Multidimensional Arrays – File handling

Basics of Programming 1

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

25 October, 2023

Content

Chapter 1

Multi-dimensional arrays

Multi-dimensional arrays

1D array	Elements of the same type, stored in the memory beside eachother
2D array	1D arrays of the same size and same type, stored in the memory beside eachother
3D array	2D arrays of the same size and same type, stored in the memory beside eachother
...	...

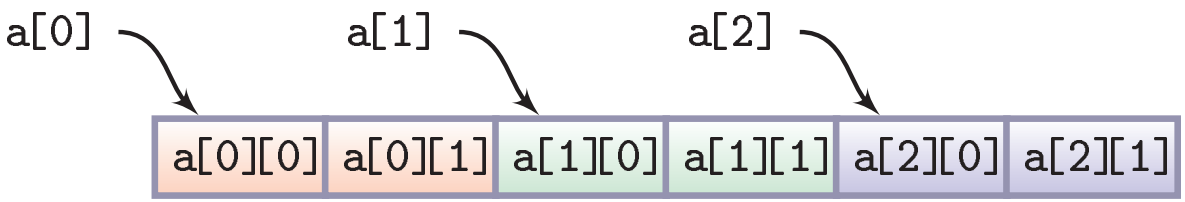
Two-dimensional

- Declaration of a 2D array:

```c
char a[3][2]; /* 3row x 2column array of characters */
              /* 3-sized array of 2-sized 1D arrays */
```

a[0][0]	a[0][1]
a[1][0]	a[1][1]
a[2][0]	a[2][1]

- In C language, storage is done row by row (the second index changes quicker)



- a[0], a[1] and a[2] are 2-sized 1D arrays

Taking a 2D array row by row

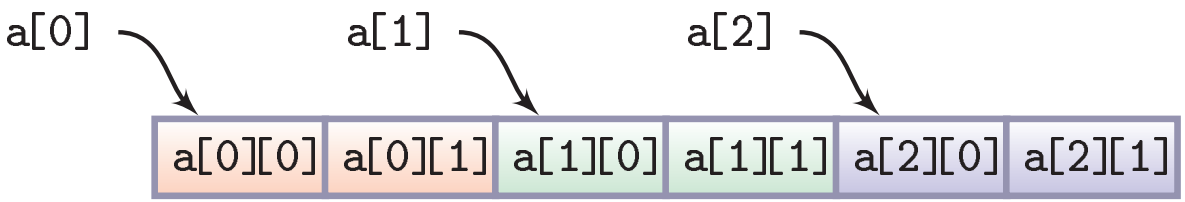
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Filling a 1D array (row) with the given element

```c
void fill_row(char row[], size_t size, char c)
{
  size_t i;
  for (i = 0; i < size; ++i)
    row[i] = c;
}
```

- Filling a 2D array row by row

```c
char a[3][2];
fill_row(a[0], 2, 'a'); /* row 0 is full of 'a' */
fill_row(a[1], 2, 'b'); /* row 1 is full of  'b' */
fill_row(a[2], 2, 'c'); /* row 2 is full of  'c' */
```



Taking a 2D array as one entity

- taking as a 2D array – only if number of columns is known

```
1  void print_array(char array[][2], size_t nrows)
2  {
3    size_t row, col;
4    for (row = 0; row < nrows; ++row)
5    {
6      for (col = 0; col < 2; ++col)
7        printf("%c", array[row][col]);
8      printf("\n");
9    }
10 }
```

- Usage of the function

```
1  char a[3][2];
2  ...
3  print_array(a, 3);        /* printing a 3-row array */
```

Taking a 2D array as one entity

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- taking 2D array as a pointer

```c
void print_array(char *array, int nrows, int ncols)
{
  int row, col;
  for (row = 0; row < nrows; ++row)
  {
    for (col = 0; col < ncols; ++col)
      printf("%c", array[row*ncols+col]);
    printf("\n");
  }
}
```

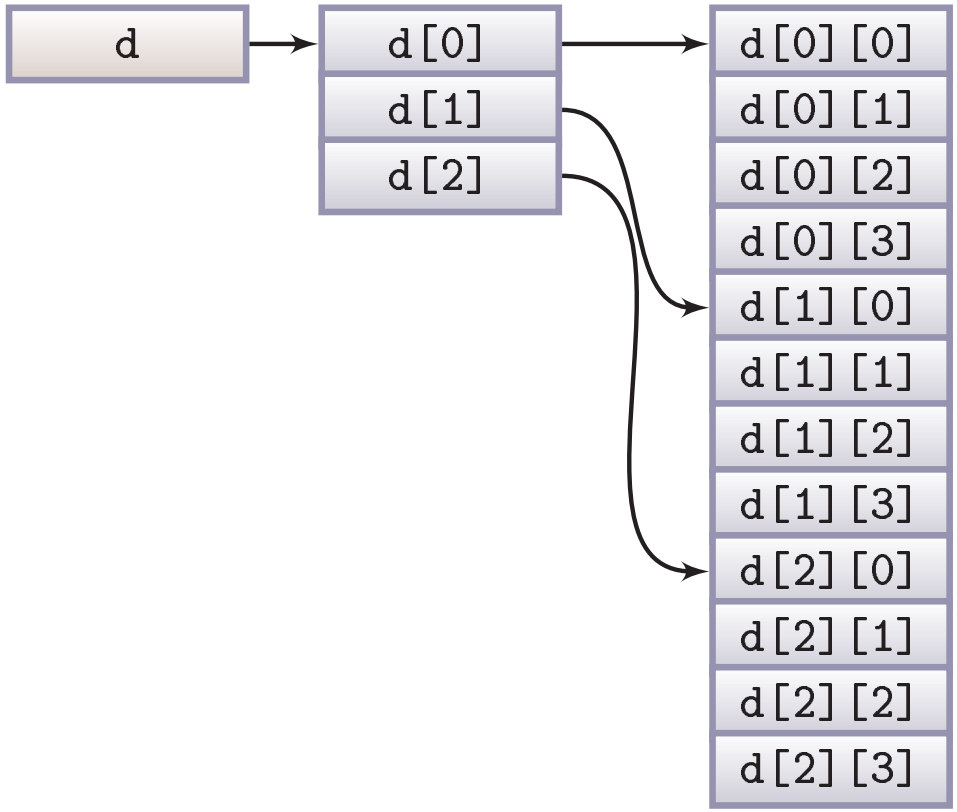
- Usage of the function

```c
char a[3][2];
...
print_array((char *)a, 3, 2); /* 3 rows 2 columns */
```

Dynamic 2D array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's allocate memory for a 2D array. We would like to use the conventional way of indexing for the array `d[i][j]`

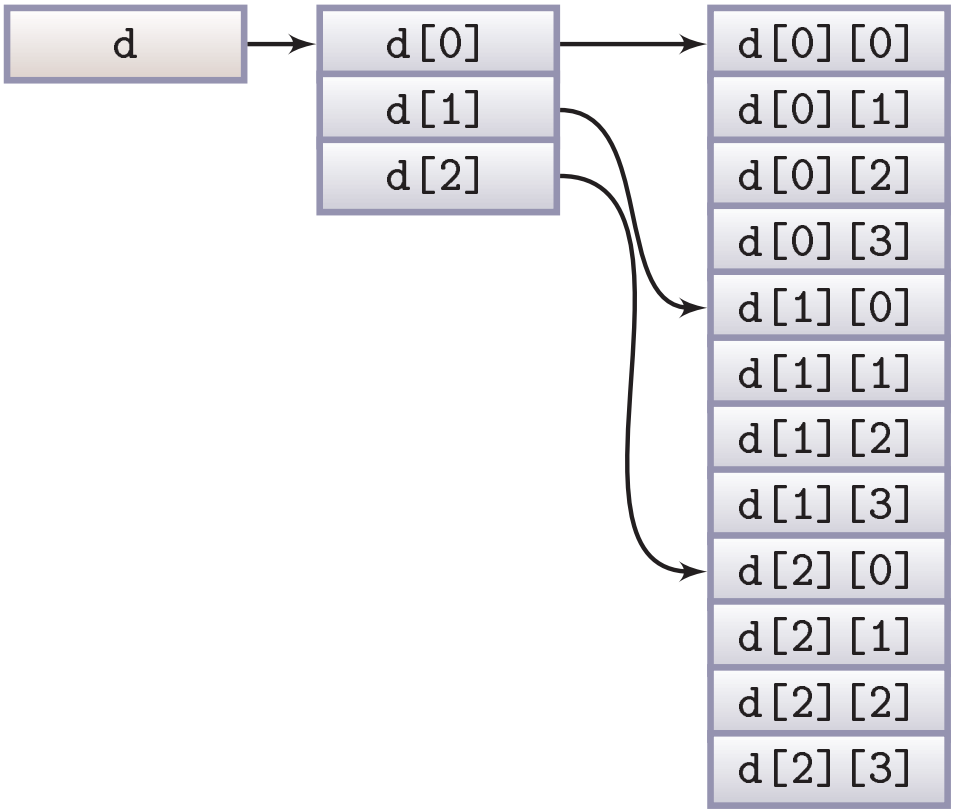



```
1  double **d =(double**)malloc(3*sizeof(double*));
2  d[0] = (double*)malloc(3*4*sizeof(double));
3  for (i = 1; i < 3; ++i)
4    d[i] = d[i-1] + 4;
```

Dynamic 2D array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Releasing the array



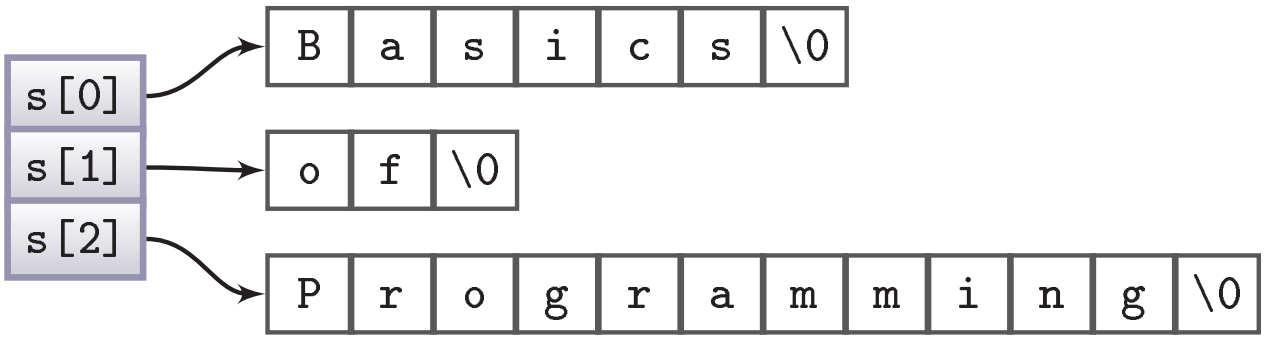

```
1  free(d[0]);
2  free(d);
```

Array of pointers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Defining an array of pointers and passing it to a function

```
1 char *s[3] = {"Basics", "of", "programming"};
2 print_strings(s, 3);
```



- Taking an array of pointers with a function

```
1 void print_strings(char *strings[], size_t size)
2 /*                   char **strings is also possible   */
3 {
4   size_t i;
5   for (i = 0; i < size; ++i)
6     printf("%s\n", strings[i]);
7 }
```

Chapter 2

File handling

Files

File

Data stored on a physical media (hard disk, CD, USB drive)

- Data stored in a file is not lost after the program is finished, it can be reloaded.
- Independently of the media, files are handled in a uniform way
- File handling:
  1. Opening the file
  2. Data writing / reading
  3. Closing the file
- Two types of files:
  - Text file
  - Binary file

Text vs. Binary

Text file – contains text, divided into lines

- `txt`, `c`, `html`, `xml`, `rtf`, `svg`

Binary file – contains binary coded data of arbitrary structure

- `exe`, `wav`, `mp3`, `jpg`, `avi`, `zip`

- As long as it makes sense, use a text file – it is more friendly.
- It is a big advantage, if not only programs, but humans too are able to read and edit our data.

Writing into a text file

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
#include <stdio.h> /* fopen, fprintf, fclose */
int main(void)
{
  FILE *fp;
  int status;

  fp = fopen("hello.txt", "w");  /* file open */
  if (fp == NULL)                 /* no success */
    return 1;

  fprintf(fp, "Hello, World!\n");  /* writing */

  status = fclose(fp);             /* closing */
  if (status != 0)
    return 1;

  return 0;
}
```

link

Opening a file


```
FILE *fopen(char *fname, char *mode);
```

- Opens the file whose name is specified in `fname` string, according to the mode given in `mode` string
- Main methods for text files:

mode		description
`"r"`	read	reading, the file must exist
`"w"`	write	writing, overwrites, if needed a new is created
`"a"`	append	writing, continues at the end, if needed a new is created

- return value is a pointer to a `FILE` structure, this is the identifier of the file
- If opening was not successfull, it returns with NULL

Closing a file


```
int fclose(FILE *fp);
```

- It closes the file referenced by the `fp` identifier
- If the closing is successful[1], it returns with 0, otherwise it returns with `EOF`.

[1] closing a file may not be successful: for example somebody has removed the pendrive while we were writing onto it.

Writing onto screen / into text file / into string

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
int  printf(           char *control, ...);
int fprintf(FILE *fp,  char *control, ...);
int sprintf(char *str, char *control, ...);
```

- The text given in the `control` string will be written
  - onto the screen
  - into a text file (previously opened for writing) with `fp` identifier
  - into a string with `str` identifier (string must be long enough)
- Using of control character (eg. `%d`) is the same as with `printf`
- Return value is the number of successfully written characters[2], it is negative in case of error

[2] If we write into a string, it automatically adds the terminating 0, but it is not counted in the return value

Reading from keyboard / text file / string


```
int  scanf(           char *control, ...);
int fscanf(FILE *fp,  char *control, ...);
int sscanf(char *str, char *control, ...);
```

- Reads in the format specified in the `control` string from the
  - keyboard
  - a text file (previously opened for reading) with `fp` identifier
  - from a string with `str` identifier
- Return value is the number of read elements, it is negative in case of error

Reading from text file

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that prints (onto the screen) the content of a text file

```c
#include <stdio.h>
int main()
{
  char c;
  FILE *fp = fopen("file.txt", "r"); /* open file */
  if (fp == NULL)
    return -1; /* was not successfull */

  /* reading until successful (we read 1 character) */
  while (fscanf(fp, "%c", &c) == 1)
    printf("%c", c);

  fclose(fp); /* close file */
  return 0;
}
```

link

- Memorize the way we read until the end of the file!

Reading from text file

A text file contains the coordinates of 2D points. Each of its line has the following format
`x:1.2334, y:-23.3`
Let's write a program that reads and processes the coordinates!

```
1  FILE *fp;
2  double x, y;
3  ...
4  /* reading as long as it is successful */
5  /* (we read 2 numbers)                 */
6  while (fscanf(fp, "x:%lf, y:%lf", &x, &y) == 2)
7  {
8    /* processing */
9  }
```

- Once again, take a look at how we read until the end of the file!

Keyboard? Monitor?


```
1 scanf("%c", &c);
2 printf("%c", c);
```

- The code segment above does not read directly from the keyboard and does not write directly onto the screen, but it reads from standard input (stdin), and writes to the standard output (stdout)
- stdin and stdout are text files
- The type of perphery or other file that is assigned to it depends on the operating system.
- Its default interpretation is as in the figure.
  - keyboard (through a console application) → stdin
  - stdout → (through a console application) monitor

Redirecting

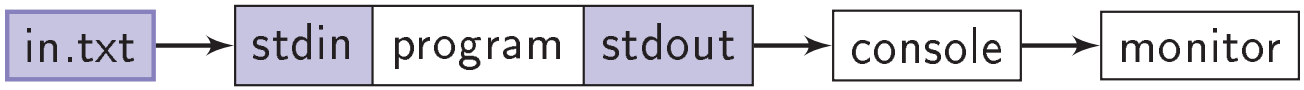
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If we start our program in the follwoing way, we can redirect the standard output: it will not print on the monitor, but into the `out.txt` text file

```
c:\>prog.exe > out.txt
```

- The standard input can also be redirected to a text file.

```
c:\>prog.exe < in.txt
```



- Of course, the 2 can be combined

```
c:\>prog.exe < in.txt > out.txt
```

stdin and stdout

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- stdin and stdout are text files that are automatically opened when starting the program
- the code segments below are equivalent

```c
char c;
printf("Hello");
scanf("%c", &c);
printf("%c", c);
```

```c
char c;
fprintf(stdout, "Hello");
fscanf(stdin, "%c", &c);
fprintf(stdout, "%c", c);
```

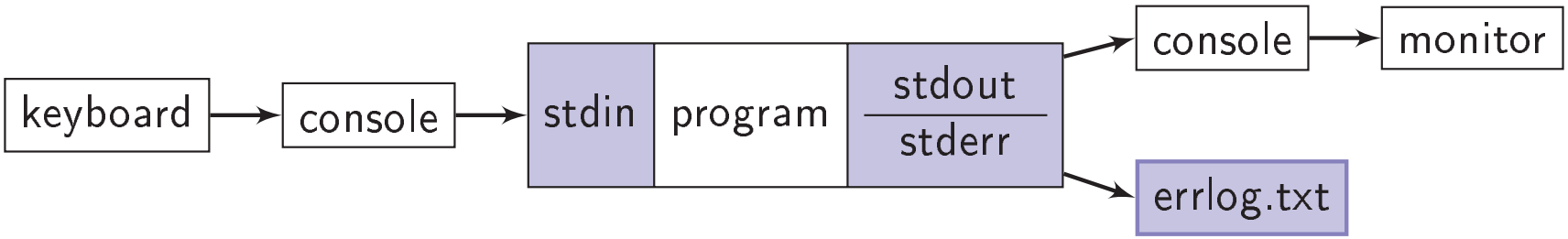
- When writing data from a text file into a text file, instead of opening a file, use the standard input and output, and the redirection options of the operating system
- We can read from the console also until the end of the file: we can emulate the end of file by entering Ctrl+Z (windows) or Ctrl+D (linux).

stdout and stderr

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The output and the error messages of the program can be separated by using the standard error output `stderr`

```
c:\>prog.exe 2> errlog.txt
```




```c
if (error)
{
  /* useful information for the user */
  printf("Please, switch it off\n");
  /* detailed information to the error output */
  fprintf(stderr, "Error code 61\n");
}
```

Binary files

- Binary file: The bit-by-bit copy of the content of the memory onto a physical data media
- The actual data depends on the inner representation
- Use it only if storing as text would be very weird – and use it in tasks if asked 
- Opening and closing the file is similar to the case of text files, but now the `b` character must be used in the `mode` string[3]

`mode`		description
`"rb"`	read	reading, the file must exist
`"wb"`	write	writing, overwrites, if needed a new is created
`"ab"`	append	writing, continues at the end, if needed a new is created

[3] For the sake of analogy, in case of text file it is typical to use `t` (`text`), but actually `fopen` will not care about it.

Reading and writing a binary file


```
size_t fwrite (void *ptr, size_t size,
               size_t count, FILE *fp);
```

- Starting from address `ptr`, it writes `count` elements (that are placed one after the other in the memory), each having `size` size into a file with `fp` identifier
- Return value is the number of written elements.

```
size_t fread (void *ptr, size_t size,
              size_t count, FILE *fp);
```

- It reads `count` elements, each having `size` size from the file with `fp` identifier to the address `ptr`
- Return value is the number of read elements

Binary files – example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- This dog_array array contains 5 dogs

```
1  typedef enum { BLACK, WHITE, RED } color_t;
2
3  typedef struct {
4    char name[11];    /* name max 10 chars + terminating */
5    color_t color;    /* colour */
6    int nLegs;        /* number of legs */
7    double height;    /* height */
8  } dog;
9
10 dog dog_array[] = /* array for storing 5 dogs */
11 {
12   { "max",   RED,    4, 1.12 },
13   { "cesar",  BLACK, 3, 1.24 },
14   { "buddy",  WHITE, 4, 0.23 },
15   { "spider", WHITE, 8, 0.45 },
16   { "daisy",   BLACK, 4, 0.456 }
17 };
```

link

Binary files – examples

- Writing the dog_array array into a binary file is this easy!

```c
fp = fopen("dogs.dat", "wb"); /* error handling!!! */
if (fwrite(dog_array, sizeof(dog), 5, fp) != 5)
{
  /* error message */
}
fclose(fp); /* here also!!! */
```

- Re-reading the dog_array array is not less easier too.

```c
dog dogs[5]; /* allocating memory */
fp = fopen("dogs.dat", "rb");
if (fread(dogs, sizeof(dog), 5, fp) != 5)
{
  /* error message */
}
fclose(fp);
```

Binary files – example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Do resist the temptation!
- If the representation of any members of the dog structure is different on mother's computer, the saved data cannot be re-read.
- Writing (saving) data into binary files without thinking makes our data non-portable
- Of course if we think, saving the data will become more difficult
  1. We have to agree on the representation
     - which bit is the LSB?
     - is it two-complement?
     - how long is mantissa?
     - are the members of the structure aligned to words? And how long is one word?
     - etc.
  2. The data must be converted first, and then written (saved)

Binary vs text

- Use text files, it is beneficial for everyone!
- Writing the dog_array array into text file

```
1 for (i = 0; i < 5; ++i) {
2   dog d = dog_array[i];
3   fprintf(fp, "%s,%u,%d,%f\n",
4     d.name, d.color, d.nLegs, d.height);
5 }
```

- Reading the dog_array array from text file[4]

```
1 dog dogs[5]; /* allocating memory */
2 for (i = 0; i < 5; ++i) {
3   dog d;
4   fscanf(fp, "%s,%u,%d,%lf",
5     d.name, &d.color, &d.nLegs, &d.height);
6   dogs[i] = d;
7 }
```

[4] we assume that the name of the dog has no whitespace characters in it

Statusflag functions


```
int feof(FILE *fp);
```

- true if we have reached the end of file, false otherwise

```
int ferror(FILE *fp);
```

- true if there was an error during read or write, false otherwise

- Most of the time we don't need them: we can use the return value of read and write functions.

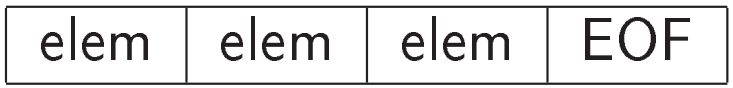
Statusflag functions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Typical mistake

```c
while (!feof(fp))
{
   /* read data element */

   /* process data element */
}
```



- `feof()` is true only if we already have read the end of file symbol.
- What have we learned about data series with termination?

```c
/* read data element */
while (!feof(fp))
{
  /* process data element */
  /* read data element */
}
```

Example: read all lines from a file to a dynamic array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Demonstrate teh string-arrays and file handling

1. Basic solution, but solution
2. open the file
3. count the lines and max char number in a line
4. allocate for string array (dynamic!)
5. read lines and put to the string array
6. close the file
7. print the lines and earese the memory

Read lines example

- Open file for read

```c
#include <stdlib.h>

int main()
{
    FILE* fp;
    fp = fopen("lines.txt", "r");
```

link

Read lines example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1   int chars = 0, maxchars = 0, lines = -1;
2   char c = 'a';
3   while (!(feof(fp))) {
4       chars = 1;
5       fscanf(fp, "%c", &c);
6       while (!(c == '\n' || c == EOF)) {
7           fscanf(fp, "%c", &c);
8           chars++;
9       }
10      lines++;
11      if (chars > maxchars)
12          maxchars = chars;
13  }
```

link

Read lines example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1     rewind(fp);//no use in HW!
2     char** strings = (char**)malloc(lines
3                               * sizeof(char*));
4     for (int i = 0; i < lines; i++) {
5         strings[i] = (char*)malloc(maxchars
6                                  * sizeof(char));
7         fscanf(fp, "%s[^\n]", strings[i]);
8     }
9
10    for (int i = 0; i < lines; i++) {
11        printf("%s\n", strings[i]);
12    }
```

link

Read lines example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1     for (int i = 0; i < lines; i++) {
2         printf("%s\n", strings[i]);
3     }
4
5     for (int i = 0; i < lines; i++) {
6         free(strings[i]);
7     }
8     free(strings);
```

link

Thank you for your attention.