

File handling

Problem:
Data that survives the program execution

Solution:
Storing data in files

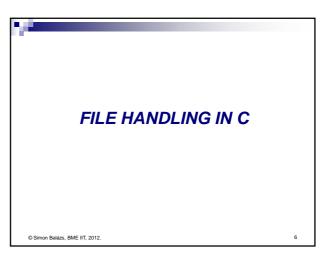
Tasks:
Write data to file from memory
Read data from file to memory

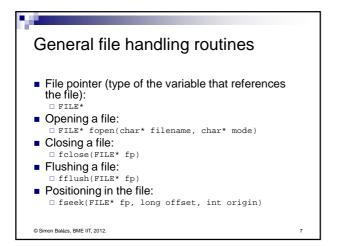
Typical file handling routines

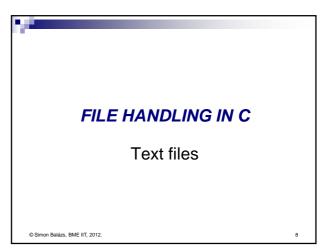
Create new file
Open existing file
Read from file
Write to file
Close file

File modes

Text file
Like a TXT file
Humanly readable
Divided into lines
Binary file
Computer readable (humanly not really)
Exact format is application specific







```
Text files
Opening and closing a text file:
#include <stdio.h>
                                         Opening mode:
                                           open for reading
int main()
                                          w - open or create new for writing
                                         a - open or create new for append
                                         r+ - open for reading and writing,
     FILE* fp = NULL;
                                             start at the beginning
     fp = fopen("a.txt", "rt");
                                         w+ - overwrite for reading and
                                             writing, start at the beginning
     fclose(fp);
                                            - open for reading and writing,
}
                                             start at the end
```

```
Writing text files

Like printf, but into a file:
fprintf(FILE* fp, char* format, ...)

A single character:
fputc(int c, FILE* fp)
```

Reading text files I.

Like scanf, but from a file:

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

returns the number of items successfully read, or EOF if end of file reached

A single character:

int fgetc(FILE* fp)

returns 0..255 if a valid character is read, or EOF if end of file reached

Reading text files II.

Safely read a single line into string:

char* fgets(char* str,
 int maxlength,
 FILE* fp)

maxlength is the maximum length of the string
inlocluding the terminating zero character

reads at most maxlength-1 characters, or less if end
of line or end of file is reached
returns the str pointer on success, or NULL on failure

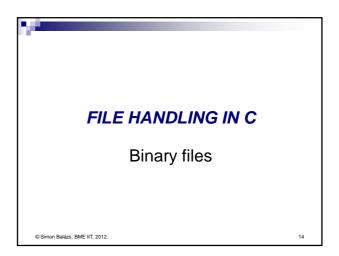
Reading from a string (use after fgets()):

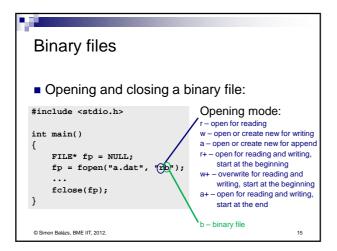
int sscanf(char* str, char* format, ...)
returns the number of items successfully read

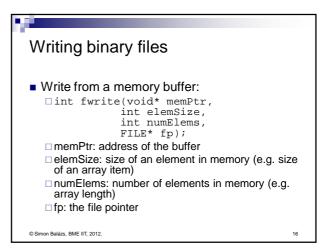
```
Text file handling sample
#include <stdio.h>

int main()
{
    FILE* fp = NULL;
    char line[80];
    int i;
    double d;

    fp = fopen("a.txt", "rt");
    while (fgets(line, 80, fp) != NULL)
    {
        sscanf(line, "%d %lf", &i, &d);
    }
    fclose(fp);
}
```







```
Reading binary files

Read into a memory buffer:

int fread(void* memPtr,
 int elemSize,
 int numElems,
 FILE* fp);

memPtr: address of the buffer
elemSize: size of an element in memory (e.g. size of an array item)
numElems: number of elements in memory (e.g. array length)
fp: the file pointer
returns the number of items actually read
```

Checking end of file To check whether the end of file is reached: int feof(FILE* fp) returns non-zero if end of file is reached Only after an unsuccessful read!

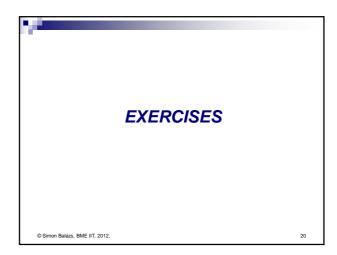
```
Binary file handling sample

#include <stdio.h>

int main()
{

   FILE* fp = NULL;
   int count = 0;
   double d[10];

   fp = fopen("a.dat", "rb");
   do
   {
      count = fread(d, sizeof(double), 10, fp);
      for (int i = 0; i < count; ++i)
      {
            printf("%lf\n", d[i]);
      }
      while (!feof(fp));
      fclose(fp);
}
```



File handling exercises Text files 1. Read in lines from a text file and print them to the standard output 2. Read in lines from a text file and print those lines that contain "apple" into another text file Hint: char* strstr(char* str1, char* str2) Binary files 3. Read integer numbers from the standard input and write them into a binary file 4. Read integer numbers from a binary file and print them to the standard output 5. Read/write complex numbers from/to binary files