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

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

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

CONTRIBUTORS

Prof. Dr. José Carlos Loureiro Ralha, Departamento de Ciência da Computação - UnB, Brasília, DF, Brasil

Prof. Dr. Alexandre Zaghetto, Departamento de Ciência da Computação - UnB, Brasília, DF, Brasil

CONTENTS IN BRIEF

CONTEÚDO

LIST OF FIGURES

LIST OF TABLES

PREFACE

A playAPC é uma biblioteca gráfica voltada para alunos que possuem pouca ou nenhuma experiência de programação. Com ela, os alunos poderão obter uma resposta visual dos exercícios, catalisando o ensino, e permite aos professores uma maior gama de liberdade na elaboração de práticas de laboratórios, uma vez que abre a possibilidade de explorar problemas gráficos, como animações ou jogos que não exijam tantos recursos.

Este livro contém uma série de exercícios separados por tópicos os quais podem auxiliar professores na elaboração de práticas de laboratórios usando a playAPC. Na introdução de cada capítulo, existe uma sessão de Pré-requisitos que indica quais as funções da playAPC serão usadas nos problemas.

O capítulo 1 contém quatro exercícios de algoritmos sequenciais, três exercícios de algoritmos condicionais e cinco exercícios de algoritmos com repetição. O capítulo 2 contém dois exercícios de vetores e dois exercícios de matrizes. O capítulo 3 contém três exercícios de funções. O capítulo 4 contém quatro exercícios de algoritmos recursivos. O capítulo 5 contém três exercícios extras, que envolvem todos os conteúdos abordados no livro.

Sinayra Pascoal Cotts Moreira

Brasília, Universidade de Brasília
Setembro, 2016

ACKNOWLEDGMENTS

Gostaria de agradecer à todos os alunos e professores que me apoiaram no desenvolvimento deste projeto. Aos professores, que toda semana surgiam com novas ideias de práticas de laboratórios e, por consequência, novas funcionalidades para a biblioteca e aos alunos, que conseguiam realizar estas mesmas práticas.

Em especial, gostaria de agradecer ao professor Ralha, por ter me apoiado inicialmente com este projeto, com toda ajuda, paciência e ideias no início do desenvolvimento da biblioteca. Também gostaria de agradecer o professor Zaghetto, por ter abraçado a proposta da biblioteca e ter apoiado o desenvolvimento deste livro, além de estar sempre surgindo com novas ideias de funcionalidades.

ACRONYMS

UnB	Universidade de Brasília
APC	Análise e Programação de Algoritmos

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 conhec-

[1] http://playapc.zaghetto.com/category/como-instalar

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 apara a realização de uma tarefa.

Pré-requisitos

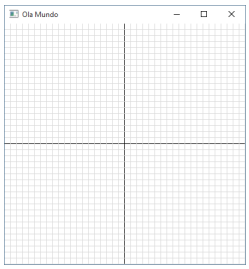
As práticas deste capítulo exigem que sejam utilizadas as funções

- `void AbreJanela(float largura, float altura, const char* titulo)`
- `void PintaFundo(int red, int green, int blue)`
- `void MostraPlanoCartesiano ( int intervalo )`
- `void Desenha()`

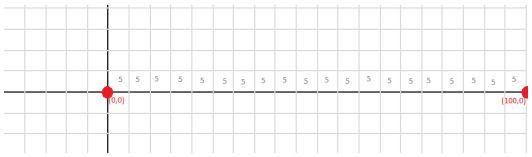
- `void Pintar ( int red , int green , int blue )`
- `int CriaCirculo ( float raio , Ponto meio)`
- `int CriaElipse ( float a , float b, Ponto meio)`
- `int CriaQuadrado( float lado , Ponto cantoesq )`
- `int CriaRetangulo ( float base , float altura , Ponto cantoesq )`
- `int CriaTriangulo ( float base , float altura , Ponto cantoesq )`
- `int CriaPoligono ( short int qtd , . . . )`

Problemas

1.1. Exiba um plano cartesiano de -100 a 100 com espaçamento de 5 unidades.



(a) Visualização de todo plano cartesiano



(b) De (0,0) até (100,0), existem 20 quadrados com 5 unidades de tamanho

Figura 1.1: Plano cartesiano de -100 à 100

1.2. Desenhe um boneco palito que utilize pelo menos uma vez as seguintes geometrias:

- Círculo
- Elipse
- Retângulo
- Triângulo
- Quadrado

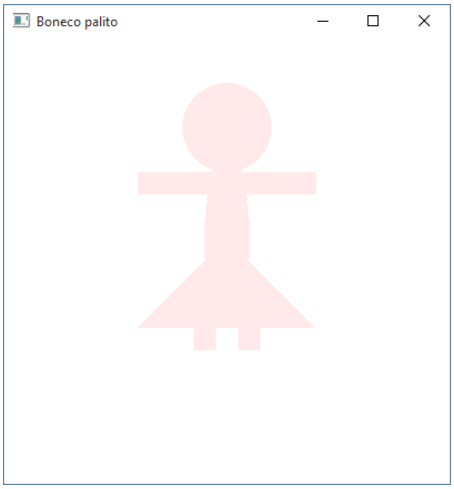


Figura 1.2: Boneco Palito

1.3. Exiba a estrela de Davi.

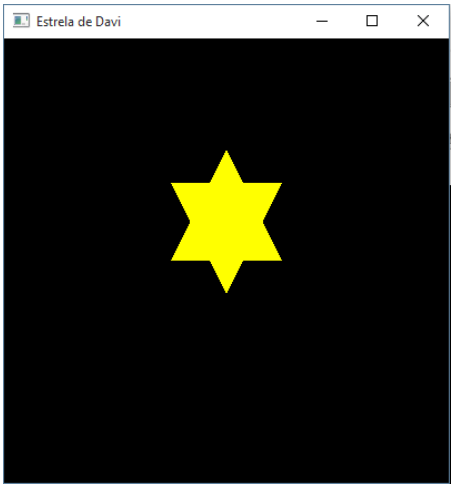


Figura 1.3: Estrela de Davi

1.4. Escreva um programa que solicite do usuário um raio de um círculo e exiba um quadrado inscrito neste círculo.

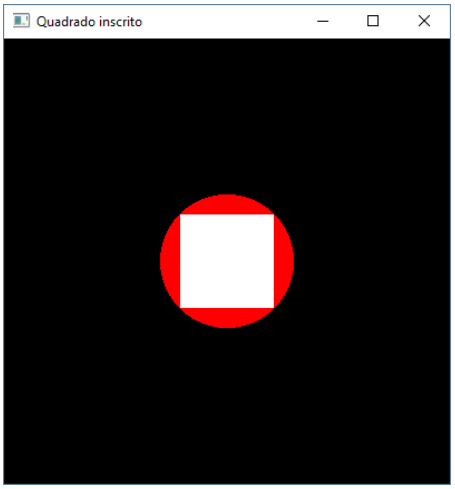


Figura 1.4: Quadrado inscrito em um círculo

Soluções

Exercício 1.1

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 PintarFundo, 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 coodernadas 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(){
4     Ponto p;
5     AbreJanela(400, 400, "Boneco palito");
6     PintarFundo(255, 255, 255);
7
8     p.x = 0;
9     p.y = 60;
```

[1] http://playapc.zaghetto.com/category/funcoes/geometrias

```
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,
            ↪ metade do menor raio da elipse, ponto central)
15      Pintar(255, 233, 234);
16
17      p.x = -40;
18      p.y = 30;
19      CriaRetangulo(80, 10, p); //(base, altura, ponto esquerdo
            ↪ inferior)
20      Pintar(255, 233, 234);
21
22      p.x = -40;
23      p.y = -30;
24      CriaTriangulo(80, 40, p); //(base, altura, ponto esquerdo
            ↪ inferior)
25      Pintar(255, 233, 234);
26
27      p.x = -15;
28      p.y = -40;
29      CriaQuadrado(10, p); //(lado, ponto esquerdo inferior)
30      Pintar(255, 233, 234);
31
32      p.x = 5;
33      p.y = -40;
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;
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     Pintar(255, 255, 0);
22
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.

Exercício 1.4

Esta prática se refere a exibir um quadrado inscrito em um círculo, a qual exercita o raciocínio matemático do aluno. O quadrado, criado pela função CriaQuadrado, precisa de um ponto de referência para ser criado, sendo este ponto o inferior esquerdo. Desta forma, o aluno teria que calcular, dado o raio do círculo, não apenas o lado do quadrado, mas também a posição que este ponto de referência precisa estar.

Listagem 1.4: Código fonte do quadrado inscrito

```
#include <playAPC/playapc.h>
#include <stdio.h>
#include <math.h>
int main(){
    float raio, lado, apotema;
    Ponto p1, p2;

    printf("Digite o valor do raio do circulo: ");
    scanf("%f", &raio);

    lado = raio * sqrt(2);
    apotema = lado/2;

    p1.x = 0;
    p1.y = 0;

    p2.x = -apotema;
    p2.y = -apotema;

    AbreJanela(400, 400, "Quadrado inscrito");
    PintarFundo(255, 255, 255);
    MostraPlanoCartesiano(10);

    CriaCirculo(raio, p1);
    Pintar(255, 0, 0);

    CriaQuadrado(lado, p2);
    Pintar(0, 0, 255);

    Desenha();

    return 0;
}
```

CAPÍTULO 2

ALGORITMOS CONDICIONAIS

Resumo

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

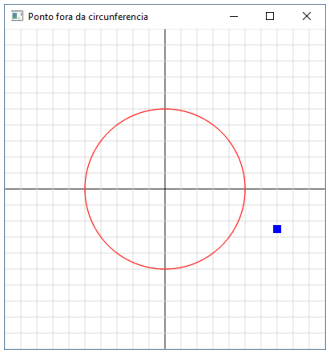
Pré-requisitos

As práticas deste capítulo exigem que sejam utilizadas as funções

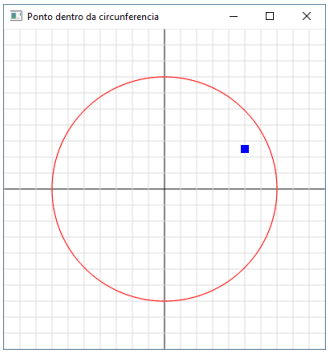
- `int CriaReta(Ponto p1 , Ponto p2)`
- `void CriaPonto(Ponto p)`
- `void Grafite(int espessura)`

Problemas

2.1. Escreva um programa que solicita do usuário um raio, a posição do centro de uma circunferência e a posição de um ponto qualquer. Exiba a cena e indique no título da janela se o ponto está dentro ou fora da circunferência.



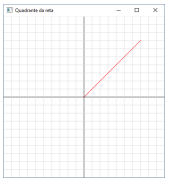
(a) Ponto está fora da circunferência



(b) Ponto está dentro da circunferência

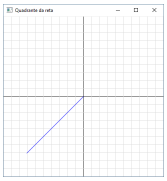
2.2. Escreva um programa que receba do usuário um valor de ângulo em graus e um valor de raio. Converta para radianos o ângulo e exiba uma reta com o raio fornecido pelo usuário e pinte-a de acordo com as seguintes regras:

- 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

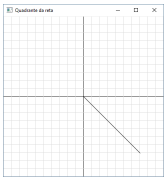


(a) Reta pertencente ao primeiro quadrante

(b) Reta pertencente ao segundo quadrante



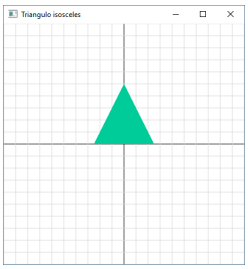
(c) Reta pertencente ao terceiro quadrante



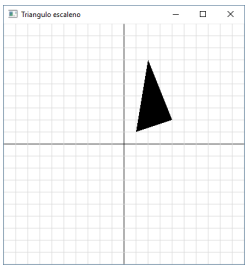
(d) Reta pertencente ao quarto quadrante

2.3. Escreva um programa em C que solicita três pontos A, B e C ao usuário, e verifica se esses valores satisfazem a condição de existência do triângulo.

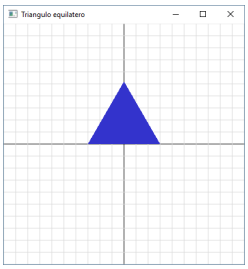
Caso essa condição seja satisfeita, exiba esse triângulo e escreva no título da janela se o triângulo é equilátero, isósceles ou escaleno [dica: não usar a função CriaTriangulo()].



(a) Triângulo isósceles



(b) Triângulo escaleno



(c) Triângulo equilátero

Soluções

Exercício 2.1

Esta prática exibe uma circunferência e indica se dado um ponto qualquer, se este ponto está dentro ou fora da circunferência, exercitando o conceito de distância entre dois pontos. Para a ampliação da espessura do ponto, para ele não ser apenas um pixel, utiliza-se a função Grafite.

Listagem 2.1: Código fonte do ponto dentro ou fora da circunferência

```
1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <math.h>
4  int main(){
5      float raio, d;
6      Ponto centro, p;
7
8      printf("Digite o valor do raio da circunferencia: ");
9      scanf("%f", &raio);
10     printf("\nDigite a posicao x do centro da circunferencia: ");
11     scanf("%f", &centro.x);
12     printf("\nDigite a posicao y do centro da circunferencia: ");
13     scanf("%f", &centro.y);
14
15     printf("\nDigite a posicao x de um ponto: ");
16     scanf("%f", &p.x);
17     printf("\nDigite a posicao y de um ponto: ");
18     scanf("%f", &p.y);
19
20     d = sqrt(pow(centro.x - p.x, 2) + pow(centro.y - p.y, 2));
21
22     if(d > raio){
23         AbreJanela(400, 400, "Ponto fora da circunferencia");
```

```
24      }
25      else{
26          AbreJanela(400, 400, "Ponto dentro da circunferencia");
27      }
28
29      PintarFundo(255, 255, 255);
30      MostraPlanoCartesiano(10);
31
32      CriaCircunferencia(raio, centro);
33      Pintar(255, 0, 0);
34
35      CriaPonto(p);
36      Pintar(0, 0, 255);
37      Grafite(10);
38      Desenha();
39
40      return 0;
41  }
```

`int CriaPonto(Ponto p)`

A função CriaPonto, na linha 35, 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 37, 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.

Exercício 2.2

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 2.2: Código fonte do quadrante da reta

```
1   #include <playAPC/playapc.h>
2   #include <stdio.h>
3   #include <math.h>
4
5   int main(){
6       int angulo;
7       float anguloRad, raio;
8       Ponto p1, p2;
9
10      printf("Digite um angulo de 0 a 360 graus:");
11      scanf("%d", &angulo);
12
13      printf("Digite um raio de 0 a 100:");
```

```
14      scanf("%f", &raio);
15
16      p1.x = 0;
17      p2.x = 0;
18
19      anguloRad = (PI * angulo)/180;
20
21      p2.y = sin(anguloRad) * raio;
22      p2.x = cos(anguloRad) * raio;
23
24
25      AbreJanela(400, 400, "Quadrante da reta");
26      PintarFundo(255, 255, 255);
27      MostraPlanoCartesiano(10);
28
29      CriaReta(p1, p2);
30
31      if(p2.x > 0){
32          if(p2.y > 0)
33              Pintar(255, 0, 0); //vermelho: 1 quadrante
34          else
35              Pintar(0, 0, 0); //preto: 4 quadrante
36      }
37      else{
38          if(p2.y > 0)
39              Pintar(0, 255, 0); //verde: 2 quadrante
40          else
41              Pintar(0, 0, 255); //azul: 3 quadrante
42      }
43
44      Desenha();
45
46  }
```

`int CriaReta(Ponto p1, Ponto p2)`

A função CriaReta, na linha 29, cria uma geometria do tipo RETA, retornando um índice deste tipo de geometria. Seu primeiro e segundo argumento são duas variáveis do tipo Ponto.

Exercício 2.3

Esta prática exercita o conceito matemático de condição de existência e de classificação de triângulo, além de exercitar ifs aninhados.

Listagem 2.3: Código fonte de exibir triângulo caso ele exista

```
1  #include <stdio.h>
2  #include <playAPC/playapc.h>
3  #include <math.h>
4
5  int main(){
6      Ponto p1, p2, p3;
7      float a, b, c, diffa, diffb, diffc, somaa, somab, somac;
```

```
8       int triangulo = 1;
9
10      printf("Indique coordenada x do ponto 1:");
11      scanf("%f", &p1.x);
12      printf("Indique coordenada y do ponto 1:");
13      scanf("%f", &p1.y);
14
15      printf("Indique coordenada x do ponto 2:");
16      scanf("%f", &p2.x);
17      printf("Indique coordenada y do ponto 2:");
18      scanf("%f", &p2.y);
19
20      printf("Indique coordenada x do ponto 3:");
21      scanf("%f", &p3.x);
22      printf("Indique coordenada y do ponto 3:");
23      scanf("%f", &p3.y);
24
25      a = sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
26      b = sqrt(pow(p3.x - p2.x, 2) + pow(p3.y - p2.y, 2));
27      c = sqrt(pow(p3.x - p1.x, 2) + pow(p3.y - p1.y, 2));
28
29      printf("a: %f \t b: %f \t c: %f \n", a, b, c);
30
31      diffa = fabs(b - c);
32      somaa = b + c;
33
34      diffb = fabs(a - c);
35      somab = a + c;
36
37      diffc = fabs(a - b);
38      somac = a + b;
39
40      if(diffa < a && a < somaa){
41          if(diffb < b && b < somab){
42              if(diffc < c && c < somac){
43
44                  if(a == b && a == c){
45                      AbreJanela(400, 400, "Triangulo equilatero");
46                      PintarFundo(255, 255, 255);
47                      MostraPlanoCartesiano(10);
48
49                      CriaPoligono(3, p1, p2, p3);
50                      Pintar(51, 51, 204);
51                  }
52                  else if(a == b || a == c || b == c){
53                      AbreJanela(400, 400, "Triangulo isosceles");
54                      PintarFundo(255, 255, 255);
55                      MostraPlanoCartesiano(10);
56
57                      CriaPoligono(3, p1, p2, p3);
58                      Pintar(0, 204, 153);
59                  }
60                  else{
61                      AbreJanela(400, 400, "Triangulo escaleno");
62                      PintarFundo(255, 255, 255);
63                      MostraPlanoCartesiano(10);
```

```
64 |
65 |                     CriaPoligono(3, p1, p2, p3);
66 |                     Pintar(0, 0, 0);
67 |                 }
68 |                 Desenha();
69 |             }
70 |             else{
71 |                 triangulo = 0;
72 |             }
73 |         }
74 |         else{
75 |             triangulo = 0;
76 |         }
77 |     }
78 |     else{
79 |         triangulo = 0;
80 |     }
81 |
82 |     if(!triangulo){
83 |         printf("Nao satisfaz existencia de triangulo");
84 |     }
85 |
86 |     return 0;
87 | }
```

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.

Pré-requisitos

As práticas deste capítulo exigem que sejam utilizadas as funções

- `int CriaGrupo()`
- `void Move(Ponto p, int grupo)`
- `int Desenha1Frame()`
- `void Gira( float theta , int index )`

- `int ApertouTecla(int tecla)`

- `int AbreImagem(const char *src)`

- `void AssociaImagem(int textura, geometrias_validas nome, int index)`

Problemas

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

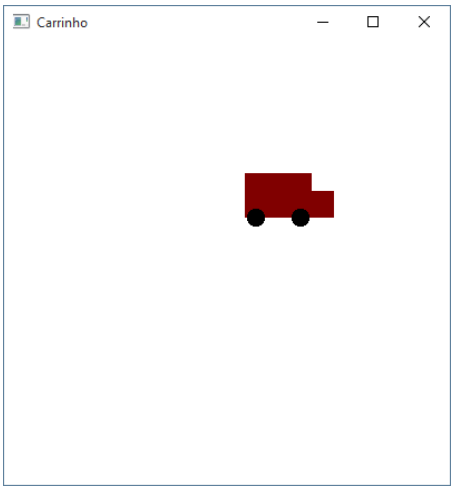


Figura 3.1: Carro se movendo da posição -100 até a posição 100

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

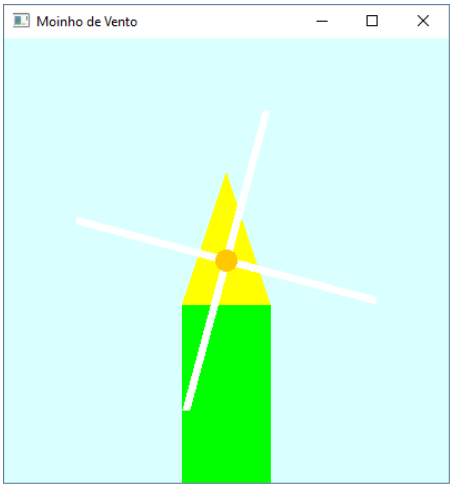


Figura 3.2: Moinho de vento

3.3. Escreva um programa utilizando a playAPC que simula simultaneamente o movimento da Terra ao redor do Sol e o movimento da Lua ao redor da Terra. Considere que as trajetórias de ambas são elípticas. No caso da Terra, o Sol é um dos focos e no caso da Lua, a Terra é um dos focos. Não é necessário simular a proporção real entre os semieixos maiores (a) da Lua e da Terra, nem a excentricidade (e) das duas trajetórias. Encontre empiricamente valores de (a) e (e) de forma que seja possível observar trajetórias elípticas. Simule, porém, a proporção real entre os movimento de translação da Terra e da Lua.

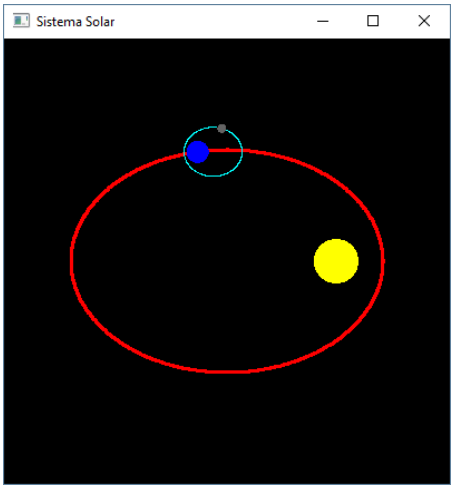
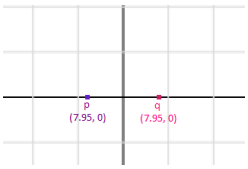


Figura 3.3: Sistema solar

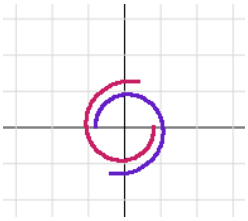
3.4. Sabe-se que a equação da espiral hiperbólica pode ser defina por

$$\begin{aligned} x &= a\cos(\theta) \\ y &= a\sin(\theta) \end{aligned}. \tag{3.1}$$

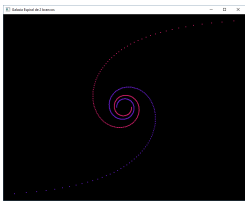
onde a é a assíntota para y e θ o ângulo equivalente ao ângulo em coordenadas polares. Para $a \leftarrow 100$ e $\theta \in (0, 4\pi)$, desenhe duas espirais hiperbólicas calculando seus pontos como é descrito na Equação 3.1. Para ponto p em (x, y) de uma das hiperbólica, o ponto p da outra espiral deve estar posicionado em $(-x, -y)$.



(a) Posição inicial aproximada de ambas as espirais



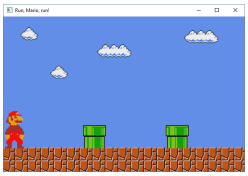
(b) 100º iteração no processo de criação das espirais



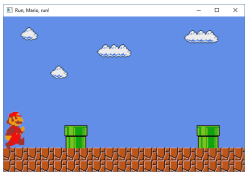
(c) Espiral completa

3.5. Crie uma animação onde o Mário deve começar no canto esquerdo da tela e seu objetivo é andar até o canto direito da tela. Porém, haverão dois canos que serão posicionados aleatoriamente no meio do caminho, forçando o Mário a pulá-los para não colidir com eles. Para criar a animação de andar, altere as imagens do retângulo onde será desenhado o Mario com a função AssociaImagem e, após essa chamada, utilize a função Desenha1Frame para renderizar a troca de imagens.

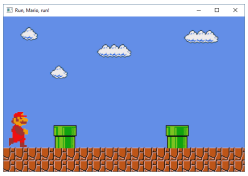
Para simplificação do problema, considere que o Mário, ao realizar o pulo, ele deva executar meia trajetória circular, onde θ varia de π até 0 e o raio do pulo seja de 40 unidades.



(a) Começo da animação



(b) Parte 1 da animação do Mário



(c) Parte 2 da animação do Mário

(d) Mário pulando

Soluções

Exercício 3.1

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

Listagem 3.1: 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 10, 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.

```
void Move(Ponto p)
void Move(Ponto p, int index)
```

A função Move, na linha 33, é uma das três funções de transformação implementadas na playAPC. Há duas formas de utilizar esta função. No caso da Listagem 3.1, seu primeiro argumento é o ponto no plano cartesiano que se deseja mover toda as geometrias e o segundo argumento o grupo que se deseja mover. O grupo será transladado até que o ponto de referência da primeira geometria deste grupo esteja no novo ponto desejado, 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 x e y' a coordenada resultante do eixo y.

$$x' = x + d_x$$
$$y' = y + d_y$$

Onde d_x e d_y são o incremento dada a posição original do ponto.

`int Desenha1Frame()`

A função Desenha1Frame, na linha 34, 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.

Exercício 3.2

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 helices. Somente o helices sofre a ação de girar.

Listagem 3.2: 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
11     moinho = CriaGrupo();
12
13         p1.x = -20;
14         p1.y = -20;
15         CriaTriangulo(40,60,p1);
16         Pintar(255, 255, 0);
17
18         p1.x = -20;
19         p1.y = -100;
20         CriaRetangulo(40,80,p1);
21         Pintar(0, 255, 0);
22
23     helices = CriaGrupo();
24
25         p1.x = 0;
26         p1.y = 0;
27         // Hélice 1
28         p2.x = 0;
29         p2.y = 70;
30         CriaReta(p1, p2);
31         Pintar(255, 255, 255);
32         Grafite(8);
33
34         // Hélice 2
35         p2.x = 70;
36         p2.y = 0;
37         CriaReta(p1, p2);
38         Pintar(255, 255, 255);
39         Grafite(8);
40
41         // Helice 3
42         p2.x = 0;
43         p2.y = -70;
44         CriaReta(p1, p2);
45         Pintar(255, 255, 255);
46         Grafite(8);
47
```

```
48          // Helice 4
49          p1.x = -70;
50          p2.y = 0;
51          CriaReta(p1, p2);
52          Pintar(255, 255, 255);
53          Grafite(8);
54
55          p1.x = 0;
56          p1.y = 0;
57          CriaCirculo(5,p1);
58          Pintar(255,200,0);
59
60      while( angulo > 0){
61          Desenha1Frame();
62          Gira(angulo, helices);
63          angulo++;
64      }
65
66      Desenha();
67      return 0;
68  }
```

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

A função Gira, na linha 62, é uma das três funções de transformação implementadas na playAPC. Há duas formas de utilizar esta função. No caso da Listagem 3.2, seu primeiro argumento é um ângulo θ em graus e seu segundo argumento indica o grupo que irá sofrer a ação de girar, recalculando a posição de cada pixel utilizando a Definição 2.

Definição 2 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.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Exercício 3.4

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 3.3: 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   Desenha();
59
```

```
60 || }
```

```
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 3.3, seu primeiro argumento é um ângulo θ em graus e ele irá girar todas as geometrias criadas de acordo com a Definição 2.

```
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.

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_Fn	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_DEL	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 ($\div$)
GLFW_KEY_KP_MULTIPLY	Tecla multiplicar do keypad ($\times$)
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

Exercício 3.5

Esta prática introduz o conceito de colisão de objetos, com uma colisão entre dois retângulos, e também ao uso de imagens. Na introdução a utilização de

imagens, não é necessário que o aluno tenha como pré-requisito o conceito de manipulação de arquivos em C. Como a função Desenha1Frame renderiza $\frac{1}{60}$ segundos, para criar o efeito de uma animação menos fluída, se faz necessário sua chamada repetidas vezes.

Listagem 3.4: Código fonte do Mario animado

```
#include <playAPC/playapc.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int main(){
    Ponto p, pcano1, pcano2, pmario;
    int imgbackground, imgcano,
        imgmario_caminhando_1, imgmario_caminhando_2,
            ↪ imgmario_caminhando_3, imgmario_pre_caminhada,
            ↪ imgmario_pre_salto, imgmario_salto;
    int grupocano, grupomario, grupoback;
    float t, x, y;
    int raio_pulo = 40;

    srand (time(NULL));
    AbreJanela(600, 385, "Run, Mario, run!");

    imgbackground = AbreImagem("Mario_Run/background.png");
    imgcano = AbreImagem("Mario_Run/cano.png");
    imgmario_caminhando_1 = AbreImagem("Mario_Run/
        ↪ mario_caminhando_1.png");
    imgmario_caminhando_2 = AbreImagem("Mario_Run/
        ↪ mario_caminhando_2.png");
    imgmario_caminhando_3 = AbreImagem("Mario_Run/
        ↪ mario_caminhando_3.png");
    imgmario_pre_caminhada = AbreImagem("Mario_Run/
        ↪ mario_pre_caminhada.png");
    imgmario_pre_salto = AbreImagem("Mario_Run/mario_pre_salto.png"
        ↪ );
    imgmario_salto = AbreImagem("Mario_Run/mario_salto.png");

    //Background
    grupoback = CriaGrupo();
        p.x = -156;
        p.y = -100;
        CriaRetangulo(312, 200, p);
        Pintar(255, 255, 255);
        AssociaImagem(imgbackground);

    //Canos
    grupocano = CriaGrupo();
        pcano1.x = -(100 - rand()%50);
        pcano1.y = -70;
        CriaQuadrado(30, pcano1);
        Pintar(255, 255, 255);
        AssociaImagem(imgcano);

```

```
42          pcano2.x = 50 + rand()%50;
43          pcano2.y = -70;
44          CriaQuadrado(30, pcano2);
45          Pintar(255, 255, 255);
46          AssociaImagem(imgcano);
47
48          grupomario = CriaGrupo();
49              p.x = -156;
50              p.y = -75;
51              CriaRetangulo(30, 55, p);
52              Pintar(255, 255, 255);
53              AssociaImagem(imgmario_pre_caminhada);
54
55
56          t = 0;
57          while(t < 100){
58              t++;
59              Desenha1Frame();
60          }
61
62          for(p.x = -156; p.x < 125; p.x += 5){
63
64              if(p.x + raio_pulo > pcano1.x && p.x < pcano1.x +
                    ↪ raio_pulo){
65                  AssociaImagem(imgmario_pre_salto);
66                  t = 0;
67                  while(t < 50){
68                      t++;
69                      Desenha1Frame();
70                  }
71
72                  AssociaImagem(imgmario_salto);
73                  t = 0;
74                  pmario = p;
75                  while(t <= PI){
76                      x = -(raio_pulo * cos(t)); //circulo (0,0) com
                            ↪ raio 30 (x esta invertido porque ta indo
                            ↪  de 0 a PI)
77                      y = 50 * sin(t);
78
79                      p.x = pmario.x + raio_pulo + x; //posicao do
                            ↪ deslocamento x + raio 30 + posicao
                            ↪ inicial do Mario
80                      p.y = pmario.y + y;
81
82                      Move(p, grupomario);
83                      Desenha1Frame();
84                      Desenha1Frame();
85                      Desenha1Frame();
86
87                      t+= 0.5;
88                  }
89                  p.y = -75;
90              }
91
```

```
92              if(p.x + raio_pulo > pcano2.x && p.x < pcano2.x +
                    ↪ raio_pulo){
93                  AssociaImagem(imgmario_pre_salto);
94                  t = 0;
95                  while(t < 50){
96                      t++;
97                      Desenha1Frame();
98                  }
99
100                 AssociaImagem(imgmario_salto);
101                 t = 0;
102                 pmario = p;
103                 while(t <= PI){
104                     x = -(raio_pulo * cos(t)); //circulo (0,0) com
                            ↪ raio 30 (x esta invertido porque ta indo
                            ↪  de 0 a PI)
105                     y = 50 * sin(t);
106
107                     p.x = pmario.x + raio_pulo + x; //posicao do
                            ↪ deslocamento x + raio 30 + posicao
                            ↪ inicial do Mario
108                     p.y = pmario.y + y;
109
110                     Move(p, grupomario);
111                     Desenha1Frame();
112                     Desenha1Frame();
113                     Desenha1Frame();
114
115                     t+= 0.5;
116                 }
117                 p.y = -75;
118             }
119
120             AssociaImagem(imgmario_caminhando_1);
121             Desenha1Frame();
122             Desenha1Frame();
123             Desenha1Frame();
124             Desenha1Frame();
125
126             Move(p, grupomario);
127
128             AssociaImagem(imgmario_caminhando_2);
129             Desenha1Frame();
130             Desenha1Frame();
131             Desenha1Frame();
132             Desenha1Frame();
133
134             p.x += 5;
135
136             Move(p, grupomario);
137
138             AssociaImagem(imgmario_caminhando_3);
139             Desenha1Frame();
140             Desenha1Frame();
141             Desenha1Frame();
142             Desenha1Frame();
```

```
143 ||
144 ||          }
145 ||
146 ||      AssociaImagem(imgmario_pre_caminhada);
147 ||      Desenha();
148 ||      return 0;
149 || }
```

```
int AbreImagem(const char *src)
```

A função AbreImagem, na linha 17, recebe como argumento o caminho, relativo ou absoluto, da imagem que se deseja carregar no programa. Ela retorna um índice associado àquela imagem, que deve ser passado como argumento para a função AssociaImagem. A função AbreImagem consegue ler imagens de extensão bmp, jpg e png.

```
void AssociaImagem(int textura)
void AssociaImagem(int textura, geometrias_validas nome, int index)
```

A função AssociaImagem, na linha 32, possui duas formas de ser utilizada. No caso da Listagem 3.4, ele associa a imagem a última geometria que foi definida, no caso, o retângulo do background. Para criar o efeito de animação no Mario, determinou-se que o retângulo Mario seria a última geometria do último grupo criado. Dessa forma, as próximas chamadas da função AssociaImagem estariam referenciando o retângulo da linha 51.

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.

Pré-requisitos

As práticas deste capítulo exigem que sejam utilizadas as funções

- `void MudaLimitesJanela(int limite)`

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

Problemas

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

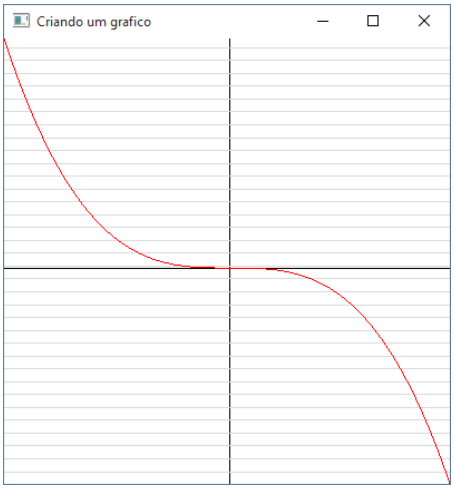


Figura 4.1: Gráfico do polinômio $-x^3$

4.2. Escreva uma programa que solicita ao usuário 20 componentes RGB do tipo int entre 0 e 255 que são armazenadas em três vetores R, G e B. Em seguida, os valores de cada vetor são filtrados por meio da filtragem de média móvel central, como indica a Equação 4.1 e o resultado é armazenado em três novos vetores Rf, Gf, Bf. O tamanho da janela de filtragem é fixo e igual a 3.

$$\begin{aligned} Rf[i] &= \frac{R[i-1]+R[i]+R[i+1]}{3} \\ Gf[i] &= \frac{G[i-1]+G[i]+G[i+1]}{3} \\ Bf[i] &= \frac{B[i-1]+B[i]+B[i+1]}{3} \end{aligned} \tag{4.1}$$

Equação 1: Filtragem dos componentes RGB do componente i com janela de filtragem igual a 3

Exiba graficamente dois vetores coloridos, um composto pelas componentes RGB originais e outro pelas componentes RfGfBf filtradas. No final, o programa deve calcular também a distância euclidiana média entre os dois vetores RGB e RfGfBf, como indica a Equação 4.2.

$$\frac{1}{n}\sum_{i=0}^{n}\sqrt{(R[i]-Rf[i])^2+(G[i]-Gf[i])^2+(B[i]-Bf[i])^2} \tag{4.2}$$

Equação 2: Cálculo da distância euclidiana média

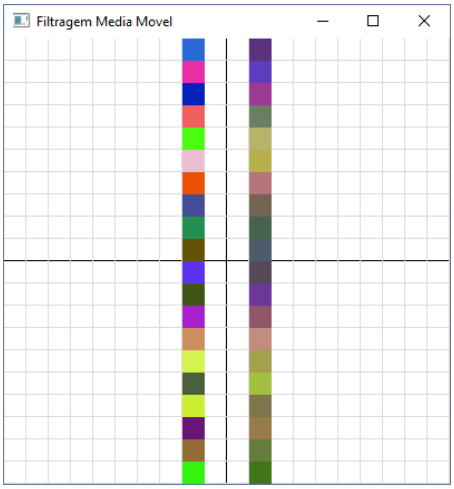


Figura 4.2: À esquerda, 20 quadrados com componentes RGB fornecidos pelo usuário. À direita, aplicação do filtro de média móvel central em cada quadrado

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, 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 do 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
22      Pintar(255, 0, 0);
23
24      Desenha();
25
26  }
```

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

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 20, 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.

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 renderização
2	Redimensiona a tela a fim de comportar toda a função dentro da janela de rendedização sem interferir na escala de ambos os eixos

Exercício 4.2

Esta prática exercita conceitos de processamento de imagens, aplicando um filtro dado os componentes RGB da figura, além de exercitar conceitos teóricos de matemática, como o funcionando do operador σ

Listagem 4.2: Código fonte do filtro de média móvel central

```
1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <math.h>
4
5  #define QTD 22
6
7  int main(){
```

```
8       int R[QTD], G[QTD], B[QTD];
9       int Rf[QTD], Gf[QTD], Bf[QTD];
10      int mfiltro = 3;
11      int somaaux[3];
12      Ponto p, pf;
13      float mse = 0, distancia;
14
15      R[0] = 0; R[QTD-1] = 0;
16      G[0] = 0; G[QTD-1] = 0;
17      B[0] = 0; B[QTD-1] = 0;
18
19      for(int i = 1; i < QTD-1; i++){
20          printf("\n****\nElemento n%d\n****\n", i);
21
22          printf("\nInsira componente R:");
23          scanf("%d", &R[i]);
24          printf("\nInsira componente G:");
25          scanf("%d", &G[i]);
26          printf("\nInsira componente B:");
27          scanf("%d", &B[i]);
28
29      }
30
31      for(int i = 1; i < QTD - 1; i++){
32          somaaux[0] = 0;
33          somaaux[1] = 0;
34          somaaux[2] = 0;
35          for(int j = -1; j < mfiltro - 1; j++){
36              somaaux[0] += R[(i + j)];
37              somaaux[1] += G[(i + j)];
38              somaaux[2] += B[(i + j)];
39          }
40          Rf[i] = somaaux[0]/mfiltro;
41          Gf[i] = somaaux[1]/mfiltro;
42          Bf[i] = somaaux[2]/mfiltro;
43
44      }
45
46      AbreJanela(400, 400, "Filtragem Media Movel");
47      PintarFundo(255, 255, 255);
48      MostraPlanoCartesiano(10);
49
50      p.x = -20;
51      p.y = 90;
52
53      pf.x = 10;
54      pf.y = 90;
55      for(int i = 1; i < QTD - 1; i++){
56          CriaQuadrado(10, p);
57          Pintar(R[i], G[i], B[i]);
58          p.y -= 10;
59
60          CriaQuadrado(10, pf);
61          Pintar(Rf[i], Gf[i], Bf[i]);
62          pf.y -= 10;
63
```

```
64          mse +=      pow(R[i] - Rf[i], 2) + pow(R[i] - Gf[i], 2) +
                ↪ pow(R[i] - Bf[i], 2)  //distancia euclidiana de R
65              +   pow(G[i] - Rf[i], 2) + pow(G[i] - Gf[i], 2) +
                    ↪ pow(G[i] - Bf[i], 2) //distancia euclidiana
                    ↪ de G
66              +   pow(B[i] - Rf[i], 2) + pow(B[i] - Gf[i], 2) +
                    ↪ pow(B[i] - Bf[i], 2); //distancia euclidiana
                    ↪  de B
67
68      }
69      mse = (sqrt(mse))/(QTD-2);
70
71      printf("A distancia euclidiana media e %.2f\n", mse);
72
73      Desenha();
74  }
```

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.

Pré-requisitos

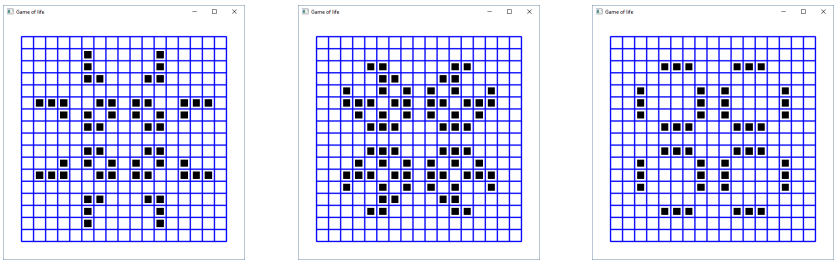
As práticas deste capítulo exigem que sejam utilizadas as funções

- `void ExtraiRGBdeBMP(const char *imagepath, int largura, int altura, int (&R)[tam_x][tam_y], int (&G)[tam_x][tam_y], int (&B)[tam_x][tam_y])`

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:

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



(a) 1ª geração (b) 2ª geração (c) 3ª geração

5.2. Implementar o filtro de média móvel para uma matriz M de inteiros (de 0 a 255) com 3 planos RGB, cada um com 100 linhas e 100 colunas. A matriz a ser filtrada é a imagem uma imagem BMP do Mario. Ela deve ser lida e em seguida mostrada na tela do computador. A imagem filtrada deve igualmente ser apresentada na tela do computador.

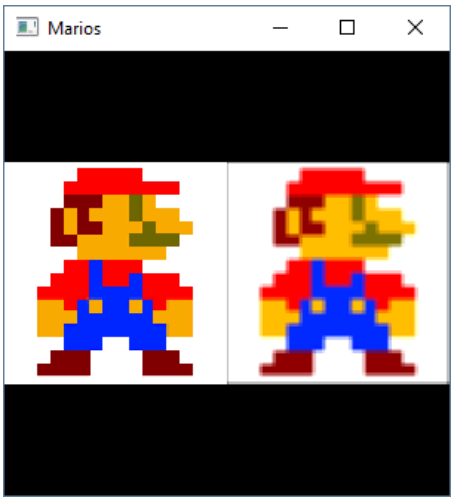


Figura 5.2: Filtro Motion Blur

Soluções

Exercício 5.1

Esta prática mostra como utilizar o retorno das 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][
       ↪ COLUNAS], geoIndex[LINHAS][COLUNAS];
15     int l, c, i, j, soma, r, s, n, ESTADO, deltaQuad=10, largQuad =
       ↪  6, marg = 2;
16     Ponto p1, p2, p3;
17
18     ////////////////////////////////
19     // Define o Janela            //
20     ////////////////////////////////
21     AbreJanela(550, 550, "Game of life");
22     PintarFundo(255, 255, 255);
23
24     ///////////////////
25     // Desehna 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 MORTO
65      for(l=0; l<LINHAS; l++){
66          for(c=0; c<COLUNAS; c++){
67              geracaoAtual[l][c]=MORTO;
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];//
                    ↪ Inferior esquerdo
83              geracaoAtual[l][(COLUNAS-1)-c] = geracaoAtual[l][c];//
                    ↪ Superior direito
84              geracaoAtual[(LINHAS-1)-l][(COLUNAS-1)-c] =
                    ↪ geracaoAtual[l][c]; // Inferior direito
85          }
86      }
87
88      while(1){
89
90          // Faz a copia da geracao atual para a geracao anterior
```

```
91          for(l = 0; l < LINHAS; l++){
92              for(c = 0; c < COLUNAS; c++){
93                  proximaGeracao[l][c] = geracaoAtual[l][c];
94              }
95          }
96
97          for(l=0; l<LINHAS; l++){
98              for(c=0; c<COLUNAS; c++){
99                  if(proximaGeracao[l][c]==1){
100                     printf("%c", 219);
101                     Pintar(0, 0, 0, QUADRADO, geoIndex[l][c]);
102                 } else{
103                     printf("%c", proximaGeracao[l][c], ' ');
104                     Pintar(255, 255, 255, QUADRADO, geoIndex[l][c])
                            ↪ ;
105                 }
106             }
107             printf("\n");
108         }
109          Desenha1Frame();
110
111         // Conta a quantidade de vizinhos de uma celula
112         for (l = 1; l < LINHAS-1; l++){
113             for(c = 1; c < COLUNAS-1; c++){
114                 ESTADO = proximaGeracao[l][c];
115                 soma = 0;
116                 for(r = l-1; r < l+2; r++)
117                     for(s = c-1; s < c+2; s++)
118                             soma+=proximaGeracao[r][s];
119
120                 if(proximaGeracao[l][c] == VIVO) soma--;
121
122                  //Define se uma celula deve morrer, permanecer
                        ↪ viva ou nascer
123                 if ((ESTADO == VIVO && soma < 2) || (ESTADO == VIVO
                        ↪  && soma > 3)){
124                     geracaoAtual[l][c] = MORTO;
125                 }else if ((ESTADO == VIVO && (soma > 2 || soma <=
                        ↪ 3)) || (ESTADO == MORTO && soma == 3)){
126                     geracaoAtual[l][c] = VIVO;
127                 }
128             }
129         }
130         waitFor (1);
131         system("cls");
132     }
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

Exercício 5.2

Esta prática, assim como o exercício 4.2, exercita conceitos de processamento de imagens, com manipulação de componentes RGB, agora distribuídos em uma matriz. Os componentes RGB são extraídos de uma imagem bitmap de 24 bits, e sua visualização também é similar ao exercício 4.2.

Listagem 5.2: Código fonte do filtro motion blur

```
1  #include <playAPC/playapc.h>
2
3  int main(){
4      int R[100][100], G[100][100], B[100][100];
5      int Raux[102][102], Gaux[102][102], Baux[102][102];
6      int Rf[102][102], Gf[102][102], Bf[102][102];
7      int soma;
8      int mfiltro = 8;
9      Ponto p;
10
```

```
11      //inicializa matrizes
12      for(int i = 0; i < 102; i++){
13          for(int j = 0; j < 102; j++){
14              Raux[i][j] = 0;
15              Gaux[i][j] = 0;
16              Baux[i][j] = 0;
17
18              Rf[i][j] = 0;
19              Gf[i][j] = 0;
20              Bf[i][j] = 0;
21          }
22      }
23
24      ExtraiRGBdeBMP("Mario.bmp", 100, 100, R, G, B);
25
26      AbreJanela(300, 300, "Marios");
27      //PintarFundo(255, 255, 255);
28
29      //Passa as cores para aux
30      for(int i = 1; i < 101; i++){
31          for(int j = 1; j < 101; j++){
32              Raux[i][j] = R[i - 1][j - 1];
33              Gaux[i][j] = G[i - 1][j - 1];
34              Baux[i][j] = B[i - 1][j - 1];
35          }
36      }
37
38      //Realiza filtragem
39      for(int i = 1; i < 101; i++){
40          for(int j = 1; j < 101; j++){
41              soma = 0;
42              soma +=     Raux[i-1][j-1] + Raux[i-1][j] + Raux[i-1][j
                    ↪ +1]
43                      +   Raux[i][j-1] + Raux[i][j] + Raux[i][j+1]
44                      +   Raux[i+1][j-1] + Raux[i+1][j] + Raux[i+1][j
                            ↪ +1];
45              Rf[i][j] = soma/mfiltro;
46
47              soma = 0;
48              soma +=     Gaux[i-1][j-1] + Gaux[i-1][j] + Gaux[i-1][j
                    ↪ +1]
49                      +   Gaux[i][j-1] + Gaux[i][j] + Gaux[i][j+1]
50                      +   Gaux[i+1][j-1] + Gaux[i+1][j] + Gaux[i+1][j
                            ↪ +1];
51              Gf[i][j] = soma/mfiltro;
52
53              soma = 0;
54              soma +=     Baux[i-1][j-1] + Baux[i-1][j] + Baux[i-1][j
                    ↪ +1]
55                      +   Baux[i][j-1] + Baux[i][j] + Baux[i][j+1]
56                      +   Baux[i+1][j-1] + Baux[i+1][j] + Baux[i+1][j
                            ↪ +1];
57              Bf[i][j] = soma/mfiltro;
58
59          }
60      }
```

```
61
62      p.y = -50;
63      for(int i = 1; i < 101; i++){
64          p.x = -100;
65          for(int j = 1; j < 101; j++){
66              CriaQuadrado(1, p);
67              Pintar(Raux[i][j], Gaux[i][j], Baux[i][j]);
68
69              p.x++;
70          }
71          p.y++;
72      }
73
74      p.y = -50;
75      for(int i = 1; i < 101; i++){
76          p.x = 0;
77          for(int j = 1; j < 101; j++){
78              CriaQuadrado(1, p);
79              Pintar(Rf[i][j], Gf[i][j], Bf[i][j]);
80
81              p.x++;
82          }
83          p.y++;
84      }
85
86      Desenha();
87  }
```

```
void ExtraiRGBdeBMP(const char *imagepath, int largura, int altura,
  int (&R)[tam_x][tam_y], int (&G)[tam_x][tam_y], int (&B)[tam_x][tam_y])
```

A função ExtraiRGBdeBMP, na linha 24, extrai os componentes RGB de uma imagem e retorna nas matrizes R, G e B. Seu primeiro argumento é o caminho da imagem, o segundo e o terceiro argumento se referem as dimensões da imagem, largura e altura, o quarto, quinto e sexto argumento são as matrizes que irão receber os componentes da imagem, respectivamente R, G e B.

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.

Pré-requisitos

As práticas deste capítulo exigem que sejam utilizadas as funções

- `void LimpaDesenho()`
- `void ApagaGrupo(int index)`

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 novas posições para os retângulos e execute novamente o procedimento de pedir valores do usuário e exibir a trajetória balística.

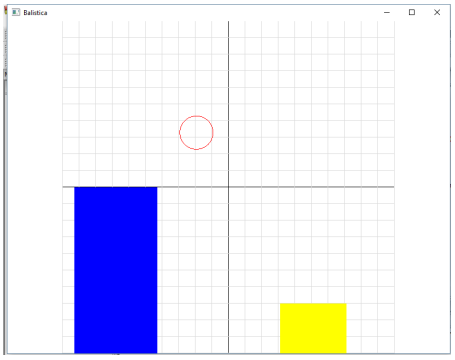
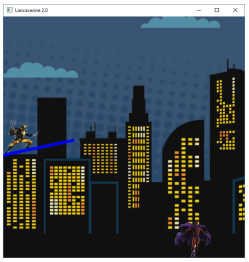
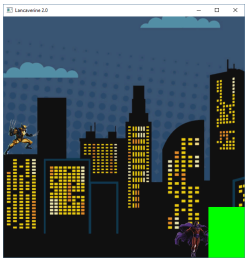


Figura 6.1: Lançador Balístico

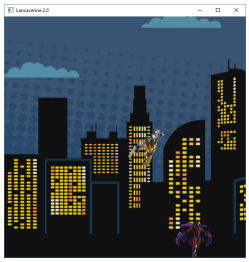
6.2. Baseado no exercício 6.1, faça o Wolverine ser lançado de um prédio, dado um ângulo e uma velocidade inicial, com o objetivo de atacar o Magneto. Se não atingir o Magneto, os dois inimigos se encaram. Se atingir o Magneto, faça o Magneto contra-atacar o Wolverine, lançando o mutante em uma trajetória retilínia na direção contrária ao ataque.



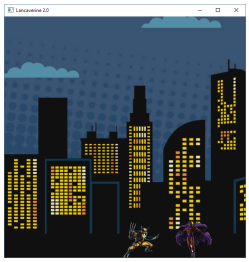
(a) Determinação do ângulo



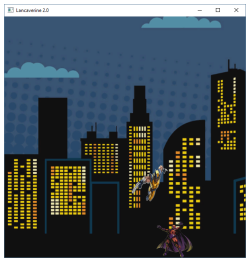
(b) Determinação da velocidade inicial



(c) Wolverine sendo lançado



(d) Wolverine e Magneto se encarando



(e) Wolverine sendo lançado pelo Magneto

6.3. 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

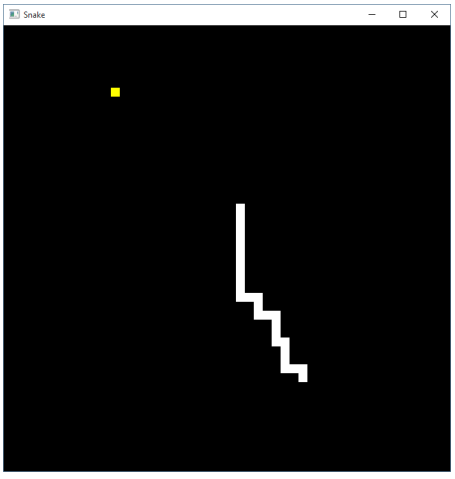


Figura 6.3: Jogo Snake

Soluções

Exercício 6.1

Esta prática ilustra como a função LimpaDesenho é usada para poder redesenhar outras cenas.

Listagem 6.1: Código fonte do lançador balístico

```
#include <playAPC/playapc.h>
#include <time.h>

int colisaoBolinhaRetangulo(int raio, int largurapredio, Ponto alvo
    ↪ , Ponto partida, Ponto movimentacao){

    if( movimentacao.y - raio <= alvo.y
        && movimentacao.y - raio > alvo.y - 1      //tolerancia
            ↪ para pousar no eixo y
        && movimentacao.x + raio < alvo.x + largurapredio
        && movimentacao.x >= alvo.x
        )
        return 1;
    else if(fabs(movimentacao.x + raio) > 100 || fabs(movimentacao.
        ↪ y + raio) > 100)
        return 2;
    else
        return 0;

}

int main(){
    Ponto p, alvo, partida;
    float theta, v0, t;
    int colisao = 1, resposta, bolinha;

    AbreJanela(800,600, "Balistica");
    srand(time(NULL));
    do{
        MostraPlanoCartesiano(10);
        PintarFundo(255, 255, 255);
    //Predio Azul///
        CriaGrupo(); //Separa os grupos - este e o grupo dos
            ↪ predios
            if(colisao == 1) //Comeca uma nova fase
                p.x = rand()%20 - 100; //-100 para comecar do lado
                    ↪ esquerdo da tela
            else //Repete fase anterior
                p.x = partida.x - 25;
            p.y = 0;
            CriaRetangulo(50, -100, p);
            Pintar(0, 0, 255);
    ////////////////////

    //Predio Amarelo//

```

```
42              if(colisao == 1) //Comeca uma nova fase
43                  alvo.x = (p.x) + rand()%50 + 100;
44              else //Repete a fase anterior
45                  alvo.x = alvo.x;
46              alvo.y = -70;
47              CriaRetangulo(40, -90, alvo);
48              Pintar(255, 255, 0);
49      ////////////////////
50
51      //Bolinha//
52          bolinha = CriaGrupo(); //Separa os grupos - este e o grupo
                ↪ da bolinha
53              partida.x = p.x + 25; //25 para estar no meio de um
                    ↪ predio de largura 50
54              partida.y = 10;
55              CriaCircunferencia(10, partida);
56              Pintar(255, 0, 0);
57      ////////////////////
58
59          Desenha1Frame();
60
61          printf("Angulo: ");
62          scanf("%f", &theta);
63          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) *
                        ↪ (9.8) * (t*t));
73                  Move(p, bolinha);
74                  t += 0.01; //tempo
75              }
76
77              colisao = colisaoBolinhaRetangulo(10, 40, alvo, partida
                    ↪ , p);
78
79              Desenha1Frame();
80          }while(!colisao);
81          //printf("Tipo de colisao: %d\n", colisao);
82
83          LimpaDesenho(); //Limpo o desenho para comecar uma nova
                ↪ fase
84          if(colisao == 1)
85              printf("\nYay! Quer jogar um novo jogo?");
86          else
87              printf("\nOh nao... Quer tentar de novo?");
88          printf("\n1 - Sim\n0 - Nao\n");
89          scanf("%d", &resposta);
90
91      }while(resposta);
```

```
92
93      Desenha();
94
95      return 0;
96  }
```

`void LimpaDesenho()`

A função LimpaDesenho, na linha 83, destrói todas as geometrias e retorna toda a playAPC para o estado padrão, com exceção dos limites do plano cartesiano.

Exercício 7.3a

Esta prática exercita o conceito de animação, assim como o exercício 3.5. Porém, ela também ilustra como utilizar o conceitos de grupos na playAPC de forma mais eficiente, liberando espaço na memória quando um grupo não é mais utilizado, como o grupo lancar, na Listagem 6.2.

Listagem 6.2: Código fonte do Lançaverine

```
1  #include <playAPC/playapc.h>
2  #include <time.h>
3
4  #define TAMANHOQUADRADO 30
5  int colisaoBolinhaRetangulo(int raio, int largurapredio, Ponto alvo
       ↪ , Ponto movimentacao){
6
7      if( movimentacao.y <= alvo.y + largurapredio
8         && movimentacao.x + raio <=  alvo.x + largurapredio
9         && movimentacao.x + raio >= alvo.x)
10          return 1;
11     else if(movimentacao.x > 100 || movimentacao.y < -100)
12         return 2;
13     else
14         return 0;
15
16 }
17
18 void defineCenario(Ponto *alvo, Ponto *partida, int *
       ↪ grupo_wolverine, int imgbackground, int imgmagneto_standing,
       ↪  int imgwolverine_standing, int colisao){
19     Ponto p;
20
21     p.x = -100;
22     p.y = -100;
23     CriaRetangulo(400, 264, p);
24     Pintar(255, 255, 255);
25     AssociaImagem(imgbackground);
26
27     if(colisao == 1) //Comeca uma nova fase
28         p.x = rand()%20 - 100; //-100 para comecar do lado esquerdo
               ↪  da tela
```

```
29      else //Repete fase anterior
30          p.x = (*partida).x - 25;
31
32      ////////////////////////////////////Magneto
            ↪ ////////////////////////////////////
33      CriaGrupo();
34
35      if(colisao == 1) //Comeca uma nova fase
36          (*alvo).x = (p.x) + rand()%50 + 100;
37      else //Repete a fase anterior
38          (*alvo).x = (*alvo).x;
39      (*alvo).y = -100;
40
41      CriaQuadrado(TAMANHOQUADRADO, *alvo);
42      Pintar(255, 255, 255);
43      AssociaImagem(imgmagneto_standing);
44      //
            ↪ ////////////////////////////////////////////////////////////////////////////////
            ↪
45
46      ////////////////////////////////////Wolverine
            ↪ ////////////////////////////////////
47      *grupo_wolverine = CriaGrupo();
48      (*partida).x = -100;
49      (*partida).y = -15;
50      CriaQuadrado(TAMANHOQUADRADO, *partida);
51      Pintar(255, 255, 255);
52      AssociaImagem(imgwolverine_standing);
53      //
            ↪ ////////////////////////////////////////////////////////////////////////////////
            ↪
54  }
55
56  float setaAngulo(Ponto partida){
57      Ponto p, p2;
58      float theta = 0;
59      int grupo_lancar;
60
61      printf("\nPressione a tecla setinha pra cima ou setinha pra
            ↪ baixo para mudar o angulo\n");
62
63      grupo_lancar = CriaGrupo(); //Setinha que ira se mover. Salvei
            ↪ seu indice pois pretendo exclui-la depois
64      p.x = 0;
65      p.y = 0;
66      p2.x = 60;
67      p2.y = 0;
68      CriaReta(p , p2); //Cria-lo na origem para nao dar problema com
            ↪  as coordenadas
69      //Materia de algebra linear
70
71      Pintar(0, 0, 255);
72      Grafite(10);
73      p = partida;
74
75      Move(p); //Move-lo para o lugar certo
```

```
76      do{
77          Desenha1Frame();
78
79          if(ApertouTecla(GLFW_KEY_UP)){
80              theta++;
81              printf("\nAngulo: %f", theta);
82              Gira(theta);
83          }
84          else if(ApertouTecla(GLFW_KEY_DOWN)){
85              theta--;
86              printf("\nAngulo: %f", theta);
87              Gira(theta);
88          }
89
90      }while(!ApertouTecla(GLFW_KEY_ENTER));
91      ApagaGrupo(grupo_lancar); //Apaga setinha do angulo
92
93      return theta;
94  }
95
96  float setaVelocidade(){
97      int grupo_lancar;
98      float v0;
99      Ponto p;
100
101     printf("\nPressione a tecla setinha pra cima ou setinha pra
            ↪ baixo para mudar a velocidade inicial\n");
102
103     grupo_lancar = CriaGrupo();
104     p.x = 70;
105     p.y = 0; //Y como 0 para nao confundir as coordenadas
106     //Materia de algebra linear
107
108     CriaRetangulo(30, 1, p); //Altura como 1 para poder
            ↪ redimensiona-lo depois
109     Pintar(0, 255, 0);
110     p.x = 70;
111     p.y = -100;
112     Move(p); //Move-lo para o lugar certo
113
114     do{
115         Desenha1Frame();
116
117         if(ApertouTecla(GLFW_KEY_UP)){
118             v0 += 10;
119             Redimensiona(1, v0);
120             printf("\nVelocidade: %f", v0);
121         }
122         else if(ApertouTecla(GLFW_KEY_DOWN)){
123             v0 -= 10;
124             Redimensiona(1, v0);
125             printf("\nVelocidade: %f", v0);
126         }
127
128     }while(!ApertouTecla(GLFW_KEY_ENTER));
129     ApagaGrupo(grupo_lancar); //Apaga retangulo de velocidade
```

```
130 |
131 |     return v0;
132 | }
133 |
134 | int main(){
135 |     Ponto p, alvo, partida;
136 |     float theta, v0, t;
137 |     int colisao = 1, resposta;
138 |     int imgmagneto_attack, imgmagneto_standing,
    |         ↪ imgmagneto_standing2,
139 |         imgwolverine_flying, imgwolverine_standing,
    |             ↪ imgwolverine_standing2,
140 |         imgbackground, imgpikachu;
141 |     int grupo_wolverine, grupo_lancar;
142 |
143 |     AbreJanela(600,600, "Lancaverine 2.0");
144 |
145 |     imgmagneto_attack = AbreImagem("Xmen/magneto_attack.png");
146 |     imgmagneto_standing = AbreImagem("Xmen/magneto_standing.png");
147 |     imgmagneto_standing2 = AbreImagem("Xmen/magneto_standing2.png")
    |         ↪ ;
148 |     imgwolverine_flying = AbreImagem("Xmen/wolverine_flying.png");
149 |     imgwolverine_standing = AbreImagem("Xmen/wolverine_standing.png
    |         ↪ ");
150 |     imgwolverine_standing2 = AbreImagem("Xmen/wolverine_standing2.
    |         ↪ png");
151 |     imgbackground = AbreImagem("Xmen/background.jpg");
152 |     imgpikachu = AbreImagem("Xmen/pikachu.png");
153 |
154 |     srand(time(NULL));
155 |     do{
156 |
157 |
158 |         LimpaDesenho(); //Limpo o desenho para comecar uma nova
    |             ↪ fase
159 |         defineCenario(&alvo, &partida, &grupo_wolverine,
    |             ↪ imgbackground, imgmagneto_standing,
    |             ↪ imgwolverine_standing, colisao);
160 |
161 |         Desenha1Frame();
162 |
163 |         theta = setaAngulo(partida);
164 |         v0 = setaVelocidade();
165 |
166 |         ApagaGrupo(grupo_wolverine); //vamos resetar o wolverine
    |             ↪ para ataque
167 |
168 |         grupo_wolverine = CriaGrupo(); //Separa os grupos - este e
    |             ↪ o grupo da bolinha
169 |             CriaQuadrado(TAMANHOQUADRADO, partida);
170 |             Pintar(255, 255, 255);
171 |             AssociaImagem(imgwolverine_flying);
172 |
173 |         t = 0;
174 |         theta = theta * PI/180;
175 |         colisao = 0;
```

```
176         do{
177             if(!colisao){
178                 p.x = partida.x + v0 * cos(theta) * t;
179                 p.y = partida.y + v0 * sin(theta) * t - ((1/2.0) *
                        ↪ (9.8) * (t*t));
180                 Move(p, grupo_wolverine);
181                 t += 0.1; //tempo
182             }
183 
184             colisao = colisaoBolinhaRetangulo(TAMANHOQUADRADO,
                    ↪ TAMANHOQUADRADO, alvo, p);
185 
186             Desenha1Frame();
187         }while(!colisao);
188 
189         if(colisao == 1){ //atingiu Magneto
190             ApagaGrupo(grupo_wolverine); //remove Wolverine
191             AssociaImagem(imgmagneto_attack); //Modifica Magneto
192 
193             grupo_wolverine = CriaGrupo(); //Separa os grupos -
                    ↪ este e o grupo da bolinha
194             CriaQuadrado(TAMANHOQUADRADO, p);
195             Pintar(255, 255, 255);
196             AssociaImagem(imgwolverine_flying);
197 
198             t = 0;
199             do{ //pausa dramatica
200                 Desenha1Frame();
201                 t += 0.1;
202             }while(t < 10);
203 
204             do{ //lanca wolverine
205                 Move(p, grupo_wolverine);
206                 p.x -= 10;
207                 p.y += 10;
208 
209                 Desenha1Frame();
210             }while(p.x + TAMANHOQUADRADO > -130);
211             ApagaGrupo(grupo_wolverine);
212 
213             AssociaImagem(imgmagneto_standing);
214 
215             CriaGrupo();
216                 p.x = -80;
217                 p.y = 32;
218                 CriaQuadrado(20, p);
219                 Pintar(255, 255, 255);
220                 AssociaImagem(imgpikachu);
221 
222             Desenha1Frame();
223         }
224 
225         if(colisao == 2){//Nao atingiu magneto
226             if(p.x > alvo.x){ //Wolverine esta na frente do Magneto
227                 AssociaImagem(imgwolverine_standing2);
228             }
```

```
229             else{ //Wolverine esta nas costas do Magneto
230                 ApagaGrupo(grupo_wolverine);
231
232                 AssociaImagem(imgmagneto_standing2);
233                 grupo_wolverine = CriaGrupo(); //Separa os grupos -
                        ↪  este e o grupo da bolinha
234                 CriaQuadrado(TAMANHOQUADRADO, p);
235                 Pintar(255, 255, 255);
236                 AssociaImagem(imgwolverine_standing);
237             }
238
239             t = 0;
240             do{ //pausa dramatica
241                 Desenha1Frame();
242                 t += 0.1;
243             }while(t < 10);
244         }
245
246         if(colisao == 1){
247             printf("\nYay! Quer jogar um novo jogo?");
248         }
249         else{
250             printf("\nOh nao... Quer tentar de novo?");
251         }
252         printf("\n1 - Sim\n0 - Nao\n");
253         scanf("%d", &resposta);
254
255     }while(resposta);
256
257     Desenha();
258
259     return 0;
260 }
```

`void ApagaGrupo(int index)`

A função ApagaGrupo, na linha 91, destrói todas as geometrias de um determinado grupo. Seu argumento é o índice do grupo, retornado pela função CriaGrupo

Exercício 6.3

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.

Listagem 6.3: Código fonte de Snake

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <playAPC/playapc.h>
```

[1] http://playapc.zaghetto.com/funcoes/extras/input/tecla-pressionada

```
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     Ponto p;
27
28     printf("Aguarde enquanto o jogo inicializa\n");
29
30     p.y = TAM/2 - 1;
31     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
   |     ↪ direcao){
94 |     int comeu = 0;
95 |
96 |     Pintar(255, 255, 255, QUADRADO, m[pos_i][pos_j].index);
97 |     m[pos_i][pos_j].direcao = direcao;
98 |
99 |     if(m[pos_i][pos_j].tipo == COMIDA)
100|         comeu = 1;
101|
102|     m[pos_i][pos_j].tipo = CABECA;
103|
104|     return comeu;
105| }
106|
107| void updateRastro(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
108|     Pintar(0, 0, 0, QUADRADO, m[pos_i][pos_j].index);
109|     m[pos_i][pos_j].tipo = VAZIO;
110| }
```

```
111
112 void sorteiaComida(tipoCelula m[TAM][TAM]){
113
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     Pintar(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
          ↪ quadrado
127     int pos_i = TAM/2; //posicao i da cabeca
128     int pos_j = TAM/2; //posicao j da cabeca
129     int aberto;
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     PintarFundo(255, 0, 0);
138     //MostraPlanoCartesiano(5);
139
140     inicializaMatriz(m, pos_i, pos_j);
141
142     sorteiaComida(m);
143
144     while(1){
145         int npos_i = pos_i, npos_j = pos_j;
146
147         updateTeclado(&direcao, &npos_i, &npos_j);
148         m[pos_i][pos_j].direcao = direcao; //ultima posicao da
              ↪ cabeca recebe direcao que cabeca foi
149         if(!bateu(m, npos_i, npos_j)){
150             int comeu;
151
152             comeu = updateCabeca(m, npos_i, npos_j, direcao);
153             if(comeu)
154                 sorteiaComida(m);
155             else{
156                 updateRastro(m, rpos_i, rpos_j);
157                 atualizaPosicao(m[rpos_i][rpos_j].direcao, &rpos_i,
                      ↪ &rpos_j);
158                 m[rpos_i][rpos_j].tipo = CORPO;
159             }
160
161             pos_i = npos_i;
162             pos_j = npos_j;
163
```

```
164             printf("C(%d, %d)\t R(%d, %d)\n", pos_i, pos_j, rpos_i,
                    ↪  rpos_j);
165         }
166         else
167             break;
168
169         aberto = Desenha1Frame();
170         if(!aberto)
171             break;
172     }
173
174     printf("O jogo acabou!\n");
175
176     if(aberto)
177         Desenha();
178
179     return 0;
180 }
```

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. Construa uma árvore binária recursiva onde o usuário oferece altura, profundidade e ângulo entre ramos. A árvore deverá ser construída tanto para o lado esquerdo, com ângulo entre os galhos de θ fornecida pelo usuário, quanto para o lado direito, com ângulo $-\theta$.

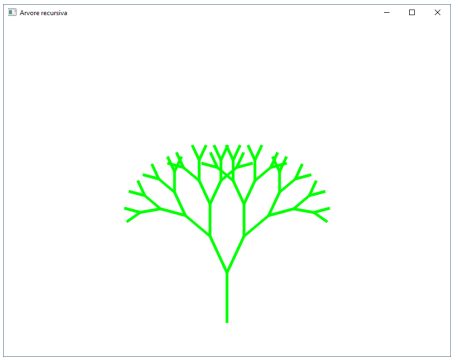


Figura 7.1: Árvore Binária com altura 30, ângulo 25 e profundidade 6

7.2. Torre de Hanói é um jogo matemático onde seu objetivo é passar os discos de uma torre A para uma torre B, utilizando uma torre C como auxiliar. Ele segue as seguintes regras:

(a) Só pode mover um disco de cada vez

(b) Só pode mover o disco que estiver mais acima de uma torre e deve-se colocar no topo de outra torre

(c) Não é permitido colocar um disco de tamanho maior em cima de um disco de tamanho menor

Ilustre a resolução da torre de Hanói para uma quantidade de 5 discos ou menos.

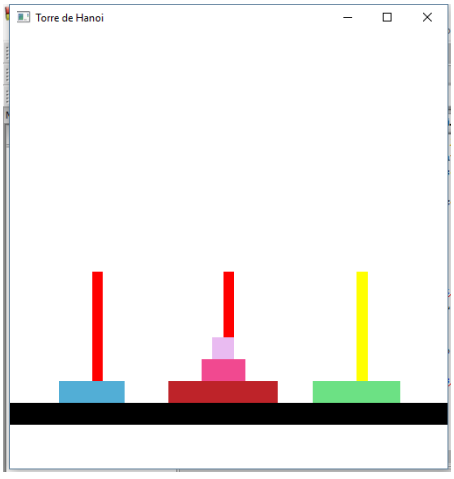


Figura 7.2: Solucionador da torre de Hanói

7.3. Sabe-se que a curva de Koch começa sendo construída com um segmento de reta de tamanho $\frac{n}{3}$. Na extremidade deste primeiro segmento, desenha-se outro segmento de reta de tamanho $\frac{n}{3}$, porém com uma curvatura de 60 em relação ao primeiro. Na extremidade deste segundo segmento, desenha-se outra curva de reta de tamanho $\frac{n}{3}$, mas agora com a curvatura de 120 em relação ao primeiro. Por fim, desenha-se outro segmento de reta de tamanho $\frac{n}{3}$ na extremidade do terceiro segmento, sem diferença de curvatura.

(a) Exiba a curva de Koch de ordem i com tamanho n igual à 180 unidades centrada na posição $(-100, 0)$.

(NOTA: a partir de um certo nível de recursão, é possível que a divisão dos segmentos de retas comecem a dividir pixels, tornando inviável a visualização)

Como sugestão, a Listagem 7.1 exemplifica um cabeçalho da função koch, onde seu primeiro argumento é o ponto de origem, seu segundo argumento é o tamanho do segmento de reta, seu terceiro argumento é a inclinação do segmento e seu quarto argumento é a variável que controla a profundidade de recursão, a qual impede que entre em loop infinito. A função retorna o segundo ponto da reta naquele nível de recursão que deverá ser criada.

Listagem 7.1: Header da função koch

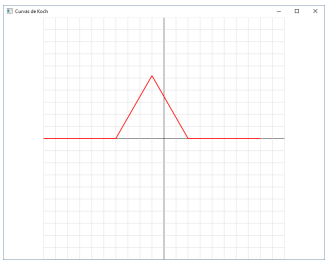
```
Ponto koch (Ponto from, double len, double theta, int order)
```

(b) Utilize a curva de Koch criada no exercício 7.3a para criar outras duas curvas, com uma diferença de angulação: a primeira curva será criada da mesma maneira no exercício 7.3a; a segunda curva será inclinada em 120 e; a terceira curva será inclinada em -120.

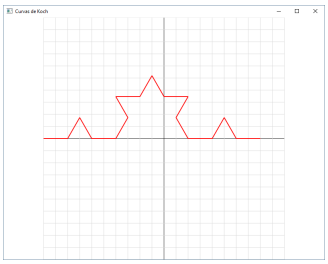
As Figuras 7.4 ilustram as três retas, onde a primeira curva, com inclinação 0, está representada em vermelho, a segunda, com inclinação de 120 está representada em verde e a terceira curva, com inclinação -120 está representada em azul.

7.4. A curva de Sierpiński é uma curva do tipo space-filling curve, a qual significa que ela tenta ocupar todo espaço disponível sem ser cruzar. Ela pode ser implementada utilizando quatro funções mutuamente recursivas, A, B, C e D. A Figura 7.5 ilustra essa curva com as 4 curvas básicas. A curva A está representado em vermelho, a curva B em verde, a curva C em azul e a curva D em amarelo, sendo as curvas que conectam estas 4 curvas básicas representadas em preto.

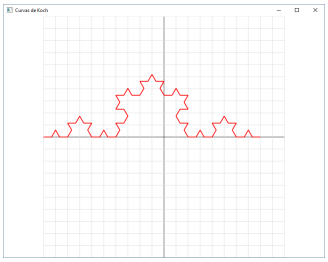
- O programa inicia com uma curva básica S dada pelo padrão A ↘ , B ↙ , C ↖ e D ↗ . As setam indicam a virada do ângulo que as retas fecham os desenhos das funções.



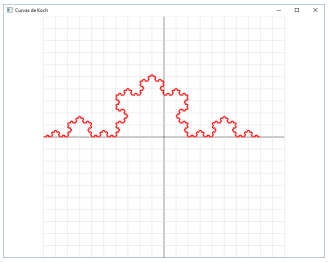
(a) Curva de Koch de ordem 1



(b) Curva de Koch de ordem 2

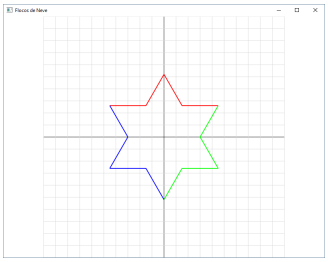


(c) Curva de Koch de ordem 3

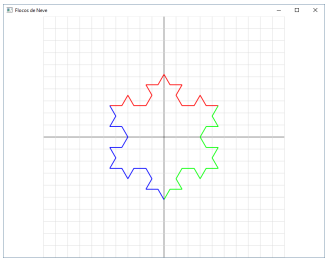


(d) Curva de Koch de ordem 6

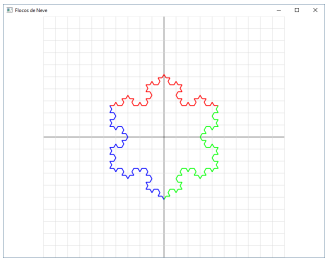
Figura 7.3: Curva de Koch



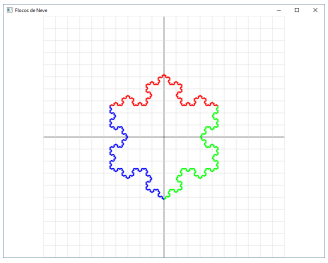
(a) Floco de neve de ordem 1



(b) Floco de neve de ordem 2



(c) Floco de neve de ordem 3



(d) Floco de neve de ordem 6

Figura 7.4: Floco de neve

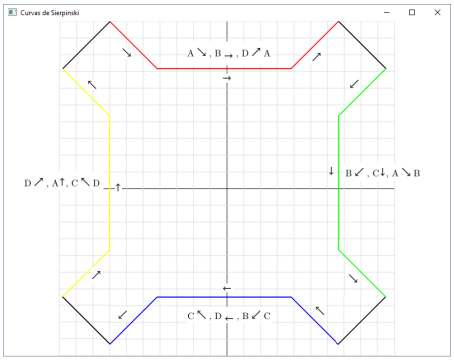


Figura 7.5: Curva de Sierpiński de ordem 1 com ângulo de 45

- A curva A é dada pelo padrão A ↘ , B → , D ↗ A
- A curva B é dada pelo padrão B ↙ , C↓, A ↘ B
- A curva C é dada pelo padrão C ↖ , D ← , B ↙ C
- A curva D é dada pelo padrão D ↗ , A↑, C ↖ D

Considere o ângulo de inclinação da curva de Sierpiński como 45, ou seja, as retas desenhadas tem uma inclinação de 45 entre si. Considere também que a curva tenha uma altura h de 40 unidades e que, caso a ordem de recusão n seja maior que 0, a altura será determinada por $\frac{h}{n^2}$. A criação das curvas se inicia no ponto p em $(-70, 100)$.

Considere que a Listagem 7.2 seja a função que calcula o próximo ponto baseando no ângulo de inclinação e desenha esta reta. Estes fatores são passados de acordo com a Listagem 7.3.

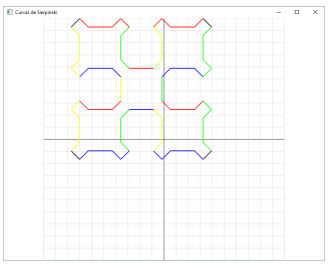
Listagem 7.2: Criação da Curva de Sierpiński

```
Ponto reta(int fator, int h, Ponto p) {
  // ang é para multiplicar por 45 graus
  Ponto p1;
  double ar = fator * 45.0 * PI/180;

  p1.x = p.x + h*cos(ar);
  p1.y = p.y + h*sin(ar);
  CriaReta(p,p1);
  Grafite(2);
  Pintar(0,0,255);
  return p1;
}
```

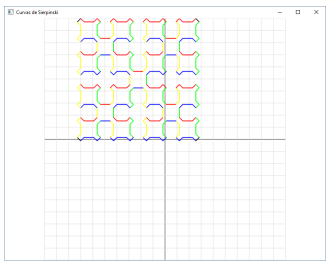
Listagem 7.3: Função main Curva de Sierpiński

```
int main() {

  Ponto p;
  int   k;
  float   h = 40;

  printf("Insira a ordem para a criacao das curvas: ");
  scanf("%d", &k);

  p.x = -70; p.y = 100; //ordem=1 cabe todo na tela


  MostraPlanoCartesiano(10);
  AbreJanela(800,600, "Curvas de Sierpinski");
  PintarFundo(255, 255, 255);


  if (k>0) h /= k*k;
  p = A(k,h, p);
  p = reta(7,h, p); //calcula o proximo ponto onde a outra reta deve comecar

  p = B(k,h, p);
  p = reta(5,h, p);

  p = C(k,h, p);
  p = reta(3,h, p);

  p = D(k,h, p);
  p = reta(1,h, p);

  Desenha();

  return 0;
}
```

Implemente as funções A, B, C e D da curva de Sierpiński.



(a) Curva de Sierpiński de ordem 2



(b) Curva de Sierpiński de ordem 3

Figura 7.6: Curva de Sierpiński

Soluções

Exercício 7.1

Esta prática reforça que os valores da estrutura Ponto podem ser ponto flutuantes.

Listagem 7.4: Árvore recursiva

```
#include <playAPC/playapc.h>

#define RAD 0.01745

void arvore(Ponto p1, float altura1, float angulo1, float
    ↪ angulo_diff, float profundidade){
    Ponto p2;
    float altura2, angulo2;

    if(0 < profundidade){
        p2.x = p1.x + altura1 * cos(angulo1 * RAD);
        p2.y = p1.y + altura1 * sin(angulo1 * RAD);
        CriaReta(p1, p2);
        Pintar(0, 255, 0);
        Grafite(5.0);

        altura2 = altura1 * 0.8;
        angulo2 = angulo1 + angulo_diff;
        arvore(p2, altura2, angulo2, angulo_diff, profundidade - 1)
            ↪ ;

        angulo2 = angulo1 - angulo_diff;
        arvore(p2, altura2, angulo2, angulo_diff, profundidade - 1)
            ↪ ;
    }

}

```

```
26 int main(){
27     Ponto p;
28     float altura, angulo, profundidade;
29
30     printf("Informe altura:");
31     scanf("%f", &altura);
32
33     printf("Informe angulo entre galhos:");
34     scanf("%f", &angulo);
35
36     printf("Informe profundidade da recursao:");
37     scanf("%f", &profundidade);
38
39     p.x = 0;
40     p.y = -80;
41
42     AbreJanela(800,600, "Arvore recursiva");
43     PintarFundo(255, 255, 255);
44
45     arvore(p, altura, 90, angulo, profundidade);
46
47     Desenha();
48
49     return 0;
50 }
```

Exercício 7.2

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 7.5: 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
       ↪ funcao CriaGrupo
15 //discoHanoi.label: qual a numeracao do disco, do menor para o
       ↪ maior
16 //discoHanoi.posicao: em qual posicao do plano cartesiano o disco
       ↪ esta
17 //discoHanoi.torre: em qual torre que ele esta
```

```
18 //discoHanoi.largura: largura do disco
19
20 //Aqui eu conto quantos discos tem em uma torre
21 int contDiscos(discoHanoi disco[], int discoindex, char torre, int
       ↪ numDiscos){
22     int i, total;
23
24     total = 0;
25     for(i = 0; i < numDiscos; i++){
26         if(i != discoindex){
27             if(disco[i].torre == torre)
28                 total++;
29         }
30     }
31     return total;
32 }
33
34 //Aqui eu movo os discos para a torre
35 void moveHanoi(int n, discoHanoi discos[], char torre, int
       ↪ numDiscos){
36     Ponto p;
37     int i, disco, discoindex, auxposicao;
38
39     /////////Procurando o disco que tenho que mover
40     for(i = 0; i < numDiscos; i++){
41         if(discos[i].label == n){
42             disco = discos[i].index; //O grupo que este disco
                   ↪ pertence
43             discoindex = i; //A posicao do vetor de discoHanoi que
                   ↪ este disco esta
44             break;
45         }
46     }
47
48     //Aqui eu movo na coordenada x o meu disco
49     if(torre == 'B'){
50         p.x = POSX_B - discos[i].largura/2;
51         discos[discoindex].torre = 'B';
52     }
53     else if (torre == 'C'){
54         p.x = POSX_C - discos[i].largura/2;
55         discos[discoindex].torre = 'C';
56     }
57     else{
58         p.x = POSX_A - discos[i].largura/2;
59         discos[discoindex].torre = 'A';
60     }
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         Desenha1Frame();
```

```
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);
               ↪                                  // H1
82         printf("\nmova disco %d de %c para %c\n", n, a, b);
83         moveHanoi(n, disco, b, numDiscos);
84
85         hanoi(n - 1, disco, c, b, a, numDiscos);
               ↪                                  // H2
86     }
87 }
88
89 int main(void){
90     int numDiscos, i;
91     discoHanoi disco[5];
92     Ponto p;
93
94     do{
95         printf("\nDigite uma quantidade de discos menor ou igual a
               ↪ 5: ");
96         scanf("%d", &numDiscos);
97     }while(numDiscos > 5 || numDiscos <= 0); //Pergunte a
           ↪ quantidade de discos de novo se o usuario colocar um
           ↪ numero maior que 5 ou menor ou igual a 0
98
99     AbreJanela(500, 500, "Torre de Hanoi");
100    PintarFundo(255, 255, 255);
101
102    ////////////////////////// Chao e torres
           ↪ //////////////////////////
103    CriaGrupo(); //Grupo das coisas que nao se mexem
104    //Chao
105    p.x = -100;
106    p.y = -80;
107    CriaRetangulo(200, 10, p);
108    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
                ↪  um grupo para poderem se mover independente
133         disco[i].largura = 10 * (numDiscos - i); //largura do disco
                ↪  pra torre ficar bonitinha
134         disco[i].posicao = p.y; //posicao original do disco
135         disco[i].torre = 'A'; //torre de partida
136         disco[i].label = numDiscos - i; //Label para os discos
137         p.x = POSX_A - disco[i].largura/2;
138 
139         CriaRetangulo(disco[i].largura, 10, p);
140         Pintar(rand()%256, rand()%256, rand()%256);
141 
142         p.y += 10;
143     }
144 
145     printf("Agora vou mostrar passo a passo como se resolve esta
            ↪ torre. Aperte enter para o proximo passo");
146     while(!ApertouTecla(GLFW_KEY_ENTER))
147         Desenha1Frame();
148 
149     hanoi(numDiscos, disco, 'A', 'B', 'C', numDiscos);
150 
151     printf("\nPronto! E assim que se resolve esta torre!");
152 
153     Desenha();
154     return 0;
155 }
```

Exercício 7.3

Item 7.3a Esta prática reforça o modo de utilização da função CriaReta, usando a função movaCaneta, na linha 10.

Listagem 7.6: 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  }
```

Item 7.3b Esta prática é uma evolução do exercício 7.3a.

Listagem 7.7: 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, int r,
       ↪  int g, int b) {
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(r,g,b);
28   }
29   else {
30     koch(from, len/3, theta, order-1, r, g, b);
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, r, g, b);
36
37     to = movaCaneta(from, theta+60, len/3);
38     from = to;
39     koch(from, len/3, theta-60, order-1, r, g, b);
40
41     to = movaCaneta(to, theta-60, len/3);
42     from = to;
43     koch(from, len/3, theta, order-1, r, g, b);
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   p0 = koch(p0, SIZE/2,    0, ordem, 255, 0, 0);
53   p0 = koch(p0, SIZE/2, -120, ordem, 0, 255, 0);
54   p0 = koch(p0, SIZE/2,  120, ordem, 0, 0, 255);
55 }
56
57
58 int main(void) {
59   int ordem = 6;
60
61
62   MostraPlanoCartesiano(10);
63   AbreJanela(LARGURA,ALTURA, "Flocos de Neve");
64   PintarFundo(255, 255, 255);
65
66   floco_de_neve(ordem);
67
68   Desenha();
69
70   return 0;
71 }
```

Exercício 7.4

Esta prática reforça o modo de utilizar a função CriaReta, usando as implementações das funções das linhas 16, 17, 18 e 19. Além disso, exercita fundamentalmente recursões mútuas entre 4 funções.

Listagem 7.8: Código fonte da curva de Sierpiński

```
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() {
79
80      Ponto p;
81      int   k;
82      float   h = 40;
83
84      printf("Insira a ordem para a criacao das curvas: ");
85      scanf("%d", &k);
86
87      p.x = -70; p.y = 100; //ordem=1 cabe todo na tela
88
89
90      MostraPlanoCartesiano(10);
91      AbreJanela(800,600, "Curvas de Sierpinski");
92      PintarFundo(255, 255, 255);
93
94
95      if (k>0) h /= k*k;
96      p = a(k,h, p); p = reta(7,h, p); //esta chamada da funcao reta
            ↪ calcula o proximo ponto onde a outra reta deve comecar
97      p = b(k,h, p); p = reta(5,h, p);
98      p = c(k,h, p); p = reta(3,h, p);
99      p = d(k,h, p); p = reta(1,h, p);
100
101     Desenha();
102
103 return 0;
104 }
```

PARTE V

EXERCÍCIOS EXTRAS

CAPÍTULO 8

EXERCÍCIOS EXTRAS

Resumo

Este capítulo reúne uma bateria de exercícios envolvendo todos os capítulos abordados durante o livro.

Problemas

8.1. A partir de um ponto no mapa, simule um geiser inundando o terreno montanhoso com água. Considere que:

- O terreno é construído randomicamente, onde sua altura varia entre 0 à 255.
- Se valor da altura igual a 0, o terreno é verde puro. Se for maior que 0, ele varia sua coloração de vermelho de modo que, ao atingir valor de altura 255, sua cor seja amarelo puro.
- A cada iteração, o valor de água sobe 10 unidades e espalha-se igualmente entre o terreno.

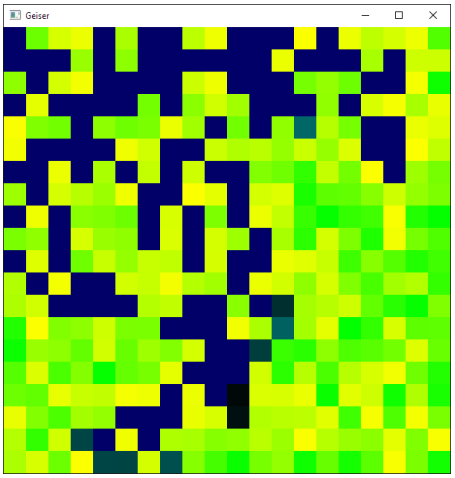


Figura 8.1: Simulador de inundação

8.2. Faça uma animação onde a lua está em órbita espiral em direção à terra. Ao atingir a terra, esta é jogada para fora da tela.

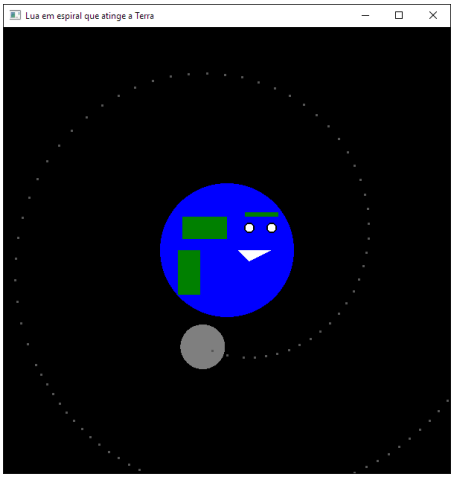


Figura 8.2: Lua em órbita espiral

8.3. Faça uma animação onde simule o funcionamento de um sonar, com os seguintes aspectos:

- Há três seres na cena: um alienígena, um caça e um ruído.
- O ruído realiza uma trajetória de um 8 deitado, ∞. Quando o ruído se aproxima do centro $(0, 0)$ do sonar, ele desaparece, reaparecendo na tela depois de um certo tempo.

- O caça realiza trajetória descrito pela Equação 8.1.

$$\begin{cases} 50\cos(\alpha)\sqrt{2\cos(2\alpha)} & = x \\ 50\sin(\alpha)\sqrt{2\cos(2\alpha)} & = y \end{cases} \tag{8.1}$$

  Onde α é incrementado durante o loop.

- O alienígena começa seu movimento, *dir* indo da esquerda para direita. Enquanto α é incrementado a cada iteração do loop, há um θ que irá variar entre 0 e 4π. Toda vez que θ atingir 4π, θ começa a variar de 0 e 4π novamente, invertendo a direção *dir* do alienígena. Sendo assim, a trajetória do alienígena é descrita pela Equação 8.2.

$$\begin{cases} 200\cos(\theta \times dir) \div \theta \times dir & = x \\ 200\sin(\theta \times dir) \div \theta \times dir & = y \end{cases} \tag{8.2}$$

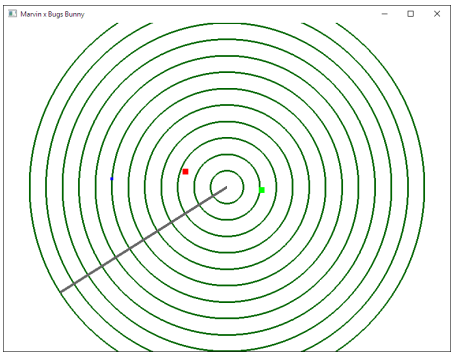


Figura 8.3: Sonar

Soluções

Exercício 8.1

Listagem 8.1: Simulador de inundação

```
1 #include <playAPC/playapc.h>
2 #include <time.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <limits.h>
6
7 #define TAM 20
8
9 typedef struct{
```

```
10      int i, j;
11      float valor;
12  }tipoVizinho;
13
14  void imprimeTerminal(float m[TAM][TAM], int comecoi, int comecoj){
15
16      printf("\n\n");
17      for(int i = 0; i < TAM; i++){
18          for(int j = 0; j < TAM; j++){
19              if(i == comecoi && j == comecoj)
20                  printf("(%3.5f) ", m[i][j]);
21              else
22                  printf("%3.5f ", m[i][j]);
23          }
24          printf("\n");
25      }
26  }
27
28  void imprimeTerminalInt(int m[TAM][TAM], int comecoi, int comecoj){
29
30      printf("\n\n");
31      for(int i = 0; i < TAM; i++){
32          for(int j = 0; j < TAM; j++){
33              if(i == comecoi && j == comecoj)
34                  printf("(%d) ", m[i][j]);
35              else
36                  printf("%3d ", m[i][j]);
37          }
38          printf("\n");
39      }
40  }
41
42  float getVizinho(float m[TAM][TAM], int i, int j){
43
44      if(i >= 0 && i < TAM){
45          if(j >= 0 && j < TAM){
46              return m[i][j];
47          }
48      }
49
50      return -1;
51  }
52
53  int getVizinhoInt(int m[TAM][TAM], int i, int j){
54
55      if(i >= 0 && i < TAM){
56          if(j >= 0 && j < TAM){
57              return m[i][j];
58          }
59      }
60
61      return -1;
62  }
63
64  void contaInunda(int mEnche[TAM][TAM], float m[TAM][TAM], float
        ↪ maior, float taxa, int qtd, int mExibe[TAM][TAM]){
```

```
65
66      if(taxa > 0){
67          taxa /= qtd;
68      }
69      for(int i = 0; i < TAM; i++){
70          for(int j = 0; j < TAM; j++){
71              if(mEnche[i][j] > 0){
72                  m[i][j] = maior + taxa;
73                  Pintar(0, 0, m[i][j], QUADRADO, mExibe[i][j]);
74                  mEnche[i][j] = 0;
75              }
76          }
77      }
78
79      printf("Nivel de agua: %f\n", maior+taxa);
80  }
81
82  void bubblesort(tipoVizinho v[], int tamanho){
83      short int f = 1;
84
85      while(f){
86          f = 0;
87          for(int i = 0; i < tamanho-1; i++){
88              if(v[i].valor > v[i+1].valor){
89                  tipoVizinho aux = v[i];
90
91                  v[i] = v[i+1];
92                  v[i+1] = aux;
93                  f = 1;
94              }
95          }
96      }
97  }
98
99  void setVizinhos(tipoVizinho v[8], int ordem[8], float m[TAM][TAM],
        ↪  int comecoi, int comecoj){
100
101     for(int i = 0; i < 8; i++)
102         ordem[i] = -1;
103
104     v[0].valor = getVizinho(m, comecoi - 1, comecoj - 1);
105     v[0].i = comecoi - 1;
106     v[0].j = comecoj - 1;
107
108     v[1].valor = getVizinho(m, comecoi, comecoj - 1);
109     v[1].i = comecoi;
110     v[1].j = comecoj - 1;
111
112     v[2].valor = getVizinho(m, comecoi + 1, comecoj - 1);
113     v[2].i = comecoi + 1;
114     v[2].j = comecoj - 1;
115
116     v[3].valor = getVizinho(m, comecoi + 1, comecoj);
117     v[3].i = comecoi + 1;
118     v[3].j = comecoj;
119
```

```
120     v[4].valor = getVizinho(m, comecoi + 1, comecoj + 1);
121     v[4].i = comecoi + 1;
122     v[4].j = comecoj + 1;
123
124     v[5].valor = getVizinho(m, comecoi, comecoj + 1);
125     v[5].i = comecoi;
126     v[5].j = comecoj + 1;
127
128     v[6].valor = getVizinho(m, comecoi - 1, comecoj + 1);
129     v[6].i = comecoi - 1;
130     v[6].j = comecoj + 1;
131
132     v[7].valor = getVizinho(m, comecoi - 1, comecoj);
133     v[7].i = comecoi - 1;
134     v[7].j = comecoj;
135     bubblesort(v, 8);
136 }
137
138 void inunda(float m[TAM][TAM], int mEnche[TAM][TAM], float *taxa,
    ↪ int comecoi, int comecoj, int *qtd, float *maior, int mExibe
    ↪ [TAM][TAM]){
139
140     tipoVizinho v[8];
141     int k = 0, ordem[8];
142
143     setVizinhos(v, ordem, m , comecoi, comecoj);
144     k = 0;
145
146     for(int i = 0; i < 8; i++){
147         if(v[i].valor == -1)
148             continue;
149
150         if(mEnche[v[i].i][v[i].j] != 1){
151
152             if(m[comecoi][comecoj] + *taxa > v[i].valor && v[i].
                ↪ valor > m[comecoi][comecoj]){
153                 float dif = (v[i].valor - m[comecoi][comecoj]);
154                 (*taxa) -= dif;
155                 if(v[i].valor > *maior)
156                     *maior = v[i].valor;
157
158                 m[comecoi][comecoj] = v[i].valor;
159                 if(*taxa > 0){
160                     (*qtd)++;
161                     ordem[k] = i;
162                     k++;
163                     mEnche[v[i].i][v[i].j] = 1;
164                 }
165             }
166             else if(v[i].valor < m[comecoi][comecoj]){
167                 float dif = (m[comecoi][comecoj] - m[v[i].i][v[i].j
                    ↪ ]);
168
169                 if((*taxa) - dif <= 0){
170                     m[v[i].i][v[i].j] += (*taxa);
171                     (*taxa) = 0;
```

```
172                     Pintar(0, m[v[i].i][v[i].j], m[v[i].i][v[i].j],
                            ↪ QUADRADO, mExibe[v[i].i][v[i].j]);
173                 }
174                 else{
175                     m[v[i].i][v[i].j] += dif;
176                     (*taxa) -= dif;
177                 }
178                 if(*taxa > 0){
179                     (*qtd)++;
180                     ordem[k] = i;
181                     k++;
182                     mEnche[v[i].i][v[i].j] = 1;
183                 }
184                 else{
185                     *taxa = 0;
186                 }
187             }
188             else if(v[i].valor == m[comecoi][comecoj]){
189                 (*qtd)++;
190                 ordem[k] = i;
191                 k++;
192                 mEnche[v[i].i][v[i].j] = 1;
193             }
194         }
195     }
196
197     for(int i = 0; i < k; i++){
198         inunda(m, mEnche, taxa, v[ordem[i]].i, v[ordem[i]].j, qtd,
                ↪ maior, mExibe);
199     }
200 }
201
202 int main(){
203     float mTerrenos[TAM][TAM];
204     int mEnche[TAM][TAM], mExibe[TAM][TAM];
205     int menor = INT_MAX;
206     int comecoi = 0, comecoj = 0;
207     float maior;
208     Ponto p;
209
210     AbreJanela(600, 600, "Geiser");
211     srand(time(NULL));
212     p.y = 90;
213     for(int i = 0; i < TAM; i++){
214         p.x = -100;
215         for(int j = 0; j < TAM; j++){
216             mTerrenos[i][j] = rand()%255;
217
218             mExibe[i][j] = CriaQuadrado(10, p);
219
220             Pintar(mTerrenos[i][j], 255, 0);
221             p.x += 10;
222
223             mEnche[i][j] = 0;
224
225             if(mTerrenos[i][j] < menor){
```

```
226                     menor = mTerrenos[i][j];
227                     comecoi = i;
228                     comecoj = j;
229                 }
230             }
231             p.y -= 10;
232         }
233         mEnche[comecoi][comecoj] = 1;
234         Pintar(255, 0, 0, QUADRADO, mExibe[comecoi][comecoj]);
235 
236         printf("****geracao inicial****\n");
237         while(!ApertouTecla(GLFW_KEY_ENTER))
238             Desenha1Frame();
239 
240         for(int geracao = 0; maior <= 255; geracao++){
241             maior = mTerrenos[comecoi][comecoj];
242             float taxa = 1;
243             int qtd = 1;
244             printf("\n****geracao %2d****\n", geracao+1);
245 
246             inunda(mTerrenos, mEnche, &taxa, comecoi, comecoj, &qtd, &
                    ↪ maior, mExibe);
247             contaInunda(mEnche, mTerrenos, maior, taxa, qtd, mExibe);
248             mEnche[comecoi][comecoj] = 1;
249 
250             Desenha1Frame();
251         }
252 
253         Desenha();
254 
255         return 0;
256 }
```

Exercício 8.2

Listagem 8.2: Lua em órbita espiral

```
1  #include <playAPC/playapc.h>
2  #include <math.h>
3 
4  int main(){
5 
6      float t, phi, dx, dy, d;
7      Ponto p, q, r;
8      int terra, lua, traj, boquinha; //grupos
9 
10     AbreJanela(600, 600, "Lua em espiral que atinge a Terra");
11     //MostraPlanoCartesiano(10);
12     PintarFundo(0, 0, 0);
13 
14 
15     terra = CriaGrupo();
16     p.x = 0;
17     p.y = 0;
18     CriaCirculo(30, p); Pintar(0, 0, 255);
```

```
19
20      //terrinhas
21      p.x = -22;
22      p.y = -20;
23      CriaRetangulo(10, 20, p); Pintar(0, 128, 0);
24      p.x = -20;
25      p.y = 5;
26      CriaRetangulo(20, 10, p); Pintar(0, 128, 0);
27      p.x = 8;
28      p.y = 15;
29      CriaRetangulo(15, 2, p); Pintar(0, 128, 0);
30      //olhinhos
31      p.x = 10;
32      p.y = 10;
33      CriaCirculo(2, p); Pintar(255, 255, 255);
34      p.x = 20;
35      p.y = 10;
36      CriaCirculo(2, p); Pintar(255, 255, 255);
37      p.x = 10;
38      p.y = 10;
39      CriaCircunferencia(2, p); Pintar(0, 0, 0);
40      p.x = 20;
41      p.y = 10;
42      CriaCircunferencia(2, p); Pintar(0, 0, 0);
43      p.x = 10;
44      p.y = 10;
45      CriaCircunferencia(2.2, p); Pintar(0, 0, 0);
46      p.x = 20;
47      p.y = 10;
48      CriaCircunferencia(2.2, p); Pintar(0, 0, 0);
49
50      //boquinha
51      p.x = 5;
52      p.y = 0;
53      q.x = 20;
54      q.y = 0;
55      r.x = 10;
56      r.y = -5;
57      CriaPoligono(3, p, q, r); Pintar(255, 255, 255);
58
59      lua = CriaGrupo();
60      p.x = 0;
61      p.y = 0;
62      CriaCirculo(10, p); Pintar(127, 127, 127);
63
64      //Cria grupo da trajetoria
65      traj = CriaGrupo();
66      q.x = (10 * 4*PI)*cos(4*PI);
67      q.y = (10 * 4*PI)*sin(4*PI);
68
69      CriaPonto(q); Grafite(2); Pintar(87, 87, 87); //Preciso criar o
            ↪ primeiro ponto
70      for (t = 4*PI; t > 0 ; t -= .05){
71          phi = 10 * t; //polar
72          p.x = phi*cos(t);
73          p.y = phi*sin(t);
```

```
74
75          Move(p, lua);
76
77          dx = q.x - p.x;
78          dy = q.y - p.y;
79          d = sqrt(dx*dx + dy*dy);
80
81          if(d > 5){ //Distancia para intercalar pontos
82              CriaPonto(p); Grafite(3); Pintar(87, 87, 87);
83              q = p;
84          }
85
86          dx = p.x;
87          dy = p.y;
88          d = sqrt(dx*dx + dy*dy);
89          if(d <= 30 + 10) //Se distancia entre os centro da lua e da
                ↪  terra for menor que a soma dos raios, colidiu
90              break;
91
92          Desenha1Frame();
93      }
94      p.x += 30; //mais raio da terra
95      p.y += 30; //mais raio da terra
96
97      //cria um grupo separado pra boquinha
98      boquinha = CriaGrupo();
99      q.x = 12;
100     q.y = -5;
101     CriaCirculo(10, q); Pintar(0, 0, 255);
102     q.x = 15;
103     q.y = 2;
104     CriaCirculo(2.5, q); Pintar(255, 255, 255);
105     Redimensiona(1, 1.5, boquinha);
106
107     q.x = 15;
108     q.y = -6;
109     for(t = 0; t < 100; t+= .05){
110         p.x += t;
111         p.y += t;
112
113         Move(p, terra);
114
115         q.x += t;
116         q.y += t;
117         Move(q, boquinha);
118         Desenha1Frame();
119     }
120
121     Desenha();
122
123     return 0;
124 }
```

Exercício 8.3

Listagem 8.3: Sonar

```
#include <playAPC/playapc.h>
#include <math.h>

int tela_radar (int aros) {
  int grp = CriaGrupo();
  Ponto p; p.x = 0; p.y = 0;
  for( ; aros > 0; aros--) {
    CriaCircunferencia (10*aros, p);
    Pintar (0, 100, 0);  Grafite(3);
  }
  return grp;
}


int ballon (void) {
  int ufo = CriaGrupo();
  Ponto p;
  p.x = -10;
  p.y = -50;
  CriaPonto(p); Pintar(255, 0, 0); Grafite(10);
  return ufo;
}

int fly (void) {
  int fs = CriaGrupo();
  Ponto p;
  p.x = -10;   p.y = -50;
  CriaPonto(p);
  Pintar(0, 255, 0); Grafite(10);
  return fs;
}

int cacas (void) {
  int cc = CriaGrupo();
  Ponto p;
  p.x = 0;       p.y = 0;
  CriaPonto(p);
  Pintar(0, 0, 255); Grafite(5);
  return cc;
}

int scanner (int range) {
  Ponto p, q;
  int reta = CriaGrupo();
  p.x = 0;     p.y = 0;
  q.x = 0;     q.y = 10*range;
  CriaReta(p, q);
  Pintar (100, 100, 100); Grafite(5);
  return reta;
}


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

  AbreJanela (900, 900, "Marvin x Bugs Bunny");
  PintarFundo (255, 255, 255);
```

```
57
58    int radar = tela_radar(12);
59    int reta = scanner(12);
60    int bb  = ballon();
61    int marvin = fly();
62    int f20 = cacas();
63    Ponto p, q, r;
64
65    int dir = 1;
66
67    /*
68      os caças F20 ficam rodando feito perus na Lemniscata de
          ↪ Bernoulli
69      equação paramétrica:
70       x = a \cos t \sqrt{2 \cos (2t)}; \qquad y = a \sin t \sqrt{2 \
          ↪ cos (2t)}
71
72     */
73
74    double theta = 2*PI, alpha = theta;
75    while(1) {
76
77      Gira(alpha*100, reta);
78      alpha += 0.01;
79
80      /*
81         cacas americanos patrulhando a casa quase branca
82       */
83      r.x = 50*cos(alpha)*sqrt(2*cos(2*alpha));
84      r.y = 50*sin(alpha)*sqrt(2*cos(2*alpha));
85      Move(r, f20);
86      //Os caças acabam aqui
87
88      //outros objetos abaixo
89      if (theta > 0) theta = theta - 0.01;
90      else {
91        theta = 4*PI;
92        dir = -dir;
93      }
94      p.x = 50*(cos(theta) + 0.5*cos(2*theta));
95      p.y = 50*(sin(theta) + 0.5*sin(2*theta))+theta;
96      //What's up, doc?
97      Move(p, bb);//the old pal bugsbunny to save the day!
98
99
100     q.x = 200*cos(theta*dir)/theta*dir;
101     q.y = 200*sin(theta*dir)/theta*dir;
102     Move(q, marvin);//Marvin, the martian, who else?
103
104
105     Desenha1Frame();
106   }
107
108   Desenha ();
109
110   return 0;
```

```
111 || }
```