F07A - Kommandoradsparametrar, filhantering

5DV149 Datastrukturer och algoritmer

Niclas Börlin
niclas.borlin@cs.umu.se

2020-02-10 Mon

Kommandoradsparametrar (1)

- Hittills har vi sett program som tar interaktiv input.
  - Programmet säger till användaren att det förväntar sig indata och läser indata från tangentbordet.
  - I många fall är detta opraktiskt eller rentav omöjligt.
- Vi vill i stället kunna ge programmet det indata det behöver direkt när det startar:
  - Vi ger programmet *programparametrar* (eller *kommandoradsparametrar*), som ur vårt C-programs synvinkel omvandlas till parametrar till funktionen `main`.
  - Parametrarna anges direkt efter namnet på den körbara filen: `program param1 param2 ...`

Kommandoradsparametrar (2)

- För att ta emot kommandoradsparametrar måste programmets `main`-funktion se ut så här:

```
int main(int argc, const char **argv){
    ...
}
```

- `argc` talar om för oss *hur många* parametrar som skickats till programmet.
- `argv` är en *array av strängar (strängpekare)* och innehåller själva parametrarna.

I praktiken...

- Betrakta följande lilla exempel:

code/cmdlineargs.c

```c
#include <stdio.h>
int main(int argc, const char ** argv)
{
        printf("Number of parameters: %d\n", argc);
        for (int i = 0; i < argc; i++) {
                printf("Parameter %d: %s\n", i, argv[i]);
        }
        return 0;
}
```

- Om vi kompilerar `gcc -o cmdlineargs cmdlineargs.c` och kör programmet `./cmdlineargs alpha bravo charlie delta` så får vi resultatet:

```
Number of parameters: 5
Parameter 0: ./cmdlineargs
Parameter 1: alpha
Parameter 2: bravo
Parameter 3: charlie
Parameter 4: delta
```

Filer

- En *fil* kan ses som en följd av minnesceller som behåller sitt innehåll även när datorn är avstängd.
- Innehållet i filerna är en sekvens av bytes, t.ex.

pos	0	1	2	3	4	5	6
dec	99	61	50	54	55	59	10
hex	63	3d	32	36	37	3b	0a
char	c	=	2	6	7	;	.

- Tecknet ’.’ betyder ’punkt eller något specialtecken som inte går att skriva ut’.
- Filer kan tolkas som binärfiler eller textfiler.
- Om filinnehållet ovan tolkas som text så motsvarar det strängen ’`c=267;`’.

Textfiler och binära filer

- Textfiler är organiserade som en sekvens av *tecken*.
- Varje tecken representeras av 1-4 bytes.
  - Standardtecknen (ASCII) är oftast 1 byte.
  - Andra tecken, t.ex. ö kodas olika.

    iso 8859-1 en byte: 246,
    utf-8 två bytes: 195, 182
- Textfiler är organiserade i *rader* av variabel längd.
- Varje rad avslutas med en *radslutssymbol* EOL (end-of-line).
  - Radslutssymbolens längd är 1-2 tecken, varierar med operativsystem.

    unix en byte: 10
    dos två bytes: 13, 10
    Mac OS en byte: 13
- Slutsatser:
  - Undvik icke-ASCII-tecken!
  - Förutsätt inte att radslut har en viss längd!

Filinnehåll, exempel

- ’c=267;’ med unix-radslut:

pos	0	1	2	3	4	5	6
dec	99	61	50	54	55	59	10
hex	63	3d	32	36	37	3b	0a
char	c	=	2	6	7	;	.

- ’c=267;’ med dos-radslut:

pos	0	1	2	3	4	5	6	7
dec	99	61	50	54	55	59	13	10
hex	63	3d	32	36	37	3b	0d	0a
char	c	=	2	6	7	;	.	.

- ’Börlin’ i UTF-8

pos	0	1	2	3	4	5	6	7
dec	66	195	182	114	108	105	110	10
hex	42	c3	b6	72	6c	69	6e	0a
char	B	.	.	r	l	i	n	.

- ’Börlin’ i ISO-8859-1

pos	0	1	2	3	4	5	6
dec	66	246	114	108	105	110	10
hex	42	f6	72	6c	69	6e	0a
char	B	.	r	l	i	n	.

Textfiler och binära filer

- Navigering i en textfil liknar en riktad lista.
- Navigering i en binärfil liknar ett fält med variabel storlek.
- Textfiler är lätta för människor att läsa medan binärfiler oftast kräver program.
  - Källkoden till ett C-program är en textfil medan den körbara koden till programmet är en binärfil.

Textfiler och binära filer

- Presentationen av filinnehållet skiljer sig åt mellan textfil och binärfil.
- För en textfil som innehåller texten

```
c = 267;
```

  - Lagras talet 267 i en textfil blir det som tre tecken '2', '6' och '7'.
    - Varje tecken har en ASCII-kod.
    - Exempelvis har tecknet '2' ASCII-koden 50 som lagras som `00110010` binärt.
- Lagras talet 267 i en binärfil lagras det som `00000001 00001011`.

Textfil och riktad lista, likheter

- Traversering av riktad lista
  - `p = first(l)`
  - `while not Isend(p,l) do`
    - `val = inspect(p,l)`
    - *do stuff with val*
    - `p = next(p,l)`
- Traversering av textfil
  - `f = open(...)`
  - `while not end-of-file(f) do`
    - `val = read-line(f)`
    - *do stuff with val*
  - `close(f)`

Textfil och riktad lista, olikheter

- Filen har en inbyggd position — filpekaren.
- Läsning från filen flyttar fram filpekaren.
- Traversering av textfil
  - `f=open(...)`
  - `(eof,val)=read-line(f)`
  - `while not eof do`
    - *do stuff with val*
    - `(eof,val)=read-line(f)`
  - `close(f)`

Filpekare, strömmar

- För att kommunicera med filer används ett mellanled, en *ström*.
  - Programmet kommunicerar med strömmen, t.ex. öppnar den för att läsa från filen. Vi säger dock oftast att vi *öppnar* filen.
  - För att komma åt strömmen används *filpekare*.
- Filpekare `FILE *` definieras i `stdio.h`, men hur den exakt ser ut är oviktigt.
- Även utskrifter till skärm och inläsning från tangentbord sköts med filpekare:
  - De standardströmmarna benämns `stdout` och `stdin`.
  - Dessutom finns en filpekare för utskrifter av felmeddelanden, `stderr`.

Öppna och stänga filer (1)

```c
#include <stdio.h>
FILE *fopen(const char *fileName, const char *mode);
int fclose(FILE *stream);
```

- För att kunna läsa från eller skriva till en fil måste den först öppnas med `fopen`.
- `fopen` returnerar filpekare vid OK, annars konstanten `NULL`.
- När man är klar med filen bör man stänga den med `fclose`.
- `fclose` returnerar 0 vid OK, annars konstanten `EOF`.
- `fclose` stänger filen och utför ett par andra "renhållningsuppgifter".
- Glömmer man att stänga filer kan man förlora data!

Öppna och stänga filer (2)

```c
#include <stdio.h>
FILE *fopen(const char *fileName, const char *mode);
int fclose(FILE *stream);
```

- Den första parametern till `fopen` är filnamnet.
- Den andra parametern till `fopen` anger vad man vill göra med filen (vilket *mode* man vill köra den i).

`mode`	betydelse	förklaring	om filen finns	om filen inte finns
`r`	*read*	Öppna för läsning	Läs från början	Fel
`w`	*write*	Skapa för skrivning	Radera innehåll	Skapa ny
`a`	*append*	Öppna för tillägg	Skriv efter slutet	Skapa ny
`r+`	*read extended*	Öppna för läsning/skrivning	Läs från början	Fel
`w+`	*write extended*	Skapa för läsning/skrivning	Radera innehåll	Skapa ny
`a+`	*append extended*	Öppna för läsning/tillägg	Skriv efter slutet	Skapa ny

- Ovanstående öppnar textfiler.
- Ett tillägg av bokstaven `b` öppnar filen som binär, t.ex. `rb+`.

code/fopenclose.c

```c
#include <stdio.h>
#include <string.h>
#include <errno.h>              /* for better error messages */
int main(void)
{
        FILE *in;
        const char *name = "input_file.txt";
        /* open for reading */
        in = fopen(name, "r");
        if (in == NULL) {
                fprintf(stderr, "Failed to open %s for reading: %s\n",
                        name, strerror(errno));
                return -1;
        }
        /* do stuff with the file ... */

        /* close files before exit */
        if (fclose(in)) {
                fprintf(stderr, "Failed to close %s: %s\n", name,
                        strerror(errno));
                return -1;
        }
        return 0;
}
```

Filslut, `EOF`

- Slutet på en fil markeras med ett speciellt slut-på-filen-tecken.
  - C använder sig av ett särskilt värde, `EOF` (*end-of-file)*, för att representera detta.
- Många av input-funktionerna från `stdio.h` returnerar `EOF` som felkod, vilket vi kan använda oss av för att styra programflödet:

```
while ( scanf("%d", &myNumber) != EOF ) {
    printf("%d\n", myNumber);
}
```

Läsa från och skriva till filer

- Motsvarigheten till `printf` och `scanf` för filer heter `fprintf` och `fscanf`.

```
int fprintf(FILE* stream, const char *format, ...);
int fscanf(FILE* stream, const char *format, ...);
char *fgets(char *str, int n, FILE* stream);
```

- `fprintf`
  - fungerar som `printf` men resultatet skrivs till en filpekare, `stream`, i stället för till skärmen.
  - returnerar antalet utskrivna tecken om OK, annars ett negativt värde.
- `fscanf`
  - `fscanf` läser text från en ström i stället för från tangentbordet.
  - returnerar antalet lyckosamma matchningar, vid fel `EOF`.
- `fgets`
  - läser en rad från en fil tills `n-1` tecken lästs, nyradstecken lästs eller filslutstecken lästs.
  - returnerar `str` om allt går bra.
  - returnerar `NULL` om något går fel eller filslut nås utan att något tecken lästs in.

Exempel 1 — Radnumrering

- Antag att vi har ett antal C-kodfiler och att vi vill skriva ut dem på papper med radnumrering.[1]
- Programmet ska heta `rownumbers` och kunna anropas med två filnamn från kommandoraden:

```
rownumbers inFile.c outFile.txt
```

- Programmet ska läsa från `inFile.c` och skapa en radnumrerad kopia i `outFile.txt`.
- Vi antar att ingen av filerna som programmet ska hantera har mer än 1000 rader eller någon rad med mer än 300 tecken.

[1] Det finns många andra hjälpmedel för det, men varför inte skriva ett eget...?

Vad måste programmet göra?

1. Kontrollera att den fått rätt indata:
   1.1 Namn på två filer, en att läsa ifrån och en att skriva till.
2. Öppna filerna.
3. Initiera en radräknare till 1.
4. För varje rad i infilen:
   4.1 Läs in raden.
   4.2 Skriv räknare + raden till utfilen.
   4.3 Räkna upp radräknaren med 1.
5. Stäng filerna.

0. Vi behöver en del variabler. . .

code/rownumbers.c

```
1  #include <stdio.h>
2
3  #define BUFSIZE 300             // Max 300 char per line
4
5  int main(int argc, const char **argv)
6  {
7          char line[BUFSIZE];     // Buffer one line at the time from input
8          FILE *in;               // Pointer to input file
9          FILE *out;              // Pointer to output file
10         int row_number = 1;
```

1. Kontrollera indata

code/rownumbers.c

```
12      // Verify number of parameters
13      if (argc <= 2) {
14              fprintf(stderr, "Usage: rownumbers inFile.c outFile.txt\n");
15              return -1;
16      }
```

- Kontrollera alltid att indatat är det du förväntat dig först av allt.
  - Rätt *antal* parametrar och rätt *typ* av parametrar.
- Är datat felaktigt och det ej går att “reparera”, avbryt programmet omedelbart!

2. Öppna filerna — felhantering

code/rownumbers.c

```
18          // Try to open the input file
19          in = fopen(argv[1], "r");
20          if (in == NULL) {
21                  fprintf(stderr, "Couldn't open input file %s\n", argv[1]);
22                  return -1;
23          }
24
25          // Try to open the output file
26          out = fopen(argv[2], "w");
27          if (out == NULL) {
28                  fclose(in);     /* Overkill, usually safe to ignore
29                                     open-for-read files */
30                  fprintf(stderr, "Couldn't open output file %s\n", argv[2]);
31                  return -1;
32          }
```

- Finns en massa som kan gå fel:
  - Filen finns inte
  - Vi vill skriva till en fil men saknar rättigheter.
  - ...
- Går något fel returnerar `fopen` `NULL`.
- Måste kollas, försöker man använda en filpekare som är `NULL` så kraschar programmet!

3-4. Läsa/skriva till filerna

code/rownumbers.c

```
34          /* Read a line at a time from the input file. Write the line,
35             preceded by a row number, to the output file. */
36          while (fgets(line, BUFSIZE, in) != NULL) {
37                  fprintf(out, "%-3d ", row_number);
38                  fprintf(out, "%s", line);
39                  row_number++;
40          }
```

5. Stänga filerna

code/rownumbers.c

```
42          // Close files and finish.
43          fclose(in);
44          fclose(out);
45          return 0;
46  }
```

rownumbers.c, part 1

```c
#include <stdio.h>

#define BUFSIZE 300             // Max 300 char per line

int main(int argc, const char **argv)
{
        char line[BUFSIZE];     // Buffer one line at the time from input
        FILE *in;               // Pointer to input file
        FILE *out;              // Pointer to output file
        int row_number = 1;

        // Verify number of parameters
        if (argc <= 2) {
                fprintf(stderr, "Usage: rownumbers inFile.c outFile.txt\n");
                return -1;
        }

        // Try to open the input file
        in = fopen(argv[1], "r");
        if (in == NULL) {
                fprintf(stderr, "Couldn't open input file %s\n", argv[1]);
                return -1;
        }

        // Try to open the output file
        out = fopen(argv[2], "w");
        if (out == NULL) {
                fclose(in);     /* Overkill, usually safe to ignore
                                   open-for-read files */
                fprintf(stderr, "Couldn't open output file %s\n", argv[2]);
                return -1;
        }
```

rownumbers.c, part 2

```
34          /* Read a line at a time from the input file. Write the line,
35             preceded by a row number, to the output file. */
36          while (fgets(line, BUFSIZE, in) != NULL) {
37                  fprintf(out, "%-3d ", row_number);
38                  fprintf(out, "%s", line);
39                  row_number++;
40          }
41
42          // Close files and finish.
43          fclose(in);
44          fclose(out);
45          return 0;
46  }
```

Exempel 2 — Skriv ut längsta rad

- Skriv ut den längsta raden i en textfil.
  - Ignorera inledande och avslutade *whitespace* (mellanslag, tab, etc.)
  - Ignorera rader som börjar med tecknet `#`.
    - Möjliggör kommentarsrader.
  - Vi antar att längsta raden är max 300 tecken.
- Programmet ska heta `longest` och kunna anropas med ett filnamn från kommandoraden: `longest file.txt`
  - Programmet ska då läsa raderna från `file.txt`.

Vad måste programmet göra?

1. Kontrollera att den fått rätt indata:
   1.1 Namn på en fil.
2. Öppna filen.
3. Initiera ”längsta kända rad”.
4. För varje rad i infilen:
   4.1 Läs in raden.
   4.2 Ignorera om blank eller kommentarsrad.
   4.3 Räkna längden (utom inledande/avslutande mellanslag).
   4.4 Om längst hittills
      4.4.1 Spara längd och sträng.
5. Stänga filen.
6. Skriv ut längst sträng.

longest.c, part 1

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>            // For bool
#include <ctype.h>              // For isspace(), etc.
#include <errno.h>              // For better error messages

/*
 * longest: Print longest string.
 *
 * Expects first command-line argument to be the name of a text file.
 * Each line is expected to be less than 300 chars long.
 * Blank lines or lines starting with '#' (comment lines) are ignored.
 *
 * longest will process the file and afterward print the longest line
 * (ignoring leading and trailing whitespace).
 *
 */
```

longest.c, part 2

```c
/* Return position of first non-whitespace character or -1 if only
   white-space is found. */
int first_non_white_space(const char *s)
{
        int i = 0; // Start at first char.
        // Advance until we hit EOL as long as we're loooking at white-space.
        while (s[i] && isspace(s[i])) {
                i++;
        }
        if (s[i]) {
                return i; // Return position of found a non-white-space char.
        } else {
                return -1; // Return fail.
        }
}

/* Return position of last non-whitespace character or -1 if only
   white-space is found. */
int last_non_white_space(const char *s)
{
        // Start at last char.
        int i = strlen(s) - 1;
        // Move back until we hit beginning-of-line as long as we're
        // loooking at white-space.
        while (i >= 0 && isspace(s[i])) {
                i--;
        }
        if (i >= 0) {
                return i; // Return position of found a non-white-space char.
        } else {
                return -1; // Return fail.
        }
}
```

longest.c, part 3

```
53  /* Return true if s only contains whitespace */
54  bool line_is_blank(const char *s)
55  {
56          // Line is blank if it only contained white-space chars.
57          return first_non_white_space(s) < 0;
58  }
59
60  /* Return true if s is a comment line, i.e. first non-whitespc char is '#' */
61  bool line_is_comment(const char *s)
62  {
63          int i = first_non_white_space(s);
64          return (i >= 0 && s[i] == '#');
65  }
```

longest.c, part 4

```
67  #define BUFSIZE 300             /* Max 300 char per line */
68
69  int main(int argc, const char **argv)
70  {
71          char line[BUFSIZE];             // Input buffer
72          char longest[BUFSIZE] = "";     // Longest string seen so far
73          int max_len = 0;                // Length of longest string seen so far
74          const char *file_name;
75          FILE *in;
76
77          // Verify number of parameters
78          if (argc <= 1) {
79                  fprintf(stderr, "Usage: longest file.txt\n");
80                  return -1;
81          }
82
83          // Try to open the input file
84          file_name = argv[1];
85          in = fopen(file_name, "r");
86          if (in == NULL) {
87                  fprintf(stderr, "Couldn't open input file %s: %s\n",
88                          file_name, strerror(errno));
89                  return -1;
90          }
```

longest.c, part 5

```
92          // Read a line at a time from the input file until EOF
93          while (fgets(line, BUFSIZE, in) != NULL) {
94                  if (line_is_blank(line) || line_is_comment(line)) {
95                          // Ignore blank lines and comment lines.
96                          continue;
97                  }
98                  // How long is the line, ignoring leading and trailing
99                  // white-space? (We know there is at least one
100                 // non-white-space char.)
101                 int beg = first_non_white_space(line);
102                 int end = last_non_white_space(line);
103                 int len = end - beg + 1;
104                 if (len > max_len) {
105                         // Longer than previous, remember length.
106                         max_len = len;
107                         // Copy string, ignoring leading/trailing whitespace.
108                         strncpy(longest, line + beg, len);
109                         // Add terminating null char.
110                         longest[len] = '\0';
111                 }
112         }
113
114         // Close files before exit
115         if (fclose(in)) {
116                 fprintf(stderr, "Failed to close %s: %s\n",
117                         file_name, strerror(errno));
118                 return -1;
119         }
120
121         // Print result
122         printf("Longest line was: '%s'.\n", longest);
123
124         return 0;
125 }
```

Körningsexempel:

- Kompilera: `gcc -Wall -o longest longest.c`
- Kör: `./longest months.txt`
- Ger utskrift:

```
Longest line was: '30, September'.
```

code/months.txt

```
1  # Days, month name
2  31, January
3  28, February
4  31, March
5  30, April
6  31, May
7  30, June
8  31, July
9  31, August
10 30, September
11 31, October
12 30, November
13 31, December
14
15         line w. spc
16
```

Övning

1. Kombinera `rownumbers.c` och `linkedlist.c` till `reverserownumbers.c` som skriver ut numrerade rader i omvänd ordning.
2. Modifiera `longest` till att tolka raderna som *tal*, *sträng* och skriver ut raden med längst strängen.
3. Modifiera `longest` till att tolka raderna som *tal*, *sträng* och skriver ut raden med högsta talet.

Andra filfunktioner

```c
#include <stdio.h>
int feof(FILE *stream);
int fseek(FILE *stream, long offset, int whence);
long ftell(FILE *stream);
void rewind(FILE *stream);
```

- `ftell` returnerar positionen för filpekaren.
- `fseek` sätter positionen för filpekaren.
  - `whence` är en av tre konstantvärden:
    - `SEEK_SET` början av filen
    - `SEEK_CUR` nuvarande position i filer
    - `SEEK_END` slutet av filen
- `rewind` sätter filpekaren till början av filen.
- `feof` returnerar sant om filpekaren är vid filens slut.

Binära filer

- Om ett program vill lagra data som ska läsas av ett annat C-program kan den göra det i en binär fil utan att bry sig om att konvertera till strängar.
- Programmet lagrar helt enkelt datorns interna representation av data.
- Ett annat program kan sedan läsa den binära filen, förutsatt att det känner till vad det är som lagrats.
- Lägg till ett `b` till den andra inparametern i `fopen`:

```
in = fopen("myFile.bin", "rb");
```

Binära filer

- För att skriva binärdata till en fil används `fwrite`

```
size_t fwrite (const void *array, size_t size, size_t count,
               FILE *stream);
```

  - `array` är en pekare till en plats i minnet, där `fwrite` ska börja läsa.
  - `size` är storleken på den datatyp som ska skrivas till filen, angedd i bytes.
  - `count` är antalet element vi vill skriva till filen.
  - `stream` är filpekaren.

- `fwrite` kommer att skriva de första `count` blocken av storlek `size` från minnet, med start vid `array`, till filen kopplad till `stream`.
- `fread` returnerar antalet framgångsrikt skrivna block.

Binära filer

- För att läsa binärdata från en fil används `fread`

```
size_t fread (void *array, size_t size, size_t count,
              FILE *stream);
```

  - `array` är en pekare till en plats i minnet, där `fread` ska börja skriva.
  - `size` är storleken på den datatyp som ska läsas från filen, angedd i bytes.
  - `count` är antalet element vi vill läsa från filen.
  - `stream` är filpekaren.

- `fread` kommer att läsa de första `count` blocken av storlek `size` från filen kopplad till `stream` och skriva till minnet, med start vid `array`.
- `fread` returnerar antalet framgångsrikt lästa block.

bindump.c, part 1

```c
#include <stdio.h>
#include <ctype.h>              // for isprint()

/*
 * bindump: Print content of binary file.
 *
 * Expects first command-line argument to be the name of a file.
 * Reads each char from the file and outputs its:
 * - position,
 * - value as integer (decimal and hexadecimal),
 * - value as character, or '.' if not printable.
 *
 */

int main(int argc, const char **argv)
{
        FILE *in;               // Pointer to input file
        int pos = 0;            // Counter for file position

        // Verify input arguments
        if (argc < 2) {
                fprintf(stderr, "Usage: bindump filename\n");
                return -1;
        }

        // Try to open the input file
        in = fopen(argv[1], "rb");
        if (in == NULL) {
                fprintf(stderr, "Couldn't open input file %s\n", argv[1]);
                return -1;
        }
```

bindump.c, part 2

```
33          // Repeat until end-of-file
34          while (!feof(in)) {
35                  unsigned char c;
36                  if (fread(&c, sizeof(c), 1, in)) {
37                          pos++;
38                          // Convert character to integer (required by printf)
39                          int i = c;
40                          // Print c or a dot if not printable
41                          if (!isprint(c)) {
42                                  c = '.';
43                          }
44                          printf("%3d: dec: %d, hex: %02x, char: %c\n", pos,
45                                 i, i, c);
46                  }
47          }
48
49          /* Close file and finish. */
50          fclose(in);
51
52          return 0;
53  }
```