```
int main(int argc, char** argv)
{
  char c = 0;
  char* commands = "ads pq"; // key commands: "left,right,rotate,confirm,pause,quit"
  int speed = 2; // sets max moves per row
  int moves_to_go = 2;
  int full = 0; // whether board is full
  init(); // initialize board an tetrominoes


  cur =
  state[
  state[

  system
  printf

  while
    {
      dr

    // process user action
    c = getchar();   // get new action
    if (c == commands[0] && !intersect(cur, state[0]-1, state[1])) state[0]--; // move left
    if (c == commands[1] && !intersect(cur, state[0]+1, state[1])) state[0]++; // move right
    if (c == commands[2] && !intersect(cur->rotated, state[0], state[1])) cur = cur->rotated;
    if (c == commands[3]) moves_to_go=0;

    // scroll down
    if (!moves_to_go--)
      {
        if (intersect(cur,state[0],state[1]+1)) // if tetromino intersected with sth
          {
            cramp_tetromino();
            remove_complete_lines();
            cur = &tetrominoes[rand() % NUM_POSES];
            state[0] = (WIDTH - cur->width)/2;
```


Manipulação de texto

Qual a saída?

```
1  #define N 10
2  int *p;
3  int i = 0;
4  int v[N];
5  for (int i = 0; i < N; i++) {
6    v[i] = i;                    /* v = [0 1 2 3 4 5 6 7 8 9] */
7  }
8  p = &(v[N-1]);
9  while (*p) {
10   --p; i++;
11 }
12 printf("%d %d", *p, i);
```

Qual a saída?

```c
#define N 10
int *p;
int i = 0;
int v[N];
for (int i = 0; i < N; i++) {
  v[i] = i;                    /* v = [0 1 2 3 4 5 6 7 8 9] */
}
p = &(v[N-1]);
while (*p) {
   --p; i++;
}
printf("%d %d", *p, i);
```

```
0 9
```

Caracteres

- Caracteres são representados como números; por isso, é necessária uma codificação (ou seja, uma associação entre números e símbolos)
- A codificação ASCII (American Standard Code for Information Interchange) é a mais comum
- Cada código ASCII possui 7 bits, portanto $2^7 = 128$ símbolos são representáveis
- Existem símbolos imprimíveis (letras, números, pontuação) e não imprimíveis (quebra de linha, fim de arquivo, etc)
- Existem outras codificações (p.ex. Latin1) que estendem ASCII para símbolos comumente presentes em outras línguas (ex. caracteres acentuados, cedilha)

Tabela ASCII

32:	33: !	34: "	35: #	36: $	37: %	38: &	39: ’
40: (	41:)	42: *	43: +	44: ,	45: -	46: .	47: /
48: 0	49: 1	50: 2	51: 3	52: 4	53: 5	54: 6	55: 7
56: 8	57: 9	58: :	59: ;	60: <	61: =	62: >	63: ?
64: @	65: A	66: B	67: C	68: D	69: E	70: F	71: G
72: H	73: I	74: J	75: K	76: L	77: M	78: N	79: O
80: P	81: Q	82: R	83: S	84: T	85: U	86: V	87: W
88: X	89: Y	90: Z	91: [	92: \	93:]	94: ^	95: _
96: ‘	97: a	98: b	99: c	100: d	101: e	102: f	103: g
104: h	105: i	106: j	107: k	108: l	109: m	110: n	111: o
112: p	113: q	114: r	115: s	116: t	117: u	118: v	119: w
120: x	121: y	122: z	123: {	124: |	125: }	126: ~	

Tabela ISO-8859-1 (latin1)

32		64	@	96	`	162	¢	194	Â	226	â
33	!	65	A	97	a	163	£	195	Ã	227	ã
34	"	66	B	98	b	164	¤	196	Ä	228	ä
35	#	67	C	99	c	165	¥	197	Å	229	å
36	$	68	D	100	d	166	¦	198	Æ	230	æ
37	%	69	E	101	e	167	§	199	Ç	231	ç
38	&	70	F	102	f	168	¨	200	È	232	è
39	'	71	G	103	g	169	©	201	É	233	é
40	(	72	H	104	h	170	ª	202	Ê	234	ê
41	)	73	I	105	i	171	«	203	Ë	235	ë
42	*	74	J	106	j	172	¬	204	Ì	236	ì
43	+	75	K	107	k	173		205	Í	237	í
44	,	76	L	108	l	174	®	206	Î	238	î
45	-	77	M	109	m	175	¯	207	Ï	239	ï
46	.	78	N	110	n	176	°	208	Ð	240	ð
47	/	79	O	111	o	177	±	209	Ñ	241	ñ
48	0	80	P	112	p	178	²	210	Ò	242	ò
49	1	81	Q	113	q	179	³	211	Ó	243	ó
50	2	82	R	114	r	180	´	212	Ô	244	ô
51	3	83	S	115	s	181	µ	213	Õ	245	õ
52	4	84	T	116	t	182	¶	214	Ö	246	ö
53	5	85	U	117	u	183	·	215	×	247	÷
54	6	86	V	118	v	184	¸	216	Ø	248	ø
55	7	87	W	119	w	185	¹	217	Ù	249	ù
56	8	88	X	120	x	186	º	218	Ú	250	ú
57	9	89	Y	121	y	187	»	219	Û	251	û
58	:	90	Z	122	z	188	¼	220	Ü	252	ü
59	;	91	[	123	{	189	½	221	Ý	253	ý
60	<	92	\	124	|	190	¾	222	Þ	254	þ
61	=	93	]	125	}	191	¿	223	ß	255	ÿ
62	>	94	^	126	~	192	À	224	à		
63	?	95	_	161	¡	193	Á	225	á		

Caracteres

`char c, *c;`
Representa uma palavra de computador com sinal (em geral, um byte=8 bits: -128,-127,...,0,1,...,127)
`unsigned char c, *c;`
Representa uma palavra de computador sem sinal (0,1,...,255)

```
1  #include <stdio.h>
2  /* exibe tabela ascii */
3  int main (void) {
4    unsigned char i;
5    for (i = 32; i < 127; i++) {
6      printf ("%3d: %1c  ", i, i);
7      if ((i + 1) % 8 == 0) printf ("\n");
8    }
9    printf ("\n");
10   return 0;
11 }
```

Caracteres constantes e brancos

`'n'`

Código ASCII do símbolo `n`

```
1 #include <stdio.h>
2 /* exibe codigo ascii dos
      caracteres 'a' a 'z' */
3 int main (void) {
4   unsigned char i;
5   for (i = 'a'; i <= 'z'; i++)
6     printf ("%3d: %1c  ", i, i);
7   printf ("\n");
8   return 0;
9 }
```

código	constante	símbolo
0	`'\0'`	null
9	`'\t'`	tab
10	`'\n'`	newline
13	`'\r'`	return
32	`' '`	space
39	`'\''`	'
65	`'A'`	A
97	`'a'`	a

Entrada e saída de caracteres

`int printf ("%c", ch), scanf ("%c", &ch);`

Escreve/Lê caractere e retorna 1 ou EOF se não foi possível

`int getchar (void), getc (FILE *f);`

Lê caractere e retorna seu valor ou EOF se não foi possível

`int putchar (char ch), putc (FILE *f, char ch);`

Escreve caractere e retorna caractere lido ou EOF

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5      int ch;
6      while( (ch = getchar ()) != EOF ){
7              printf ("Caractere: %c\n", ch);
8      }
9      return 0;
10 }
```

Manipulação de caracteres

`int isalpha(int c), isdigit(int c)`
Verifica se caractere é uma letra/dígito
`int tolower(char c), toupper(char c)`
Converte caractere maiúsculo/minúsculo em equivalente minúsculo/maiúsculo

```
1 char c = tolower ('A');
2 if (isaplha (c) && (c == 'a')) printf ("A");
```

Strings

`char *s;`

Cadeia de caracteres terminada em `'\0'`

Uma string é armazenada em um vetor e representada por um ponteiro para o primeiro caractere

Tamanho ou comprimento de uma string: número de caracteres menos um (sem contar o `'\0'`)

```
1 /* string de tamanho 6 contendo "MAC122" */
2 char s1[] = {'M', 'A', 'C', '1', '2', '2', '\0', 'P', '1'};
3 /* string de tamanho 5 contendo "M A C" */
4 char s2[] = {'M', ' ', 'A', ' ', 'C', '\0'};
5 /* não é string */
6 char s3[] = {'M', 'A', 'C'};
```

Strings constantes

Representadas por uma sequência de caracteres delimitados por aspas

`(char *) "string"`

```
1 char *s = "MAC 122";
2 /* s[0] = 'M', s[1] = 'A', s[2] = 'C', s[3] = ' ',
3    s[4] = '1', s[5] = '2', s[6] = '2', s[7] = '\0'  */
```

Strings

```
1 /* Conta o número de espaços em uma string */
2 int conta_espaco (char *s) {
3     /* como? */
4 }
```

Strings

```
1 /* Conta o número de espaços em uma string */
2 int conta_espaco (char *s) {
3   int i, n = 0;
4   for (i=0; s[i] != '\0'; i++)
5       if (s[i] == ' ') n++;
6   return n++;
7 }
```

Strings

```c
/* Conta o número de espaços em uma string
 * usa um ponteiro como iterador
 */
int conta_espaco2 (char *s) {
  int n = 0; char *p;
  /* como? */
  return n++;
}
```

Strings

```
1 /* Conta o número de espaços em uma string
2  * usa um ponteiro como iterador
3  */
4 int conta_espaco2 (char *s) {
5   int n = 0; char *p;
6   for (p=s; *p != '\0'; p++)
7       if (*p == ' ') n++;
8   return n++;
9 }
```

Qual a diferença entra `"A"` e `'A'`?

Qual a diferença entra `"A"` e `'A'`? `"A"` é a string `'A', '\0'`

Qual o tamanho da string `"MAC\n"`?

Qual o tamanho da string `"MAC\n"`? `4`

Biblioteca string

`unsigned int strlen(char *s)`

Retorna tamanho de uma string

`string strcpy(char *s, char *t)`

Copia string `t` na string `s`

`int strcmp(char *s, char *t)`

Compara strings de acordo com ordem lexicográfica

Biblioteca string

`unsigned int strlen(char *s)`

Retorna tamanho de uma string

```
1  #include <string.h>
2
3  unsigned int strlen (char *s) {
4    int i;
5    for (i = 0; s[i] != '\0'; i++) ;
6    return i;
7  }
8
9  char *s = "ABC";
10 printf ("%d", strlen(s));
```

Biblioteca string

`string strcpy(char *s1, char *s2)`

Copia string `s2` na string `s1`, devolve ponteiro para `s1`

```
1  #include <string.h>
2
3  string strcpy (char *str1, char *str2) {
4    int i;
5    for (i = 0; str2[i] != '\0'; i++) str1[i] = str2[i];
6    str1[i] = '\0';
7    return str1;
8  }
9
10 char s[10];
11 strcpy (s, "ABC");
12 printf ("%s", s); /* Exibe ABC */
```

Ordem lexicográfica

Strings são ordenadas como verbetes num dicionário

Dadas strings distintas `s` e `t`, deixe `k` ser o menor inteiro tal que `s[k]` difere de `t[k]`:

- Se `s[k] < t[k]` então `s` é lexicograficamente menor que `t`
- Se `s[k] > t[k]` então `s` é lexicograficamente maior que `t`

```
abc < acb
a < aa
apartamento < casa
apartamento > Casa
```

Biblioteca string

`int strcmp(char *s, char *t)`

Compara strings de acordo com ordem lexicográfica

Devolve um número negativo se `s` for menor que `t`, zero se elas forem iguais, e um número positivo se `s` for maior que `t`

```
1  #include <string.h>
2
3  int strcmp (char *s, char *t) {
4    int i;
5    unsigned char usi, uti;
6    for (i = 0; s[i] == t[i]; i++)
7      if (s[i] = '\0') return 0;
8    usi = s[i]; uti = t[i];
9    return usi - uti;
10 }
11
12 char s[10];
13 strcpy (s, "ABC");
14 printf ("%s", s);
```

Função `printf`

`int printf(const char *s, ...)`

Exibe argumentos formatados de acordo com `s`

string s contém formatação: %d indica inteiro, %s string, %c char, %f double (há outras possibilidades; ver Wikipedia)

```
1  #include <stdio.h>
2  int i; char *s = "Ola'";
3  for (i = 0; i < 5; i++) printf ("i = %d ", i);
4  printf("\n");
5  printf("%s, mundo!", s);
6
```

```
i = 0  i = 1  i = 2  i = 3  i = 4
Ola, mundo!
```

Scanf

int `scanf` (const char *s, ...)
Lê caracteres que satisfazem o formato em `s` e os armazena nas variáveis listadas; deixa na entrada os demais caracteres não lidos

- variáveis são passadas por referência (ponteiros)
- string `s` pode conter espaços, *tabs*, caracteres não brancos, e conversões `% [*] [largura] [mod] conversão`, onde conversão pode ser `c`, `d`, `f`, `s`, ...

```
1 int idade;
2 printf("Idade? "); scanf("%d", &idade);
3 printf ("Você tem %d anos", idade);
```

Scanf

int `scanf` (const char *s, ...)
Lê caracteres que satisfazem o formato em `s` e os armazena nas variáveis listadas; deixa na entrada os demais caracteres não lidos

Atenção:

```
1  scanf("Idade: %d", &idade);
```

espera entrada contendo `Idade:` seguida de um número inteiro – não exibe mensagem `Idade:` e depois coleta inteiro!

Scanf

int `scanf` (const char *s, ...)

Lê caracteres que satisfazem o formato em `s` e os armazena nas variáveis listadas; deixa na entrada os demais caracteres não lidos

- `%` `[*]` `[largura]` `[mod]` `conversão`, onde conversão pode ser `c`, `d`, `f`, `s`, ...
- Conversão termina quando um espaço é encontrado (`' '`, `'\t'`, `'\n'`), quando o padrão não é mais satisfeito ou quando a largura é excedida (o que acontecer primeiro).

```
1 /* nome.c */
2  char nome[100];
3  printf("Nome completo: ");
4  scanf("%99s", nome); /* Nome e sobrenome */
5  printf ("Ola, %s!", nome); /* primeiro nome */
```

Scanf

int `scanf` (const char *s, ...)
Lê caracteres de acordo com formato em `s`; devolve número de campos lidos e atribuídos

```
1 /* data.c */
2  int dia, mes, ano;
3
4  printf("Que dia é hoje? [dia/mês/ano] ");
5
6  scanf ("%2d/%2d/%4d", &dia, &mes, &ano);
7
8  printf ("Hoje é %d/%d/%d\n", dia, mes, ano);
```

Scanf

int `scanf` (const char *s, ...)
Lê caracteres de acordo com formato em `s`; devolve número de campos lidos e atribuídos

```
1 /* data2.c */
2  int dia, mes, ano; char sep;
3
4  printf("Que dia é hoje? [dia/mês/ano] ");
5
6  scanf ("%2d%c%2d/%4d", &dia, &sep, &mes, &ano);
7
8  printf ("Hoje é %d%c%d%c%d\n", dia, sep, mes, sep, ano);
```

Qual a saída?

```
1  int valor=0; char unidade='\0';
2
3  scanf ("%d%c", &valor, &unidade); /* Entrada: M100M */
4
5  printf ("%d%c", valor, unidade);  /* Saida ? */
```

Qual a saída?

```
1  int valor=0; char unidade='\0';
2
3  scanf ("%d%c", &valor, &unidade); /* Entrada: M100M */
4
5  printf ("%d%c", valor, unidade);  /* Saida ? */
```

0

Qual a saída?

```
1  int valor=0; char unidade='\0';
2
3  scanf ("%d%c", &valor, &unidade); /* Entrada: 100M */
4
5  printf ("%d%c", valor, unidade);  /* Saida ? */
```

Qual a saída?

```
1  int valor=0; char unidade='\0';
2
3  scanf ("%d%c", &valor, &unidade); /* Entrada: 100M */
4
5  printf ("%d%c", valor, unidade);  /* Saida ? */
```

100M

Qual a saída?

```c
/* strings.c */
char nome[100], sobrenome[100];

printf ("Nome:");
scanf ("%s", nome); /* Denis */
printf ("Sobrenome:");
scanf ("%s", sobrenome); /* Deratani Maua */
printf ("%s, %s\n", sobrenome, nome); /* Saida ? */
```

Qual a saída?

```c
/* strings.c */
char nome[100], sobrenome[100];

printf ("Nome:");
scanf ("%s", nome); /* Denis */
printf ("Sobrenome:");
scanf ("%s", sobrenome); /* Deratani Maua */
printf ("%s, %s\n", sobrenome, nome); /* Saida ? */
```

```
Deratani, Denis
```

Qual o problema?

```c
/* numero.c */
int n;
printf ("Digite um numero, por favor: ");
/* ler entrada até que algum número seja digitado */
while (scanf("%d", &n) == 0) {
  printf ("Você deve digitar um número: ");
}
printf ("Número: %d\n", n);
```

Entrada e saída

Um arquivo é uma sequência de dados salva em memória secundária (disco rígido, usb drive etc); diferentemente dos dados na memória (primária), os dados num arquivo só podem ser acessados sequencialmente

arquivos são tratados pelo tipo de dados `FILE` definido em `stdio.h`:

```
1 typedef struct {
2     int             _cnt;  /* num. caracteres disponíveis */
3     unsigned char *_ptr;   /* proximo caractere */
4     unsigned char *_base;  /* estoque de caracteres */
5     unsigned char  _flag;  /* estado do stream */
6     unsigned char  _file;  /* descritor no sistema Unix */
7 } FILE;
```

FILE *`fopen` (char *s, char* m)
Abre arquivo de dados de nome `s` no modo `m` e retorna ponteiro para descritor; use modo “r” para leitura, “w” para escrita (ou combinações)
int `fclose` (FILE *f)
Fecha arquivo

```
1 #include <stdio.h>
2 int main (void) {
3  FILE *entrada = fopen ("dados.txt", "r");
4  if (entrada == NULL) {
5     printf ("Erro ao abrir arquivo 'dados.txt'\n");
6     return 1;
7  }
8  fclose (entrada);
9 }
```

FILE *`fscanf` (FILE* f, char *s, ...)

Lê dados do arquivo `f` de acordo com `s`; retorna número de variáveis lidas

```
1  #include <stdio.h>
2  int main (void) {
3    int k, x, soma=0;
4    FILE *entrada = fopen ("dados.txt", "r");
5    while (1) {
6      k = fscanf (entrada, "%d", &x);
7      if (k != 1) break;
8      soma += x;
9    }
10   printf ("soma: %d\n", soma);
11   fclose (entrada);
12   return 0;
13 }
```

FILE *`fprintf` (FILE* f, char *s, ...)

Grava dados no arquivo `f` de acordo com `s`; retorna número de variáveis escritas

```
1 #include <stdio.h>
2 int main (void) {
3  int x,y,z;
4  FILE *saida = fopen ("dados.txt", "w");
5  scanf ("%d %d %d", &x, &y, &z);
6  fprintf (saida, "%d %d %d\n", x, y, z);
7  fclose (saida);
8  return 0;
9 }
```

Arquivos de texto

Um arquivo de texto é um arquivo contendo apenas caracteres e quebras de linha (símbolos representado quebras de linha).

- Em *nix, quebra de linhas são representadas por `\n`
- No Windows, quebra de linhas são representadas por `\r\n`
- No Mac OS, costumava usar `\r` para quebras de linha, mas recentemente mudou para `\n`

O símbolo `\r` em geral não é exibido em terminais linux; portanto uma maneira de produzir quebras de linha de maneira agnóstica ao sistema operacional é

```
1 printf ("Para todos os gostos\r\n");
```

Entrada e saída padrões

O teclado (descritor `stdin`) é o arquivo de entrada padrão e está sempre aberto; o terminal (descritor `stdout`) é o arquivo de saída padrão e está sempre aberto

```c
#include <stdio.h>
int main (void) {
  int x, y;
  fscanf (stdin, "%d %d", &x, &y);
  fprintf (stdout, "%d + %d = %d\n", x, y, x+y);
  return 0;
}
```

`int putc(int char, FILE *f)`

Escreve um caractere num arquivo e incrementa a posição atual; retorna o caractere escrito ou `EOF` se algum erro ocorrer

`int getc(FILE *f)`

Lê um caractere de um arquivo e incrementa a posição atual; retorna o caractere lido ou `EOF` se não houver mais dados no arquivo

```
1 #include <stdio.h>
2 int main (void) {
3   int c = 0;
4   FILE *entrada = fopen ("dados.txt", "r");
5   FILE *saida = fopen ("dados.copia.txt", "w");
6   while ((c = getc (entrada)) != EOF) putc (c, saida);
7   fclose (entrada); fclose (saida);
8 }
```

Entrada e saída de strings

`char *fgets (char *str, int n, FILE *f);`

Lê uma linha de no máximo `n` caracteres do arquivo especificado e armazena dados em `str`; retorna o valor de `str` ou EOF

```
1 char nome[100];
2 printf ("Nome completo: ");
3 fgets (nome, 100, stdin);
4 printf("Ola, %s", nome);
```

Qual a saída?

```
1 /* strings2.c */
2 char nome[100], sobrenome[100];
3 printf ("Nome: ");
4 fgets (nome, 100, stdin); /* Denis */
5 printf ("Sobrenome: ");
6 fgets (sobrenome, 100, stdin); /* Deratani Maua */
7 printf ("%s, %s\n", sobrenome, nome);
```

Qual a saída?

```c
/* strings2.c */
char nome[100], sobrenome[100];
printf ("Nome: ");
fgets (nome, 100, stdin); /* Denis */
printf ("Sobrenome: ");
fgets (sobrenome, 100, stdin); /* Deratani Maua */
printf ("%s, %s\n", sobrenome, nome);
```

```
Deratani Maua
, Denis
```

Qual a saída?

```c
/* strings3.c */
char nome[100], sobrenome[100];
int n;
printf ("Nome: ");
fgets (nome, 100, stdin); /* Denis */
n = strlen (nome); nome[n-1] = '\0';
printf ("Sobrenome: ");
fgets (sobrenome, 100, stdin); /* Deratani Maua */
n = strlen (sobrenome); sobrenome[n-1] = '\0';
printf ("%s, %s\n", sobrenome, nome);
```

Qual a saída?

```c
/* strings3.c */
char nome[100], sobrenome[100];
int n;
printf ("Nome: ");
fgets (nome, 100, stdin); /* Denis */
n = strlen (nome); nome[n-1] = '\0';
printf ("Sobrenome: ");
fgets (sobrenome, 100, stdin); /* Deratani Maua */
n = strlen (sobrenome); sobrenome[n-1] = '\0';
printf ("%s, %s\n", sobrenome, nome);
```

```
Deratani Maua, Denis
```

Para casa

Exercícios 4A-4C