /* wirklich richtig so? */
  }
}

void print_word (gchar *word, gint *n,
                 gpointer data) {
  g_print ("%s : %d\n", word, *n);
}
```

```
int main (int argc, char *argv[]) {
 ...

  GHashTable *index;
  index = g_hash_table_new (g_str_hash,
                            g_str_equal);

  while
    ...
    while ((word = next_word (&start))) {
      insert (index, word, n_line);
    }
    ++n_line;
  }

  g_hash_table_foreach (index,
                (GHFunc) print_word, NULL);

    return 0;
}
```

GList

```
typedef struct _GList GList;

struct _GList
{
  gpointer data;
  GList *next;
  GList *prev;
};
```

```
GList  *list = NULL;

int *number =< g_new (int, 1);
*number = 23;

list = g_list_append (list, number);
```

GList – Operationen

... die wichtigsten – volle Information wie immer in der Dokumentation !

```
g_list_append ()
g_list_prepend ()
g_list_insert ()
g_list_insert_before ()

g_list_find ()

g_list_next ()
g_list_foreach ()
```

Programmieraufgabe V

Es soll ein Programm index erstellt werden, das die Vorkommen einzelner Worte in einer beliebigen Eingabe bestimmt.

Der Aufruf soll wie (fast) immer ohne Parameter erfolgen:

```
index
```

Folgende Anforderungen sollen dabei erfüllt werden:

Als ein "Wort" soll wie immer die jeweils die längste zusammenhängende Folge von Eingabezeichen gelten, die ausschließlich aus alphabetischen Zeichen besteht.

Jedes Wort soll gefolgt von allen Zeilennummern, in denen es auftritt auf einer eigenen Zeile ausgegeben werden.

Bei mehrfachem Auftreten eines Wortes in einer Zeile soll diese Zeile für jedes Vorkommen genau einmal ausgegeben werden.