```
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;
```

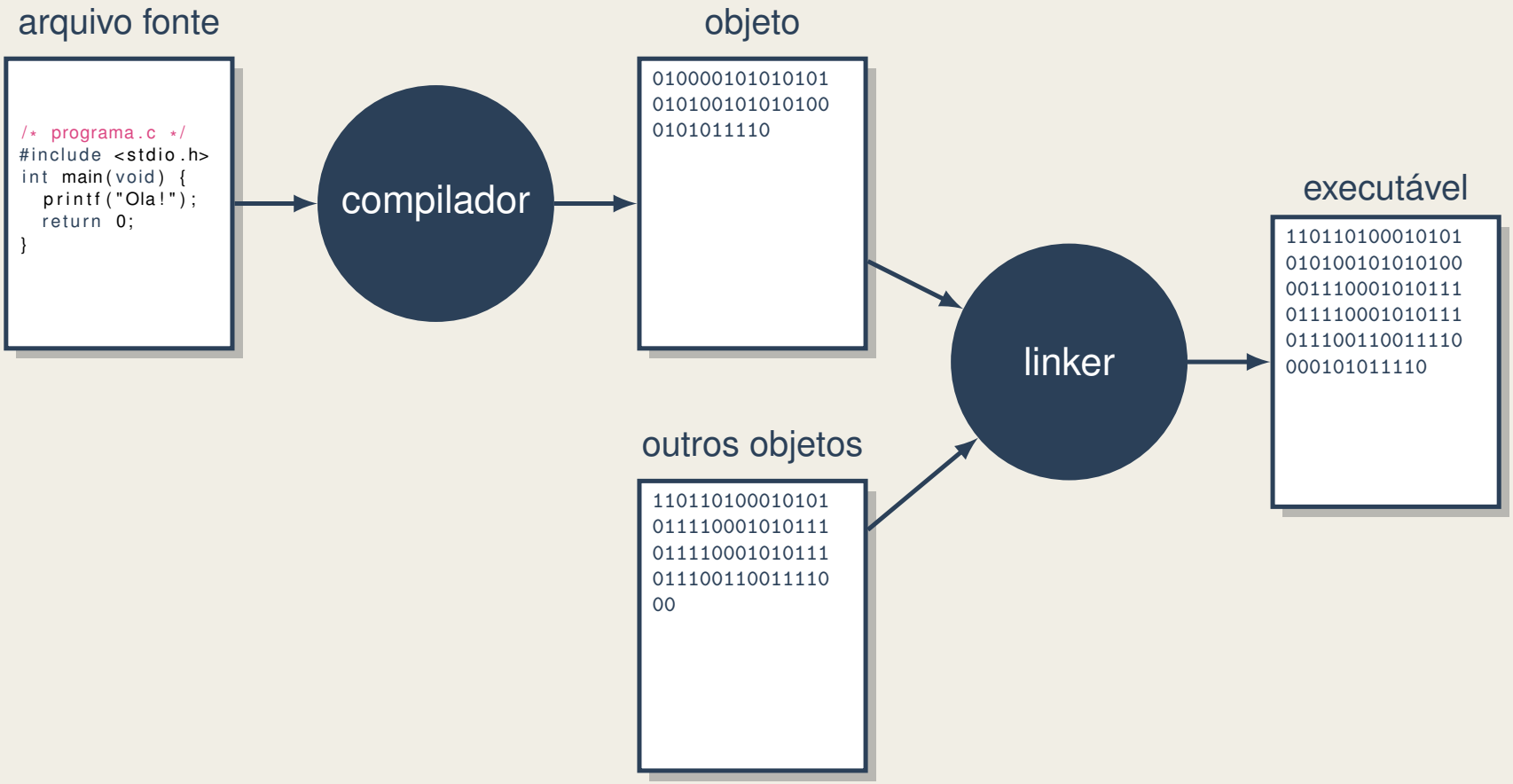
MAC122 - PRINCÍPIOS DE DESENVOLVIMENTO DE ALGORITMOS

Introdução à linguagem C

Material didático suplementar

- Site Projetos de Algoritmos (Prof. Feofiloff):
  `http://www.ime.usp.br/~pf/algoritmos/`
- PANDA IME: `https://panda.ime.usp.br/panda/c`
- The C Book:
  `http://publications.gbdirect.co.uk/c_book/`
- Programar em C:
  `https://pt.wikibooks.org/wiki/Programar_em_C`

Processo de compilação



Anatomia de um programa C

```c
/* Este programa imprime uma lista
 * de numeros e seus quadrados
 */
#include <stdio.h>

#define N 10

/* Recebe um inteiro n e devolve seu
 * quadrado
 */
int quad(int n);

int main(int argc, char *argv[]) {
  int i;
  for (i=0; i < N; i++) {
     printf("%d %d\n", i, quad(i));
  }
  return 0;
}

int quad(int n) {
  return n*n;
}
```

descrição do programa

inclusões de bibliotecas

constantes

protótipos de funções

função principal

definições de funções

Compiladores e IDEs

Compiladores:

- *nix e Mac OS X: gcc
- Windows: gcc, LCC

Ambientes de desenvolvimento:

- VSCode: `https://code.visualstudio.com`
- Editores de texto comum (Notepad), Sublime
- Emacs, vim
- Dev-C++, DJGPP, Eclipse, Xcode, Visual C++
- Code::Blocks

O padrão C

- Existem muitos padrões de linguagem C
- Eles concordam na maior parte, mas diferem em alguns pontos
- Vamos adotar o padrão ISO C99: `https://en.wikipedia.org/wiki/C99`
  - Versão mais atual (compatível com C99) é a C17 de 2017.
- No gcc compile com `-std=c99` (EPs serão corrigidos com essa *flag*!)

Palavras reservadas

não podem ser utilizadas para nenhuma outra função:

```
1  auto
2  break
3  case
4  char
5  const
6  continue
7  default
8  do
9  double
10 else
11 enum
12 extern
13 float
14 for
15 goto
16 if
17 int
18 long
19 register
20 return
21 short
22 signed
23 sizeof
24 static
25 struct
26 switch
27 typedef
28 union
29 unsigned
30 void
31 volatile
32 while
```

Programa mínimo

```c
int main(void)
{
    /* retorno indica que programa terminou conforme esperado */
    return 0;
}
```

Programa básico

```c
#include <stdio.h>

int main(void)
{
    /* exibe mensagem na tela */
    printf("Um programa basico.\n");
    return 0;
}
```

Programa básico

Compilar/Executar programa no e-disciplinas e linha de comando

Em *nix:

```
gcc -std=c99 -o programa.exe programa.c
. programa.exe
```

Tipos de dados nativos

Tipo de dados: Conjunto de valores equipado com operações

tipo	objeto representado	tamanho
`char`	palavra com sinal	1
`unsigned char`	palavra sem sinal	1
`int`	inteiros	4
`unsigned int`	inteiros não-negativos (naturais)	4
`float`	reais	4
`double`	reais	8
`long double`	reais	16

Para saber tamanho (em bytes): `sizeof(tipo)`

Tipo	Valores
`char`	-128 a +127
`unsigned char`	0 a 255
`int`	`-INT_MAX` a `INT_MAX-1`
`unsigned int`	0 a `INT_MAX`

O valor de `INT_MAX` depende da arquitetura do sistema e do compilador e é definido na biblioteca `limits.h`

Declaração de variáveis

`tipo nome1, nome2, ...;`

Aloca espaço em memória para uma ou mais variavéis de um mesmo determinado tipo e identifica o espaço pelo(s) nome(s) dado(s)

Valor da variável é indefinido

```
1 int i;
2 double x;
3 int i, j;
4 float z, y, w;
5 char c;
```

Expressões

Combinação de operadores e operandos: constantes, variáveis, atribuições, operadores artiméticos e lógicos, parênteses, vírgula e funções

Toda expressão em C possui um valor

```
1  s = 2 + 3;
2  2 * sqrt(4);
3  x > y + 3;
4  a < b && b < c;
```

Atribuição

`variavel = expressao;`

Define o valor da variável

Valor da expressão é o valor atribuído

```
1  int i; /* declara                */
2  i = 1; /*   e depois define */
3  int j = 0; /* declara e define */
4  x = y;
5  s = 2 + 3;
6  z = sqrt(4);
7  q = (q = 3);
8  a = b = c = d = 1;
```

Operações aritméticas

```
+ - * / %
```

```
long double op double = long double
double op float = double
float op int = float
int op unsigned int = int
unsigned int op char = int
char op unsigned char = int
```

Qual a saída?

```c
int i = 1 / 3 * 100;
printf("%d", i);
```

Qual a saída?

```c
int i = 1 / 3 * 100;
printf("%d", i);
```

0

Qual a saída?

```c
int i = 1 / 3 * 100.0;
printf("%d", i);
```

Qual a saída?

```c
int i = 1 / 3 * 100.0;
printf("%d", i);
```

0

Qual a saída?

```c
int i = 1 / 3.0 * 100;
printf("%d", i);
```

Qual a saída?

```c
int i = 1 / 3.0 * 100;
printf("%d", i);
```

33

Qual a saída?

```c
int i = 100 / 3;
printf("%d", i);
```

Qual a saída?

```c
int i = 100 / 3;
printf("%d", i);
```

33

Qual a saída?

```
1 unsigned char i = 255;
2 i = i + 1;
3 printf("%d", i);
```

Qual a saída?

```
1 unsigned char i = 255;
2 i = i + 1;
3 printf("%d", i);
```

0

Qual a saída?

```
1 char i = 127;
2 i = i + 1;
3 printf("%d", i);
```

Qual a saída?

```c
char i = 127;
i = i + 1;
printf("%d", i);
```

```
-128
```

Qual a saída?

```
1 unsigned i, j;
2 i = 2; j = 3;
3 printf("%d", i-j);
```

Qual a saída?

```
1 unsigned i, j;
2 i = 2; j = 3;
3 printf("%d", i-j);
```

```
-1
```

Comparadores

`<, <=, >, >=, ==, !=`

Retornam 1 ou 0

```
1 /* j = 99, a = 10, b = 20 */
2 j != 99; /* 0 */
3 a < b;   /* 1 */
4 a >= b;  /* 0 */
5 a == b;  /* 0 */
6 a = (a < b); /* a = 1 */
```

Comparações

`<, <=, >, >=, ==, !=`

Retornam 1 ou 0

```
long double comp double = long double
double comp float = double
float comp int = float
unsigned int comp int = unsigned int
int comp char = int
char comp unsigned char = int
```

Qual o problema?

```
1 /* exibe numeros de 0 a 255 */
2 unsigned char i;
3 for (i=0; i < 256; i++)
4   printf("%d ", i);
```

Operadores lógicos

`&&, ||, !`

Operam sobre expressões lógicas:

- Conjunção: `A && B` é 1 se `A` e `B` são verdadeiros e 0 c.c.
- Disjunção: `A || B` é 1 se `A` ou `B` é verdadeiro (ou ambos) e 0 c.c.
- Negação: `!A` é verdadeiro se `A` é falso

```
1 /* j = 99, a = 10, b = 20 */
2 j != 99 && a < b; /* 0 */
3 j == 99 && a < b; /* 1 */
4 a = (j != 99 || a < b) /* a = 1 */
5 !(a == b); /* 1 */
```

Expressões lógicas

`&&, ||, !`

Expressões aritméticas são convertidas em expressões lógicas assumindo-se NÃO 0 (diferente de 0) como verdadeiro e zero (0) como falso

```
1 /* j = 99, a = 10, b = 20 */
2 (j+1) && a < b; /* 1 */
3 (j-99) && a < b; /* 0 */
```

Curto-circuito: expressões são analisadas da esquerda para a direita e somente se necessário

```
1 j < 100 && v[j] > 0; /* seguro   */
2 v[j] > 0 && j < 100; /* inseguro */
```

Precedência de operadores

- Multiplicação e divisão: `a * b / c`
- Módulo (resto da divisão): `a % 2`
- Soma e subtração: `a + b - c`
- Comparação ordinal: `a < b, a >= b`
- Igualdade/desigualdade: `a == b, a != b`
- Conjunção e disjunção: `a && b, a || b`
- Atribuição: `a = 2, b = a`

Estruturas de bloco

Um bloco de comandos é um pedaço de código contido entre os símbolos { e }:

```
1   {
2       int a = 1, b = 2;
3       {
4           int b = 3;
5           int c = a + b; /* c = 4 */
6       }
7   }
8
```

- variáveis devem ser declaradas no início de um bloco
- blocos podem ser aninhados; o bloco interno tem acesso às variáveis dos blocos externos mas não o contrário
- variáveis homônimas podem existir em blocos (aninhados) distintos

Estruturas condicionais

`if (cond) bloco1 [else bloco]`

Executa primeiro bloco se expressão `cond` é diferente de zero (caso contrário executa segundo bloco)

```
1 int a, b = 1;
2 if (b > 0) { a = 2; }
3 else { a = 1; }
```

`while (cond) bloco`

Executa blocos enquanto a condição for verdadeira

```
1 int a = 0, b = 0;
2 while (a < 8) {
3   a += 2; b++;
4 }
```

a=8, b=4

Estruturas repetitivas

`for (init; cond; incr) bloco`

Executa `init` uma vez, depois executa `bloco` seguido por `incr` enquanto cond for verdadeira

```
1 int a, b=0;
2 for (a = 0; a < 8; a+=2) {
3   b++;
4 }
```

`a=8, b=4`

Variáveis dinâmicas

são criadas toda vez que bloco é executado e destruídas quando bloco é deixado

```
1 int a;
2 for (a = 0; a < 8; a+=2) {
3   int b = 0;
4   b++;
5 }
```

a=8, b=1

Variáveis estáticas

são criadas na primeira execução do bloco e mantêm seus valores nas execuções seguintes

```
1 int a;
2 for (a = 0; a < 8; a+=2) {
3   static int b = 0;
4   b++;
5 }
```

a=8, b=5

Qual a saída?

```
1 int a = 0, b = 1;
2 if (a = b) {
3   printf ("iguais");
4 } else {
5   printf ("diferentes");
6 }
```

Qual a saída?

```
1 int a = 0, b = 1;
2 if (a = b) {
3  printf ("iguais");
4 } else {
5  printf ("diferentes");
6 }
```

```
iguais
```

Para casa

1. Acessa a página da disciplina
2. Revisar slides dessa aula
3. Resolver os exercícios 0, 1A-1C