Reading from and writing into files

Timo Karvi

September, 2012

File pointers I

- Files are handled through file pointers which are defined using the type FILE:

```
FILE* fp;
```

- A concrete file can be opened and connected to `fp` using `fopen` function:

```
fp = fopen("fileA", "r");
```

  Here, "fileA" is a file name and "r" means that the file is opened for reading. Instead of "r", other possibilities are: "w" (write), "a" (append), "rb" (read binary), "wb" (write binary).

- A file is closed with `fclose` command:

```
fclose(fp);
```

- In the next example we use these functions with `putc` and `puts`. The first prints a character into a stream, the latter prints a string into the standard stream. Streams are either standard streams (screen, keyboard) or user defined (for example files).

Example 1: Writing characters to a disk file I

```
/* This program reads characters typed at the keyboard
   and writes them
   one at a time to a disk file.
*/

#include <stdio.h>

int main(void)
{
  FILE *fptr;
  char ch;
  puts("Enter what you want to save in the file.");
  puts("End your input with a Return key.");

  fptr = fopen("file-a.txt","w");
```

Example 1: Writing characters to a disk file II

```
  while ((ch = getchar()) != '\n')
    putc(ch, fptr);
  fclose(fptr);

  puts("Your input string is saved in file-a.txt");
  puts("End of my act!");

return 0;
}
```

Example 2: Reading from and writing characters to a disk file I

- In Example 2, we use the previous i/o functions, this time with error checking. The " character in the fopen statement means that the file is openend for reading and writing.
- In addition, we the `exit(int)` statement. It stops the execution of the program and outputs the error code given as a parameter.
- Second extra function is the `rewind` function. `rewind(fptr)` moves the file position indicator to the beginning of the stream pointed by fptr.

Example 2: Reading from and writing characters to a disk file II

```
/* This program reads characters typed at the keyboard
   and writes them
   one at a time to a disk file,
   then reads from the same file and
   displays the characters.
*/

#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  FILE *fptr;
  int ch;
  puts("Enter what you want to save in the file.");
```

Example 2: Reading from and writing characters to a disk file III

```
puts("End your input with a Return key.");

if ((fptr = fopen("file-a.txt", "w+")) == NULL)
{
  printf("Failed to open file: file-a\n");
  exit(1);
}

while ((ch = getchar()) != '\n')
    putc(ch, fptr);

rewind( fptr);

puts("The contents of the file file-a.txt is : ");
```

Example 2: Reading from and writing characters to a disk file IV

```
while ((ch = getc(fptr)) != EOF)
    putchar(ch);

fclose(fptr);

puts("\nEnd of my act! :-) ");

return 0;
}
```

Example 3: Reading and writing strings I

- In Example 3, we use the library `string` and its functions.
- `fgets` reads a string from a stream to an array. The third parameter is the length of the string. (The other option to read a string from the standard input is gets, but its use is dangerous, because it has no parameter for the length of the string. That's why its use is prohibited or at least the system gives a warning.)
- `strlen` gives the elngth of the string.
- `fputs` writes a string to a file.
- We add a newline character to the end of the string with `strcat` function.

Example 3: Reading and writing strings II

```
/* This program reads and writes strings from and to a file.
*/


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

int main(void)
{
  FILE *fptr;
  char string [ 81 ];
  puts("Enter what you want to save in a file.");
  puts("End each line with a Return key.");
  puts("Enter a Return key at the beginning of the line to EN
```

Example 3: Reading and writing strings III

```
if ((fptr = fopen("file-a.txt", "w+")) == NULL)
  {
    printf("Failed to open file: file-a.txt\n");
    exit(1);
  }

while ((strlen(fgets(string,81,stdin))) > 1)
  {
    strcat(string, "\n");
    fputs(string, fptr);
  }

rewind(fptr);

puts("The contents of the file file-a.txt are: ");
```

Example 3: Reading and writing strings IV

```
  while ((fgets(string, 80, fptr)) != NULL)
    printf("%s", string);

  fclose(fptr);

  puts("End of my act! :-) \n");

  return 0;
}
```

Reading and writing binary data I

- Two functions from the `stdio` library are used to perform binary I/O:

```
int fread( void* ar, size_t sz, size_t, count, FILE* fptr)
int fwrite( void* ar, size_t sz, size_t, count, FILE* fptr
```

- In `fread`, the first parameter is is a pointer to the array in which the data will be stored. The type `void*` is used, because the function must be able to read data into an array, whose type is unknown. The function need not know the base type of the array, because it simply copies the bytes of data directly from the stream into the memory; no conversion is done.
- The second parameter specifies the size of the base type of the data being read.
- The third parameter indicates how many data items will be read in one input operation.
- The function returns the number of bytes actually read.

Reading and writing binary data II

```
/* This program reads and writes a structure to a disk file.
    User-defined
   functions are defined to handle the file I/O.
*/

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


struct car {
  char maker [ 40 ], model [ 40 ];
  int year; };

FILE * fp;
```

Reading and writing binary data III

```
int main(void)
{
  struct car FirstCar, SecondCar;

  /* declaring functions */

  void input(struct car * pointer);
  void open_it(void);
  void write_it(struct car);
  struct car read_it(void);
  void output(struct car temp);
  void close_it(void);

  input(&FirstCar);
  open_it();
  write_it(FirstCar);
```

Reading and writing binary data IV

```
  rewind(fp);
  SecondCar = read_it();
  output(FirstCar);
  output(SecondCar);
  fputs("End of my act! :-) \n", stdout);

  return 0;
}
```

Reading and writing binary data V

```
void input(struct car * pointer)
{

  /* This function takes a pointer to a structure of
     car type.
     It reads data items from the keyboard and
     saves them in the
     specified structure members.
  */

  fputs("*** Car Information System ***\n", stdout);
  fputs("Maker? ", stdout);
  gets(pointer -> maker);
  fputs("Model? ", stdout);
  gets(pointer -> model);
  fputs("Year? ", stdout);
```

Reading and writing binary data VI

```
    fscanf(stdin, "%d", &pointer -> year);
    return;
}
```

Reading and writing binary data VII

```
void output(struct car temp)
{
  /* This function takes a structure of type car.
     It displays the member
     variables of the specified structure.
  */

  fprintf(stdout, "your car: %s, %s, %d\n",
          temp.maker, temp.model,
             temp.year);
  return;
}
```

Reading and writing binary data VIII

```
void open_it(void)
{

  /* This function opens the binary file of cars
     for read and write. */

  if ((fp = fopen("CarInfo.bin", "wb+")) == NULL)
  {
    fprintf(stderr, "Failed to open file: CarInfo.bin\n");
    exit(1);
  }
  return;
}
```

Reading and writing binary data IX

```
void close_it(void)
{
  fclose(fp);
}
```

Reading and writing binary data X

```
void write_it(struct car temp)
{
  fwrite(&temp, sizeof(temp), 1, fp);
  return;
}

struct car read_it(void)
{
  struct car temp;
  fread(&temp, sizeof(temp), 1, fp);
  return(temp);
}
```

Command line arguments I

- The next example shows how to use command line arguments. Similarity with Java is clear.
- We read the file from the end to the beginning and for this we need new file functions.
- `int fseek( FILE *stream, long offset, int origin )` sets the file position data for the given stream. The origin variable can have one of the following values:
  - `SEEK_SET`: Seek from the start of the file.
  - `SEEK_CUR`: Seek from the current location.
  - `SEEK_END`: Seek from the end of the file.
- `long ftell( FILE *stream )` returns the current file position for stream, or -1 if an error occurs.

Command line arguments II

```
/* This program accepts command-line arguments and displays th
   file in reverse order. In our example, this file is called
   reverse.fil, and contains the gibberish sequence of chracte
   follows:

  .C gninihs ot C morF

*/


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

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

Command line arguments III

```
FILE * fpt;

FILE * open_it(char * filename, char * filemode);
void reverse(FILE * filename);


if (argc < 2)
  {
    fprintf(stdout, "You forgot the filename!\n");
    exit(1);
  }

fpt = open_it(argv[1], "r");
fprintf(stdout, "Contents of the %s file reversed:\n",
                  argv[1]);
reverse(fpt);
```

Command line arguments IV

```
  fclose(fpt);
  printf("\n");
}


FILE * open_it(char * filename, char * filemode)
{
  FILE *fp;
  if ((fp=fopen(filename, filemode)) == NULL)
    {
      fprintf(stderr, "Failed to open file: %s\n", filename);
      exit(1);
    }
  return(fp);
}
```

Command line arguments V

```
void reverse(FILE * filename)
{
  long last, count;
  char ch;

  /* move the file position indicator
     to the end of the file */

  fseek(filename, 0L, SEEK_END);

  /* store the value of the file position indicator
     in the variable last */

  last = ftell(filename);
```

Command line arguments VI

```
  for (count = 1L; count <= last; ++count)
    {
      fseek(filename, -count, SEEK_END);
      ch = getc(filename);
      putc(ch,stdout);
    }
  return;
}
```