

Вход и изход

Пламен Танов
Ненко Табаков

Технологично училище „Електронни системи“
Технически университет – София

версия 0.1

Въведение₁

- Входът и изходът не са част от самия език, те се поддържат от набор от стандартни библиотеки
 - **ANSI C** дефинира прецизно библиотечните функции за вход и изход, така че те да могат да съществуват за различни платформи.
 - Програмите, които използват стандартните библиотеки за вход и изход могат да се ползват на различни платформи без нужда от промяна (разбира се има нужда от компилиране за съответната система)
-
-

Въведение₂

- За всяка една функция трябва добре да се знае **какво прави**, какви **аргументи** получава и какъв **резултат** връща
 - Името на функцията описва приложението ѝ
 - В повечето функции, които връщат **char *** ако по време на изпълнението им възникне грешка връщат константата **NULL**
 - В повечето функции, които връщат **int** ако по време на изпълнението им възникне грешка връщат константата **EOF**
-
-

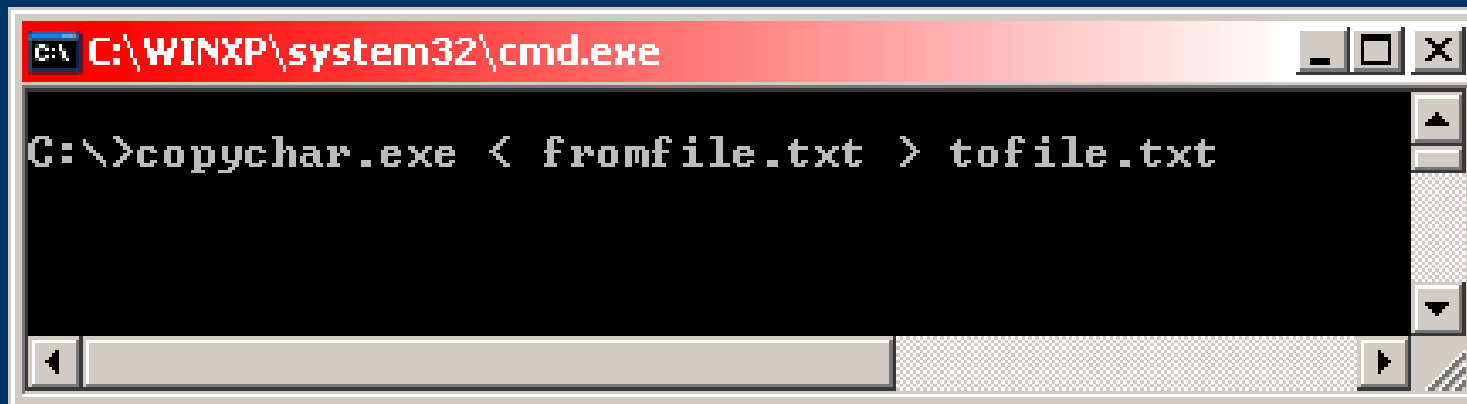
getchar()

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

- Връща следващия символ от стандартния вход (обикновено клавиатурата) или константата **EOF**, ако е достигнат край на файла
 - Възможно е да е дефиниран като макрос
 - Смяна на стандартния вход с текст от файл:
prog.exe < infile
 - Смяна на стандартния вход със стандартния изход на друга програма:
otherprog.exe | prog.exe (каквото изведе **otherprog.exe**, ще бъде прочетено от **prog.exe**)
-
-


Пренасочване на стандартния вход и изход

Това е функция на операционната система.
При Linux, Windows и DOS:



```
C:\WINXP\system32\cmd.exe
C:\>copychar.exe < fromfile.txt > tofile.txt
```

A screenshot of a Windows command prompt window. The title bar reads "C:\WINXP\system32\cmd.exe". The command prompt shows the command "C:\>copychar.exe < fromfile.txt > tofile.txt" entered. The window has standard Windows window controls (minimize, maximize, close) and a scrollbar.



```
C:\WINXP\system32\cmd.exe
C:\>type bigfile.txt |more.exe
```

A screenshot of a Windows command prompt window. The title bar reads "C:\WINXP\system32\cmd.exe". The command prompt shows the command "C:\>type bigfile.txt |more.exe" entered. The window has standard Windows window controls (minimize, maximize, close) and a scrollbar.

putchar()

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

- Извежда символ на стандартния изход (обикновено монитора). Връща изведения символ или константата **EOF**, ако е възникнала грешка
 - Възможно е да е дефиниран като макрос
 - Смяна на стандартния изход с текстов файл:
prog.exe > outfile
 - Смяна на стандартния изход със стандартния вход на друга програма:
prog.exe | otherprog.exe (каквото изведе **prog.exe**, ще бъде прочетено от **otherprog.exe**)
-
-

Пример

```
#include <stdio.h> //тук са описани getchar() и putchar()
#include <ctype.h> //тук е описана функцията tolower()

int main() /* lower: convert input to lower case*/
    int c
    while ((c = getchar()) != EOF) { //докато не стигне EOF
        putchar(tolower(c)); //се извежда c, ако е буква-малка!
    }
    return 0;
}
```

printf()

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

- Конвертира, форматира и извежда аргументите на стандартния изход
- **format** оказва как точно да се форматират аргументите
- Връща броя на изведените символи
- Съвместима е с **putchar()** и **getchar()** и могат да се използват съвместно без проблем

Форматиращ низ₁

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

Състои се от два типа данни:

- обикновени, които директно се извеждат на екрана и
 - спецификатори, всеки един от които служи за извеждане на следващия аргумент от списъка **arg1, arg2, ...**
 - Всеки спецификатор започва с % и завършва с тип на аргумента
-
-

Форматиращ низ₂

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

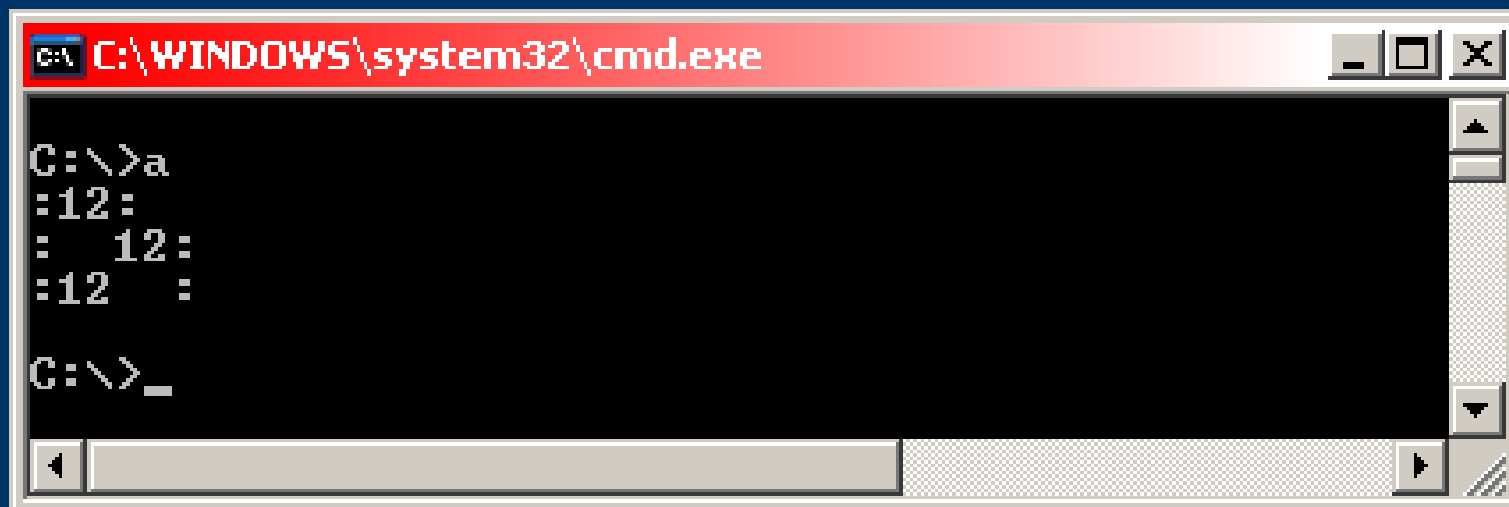
Между % и типа на аргумента може да се добавят следните определители (в този ред):

- Знак „-“ - определя ляво подравняване
 - Минимален брой изведени символи — при необходимост се добавят интервали от ляво или от дясно (в зависимост от това дали има „-“)
 - „.“ - за разделяне на минималния брой и числото за прецизност
-
-

Пример

```
#include <stdio.h>

int main() {
    printf(":%d:\n", 12);    //:12:
    printf(":%4d:\n", 12);  //:  12:
    printf(":%-4d:\n", 12); //:12  :
    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt is at "C:\>a", and the output is:

```
C:\>a
:12:
:  12:
:12  :
```

The prompt is now "C:\>_".

Форматиращ низ₃

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

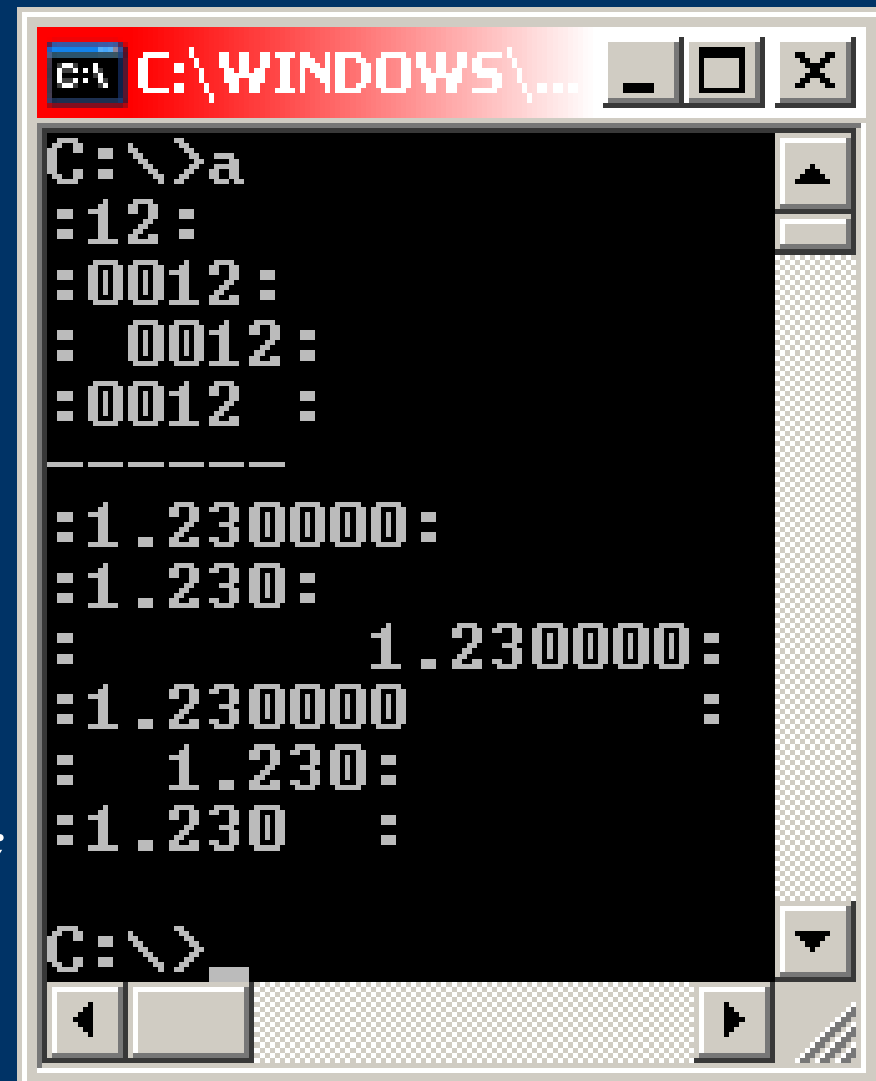
Числото за прецизност има различен смисъл приложено към аргумент от различен тип:

- Низ – максималния брой символи от низа, които ще бъдат изведени
 - Реално число – броят на цифрите след десетичната запетая (6 по подразбиране)
 - Цяло число – минималния брой цифри от числото, при необходимост се дописват нули от ляво на числото
-
-

Пример

```
#include <stdio.h>

int main() {
    printf(":%d:\n", 12);
    printf(":%.4d:\n", 12);
    printf(":%5.4d:\n", 12);
    printf(":%-5.4d:\n", 12);
    printf("-----\n");
    printf(":%f:\n", 1.23);
    printf(":%.3f:\n", 1.23);
    printf(":%15f:\n", 1.23);
    printf(":%-15f:\n", 1.23);
    printf(":%7.3f:\n", 1.23);
    printf(":%-7.3f:\n", 1.23);
    return 0;
}
```



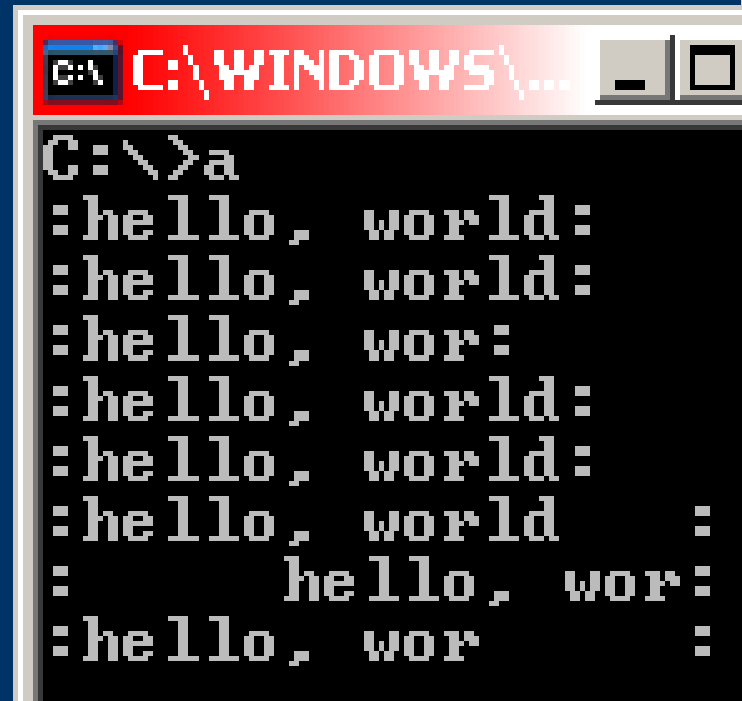
```
C:\WINDOWS\...
C:\>a
:12:
:0012:
: 0012:
:0012 :
-----
:1.230000:
:1.230:
:                1.230000:
:1.230000      :
:  1.230:
:1.230  :
```

Пример

```
#include <stdio.h>

int main() {
    printf(":%s:\n", "hello, world");
    printf(":%10s:\n", "hello, world");
    printf(":%.10s:\n", "hello, world");
    printf(":%-10s:\n", "hello, world");
    printf(":%.15s:\n", "hello, world");
    printf(":%-15s:\n", "hello, world");
    printf(":%15.10s:\n", "hello, world");
    printf(":%-15.10s:\n", "hello, world");

    return 0;
}
```



The screenshot shows a Windows command prompt window with a red title bar. The window title is "C:\WINDOWS\...". The command prompt shows the following output:

```
C:\>a
:hello, world:
:hello, world:
:hello, wor:
:hello, world:
:hello, world:
:hello, world :
:      hello, wor:
:hello, wor      :
```

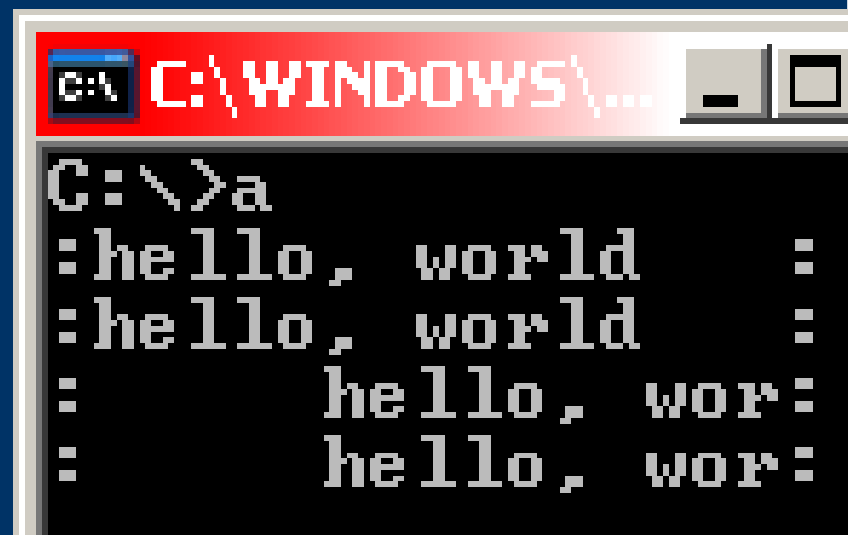
Форматиращ низ₄

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

Ако на мястото на минимален брой изведени символи или на числото за прецизност се сложи „*“ то числото се взема от списъка с аргументи

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

```
int main() {  
printf(":%-15s:\n", "hello, world");  
printf(":%-*s:\n", 15, "hello, world");  
printf(":%15.10s:\n", "hello, world");  
printf(":%*.*s:\n", 15, 10,  
        "hello, world");  
  
return 0;  
}
```



```
C:\WINDOWS\...  
C:\>a  
:hello, world :  
:hello, world :  
:      hello, wor:  
:      hello, wor:
```

Тип на аргумента₁

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

- **d, i** – **int**, десетично число
 - **u** – **int**, десетично число, без знак
 - **o** – **int**, в осмична бройна система, без знак
 - **X, x** – **int**, в 16 бройна система, без знак
(**ABCDEF, abcdef** за числата от 10 до 15)
 - **c** – **int (char)**, СИМВОЛ
 - **s** – **char ***, низ до нулев символ за край
 - **p** – **void ***, указател (зависи от реализацията)
 - **%** - не се взима аргумент от списъка, извежда се %
-
-

Тип на аргумента₂

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

- **f** – **double**, формат: **[-]m.ddddddd**
 - **E**, **e** – **double**, формат: **[-]m.dddddddE+/-xx**
или **[-]m.ddddddde+/-xx** (с експонента)
 - **G**, **g** – **double**, ако експонентата е по-малка от -4 или е по-голяма или равна на прецизността се използва **%E**, **%e** за форматиране, иначе **%f**
-
-

Често срещана грешка

- Ако искаме да изведем низ, в който има символ %

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

```
int main () {
```

```
    char * s = "string with bad char: %d";
```

```
    printf(s); // ГРЕШНО!!!
```

```
    printf("\n");
```

```
    printf("%s", s);
```

```
    return 0;
```

```
}
```

```
C:\>a
```

```
string with bad char: 2280688
```

```
string with bad char: %d
```

sprintf()

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

- Като **printf()**, но не извежда на екрана, а записва в променливата **string**

Списък от аргументи с променлив брой променливи

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

- Неименуваният списък с променлив брой аргументи („...“) може да е единствено на края
 - За обхождането на аргументите се използва макрос, дефиниран във файла **<stdarg.h>**
 - Използват се
 - **va_list ap;** //тип на променливата, с която обхождаме аргументите
 - **va_start(ap, последен аргумент);** //начало
 - **a = va_arg(ap, тип на аргумент);** //взимаме аргумент
 - **va_end(ap);** //край на обхождането
-
-

Пример₁

```
#include <stdarg.h>
/* minprintf: ограничен printf със променлив брой аргументи
*/
void minprintf(char *fmt, ...) {
    va_list ap;
    /* последователно обхождаме неименованите аргументи
    посредством ap */
    char *p, *sval;
    int ival;
    double dval;

    va_start(ap, fmt);
    /* ap сочи към първия неименован аргумент */

    for (p = fmt; *p; p++) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
    }
}
```

Пример₂

```
switch (*++p) {
    case 'd': //текущия аргумент е int
        ival = va_arg(ap, int);
        printf("%d", ival);
        break;
    case 'f': //текущия аргумент е double
        dval = va_arg(ap, double);
        printf("%f", dval);
        break;
    case 's': //текущия аргумент е char *
        for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
        break;
    default:
        putchar(*p);
        break;
}
va_end(ap); /* освобождаваме памет и др. */
}
```

scanf() ₁

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

- Аналог на printf(), за въвеждане на данни от стандартния вход
- **format** ползва сходен формат като при printf()
- Неименуваните аргументи **трябва да са указатели!**
В тях се записват съответните данни
- Връща броя на успешно прочетените аргументи (**не брой символи**) или константата **EOF**, ако е достигнат край на файла

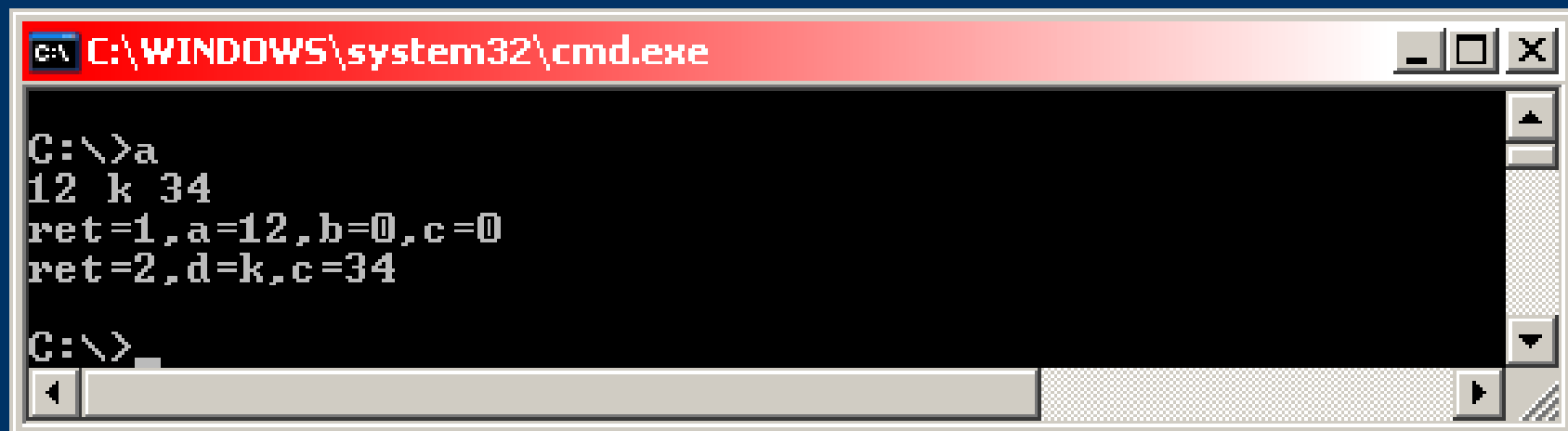
*scanf()*₂

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

- Ако някой аргумент не съвпадне по тип, **scanf()** спира да въвежда данни
- Повторното извикване продължава след последния успешно преобразуван елемент
- Съвместима е с другите функции за въвеждане на данни от стандартния вход

Пример

```
#include <stdio.h>
int main () {
    int a=0, b=0, c=0, ret=0;
    char d = 'a';
    ret = scanf("%d %d %d", &a, &b, &c);
    //прочита само 12, тъй като k не е число и спира до него
    printf("ret=%d,a=%d,b=%d,c=%d\n", ret, a, b, c);
    //продължава да чете от последния правилно прочетена
    //променлива (в случая a) (прочита k 34):
    ret = scanf("%c %d",&d, &c);
    printf("ret=%d,d=%c,c=%d\n", ret, d, c);
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\>a
12 k 34
ret=1,a=12,b=0,c=0
ret=2,d=k,c=34
C:\>
```

Форматиращ низ₁

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

Съдържа:

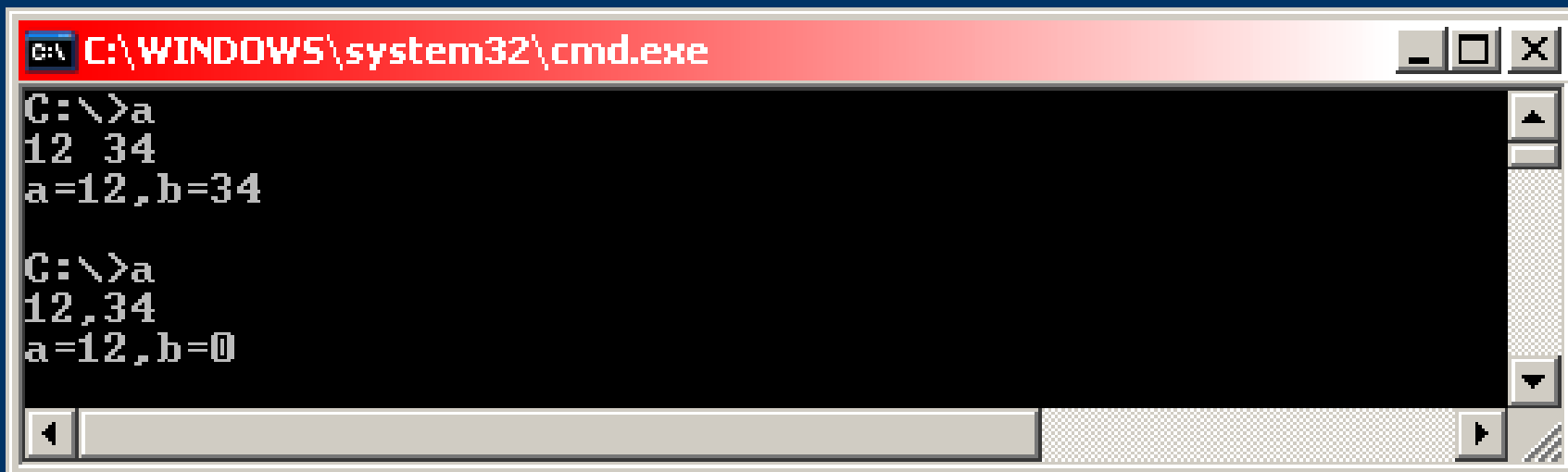
- Празни символи или интервали, които не се игнорират („\t“ „\n“ „\r“ **vertical tab, formfeed**)
- Нормални символи (не %), които се очакват да се срещнат в следващите не празни символи



Пример

```
#include <stdio.h>
int main () {
    int a=0, b=0;
    scanf("%d %d", &a, &b); //разделител м/у числата е SPACE
    printf("a=%d,b=%d\n", a, b);

    return 0;
}
```



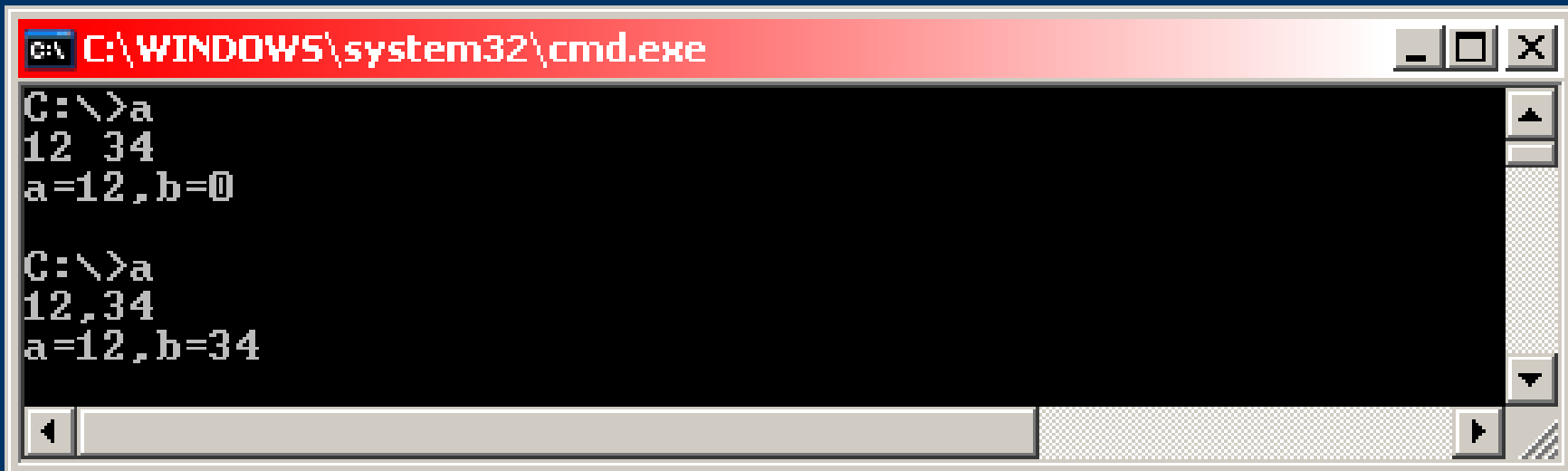
```
C:\WINDOWS\system32\cmd.exe
C:\>a
12 34
a=12,b=34

C:\>a
12,34
a=12,b=0
```

Пример

```
#include <stdio.h>
int main () {
    int a=0, b=0;
    scanf("%d,%d", &a, &b); //разделител м/у числата е ,
    printf("a=%d,b=%d\n", a, b);

    return 0;
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
C:\>a
12 34
a=12,b=0

C:\>a
12,34
a=12,b=34
```

The first execution shows the program reading two integers separated by a space (12 34) and printing them separated by a comma (a=12,b=0). The second execution shows the program reading two integers separated by a comma (12,34) and printing them separated by a comma (a=12,b=34).

Форматиращ низ₂

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

Съдържа:

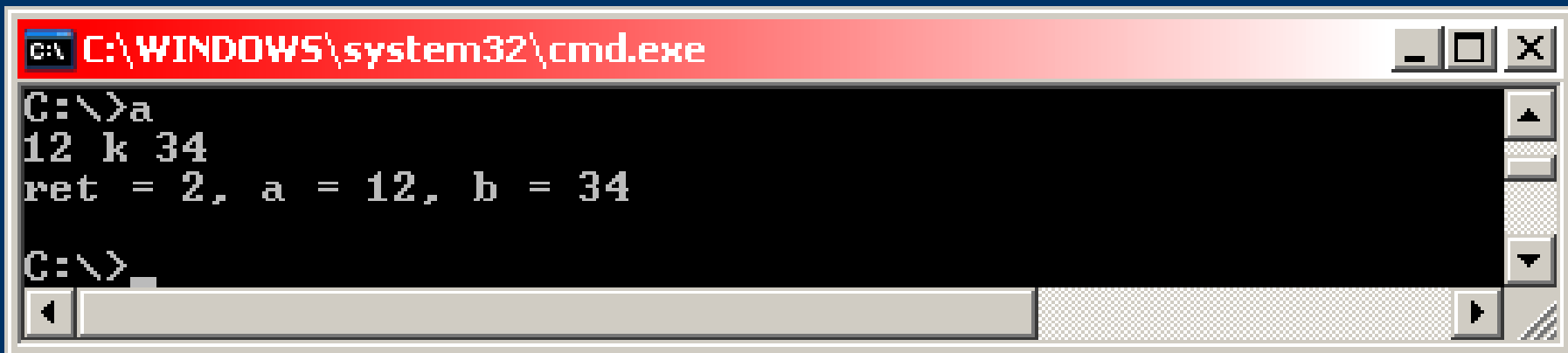
- Спецификатори, започващи с %, след което незадължителни * (оказващ че преобразуваният елемент няма да се запише в променлива от списъка **arg1, arg2, ...** - ще се пропусне), число за максимална дължина и тип на аргумента (указател към какъв тип е съответния **arg1, arg2, ...**)

Пример

```
#include <stdio.h>
int main () {
    int a, b, ret;
    ret = scanf("%d %*c %d", &a, &b);
    //има само два аргумента (a и b, a %*c не се записва никъде) !

    /* ret = 2 - успешно са променени стойностите и на двете
    променливи (a и b) */
    printf("ret = %d, a = %d, b = %d\n", ret, a, b);

    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\>a
12 k 34
ret = 2, a = 12, b = 34
C:\>
```

Тип на аргумента₁

- **d** – **int ***, десетично число
 - **i** – **int ***, число, което може да е в 8 бройна система (ако започва с 0), или 16 (с 0x или 0X)
 - **o** – **int ***, число в 8 бройна система
 - **x** – **int ***, число в 16 бройна система
 - **u** – **unsigned int ***, десетично число без знак
 - **c** – **char ***, символи (по подразбиране на брой 1), дори и празни (освен при %1c)
 - **s** – **char ***, низ, достатъчно дълъг и за '\0'
 - **e, f, g** – **float ***, възможност за знак, десетична точка и експонента
 - **%** - СИМВОЛЪТ %, не се прави преобразуване
-
-

Тип на аргумента₂

- Пред символите **d**, **i**, **o**, **u** и **x** може да се постави **h**, за да се окаже, че указателя е към **short** на съответния тип (т.е. **short int ***)
 - Пред символите **d**, **i**, **o**, **u** и **x** може да се постави **l**, за да се окаже, че указателя е към **long** на съответния тип (т.е. **long int ***)
-
-

Пример

```
#include <stdio.h>
main() { /* суматор */
    double sum, v;
    sum = 0;
    while (scanf("%lf", &v) == 1) {
        printf("\t%.2f\n", sum += v);
    }

    return 0;
}
```

Пример

- Ако искаме да прочетем дата във формат:
22 февруари 2007:

```
int day, year;  
char monthname[20];  
scanf("%d %s %d", &day, monthname, &year);
```

monthname е указател и за това **няма** * пред него!

- Ако искаме да прочетем дата във формат:
22/02/2007:

```
int day, month, year;  
scanf("%d/%d/%d", &day, &month, &year);
```



sscanf()

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

- Като **scanf()**, но не въвежда от клавиатурата, а от променливата **string**
- **arg1, arg2, ...** са **указатели**, в които се записват съответните данни!

Пример

```
//ако не знаем в какъв формат е въведена датата
while (getline(line, sizeof(line)) > 0) {
    if (sscanf(line, "%d %s %d", &day, monthname, &year) == 3)
        printf("valid: %s\n", line); /* 22 февруари 2007 */
    else if (sscanf(line, "%d/%d/%d", &day, &month, &year) == 3)
        printf("valid: %s\n", line); /* dd/mm/yy */
    else
        printf("invalid: %s\n", line); /* грешен формат */
}
```

Често срещана грешка

- Списъка с аргументи трябва да е от указатели!

```
//често се пише:  
scanf ("%d", n) ;//ГРЕШНО!!!  
//вместо:  
scanf ("%d", &n) ;
```

Работа с файлове

- Преди да можем да записваме или четем от файл трябва той да бъде **отворен със съответните права** (за запис или четене)
 - След като приключим работата с файла той **трябва да бъде затворен**
 - За работа с файлове се използва така нареченият файлов указател (**file pointer**): **FILE ***, описан в **<stdio.h>**
-
-

Отваряне на файл₁

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

- Връща файловия указател, с който по-късно можем да манипулираме файла или **NULL** ако възникне грешка
 - Като аргументи получава име на файл и режим на достъп
 - Режимът на достъп може да бъде четене ("**r**"), запис ("**w**") и добавяне ("**a**"). Някои системи различават работата с текстов и двоичен файл и за последния се добавя "**b**" (т.е. режимът е "**rb**", "**wb**", "**ab**")
-
-

Отваряне на файл₂

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

- Ако файлът не съществува и го отваряме за запис или добавяне то той **бива създаден** (ако е възможно)
 - Ако отваряме съществуващ файл за запис старото му съдържание **се изтрива**
 - Ако отваряме съществуващ файл за добавяне старото му съдържание **се запазва**, новото се добавя към края на файла
 - Ако отваряме за четене несъществуващ файл или нямаме права за достъп върху него **възниква грешка** и **fopen()** връща **NULL**
-
-

Пример

```
#include <stdio.h>
#define FILENAME "data.txt"
int main() {
    FILE *fp;
    //отваряме файла
    fp = fopen(FILENAME, "r");
    if (fp == NULL) {
        printf("error occured while opening %s", FILENAME);
        return -1;
    }

    //... четене от файла fp ...

    //затваряме файла:
    fclose(fp);
    return 0;
}
```

Четене и запис на символ

```
int getc(FILE *fp);  
int putc(int c, FILE *fp);
```

- **getc()** връща следващия символ от файла **fp** или **EOF** ако се стигне до края му или възникне грешка
 - **putc()** записва символа **c** във файла **fp** и го връща като резултат. Ако възникне грешка по време на записа функцията връща **EOF**
 - Както **getchar()** и **putchar()**, така и **getc()** и **putc()** могат да са реализирани като макроси
-
-

Стандартни файлове

`stdin`, `stdout`, `stderr`

- При стартирането на програма операционната система отваря три файла:
 - Стандартен вход (**`stdin`**)
 - Стандартен изход (**`stdout`**)
 - Стандартна грешка (**`stderr`**)
 - По подразбиране **`stdin`** е свързан към клавиатурата, **`stdout`** и **`stderr`** към екрана
 - **`stdin`** и **`stdout`** могат да бъдат пренасочени **преди стартирането** на програмата, но **не могат** да бъдат променяни по време на изпълнение (**те са константи**)
 - **`stdin`**, **`stdout`** и **`stderr`** са описани в **`<stdio.h>`**
-
-

Пример

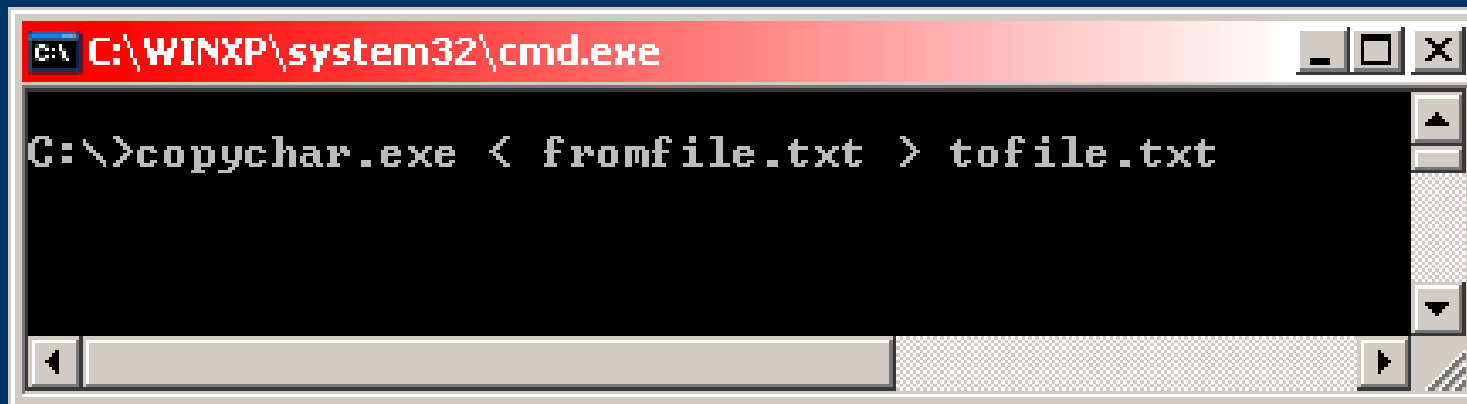
```
//функциите getchar() и putchar() могат да се дефинират  
//като макроси по следния начин:
```

```
#define getchar() getc(stdin)  
#define putchar(c) putc((c), stdout)
```

```
//stdin е стандартния вход, stdout – стандартния изход
```


Пренасочване на стандартния вход и изход

Това е функция на операционната система.
При Linux, Windows и DOS:



```
C:\WINXP\system32\cmd.exe
C:\>copychar.exe < fromfile.txt > tofile.txt
```

A screenshot of a Windows command prompt window. The title bar reads "C:\WINXP\system32\cmd.exe". The command prompt shows the command "C:\>copychar.exe < fromfile.txt > tofile.txt" entered and executed. The window has standard Windows window controls (minimize, maximize, close) and a scrollbar.



```
C:\WINXP\system32\cmd.exe
C:\>type bigfile.txt |more.exe
```

A screenshot of a Windows command prompt window. The title bar reads "C:\WINXP\system32\cmd.exe". The command prompt shows the command "C:\>type bigfile.txt |more.exe" entered and executed. The window has standard Windows window controls (minimize, maximize, close) and a scrollbar.

Пример₁

```
#include <stdio.h>
//копира файлове, подадени като аргументи в командния ред
//върху стандартния изход. Използване: cat.exe file1 file2 ...
int main(int argc, char *argv[]) {
    FILE *fp;
    void filecopy(FILE *, FILE *); //по-долу е дефинирана

    if (argc == 1) //няма аргументи-ще ползваме стандарт. вход
        filecopy(stdin, stdout); //копираме stdin в stdout
    else //имаме аргументи в командния ред - обхождаме ги:
        while(--argc > 0)
            if ((fp = fopen(++argv, "r")) == NULL) {
                printf("cat: can't open %s\n", *argv);
                return 1; //връщаме грешка
            } else { //файлът е отворен успешно
                filecopy(fp, stdout); //копираме го в изхода
                fclose(fp); //затваряме файла
            }
    return 0;
}
```

Пример₂

```
/* filecopy: копира файла ifr в ofr */
void filecopy(FILE *ifr, FILE *ofr) {
    int c;
    while ((c = getc(ifr)) != EOF) {
        //докато има символи за четене - четем от ifr
        putc(c, ofr);           //записва поредния символ в ofr
    }
}
```

Форматиран вход и изход във файл

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

- Същите като `scanf()` и `printf()`, но с един аргумент повече – файловият указател, от който да се извършва четене/запис

Затваряне на файл

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

- Затваря файл и освобождава използваните ресурси
 - Записва буферизираните данни във файла
 - Добра практика е файлът да се **затваря веднага след** като сме приключили работата си с него и да се **отваря непосредствено преди** първото действие с него, а не да е отворен без да се ползва
 - Извиква се автоматично за всеки един отворен файл при успешно завършване на приложението, но е силно препорачително да се **извиква явно!!!**
 - Повечето ОС имат лимит на отворените файлове
-
-

Обработка на грешки – `stderr` и `exit()`

```
void exit(int code);
```

- За извеждане на предупредителни съобщения и грешки се използва `stderr`. Така те могат да бъдат забелзязани по-лесно от потребителя
 - Функцията `exit()` се използва за изход от програмата. С аргумента `code` се предоставя възможност приложението да съобщи дали успешно е приключило своята работа или е възникнала грешка
 - По конвенция число различно от 0 сигнализира за грешка, 0 – за успешно завършване
-
-

Пример

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fp;
    void filecopy(FILE *, FILE *);
    char *prog = argv[0]; /* program name for errors */
    if (argc == 1 ) /* no args; copy standard input */
        filecopy(stdin, stdout);
    else
        while (--argc > 0)
            if ((fp = fopen(*++argv, "r")) == NULL) {
                fprintf(stderr, "%s: can't open %s\n", prog, *argv);
                exit(1);
            } else {
                filecopy(fp, stdout);
                fclose(fp);
            }
            if (ferror(stdout)) {
                fprintf(stderr, "%s: error writing stdout\n", prog);
                exit(2);
            }
        exit(0);
}
```

Други функции за грешки при работа с файлове

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

- **ferror()** проверява дали са възникнали грешки при четене/запис на **fp**, връща **0** ако няма грешки, иначе число различно от **0** (например няма място на диска)
 - **feof()** проверява дали е достигнат края на **fp**, връща **0** ако не е достигнат. Ако е достигнат връща число различно от **0**
-
-

Въвеждане и извеждане на редове

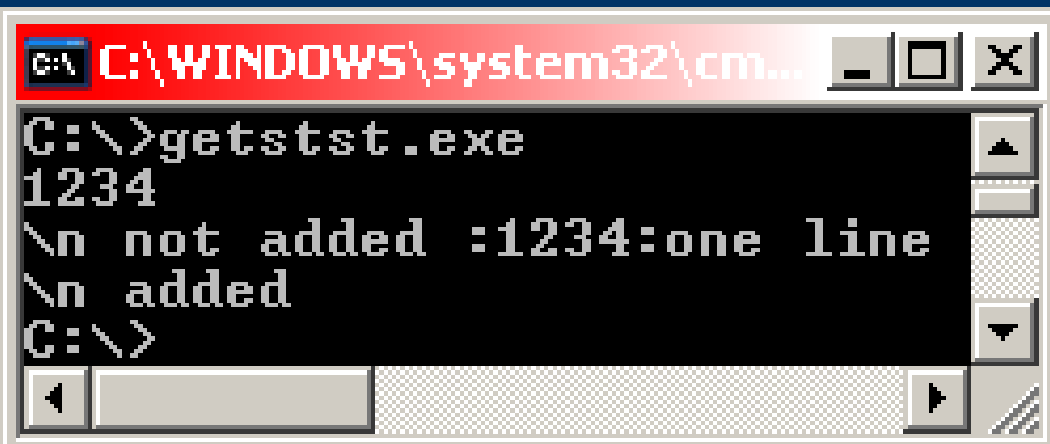
```
char *fgets(char *line, int maxline, FILE *fp);  
int fputs(char *line, FILE *fp);
```

- **fgets ()** чете цял ред (**включително** и '**\n**' накрая), но не повече от **maxline-1** символа, от файла **fp** и ги записва в **line**. **Поставя** '**\0**' на края на **line**. Връща **line** или **NULL** при край на файл или грешка
 - **fputs ()** извежда **line** в **fp**, връща **EOF** при грешка. Ако всичко е нормално връща положително число
 - Функциите **gets ()** и **puts ()** ползват **stdout** и **stdin**. Разликата е, че **gets ()** маха '**\n**' от **line**, а **puts ()** слага '**\n**' след като изведе **line**
-
-

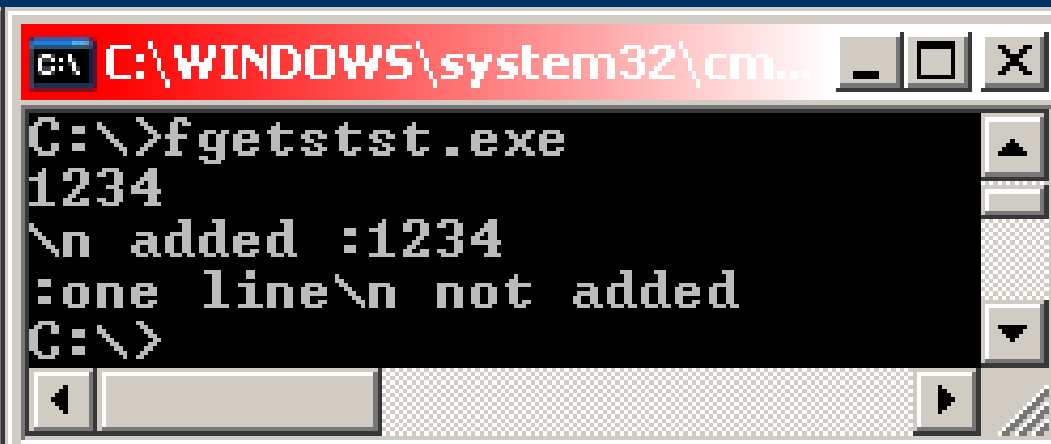
Пример

```
#include <stdio.h>
int main () {
    char s[50];
    gets(s);
    printf("\n not added:%s:",s);
    puts("one line");
    printf("\n added");
    return 0;
}
```

```
#include <stdio.h>
int main () {
    char s[50];
    fgets(s, 50, stdin);
    printf("\n added :%s:", s);
    fputs("one line", stdout);
    printf("\n not added");
    return 0;
}
```



```
C:\WINDOWS\system32\cmd...
C:\>getstst.exe
1234
\n not added :1234:one line
\n added
C:\>
```



```
C:\WINDOWS\system32\cmd...
C:\>fgetstst.exe
1234
\n added :1234
:one line\n not added
\n not added
C:\>
```

gets()

```
char *gets(char *line);
```

- **gets()** записва в **line** всички символи от стандартния вход, докато не срещне нов ред
- Не е препоръчително да се ползва функцията **gets()**, защото няма начин да се зададат максималният брой символи, които да се прочетат – може да продължи да пише в паметта и след края на променливата **line**

```
> gcc getstst.c
/tmp/cckTxiYM.o: In function 'main':
getstst.c:(.text+0x17): warning: the 'gets' function is
dangerous and should not be used.
```

Операции с низове (*string.h*)

- `strcat(s,t)` добавя `t` към края на `s`
 - `strncat(s,t,n)` добавя `n` символа от `t` в края на `s`
 - `strcmp(s,t)` сравнява низове. Връща $0 <$, 0 или >0 , съответно за $s < t$, $s == t$, $s > t$
 - `strncmp(s,t,n)` като `strcmp()`, но за първите `n` символа
 - `strcpy(s,t)` копира `t` в `s`
 - `strncpy(s,t,n)` копира най-много `n` символа от `t` в `s`
 - `strlen(s)` връща дължината на `s`
 - `strchr(s,c)` връща указател към първия символ `c` в низа `s` или `NULL` ако не е срещнат
 - `strrchr(s,c)` връща указател към последния символ `c` в низа `s` или `NULL` ако не е срещнат
-
-

Операции със символи (*ctype.h*)

Тези функции връщат `int`, който е както следва:

- `isalpha(c)` !0 ако `c` е буква, 0 - ако не е буква
 - `isupper(c)` !0 ако `c` е главна буква, 0 – ако е малка
 - `islower(c)` !0 ако `c` е малка буква, 0 – ако е главна
 - `isdigit(c)` !0 ако `c` е цифра, 0 – ако не е цифра
 - `isalnum(c)` !0 ако `c` е цифра или буква, 0 – вс. останало
 - `isspace(c)` !0 ако `c` е интервал, `tab`, `newline`, `return`, `formfeed`, `vertical tab`
 - `toupper(c)` връща `c` като главна буква или самото `c`, ако то не е буква
 - `tolower(c)` връща `c` като малка буква
-
-