Programação Estruturada

Prof. Rodrigo Hausen
http://progest.compscinet.org

Entrada e Saída
Parte 1

ENTRADA E SAÍDA DE DADOS

Até agora, nossa interação com os programas feitos em C foi feita quase que exclusivamente usando o interpretador cling como intermediário.

Toda vez que passamos parâmetros para alguma função no cling, o interpretador se encarregou de **ler os dados**, interpretá-los, e **imprimir** o resultado da execução de cada função.

Hoje, vamos começar a instruir nossos programas a processarem a entrada e saída de dados por meios próprios.

ENTRADA E SAÍDA PADRÃO

O dispositivo padrão para a **saída** de dados em um programa C geralmente é o **terminal** (“tela”) onde o programa está sendo executado.

O dispositivo padrão de **entrada** de dados geralmente é o **teclado**.

A entrada e saída padrão podem ser **redirecionadas** para outros dispositivos, para um arquivo ou até mesmo para a entrada/saída de outros programas.

Vamos começar trabalhando com a saída padrão.

SAÍDA PADRÃO

No Linux, edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  puts("Alô mundo!");
}
```

Compile-o, executando no terminal:

```
gcc alomundo.c -o alomundo
```

Execute-o com:

```
./alomundo
```

SAÍDA PADRÃO

```
usuario@maquina:~$ 
```

SAÍDA PADRÃO

```
usuario@maquina:~$ .
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./a
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./al
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alo
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alom
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomu
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomun
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomund
```

SAÍDA PADRÃO

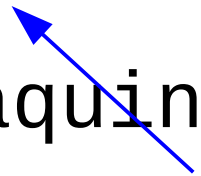
```
usuario@maquina:~$ ./alomundo
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$
```



```
puts("Alô mundo!");
```

Imprime o texto “Alô mundo!” na saída padrão, seguido de fim de linha.

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$ ./alomundo > arq.txt
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$ ./alomundo > arq.txt
usuario@maquina:~$
```

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$ ./alomundo > arq.txt
usuario@maquina:~$
```

Onde foi parar o texto “Alô mundo”?

Abra o arquivo arq.txt

O que há nele?

O que faz o caractere “>” na linha de comando?

SAÍDA PADRÃO

```
usuario@maquina:~$ ./alomundo
Alô mundo!
usuario@maquina:~$ ./alomundo > arq.txt
usuario@maquina:~$
```

Onde foi parar o texto “Alô mundo”?

Abra o arquivo arq.txt

O que há nele?

O que faz o caractere “>” na linha de comando?

Redireciona a saída padrão para um arquivo.

SAÍDA PADRÃO

A saída padrão, que **geralmente** é o terminal, pode ser redirecionada para um arquivo usando-se “> nome_de_arquivo” após o nome do programa.

Seu programa não precisa saber que a saída é para o terminal ou para um arquivo. Basta usar a saída padrão por meio das funções da biblioteca stdio.h (STandard Input/Output).

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Vamos ver as funções mais comuns da biblioteca para escrever na saída padrão.

- **int** puts(**const char** *****_s_)

  Escreve a string _s_, seguida de uma quebra de linha, na saída padrão.

  Retorna um número não-negativo caso a operação seja bem sucedida ou a constante EOF em caso de erro.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

- **int** printf(**const char** *****_formato_, ...);

  Escreve na saída padrão de acordo com a string *formato* e com os parâmetros que a seguem.

  A string *formato* pode ser qualquer coisa. A maioria dos caracteres é impressa como tal, **mas atenção para o caractere %!** Ele será explicado a seguir.

  Retorna o número de caracteres impressos ou um número negativo em caso de erro.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  puts("Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  puts("Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  (
"Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!\n");
}
```

Compile e execute.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!\n");
}
```

Compile e execute.

Resultado: o mesmo que com puts.

Se a string **não possui** o caractere **%**,
puts("...") equivale a printf("...\n")

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%sAlô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\nAlô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n"Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n",Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!\n");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!");
}
```

Compile e execute.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!");
}
```

Compile e execute.

A diretiva **%s**, contida no parâmetro *formato*, imprime a string passada como parâmetro.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s\n", "Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô", mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô", mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô", "mundo!");
}
```

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô", "mundo!");
}
```

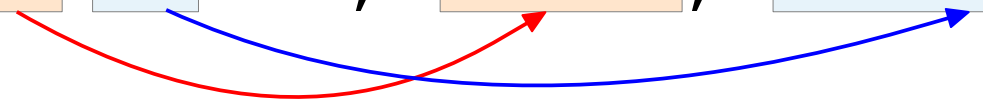
Compile e execute.

FUNÇÕES DE SAÍDA DA BIBLIOTECA PADRÃO

Edite novamente o programa alomundo.c

```
#include <stdio.h>

int main(void) {
  printf("%s %s\n", "Alô", "mundo!");
}
```



Compile e execute.

Cada diretiva %… corresponde a um parâmetro na ordem em que aparecem em *formato*.

DIRETIVAS DA FUNÇÃO PRINTF

Mais comuns:

%**s** – imprime o **char** * (**s**tring) passado como parâmetro. Deve ser um ponteiro **válido**.
%**d** – imprime **int** na base 10 (**d**ecimal)
%**ld** – imprime **long int** na base 10
%**u** – imprime **unsigned int** na base 10
%**f** – imprime **float**[1] na base 10
%**lf** – imprime **double**[1] na base 10 (**l**ong **f**loat)
%**e** – imprime **float** ou **double** em notação científica (constante e **e**xpoente)
%g – escolhe automaticamente a representação mais curta entre %f e %e

[1] Na prática, printf converte float para double, logo %f e %lf têm o mesmo significado.

DIRETIVAS DA FUNÇÃO PRINTF

Obtenha o arquivo alunos.c no site.

Crie o programa imprimealunos.c que contém uma função *imprimetxt*, que imprime RA, nome, faltas, nota da P1 e nota da P2, nesta ordem, separadas por três espaços. Os dados de cada aluno devem ser impresos em linhas separadas.

A função *main* deve chamar a função *imprimealunos*.

imprimealunos.c

```
#include <stdio.h>
#include "alunos.c"

void imprimetxt(int n, int ra[n],
                const char *nome[n],
                int falta[n],
                float p1[n], float p2[n]) {
  ...
}

int main(void) {
  imprimetxt(sizeof(ra)/sizeof(ra[0]),
             ra, nome, falta, p1, p2);
}
```

```
10048122   Vitoria Barros Costa   1   10.000000   6.000000
10105423   Matheus Alencar Castilho   1   5.500000   4.000000
10093021   Leonor Duncan Azevedo   3   6.000000   5.000000
20067223   Joao Carlos Esteves Bertrand Lopes da Silva Carvalho
  1   10.000000   1.000000
10117320   Lavinia Barreto Cunha   12   0.000000   10.000000
20032021   Yasmin Cunha Lima   2   10.000000   0.000000
20065721   Kaua dos Santos Carvalho   0   10.000000   10.000000
20101720   Murilo D'Alem-Mar Correia   1   9.000000   8.000000
10055420   Matheus Socorro Rodrigues   1   7.000000   5.000000
10063523   Gabriela Dias Nascimento   0   9.000000   10.000000
20013623   Carla Fernandes Pereira   0   8.000000   4.000000
10047720   Rodrigo Lorenzo Lima   0   10.000000   5.000000
20080423   Agatha Santiago Almeida   1   10.000000   10.000000
10106120   Tiago Arcoverde Goncalves   1   5.500000   1.000000
10103921   Yasmin Pancetti Barros   0   7.000000   10.000000
20086522   Matheus Oliveira Castro   1   9.000000   4.000000
10111122   Gabriel Basile Almeida   3   6.000000   4.000000
20047323   Pedro Camargo Ferreira   0   10.000000   5.000000
10028421   Vinicius Coutinho Fernandes   0   10.000000
4.000000
10077121   Caua Leonel Dias   0   9.500000   5.000000
20082721   Bruno Ambrosio Barbosa   0   10.000000   3.000000
10085822   Fernanda Calixto Cardoso   1   9.000000   1.000000
```

TAMANHO DO CAMPO

Cada diretiva %d, %ld, %u %s, %f, %lf e %g pode ter um **tamanho de campo**, um número que vai entre o sinal % e o caractere que descreve o formato.

Ex.:

```
printf("%d %f\n", 10, 10);
printf("%d %f\n", 1, 7.5);
```

Resulta em colunas desalinhadas

```
10 10.000000
1 7.5000000
```

TAMANHO DO CAMPO

Cada diretiva %d, %ld, %u %s, %f, %lf e %g pode ter um **tamanho de campo**, um número que vai entre o sinal % e o caractere que descreve o formato.

Ex.:

```
printf("%2d %4.1f\n", 10, 10);
printf("%2d %4.1f\n", 1, 7.5);
```

Resulta em colunas alinhadas

```
10 10.0
 1  7.5
```

TAMANHO DO CAMPO

Cada diretiva %d, %ld, %u %s, %f, %lf e %g pode ter um **tamanho de campo**, um número que vai entre o sinal % e o caractere que descreve o formato.

2 colunas para o int

4 colunas para o float, com 1 coluna para parte fracionária

Ex.:

```
printf("%2d %4.1f\n", 10, 10);
printf("%2d %4.1f\n", 1, 7.5);
```

espaço entre as colunas

Resulta em colunas alinhadas

```
10 10.0
 1  7.5
```

TAMANHO DO CAMPO

Volte à função *imprimetxt* e reserve:

- 8 colunas para o RA
- 30 colunas para o nome
- 2 colunas para as faltas
- 4 colunas e 1 casa após a vírgula para as notas da p1 e da p2

10048122	Vitoria Barros Costa	1	10.0	6.0
10105423	Matheus Alencar Castilho	1	5.5	4.0
10093021	Leonor Duncan Azevedo	3	6.0	5.0
20067223	Joao Carlos Esteves Bertrand Lopes da Silva Carvalho	1	10.0	1.0
10117320	Lavinia Barreto Cunha	12	0.0	10.0
20032021	Yasmin Cunha Lima	2	10.0	0.0
20065721	Kaua dos Santos Carvalho	0	10.0	10.0
20101720	Murilo D'Alem-Mar Correia	1	9.0	8.0
10055420	Matheus Socorro Rodrigues	1	7.0	5.0
10063523	Gabriela Dias Nascimento	0	9.0	10.0
20013623	Carla Fernandes Pereira	0	8.0	4.0
10047720	Rodrigo Lorenzo Lima	0	10.0	5.0
20080423	Agatha Santiago Almeida	1	10.0	10.0
10106120	Tiago Arcoverde Goncalves	1	5.5	1.0
10103921	Yasmin Pancetti Barros	0	7.0	10.0
20086522	Matheus Oliveira Castro	1	9.0	4.0
10111122	Gabriel Basile Almeida	3	6.0	4.0
20047323	Pedro Camargo Ferreira	0	10.0	5.0
10028421	Vinicius Coutinho Fernandes	0	10.0	4.0
10077121	Caua Leonel Dias	0	9.5	5.0
20082721	Bruno Ambrosio Barbosa	0	10.0	3.0
10085822	Fernanda Calixto Cardoso	1	9.0	1.0
10110722	Jose Roberto Serafim Cardoso	0	9.0	6.0
20024820	Tiago Fonseca Cardoso	5	2.0	4.0

TAMANHO DO CAMPO

Ainda há pequenos ajustes a fazer na função *imprimetxt*:

1. *nome* está alinhado pela direita, mas geralmente texto é alinhado pela esquerda
2. há nomes de alunos que possuem mais de 30 caracteres

Para resolver 1, troque o formato **%30s** para **%-30s**. Compile e execute.

TAMANHO DO CAMPO

Ainda há pequenos ajustes a fazer na função *imprimetxt*:

1. *nome* está alinhado pela direita, mas geralmente texto é alinhado pela esquerda
2. há nomes de alunos que possuem mais de 30 caracteres

Para resolver 1, troque o formato **%30s** para **%-30s**. Compile e execute.

Para resolver 2, altere novamente o formato **%-30s** para **%-30.30s**. Compile e execute.

10048122	Vitoria Barros Costa	1	10.0	6.0
10105423	Matheus Alencar Castilho	1	5.5	4.0
10093021	Leonor Duncan Azevedo	3	6.0	5.0
20067223	Joao Carlos Esteves Bertrand L	1	10.0	1.0
10117320	Lavinia Barreto Cunha	12	0.0	10.0
20032021	Yasmin Cunha Lima	2	10.0	0.0
20065721	Kaua dos Santos Carvalho	0	10.0	10.0
20101720	Murilo D'Alem-Mar Correia	1	9.0	8.0
10055420	Matheus Socorro Rodrigues	1	7.0	5.0
10063523	Gabriela Dias Nascimento	0	9.0	10.0
20013623	Carla Fernandes Pereira	0	8.0	4.0
10047720	Rodrigo Lorenzo Lima	0	10.0	5.0
20080423	Agatha Santiago Almeida	1	10.0	10.0
10106120	Tiago Arcoverde Goncalves	1	5.5	1.0
10103921	Yasmin Pancetti Barros	0	7.0	10.0
20086522	Matheus Oliveira Castro	1	9.0	4.0
10111122	Gabriel Basile Almeida	3	6.0	4.0
20047323	Pedro Camargo Ferreira	0	10.0	5.0
10028421	Vinicius Coutinho Fernandes	0	10.0	4.0
10077121	Caua Leonel Dias	0	9.5	5.0
20082721	Bruno Ambrosio Barbosa	0	10.0	3.0
10085822	Fernanda Calixto Cardoso	1	9.0	1.0
10110722	Jose Roberto Serafim Cardoso	0	9.0	6.0

DIRETIVAS DA FUNÇÃO PRINTF

Outras diretivas:

%% – imprime % (não corresponde a nenhum parâmetro após o *formato*)

%c – imprime caractere passado como parâmetro

%p – imprime endereço de ponteiro em hexadecimal

%x – imprime **unsigned int** em hexadecimal

%o – imprime **unsigned int** em octal

ENTRADA DE DADOS

Há várias maneiras de fornecer dados a um programa em C:

- por argumentos de linha de comando
- pela entrada padrão
- por arquivos
- por meio de dispositivos

Hoje vamos ver as duas primeiras formas.

ARGUMENTOS DE LINHA DE COMANDO

Um programa completo em C deve ter uma função *main*, declarada como:

```
int main(void) {
   ...
}
```

ou como:

```
int main(int argc, char *argv[]) {
   ...
}
```

A segunda forma permite obter os argumentos passados para o programa.

ARGUMENTOS DE LINHA DE COMANDO

Edite o programa teste.c, compile-o e execute-o:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  for (i=0; i<argc; ++i) {
    printf("argv[%d] é \"%s\"\n", i, argv[i]);
  }
}
```

```
$ 
```

ARGUMENTOS DE LINHA DE COMANDO

Edite o programa teste.c, compile-o e execute-o:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  for (i=0; i<argc; ++i) {
    printf("argv[%d] é \"%s\"\n", i, argv[i]);
  }
}
```

```
$ gcc teste.c -o teste
```

ARGUMENTOS DE LINHA DE COMANDO

Edite o programa teste.c, compile-o e execute-o:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  for (i=0; i<argc; ++i) {
    printf("argv[%d] é \"%s\"\n", i, argv[i]);
  }
}
```

```
$ gcc teste.c -o teste
$
```

ARGUMENTOS DE LINHA DE COMANDO

Edite o programa teste.c, compile-o e execute-o:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  for (i=0; i<argc; ++i) {
    printf("argv[%d] é \"%s\"\n", i, argv[i]);
  }
}
```

```
$ gcc teste.c -o teste
$ ./teste arg1 arg2
```

ARGUMENTOS DE LINHA DE COMANDO

Edite o programa teste.c, compile-o e execute-o:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  for (i=0; i<argc; ++i) {
    printf("argv[%d] é \"%s\"\n", i, argv[i]);
  }
}
```

```
$ gcc teste.c -o teste
$ ./teste arg1 arg2
argv[0] é "./teste"
argv[1] é "arg1"
argv[2] é "arg2"
```

ARGUMENTOS DE LINHA DE COMANDO

```
int main(int argc, char *argv[]) {
  ...
}
```

Quantidade de elementos no array `argv`*.*

*Array de argumentos passados para o programa. Cada argumento é uma string (char *)*

```
$ ./teste arg1 arg2
argv[0] é "./teste"
argv[1] é "arg1"
argv[2] é "arg2"
```

`argv[0]` ***sempre*** *é a forma como o programa foi chamado*

Os argumentos passados ao programa, se existirem, serão `argv[1]`*,* `argv[2]`*,...*

ARGUMENTOS DE LINHA DE COMANDO

Volte a imprimealunos.c e faça as seguintes alterações.

1 O programa deve receber exatamente um argumento, uma string “txt” ou “csv”, que determina qual o formato da saída

2 Implemente uma função *ajuda*, que será executada se não for passado nenhum argumento ao programa, ou se for passado mais de um argumento, ou se o argumento não for nenhuma das strings acima. Neste caso, o programa deve terminar após mostrar o texto de ajuda

ARGUMENTOS DE LINHA DE COMANDO

3 Se o argumento for “txt” chama a função imprimetxt

4 Se o argumento for “csv” chama a função imprimecsv, que imprime os dados no formato CSV. Os nomes **devem** ser limitados a, no máximo, 60 bytes e os floats devem ser impressos com 1 casa decimal após o ponto.

Formato CSV (Comma Separated Values):
lin1 col1,lin1 col2,…,lin1 col*N*
lin2 col1,lin2 col2,…,lin2 col*N*
…

ARGUMENTOS DE LINHA DE COMANDO

```
$ ./imprimealunos
```

ARGUMENTOS DE LINHA DE COMANDO

```
$ ./imprimealunos
Ajuda: imprimealunos [csv|txt]
$
```

ARGUMENTOS DE LINHA DE COMANDO

```
$ ./imprimealunos
Ajuda: imprimealunos [csv|txt]
$ ./imprimealunos csv
```

ARGUMENTOS DE LINHA DE COMANDO

```
$ ./imprimealunos
Ajuda: imprimealunos [csv|txt]
$ ./imprimealunos csv
10048122,Vitoria Barros Costa,1,10.0,6.0
10105423,Matheus Alencar Castilho,1,5.5,4.0
10093021,Leonor Duncan Azevedo,3,6.0,5.0
20067223,Joao Carlos Esteves Bertrand Lopes da Silva,…
10117320,Lavinia Barreto Cunha,12,0.0,10.0
20032021,Yasmin Cunha Lima,2,10.0,0.0
20065721,Kaua dos Santos Carvalho,0,10.0,10.0
20101720,Murilo D'Alem-Mar Correia,1,9.0,8.0
10055420,Matheus Socorro Rodrigues,1,7.0,5.0
10063523,Gabriela Dias Nascimento,0,9.0,10.0
20013623,Carla Fernandes Pereira,0,8.0,4.0
10047720,Rodrigo Lorenzo Lima,0,10.0,5.0
20080423,Agatha Santiago Almeida,1,10.0,10.0
10106120,Tiago Arcoverde Goncalves,1,5.5,1.0
10103921,Yasmin Pancetti Barros,0,7.0,10.0
20086522,Matheus Oliveira Castro,1,9.0,4.0
```

REDIRECIONANDO A SAÍDA

Redirecione a saída de imprimealunos csv para o arquivo alunos.csv

```
$ ./imprimealunos csv > alunos.csv
```

Arquivos CSV podem ser importados como planilhas no LibreOffice Calc.

Ou podem ser processados por outros programas feitos em C.

Para isso, precisamos primeiramente aprender como fazer **entrada de dados pela entrada padrão**.

LENDO A ENTRADA PADRÃO

Há diversas funções para ler dados da entrada padrão.

Começaremos por:

- **int** scanf(**const char** *****_formato, ..._);

Lê dados **formatados** da entrada padrão. A string *formato* deve conter **especificadores de conversão**. O resultado da(s) conversão (ões), se houver, serão armazenados nas posições indicadas pelos **ponteiros** passados como argumentos.

Cada ponteiro deve ser de um tipo apropriado ao valor retornado pela conversão especificada.

ESPECIFICADORES DE CONVERSÃO PARA SCANF

%d – converte para **int**

%u – converte para **unsigned int**

%ld – converte para **long int**

%f – converte para **float**

%lf – converte para **double**

Os especificadores de conversão para scanf são parecidos com as diretivas de printf, **MAS CUIDADO: eles não são idênticos!**

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Exemplos: (execute no cling)

Lê o próximo inteiro e o próximo float da entrada padrão

```
[cling]$ #include <stdio.h>
[cling]$ int i; float f;
[cling]$ scanf("%d %f", &i, &f)
```

Digite “10 9.9” e veja os valores armazenados em i e f.

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Exemplos: (execute no cling)

Lê o próximo inteiro e o próximo float da entrada padrão

```
[cling]$ #include <stdio.h>
[cling]$ int i; float f;
[cling]$ scanf("%d %f", &i, &f)
```

Digite “10 9.9” e veja os valores armazenados em i e f.

```
[cling]$ i
(int) 10
[cling]$ f
(float) 9.90000f
```

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Reinicie o cling e execute novamente

```
[cling]$ #include <stdio.h>
[cling]$ int i; float f;
[cling]$ scanf("%d %f", &i, &f)
```

Desta vez, digite “10 abc 9.9” e veja o resultado.

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Reinicie o cling e execute novamente

```
[cling]$ #include <stdio.h>
[cling]$ int i; float f;
[cling]$ scanf("%d %f", &i, &f)
```

Desta vez, digite “10 abc 9.9” e veja o resultado.

```
[cling]$ i
(int) 10
[cling]$ f
(float) 0.00000f
```

Apenas uma das conversões foi bem sucedida. Observe que scanf retornou 1, a quantidade de conversões que foram executadas corretamente.

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Para ler strings usando scanf, devem ser usados os especificadores:

%***n***[***xyz…***] - lê, no máximo, ***n*** caracteres **iguais a *x* ou *y* ou *z* ou** …

%***n***[^***xyz…***] - lê, no máximo, ***n*** caracteres **diferentes** de ***x***, ***y***, ***z***, …

Exemplos:

```
char telefone[21];
scanf("%20[-+()0123456789]", telefone);
```

Um byte a menos que o tamanho do array, para poder armazenar o caractere nulo ao final.

ESPECIFICADORES DE CONVERSÃO PARA SCANF

Para ler strings usando scanf, devem ser usados os especificadores:

%***n***[***xyz…***] - lê, no máximo, ***n*** caracteres **iguais a *x* ou *y* ou *z* ou** …

%***n***[^***xyz…***] - lê, no máximo, ***n*** caracteres **diferentes** de ***x***, ***y***, ***z***, …

Exemplos:

```
char telefone[21];
scanf("%20[-+()0123456789]", telefone);
```

Caracteres permitidos: algarismos, parênteses, + e -
*Se - for permitido, **sempre** deve vir primeiro.*

LENDO CSV DA ENTRADA PADRÃO

Crie o arquivo avaliaalunos.c e, na função *main*, use scanf para ler o CSV da entrada padrão, linha por linha, e em seguida calcular a média de cada aluno, o conceito final dele nessa disciplina e imprimir RA, nome, média e conceito no formato CSV.

Média: 0,4 P1 + 0,6 P2

Conceito: se faltas > 6, então letra O
caso contrário, média < 5 é F
5 ≤ média < 6 é D
6 ≤ média < 7,5 é C
7,5 ≤ média < 9 é B
média ≥ 9 é A

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
```

(por que nada acontece?)

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
```

Digite uma linha válida no formato CSV

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
```

Pressione Ctrl + C para parar o programa.

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
^C
$
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
^C
$
```

Vamos redirecionar o arquivo alunos.csv para a entrada padrão do programa.

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
^C
$ ./avaliaalunos < alunos.csv
```

LENDO CSV DA ENTRADA PADRÃO

```
$ ./avaliaalunos
10048122,Vitoria Barros Costa,1,10.0,6.0
10048122,Vitoria Barros Costa,7.6,B
^C
$ ./avaliaalunos < alunos.csv
```

Redireciona o arquivo alunos.csv para a entrada padrão do programa.

ENTRADA E SAÍDA EM C

A biblioteca stdio.h carrega uma herança de mais de **40 anos** de implementações.

Vantagem: estabilidade.

Desvantagem: as especificações das funções são antigas e não podem ser alteradas. Uma reimplementação poderia eliminar vários problemas.

Exemplo de problemas:

- a função *gets* é insegura e **não deve ser usada nunca**. Mas continua na biblioteca
- entrada/saída de strings é muito primitiva (problemas com acentos, etc.)

ENTRADA E SAÍDA EM C

A função *scanf* **jamais** deve ser usada para ler dados do usuário, uma vez que o usuário pode fornecer dados errôneos!
(The User is Drunk - youtu.be/r2CbbBLVaPk)

Há problemas ao usar *scanf* caso os dados não sejam bem formatados.

Se tiver de lidar com entrada direta do usuário, **sempre** leia os dados para string, depois converta (funções *strtol*, *strtod*, etc)

Recomenda-se usar **apenas** as funções *getchar* e *fgets* para entrada direta do usuário.

ENTRADA DO USUÁRIO

Exemplificando os problemas:

Crie o arquivo digitealgo.c e faça um programa que imprima na tela:

Digite algo para continuar.

E espere até que o usuário digite algo. Se o usuário pressionar alguma tecla, o programa imprime o caractere digitado e pede para o usuário digitar um número inteiro. Em seguida, imprime o número.

Use a função *getchar* e scanf.

ENTRADA DO USUÁRIO

```
#include <stdio.h>
int main(void) {
  char c;
  int n;
  puts("Digite algo para continuar.");
  c = getchar();
  printf("Você digitou: %c\n", c);
  puts("Digite um inteiro:");
  scanf("%d", &n);
  printf("O número é %d\n", n);
}
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
```

Digite um número qualquer, mas não pressione Enter.

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
1
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
12
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
12
```

Ô diacho... por que não continua?

getchar não deveria pegar só UM caractere?

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
123
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
1234
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
12345
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
12345
```

OK... isto não tem graça!

Pressione Enter

ENTRADA DO USUÁRIO

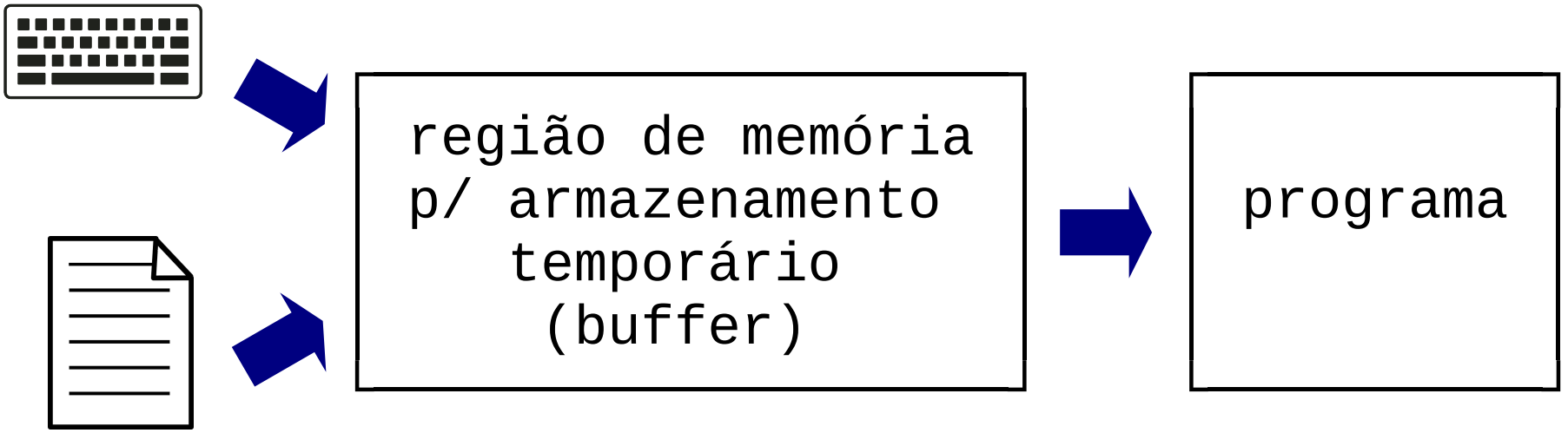
```
$ ./digitealgo
Digite algo para continuar.
12345
Você digitou: 1
Digite um inteiro:
O número é 2345
```

ENTRADA DO USUÁRIO

```
$ ./digitealgo
Digite algo para continuar.
12345
Você digitou: 1
Digite um inteiro:
O número é 2345
```

MAS HEIN!? Por que meu programa não esperou que eu digitasse o número inteiro?

BUFFERIZAÇÃO



A entrada padrão, venha ela do teclado ou de um arquivo, é **buferizada por linha**:

1. os dados são lidos do teclado/arquivo
2. então são armazenados em uma região da memória até que o usuário pressione Enter (ou até o buffer encher)
3. só então os dados são disponibilizados para as funções de stdio.h

ENTRADA DO USUÁRIO

```
#include <stdio.h>
int main(void) {
  char c;
  int n;
  puts("Digite Enter para continuar.");
  do {
    c = getchar();
  } while (c != '\n' && c != -1);
  if (c == -1) return 1;
  puts("Digite um inteiro:");
  scanf("%d", &n);
  printf("O número é %d\n", n);
}
```

PARA CASA

Usando as funções *getchar*, *strtol* e *isspace*, crie a função:

int lelong(**const char** ***prompt,** **long** ***n)

que imprime o prompt na tela e lê um número inteiro do usuário. Se a entrada do usuário for incorreta, repete o processo. Se for correta, coloca o número em n.

A função retorna 1 caso a conversão seja bem sucedida ou 0 em caso de erro.

Use um array com 100 caracteres. Despreze todos os caracteres em branco (use *isspace*) no início da entrada do usuário.

Teste:

```
int main(void) {
  long n;
  int ok = lelong("Digite inteiro: ", &n);
  if (ok) {
    printf("Leu: %ld\n", n);
  } else {
    printf("Erro\n", n);
  }
}
```

Teste:

```
$ ./lelong
```

Teste:

```
$ ./lelong
Digite inteiro:
```

Teste:

```
$ ./lelong
Digite inteiro: inteiro
```

Teste:

```
$ ./lelong
Digite inteiro: inteiro
Digite inteiro: 10
```

Teste:

```
$ ./lelong
Digite inteiro: inteiro
Digite inteiro: 10
Leu: 10
$
```

Teste:

```
$ ./lelong
Digite inteiro: inteiro
Digite inteiro: 10
Leu: 10
$ ./lelong
```

Teste:

```
$ ./lelong
Digite inteiro: inteiro
Digite inteiro: 10
Leu: 10
$ ./lelong
Digite inteiro:
```

Pressione Ctrl + D (se estiver no Linux/Mac OS) ou Ctrl + Z (no Windows)

Estas combinações de teclas sinalizam ao programa que a entrada padrão foi finalizada, ou seja, não há mais dados a serem entrados.

Teste:

```
$ ./lelong
Digite inteiro: inteiro
Digite inteiro: 10
Leu: 10
$ ./lelong
Digite inteiro: Erro
$
```