Lecture 9 - data files

Input and output

Remark Input and output functions are not the part of the language.

Standard input and output

In the standard input and output library a simple character model of input and output is implemented.
Stream of characters is built of strings of lines; each line is ended with the new line character.

Reading one character from the standard input:

```
int getchar(void);
```

This function returns after each call next character from the input or EOF, if the end of file is met.

Constant EOF is defined in the header file ¡stdio.h¿.

Writing one character of data

```
int putchar(int );
```

This function returns value of the written character or EOF (as an indicator of an error).

In each source file of the program that makes use of the library functions realizing input-output operations, the following statement must appear before the first call:

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

For programs reading only one data stream and writing only to one output stream use of **getchar, putchar** and **printf** is sufficient.
Example:

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

main()   /* lower: change capital letters to small  */
{
   int c;
   while ((c=getchar()) != EOF)
      putchar(tolower(c));
   return 0;
}
```

Formatted output - function *printf*

```
   int printf(char *format,arg1, arg2,...);
```

The returned value is equal to the number of successfully printed variables.
Remark The *printf* has got a variable list of arguments, whose number is determined on the basis of the first of them. Therefore:

```
printf(s); /* Dangereous; bad,
           if there  %  is in s  */
printf("%s",s);  /* Safe        */
```

Function *sprintf*:

```
int sprintf
  (char *string, char *format, arg1, arg2,...);
```

Formatted input - function *sscanf*

```
int scanf(char *format, ... );

int sscanf
  (char *string, char *format, arg1, arg2, ... );
```

Argument *format* specifies format of the output; *arg1, arg2, etc.*. must be pointers (they indicate the place of storing the input data). The *sscanf* function ends its execution either after interpreting the whole format or if a data type is not consistent with the specification.

It returns either the number of correctly read and stored data or EOF character if the end of file is met.

Data files

A file may be opened by means of the library function *fopen*. Below necessary declarations are given:

```
FILE *fp;
FILE *fopen(char *name, char *mode);
```

The **fopen** function returns a pointer to a certain structure with composition defined in the header file *¡stdio.h¿* of name **FILE**, that contains the following informations of the file:

- buffer placement,
- current character position in the buffer, type of access to the file (reading, writind, etc.)
- signals of an error or the end of file appearance.

Call of the *fopen* function inside a program has the following form:

```
   fp=fopen(name, mode);
```

Meaning of arguments:

name - name of the file

mode - access type:

- reading "r"
- writing "w"
- appending "a"

Some comments on the work with files:

- opening of a nonexisting file causes its creation,
- opening for writing of an existing file destroys its previous content,
- opening of an exisitnig file to append presrves its previous content,
- any trial to read from a nonexisting file is an error,
- if *fopen* cannot open a file it returns value NULL.

Reading from file and writing to file

```
int getc(FILE *fp); /* to read character  */
/* returns  character or  EOF, if error   */

int putc(FILE *fp); /* to write character  */
/* returns value of the character or EOF if error  */
```

Formatted reading and writing

```
int fscanf(FILE * fp, char *format, ... );

int fprintf(FILE *fp, char *format, ... );
```

While starting execution of a program the operating system is responsible for opening three files and giving the program their pointers:

stdin - standard input (normallly connected with the keyboard)

stdout - standard output (normallly connected with the screen

stderr - standard errors output (also connected with screen)

Example of the program concatenating the contents of two files:

```
#include <stdio.h>
/* cat: concatenate the files contents; version 1 */
main (int argc, char *argv[])
{
   FILE *fpstr;
   void filecopy(FILE *, FILE *);
   if (argc == 1) /* without arguments;
                     copy from  stdin */
     filecopy(stdin, stdout);
   else
     while (--argc > 0)
       if ((fpstr = fopen(*++argv,"r")) == NULL) {
         printf("cat: I cannot open %s\n",*argv);
         return 1;
       } else {
         filecopy(fpstr, stdout);
         fclose(fpstr);
       }
   return 0;
}
/* filecopy: copy content of file ifp
   to file ofp  */
void filecopy(FILE *ifp, FILE *ofp)
{
   int c;
   while ((c=getc(ifp)) != EOF)
     putc(c,ofp);
}
```

Error service - file *stderr* and function *exit*

Corrected program cat (so that it writes signals of errors on the screen):

```
#include <stdio.h>
#include <stdio.h>
/* cat: concatenate file content; version  2 */
main (int argc, char *argv[])
{
   FILE *fpstr;
   void filecopy(FILE *, FILE *);
   char *prog = argv[0];
   /* name of program generating signals */

   if (argc == 1) /* without arguments:
                     copy from stdin */
     filecopy(stdin, stdout);
   else
     while (--argc > 0)
       if ((fpstr = fopen(*++argv,"r")) == NULL) {
         fprintf(stderr,
            "%s: I cannot open  %s\n",prog,*argv);
         exit(1);
       } else {
         filecopy(fpstr, stdout);
         fclose(fpstr);
       }
   if (ferror(stdout)) {
      fprintf(stderr,
         "%s: Writing erros to  stdout\n", prog);
      exit(2);
   }
   exit(0);
}
```

Entering and outputting text

```
char *fgets(char *line, int maxline, FILE *fp);
char *fputs(char *line, FILE *fp);
```

Function **fgets** reads subsequent line from the file pointed by **fp** (together with the newline character) and stores it in an character array **line**.
Returns:

a) normally - pointer to **line**;

b) **NULL** - after detecting an error or meeting the end of file.

Function **fputs** prints out an indicated text (it doesn't need to contain the new line character).
Normally returns 0, in the case of an error - **EOF**.

Realization of **fgets** and **fputs** in the standard library:

```
/* fgets: take at least  n  characters
   from file  iop */
char *fgets(char *s, int n, FILE *iop)
{
   register int c;
   register char *cs;
   cs=s;
   while (--n > 0 && (c=getc(iop)) != EOF)
      if ((*cs++=c) == '\n')
        break;
      *cs='\0';
      return (c==EOF && cs ==s) ? NULL:s;
}
/* fputs: print out  s  to file  iop */
int fputs(char *s, FILE *iop)
{
   int c;
   while (c=*s++)
      putc(c,iop);
   return ferror(iop) ? EOF:0;
}
```

large **Functions determining positions inside a file**

```
int fseek(FILE *stream, long offset, int origin);
```

Function **fseek** determines position in stream **stream**; next reading or writing will start from data at that new position. At binary files new position will be at place of **offset** characters distance from the reference point (**origin**); **origin** may have values:

```
SEEK_SET - beginning of file,
SEEK_CUR - current position,
SEEK_END - end of file.
```

At text files value **offset** must be equal either to zero or value returned by previously called function **ftell** (in that case **origin** must be equal to value `SEEK_SET`).

Function **fseek** returns nonzero value when an error appears.

```
long ftell(FILE *stream);
```

Function **ftell** returns the current position value in the **stream** or **-1L** in the case of an error.

```
void rewind(FILE *stream);
```

Call of the **rewind(fp)** function is equivalent to the following sequence of calls

```
fseek(fp,0L,SEEK_SET);    clearerr(fp);
```

Prototype of function storing the current position:

```
int fgetpos(FILE *stream, fpos_t *ptr);
```

Function **fgetpos** stores the current position in stream **stream** in place pointed by ***ptr**. Later one may use it in function **fsetpos**. Type `fpos_t` is an appropriate type of object used to store such value. In the case of error function **fgetpos** returns value different from zero.

```
int fsetpos(FILE *stream, const fpos_t *ptr);
```

Function **fsetpos** sets the current stream **stream** position to value stored previously by function **fgetpos** in place pointed by ***ptr**. In the case of an error function **fsetpos** returns value different from zero.