

PLAYAPC

PLAYAPC

Biblioteca gráfica para programadores inexperientes

Sinayra Pascoal Cotts Moreira
Universidade de Brasília

Prof. Dr. José Carlos Loureiro Ralha
Universidade de Brasília

Prof. Dr. Alexandre Zaghetto,
Universidade de Brasília



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright ©2007 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department with the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data:

Survey Methodology / Robert M. Groves . . . [et al].
p. cm.—(Wiley series in survey methodology)
“Wiley-Interscience.”
Includes bibliographical references and index.
ISBN 0-471-48348-6 (pbk.)
1. Surveys—Methodology. 2. Social sciences—Research—Statistical methods. I. Groves, Robert M. II. Series.

HA31.2.S873 2007
001.4'33—dc22
Printed in the United States of America.

2004044064

10 9 8 7 6 5 4 3 2 1

À todos os alunos que
queiram fazer trabalhos
bonitinhos na primeira
matéria de computação
da UnB

CONTRIBUTORS

CONTENTS IN BRIEF

PART I ALGORITMOS SEQUENCIAIS, CONDICIONAIS E COM REPETIÇÕES

1 Algoritmos sequenciais	3
2 Algoritmos condicionais	11
3 Algoritmos com repetição	15

PART II ESTRUTURA DE DADOS N-DIMENSIONAIS HOMOGÊNEAS

4 Vetores	25
5 Matrizes	29

PART III SUBALGORITMOS

6 Subalgoritmos	37
------------------------	-----------

PART IV RECURSÃO

7 Recursão	47
-------------------	-----------

CONTEÚDO

List of Figures	xi
List of Tables	xiii
Preface	xv
Acknowledgments	xvii
Acronyms	xix
Glossary	xxi
List of Symbols	xxiii
Introduction	xxv
Sinayra Pascoal Cotts Moreira.	
References	xxvi

PART I ALGORITMOS SEQUENCIAIS, CONDICIONAIS E COM REPETIÇÕES

1 Algoritmos sequenciais	3
	ix

2	Algoritmos condicionais	11
----------	--------------------------------	-----------

3	Algoritmos com repetição	15
----------	---------------------------------	-----------

PART II ESTRUTURA DE DADOS N-DIMENSIONAIS HOMOGÊNEAS

4	Vetores	25
----------	----------------	-----------

5	Matrizes	29
----------	-----------------	-----------

PART III SUBALGORITMOS

6	Subalgoritmos	37
----------	----------------------	-----------

PART IV RECURSÃO

7	Recursão	47
----------	-----------------	-----------

LIST OF FIGURES

1.1	Plano Cartesiano de -100 à 100	4
1.2	Boneco Palito	6
1.3	Estrela de Davi	9
2.1	Determinação do quadrante de uma reta baseado no ângulo de inclinação dela	12
3.1	Duas espirais hiperbólicas girando	16
3.2	Carro se movendo da posição -100 até a posição 100	18
3.3	Moinho de vento	21
4.1	Gráfico do polinômio $-x^3$	26
5.1	Pulsar	30
6.1	Lançador Balístico	38
6.2	Jogo Snake	41
7.1	Solucionador da torre de Hanói	48
		xi

xii LIST OF FIGURES

7.2	Curva de Koch de ordem 3	52
7.3	Floco de neve	53
7.4	Curva de Sierpiski	55

LIST OF TABLES

3.1	Teclas reconhecidas pela playAPC	19
4.1	Valor de verTipo da função CriaGrafico	27
5.1	Valor de nome da função Pintar	34

PREFACE

This is an example preface. This is an example preface. This is an example preface. This is an example preface.

R. K. Watts

Durham, North Carolina
September, 2007

ACKNOWLEDGMENTS

From Dr. Jay Young, consultant from Silver Spring, Maryland, I received the initial push to even consider writing this book. Jay was a constant “peer reader” and very welcome advisor during this year-long process.

To all these wonderful people I owe a deep sense of gratitude especially now that this project has been completed.

G. T. S.

ACRONYMS

UnB	Universidade de Brasília
APC	Análise e Programação de Algoritmos

GLOSSARY

NormGibbs	Draw a sample from a posterior distribution of data with an unknown mean and variance using Gibbs sampling.
pNull	Test a one sided hypothesis from a numerically specified posterior CDF or from a sample from the posterior
sintegral	A numerical integration using Simpson's rule

SYMBOLS

- A Amplitude
- $\&$ Propositional logic symbol
- a Filter Coefficient
- \mathcal{B} Number of Beats

INTRODUCTION

Sinayra Pascoal Cotts Moreira.

Departamento de Ciência da Computação - UnB
Brasília, DF, Brasil

O índice de reprovação nas matérias iniciais do curso de Ciência da Computação da UnB tem crescido a cada semestre, bem como o índice de evasão. Apesar das tentativas de criar mais horários de plantão de dúvidas e maior disponibilidade dos monitores para essas disciplinas, o desinteresse se mantém. Visando aumentar o interesse dos alunos pelo curso, está sendo desenvolvida uma biblioteca gráfica 2D simplificada denominada playAPC. Para o discente, a playAPC deve ser usada para consolidar os conceitos aprendidos em Análise e Programação de Algoritmos (APC) através de modelagem gráfica. Dessa forma, os alunos podem interagir com outras disciplinas do curso de modo lúdico.

A playAPC foi desenvolvida utilizando a linguagem C++, a API OpenGL e a biblioteca GLFW 2.7. A API OpenGL deve ser suportada pela placa de vídeo presente no computador, sendo exigido a versão 1.3 no mínimo. O tutorial para instalação tanto da GLFW quanto da própria playAPC está disponível em detalhes no site Guia de Referência da playAPC ¹. Apesar da playAPC ter sido desenvolvida em C++, o seu uso é focado primariamente para alunos que estejam a programar em C, ou seja, não é necessário conhe-

¹<http://pt-br.playcb.wikia.com/wiki/Categoria:Instala%C3%A7%C3%A3o>

imento de C++ para utilizar a biblioteca, apenas utilizar a toolchain do g++ para compilar.

Neste livro, será disponibilizado uma série de exercícios usando da playAPC focando auxiliar os professores da Univerdade de Brasília (UnB) a desenvolverem novas práticas de laboratórios das turmas de APC.

REFERENCES

- [1] OpenGL SuperBible. Pearson Education Inc, 6 edition, 2014.
- [2] Marcus Geelnard and Camilla Berglund. GLFW - Reference guide, 2010. API version 2.7.
- [3] Brian W. Kernighan and Dennis M. Ritchie. The C Programming Language. 1989.
- [4] Stanley B. Lippman, Josés Lajoile, and Barbara Moo. C++ Primer. 2013.

PARTE I

ALGORITMOS SEQUENCIAIS, CONDICIONAIS E COM REPETIÇÕES

CAPÍTULO 1

ALGORITMOS SEQUENCIAIS

Resumo

Estrutura sequencial é um conjunto de instruções que serão executadas em sequência. A sequência de cada instrução deve ser seguida para a realização de uma tarefa.

Problemas

- 1.1. Exiba um plano cartesiano de -100 a 100 com espaçamento de 5 unidades.
- 1.2. Desenhe um boneco palito que utilize pelo menos uma vez as seguintes geometrias:
 - Círculo
 - Elipse
 - Retângulo

- Triângulo
- Quadrado

1.3. Exiba a estrela de Davi.

Soluções

Exercício 1.1

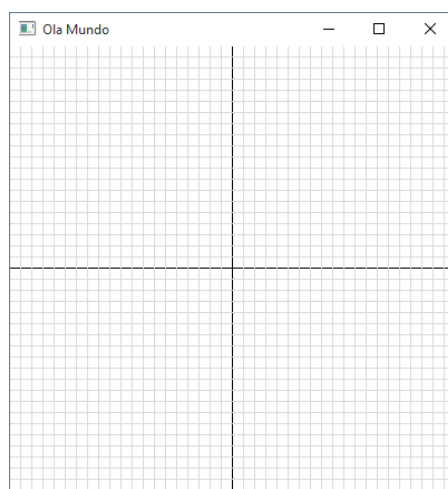


Figura 1.1 Plano Cartesiano de -100 à 100

Esta prática se refere a exibir um Plano Cartesiano na tela com espaçamento de 5 em 5 unidades, tanto no eixo x quanto no eixo y. Com ela, o aluno poderá notar a importância da ordem de chamada de funções da playAPC e a necessidade das funções `AbreJanela` e `Desenha`, além de verificar, com um exemplo simples, se a playAPC foi corretamente bem instalada.

Listagem 1.1 Código fonte de Plano Cartesiano

```
1 | #include <playAPC/playapc.h>
2 | int main() {
3 |
4 |     AbreJanela(400, 400, "Ola Mundo");
5 |
6 |     PintarFundo(255, 255, 255);
7 |     MostraPlanoCartesiano(5);
8 |
9 |     Desenha();
10 | }
```

```
void AbreJanela(float largura, float altura, const char* titulo)
```

A função `AbreJanela`, na linha 4, inicializa todas as variáveis utilizadas pela biblioteca, e preferencialmente é a chamada antes de qualquer outra função da `playAPC`. Por padrão, o plano de renderização está limitado de $(-100, 100)$ em coordenadas x, y do plano cartesiano. Este valor pode ser alterado utilizando a função `MostraPlanoCartesiano` antes de chamar `AbreJanela`. Seu primeiro argumento se refere a largura da janela, o segundo a altura, sendo ambos do tipo inteiro, e o terceiro se refere ao nome que a janela terá, sendo uma string.

```
void PintaFundo(int red, int green, int blue)
```

A função `PintaFundo`, na linha 6, é específica para pintar o fundo da janela de contexto aberto pela função `AbreJanela`. Seu argumentos utiliza o sistema de cores RGB (`red`, `green`, `blue`), utilizando a escala de 0 até 255.

```
void MostraPlanoCartesiano(int intervalo)
```

A função `MostraPlanoCartesiano`, na linha 7 exibe o plano de coordenadas cartesianas, plano utilizado para o posicionamento das geometrias criadas pela `playAPC`. Como as unidades no plano cartesiano não se referem ao posicionamento direto do pixel, a exibição do plano cartesiano com esta função serve de auxílio para o usuário posicionar suas geometrias na janela sem se preocupar com redimensionamento ou posição que a janela se encontra na tela do usuário. Para $x = 0$ e $y = 0$, as retas são pretas e as demais são cinza. Seu único argumento se refere de quantas em quantas unidades do plano terão uma reta vertical e horizontal da cor cinza.

```
void Desenha()
```

A função `Desenha`, na linha 9, realiza o loop de renderização. Todas as geometrias criadas até esta chamada de função serão renderizadas e permanecerão estáticas, não havendo a possibilidade de posteriores animações. Para encerrar o loop de renderização, basta fechar a janela clicando no botão de fechar ou apertando a tela `ESC`. Após fechar a janela, todo o contexto da `playAPC` será encerrado e as áreas de memórias alocadas serão liberadas.

Exercício 1.2

Esta prática se refere a exibir um boneco palito e praticar a grande maioria das geometrias pré-definidas existentes na `playAPC`. Os argumentos de cada função podem ser consultados no Guia de Referência da `playAPC` ¹

Listagem 1.2 Código fonte do boneco palito

```
1 | #include <playAPC/playapc.h>
2 |
3 | int main() {
```

¹<http://pt-br.playAPC{}.wikia.com/wiki/Categoria:Geometrias>

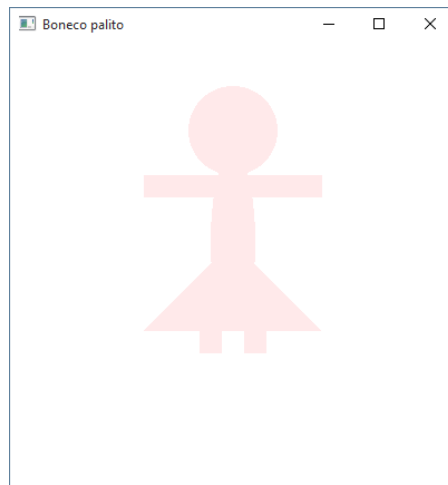


Figura 1.2 Boneco Palito

```

4      Ponto p;
5      AbreJanela(400, 400, "Boneco palito");
6      PintarFundo(255, 255, 255);
7
8      p.x = 0;
9      p.y = 60;
10     CriaCirculo(20, p); //(raio, ponto central)
11     Pintar(255, 233, 234);
12
13     p.y = 10;
14     CriaElipse(10, 40, p); //(metade do maior raio da elipse,
15     ↪ metade do menor raio da elipse, ponto central)
16     Pintar(255, 233, 234);
17
18     p.x = -40;
19     p.y = 30;
20     CriaRetangulo(80, 10, p); //(base, altura, ponto esquerdo
21     ↪ inferior)
22     Pintar(255, 233, 234);
23
24     p.x = -40;
25     p.y = -30;
26     CriaTriangulo(80, 40, p); //(base, altura, ponto esquerdo
27     ↪ inferior)
28     Pintar(255, 233, 234);
29
30     p.x = -15;
31     p.y = -40;
32     CriaQuadrado(10, p); //(lado, ponto esquerdo inferior)
33     Pintar(255, 233, 234);

```

```

34 |     CriaQuadrado(10, p);
35 |     Pintar(255, 233, 234);
36 |
37 |     Desenha();
38 | }

```

```

struct Ponto{
    float x;
    float y;
}

```

Ponto, na linha 4, é uma estrutura do tipo float com dois membros, x e y, os quais devem ser utilizados como coordenadas do plano cartesiano 2D. Esta estrutura possui sobrecarga para os seguintes operadores =, +, -, +=, -=, == e !=.

▪ =

```

1 |     Ponto p1, p2;
2 |     (...)
3 |
4 |     p1 = p2;

```

▪ + (ou -)

```

1 |     Ponto p1, p2, p3;
2 |     (...)
3 |
4 |     p1 = p2 + p3;

```

▪ += (ou -=)

```

1 |     Ponto p1, p2;
2 |     (...)
3 |
4 |     p1 += p2;

```

▪ == (ou !=)

```

1 |     Ponto p1, p2;
2 |     (...)
3 |
4 |     if(p1 == p2){
5 |         (...)
6 |     }

```

```

void Pintar(int red, int green, int blue);
void Pintar(int red, int green, int blue, geometrias_validas nome, int index);

```

A função Pintar, na linha 11 pode ser utilizada de duas formas. No caso da Listagem 1.2, a última geometria criada receberá a cor definida por esta função, utilizando o sistema de cores RGB. A segunda forma de utilizar esta função está ilustrada na Listagem 5.1.

```
int CriaCirculo(float raio, Ponto meio)
```

A função `CriaCirculo`, na linha 10, cria uma geometria do tipo `CIRCULO`, retornando um índice deste tipo de geometria. Seu primeiro argumento é o tamanho do raio e o segundo argumento é onde estará centrado o círculo.

```
int CriaElipse(float a, float b, Ponto meio)
```

A função `CriaElipse`, na linha 14, cria uma geometria do tipo `ELIPSE`, retornando um índice deste tipo de geometria. Seu primeiro argumento é a metade do maior eixo da elipse, o segundo é a metade do menor eixo da elipse e o terceiro argumento se refere onde a elipse estará centrada.

```
int CriaQuadrado(float lado, Ponto cantoesq)
```

A função `CriaQuadrado`, na linha 29, cria uma geometria do tipo `QUADRADO`, retornando um índice deste tipo de geometria. Seu primeiro argumento é o tamanho do lado do quadrado e o segundo argumento é onde ficará localizado o ponto esquerdo inferior da geometria

```
int CriaRetangulo(float base, float altura, Ponto cantoesq)
```

A função `CriaRetangulo`, na linha 19, cria uma geometria do tipo `RETANGULO`, retornando um índice deste tipo de geometria. Seu primeiro argumento é a base do retângulo, o segundo a altura não-negativa dele e o último é onde ficará localizado o ponto esquerdo inferior da geometria

```
int CriaTriangulo(float base, float altura, Ponto cantoesq)
```

A função `CriaTriangulo`, na linha 24, cria uma geometria do tipo `TRIANGULO`, retornando um índice deste tipo de geometria. Seu primeiro argumento é a base do triângulo, o segundo a altura não-negativa dele e o último é onde ficará localizado o ponto esquerdo inferior da geometria

Exercício 1.3

Esta prática se refere a exibir a estrela de Davi, feita com dois triângulos. Um triângulo foi criado com a função `CriaTriangulo` e o outro com a função `CriaPoligono`. Verificamos nesta prática os argumentos de `CriaTriangulo` (base, altura e ponto esquerdo inferior) e, como não há como ter altura negativa, teve a necessidade de criar um polígono definido pelos três pontos `p1`, `p2` e `p3` para criar-se um triângulo de cabeça pra baixo.

Listagem 1.3 Código fonte da Estrela de Davi

```
1 | #include <playAPC/playapc.h>
2 |
3 | int main(){
4 |     Ponto p1, p2, p3;
5 |     AbreJanela(400, 400, "Estrela de Davi");
6 |
7 |     p1.x = -25;
```

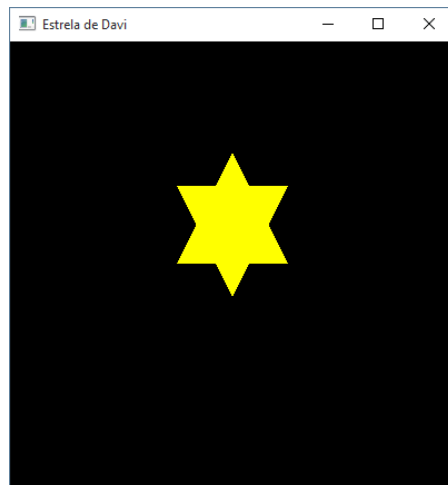


Figura 1.3 Estrela de Davi

```

8      p1.y = 0;
9      CriaTriangulo(50, 50, p1); //(base, altura)
10     Pintar(255, 255, 0);
11
12     p1.x = -25;
13     p1.y = 35;
14
15     p2.x = 25;
16     p2.y = 35;
17
18     p3.x = 0;
19     p3.y = -15;
20     CriaPoligono(3, p1, p2, p3); //(quantidade de pontos, p1, p2,
21     ↪ ...)
22     Pintar(255, 255, 0);
23     Desenha();
24 }

```

```
int CriaPoligono(short int qtd, ...)
```

A função `CriaPoligono`, na linha 20, cria uma geometria do tipo `POLIGONO`, retornando um índice deste tipo de geometria. Seu primeiro argumento é a quantidade de pontos que serão passados para esta função, e os seguintes argumentos serão os pontos propriamente ditos. Note que a `playAPC` é limitada no aspecto que esta função só consegue renderizar figuras convexas. Caso haja a necessidade de criação de figuras não-convexas, será necessário "quebrar" a geometria não-convexa em duas ou mais geometrias convexas.

CAPÍTULO 2

ALGORITMOS CONDICIONAIS

Resumo

Estrutura condicional expõe que a instrução ou bloco de instrução só seja executada se a condição for verdadeira.

Problemas

- 2.1. Escreva um programa que receba do usuário um valor de ângulo em graus, converta para radianos, exiba uma reta com comprimento de 50 unidades e pinte-a de acordo com as seguintes regras:
- Se a reta pertencer ao primeiro quadrante, pinte-a de vermelho
 - Se a reta pertencer ao segundo quadrante, pinte-a de verde
 - Se a reta pertencer ao terceiro quadrante, pinte-a de azul
 - Se a reta pertencer ao quarto quadrante, pinte-a de preto

Soluções

Exercício 2.1

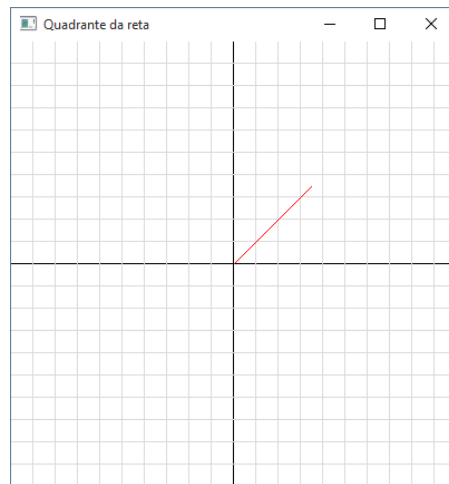


Figura 2.1 Determinação do quadrante de uma reta baseado no ângulo de inclinação dela

Esta prática exibe uma reta com cor variada de acordo com qual quadrante ela pertence. A função `Pintar` neste caso se refere a única geometria criada no programa, no caso, a reta.

Listagem 2.1 Código fonte do quadrante da reta

```

1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <math.h>
4
5  #define HIP 50
6  #define PI 3.14
7  int main(){
8      int angulo;
9      float anguloRad;
10     Ponto p1, p2;
11
12     printf("Digite um angulo de 0 a 360 graus:");
13     scanf("%d", &angulo);
14
15     p1.x = 0;
16     p2.x = 0;
17
18     anguloRad = (PI * angulo)/180;
19
20     p2.y = sin(anguloRad) * HIP;
21     p2.x = cos(anguloRad) * HIP;
22

```

```

23 | AbreJanela(400, 400, "Quadrante da reta");
24 | PintarFundo(255, 255, 255);
25 | MostraPlanoCartesiano(10);
26 |
27 | CriaReta(p1, p2);
28 |
29 | if(p2.x > 0){
30 |     if(p2.y > 0)
31 |         Pintar(255, 0, 0); //vermelho: 1 quadrante
32 |     else
33 |         Pintar(0, 0, 0); //preto: 4 quadrante
34 | }
35 | else{
36 |     if(p2.y > 0)
37 |         Pintar(0, 255, 0); //verde: 2 quadrante
38 |     else
39 |         Pintar(0, 0, 255); //azul: 3 quadrante
40 | }
41 |
42 | Desenha();
43 |
44 | }

```

```
int CriaReta(Ponto p1, Ponto p2)
```

A função CriaReta, na linha 27, cria uma geometria do tipo RETA, retornando um índice deste tipo de geometria. Seu primeiro e segundo argumento são duas variáveis do tipo Ponto.

CAPÍTULO 3

ALGORITMOS COM REPETIÇÃO

Resumo

Estruturas de repetição são criadas para que diversas instruções sejam executadas um determinado número de vezes, enquanto a condição se manter verdadeira.

Problemas

3.1. Sabendo que a equação hiperbólica pode ser definida por

$$\begin{aligned}x &= a \cos(\theta) \\ y &= a \sin(\theta)\end{aligned}$$

onde a é a assíntota para y e θ o ângulo equivalente ao ângulo em coordenadas polares, exiba duas espirais hiperbólicas, onde uma delas está invertida em relação a outra e coloque-as para girar.

3.2. Exiba um carrinho se movendo de -100 à 100 .

3.3. Construa um moinho de vento e coloque apenas as hélices para girar.

Soluções

Exercício 3.1

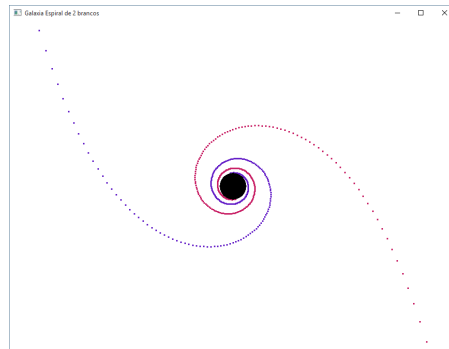


Figura 3.1 Duas espirais hiperbólicas girando

Esta prática ilustra como a função `Desenha1Frame` pode ser utilizada. Na linha 23 até a linha 36, a cada iteração são criados dois pontos, um de cada espiral.

Listagem 3.1 Código fonte da galáxia expiral

```
1  #include <playAPC/playapc.h>
2  #include <math.h>
3
4  int main (int argc, char * argv[]) {
5
6      AbreJanela (960,960, "Galaxia Espiral de 2 brancos");
7      /*
8       * Espiral Hiperbólica: equação em coordenadas cartesianas
9       *   x = a*cos(t)/t
10      *   y = a*sin(t)/t
11
12      * a é a assintota para y (reta paralela ao eixo x)
13      * t equivalente ao angulo em coordenadas polares
14      */
15      Ponto p, q, r;
16
17      // for (double t = 0; t < 4*PI; t += .01){
18
19          /* espiral hiperbolica , caminhando do "fim" pro centro (0,0)
20          p.x = 100*cos(t)/t;
21          p.y = 100*sin(t)/t;
22          */
23      for (double t = 4*PI; t > 0 ; t -= .05)
24          p.x = 100*cos(t)/t;          q.x = -p.x;
```

```

25 | p.y = 100*sin(t)/t;          q.y = -p.y;
26 |
27 | CriaPonto (p);
28 | Pintar (200, 30, 100);
29 | Grafite(3);
30 |
31 | CriaPonto (q);
32 | Pintar (100, 30, 200);
33 | Grafite(3);
34 |
35 | Desenha1Frame();
36 | }
37 |
38 |
39 | //A massive Black Hole in the very centre
40 | //If you want to see (the unseeable) black hole
41 | //paint the background on a different colour
42 | r.x =0;      r.y = 0;
43 | CriaCirculo(8, r);
44 | Pintar (0, 0, 0);
45 |
46 | for (double t=0; ; t += .5) {
47 |     Gira(t);
48 |     Desenha1Frame();
49 |
50 |     //Depois de um tempinho, pinta o fundo de branco pra mostrar
51 |     //o buraco negro
52 |     if ( t > 200 ) PintarFundo (255, 255, 255);
53 |
54 |     //quebra o loop e encerra o programa
55 |     if (ApertouTecla(GLFW_KEY_ENTER)) return 0;
56 | }
57 |
58 | }

```

`int CriaPonto(Ponto p)`

A função `CriaPonto`, na linha 27, cria uma geometria do tipo `PONTO`, retornando um índice deste tipo de geometria. Seu único argumento é uma variável do tipo `Ponto`. Uma geometria do tipo `PONTO` é renderizada como um pixel.

`void Grafite(int espessura)`

A função `Grafite`, na linha 29, aumenta as linhas de rasterização da última geometria criada, variando de 1 a ∞ . Por padrão, todas as geometrias começam com esta linha igual a 1. Esta função pode ser usada para deixar mais visível geometrias do tipo `PONTO`, que possuem 1 pixel de tamanho.

`int Desenha1Frame()`

A função `Desenha1Frame`, na linha 35, renderiza pelo menos $\frac{1}{60}$ segundos da cena, possuindo um controle de 60 frames por segundo. Caso o usuário feche a janela ou aperte a tecla `ESC`, esta função retornará 0 e encerrará o processo de renderização. Caso contrário, retornará 1.

```
void Gira(float theta)
void Gira(float theta, int index)
```

A função `Gira`, na linha 47, é uma das três funções de transformação implementadas na `playAPC`. Há duas formas de utilizar esta função. No caso da Listagem 3.1, seu argumento é um ângulo θ em graus e ele irá girar todas as geometrias criadas, recalculando a posição de cada pixel utilizando a Definição 1.

Definição 1 Seja x a coordenada do eixo x original do ponto, y a coordenada do eixo y original do ponto, x' a coordenada resultado do eixo, y' a coordenada resultante do eixo y e θ o ângulo em graus de rotação.

$$\begin{aligned}x' &= x \cos \theta - y \sin \theta \\y' &= x \sin \theta + y \cos \theta\end{aligned}$$

```
int ApertouTecla(int tecla)
```

A função `ApertouTecla`, na linha 55, verifica se o usuário pressionou a tecla `tecla` naquela cena. Seu único argumento pode variar de acordo com a Tabela 3.1.

Exercício 3.2

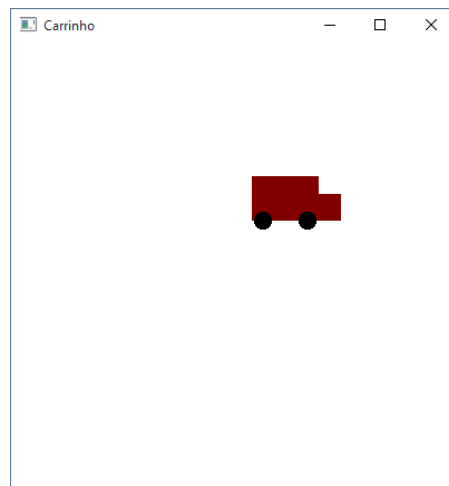


Figura 3.2 Carro se movendo da posição -100 até a posição 100

Esta prática exibe um carro construído com dois retângulos e dois círculos, agrupados com a função `CriaGrupo`, movendo-se da posição -100 até a posição

Tabela 3.1 Teclas reconhecidas pela playAPC

Valor	Descrição
GLFW_KEY_ n	Teclas alfanuméricas ($n \in (0..9)$ ou $n \in (A..Z)$)
GLFW_KEY_SPACE	Espaço
GLFW_KEY_ESC	Escape
GLFW_KEY_F n	Function key ($n \in (0..25)$)
GLFW_KEY_LEFT	Seta para esquerda
GLFW_KEY_UP	Seta para cima
GLFW_KEY_DOWN	Seta para baixo
GLFW_KEY_RIGHT	Seta para direita
GLFW_KEY_LCONTROL	Control esquerdo
GLFW_KEY_RCONTROL	Control direito
GLFW_KEY_LALT	Alt esquerdo
GLFW_KEY_RALT	Alt direito
GLFW_KEY_TAB	Tabulador
GLFW_KEY_ENTER	Enter
GLFW_KEY_BACKSPACE	Backspace
GLFW_KEY_INSERT	Insert
GLFW_KEY_DELETE	Delete
GLFW_KEY_PAGEUP	Page up
GLFW_KEY_PAGEDOWN	Page down
GLFW_KEY_HOME	Home
GLFW_KEY_END	End
GLFW_KEY_KP_ n	Teclas numéricas do keypad ($n \in (0..9)$)
GLFW_KEY_KP_DIVIDE	Tecla dividir do keypad (÷)
GLFW_KEY_KP_MULTIPLY	Tecla multiplicar do keypad (×)
GLFW_KEY_KP_SUBTRACT	Tecla subtrair do keypad (-)
GLFW_KEY_KP_ADD	Tecla adição do keypad (+)
GLFW_KEY_KP_EQUAL	Tecla igual do keypad (=)
GLFW_KEY_KP_NUMLOCK	Tecla Numlock do keypad (=)
GLFW_KEY_CAPS_LOCK	Caps lock
GLFW_KEY_SCROLL_LOCK	Scroll lock
GLFW_KEY_PAUSE	Pause
GLFW_KEY_MENU	Menu

100. Nota-se que todas as geometrias que estão abaixo da função `CriaGrupo` pertencem a um único grupo, o grupo carro.

Listagem 3.2 Código fonte do carro andando

```

1  #include <playAPC/playapc.h>
2
3  int main(){
4      Ponto p;
5      int carro;
6
7      AbreJanela(400, 400, "Carrinho");
8      PintarFundo(255, 255, 255);
9
10     carro = CriaGrupo();
11         p.x = -100;
12         p.y = 20;
13         CriaRetangulo(30, 20, p);
14         Pintar(128, 0, 0);
15
16         p.x = -80;
17         p.y = 20;
18         CriaRetangulo(20, 12, p);
19         Pintar(128, 0, 0);
20
21         p.x = -95;
22         p.y = 20;
23         CriaCirculo(4, p);
24         Pintar(0, 0, 0);
25
26         p.x = -75;
27         p.y = 20;
28         CriaCirculo(4, p);
29         Pintar(0, 0, 0);
30
31     p.y = 20;
32     for(p.x = -100; p.x < 100; p.x++){
33         Move(p, carro);
34         Desenha1Frame();
35     }
36     Desenha();
37 }
```

`int CriaGrupo()`

A função `CriaGrupo`, na linha 33, agrupará todo um conjunto de geometrias, associando todas a uma única variável, em um único conjunto. Desta forma, é possível transformar um conjunto de geometrias de forma independente, apenas referenciando a variável do grupo.

Exercício 3.3

Esta prática exibe um moinho de vento criado com um grupo composto por um triângulo e um retângulo, o grupo moinho, e outro grupo composto pelas hélices, o grupo grupo. Somente o grupo sofre a ação de girar.



Figura 3.3 Moinho de vento

Listagem 3.3 Código fonte do moinho

```

1  #include <playAPC/playapc.h>
2
3  int main (){
4      int angulo = 1;
5      int helices , moinho;
6      Ponto p1, p2;
7
8      AbreJanela(400, 400, "Moinho de Vento" );
9      PintarFundo(217,255,255);
10     //MostraPlanoCartesiano(10);
11
12     moinho = CriaGrupo();
13
14     p1.x = -20;
15     p1.y = -20;
16     CriaTriangulo(40,60,p1);
17     Pintar(255, 255, 0);
18
19     p1.x = -20;
20     p1.y = -100;
21     CriaRetangulo(40,80,p1);
22     Pintar(0, 255, 0);
23
24     helices = CriaGrupo();
25
26     p1.x = 0;
27     p1.y = 0;
28     // Hélice 1
29     p2.x = 0;
30     p2.y = 70;
31     CriaReta(p1, p2);

```



```
32     Pintar(255, 255, 255);
33     Grafite(8);
34
35     // Hélice 2
36     p2.x = 70;
37     p2.y = 0;
38     CriaReta(p1, p2);
39     Pintar(255, 255, 255);
40     Grafite(8);
41
42     // Helice 3
43     p2.x = 0;
44     p2.y = -70;
45     CriaReta(p1, p2);
46     Pintar(255, 255, 255);
47     Grafite(8);
48
49     // Helice 4
50     p1.x = -70;
51     p2.y = 0;
52     CriaReta(p1, p2);
53     Pintar(255, 255, 255);
54     Grafite(8);
55
56     p1.x = 0;
57     p1.y = 0;
58     CriaCirculo(5,p1);
59     Pintar(255,200,0);
60
61     while( angulo > 0){
62         DesenhaFrame();
63         Gira(angulo, helices);
64         angulo++;
65     }
66
67     Desenha();
68     return 0;
69 }
```

PARTE II

ESTRUTURA DE DADOS N-DIMENSIONAIS HOMOGÊNEAS

CAPÍTULO 4

VETORES

Resumo

Vetores são um tipo de estrutura que podem armazenar um tamanho fixo de elementos do mesmo tamanho e mesmo tipo, alocados em memória contígua. Utiliza-se vetores como um tipo de lista unidimensional, acessada através de índices.

Problemas

4.1. Exiba o gráfico do polinômio $-x^3$ para $-50 \leq x \leq 50$.

Soluções

Exercício 4.1

Esta prática mostra como construir um gráfico a partir de um vetor de Pontos. Cada posição em y de cada ponto é calculada dentro do loop. Por padrão,

playAPC, Primeira edição.
By Sinayra P.C. Moreira Copyright © 2016 John Wiley & Sons, Inc.

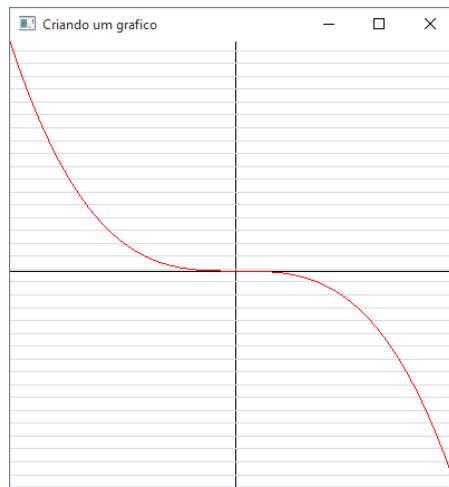


Figura 4.1 Gráfico do polinômio $-x^3$

os limites da janela de exibição da playCB vão de -100 à 100, entretanto, os valores em y nesta função variam de -125.000 até 125.000 , tendo a necessidade de mudar o limite de exibição com a função `MudaLimitesJanela(125000)`.

Listagem 4.1 Código fonte de polinômio

```

1  #include <playAPC/playapc.h>
2  #include <math.h>
3
4  int main(){
5      Ponto p[100];
6      int i, j;
7
8      MudaLimitesJanela(125000);
9
10     AbreJanela(400, 400, "Criando um grafico");
11     PintarFundo(255, 255, 255);
12     MostraPlanoCartesiano(7000);
13
14     j = -50;
15     for(i = 0; i < 100; i++, j++){
16         p[i].x = j;
17         p[i].y = -pow(p[i].x, 3);
18     }
19
20     CriaGrafico(100, p, 1);
21     Pintar(255, 0, 0);
22
23     Desenha();
24
25 }
```

```
void MudaLimitesJanela(int limite);
```

Tabela 4.1 Valor de verTipo da função CriaGrafico

Valor	Descrição
0	Não redimensiona a tela
1	Redimensiona a tela a fim de comportar toda a função dentro da janela de renderiza
2	Redimensiona a tela a fim de comportar toda a função dentro da janela de rendediz
sem interferir na escala de ambos os eixos	

A função MudaLimitesJanela, na linha 83, muda o limite da visualização do plano cartesiano, indo de valores $-limite$ até $limite$, tanto em x quanto em y . Esta função deve ser chamada antes da função AbreJanela.

```
int CriaGrafico(short int index, Ponto *p, short int verTipo);
```

A função CriaGrafico, na linha ??, cria uma geometria do tipo GRAFICO, retornando um índice deste tipo de geometria. A partir de um vetor de pontos p de tamanho $index$, esta função cria uma sequência de retas que ligam cada ponto definido pelo vetor. O seu terceiro argumento, $verTipo$, se refere o modo de visualização do gráfico, recebendo os parâmetros descritos na Tabela 4.1.

CAPÍTULO 5

MATRIZES

Resumo

Assim como vetores, matrizes são um tipo de estrutura que armazena dados de mesmo tamanho e mesmo tipo, mas são utilizadas de maneira n-dimensional. O modo mais comum de utilizar matriz é usando-a na forma bidimensional, onde os dados são tratados como se estivessem numa tabela, com linhas e colunas.

Problemas

- 5.1. Mostre graficamente o jogo da vida para uma matriz com 17 linhas e 17 colunas com a seguinte população inicial onde a população inicial estará VIVA para as seguintes posições na matriz:

(1, 5), (2, 5), (3, 5), (3, 6), (5, 1), (5, 2), (5, 3), (5, 6), (5, 7), (6, 3), (6, 5), (6, 7), (7, 5), (7, 6)

Soluções

Exercício 5.1

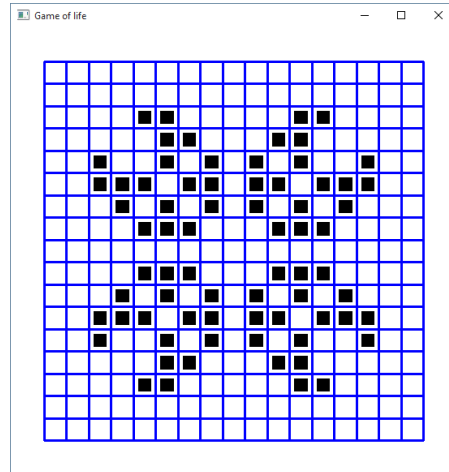


Figura 5.1 Pulsar

Esta prática mostra como utilizar o retorno da função `CriaQuadrado`, que esta retorna o índice da geometria criada. O seu índice é utilizado na função `Pintar`, que recebe, além do índice, o tipo da geometria. Como foi utilizado a função `CriaQuadrado`, o tipo de geometria é `QUADRADO`. Se fosse utilizado `CriaCirculo`, seria utilizado o tipo `CIRCULO` e assim sucessivamente.

Listagem 5.1 Código fonte do jogo da vida

```

1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #define LINHAS 17
6  #define COLUNAS 17
7  #define VIVO 1
8  #define MORTO 0
9
10 void waitFor (unsigned int );
11
12 int main(){
13
14     int geracaoAtual[LINHAS][COLUNAS], proximaGeracao[LINHAS][
15         ↳ COLUNAS], geoIndex[LINHAS][COLUNAS];
16     int l, c, i, j, soma, r, s, n, ESTADO, deltaQuad=10, largQuad =
17         ↳ 6, marg = 2;
18     Ponto p1, p2, p3;
19
20     //////////////////////////////////

```

```

19 // Define o Janela //
20 ///////////////////////////////////////////////////
21 AbreJanela(550, 550, "Game of life");
22 PintarFundo(255, 255, 255);
23
24 ///////////////////////////////////////////////////
25 // Desenha Grid //
26 ///////////////////////////////////////////////////
27
28 // Desenha as retas horizontais
29 p1.x = -85;
30 p1.y = 85;
31 p2.x = 85;
32 p2.y = 85;
33 for(i = 0; i < LINHAS+1; i++){
34   CriaReta(p1, p2); Pintar(0, 0, 255); Grafite(3); //cima
35   p1.y -= deltaQuad;
36   p2.y -= deltaQuad;
37 }
38 // Desenha as retas verticais
39 p1.x = -85;
40 p1.y = 85;
41 p2.x = -85;
42 p2.y = -85;
43 for(i = 0; i < COLUNAS+1; i++){
44   CriaReta(p1, p2); Pintar(0, 0, 255); Grafite(3); //cima
45   p1.x += deltaQuad;
46   p2.x += deltaQuad;
47 }
48
49 // Desenha celulas
50 p1.y = 85-deltaQuad+marg;
51 for(i = 0; i < LINHAS; i++){
52   p1.x = -85+marg;
53   for(j = 0; j < COLUNAS; j++){
54     geoIndex[i][j] = CriaQuadrado(largQuad, p1);
55     Pintar(0, 0, 0, QUADRADO, geoIndex[i][j]);
56     p1.x += deltaQuad;
57   }
58   p1.y -=deltaQuad;
59 }
60
61 ///////////////////////////////////////////////////
62 // Joga o jogo //
63 ///////////////////////////////////////////////////
64 // Preenche o fundo com o valor MORIO
65 for(l=0; l<LINHAS; l++){
66   for(c=0; c<COLUNAS; c++){
67     geracaoAtual[l][c]=MORIO;
68   }
69 }
70
71 // Define as celulas vivas no no canto superior esquerdo
72 geracaoAtual[1][5] = VIVO;
73 geracaoAtual[2][5] = VIVO;
74 geracaoAtual[3][5] = VIVO; geracaoAtual[3][6] = VIVO;

```

```

75     geracaoAtual[5][1] = VIVO; geracaoAtual[5][2] = VIVO;
       ↳ geracaoAtual[5][3] = VIVO; geracaoAtual[5][6] = VIVO;
       ↳ geracaoAtual[5][7] = VIVO;
76     geracaoAtual[6][3] = VIVO; geracaoAtual[6][5] = VIVO;
       ↳ geracaoAtual[6][7] = VIVO;
77     geracaoAtual[7][5] = VIVO; geracaoAtual[7][6] = VIVO;
78
79     // Realiza a reflexao do padrao
80     for (l = 0; l < LINHAS/2; l++){
81         for (c = 0; c < COLUNAS/2; c++){
82             geracaoAtual[(LINHAS-1)-l][c] = geracaoAtual[l][c]; //
83             ↳ Inferior esquerdo
84             geracaoAtual[l][(COLUNAS-1)-c] = geracaoAtual[l][c]; //
85             ↳ Superior direito
86             geracaoAtual[(LINHAS-1)-l][(COLUNAS-1)-c] =
87             ↳ geracaoAtual[l][c]; // Inferior direito
88         }
89     }
90
91     while(1){
92
93         // Faz a copia da geracao atual para a geracao anterior
94         for (l = 0; l < LINHAS; l++){
95             for (c = 0; c < COLUNAS; c++){
96                 proximaGeracao[l][c] = geracaoAtual[l][c];
97             }
98         }
99
100        for (l=0; l<LINHAS; l++){
101            for (c=0; c<COLUNAS; c++){
102                if (proximaGeracao[l][c]==1){
103                    printf("%c", 219);
104                    Pinta(0, 0, 0, QUADRADO, geoIndex[l][c]);
105                } else{
106                    printf("%c", proximaGeracao[l][c], ' ');
107                    Pinta(255, 255, 255, QUADRADO, geoIndex[l][c])
108                    ↳ ;
109                }
110            }
111            printf("\n");
112        }
113        DesenhaFrame();
114
115        // Conta a quantidade de vizinhos de uma celula
116        for (l = 1; l < LINHAS-1; l++){
117            for (c = 1; c < COLUNAS-1; c++){
118                ESTADO = proximaGeracao[l][c];
119                soma = 0;
120                for (r = l-1; r < l+2; r++){
121                    for (s = c-1; s < c+2; s++){
122                        soma+=proximaGeracao[r][s];
123                    }
124                }
125
126                if (proximaGeracao[l][c] == VIVO) soma--;
127
128                //Define se uma celula deve morrer, permanecer
129                ↳ viva ou nascer

```

```

123         if ((ESTADO == VIVO && soma < 2) || (ESTADO == VIVO
124             ↪ && soma > 3)){
125             geracaoAtual[1][c] = MORTO;
126         }else if ((ESTADO == VIVO && (soma > 2 || soma <=
127             ↪ 3)) || (ESTADO == MORTO && soma == 3)){
128             geracaoAtual[1][c] = VIVO;
129         }
130     }
131     waitFor (1);
132     system("cls");
133 }
134 Desenha();
135 return 0;
136 }
137
138 void waitFor (unsigned int secs) {
139     unsigned int retTime;
140     retTime = time(0) + secs;    // Get finishing time.
141     while (time(0) < retTime);  // Loop until it arrives.
142 }

```

```

void Pintar(int red, int green, int blue);
void Pintar(int red, int green, int blue, geometrias_validas nome, int index);

```

A função `Pintar`, na linha 55 pode ser utilizada de duas formas. No caso da Listagem 5.1, como cada índice do tipo `QUADRADO` está salvo na matriz `geoIndex`, na linha 54, este índice é passado para o quinto argumento da função `Pintar`. Desta forma, apenas aquela geometria com o valor daquele índice será pintada com as cores especificadas pelos valores `red`, `green` e `blue`. O quarto argumento é justamente o tipo de geometria, podendo variar de acordo com a função `Cria` utilizado. A outra forma de utilizar esta função está ilustrada na Listagem 1.2. De forma resumida, o argumento `nome`, do tipo `geometrias_validas`, pode receber os valores descritos na Tabela 5.1.

Tabela 5.1 Valor de nome da função Pintar

Valor	Retorno da função
CIRCULO	CriaCirculo
QUADRADO	CriaQuadrado
CIRCUNFERENCIA	CriaCircunferencia
RETANGULO	CriaRetangulo
ELIPSE	CriaElipse
GRAFICO	CriaGrafico
PONTO	CriaPonto
RETA	CriaReta
POLIGONO	CriaPoligono e CriaPoligonoVetor
TRIANGULO	CriaTriangulo

PARTE III

FUNÇÕES

CAPÍTULO 6

FUNÇÕES

Resumo

Uma função é um conjunto de instruções que, ao final da função, executa uma tarefa. Todo programa C possui pelo menos uma função, a `main`.

Problemas

- 6.1. Crie dois retângulos e posicione-os aleatoriamente em x . Coloque uma circunferência no topo do primeiro retângulo e receba do usuário dois valores: ângulo e velocidade. Dado estes valores, calcule e exiba a trajetória balística da circunferência sendo lançada para o outro retângulo. Exiba mensagem caso o usuário consiga acertar o prédio ou não e, em seguida, caso o usuário deseje jogar novamente, sorteie novamente a posição dos retângulos.
- 6.2. Crie o jogo Snake com as seguintes configurações
 - A cabeça não pode estar na mesma posição que o corpo

- A cabeça não pode estar na mesma posição que a parede

SUGESTÃO: Utilize a lógica do exercício 4.1

Soluções

Exercício 6.1

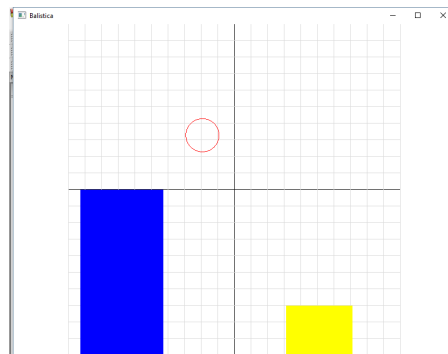


Figura 6.1 Lançador Balístico

Esta prática ilustra como a função `LimpaDesenho` é usada para poder re-desenhar outras cenas.

Listagem 6.1 Código fonte do lançador balístico

```

1  #include <playAPC/playapc.h>
2  #include <time.h>
3
4  int colisaoBolinhaRetangulo(int raio, int largurapredio, Ponto alvo
   ↪ , Ponto partida, Ponto movimentacao){
5
6      if( movimentacao.y - raio <= alvo.y
7         && movimentacao.y - raio > alvo.y - 1      //tolerancia
8         ↪ para pousar no eixo y
9         && movimentacao.x + raio < alvo.x + largurapredio
10        && movimentacao.x >= alvo.x
11        )
12        return 1;
13    else if(fabs(movimentacao.x + raio) > 100 || fabs(movimentacao.
14             ↪ y + raio) > 100)
15        return 2;
16    else
17        return 0;
18
19 }
20
21 int main(){
22     Ponto p, alvo, partida;

```

```

21 float theta, v0, t;
22 int colisao = 1, resposta, bolinha;
23
24 AbreJanela(800,600, "Balistica");
25 srand(time(NULL));
26 do{
27     MostraPlanoCartesiano(10);
28     PintarFundo(255, 255, 255);
29     ////////////////////////////////////Predio Azul
30     ↪ ////////////////////////////////////
31     CriaGrupo(); //Separa os grupos - este e o grupo dos
32     ↪ predios
33     if(colisao == 1) //Comeca uma nova fase
34         p.x = rand()%20 - 100; //-100 para comecar do lado
35         ↪ esquerdo da tela
36     else //Repete fase anterior
37         p.x = partida.x - 25;
38     p.y = 0;
39     CriaRetangulo(50, -100, p);
40     Pintar(0, 0, 255);
41
42     //
43     ↪ ////////////////////////////////////
44     ↪
45
46     ////////////////////////////////////Predio Amarelo
47     ↪ ////////////////////////////////////
48
49     if(colisao == 1) //Comeca uma nova fase
50         alvo.x = (p.x) + rand()%50 + 100;
51     else //Repete a fase anterior
52         alvo.x = alvo.x;
53     alvo.y = -70;
54     CriaRetangulo(40, -90, alvo);
55     Pintar(255, 255, 0);
56
57     //
58     ↪ ////////////////////////////////////
59     ↪
60
61     ////////////////////////////////////Bolinha
62     ↪ ////////////////////////////////////
63     bolinha = CriaGrupo(); //Separa os grupos - este e o grupo
64     ↪ da bolinha
65     partida.x = p.x + 25; //25 para estar no meio de um
66     ↪ predio de largura 50
67     partida.y = 10;
68     CriaCircunferencia(10, partida);
69     Pintar(255, 0, 0);
70
71     //
72     ↪ ////////////////////////////////////
73     ↪
74
75     Desenha1Frame();
76
77     printf("Angulo: ");
78     scanf("%f", &theta);
79     printf("\nVelocidade: ");

```

```

64     scanf("%f", &v0);
65
66     t = 0;
67     theta = theta * PI/180;
68     colisao = 0;
69     do{
70         if(!colisao){
71             p.x = partida.x + v0 * cos(theta) * t;
72             p.y = partida.y + v0 * sin(theta) * t - ((1/2.0) *
73                 ↪ (9.8) * (t*t));
74             Move(p, bolinha);
75             t += 0.01; //tempo
76         }
77         colisao = colisaoBolinhaRetangulo(10, 40, alvo, partida
78             ↪ , p);
79         Desenha1Frame();
80     }while(!colisao);
81     //printf("Tipo de colisao: %d\n", colisao);
82
83     LimpaDesenho(); //Limpo o desenho para começar uma nova
84     ↪ fase
85
86     if(colisao == 1)
87         printf("\nYay! Quer jogar um novo jogo?");
88     else
89         printf("\nOh nao... Quer tentar de novo?");
90     printf("\n1 - Sim\n0 - Nao\n");
91     scanf("%d", &resposta);
92
93     }while(resposta);
94
95     Desenha();
96
97     return 0;
98 }

```

```
void LimpaDesenho()
```

A função `LimpaDesenho`, na linha ??, destrói todas as geometrias e retorna toda a `playAPC` para o estado padrão, com exceção dos limites do plano cartesiano.

Exercício 6.2

Esta prática ilustra como a função `ApertaTecla` e `MudaLimitesJanela` podem ser utilizadas: a primeira para lidar com input de teclado ¹ e a segunda para ajustar o plano onde as geometrias serão desenhadas.

¹http://pt-br.playcb.wikia.com/wiki/Aperta_Tecla

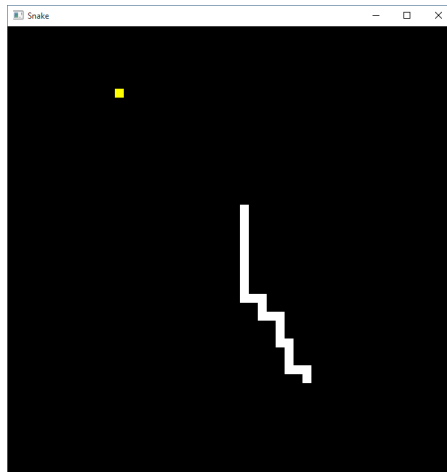


Figura 6.2 Jogo Snake

Listagem 6.2 Código fonte de Snake

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <playAPC/playapc.h>
4
5 #define ESQ GLFW_KEY_LEFT
6 #define DIR GLFW_KEY_RIGHT
7 #define CIMA GLFW_KEY_UP
8 #define BAIXO GLFW_KEY_DOWN
9
10 #define TAM 50
11
12 typedef enum{
13     CABECA,
14     CORPO,
15     COMIDA,
16     VAZIO
17 }tipoCobra;
18
19 typedef struct{
20     int direcao; //se tipo nao for vazio, indica direcao
21     tipoCobra tipo; //como sera pintado
22     int index; //indice do quadrado a ser pintado
23 }tipoCelula;
24
25 void inicializaMatriz(tipoCelula m[TAM][TAM], int pos_i, int pos_j)
26     ↪ {
27     Ponto p;
28
29     printf("Aguarde enquanto o jogo inicializa\n");
30
31     p.y = TAM/2 - 1;
32     for(int i = 0; i < TAM; i++){

```

```

32     p.x = -(TAM/2);
33     for(int j = 0; j < TAM; j++){
34         m[j][i].index = CriaQuadrado(1, p);
35         m[j][i].tipo = VAZIO;
36         //printf("m[%d][%d] = %d - P(%f,%f)\n", i, j, m[i][j].
           ↪ index, p.x, p.y);
37         Pintar(0, 0, 0);
38         p.x++;
39     }
40     p.y--;
41 }
42 Pintar(255, 255, 255, QUADRADO, m[pos_i][pos_j].index);
43 m[pos_i][pos_j].direcao = CIMA;
44 m[pos_i][pos_j].tipo = CABECA;
45 }
46
47 int bateu(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
48     if(m[pos_i][pos_j].tipo == CORPO || m[pos_i][pos_j].tipo ==
           ↪ CABECA){
49         printf("CORPO\n");
50         return 1;
51     }
52     else if(pos_i >= TAM || pos_j >= TAM || pos_i < 0 || pos_j < 0)
           ↪ {
53         printf("LIMITE DA TELA\n");
54         return 1;
55     }
56
57     return 0;
58 }
59 void atualizaPosicao(int direcao, int *npos_i, int *npos_j){
60     switch(direcao){
61         case ESQ:
62             (*npos_i)--;
63             break;
64         case DIR:
65             (*npos_i)++;
66             break;
67         case CIMA:
68             (*npos_j)--;
69             break;
70         case BAIXO:
71             (*npos_j)++;
72             break;
73     }
74 }
75
76 void updateTeclado(int *direcao, int *npos_i, int *npos_j){
77     if(ApertouTecla(ESQ) && *direcao != DIR)
78         (*direcao) = ESQ;
79
80     if(ApertouTecla(DIR) && *direcao != ESQ)
81         (*direcao) = DIR;
82
83     if(ApertouTecla(CIMA) && *direcao != BAIXO)
84         (*direcao) = CIMA;

```

```

85
86     if (ApertouTecla(BAIXO) && *direcao != CIMA)
87         (*direcao) = BAIXO;
88
89     atualizaPosicao((*direcao), npos_i, npos_j);
90 }
91
92 //retorna se comeu comida
93 int updateCabeca(tipoCelula m[TAM][TAM], int pos_i, int pos_j, int
94     ↪ direcao){
95     int comeu = 0;
96
97     Pinta(255, 255, 255, QUADRADO, m[pos_i][pos_j].index);
98     m[pos_i][pos_j].direcao = direcao;
99
100     if (m[pos_i][pos_j].tipo == COMIDA)
101         comeu = 1;
102
103     m[pos_i][pos_j].tipo = CABECA;
104
105     return comeu;
106 }
107
108 void updateRastro(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
109     Pinta(0, 0, 0, QUADRADO, m[pos_i][pos_j].index);
110     m[pos_i][pos_j].tipo = VAZIO;
111 }
112
113 void sorteiaComida(tipoCelula m[TAM][TAM]){
114     int pos_i, pos_j;
115     do{
116         pos_i = rand()%TAM;
117         pos_j = rand()%TAM;
118     }while(m[pos_i][pos_j].tipo != VAZIO);
119
120     Pinta(255, 255, 0, QUADRADO, m[pos_i][pos_j].index);
121
122     m[pos_i][pos_j].tipo = COMIDA;
123 }
124
125 int main(){
126     tipoCelula m[TAM][TAM]; //-100 a 100 pra cima e pra baixo, cada
127     ↪ quadrado
128     int pos_i = TAM/2; //posicao i da cabeca
129     int pos_j = TAM/2; //posicao j da cabeca
130
131     int rpos_i = pos_i; //posicao i do rabo
132     int rpos_j = pos_j; //posicao j do rabo
133
134     int direcao = CIMA;
135     MudaLimitesJanela(TAM/2);
136     AbreJanela(650, 650, "Snake");
137     PintaFundo(255, 0, 0);
138     //MostraPlanoCartesiano(5);

```

```

139     inicializaMatriz(m, pos_i, pos_j);
140
141     sorteiaComida(m);
142
143     while(1){
144         int npos_i = pos_i, npos_j = pos_j;
145
146         updateTeclado(&direcao, &npos_i, &npos_j);
147         m[pos_i][pos_j].direcao = direcao; //ultima posicao da
            ↪ cabeca recebe direcao que cabeca foi
148         if(!bateu(m, npos_i, npos_j)){
149             int comeu;
150
151             comeu = updateCabeca(m, npos_i, npos_j, direcao);
152             if(comeu)
153                 sorteiaComida(m);
154             else{
155                 updateRastro(m, rpos_i, rpos_j);
156                 atualizaPosicao(m[rpos_i][rpos_j].direcao, &rpos_i,
            ↪ &rpos_j);
157                 m[rpos_i][rpos_j].tipo = CORPO;
158             }
159
160             pos_i = npos_i;
161             pos_j = npos_j;
162
163             printf("C(%d, %d)\t R(%d, %d)\n", pos_i, pos_j, rpos_i,
            ↪ rpos_j);
164         }
165         else
166             break;
167
168         Desenha1Frame();
169     }
170
171     printf("O jogo acabou!\n");
172
173     Desenha();
174     return 0;
175 }

```

PARTE IV

RECURSÃO

CAPÍTULO 7

RECURSÃO

Resumo

Recursão é uma sub-rotina que pode invocar a si mesma, contendo, a cada chamada, uma pedaço menor da solução final.

Problemas

- 7.1. Exiba passo a passo a solução da Torre de Hanói para, no máximo, 5 discos.
- 7.2. Sabendo que a curva de Koch é construída que toma, como base, um segmento de reta de tamanho n . Na extremidade deste primeiro segmento, desenha-se outro segmento de reta de tamanho n , porém com uma curvatura de 60 em relação ao primeiro. Na extremidade deste segundo segmento, desenha-se outra curva de reta de tamanho n , mas agora com a curvatura de 120 em relação ao primeiro. Por fim, desenha-se outro segmento de reta de tamanho n na extremidade do terceiro segmento.

- (a) Exiba a curva de Koch de ordem n
 - (b) Exiba um floco de neve de ordem n baseado na curva de Koch
- 7.3. A curva de Sierpiński preenche todo o espaço disponível sem cortar suas próprias linhas, sendo mutuamente recursiva entre quatro funções. AQUI TEM UMA EXPLICAÇÃO DAS REGRAS DESSA CURVA. Exiba a curva de Sierpiński de ordem n .

Soluções

Exercício 7.1

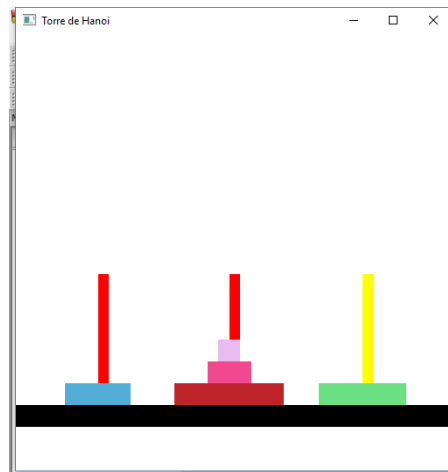


Figura 7.1 Solucionador da torre de Hanói

Esta prática ilustra como mover cada geometria utilizando retorno da função `CriaGrupo` para mover cada peça da torre de Hanoi para uma posição específica.

Listagem 7.1 Código fonte da torre de Hanói

```

1  #include <stdio.h>
2  #include <playAPC/playapc.h>
3
4  #define CHAO -70
5  #define POSX_A -62.5
6  #define POSX_B -2.5
7  #define POSX_C 58.5
8
9  typedef struct{
10     int index, largura, label;
11     float posicao;

```

```

12     char torre;
13 }discoHanoi;
14 //discoHanoi.index : se refere ao indice do grupo, retornado pela
15     ↪ funcao CriaGrupo
16 //discoHanoi.label: qual a numeracao do disco, do menor para o
17     ↪ maior
18 //discoHanoi.posicao: em qual posicao do plano cartesiano o disco
19     ↪ esta
20 //discoHanoi.torre: em qual torre que ele esta
21 //discoHanoi.largura: largura do disco
22
23 //Aqui eu conto quantos discos tem em uma torre
24 int contDiscos(discoHanoi disco[], int discoindex, char torre, int
25     ↪ numDiscos){
26     int i, total;
27
28     total = 0;
29     for(i = 0; i < numDiscos; i++){
30         if(i != discoindex){
31             if(disco[i].torre == torre)
32                 total++;
33         }
34     }
35     return total;
36 }
37
38 //Aqui eu movo os discos para a torre
39 void moveHanoi(int n, discoHanoi discos[], char torre, int
40     ↪ numDiscos){
41     Ponto p;
42     int i, disco, discoindex, auxposicao;
43
44     ///////////Procurando o disco que tenho que mover
45     for(i = 0; i < numDiscos; i++){
46         if(discos[i].label == n){
47             disco = discos[i].index; //O grupo que este disco
48                 ↪ pertence
49             discoindex = i; //A posicao do vetor de discoHanoi que
50                 ↪ este disco esta
51             break;
52         }
53     }
54
55     //Aqui eu movo na coordenada x o meu disco
56     if(torre == 'B'){
57         p.x = POSX_B - discos[i].largura/2;
58         discos[discoindex].torre = 'B';
59     }
60     else if (torre == 'C'){
61         p.x = POSX_C - discos[i].largura/2;
62         discos[discoindex].torre = 'C';
63     }
64     else{
65         p.x = POSX_A - discos[i].largura/2;
66         discos[discoindex].torre = 'A';
67     }

```

```

61
62 //posicao do chao + quantos discos tem naquela torre
63 p.y = CHAO + contDiscos(discos, discoindex, torre, numDiscos) *
    ↪ 10; //vezes largura de cada disco
64
65 Move(p, disco);
66
67 while(!ApertouTecla(GLFW_KEY_ENTER))
68     DesenhaFrame();
69 }
70
71 void hanoi(int n, discoHanoi disco[], char a, char b, char c, int
    ↪ numDiscos){
72     /* mova n discos do pino a para o pino b usando
73        o pino c como intermediario */
74
75     if (n == 1){
76         printf("\nmova disco %d de %c para %c\n", n, a, b);
77         moveHanoi(n, disco, b, numDiscos);
78     }
79     else
80     {
81         hanoi(n - 1, disco, a, c, b, numDiscos);
82         ↪ // H1
83         printf("\nmova disco %d de %c para %c\n", n, a, b);
84         moveHanoi(n, disco, b, numDiscos);
85
86         hanoi(n - 1, disco, c, b, a, numDiscos);
87         ↪ // H2
88     }
89 }
90
91 int main(void){
92     int numDiscos, i;
93     discoHanoi disco[5];
94     Ponto p;
95
96     do{
97         printf("\nDigite uma quantidade de discos menor ou igual a
98             ↪ 5: ");
99         scanf("%d", &numDiscos);
100     }while(numDiscos > 5 || numDiscos <= 0); //Pergunte a
101     ↪ quantidade de discos de novo se o usuario colocar um
102     ↪ numero maior que 5 ou menor ou igual a 0
103
104     AbreJanela(500, 500, "Torre de Hanoi");
105     PintarFundo(255, 255, 255);
106
107     //////////////////////////////////// Chao e torres
108     ↪ ////////////////////////////////////
109     CriaGrupo(); //Grupo das coisas que nao se mexem
110     //Chao
111     p.x = -100;
112     p.y = -80;
113     CriaRetangulo(200, 10, p);
114     Pintar(0, 0, 0);

```

```

109
110 //A
111 p.x = POSX_A;
112 p.y = CHAO;
113 CriaRetangulo(5, 60, p);
114 Pintar(255, 0, 0);
115
116 //B
117 p.x = POSX_B;
118 p.y = CHAO;
119 CriaRetangulo(5, 60, p);
120 Pintar(255, 0, 0);
121
122 //C
123 p.x = POSX_C;
124 p.y = CHAO;
125 CriaRetangulo(5, 60, p);
126 Pintar(255, 255, 0);
127
128 //////////////// Discos //////////////////////////
129 p.y = CHAO;
130 for (i = 0; i < numDiscos; i++){
131
132     disco[i].index = CriaGrupo(); //Cada disco tem que estar em
133     ↪ um grupo para poderem se mover independente
134     disco[i].largura = 10 * (numDiscos - i); //largura do disco
135     ↪ pra torre ficar bonitinha
136     disco[i].posicao = p.y; //posicao original do disco
137     disco[i].torre = 'A'; //torre de partida
138     disco[i].label = numDiscos - i; //Label para os discos
139     p.x = POSX_A - disco[i].largura/2;
140
141     CriaRetangulo(disco[i].largura, 10, p);
142     Pintar(rand()%256, rand()%256, rand()%256);
143
144     p.y += 10;
145 }
146
147 printf("Agora vou mostrar passo a passo como se resolve esta
148     ↪ torre. Aperte enter para o proximo passo");
149 while (!ApertouTecla(GLFW_KEY_ENTER))
150     DesenhaFrame();
151
152 hanoi(numDiscos, disco, 'A', 'B', 'C', numDiscos);
153
154 printf("\nPronto! E assim que se resolve esta torre!");
155
156 Desenha();
157 return 0;
158 }

```

Exercício 7.2

Item 7.2a Esta prática reforça o modo de utilização da função `CriaReta`, usando a função `movaCaneta`, na linha 10.

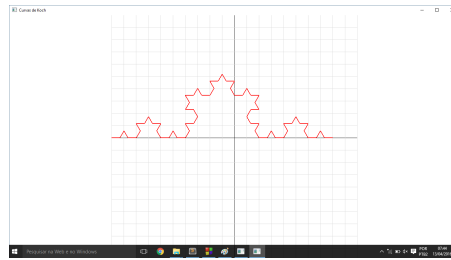


Figura 7.2 Curva de Koch de ordem 3

Listagem 7.2 Código fonte da curva de koch

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <math.h>
4  #include <playAPC/playapc.h>
5
6  #define LARGURA 1962//800
7  #define ALTURA 1280//600
8  #define SIZE 180
9
10 Ponto movaCaneta(Ponto from, double theta, double len) {
11     Ponto to;
12     double rad = theta*PI/180;
13     to.x = from.x + len*cos(rad);
14     to.y = from.y + len*sin(rad);
15     return to;
16 }
17
18
19 void koch (Ponto from, double len, double theta, int order) {
20     Ponto to;
21     double rad = theta*PI/180;
22
23     if (order==0) {
24         to.x = from.x + len*cos(rad);
25         to.y = from.y + len*sin(rad);
26         CriaReta(from, to); Grafite(2); Pintar(255,0,0);
27     }
28     else {
29         koch(from, len/3, theta, order-1);
30
31         to = movaCaneta(from, theta, len/3);
32         from = to; //dispensável.
33         // seu uso é para tornar intuitiva a chamada abaixo
34         koch(from, len/3, theta+60, order-1);
35
36         to = movaCaneta(from, theta+60, len/3);
37         from = to;
38         koch(from, len/3, theta-60, order-1);
39
40         to = movaCaneta(to, theta-60, len/3);

```

```

41     from = to;
42     koch(from, len/3, theta, order-1);
43 }
44 }
45
46
47 int main(void) {
48     int ordem = 1;
49     Ponto p0;
50
51     AbreJanela(LARGURA,ALTURA, "Curvas de Koch");
52     MostraPlanoCartesiano(10);
53     PintarFundo(255, 255, 255);
54
55     p0.x = -100; p0.y = 0.0;
56
57
58     koch(p0, SIZE, 0, ordem);
59
60
61
62     Desenha();
63
64     return 0;
65 }

```

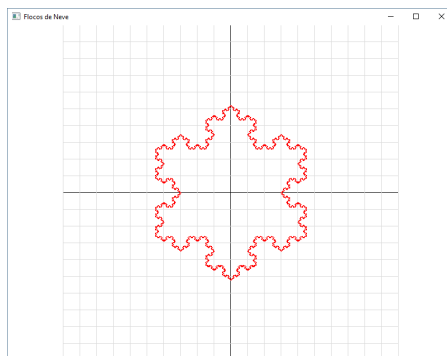


Figura 7.3 Floco de neve

Listagem 7.3 Código fonte do floco de neve

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <math.h>
4  #include <playAPC/playapc.h>
5
6  #define LARGURA 800 //1962
7  #define ALTURA 600 //1280
8  #define SIZE 180
9

```



```

10 Ponto movaCaneta(Ponto from, double theta, double len) {
11     Ponto to;
12     double rad = theta*PI/180;
13     to.x = from.x + len*cos(rad);
14     to.y = from.y + len*sin(rad);
15     return to;
16 }
17
18
19 Ponto koch (Ponto from, double len, double theta, int order) {
20     Ponto to;
21     double rad = theta*PI/180;
22
23     if (order==0) {
24         to.x = from.x + len*cos(rad);
25         to.y = from.y + len*sin(rad);
26
27         CriaReta(from, to); Grafite(2); Pintar(255,0,0);
28     }
29     else {
30         koch(from, len/3, theta, order-1);
31
32         to = movaCaneta(from, theta, len/3);
33         from = to; //dispensável.
34         // seu uso é para tornar intuitiva a chamada abaixo
35         koch(from, len/3, theta+60, order-1);
36
37         to = movaCaneta(from, theta+60, len/3);
38         from = to;
39         koch(from, len/3, theta-60, order-1);
40
41         to = movaCaneta(to, theta-60, len/3);
42         from = to;
43         koch(from, len/3, theta, order-1);
44     }
45     return movaCaneta(from, theta, len/3);
46 }
47
48 void floco_de_neve(int ordem) {
49     Ponto p0;
50     p0.x = -45.0; p0.y = 26.0; //centro do floco de neve
51
52
53     p0 = koch(p0, SIZE/2, 0, ordem);
54     p0 = koch(p0, SIZE/2, -120, ordem);
55     p0 = koch(p0, SIZE/2, 120, ordem);
56
57
58 }
59
60
61 int main(void) {
62     int ordem = 6;
63
64
65     MostraPlanoCartesiano(10);

```

```

66 | AbreJanela(LARGURA,ALTURA, "Flocos de Neve");
67 | PintaFundo(255, 255, 255);
68 |
69 | floco_de_neve(ordem);
70 |
71 | Desenha();
72 |
73 | return 0;
74 | }

```

Exercício 7.3

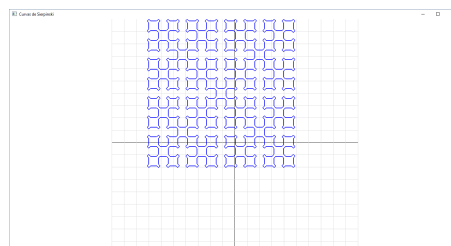


Figura 7.4 Curva de Sierpinski

Esta prática reforça o modo de utilizar a função `CriaReta`, usando as implementações das funções das linhas 16, 17, 18 e 19.

Listagem 7.4 Curva de Sierpinski

```

1 | #include <stdlib.h>
2 | #include <stdio.h>
3 | #include <math.h>
4 | #include <playAPC/playapc.h>
5 |
6 |
7 | // desenho realizado no estilo conhecido como Turtle Graphics
8 | #define LARGURA 1962 //800
9 | #define ALTURA 1280 //600
10 | #define SIZE 180
11 |
12 |
13 |
14 | typedef unsigned int Cardinal;
15 |
16 | Ponto a (Cardinal k, float h, Ponto p);
17 | Ponto b (Cardinal k, float h, Ponto p);
18 | Ponto c (Cardinal k, float h, Ponto p);
19 | Ponto d (Cardinal k, float h, Ponto p);
20 |
21 |
22 | Ponto reta(Cardinal fator, Cardinal ext, Ponto p) {
23 |     // ang é para multiplicar por 45 graus

```

```

24     Ponto p1;
25     double ar = fator * 45.0 * PI/180;
26
27     p1.x = p.x + ext*cos(ar);
28     p1.y = p.y + ext*sin(ar);
29     CriaReta(p,p1);
30     Grafite(2);
31     Pintar(0,0,255);
32     return p1;
33 }
34
35
36
37 Ponto a (Cardinal k, float h, Ponto p) {
38     if ( k > 0 ) {
39         p=a(k-1,h, p); p=reta(7,h, p);
40         p=b(k-1,h, p); p=reta(0,2*h, p);
41         p=d(k-1,h, p); p=reta(1,h, p);
42         p=a(k-1,h, p);
43     }
44     return p;
45 }
46
47 Ponto b (Cardinal k, float h, Ponto p) {
48     if ( k > 0 ) {
49         p=b(k-1,h, p); p=reta(5,h, p);
50         p=c(k-1,h, p); p=reta(6,2*h, p);
51         p=a(k-1,h, p); p=reta(7,h, p);
52         p=b(k-1,h, p);
53     }
54     return p;
55 }
56
57 Ponto c (Cardinal k, float h, Ponto p) {
58     if ( k > 0 ) {
59         p=c(k-1,h, p); p=reta(3,h, p);
60         p=d(k-1,h, p); p=reta(4,2*h, p);
61         p=b(k-1,h, p); p=reta(5,h, p);
62         p=c(k-1,h, p);
63     }
64     return p;
65 }
66
67 Ponto d (Cardinal k, float h, Ponto p) {
68     if ( k > 0 ) {
69         p=d(k-1,h, p); p=reta(1,h, p);
70         p=a(k-1,h, p); p=reta(2,2*h, p);
71         p=c(k-1,h, p); p=reta(3,h, p);
72         p=d(k-1,h, p);
73     }
74     return p;
75 }
76
77
78 int main(void) {
79

```

```
80 | Ponto p;  
81 | int i = 4; //dai pra cima a coisa fica grande e mais lenta  
82 | float h = 40;  
83 |  
84 | p.x = -70; p.y = 100; //ordem=1 cabe todo na tela  
85 |  
86 |  
87 | MostraPlanoCartesiano(10);  
88 | AbreJanela(LARGURA,ALTURA, "Curvas de Sierpinski");  
89 | PintarFundo(255, 255, 255);  
90 |  
91 |  
92 | if (i>0) h /= i*i;  
93 | p=a(i,h, p); p=reta(7,h, p);  
94 | p=b(i,h, p); p=reta(5,h, p);  
95 | p=c(i,h, p); p=reta(3,h, p);  
96 | p=d(i,h, p); p=reta(1,h, p);  
97 |  
98 |  
99 | Desenha();  
100 |  
101 | return 0;  
102 | }
```

