Lecture 3 – Data Input and Output

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

Standard input and output

In the standard library a simple model of the character by character input and output has been implemented.
The stream of characters consists of a sequence of lines; each line is ended by the new line sign.

Single-character standard input:

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

Function getchar after each call gives the next character from the input or EOF, when the end of the file is encountered. Constant EOF is defined in the header file ¡stdio.h¿.

Single-character output

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

Function putchar returns the value of the output character or EOF (as a message of an error appearamce).

At each source file of the program, which makes use of the library functions realizing the input-output operations before the first call must appear:

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

For programs reading only one input data stream and writing to one output data stream use of functions **getchar, putchar** i **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,...);
```

Returned value is equal to the number of printed successfully variables.

Format includes:

- ordinary characters copied directly to the output stream
- specifications of conversions of subsequent arguments (their beginning is marked by sign `%`

Between the sign `%` and the sign of conversion may appear in the following order:

- minus sign, forcing shifting of the converted argument to the left-hand border of the field
- number stating the minimal size of the field
- point character separating the size of the field from its precision
- number determining precision (for texts - maximal number of characters) number of digits after point for the floating point value or minimal number of digits for the integer value
- one of the letters h if an integer argument should be written as *short*, or l (letter l) - if as *long*

Basic conversions of the printf function		
Sign	Argument type	Output value
d, i	int	decimal number
o	int	octal number without sign (except number 0)
x, X	int	unsigned hexadecimal number 0x or 0X with letters abcdef or ABCDEF for 10,11,12,13,14,15
u	int	unsigned decimal number
c	char	single character
s	char *	sequence of signs written till the meet of the end of the string character or exhaustion of the number of signs determined by the precision
f	double	[-]m.dddddd, where number of digits d is determined by the precision (default value is 6)
e, E	double	[-]m.ddddddde+-xx or [-]m.ddddddE+-xx, where number of digits d is determined by the precision (default 6)
g, G	double	in format %e (%E), when the power is smaller than -4, or otherwise greater than or equal to the precision; otherwise %f; it doesn't print insignificant zeros and closing decimal point
p	void *	pointer; form depends on the implementation
%		the argument is not converted; the sign % is printed

The field width or the precision may be replaced in the specification by the sign *. Then the subsequent argument of the printf replaces the value represented by the *, for exmple:

```
printf("%.*s", max,s);
```

Various conversions of the string "ahoj, przygodo" (14 characters) in the presence of different format specifications:

```
%s              :ahoj, przygodo:
%10s            :ahoj, przygodo:
%.10s           :ahoj, przy:
%-10s           :ahoj, przygodo:
%.20s           :ahoj, przygodo:
%-20s           :ahoj, przygodo      :
%20.10s         :          ahoj, przy:
%-20.10s        :ahoj, przy          :
```

Remark: The printf function has got a variable list of arguments, whose number is determined on the basis of the first of them. Hence:

```
printf(s); /* Dangerous; bad,
           if s contains sign  % */
printf("%s",s);  /* Safe  */
```

Funkcja sprintf:

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

Formattted input - the scanf function

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

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

Argument *format* specifies the input format; *arg1, arg2, etc.* should be pointers (indicatiung the placement of the input data. The *scanf* function closes reading either after interpreting the whole format, or if the datum type is not in accordance with the specification.

It returns the number of correctly read and memorized data, or if it encounters the end of the file the EOF.

In the *format* argument may appear:

- spaces and the tabulators - they are ignored
- ordinary black characters (not %), which we expect to meet in the input data stream.
- conversions specifications consisting of the character % , optional character * stopping the assignement, optional number determining the maximal field size, one of the optional characters h, l or L determining the result size and of the conversion sign.

Sign * indicates, that the subsequent input field should be omitted (the assignement shall not take place).

Basic scanf function conversions		
Sign	Input datum	Argument type
d	integer decimal number	int *
i	integer number; possibly it may appear in the octal form (with the leading 0) or hexadecimal (with the leading 0x or 0X)	int *
o	octal integer number (together with the leading 0 or without it)	int *
u	unsigned integer decimal number	unsigned int *
x	hexadecimal integer number (with or without the leading 0x or 0X, albo bez)	int *
c	characters; other input characters (default 1) are placed in the indicated array; the normal rule of omitting the white characters is not observed; to read the nearest black character one should use % 1s	char *
s	text (but not string, i.e. string of characters appearing without the quotation marks); argument should indicate the array of characters of the size sufficient to accept the text together with the added at the end end of string character	char *
e, f, g	floating-point number with an optional sign, optional decimal point and optional exponent	float *
%	exactly the sign % ; none assignement shall take place	

Precising characters:

- h - with respect to the conversion signs n d,i,o,u,x it informs that the argument is a pointer to an object of the *short* type
- l - argument should be a pointer to an object of the *long* type for the integer data; and to an object of double type for the floating-point data.

The gets and puts functions

They facilate the transfer of strings between the computer and the standard input/output devices.

```
int getchar(void)
char *gets(char *s)
int putc(int c, FILE *stream)
int putchar(int c)
int puts(const char *s)
int ungetc(int c, FILE * stream)
```

Example:

```
#include <stdio.h>

void main()    /*  read and write a line of text  */
{
   char line[80];
   gets(line);
   puts(line);
}
```