Introduction to Programming in C

(IS-FEE-10061S)

Białystok University of Technology
Faculty of Electrical Engineering

Academic year 2023/2024

Workshop no. 13 (28.05.2024)

Jarosław Forenc, PhD

Topics

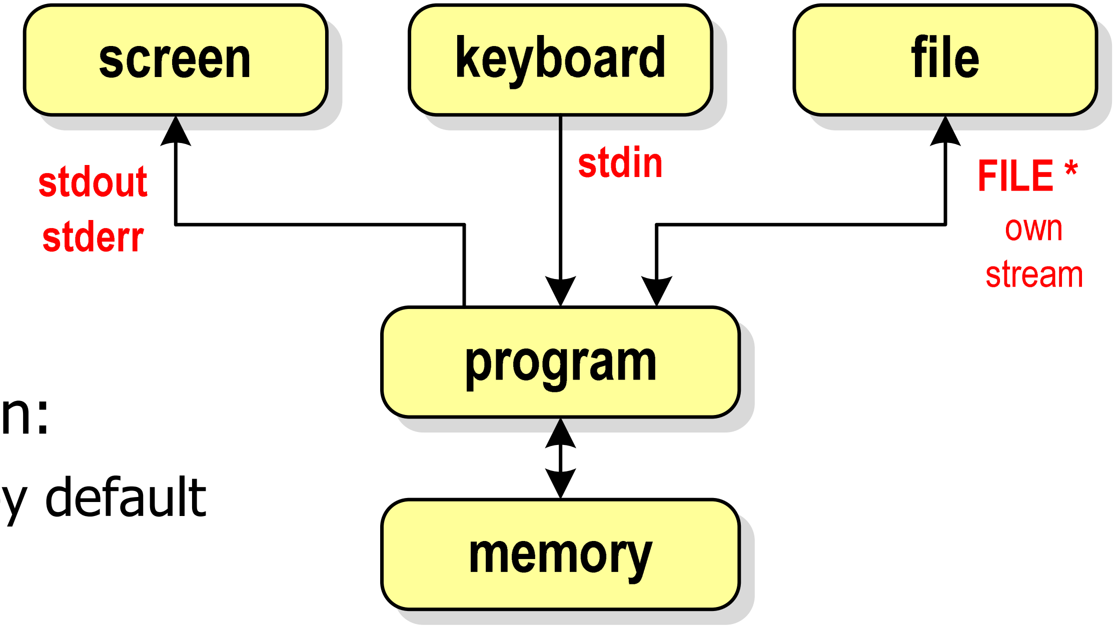
- Input/Output operations in C language
  - streams
  - types of standard I/O functions
- File operations
  - opening and closing
  - character operations
  - string operations
  - formatted operations

Input/Output operations in C language

- I/O operations are not part of the C language - they were implemented as external functions, located in the libraries supplied with the compiler
- Standard I/O operations are based on streams
- The stream is an abstract concept - its name comes from the analogy between the flow of data and, for example, water
- Streams are represented by variables which are pointers to FILE type structures (definition in stdio.h file)
- Each program automatically creates and opens 3 standard streams:
  - stdin - standard input, associated with the keyboard
  - stdout - standard output, associated with the monitor screen
  - stderr - standard output for error messages, associated with the monitor screen

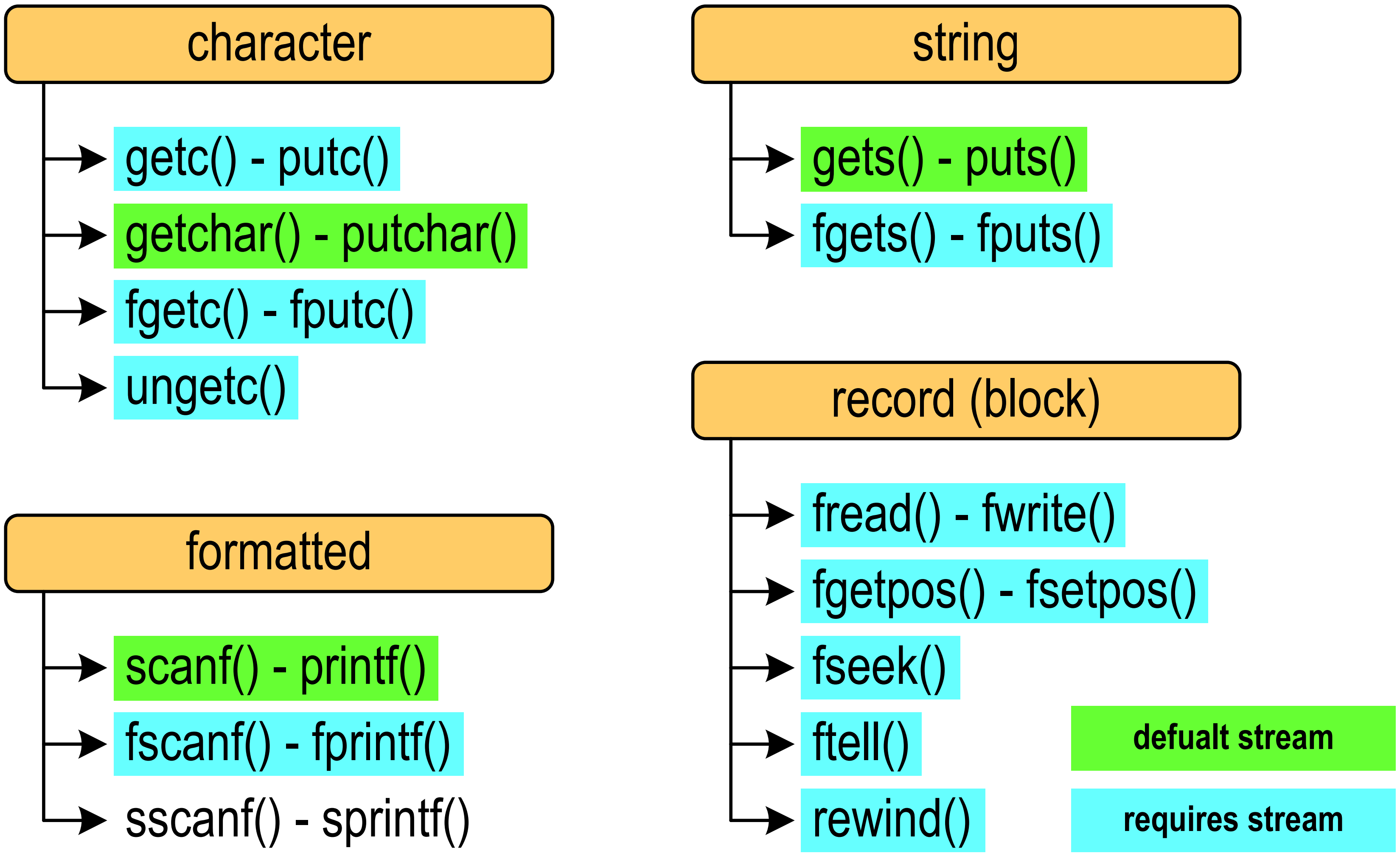
Streams

- Transfer of data in a computer program



- Standard I/O functions can:
  - use a standard stream by default (stdin, stdout, stderr)
  - require a stream (our own, stdin, stdout, stderr)
- The scanf() function implicitly uses stdin
- The printf() function implicitly uses stdout

Types of standard I/O functions



character
- 🟦 getc() - putc()
- 🟩 getchar() - putchar()
- 🟦 fgetc() - fputc()
- 🟦 ungetc()

formatted
- 🟩 scanf() - printf()
- 🟦 fscanf() - fprintf()
- sscanf() - sprintf()

string
- 🟩 gets() - puts()
- 🟦 fgets() - fputs()

record (block)
- 🟦 fread() - fwrite()
- 🟦 fgetpos() - fsetpos()
- 🟦 fseek()
- 🟦 ftell()
- 🟦 rewind()

🟩 **defualt stream**

🟦 **requires stream**

File operations

- A stream is associated with a file by opening it, and this connection is broken by closing the stream
- File processing operations usually consist of three parts

1. Opening a file (stream):

   - functions: fopen()

2. File (stream) operations, e.g. reading, writing:

   - functions for text files: fprintf(), fscanf(), fgetc(), fputc(), fgets(), fputs()…
   - functions for binary files: fread(), fwrite(), …

3. Closing a file (stream):

   - functions: fclose()

File open - fopen()

FOPEN stdio.h

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

- Opens a file fname, the name can include the entire path to the file
- mode specifies the file opening mode:
  - "r" - read
  - "w" - write - if the file does not exist, it will be created; if the file exists, its previous content will be deleted
  - "a" - write (append) - add data to the end of the existing file; if the file does not exist, it will be created

File open - fopen()

FOPEN stdio.h

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

- Returns a pointer to the FILE structure associated with the open file
- If the file fails to open, it returns NULL
- Always check that the file was successfully opened
- When a file is opened, we refer to it by the file pointer
- By default, the file is opened in text mode; adding the letter "b" in the open mode specifies binary mode

File open - fopen()

- Open the file in text mode, read only

```
FILE *fp;

fp = fopen("data.txt","r");
```

- Open file in binary mode, write only

```
fp = fopen("c:\\base\\data.bin","wb");
```

- Open the file in text mode, write only

```
fp = fopen("result.txt","wt");
```

File close - fclose()

FCLOSE stdio.h

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

- Closes the file pointed to by fp
- Returns 0 (zero) if the file was closed successfully
- Returns EOF on error

```
#define EOF     (-1)
```

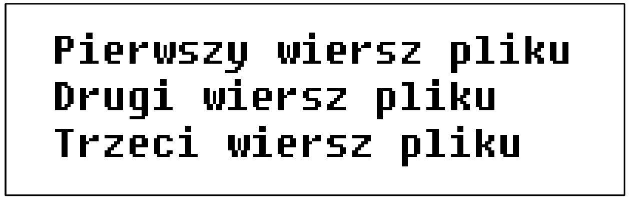
- After a file is closed, the fp pointer can be used to open another file
- Multiple files can be open at the same time in the program

File open and close - example

```
#include <stdio.h>

int main(void)
{
    FILE *fp;

    fp = fopen("file.txt","w");
    if (fp == NULL)
    {
        printf("File open error.\n");
        return (-1);
    }

    /* file operations */

    fclose(fp);

    return 0;
}
```

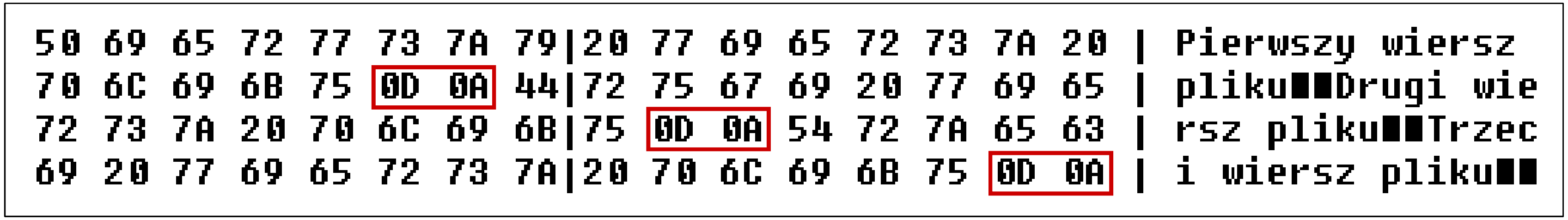
Format (file) text and binary

- The elements of a text file are lines of various lengths
- On DOS/Windows systems, each line of a text file ends with a pair of characters:
    - CR (carriage return) - ASCII code: 13 (decimal) = 0D (hex) = '\r'
    - LF (line feed) - ASCII code: 10 (decimal) = 0A (hex) = '\n'
- Suppose the text file has the following form:



```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

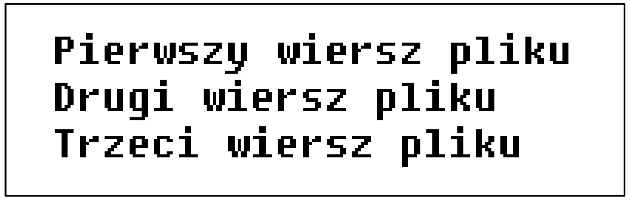
- The actual content of the file is as follows:



```
50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz 
70 6C 69 6B 75 0D 0A 44|72 75 67 69 20 77 69 65 | pliku■■Drugi wie
72 73 7A 20 70 6C 69 6B|75 0D 0A 54 72 7A 65 63 | rsz pliku■■Trzec
69 20 77 69 65 72 73 7A|20 70 6C 69 6B 75 0D 0A | i wiersz pliku■■
```

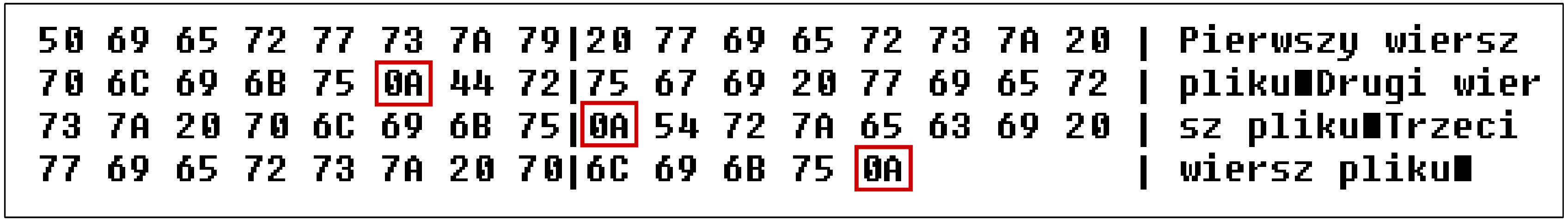
Format (file) text and binary

- In Linux, each line of a text file ends with only one character::
    - LF (line feed) - ASCII code: 10 (decimal) = 0A (hex) = '\n'
- Suppose the text file has the following form:



```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

- The actual content of the file is as follows:



```
50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
70 6C 69 6B 75 0A 44 72|75 67 69 20 77 69 65 72 | pliku■Drugi wier
73 7A 20 70 6C 69 6B 75|0A 54 72 7A 65 63 69 20 | sz pliku■Trzeci
77 69 65 72 73 7A 20 70|6C 69 6B 75 0A          | wiersz pliku■
```

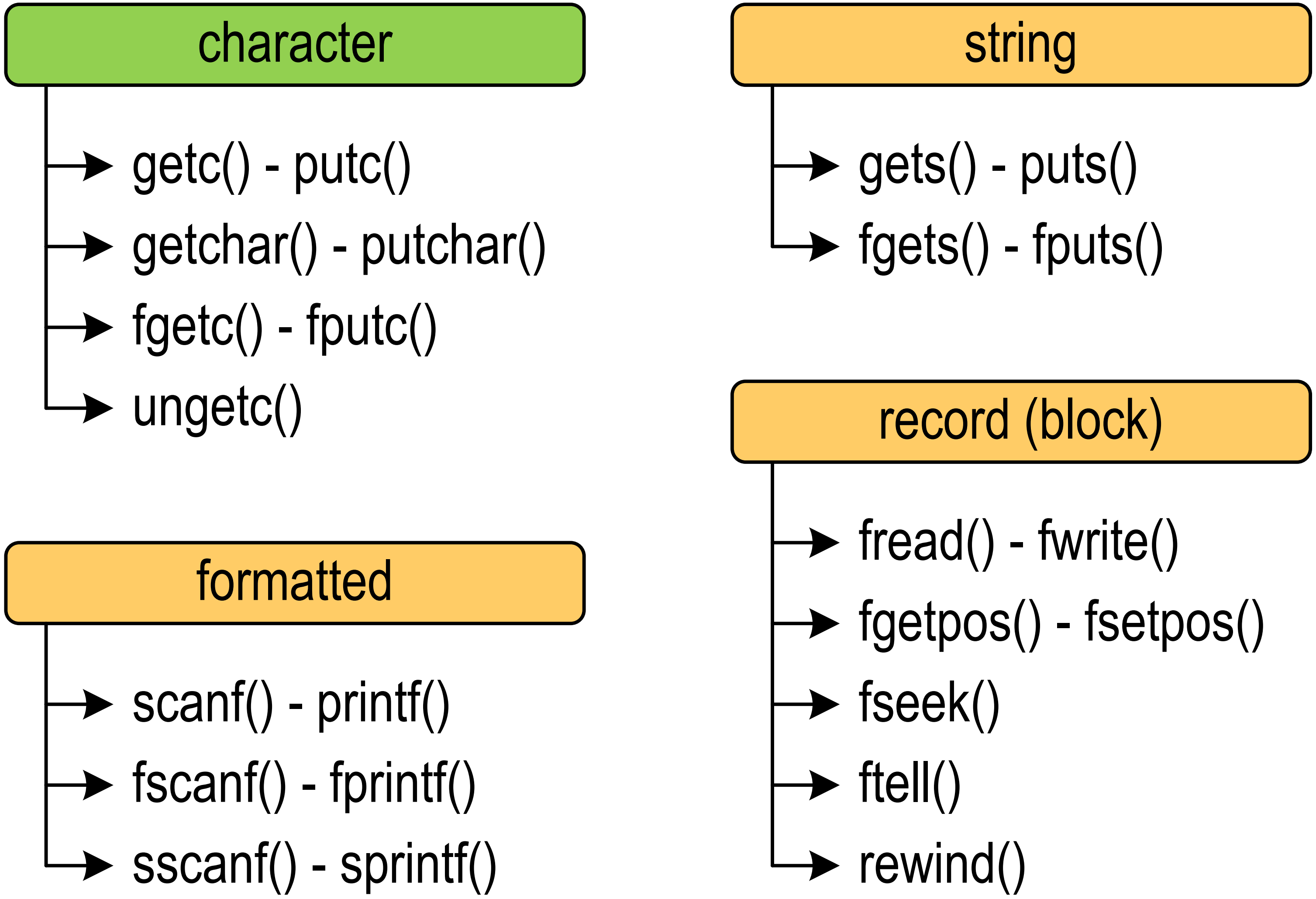
- Binary files do not have a strictly defined structure

File opening modes: text and binary

```
FILE *fp1, *fp2;
fp1 = fopen("data.txt","r");   // or "rt"
fp2 = fopen("data.dat","rb")
```

- Differences between the text and binary modes of file opening relate to the different treatment of CR and LF characters
- In text mode:
  - when reading a file, the pair of characters CR, LF is translated to a newline character (LF)
  - when saving a file, the newline character (LF) is saved as two characters (CR, LF)
- In binary mode:
  - when reading and writing, the pair of characters CR, LF is always treated as two characters

Character operations



character

- getc() - putc()
- getchar() - putchar()
- fgetc() - fputc()
- ungetc()

formatted

- scanf() - printf()
- fscanf() - fprintf()
- sscanf() - sprintf()

string

- gets() - puts()
- fgets() - fputs()

record (block)

- fread() - fwrite()
- fgetpos() - fsetpos()
- fseek()
- ftell()
- rewind()

Character operations

GETC stdio.h

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

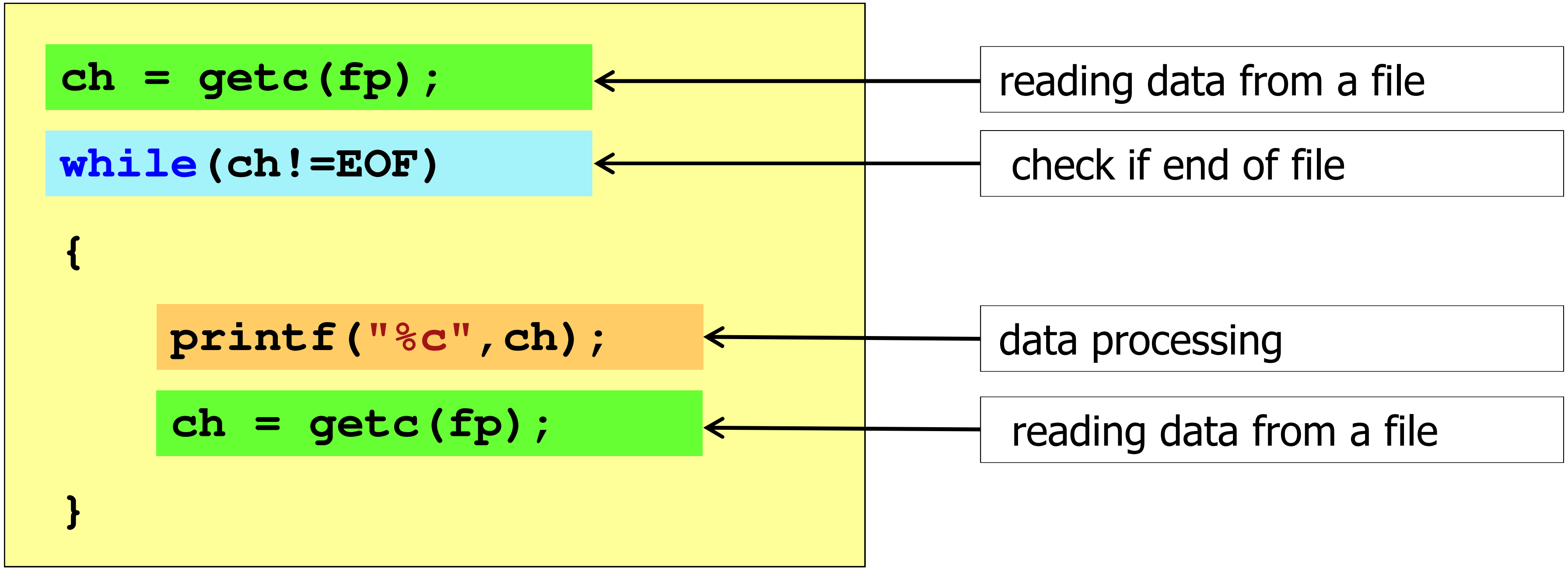
- Reads one character from the current position of the open fp stream and updates the position
- The fp variable should point to a FILE structure representing the stream associated with the open file or one of the standard streams (e.g. stdin)
- If the execution was successful, the function returns the integer value of the code of the read character (type int)
- If an error occurred or the end-of-file marker was read, the function returns EOF
- The fgetc() function works the same way as getc()

Character operations - example

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    int    ch;

    fp = fopen("test.txt","r");

    ch = getc(fp);
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch = getc(fp);
    }

    fclose(fp);
    return 0;
}
```

File processing scheme

- A typical scheme for reading data from a file



```
ch = getc(fp);
while(ch!=EOF)
{
    printf("%c",ch);
    ch = getc(fp);
}
```

reading data from a file

check if end of file

data processing

reading data from a file

Character operations - example

- Reading and displaying the contents of a text file

```
ch = getc(fp);
while(ch!=EOF)
{
    printf("%c",ch);
    ch = getc(fp);
}
```

can be written in a shorter form:

```
while((ch=getc(fp))!=EOF)
    printf("%c",ch);
```

Character operations

PUTC

stdio.h

```
int putc(int ch, FILE *fp);
```

- Writes character (ch) to the open stream represented by the fp argument
- The fp variable should point to a FILE structure representing the stream associated with the open file or one of the standard streams (e.g. stdout)
- If the execution was successful, the function returns the saved character
- If an error occurred, the function returns EOF
- The fputc() function works the same way as putc()

Example: saving the alphabet to a text file

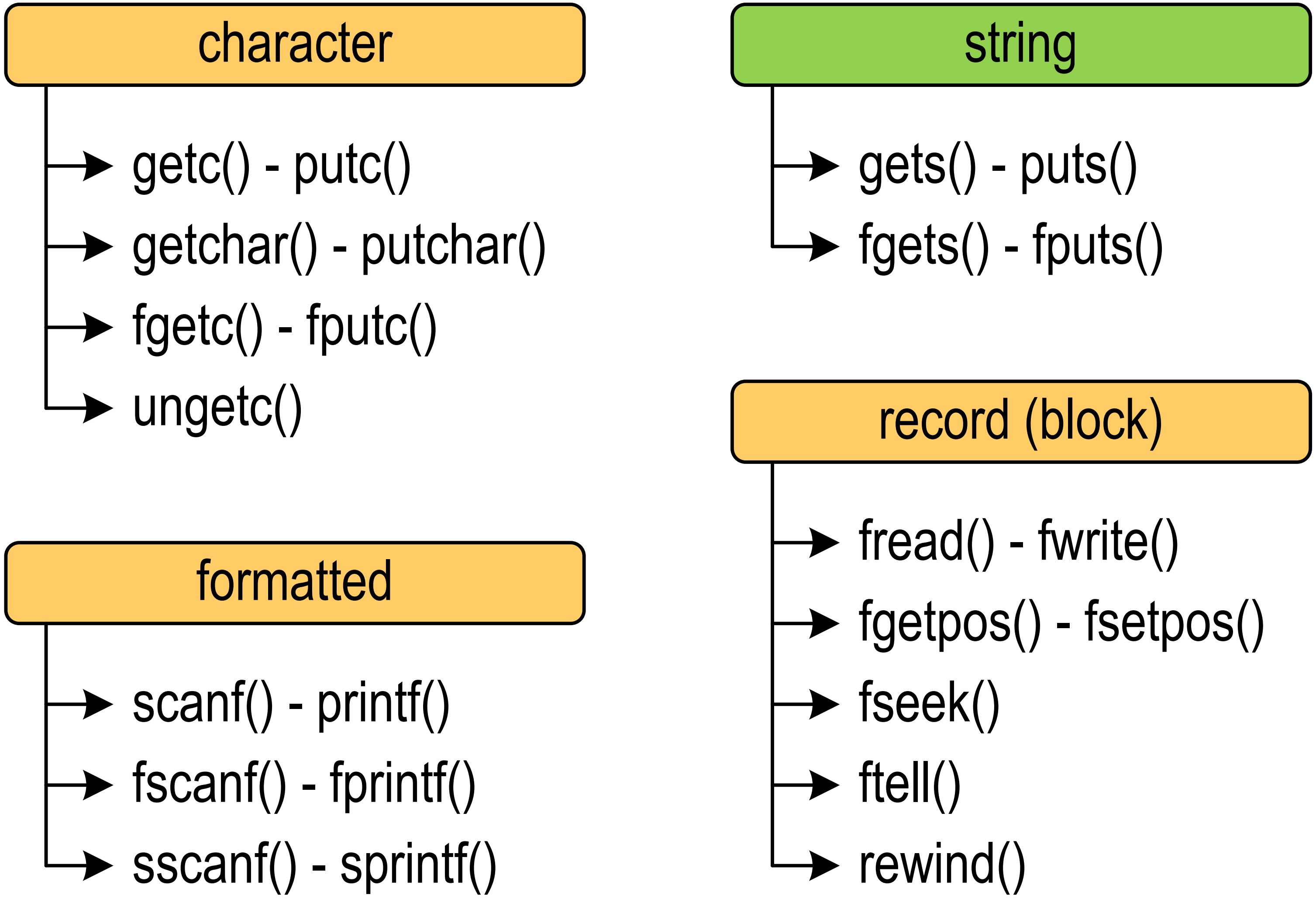
```
#include <stdio.h>

int main(void)
{
    FILE *fp = fopen("alphabet.txt","w");

    for (int i='A'; i<='Z'; i++)
        putc(i,fp);

    fclose(fp);

    return 0;
}
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

- Using the stdout stream, we can display the alphabet on the screen

```
for (int i='A'; i<='Z'; i++)
    putc(i,stdout);
```

String operations



character
- getc() - putc()
- getchar() - putchar()
- fgetc() - fputc()
- ungetc()

formatted
- scanf() - printf()
- fscanf() - fprintf()
- sscanf() - sprintf()

string
- gets() - puts()
- fgets() - fputs()

record (block)
- fread() - fwrite()
- fgetpos() - fsetpos()
- fseek()
- ftell()
- rewind()

String operations

FGETS stdio.h

```
char* fgets(char *buf, int max, FILE *fp);
```

- Reads characters from the open stream represented by fp and writes them to the memory buffer pointed to by buf
- Reading of characters is terminated after encountering the line break '\n' or reading max-1 characters
- After the last character read, it puts '\0' into the buf
- If the execution was successful, the function returns a pointer to the string buf
- If an error occurred or an end-of-file marker was encountered, the function returns NULL

String operations

FPUTS stdio.h

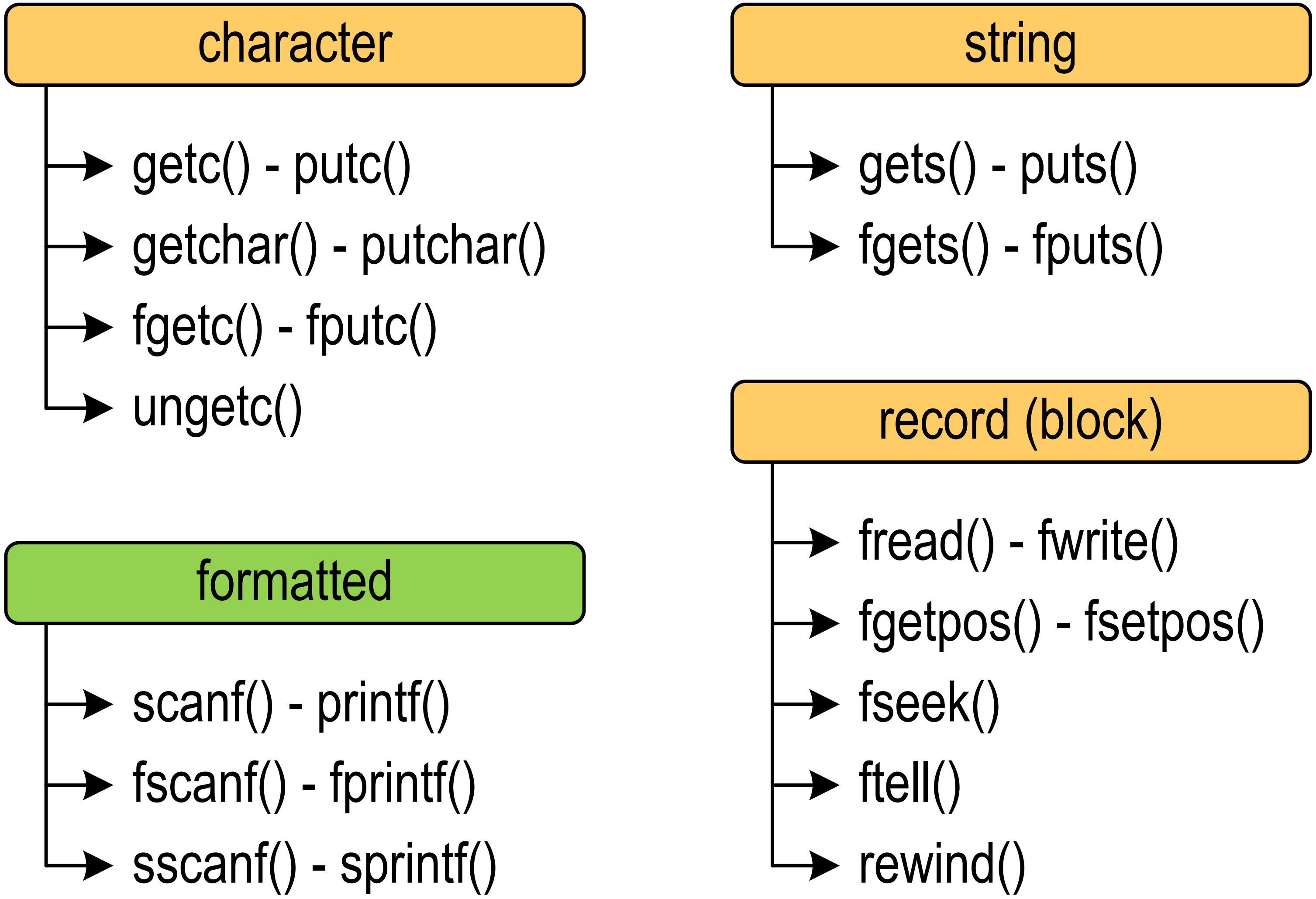
```
int fputs(const char *buf, FILE *fp);
```

- Writes the string buf to the fp stream, excluding the '\n' character
- If the execution was successful, the function returns the last character printed
- If an error occurred, the function returns EOF

String operations - example

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    char  buf[15];

    fp = fopen("test.txt","r");

    while (fgets(buf,15,fp)!=NULL)
        fputs(buf,stdout);

    fclose(fp);

    return 0;
}
```

Formatted operations



character
- getc() - putc()
- getchar() - putchar()
- fgetc() - fputc()
- ungetc()

string
- gets() - puts()
- fgets() - fputs()

formatted
- scanf() - printf()
- fscanf() - fprintf()
- sscanf() - sprintf()

record (block)
- fread() - fwrite()
- fgetpos() - fsetpos()
- fseek()
- ftell()
- rewind()

Formatted operations

SCANF stdio.h

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

- Reads data from stdin stream (keyboard)

FSCANF stdio.h

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

- Reads data from an open stream (file) fp

SSCANF stdio.h

```
int sscanf(char *buf, const char *format,...);
```

- Reads data from the memory buffer pointed to by buf

Formatted operations

PRINTF stdio.h

```
int printf(const char *format,...);
```

- Output data to stdout (screen)

FPRINTF stdio.h

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

- Outputs data to the open stream (file) fp

SPRINTF stdio.h

```
int sprintf(char *buf, const char *format,...);
```

- Outputs data to the memory buffer pointed to by buf

Formatted operations - example

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    FILE  *fp; float x; int i;

    srand((unsigned int)time(NULL));
    fp = fopen("numbers.txt","w");
    for (i=0; i<10; i++)
    {
        x = (float)rand()/RAND_MAX*100;
        fprintf(fp,"%f\n",x);
    }
    fclose(fp);

    return 0;
}
```

```
3.830073
70.848717
99.322487
19.812616
7.132175
49.134800
10.238960
18.668173
8.914456
69.258705
```

Formatted operations - example

```
#include <stdio.h>

int main(void)
{
    FILE  *fp;
    int   age = 21;
    float height = 1.78f;
    char  fname[10] = "John", lname[10] = "Smith";

    fp = fopen("data.txt","w");
    fprintf(fp,"First name: %s\n",fname);
    fprintf(fp,"Last name:  %s\n",lname);
    fprintf(fp,"Age:        %d\n",age);
    fprintf(fp,"Height:     %.2f [m]\n",height);
    fclose(fp);

    return 0;
}
```

```
First name: John
Last name:  Smith
Age:        21
Height:     1.78 [m]
```

I/O error handling

FEOF

stdio.h

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

- Checks whether end-of-file was reached during the last input operation on stream fp
- Returns nonzero if end-of-file was detected during the last input operation, otherwise returns 0 (zero)

Example

- Reading data of various types from a text file

```
Smith John 15-12-2000
Johnson Emily 03-05-1997
Brown Michael 23-05-1995
Williams Jessica 14-01-1990
Jones Matthew 03-11-1995
Taylor Samantha 12-06-1998
Davis Christopher 31-12-1996
Miller Ashley 01-01-1997
```

```
John         Smith        age: 24
Emily        Johnson      age: 27
Michael      Brown        age: 29
Jessica      Williams     age: 34
Matthew      Jones        age: 29
Samantha     Taylor       age: 26
Christopher  Davis        age: 28
Ashley       Miller       age: 27
```

Example

```c
#include <stdio.h>

int main()
{
    FILE *fp;
    char ln[20], fn[20];
    int d, m, y;

    fp = fopen("persons.txt","r");
    fscanf(fp,"%s %s %d-%d-%d",ln,fn,&d,&m,&y);
    while(!feof(fp))
    {
        printf("%-12s %-12s age: %d\n",fn,ln,2024-y);
        fscanf(fp,"%s %s %d-%d-%d",ln,fn,&d,&m,&y);
    }
    fclose(fp);

    return 0;
}
```

Example

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char ln[20], fn[20];
    int d, m, y;

    fp = fopen("persons.txt","r");
    fscanf(fp,"%s %s %d-%d-%d",ln,fn,&d,&m,&y);
    while(!feof(fp))
    {
        printf("%-12s %-12s age: %d\n",fn,ln,2024-y);
        fscanf(fp,"%s %s %d-%d-%d",ln,fn,&d,&m,&y);
    }
    fclose(fp);

    return 0;
}
```

```
John         Smith        age: 24
Emily        Johnson      age: 27
Michael      Brown        age: 29
Jessica      Williams     age: 34
Matthew      Jones        age: 29
Samantha     Taylor       age: 26
Christopher  Davis        age: 28
Ashley       Miller       age: 27
```

End of workshop no. 13

Thank you for your attention!