1. Opening Command Window

- Create a shortcut for cmd (command).
- Change the working folder to the folder with the filters and assign it a meaningful name.
- Double click the icon.

To facilitate entering commands do not forget that:

- In order to eliminate typing errors use TAB key to show and select a proper name.
- The arrow keys could be used to find previously used commands.
- A sequence of commands could be saved in a batch file (*.bat).
- Batch files could have parameters.
- .. refers to the parent directory.

Most useful commands:

- Changing the current folder (cd)
- Creating a folder (mkdir);
- Deleting a folder (rmdir);
- Deleting a file (del);
- Showing the content of a file (type);
- Listing the content of a directory (dir);
- Renaming a file (rename);
- Making a copy of a file (copy);
- Sorting a text file in an ascending or descending order (sort);
- Selecting lines that contain a specified string (find);
- More complex string search (findstr) ;
- Making a copy of all files in a specified folder and in all its subfolders (xcopy).

A file or files effected by a command are identified by:

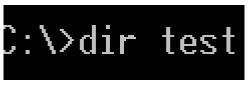
1. Namepath (relative or absolute)
2. Name with wildcards (* for string and ? for one character)

The system "knows" the current folder.
System automatically puts the current location in front of a relative path name.
Absolute path names start with the root directory of the drive.
Description of syntax and semantics with examples:

http://www.computerhope.com

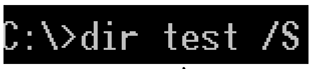
Example:

C:\test: [info.txt]
C:\test\source : [1.c; 2.c; 1.h]
C:\test\rescue: [empty]
C:\test\desc: [Lab1.txt; Lab1.pdf]



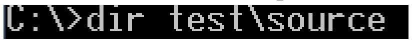
```
C:\>dir test
```

```
2012-10-08  11:03    <DIR>          .
2012-10-08  11:03    <DIR>          ..
2012-10-08  10:59    <DIR>          desc
2012-10-08  11:03                44 info.txt
2012-10-08  10:58    <DIR>          rescue
2012-10-08  11:00    <DIR>          source
```



```
C:\>dir test /S
```

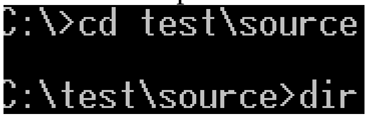
```
Katalog: C:\test
2012-10-08  11:03    <DIR>          .
2012-10-08  11:03    <DIR>          ..
2012-10-08  10:59    <DIR>          desc
2012-10-08  11:03                44 info.txt
2012-10-08  10:58    <DIR>          rescue
2012-10-08  11:00    <DIR>          source
               1 plik(˘w)                44 bajt˘w
 Katalog: C:\test\desc
2012-10-08  10:59    <DIR>          .
2012-10-08  10:59    <DIR>          ..
2012-10-02  23:14            51˙823 lab1.pdf
2012-10-01  09:45                48 Lab1.txt
               2 plik(˘w)            51˙871 bajt˘w
 Katalog: C:\test\rescue
2012-10-08  10:58    <DIR>          .
2012-10-08  10:58    <DIR>          ..
               0 plik(˘w)                 0 bajt˘w
 Katalog: C:\test\source
2012-10-08  11:00    <DIR>          .
2012-10-08  11:00    <DIR>          ..
2012-10-01  09:45                48 1.c
2012-10-01  09:45                48 1.h
2012-10-01  09:45                48 2.c
               3 plik(˘w)               144 bajt˘w
```



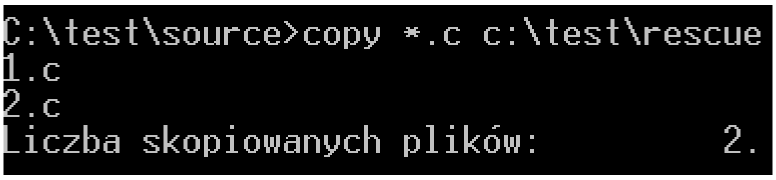
```
C:\>dir test\source
```

```
2012-10-08  11:00    <DIR>          .
2012-10-08  11:00    <DIR>          ..
2012-10-01  09:45                48 1.c
2012-10-01  09:45                48 1.h
2012-10-01  09:45                48 2.c
```

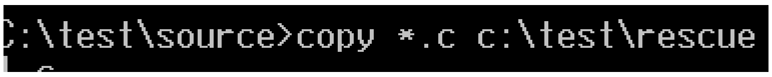
The same output for:



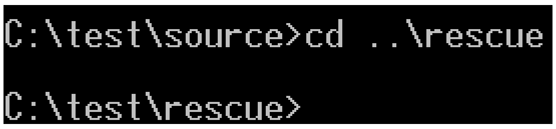
```
C:\>cd test\source

C:\test\source>dir
```



```
C:\test\source>copy *.c c:\test\rescue
1.c
2.c
Liczba skopiowanych plików:         2.
```



```
:\test\source>copy *.c c:\test\rescue
```



```
C:\test\source>cd ..\rescue

C:\test\rescue>
```

2. C rudiments

2.1. Variables

Each variable is a name for a piece of memory that is capable of storing values of a certain type. Once a variable is created its location and type do not change.
The basic types are:

- int
- float
- char

There is no special type do boolean values. The integer value of '0' is regarded as false and ANY other value as true.
Pointers are variables that DO NOT store values but point (refer to) variables. A pointer can refer to different variables (one at a time) but all of them must have the same type.
A string is not a distinct type. It is a pointer to a char variable.
All string processing functions ASSUME that 0 end the string.
The functions are described by the string.h header file.
The developer is solely responsible for assuring that the memory for hosting data is available.

2.2. Functions

Functions are like verbs they describe actions upon variables.

- The declaration of each function you use should be known to the compiler via including a proper header file.
- The declaration contains:
  - function name
  - functions type
  - the types (not names) of all its parameters.
- You must know the definition on operations of each function you use.

3. Writing your own filters

It is only possible using the standard Input / Output library functions.
If we want to write our own filters we must use proper functions for the input and output of data. There are two types in input/output:

- the buffered input/output – the systems collects first a whole line thus enabling the use to make correction.
- the un-buffered input/output – the keystrokes are directly available to the program.

We are obliged to uses the Buffered Input Output.

- After pressing a key the character is not immediately available to a program.
- In the buffered mode of operation the system waits for the return key to be pressed.
- The data is read by the system line by line.

- The data could be read by a program line by line or character by character or both.

This enable us to make modifications in the text that we enter.
This is not the preferred mode of operation for a word processor.

Before starting to write a filter it is NECESSARY to identify the way it should operate.
Is it :

- Line
- Word
- Character

oriented. The type of processing determines the set of functions to use.

In the case in input functions it is absolutely necessary to check the end of data conditions.

With redirection the end-of-data marker is sent automatically by the operating system.
While entering data from the keyboard we have to use ^Z.

The basic functions:

3.1. Line oriented processing

3.1.1. char *gets(char *s);

Reads a line from stdin.

gets collects a string of characters terminated by a new line from the standard input stream stdin and puts it into s. The new line is replaced by a null character (\0) in s.

gets allows input strings to contain certain whitespace characters (spaces, tabs). gets returns when it encounters a new line; everything up to the new line is copied into s.

The gets function is not length-terminated. If the input string is sufficiently large, data can be overwritten and corrupted. The fgets function provides better control of input strings.

Return Value:

- On success, gets returns the string argument s.
- On end-of-file or error, it returns NULL (0)

The programmer is responsible for providing enough space for the input data.

```
//                  NOT PROPER!!!!!
char *line;
while (gets(line)!=NULL)
{
        puts(line);
}
```

The variable line has proper type (no compilation error) but its value is random ant such a code most likely leads to execution error.

```
char line[100];
while (gets(line)!=NULL)
{
        puts(line);
}
```

The name of an array is the pointer to its very first element.

3.1.2. fgets

char *fgets(char *s, int n, FILE *stream);
Gets a string from a stream.

1. fgets reads characters from stream into the string s. The function stops reading when it reads either n - 1 characters or a newline character whichever comes first.
2. fgets retains the newline character at the end of s. A null byte is appended to s to mark the end of the string.

It is far more safe then the gets function. It cuts input data into pieces if necessary.

One can also read data from standard input:

```
char oneLine[20];
if (fgets(oneLine, sizeof(oneLine), stdin)!=NULL)
{
        printf("%s", oneLine);
}

// New line is not eliminated
// sizeof is an operator
```

Remember:
It is not necessary to open or close stdin or stdout data streams.

Return Value:

- On success fgets returns the string pointed to by s.
- It returns NULL on end-of-file or error.

You have to check the return value, otherwise you do not know whether the reading of data has succeeded or not. It may easily result in an endless loop or processing non-relevant data.

3.1.3. puts

int puts(const char *s);
Outputs a string to stdout.

puts copies the null-terminated string s to the standard output stream stdout and appends a newline character.

Return Value:
On successful completion, puts returns a nonnegative value.

Otherwise, it returns a value of EOF.

3.2. Word oriented processing

3.2.1. scanf

scanf is short for scan formatted and is a control parameter used to read and store a string of characters. The string is usually read from an input source, but may come from other sources too.

scanf (“format string”, argument list);

The format string must be a text enclosed in double quotes. It contains the information for interpreting the entire data for connecting it into internal representation in memory.

Example:

- integer (%d) ,
- float (%f) ,
- character (%c)
- string (%s).

The argument list contains a list of variables each preceded by the address list and separated by comma. The number of argument is not fixed; however corresponding to each argument there should be a format specifier. Inside the format string the number of argument should tally with the number of format specifier.

The function scanf returns:

-1 (End of data)

or number of accepted values. It MUST be checked.

Example:

Input line:

age: 21 name Smith

```
int age;
char name[20]
if (scanf (“age: %d name%s”, &age, name)==2)
        puts(“its OK”);
else
        puts(“wrong data format”);
```

REMARK:

The function sscanf works exactly like scanf but reads data from a sting that is its first parameter:

```
char s=” age: 21 name Smith”;
int age;
char name[20];
if (sscanf (s,“age: %d name%s”, &age, name)==2)
        puts(“its OK”);
else
        puts(“wrong data format”);
```

3.2.2. printf

Syntax : printf (“format string”, [argument list]);

To output a word use:

```
char *word=”test”;  // initialization of a pointer
printf(word);
or
printf(“%s”,word);
or
printf(“this is my word: %s\n”, word);
```

Format string may be a collection of escape sequence or/and conversion specification or/and string constant. The format string directs the printf function to display the entire text enclosed within the double quotes without any change.

Escape sequence:

Escape sequence is a pair of characters. The first letter is a slash followed by a character. Escape sequence help us to represent within the format string invisible and non-printed character although there are physically two characters in any escape sequence. It actually represents only one. The various escape sequences are

Escape sequence	Meaning
\n	New line
\t	Tab
\b	Back space
\a	Bell
\o	Null character
\?	To print question mark
\\	To print slash
\'	To print single quote
\"	To print double quote

Example:

```
int res;
….
printf(“the result is:\t%d\n”, res);
```

3.3. Character oriented processing

3.3.1. getchar

int getchar(void);
Gets character from stdin.

getchar is a macro that returns the next character on the named input stream stdin.
It is defined to be getc(stdin) that is the preprocessor makes the replacement.
Note: Do not use this function in Win32 GUI applications.

Return Value:
- On success, getchar returns the character read, after converting it to an int without sign extension.
- On end-of-file or error, it returns EOF. You have to check the return value, otherwise you do not know whether the reading of data has succeeded or not.

3.3.2. putchar

int putchar(int c);
putchar is a macro defined to be putc(c, stdout).

Note: For Win32 GUI applications, stdout must be redirected.

Return Value
- On success, putchar returns the character c.
- On error, putchar returns EOF.

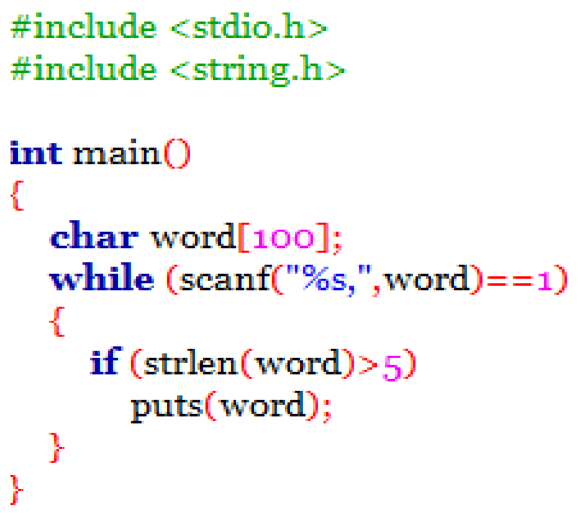
4. Examples

4.1. FILTER #1. (character processing)

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

void main()
{
   long allCharNo=0;
   long allDigitNo;
   int currChar;
   allDigitNo=0;
   while ( (currChar=getchar()) != EOF)
        {
           allCharNo++;
            if (isdigit(currChar))
               allDigitNo++;
        }
   printf("There were: %ld in input %ld characters\n", allDigitNo,allCharNo);
}
```

Remarks:
- stdio.h for the definition of EOF;
- printf is extremely powerful and flexible ;
- new line conversion (UNIX / WINDOWS), in windows the result is different from the file size.

4.2. Filter Long Words



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

int main()
{
   char word[100];
   while (scanf("%s,",word)==1)
   {
      if (strlen(word)>5)
         puts(word);
   }
}
```

4.3. Filter #2 LineCount

```
#include <stdio.h>
void main()
{
     int iTest=0;
     char oneLine[200];
     while (fgets(oneLine, sizeof(oneLine),stdin))
     {
          iTest++;
     }
     printf("%d\n", iTest);
}
```

4.4. Filter #3 (EveryOther)

```
#include <stdio.h>
void main()
{
     int iTest=0;
     char oneLine[200];
     while (fgets(oneLine, sizeof(oneLine),stdin))
     {
          if (iTest==1)
          {
               printf(oneLine);
               iTest=0;
          }
          else
          {
               iTest=1;
```

```
                    }
          }
}
```