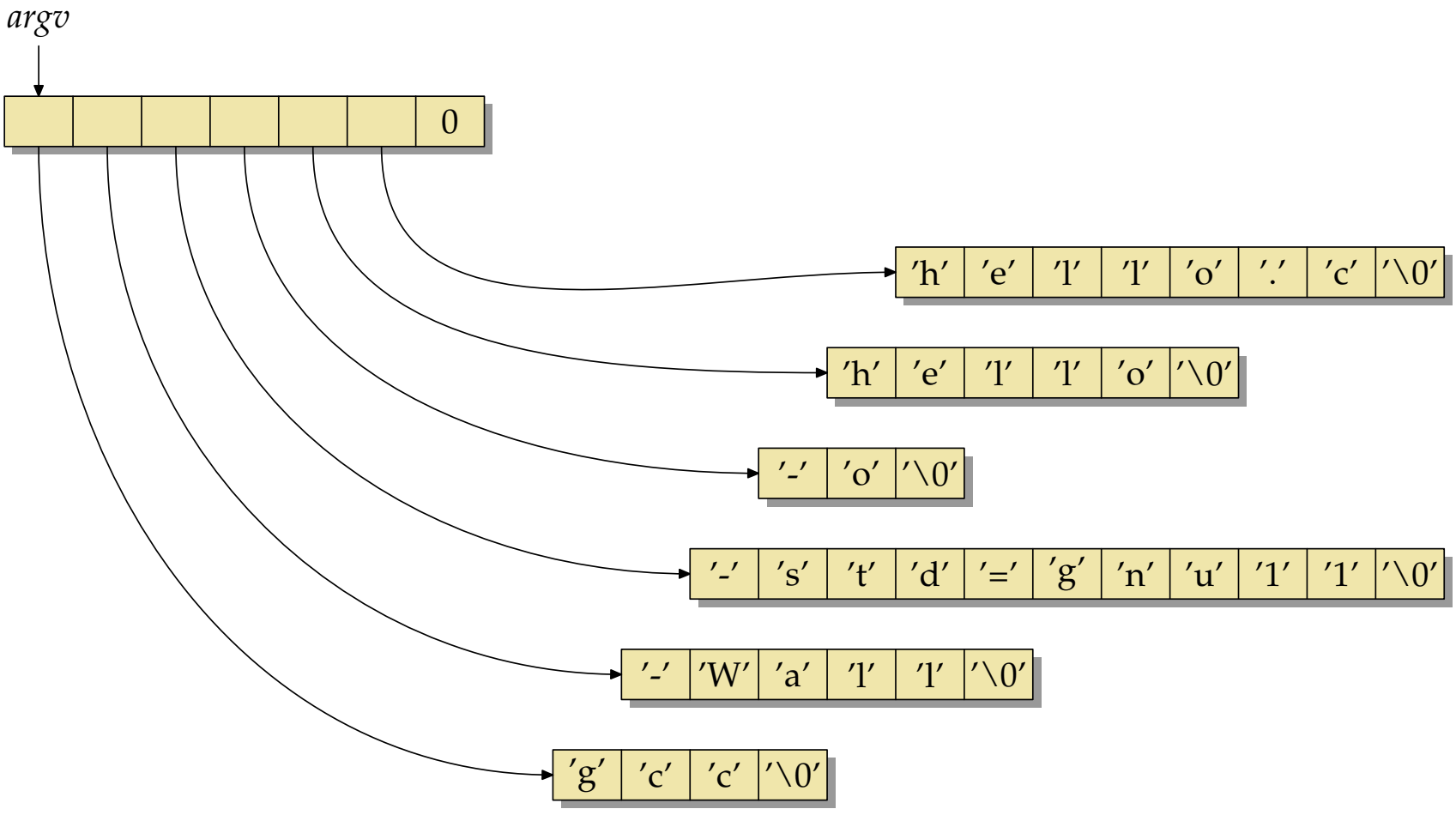
```
int main(int argc, char* argv[]) {
   /* ... */
}
```

- *main* erhält gemäß dem Standard zwei Parameter, *argc* und *argv*, die der Übermittlung der Kommandozeilenparameter dienen.
- *argc* enthält die Zahl der Parameter, wobei der Kommandoname mitgezählt wird.
- *argv* ist ein Zeiger auf ein Array von Zeigern, das auf die einzelnen Kommandozeilenparameter verweist.



- Dies ist die Repräsentierung der Kommandozeilenparameter für „gcc -Wall -std=gnu11 -o hello hello.c“. *argc* hätte hier den Wert 6.

args.c

```
#include <stdio.h>

int main(int argc, char* argv[]) {
   printf("Command name: %s\n", argv[0]);
   printf("Number of command line arguments: %d\n", argc-1);
   for (int i = 1; i < argc; i++) {
      printf("Argument %d: %s\n", i, argv[i]);
   }
}
```

- Dieses Programm gibt den Kommandonamen (in *argv*[0]) und die übrigen Kommandozeilenparameter aus.
- Der Kommandoname ist normalerweise der Name, mit dem ein Kommando aufgerufen worden ist.

args2.c

```
#include <stdio.h>

int main(int argc, char* argv[]) {
   char* cmdname = *argv++; --argc;
   printf("Command name: %s\n", cmdname);
   printf("Number of command line arguments: %d\n", argc);
   while (argc-- > 0) {
      printf("Argument: %s\n", *argv++);
   }
}
```

- Alternativ können die Kommandozeilenparameter auch sukzessive „konsumiert“ werden, indem entsprechend *argc* gesenkt und *argv* weitergesetzt wird.
- Dann sollte aber die Invariante eingehalten werden, dass *argc* die Zahl der noch unter *argv* verbleibenden Parameter angibt.

mygrep.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
   char line[256];

   if (argc != 2) {
      fprintf(stderr, "Usage: %s pattern\n", argv[0]);
      exit(1);
   }

   while (fgets(line, sizeof line, stdin)) {
      if (strstr(line, argv[1])) {
         fputs(line, stdout);
      }
   }
}
```

- *strstr* sucht nach dem ersten Vorkommen des zweiten Parameter in dem ersten Parameter und liefert, falls gefunden, einen Zeiger darauf zurück, ansonsten 0.

- Per Konvention beginnen Optionen in der Kommandozeile mit dem Minuszeichen „-“.
- Hinter dem Minuszeichen können dann ein oder mehrere Optionen folgen, die typischerweise mit nur einem Buchstaben benannt werden.
- Es gibt auch Kommandos, die längere Optionsnamen unterstützen. Dann können aber Optionen nicht mehr in einem Parameter integriert werden oder die Optionen müssen anders beginnen. GNU-Werkzeuge verwenden dafür gerne das doppelte Minuszeichen „--“.
- Zwei alleinstehende Minuszeichen beenden die Folge der Optionen.
- Danach folgen typischerweise Pflichtargumente (etwa der zu suchende Text) und/oder Eingabedateien.
- Diese Konventionen sind im POSIX-Standard festgehalten.

- Zusätzlich zu dem zu suchenden Text soll es möglich sein. Optionen anzugeben (beides in Nachbildung des originalen *grep*-Kommandos):
  - Die Option „-n“ (*number*) soll die jeweilige Zeilennummer mit ausgeben.
  - Die Option „-v“ (*veto*) soll dazu führen, dass nur die Zeilen ausgegeben werden, die den Suchtext *nicht* enthalten.
- Die Optionen sollen kombinierbar sein, d.h. „-n“ und „-x“ können als zwei getrennte Parameter angegeben werden oder auch kombiniert, also etwa „-nx“ oder „-xn“.
- Der Konvention folgend soll „--“ die Optionen beenden. Bei dem Kommando „mygrep1 -n -- -1“ wird „-1“ nicht als die (nicht vorhandene) Option „1“ interpretiert, sondern als der Suchtext „-1“.

Suchkommando mit Optionen

mygrep1.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

int main(int argc, char *argv[]) {
   char *cmdname = *argv++; --argc; /* take command name */
   bool opt_v = false; /* option -v: print non-matching lines */
   bool opt_n = false; /* option -n: emit line numbers */

   /* process options ... */

   /* do the actual work */
   char line[256]; /* input line */
   int lineno = 0; /* current line number */
   while (fgets(line, sizeof line, stdin)) {
      lineno++;
      if (!strstr(line, pattern) == opt_v) {
         if (opt_n) {
            printf("%d: ", lineno);
         }
         fputs(line, stdout);
      }
   }
}
```

Suchkommando mit Optionen

mygrep1.c

```
/* process options */
for(; argc > 0 && **argv == '-'; argc--, argv++) {
   /* per convention we interpret "--" as end of options */
   if (argv[0][1] == '-' && argv[0][2] == 0) {
      argc--; argv++; break;
   }
   if (argv[0][1] == 0) {
      /* got just a "-" without anything following */
      fprintf(stderr, "%s: empty option\n", cmdname);
      return 1;
   }
   /* process individual options within an argument */
   for (char* s = *argv + 1; *s; s++) {
      switch (*s) {
         case 'v' : opt_v = true; break;
         case 'n' : opt_n = true; break;
         default:
            fprintf(stderr, "%s: illegal option '%c'\n", cmdname, *s);
            return 1;
      }
   }
}

/* just one remaining argument with the pattern is expected */
if (argc != 1) {
   fprintf(stderr, "Usage: %s [-nv] pattern\n", cmdname);
   return 1;
}
char* pattern = *argv++; --argc;
```

Unterstützung beliebig langer Zeilen

mygrep2.c

```
char* readline(FILE* fp) {
   int len = 32;
   char* cp = malloc(len);
   if (!cp) return 0;
   int i = 0;
   int ch;
   while ((ch = getc(fp)) != EOF && ch != '\n') {
      cp[i++] = ch;
      if (i == len) {
         /* double the allocated space */
         len *= 2;
         char* newcp = realloc(cp, len);
         if (!newcp) {
            free(cp);
            return 0;
         }
         cp = newcp;
      }
   }
   if (i == 0 && ch == EOF) {
      free(cp);
      return 0;
   }
   cp[i++] = 0;
   return realloc(cp, i); /* free unused space */
}
```

- Bislang waren die Optionen nur **bool**-wertig.
- Gelegentlich haben diese aber einen größeren Wertebereich, z.B. eine ganze Zahl oder ein Dateiname.
- *grep* kennt z.B. eine Option, die den jeweils anzugebenden Kontext spezifiziert (Zahl der zu zeigenden Zeilen davor und danach).
- Mit „-c 3“ sind beispielsweise drei Zeilen Kontext darzustellen.
- Konventionellerweise ist es aber auch zulässig, „-c3“ anzugeben oder in Kombination: „-nc3“.

Optionen mit Werten

mygrep3.c

```
for (char* s = *argv + 1; *s; s++) {
   switch (*s) {
      case 'c' :
         if (s[1]) {
            ++s;
         } else {
            ++argv; --argc;
            if (!argc) {
               fprintf(stderr,
                  "%s: argument for option 'c' missing\n", cmdname);
            }
            s = *argv;
         }
         /* pick argument from the rest of s[] */
         for (; *s; ++s) {
            if (!isdigit(*s)) {
               fprintf(stderr,
                  "%s: digits expected for option 'c'\n", cmdname);
               return 1;
            }
            context = context * 10 + *s - '0';
         }
         --s; /* break from outer for loop */
         break;
      case 'v' : opt_v = true; break;
      case 'n' : opt_n = true; break;
      default:
         fprintf(stderr, "%s: illegal option '%c'\n", cmdname, *s);
         return 1;
   }
}
```

- Es ist unerfreulich und fehleranfällig, die Kommandozeilenbearbeitung von Optionen „per Hand“ vorzunehmen.
- Es gibt daher im Rahmen des POSIX-Standards die Funktion *getopt*, die die Konventionen unterstützt.
- *getopt* erhält *argc* und *argv* (einschließlich dem Kommandonamen) und eine Optionsspezifikation, bestehend aus den Buchstaben der Optionennamen. Steht ein „:“ hinter einer Option, so erwartet diese einen Wert.
- Unsere *grep*-Nachimplementierung bräuchte also die Spezifikation „c:nv“.

Verarbeitung der Optionen mit *getopt*

mygrep4.c

```
char usage[] = "Usage: %s [-c context] [-nv] pattern\n";
/* external variables set by getopt() */
extern char* optarg;
extern int optind;
/* process options */
int option;
while ((option = getopt(argc, argv, "c:nv")) != -1) {
   switch (option) {
      case 'c':
         context = atoi(optarg); break;
      case 'n':
         opt_n = true; break;
      case 'v':
         opt_v = true; break;
      default:
         fprintf(stderr, usage, cmdname); return 1;
   }
}
argc -= optind; argv += optind; /* skip options processed by getopt() */
/* just one remaining argument with the pattern is expected */
if (argc != 1) {
   fprintf(stderr, usage, cmdname); return 1;
}
char* pattern = *argv++; --argc;
```

mygrep5.c

```
/* compile pattern */
regex_t regex; /* compiled regular expression */
unsigned int regex_flags = REG_NOSUB;
if (opt_i) {
   regex_flags |= REG_ICASE; /* ignore case */
}
if (opt_e) {
   regex_flags |= REG_EXTENDED; /* supported egrep syntax */
}
unsigned int regex_error = regcomp(&regex, pattern, regex_flags);
if (regex_error) {
   char errbuf[128];
   regerror(regex_error, &regex, errbuf, sizeof errbuf);
   fprintf(stderr, "%s: invalid regular expression: %s\n",
      cmdname, errbuf);
   return 1;
}
```

- Die POSIX-Bibliothek bietet auch eine Bibliothek für reguläre Ausdrücke an. Hier muss zunächst der reguläre Ausdruck in eine interne Datenstruktur übersetzt werden.

mygrep5.c

```
if ((regexec(&regex, line, 0, 0, 0) != 0) == opt_v) {
```

- Statt *strstr* wird dann *regexec* verwendet mit der zuvor einmal erstellten Datenstruktur.
- *regexec* liefert 0 zurück, wenn der reguläre Ausdruck für einen Teil der Zeichenkette zutrifft.
- *regexec* unterstützt auch das Extrahieren von Teilen der Zeichenkette mit Hilfe von Klammern-Ausdrücken. Davon wird hier aber kein Gebrauch gemacht und deswegen sind zwei Parameter auf 0 gesetzt.