Standard I/O in C

1. Overview

The learning objective of this lab is:

- To understand the structure of simple, one function programs in C
- To use simple input and output statements (reading/writing from/to the standard I/O files).

2. Brief theory reminder

The standard device is the one used to start a program. There are two files attached to this device: the standard input file (named **stdin**) and the standard output file (named **stdout**). These files are *sequential* files.

The frequently used C/C++ library functions for I/O operations are:

- For input: **getch, getche, gets, scanf, sscanf** ;
- For output: **putch, puts, printf, sprintf**.

Macros **getchar,** for input, and **putchar**, for output, supplement these set.

2.1. getch, getche and putch functions (MS-DOS/Windows specific)

The functions presented in this section are non-portable (platform-specific). They must be avoided in portable programs.

getch reads without echoing a character when a character is input from the keyboard. The key may have an ASCII code or a special function attached. In the first case, the function returns the ASCII code for that character. In the second case the function must be invoked twice: at the first call, it returns the value zero, and at the second call it returns a specific value for the pressed key.

getche has similar functionality. The difference is that the read character is echoed.

When **getch** or **getche** are invoked, the system waits for a key to be pressed.

putch sends to the output device (e.g. screen) a character which corresponds to the ASCII code used as its argument. Printable characters have ASCII codes in the range [32,126]. Codes outside this range cause various image patterns to be displayed. This function returns its argument.

The prototypes for these functions are in the header file **conio.h**[1]. They are as follows:

[1] **conio.h** is a header file used in old MS-DOS compilers, to create text mode user interfaces, however, it is not part of the C programming language, the C standard library, ISO C or by POSIX.
This header declares several useful library functions for performing "console input and output" from a program. Most C compilers that target DOS, Windows 3.x, Phar Lap, DOSX, OS/2, or Win32 have this header and supply the concomitant library functions in the default C library. Most C compilers that target UNIX and Linux do not have this header and do not supply the concomitant library functions.
The library functions declared by `conio.h` vary significantly from compiler to compiler. As originally implemented in Microsoft's Visual C++ the various functions mapped directly to the first few DOS int 21h functions. But the library supplied with Turbo C++ and Borland C++ did not use the DOS API but instead accessed video RAM directly for output and used BIOS interrupt calls.
Compilers that targeted non-DOS operating systems, such as Linux, Win32 and OS/2, provided different implementations of these functions.

```
int getch(void);
int getche(void);
int putch(int ch);
```

A usage example is:

```
 /* Program  L1Ex1.c */

#include <conio.h>
int main()
{
                putch(getch());
                getch();
                return 0;
}
```

2.2. gets and puts functions

gets reads with echo from the standard input device an ASCII string, and places this string in the variable given as its argument. Upon return reading continues with:

- Character '\n' (newline), which is replaced by character '\0' (null). In this case, **gets** return the start address for the memory area where read characters are stored.
- End of file marker (CTRL/Z). In this case, **gets** returns the value zero.

puts displays on the standard output device a character string corresponding to the ASCII codes of the characters stored at the address given as its argument. The null character ('\0') is interpreted as newline ('\n'). This function returns the code of the last displayed character or the value –1 in case that an error occurred.
Prototypes for these functions can be founding the header file **stdio.h**. They are:

```
        char *gets (char *s);
        int  puts (const char *s);
```

Here ***s** denotes a pointer to an ASCII string (e.g. a string variable).

A usage example is:

```
/*   Program L1Ex2.c  */

#include <stdio.h>
int main()
{
        char s[200];
        printf("\nPlease input a character string and press  ENTER\n");
        gets(s);
        printf("\nThe character string is\n");
        puts(s);
        return 0;
}
```

2.3. scanf and printf

scanf is intended for inputting data which are interpreted as specified by formatting information. Input data are converted from their external representations to the corresponding internal representations and are stored in the variables provided as arguments. End of input is signaled when the ENTER key is pressed.

The prototype for **scanf** is located in the header file named stdio.h**,** and is as follows**:**

int scanf(const char *format {, variable list});

scanf returns the number of input fields successfully converted or the value –1 (EOF) if the end of file (EOF for short) is encountered during conversion. EOF can be signaled by pressing CTRL/Z in Windows/MS-DOS operating systems.

The format for **scanf** is specified as a character string enclosed in double quotation marks ("). The format string includes format specifiers, which define the conversion rules from output to input representations. A format specifier is composed of:

- The character '**%**'.
- The optional character '*****', which indicates that the result of the input data conversion will not be assigned to any variable.
- An optional decimal number, which defines the maximum length of the field controlled by the format specifier.
- One or two letters, which define the type of conversion.

The field controlled by the format begins with the first non-whitespace character (whitespace means blanks or tabs), and is terminated, as the case stands:

- At the character following a whitespace.
- At the character which does not satisfy the type of conversion.
- At the character where the maximum length of the field is reached.

When using the standard input, the data is read after the user presses the ENTER key. The address of a variable is specified as **&variable_name** for **scalars** and as **array_variable_name (note that there is *no & in front)*** for single dimensional arrays (e.g. for character strings).

The letters defining the type of conversion are given in the table below.

Letter	Type of read data
c	Character (char)
s	Character string (char *)
d	Decimal integer
o	Octal integer
x, X	Hexadecimal integer
u	Unsigned integer
f	Floating point (real) number
ld, lo, lx, lX	Long
lu	Unsigned long
lf/Lf	Double/long double

printf is used for printing formatted data, e.g. on the screen. Data are converted from the internal representation into an external representation using the specified format descriptors.

The prototype for **printf** is located in the header file **stdio.h,** and is as follows:

int printf(const char *format {,expression list});

The format is, as with **scanf**, given as a character string. It contains character sequences which are to be printed and format descriptors.
A format descriptor (aka format specifier) contains:

- The character '**%**'.
- An optional minus ('-') character, which specifies that the data to be printed will be left-justified (the default justification is right-justification).
- An optional decimal number, which defines the minimum length (character count) of the field which will hold the printed data.
- And optional dot ('.') followed by a decimal number specifying the precision.
- One or two letters, which define the type of conversion. In addition to the letters used with **scanf**, letters '**e**' or '**E**' can also be used for printing floating point data (in single or double precision) in scientific notation, letters '**g**' or '**G**' specifying that the floating point data is to be printed in either the usual way or in scientific notation such as it yields a field of a minimum length.

printf returns the number of successfully printed characters on success, and -1 in case an error occurs.

Usage examples:

```
/*  Program L1Ex3.c  */

#include <stdio.h>
int main( )
{
        int a;
        float b, c;
        printf("\nPlease input the integer value of a=");
        scanf("%5d",&a);
        printf("\nPlease input the value of the real number b=");
        scanf("%5f",&b);
        c=a+b;
        printf("\nthe value of the sum c=a+b is: %6.3f\n",c);
        return 0;
}
```

2.4. sscanf and sprintf

As a difference from the functions **scanf** and **printf**, **sscanf** and **sprintf** have a supplemental first argument indicating the memory area holding ASCII strings. **sscanf** uses that area to read characters instead the standard input. For **sprintf** the memory area indicated by its first argument is used for output, instead of the standard output device (e.g instead of the screen).
Prototypes for these functions are supplied in **stdio.h**:

int sscanf (const char *buffer, const char *format {,variable list});
int sprintf (char *buffer, const char *format {,expression list});

Usage sample:

```
/* Program L1Ex4.c */
```

```
#include <stdio.h>
int main()
{
        char s[100], q[100];
        int a, b;
        float c, d;
        printf ("\nInput on the same row values for integer a and real c\n\
                separated by a whitespecs character\n\
                and followed by ENTER\n");
        // gets(s); DANGEROUS. NO CONTROL ON THE NUMBER OF CHARS
        // instead, use, e.g.
        fgets(s, sizeof(s), stdin);
        sscanf(s,"%d %f", &a, &c);
        printf("\n a=%4d  c=%8.3f\n", a, c);
        sprintf(q,"%4d  %8.3f\n", a, c);
        sscanf(q,"%d %f",&b,&d);
        printf("\n b=%5d  d=%9.4f\n",b,d);
        return 0;
}
```

2.5. getchar and putchar

The **getchar** macro enables reading with echo of ASCII characters (no special keys allowed). The characters keyed in are placed in a buffer and not converted till the ENTER key is pressed. Upon return, the ASCII code of the first input character is returned. When invoked again the next input character is returned, a.s.o. When EOF is encountered, **getchar** returns the value –1.

The **putchar** macro prints the character given as an argument.

Macros **getchar** and **putchar** are defined in **stdio.h** as follows:

```
int getchar(void);
int putchar (int c);
```

and are invoked in the same way as **getch** and **putch**.

Usage example:

```
/*  Program L1Ex5.c  */

#include <stdio.h>
int main( )
{
        putchar(getchar());
        putchar('\n');
        return 0;
}
```

3. Lab Tasks

3.1. Run the programs given as examples and analyze the output.

3.2. Write a program to check how **getch** is executed when a special key (F1 to F12, CTRL/another key) is pressed. Hint: print the value returned as a decimal/hexadecimal number.

3.3. Write a program to check what **putch** outputs when its argument is a value outside the range character values [32,126].

3.4. Write a program to print the ASCII codes for the keys of your keyboard. Hint: use printf() with a proper descriptor for output.

3.5. Write a program to print the characters corresponding to the ASCII codes in the range [32,126].

3.6. Write a program to containing invocation(s) of **gets(s)**, where **s** is an array. Check the contents of each array member. Hint: use the debugger to set a breakpoint after the invocation of gets() and inspect the memory area containing the result. Why the newline character ('\n') was replaced by '\0'?

3.7. Write a program to read a lowercase letter string and print the corresponding uppercase letter string. Note that there is no function which operates on strings and converts uppercase->lowercase and viceversa.

3.8. Write a program to read a string with only capitals (uppercase letters), and print the corresponding lowercase letter string.

3.9. Write a program to calculate the sum, difference, product and quotient of a pair of real numbers. Output the results in a tabular format, similar to the one below (you do not have to draw lines):

x	y	x+y	x-y	x*y	x/y

3.10. Write a program to print the number **π = 3.14159265** using various (floating point) format descriptors.

3.11. Write a program to print an integer read from the standard input as an octal and a hexadecimal number.