

brazil



# **Universidade de Brasília**

**Instituto de Ciências Exatas  
Departamento de Ciência da Computação**

## **playAPC: Projeto e Desenvolvimento de uma Biblioteca Gráfica como Ferramenta Didática para Algoritmos e Programação de Computadores**

Sinayra Pascoal Cotts Moreira

Monografia apresentada como requisito parcial  
para conclusão do Bacharelado em Ciência da Computação

Orientador  
Prof. Dr. Alexandre Zaghetto

Coorientador  
Prof. Dr. José Carlos Loureiro Ralha

Brasília  
2016

Universidade de Brasília — UnB  
Instituto de Ciências Exatas  
Departamento de Ciência da Computação  
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio de Almeida

Banca examinadora composta por:

Prof. Dr. Alexandre Zaghetto (Orientador) — CIC/UnB  
Prof. Dr. Carla Maria Chagas e Cavalcante Koike — CIC/UnB  
Prof. Dr. Marcus Vinícius Chaffim Costa — FGA/UnB

#### **CIP — Catalogação Internacional na Publicação**

Moreira, Sinayra Pascoal Cotts.

playAPC: Projeto e Desenvolvimento de uma Biblioteca Gráfica como Ferramenta Didática para Algoritmos e Programação de Computadores / Sinayra Pascoal Cotts Moreira. Brasília : UnB, 2016.

251 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. OpenGL, 2. Computação Gráfica, 3. Biblioteca, 4. Didático, 5. Lúdico,  
6. Visual

CDU 004

Endereço: Universidade de Brasília  
Campus Universitário Darcy Ribeiro — Asa Norte  
CEP 70910-900  
Brasília—DF — Brasil



# **Universidade de Brasília**

**Instituto de Ciências Exatas  
Departamento de Ciência da Computação**

## **playAPC: Projeto e Desenvolvimento de uma Biblioteca Gráfica como Ferramenta Didática para Algoritmos e Programação de Computadores**

**Sinayra Pascoal Cotts Moreira**

**Monografia apresentada como requisito parcial  
para conclusão do Bacharelado em Ciência da Computação**

**Prof. Dr. Alexandre Zaghetto (Orientador)**

**CIC/UnB**

**Prof. Dr. Carla Maria Chagas e Cavalcante Koike      Prof. Dr. Marcus Vinícius Chaffim Costa**  
**CIC/UnB     FGA/UnB**

**Prof. Dr. Rodrigo Bonifácio de Almeida  
Coordenador do Bacharelado em Ciência da Computação**

**Brasília, 18 de novembro de 2016**

# **Dedicatória**

Dedicamos à todos os professores que queiram ter mais liberdade criativa ao desenvolver exercícios de programação e à todos os alunos que, mesmo com pouca ou nenhuma experiência em programação, queiram fazer trabalhos divertidos, coloridos e interessantes.

# **Agradecimentos**

Agradeço ao professor Ralha, que sempre ofereceu apoio e entusiasmo ao projeto, além do grande auxílio no desenvolvimento tanto da própria biblioteca como material de ensino; à professora Carla Castanho, pela oportunidade de tutoria e o contato direto com os alunos; ao professor Zaghetto pela oportunidade de Projeto de Iniciação Científica e pela confiança ao aplicar a biblioteca em uma turma de APC, além de estar sempre sugerindo novas ideias de funcionalidades; e à todos os alunos de Algoritmos e Programação de Computadores que retornaram comentários sobre o uso da biblioteca, em especial aos alunos Bernardo Ferreira Santos Ximbre, Pedro Paulo de Pinho Matos e Pedro Augusto Coelho Nunes por ter utilizado a biblioteca de forma tão criativa.

# Resumo

Nos cursos de computação, a primeira matéria de programação é fundamental para que o aluno possa progredir na área, porém sua aprendizagem não é trivial. Um dos métodos recorridos para auxiliar o ensino, independente da área, é o uso de recursos gráficos. Desta forma, foi proposta uma biblioteca gráfica denominada PLAYAPC para ser aplicada no primeiro semestre da matéria de computação dos cursos de Engenharia da Computação, Engenharia Mecatrônica, Ciência da Computação e Licenciatura em Computação, na disciplina **Algoritmos e Programação de Computadores (APC)** da **Universidade de Brasília (UnB)**. A biblioteca utiliza como recurso gráfico a **API OpenGL** e foi feita na linguagem C++. Apesar disso, essas escolhas de tecnologia em nada interferem no processo de aprendizado da matéria **APC** que, para alguns professores, é ministrada na linguagem C. Esta monografia descreve o processo desde o levantamento de requisitos para o desenvolvimento da biblioteca até a sua aplicação em aulas de laboratório.

**Palavras-chave:** OpenGL, Computação Gráfica, Biblioteca, Didático, Lúdico, Visual

# Abstract

In computer's relative courses, the first programming subject is essencial for the undergraduate student grow up in the area, but his learning is not trivial. One of the methods to assist the teaching, regardless the area, is the use of graphic resources. It was proposed a graphic libray called **PLAYAPC** to be applied in the first semester at **UnB**, in Computer's Engineering course, Mechatronic Engineering course, Bacharel on Computer Science course and Major in Computer course, in the discipline **APC**. The library uses as graphic resources the **API OpenGL** and was written in C++ programming language. Even with theses tecnology choises, it will not complicate the learning processing in the discipline, which, for some professors, it is given in C language. This paper describes the survey process requirement from the development of the library to its application in pratical classes.

**Keywords:** OpenGL, Computer Graphics, Library, Didactic, Playful, Visual

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Fundamentação Teórica e Trabalhos Correlatos</b>	<b>3</b>
2.1	Fundamentação Teórica . . . . .	3
2.1.1	Aspectos de Engenharia de Software . . . . .	3
2.1.1.1	Requisitos . . . . .	4
2.1.1.2	Desenvolvimento . . . . .	4
2.1.1.3	Validação . . . . .	5
2.1.1.4	Evolução . . . . .	6
2.1.2	Aspectos de Computação Gráfica . . . . .	7
2.1.2.1	Aspectos Técnicos da OpenGL . . . . .	14
2.1.2.1.1	Translação . . . . .	14
2.1.2.1.2	Rotação . . . . .	15
2.1.2.1.3	Escala . . . . .	15
2.1.2.1.4	Visualização . . . . .	16
2.1.3	Aspectos Educacionais . . . . .	17
2.1.3.1	Tipo de dados . . . . .	19
2.1.3.2	Estruturas de controle . . . . .	19
2.1.3.2.1	Estruturas condicionais . . . . .	19
2.1.3.2.2	Estruturas de repetição . . . . .	19

2.1.3.3	Estruturas de dados multidimensionais . . . . .	19
2.1.3.3.1	Unidimensional . . . . .	19
2.1.3.3.2	N-dimensional . . . . .	19
2.1.3.4	Subalgoritmos . . . . .	20
2.1.3.5	Recursão . . . . .	20
2.2	Trabalhos Correlatos . . . . .	20
2.2.1	Soluções alternativas . . . . .	22
2.2.1.1	PlayLib . . . . .	22
2.2.1.2	WinBGIm . . . . .	22
2.2.1.3	SDL . . . . .	23
2.2.1.4	Turtle Grafik . . . . .	24
2.2.2	Comparação entre as alternativas e a proposta . . . . .	24
<b>3</b>	<b>Solução Proposta</b>	<b>27</b>
3.1	Projeto da Biblioteca . . . . .	27
3.1.1	Levantamento de requisitos . . . . .	27
3.1.2	Requisitos . . . . .	31
3.1.2.1	Requisitos não-funcionais . . . . .	31
3.1.2.1.1	Inicializar janela de contexto . . . . .	31
3.1.2.1.2	Renderizar contexto . . . . .	34
3.1.2.2	Requisitos funcionais para renderização de formas geométricas	34
3.1.2.2.1	Tipo de dado Ponto . . . . .	34
3.1.2.2.2	Forma geométrica Ponto . . . . .	35
3.1.2.2.3	Forma geométrica Reta . . . . .	36
3.1.2.2.4	Forma geométrica Polígono . . . . .	36
3.1.2.2.5	Forma geométrica Círculo . . . . .	36
3.1.2.2.6	Forma geométrica Circunferência . . . . .	36

3.1.2.2.7	Forma geométrica Elipse . . . . .	37
3.1.2.2.8	Forma geométrica Triângulo . . . . .	37
3.1.2.2.9	Forma geométrica Quadrado . . . . .	37
3.1.2.2.10	Forma geométrica Retângulo . . . . .	37
3.1.2.2.11	Renderização de gráficos . . . . .	37
3.1.2.2.12	Colorir formas geométricas . . . . .	38
3.1.2.2.13	Mostrar imagens em formas geométricas . . . . .	39
3.1.2.3	Requisitos funcionais para animação . . . . .	39
3.1.2.3.1	Renderizar cena dentro de um laço de repetição . .	40
3.1.2.3.2	Agrupando mais de uma geometria . . . . .	40
3.1.2.3.3	Movimentação de uma ou mais geometrias . . . . .	40
3.1.2.3.4	Rotação de uma ou mais geometrias . . . . .	41
3.1.2.3.5	Redimensionar uma ou mais geometrias . . . . .	41
3.1.2.3.6	Limpar a cena . . . . .	41
3.1.2.4	Requisitos funcionais para entrada de mouse e teclado . . . . .	42
3.1.2.4.1	Entrada de teclado . . . . .	42
3.1.2.4.2	Entrada de mouse . . . . .	42
3.1.2.5	Requisitos funcionais para outros tipos de funções . . . . .	44
3.1.2.5.1	Colorir plano de fundo da janela . . . . .	44
3.1.2.5.2	Plano de renderização . . . . .	44
3.1.2.5.3	Componentes RGB de uma imagem . . . . .	45
3.1.2.5.4	Alterar espessura da borda . . . . .	45
3.1.3	Arquitetura da biblioteca . . . . .	45
3.2	Apresentação da biblioteca . . . . .	49
3.2.1	Criação da janela de contexto da PLAYAPC . . . . .	51
3.2.2	Criação dos discos de Hanoi . . . . .	53
3.2.3	Movimentação dos discos . . . . .	61

<b>4 Resultados Experimentais</b>	<b>69</b>
4.1 Práticas sugeridas pela PLAYAPC . . . . .	69
4.1.1 Tipo de dados e estruturas de controle . . . . .	69
4.1.2 Vetores e Matrizes . . . . .	75
4.1.3 Subalgoritmos . . . . .	80
4.1.4 Recursão . . . . .	82
4.2 Aceitação por parte dos professores em participar do experimento de aplicar a PLAYAPC . . . . .	88
4.3 Aceitação por parte dos alunos . . . . .	89
4.3.1 Instalação da biblioteca . . . . .	90
4.3.2 Utilização da biblioteca . . . . .	91
<b>5 Conclusão</b>	<b>94</b>
<b>Referências</b>	<b>96</b>
<b>Apêndice</b>	<b>99</b>
<b>A Apostila de exercícios da PLAYAPC</b>	<b>100</b>
<b>B Guia de referência da PLAYAPC</b>	<b>101</b>
B.1 Instalação . . . . .	102
B.2 Documentação . . . . .	102
B.3 Lista de funções . . . . .	103
B.4 Exemplos . . . . .	103
<b>C Questionário qualitativo sobre o uso da PLAYAPC</b>	<b>104</b>
<b>D Logo da PLAYAPC</b>	<b>106</b>

<b>E Códigos fontes das práticas sugeridas</b>	<b>107</b>
E.1 Tipo de dados e estruturas de controle . . . . .	107
E.2 Vetores e Matrizes . . . . .	120
E.3 Subalgoritmos . . . . .	133
E.4 Recursão . . . . .	143
<b>Anexo</b>	<b>151</b>
<b>I Material sobre recursão usando PLAYAPC criado pelo professor Ralha</b>	<b>152</b>
<b>II Enunciado do Certificado de Bravura</b>	<b>169</b>
<b>III Ementas dos cursos iniciais de computação</b>	<b>173</b>

# Listas de Figuras

1.1	Relação de aprovação e reprovação/trancamento dos alunos de APC da turma A (Ciência da Computação) . . . . .	2
2.2	Exemplo de computação gráfica em 1960: Programa Sketchpad <sup>1</sup> . . . . .	7
2.1	Linha do tempo resumida da Computação Gráfica: a) Anos 60 à anos 70; b) Anos 70 à anos 80; c) Anos 80 à anos 90; . . . . .	8
2.3	Exemplo de computação gráfica em 1970: a) Jogo Pong; b) Jogo Space Invaders.	9
2.4	Exemplo de computação gráfica em 1980: a) Filme Star Wars; b) Jogo Winning Run; c) Curta <i>The Adventures of André and Wally B.</i> . . . . .	10
2.5	Exemplo de computação gráfica em 1990: Jogo Wolfenstein 3D <sup>2</sup> . . . . .	11
2.6	Representação gráfica do <i>pipeline</i> fixo <sup>3</sup> . Em verde, estágio do <i>pipeline</i> em que o programador tem acesso direto; em vermelho, estágios do <i>pipeline</i> que o programador não tem acesso direto e precisa utilizar funções pré-definidas que precisam sempre passar pelo nível de aplicação até atingir o nível que ela é executada. Os balões nos estágios do <i>pipeline</i> exemplificam funções que só executam naquele nível. . . . .	12
2.7	Translação: a) Posição original; b) Transladado. . . . .	15
2.8	Rotação: a) Posição original; b) Quadrado rotacionado; c) Ponto rotacionado. . . . .	15
2.9	Redimensionamento: a) Figura original; b) $\alpha_x = 2, \alpha_y = 1$ ; c) $\alpha_x = 1, \alpha_y = 2$ .	16
3.1	Conteúdos ministrados em relação a quantidade de universidades que os abordam. . . . .	29
3.2	Caso quando $largura \neq altura$ . A exibição plano cartesiano está limitado de $-100$ unidades à $100$ unidades, tanto em $x$ quanto em $y$ , onde $100$ unidades equivalem à $300$ pixels, ou $200$ unidades equivalem à $600$ pixels. . . . .	32

3.3	Renderização de um ponto vermelho em (133, 0). O ponto está exagerado para melhor visualização. . . . .	33
3.4	Organização da arquitetura da PLAYAPC. . . . .	46
3.5	Diagrama de Sequência da função <i>AbreJanela</i> . . . . .	47
3.6	Resolução da torre de Hanói para 3 discos. . . . .	50
3.7	Janela de contexto. . . . .	52
3.8	Mesa. . . . .	54
3.9	Torres. . . . .	55
3.10	Último disco. . . . .	58
3.11	Todos os discos. . . . .	59
3.12	Solução da torre de Hanói para 3 discos: a) Início; b) Passo 1; c) Passo 2; d) Passo 3; e) Passo 4; f) Passo 5; g) Passo 6; h) Torre solucionada. . . . .	65
4.1	Plano cartesiano de -100 à 100: a) Todo plano; b) Parte do plano. . . . .	69
4.2	Boneco Palito. . . . .	70
4.3	Estrela de Davi. . . . .	70
4.4	Quadrado inscrito em um círculo. . . . .	71
4.5	Verificação se ponto está dentro ou fora da circunferência: a) Fora; b) Dentro. .	71
4.6	Identificação de quadrante: a) Primeiro quadrante; b) Segundo quadrante; c) Terceiro quadrante; d) Quarto quadrante. . . . .	72
4.7	Existência de triângulos: a) Isósceles; b) Escaleno; c) Equilátero. . . . .	72
4.8	Carro se movendo da posição -100 até a posição 100. . . . .	73
4.9	Moinho de vento. . . . .	73
4.10	Sistema solar. . . . .	74
4.11	Espiral hiperbólica: a) Primeira iteração; b) Centésima iteração; c) Espirais. .	75
4.12	Animação do Mário: a) Início; b) e c) Mário andando; d) Mário pulando. . . . .	75
4.13	Gráfico do polinômio $-x^3$ . . . . .	76

4.14	À esquerda, 20 quadrados com componentes RGB fornecidos pelo usuário. À direita, aplicação do filtro de média móvel central em cada quadrado. . . . .	77
4.15	Jogo da vida: a) Primeira geração; b) Segunda geração; c) Terceira geração. . . . .	78
4.16	Filtro Motion Blur. . . . .	78
4.17	Batalha Naval: a) Início; b) Decorrer do jogo; c) Fim do jogo. . . . .	79
4.18	Lançador Balístico. . . . .	80
4.19	Lançaverine: a) Ângulo de lançamento; b) Velocidade inicial; c) Wolverine percorrendo trajetória; d) Caso quando Wolverine não acerta o Magneto; e) Caso quando Wolverine acerta o Magneto. . . . .	81
4.20	Jogo Snake. . . . .	82
4.21	Árvore Binária com altura 30, ângulo 25° e profundidade 6. . . . .	82
4.22	Solucionador da torre de Hanói. . . . .	83
4.23	Curva de Koch: a) Ordem 1; b) Ordem 2; c) Ordem 3; d) Ordem 6. . . . .	84
4.24	Floco de neve: a) Ordem 1; b) Ordem 2; c) Ordem 3; d) Ordem 6. . . . .	85
4.25	Curva de Sierpiński de ordem 1 com ângulo de 45°. . . . .	86
4.26	Curva de Sierpiński: a) Ordem 2; b) Ordem 3. . . . .	88
4.27	Gráfico com o balanço geral das respostas dos alunos de APC das turmas E e H. Respostas em branco foram consideradas com valor 0. . . . .	89
4.28	Prática 3 de Algoritmos Sequenciais, realizada pelo aluno Bernardo Ferreira Santos Ximbre, APC turma E de 2/2016. . . . .	92
4.29	Prática 1 de Algoritmos com Repetição, realizada pelo aluno Pedro Paulo de Pinho Matos, CB turma B de 2/2014. . . . .	92
4.30	Prática 4 de Vetores, realizada pelo aluno Pedro Augusto Coelho Nunes, APC turma E de 2/2016. . . . .	93
B.1	Tela principal do Guia de Referência. . . . .	101
D.1	Logo oficial da PLAYAPC: a)Colorido; b) Tons de cinza. . . . .	106

# **Lista de Tabelas**

2.1	Universidades que utilizam recursos de computação gráfica para auxílio no ensino.	21
3.1	Universidades pesquisadas.	30
3.2	Valor de <i>verTipo</i> da função <i>CriaGrafico</i> .	38
3.3	Valor de <i>nome</i> da função <i>Pintar</i> e <i>AssociaImagem</i> .	39
3.4	Teclas reconhecidas pela PLAYAPC.	43
3.5	Botões de mouse reconhecidos pela PLAYAPC.	43

# Capítulo 1

## Introdução

A aprendizagem de lógicas e métodos de programação é fundamental para cursos relacionados a computação. Desta forma, um aluno só conseguirá avançar nos estudos desta área se tiver bem consolidado os conceitos de programação, tornando importante o seu desempenho nas matérias iniciais dos cursos de Engenharia, Ciência da Computação e Licenciatura em Computação. Apesar disso, a aprendizagem destes conceitos não é trivial.

Por exemplo, o índice de reprovação em [Algoritmos e Programação de Computadores \(APC\)](#), oferecido pelo Departamento de Ciência da Computação da [Universidade de Brasília \(UnB\)](#), tem crescido a cada semestre [1]. Apesar das tentativas de criar mais horários de plantão de dúvidas e maior disponibilidade dos monitores para a disciplina, a taxa de reprovação continua crescente. A Figura 1.1 descreve o índice de aprovação e reprovação/trancamento. Para os alunos aprovados, foram considerados a soma da quantidade de alunos que obtiveram menção CC, MM, MS e SS; para os alunos reprovados ou que realizaram trancamento, as menções MI, II, SR, TR e TJ.

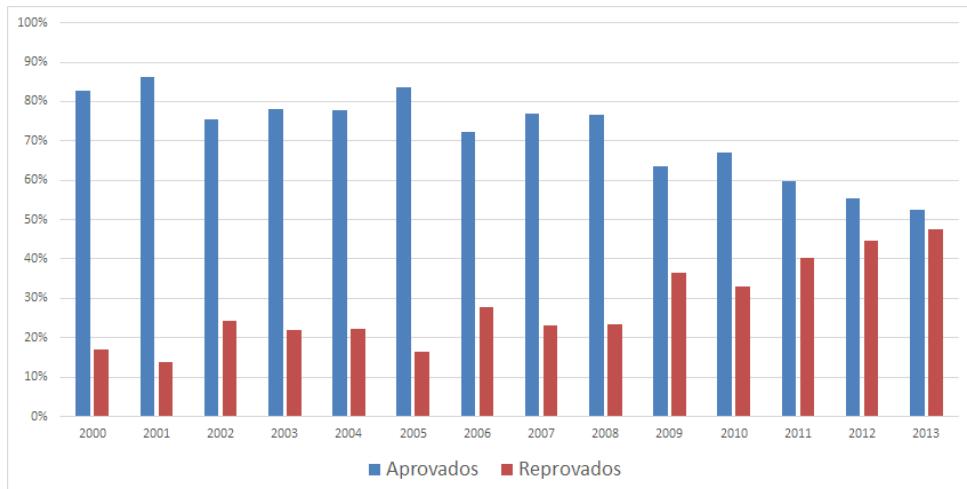


Figura 1.1: Relação de aprovação e reprovação/trancamento dos alunos de **APC** da turma A (Ciência da Computação).

O uso de modelagem gráfica como elemento catalizador para despertar ou aumentar o interesse em programação é proposto em diversos estudos apresentados em fóruns de educação, como o SBIE [2]. Outras universidades que oferecem o curso de computação também recorreram a elementos gráficos para auxiliar o aprendizado de programação, como ilustra a Tabela 2.1.

Visando dar suporte às políticas pedagógicas e aumentar o interesse dos alunos pela disciplina inicial de programação do Bacharelado, foi desenvolvida uma biblioteca gráfica 2D denominada **PLAYAPC**. Se a sequência de aulas for corretamente planejada com o uso da **PLAYAPC**, ao final do semestre letivo, os alunos deverão ser capazes de programar animações e jogos simples.

Este documento está organizado em seis capítulos. O Capítulo 2 apresenta a fundamentação teórica, tanto para o desenvolvimento da biblioteca quanto para sua aplicação, e apresenta uma comparação entre a proposta e outros projetos similares. O Capítulo 3 explica com maiores detalhes a proposta, exibindo seus requisitos, sua arquitetura e apresenta um tutorial explicando passo a passo sobre como utilizar as funções mais básicas da biblioteca. O Capítulo 4 apresenta sugestões de práticas de laboratório para serem aplicadas ao longo de um semestre, separadas por tópicos, e expõe comentários sobre o uso da biblioteca por parte dos professores e alunos. O Capítulo 5 destaca os principais problemas enfrentados e sugere trabalhos futuros utilizando como base este projeto.

# **Capítulo 2**

## **Fundamentação Teórica e Trabalhos Correlatos**

### **2.1 Fundamentação Teórica**

#### **2.1.1 Aspectos de Engenharia de Software**

A importância de se projetar um software antes de implementá-lo é para que, mesmo que aumente a complexidade ou funcionalidades com o tempo, o software não seja comprometido a ponto dele não conseguir atingir todos os objetivos [3]. A engenharia de software é, portanto, uma ferramenta para auxiliar desenvolvedores, incluindo técnicas de especificação, documentação, arquitetura e evolução de um programa. À medida que um programa evolui, ele precisa manter sua qualidade, que descreve como um software deve se comportar enquanto estiver executando e também descreve a estrutura e organização do sistema, associando esta descrição a uma documentação [4].

Desta forma, o processo de criação de um software segue quatro passos bem definidos:

1. Especificação ou Requisitos, o cliente e os desenvolvedores definem o software, bem como suas funcionalidades;
2. Desenvolvimento, é feita a arquitetura do software e sua implementação;
3. Validação, são aplicados testes para assegurar que os requisitos foram cumpridos;
4. Evolução, o software é modificado de acordo com novas demandas do cliente.

As mudanças rápidas de requisitos são uma consequência inevitável nos dias de hoje, tornando quase inviável a entrega de um software completamente funcional com todos os requisitos completos de uma única vez. Por conta disso, para equipes pequenas, opta-se por entregas incrementais, com pequenos intervalos de lançamento do software, e para cada lançamento um dos requisitos é implementado e validado. Este método de desenvolvimento é conhecido como método ágil [4]. Neste projeto, a metodologia ágil foi utilizada nos itens de desenvolvimento e de validação.

### 2.1.1.1 Requisitos

Requisito, na área de Engenharia de Software, pode ser definido como a especificação do que o software deve fazer, sendo um dos pilares no desenvolvimento de software [4]. Os requisitos refletem o que os usuários finais desejam que o programa deva realizar, podendo ser descritos tanto em alto nível de abstração (requisitos de usuário) ou descritos de forma mais detalhada que explica o nível de funcionamento de funções (requisitos do sistema). Além destes dois modos de descrever os requisitos, há ainda como classificá-los dependendo do que eles especificam. Se o requisito descrever o serviço que o sistema deve oferecer, explicando a resposta que ele deve fornecer ao usuário dada uma situação específica, e como deve ser seu comportamento em tempo de execução, este requisito é classificado como funcional. Se o requisito descrever as restrições do sistema, como tempo de desenvolvimento e tecnologia utilizada, então este requisito é classificado como não-funcional.

Para o levantamento de requisitos, podem ser aplicadas diversas formas de reunião com o cliente. Uma delas é o *brainstorm*, quando as equipes se expressem individualmente sobre determinado assunto, sem restrições criativas e sem restrições quanto à exploração de opções, com o foco de quantidade ao invés de qualidade. Todas as ideias são anotadas e depois são analisadas pelo grupo [5]. Só após a análise das ideias é que se descrevem os requisitos, seja da forma de requisitos do usuário ou da forma de requisitos do sistema [4].

### 2.1.1.2 Desenvolvimento

Para aumentar modularidade e reusabilidade, funções de uso comum são incluídas em uma biblioteca. Uma biblioteca contém uma coleção de arquivos objetos, que podem ser ligados a um programa [6]. Dada a documentação que explica as operações que serão oferecidas ao usuário, sua **Application Programming Interface (API)** [7], sua implementação pode ser fornecida ao usuário de duas formas: estática ou dinâmica. Uma biblioteca estática é armazenada em disco, num arquivo de extensão *.a* ou *.lib*, o qual é concatenado ao arquivo executável que faz referência a funções da biblioteca. Já a ideia por trás de uma biblioteca dinâmica é ter apenas uma

cópia das funções mais utilizadas armazenadas em memória, onde diferentes aplicações possam acessar as mesmas funções ao invés de cada uma ter sua própria cópia [6]. Neste trabalho, optou-se por entregar aos usuários o projeto na forma de biblioteca dinâmica, onde o passo a passo da instalação da PLAYAPC está descrito no Guia de Referência (Apêndice B).

O paradigma orientado a objetos, usado neste projeto, utiliza o conceito de objetos para abstrair problemas. Um objeto é uma instância de uma classe que é criado e manipulado em tempo de execução. Apesar de se abstrair problemas utilizando o conceito de objetos, sua implementação é descrita por uma classe [8].

Classe é uma estrutura que possui atributos (variáveis de um objeto) e implementa métodos (funções de um objeto). Ela abstrai um conjunto de funcionalidades, ou seja, todos os seus atributos e métodos possuem relação com a característica que se deseja abstrair. Sub-classes são classes que herdam estas características, utilizando o princípio de herança. Herança define uma sub-classe hierarquicamente inferior que herda características de uma classe hierarquicamente superior [8].

Neste trabalho, os problemas abstraídos foram os de computação gráfica, desta forma, laço de renderização, limpeza de *buffer* ou detalhes de funcionamento da API OpenGL [9, 10, 11, 12, 13] foram abstraídos e foi criada uma interface com estes elementos de computação gráfica. Esta interface é chamada de encapsulamento, no qual existem módulos bem definidos que realizam uma tarefa, sem o usuário necessariamente saber todas classes que foram utilizadas para a implementação desta tarefa [8].

De modo a descentralizar uma grande aplicação, é viável a criação de um conjunto de classes que possam ter propósitos mais genéricos e que possam ser separadas em conjunto de funcionalidades, como uma camada que lida somente com a OpenGL, outra camada que lida somente com eventos, como entrada de teclado, criação de objetos, e uma última camada que cria uma interface amigável para o usuário. Desta forma, cada conjunto de classes presentes em uma camada são auto-suficientes e sua comunicação com outras camadas reduz-se a simples chamadas de métodos, definindo-se assim uma arquitetura em camadas. Com isso, aumenta a manutibilidade do projeto [8].

### 2.1.1.3 Validação

Testes apresentam tanto para os desenvolvedores quanto para os usuários o software por completo, assegurando que ele atingiu todos os requisitos, e também expõe comportamentos que não estavam previstos [4]. Para a verificação de ambos os casos, os testes são realizadas em duas etapas:

- Aplicar uma série de casos no programa para assegurar que ele funcione de modo esperado [4].
- Expor falhas do sistema que não necessariamente implicam em como o software será utilizado pelos usuários finais [4].

Neste projeto, optou-se por uma metologia de desenvolvimento de testes conhecida como **Test-Driven Development (TDD)**. TDD é um método que a cada nova implementação de função do software, é feito um teste para a função recém-implementada, e não são feitas novas implementações até esta ser aprovada [4]. Não foi utilizado nenhum *framework* para a realização de testes, e sim foram criados programas simples, passando diferentes parâmetros para a função recém-implementada, onde se verificava se o resultado gráfico atendia o esperado. Alguns dos testes estão expostos no Guia de Referência, na parte de Documentação (Apêndice B).

#### 2.1.1.4 Evolução

A evolução de um software é um processo que reflete as necessidades dos usuários finais, seja por meio de melhorias, novas funcionalidades ou uma crítica sobre um defeito que não foi detectado na etapa de validação. Estas necessidades devem refletir em novos requisitos, que vão passar por uma nova etapa de desenvolvimento, validação e assim entregue novamente aos usuários finais, repetindo indefinidamente este ciclo enquanto o software estiver ativo. Um bom software possui uma longa vida-útil [4].

Neste projeto, para a avaliação de necessidade de alteração do comportamento das funções ou reformulação de algum dos requisitos, além de reuniões com professores interessados, foi proposto para o semestre 2016/2 a avaliação do projeto pelos alunos por meio de questionário de satisfação de uso. Este tipo de questionário é utilizado quando deseja-se avaliar um produto, caso este atenda as necessidades dos usuários finais e como eles o utilizam [14].

Um questionário de satisfação do cliente é um questionário fácil de apresentar, fácil de preencher e fácil de ser processado. Suas perguntas são relevantes ao produto, são concisas, não possui itens ambíguos, não são compostas e não possuem dupla-negativa [14]. Para permitir várias respostas de graus diferentes, utiliza-se o método de resposta *Likert*, em que são apresentadas múltiplas escolhas de respostas numa escala onde o participante pode concordar ou discordar em diferentes graus [15].

Após a aplicação dos questionários, cada item pode ser avaliado separadamente e é feito o cálculo de média e a mediana nos itens que se utilizou o método *Likert* [15], tendo um balanço geral do produto no período pesquisado.

## 2.1.2 Aspectos de Computação Gráfica

A computação gráfica é uma comunicação visual por meio da tela do computador, tanto no sentido computador-humano quanto no sentido humano-computador. É uma área multidisciplinar que envolve física, matemática, percepção humana, engenharia e arte: física para modelar a luz e realizar simulações, matemática para descrever as formas geométricas, engenharia para otimizar tempo de uso de processador e memória e arte combinada com percepção humana para deixar esta comunicação mais efetiva [16].

O estudo nessa área se iniciou por volta de 1963, com o Ivan Sutherland, que implementou o primeiro programa que utilizava uma interface gráfica, o Sketchpad. Sketchpad recebia do usuário pontos  $x$  e  $y$  em sua tela e exibia vetores ligando esses pontos, demonstrando o uso da computação de forma artística e técnica. O Sketchpad usava o computador Lincoln TX-2, desenvolvido especificamente para o primeiro Sketchpad [17].

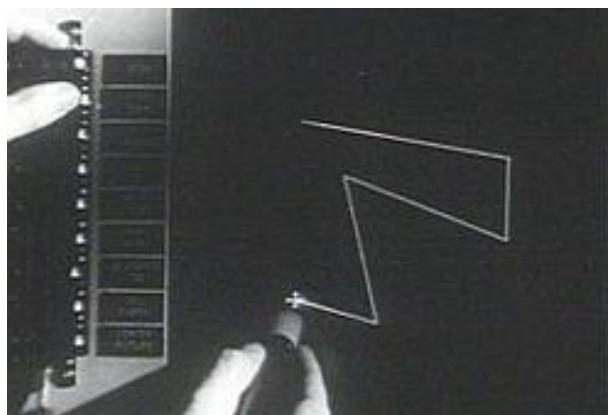
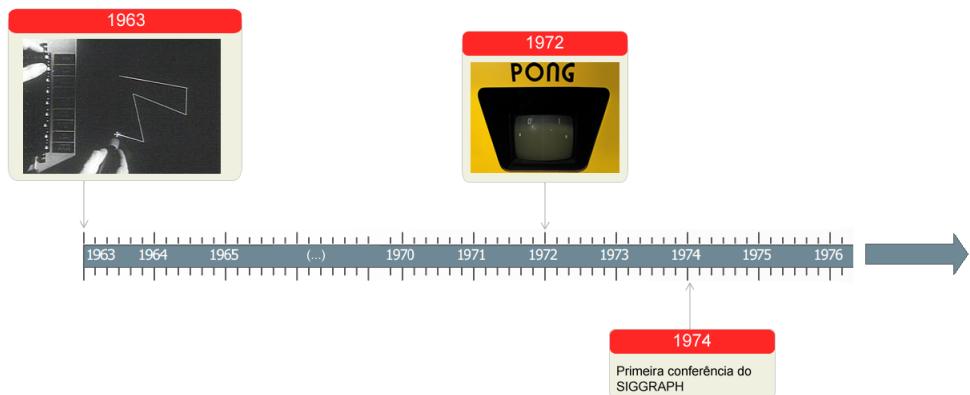


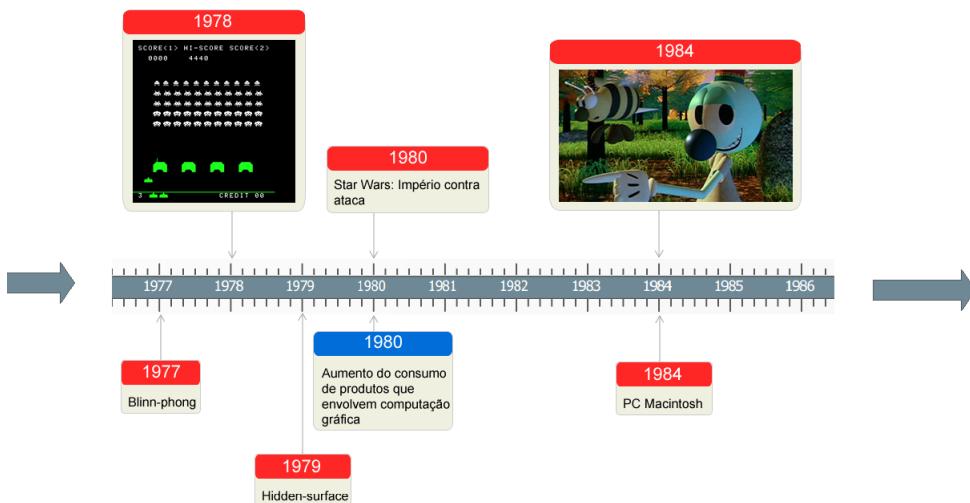
Figura 2.2: Exemplo de computação gráfica em 1960: Programa Sketchpad <sup>1</sup>.

No final desta década, criou-se o grupo especial interessados em computação gráfica, *Special Interest Group on Computer Graphics* (SIGGRAPH), que organizava conferências, determinava padrões e publicava artigos dentro da área de computação gráfica, cuja a primeira conferência foi realizada em 1974 [18]. Também naquele período, iniciou-se a produção de jogos eletrônicos utilizando *sprites* 2D, que são imagens bidimensionais que se movimentam na tela[19]. Exemplos de jogos desta época foram o *Pong*, sendo o primeiro jogo comercial em fliperamas em 1972 [20], e *Space Invaders* em 1978, sendo um dos primeiros jogos a utilizar uma grande quantidade de *sprites* na tela e precursor da importância da temática de um jogo [21]. Ambos os jogos utilizavam o controlador de vídeo *Fujitsu MB14241*, especializado em criação e renderização de *sprites* 2D [22], sendo renderização definida como processo que o computador utiliza para criar uma imagem dado um modelo [12]. Paralelamente, na Universidade de

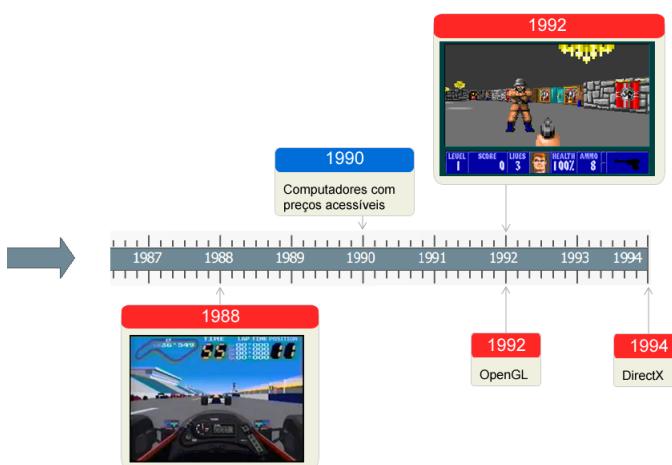
<sup>1</sup>Fonte: <http://www.nakedobjects.org/book/section5.html>



(a) 1963 à 1976.



(b) 1977 à 1986.



(c) 1987 à 1994.

Figura 2.1: Linha do tempo resumida da Computação Gráfica: a) Anos 60 à anos 70; b) Anos 70 à anos 80; c) Anos 80 à anos 90;

Utah, um grupo de pesquisadores avançavam a pesquisa para o campo 3D, estudando como determinar qual objeto está na frente ou atrás do espectador, iniciando os estudos de algoritmos de *hidden-surface* [23], e como o objeto teria um aspecto de volume, iniciando os estudos de sombreamento no próprio objeto, como o algortimo de sombreamento de Blinn-Phong [24].

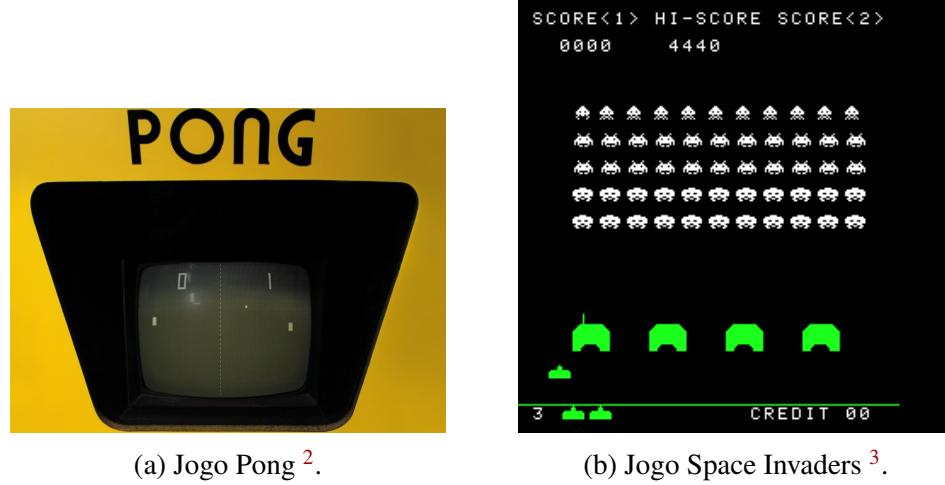


Figura 2.3: Exemplo de computação gráfica em 1970: a) Jogo Pong; b) Jogo Space Invaders.

A partir dos anos 1980, a necessidade de desenvolvedores gráficos aumentou drasticamente e isso se refletiu nos produtos comercializados à época. Entre os mais populares, o computador pessoal Macintosh, que popularizou e padronizou o uso de interface gráfica para usuários [25]; o segundo filme da trilogia original de *Star Wars*, com o avanço na utilização da técnica *chroma-key* pela empresa Lucasfilm [26]; milhares de cópias de jogos para Atari, Sega e Nintendo foram vendidos [21], enquanto nos fliperamas começavam a utilizar as primeiras placas gráficas dedicadas para renderização 3D, como a *Namco System 21* [22] para o jogo *Winning run* [27]; e o primeiro curta metragem totalmente gerado por computador, o *The Adventures of André and Wally B.*, desenvolvido pela Pixar, que iniciou o estudo na área de *shaders* [28], programas que realizam ajuste no nível de cores em objetos 3D.

<sup>2</sup>Fonte: <https://todayinhistoryblog.wordpress.com/tag/pong/>

<sup>3</sup>Fonte: <http://www.brentradio.com/SpaceInvaders.htm>



(a) Cena do filme Star Wars episódio V - O império contra ataca <sup>4</sup>.

(b) Jogo Winning Run <sup>5</sup>



(c) Cena do curta *The Adventures of André and Wally B.* <sup>6</sup>

Figura 2.4: Exemplo de computação gráfica em 1980: a) Filme Star Wars; b) Jogo Winning Run; c) Curta *The Adventures of André and Wally B.*

Em 1990, à medida que os computadores pessoais aumentavam sua capacidade de processamento a preços mais acessíveis [29], surgiu a necessidade de padronizar processamento gráfico e quais seriam os padrões de projeto adotados pelos desenvolvedores de hardwares e de softwares gráficos. A primeira geração de **Graphics Processing Unit (GPU)** renderizava somente um *pixel* por ciclo de *clock*, implicando que a *CPU* enviada mais informação que a **GPU** conseguia processar, como por exemplo o RIVA TNT, da empresa Nvidia [30]. Ainda assim, nesta época foram lançados jogos que se utilizavam desta nova tecnologia, como Wolfenstein 3D em 1992 [27]. Apenas anos mais tarde foram adicionados *pipelines* e núcleos na **GPU** para estes *pixels* serem processados paralelamente [30].

<sup>4</sup>Fonte: <https://kharisampson.wordpress.com/tag/star-wars/>

<sup>5</sup>Fonte: <http://www.gamespot.com/forums/games-discussion-1000000/evolution-of-video-game-graphics-29361008/>.

<sup>6</sup>Fonte: [http://pixaranimationshorts.wikia.com/wiki/The\\_Adventures\\_Of\\_Andre\\_and\\_Wally\\_B](http://pixaranimationshorts.wikia.com/wiki/The_Adventures_Of_Andre_and_Wally_B)



Figura 2.5: Exemplo de computação gráfica em 1990: Jogo Wolfenstein 3D <sup>7</sup>.

Para padronizar o uso dos recursos das **GPU** que estavam sendo lançadas, foram criadas duas **API** naquela época, a DirectX, lançado em 1994 exclusivamente feito para plataformas Windows, e a OpenGL, lançada em 1992 para diversas as plataformas [31]. A **API** DirectX lida não apenas com renderização de objetos 3D, mas também gerencia som, entrada de dispositivos e comunicação entre redes [32]. A OpenGL assume que a alocação de memória responsável pela exibição de *pixels* na tela seja feita pelo sistema operacional, ou seja, ela é responsável somente pela atualização do *buffer* de memória gráfico, mais especificamente, o *Frame Buffer* [33], o que garante sua portabilidade [34].

Devido a sua proposta multiplataforma, a OpenGL possui inúmeras aplicações<sup>8</sup>, como interfaces gráficas para aplicações diversas e jogos tanto para consoles quanto para computadores, é aplicada em diferentes linguagens de programação, como C/C++, Java, ADA, Python, Fortran e Perl, e está em constante atualização e reformulação, sendo a mais recente conhecida como Vulkan [35], uma versão mais flexível da OpenGL 4.5 e compatível com dispositivos mais modernos. Por isso, a OpenGL tornou-se parte indispensável do *design* de grande parte dos hardwares gráficos [30].

As primeiras **GPU** possuíam a arquitetura conhecida como *pipeline* fixo: quando os programadores enviavam dados para **GPU**, os dados não poderiam ser modificados [30]. A Figura 2.6 exibe a arquitetura do *pipeline* fixo utilizando exemplos de funções da OpenGL.

<sup>7</sup>Fonte: [https://3drealms.com/catalog/wolfenstein-3d\\_25/](https://3drealms.com/catalog/wolfenstein-3d_25/)

<sup>8</sup><https://www.opengl.org/news/categories/C3>

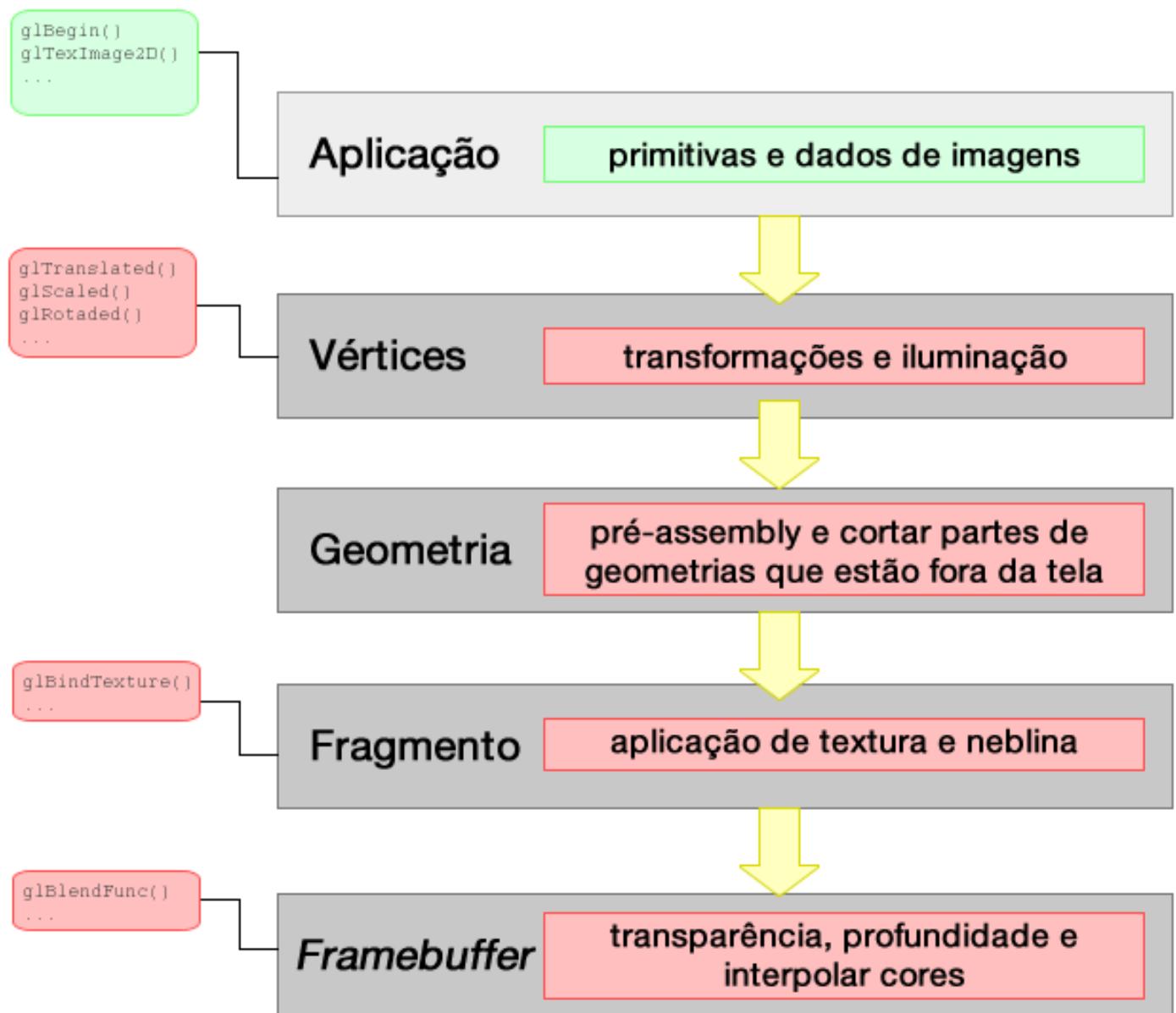


Figura 2.6: Representação gráfica do *pipeline* fixo<sup>9</sup>. Em verde, estágio do *pipeline* em que o programador tem acesso direto; em vermelho, estágios do *pipeline* que o programador não tem acesso direto e precisa utilizar funções pré-definidas que precisam sempre passar pelo nível de aplicação até atingir o nível que ela é executada. Os balões nos estágios do *pipeline* exemplificam funções que só executam naquele nível.

<sup>9</sup>Adaptado de: [https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl\\_shaders/opengl\\_shaders.html](https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl_shaders/opengl_shaders.html)

Para acompanhar esta arquitetura, as **API** definiam o método de programação conhecido como modo imediato, os dados que seriam processados pela **GPU** só eram enviados no momento de execução do programa [36].

Anos mais tarde, este método mostrou-se extremamente ineficiente [37], pois a **GPU** passa um tempo considerável de forma ociosa [38]. Somente a partir de 2002 que as **GPU** permitiram que programadores utilizassem *shaders* em suas aplicações, modificando os dados enviados para **GPU** em diferentes estágios do *pipeline* [30]. Esta mudança de paradigma refletiu na versão 2.0 da OpenGL, que descontinuou o modo imediato.

Como a computação gráfica é uma área relativamente nova e em constante atualização, consideramos neste trabalho que o público alvo tem acesso, a pelo menos, placas de vídeo com *pipeline* fixo, compatível com a versão 1.3 da OpenGL. Nesta versão, como a OpenGL disponibiliza apenas funções de baixo nível, tratando-se diretamente com o driver OpenGL da placa de vídeo, optou-se por utilizar a biblioteca GLU, *OpenGL Utility Toolkit*, que possui um conjunto de funções de alto nível, como implementações de transformações lineares, mapeamento da tela e algumas primitivas geométricas [39].

Há diversos tipos de biblioteca que podem ser utilizadas em conjunto com a OpenGL que facilitam o seu uso. Neste projeto, optou-se por utilizar uma biblioteca para o gerenciamento da janela que são renderizadas cenas utilizando OpenGL e outra biblioteca para o mapeamento de textura [9], que utiliza uma imagem, vetor de *pixels*, para influenciar a cor do fragmento no ponto da superfície do polígono. A GLFW 2.7 é uma API específica para o uso com a OpenGL feita com a linguagem C. Sua proposta é, além de facilitar o gerenciamento de uma janela de contexto, inclui as seguintes funcionalidades:

- entrada de teclado, mouse e joystick
- maior precisão de tempo
- suporte a *mult-threading*
- suporte de *query* e uso de extensões da OpenGL
- suporte básico de carregamento de imagens

SOIL é uma biblioteca feita em C que é usada para carregar texturas usando OpenGL, com a proposta de ser simples e multi-plataforma. Ela consegue ler imagens no formato BMP, JPG, PNG, TGA, DDS, PSD e HDR, carregando as imagens diretamente como texturas [40].

### 2.1.2.1 Aspectos Técnicos da OpenGL

Os modelos renderizados pela OpenGL são construídos com formas geométricas primitivas, como ponto, linha e triângulo, sendo eles definidos pelos seus vértices, representados da forma

$p = (x, y, z)$  ou da forma matricial  $p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$  [12, 9]. Devido a esta forma de representação das

primitivas, as operações realizadas pela OpenGL seguem definições da Álgebra linear, que é uma área da matemática derivada da geometria analítica no plano e espaço e busca a resolução de sistemas lineares [41].

Um dos conceitos essenciais para a computação gráfica é que as operações possíveis são feitas dentro de um espaço vetorial, que é definido por:

- um conjunto não vazio  $V$ , cujo os elementos são chamados de vetores
- conjunto numérico  $K$ , cujo elementos são chamados de escalares
- uma soma vetorial: operação que associa a cada dois vetores  $v, u \in V$  um vetor  $v+u \in V$
- uma multiplicação por escalar: operação que associa a cada vetor  $v \in V$  e um escalar  $k \in K$  um vetor  $kv \in V$

Tipicamente, haverão diversas geometrias em posições específicas em uma cena. Quando deseja-se modificar estas geometrias, é necessário utilizar transformações lineares para modificar suas posições originais [12]. Seja  $V$  e  $W$  dois espaços vetoriais, define-se [42] uma função de transformação linear  $T$  como:

$$T : V \rightarrow W$$

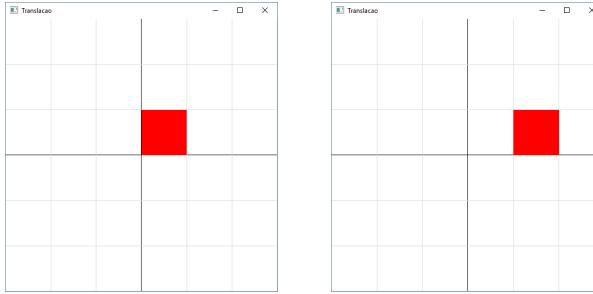
Para cada elemento  $v \in V$  e  $w \in W$  teremos:

$$w = T(v)$$

#### 2.1.2.1.1 Translação

Translação é um deslocamento de um ponto a outro ponto em um plano. Dado o vetor de deslocamento  $d$ , calcula-se para todos os pontos da figura sua translação pela Equação 2.1 [9].

$$T(v) = v + d \quad (2.1)$$



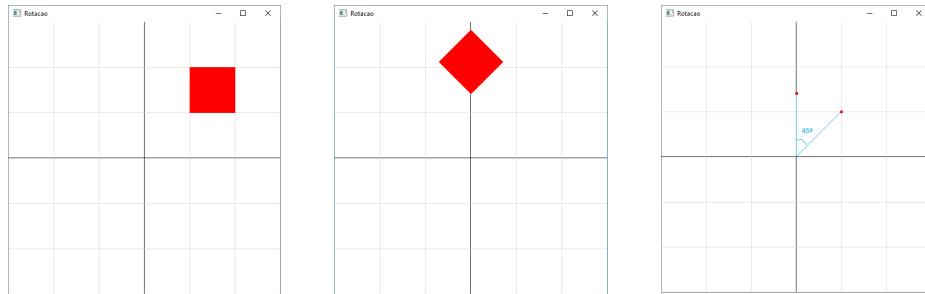
(a) Figura na posição original.  
 (b) Figura na posição transladada em 10 unidades.

Figura 2.7: Translação: a) Posição original; b) Transladado.

### 2.1.2.1.2 Rotação

Dado um ângulo  $\Theta$ , rotaciona-se a figura em torno da origem do sistema. Um ponto particular  $(x, y)$  será rotacionado para a posição  $(x', y')$  dado um ângulo  $\Theta$ . Este cálculo é feito como ilustra a Equação 2.2 [9].

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.2)$$



(a) Figura na posição original.  
 (b) Figura na posição rotacionada em 45 graus.  
 (c) Ponto rotacionado em 45 graus.

Figura 2.8: Rotação: a) Posição original; b) Quadrado rotacionado; c) Ponto rotacionado.

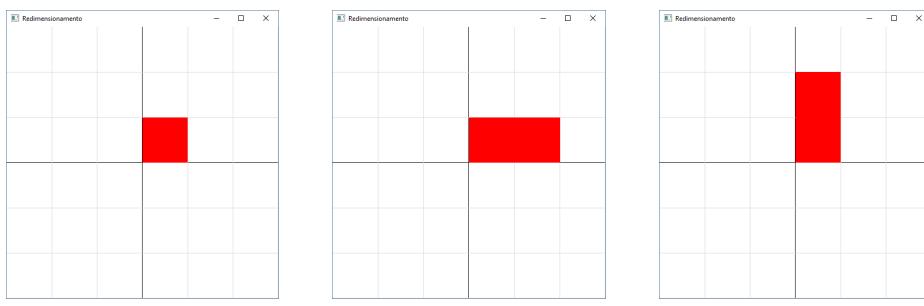
### 2.1.2.1.3 Escala

Escala é uma transformação aplicada em um corpo não uniforme, que o faz maior ou menor. Dada quatro constantes  $\alpha_x$ ,  $\alpha_y$ ,  $\alpha_z$  e  $\alpha_w$ , sendo  $\alpha_w = 1$ , o cálculo de sua transformação de

escala, para cada ponto, é feito como ilustra a Equação 2.3 [12].

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & \alpha_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \quad (2.3)$$

Para o caso de manipulação de objetos em duas dimensões, os termos  $z'$ ,  $\alpha_z$ ,  $z$  e  $w$  são iguais a 1.



(a) Figura original.  
(b) Figura redimensionada em dobro no eixo x.

(c) Figura redimensionada em dobro no eixo y.

Figura 2.9: Redimensionamento: a) Figura original; b)  $\alpha_x = 2, \alpha_y = 1$ ; c)  $\alpha_x = 1, \alpha_y = 2$ .

#### 2.1.2.1.4 Visualização

Se exibirmos modelos geométricos utilizando diretamente as coordenadas da tela do computador, provavelmente as coordenadas do modelo não estariam na mesma proporção que as coordenadas da tela, devido a diferença de alcance de ambos os sistemas. Desta forma, utilizamos transformações nas matrizes relacionadas a visualização para ajustar o sistema de coordenadas de ambos, projetando o modelo de forma coerente na tela [12].

O processo para produzir a visualização 2D de uma cena 3D na janela é análogo ao de tirar uma foto. Enumerando os passos para tirar a foto, devemos:

1. Ajustar a posição da câmera, com as lentes apontando para o local desejado (transformações na Matriz de *viewing*, matriz responsável pela orientação do sistema e volume de alcance da renderização);
2. Mover o objeto para posição desejada (transformações do modelo: translação, rotação, redimensionamento);

3. Escolher a lente e ajustar o zoom (transformações nas Matrizes de Projeção, matriz responsável pela manipulação de coordenadas, se o objeto estará próximo ou distante de algum ponto de referência);
4. Tirar a foto (aplicar transformações) e;
5. Escolher tamanho da foto (transformações na Matriz de *viewport*, matriz responsável pela conversão de coordenadas do sistema para coordenadas da janela);

Para a visualização de um sistema 2D [34], que é um caso particular do 3D, é necessário ajustar somente a Matriz de Projeção Ortogonal. Esta matriz indica que um dos eixos não será utilizado (normalmente o eixo  $z$ ), mantendo o valor de  $-1$  e  $1$  para seus limites, portanto não há perspectiva. Ela é ajustada determinando os limites dos eixos  $x$  e  $y$ , sendo necessário também calcular o aspecto da visualização de acordo com o tamanho em *pixel* da janela de visualização.

### 2.1.3 Aspectos Educacionais

Educação é um conjunto de normas que desenvolvem hábitos, costumes e valores de uma sociedade, ou seja, é um processo contínuo de desenvolvimento das faculdades físicas, intelectuais e morais do ser humano. Em estabelecimentos de ensino, a educação é entregue aos indivíduos com intuito de desenvolver o raciocínio lógico e crítico dos alunos. O modelo predominantemente empregado nestes estabelecimentos é o modelo expositivo, o ensino está centrado no professor e o aluno transcreve nas avaliações o conhecimento adquirido pelos professores [43]. Entretanto, segundo o Dr. em Psicologia Social Douglas Tavares Borges Leal: “A aula expositiva caracteriza-se pela autoridade do professor diante de seu aluno, provocando sérios problemas de comunicação”. O Dr. também afirma que: “O conteúdo a ser aprendido é apresentado ao aprendiz na sua forma final e a tarefa da aprendizagem não envolve nenhuma descoberta independente por parte do estudante”. Questiona-se, então, a qualidade de informação que o aluno absorve, uma vez que não é medido a capacidade crítica do aluno, e sim a capacidade de replicar informação [44].

Nos cursos de computação, mesmo sendo uma área em constante atualização, também é aplicado esta metodologia de ensino. Devido a restrições deste método, observa-se que os principais problemas no ensino de programação são [45]:

- A prática pode não ser suficiente para o aluno estudar o necessário para programar;
- O déficit por parte do aluno de habilidades lógicas e matemáticas dificultam detecção de erros do próprio código;

- É exigido alto nível de abstração durante o desenvolvimento do programa, precisa-se observar a sintaxe da linguagem e a construção do algoritmo e;
- Problemas pessoais, acarretando em falta de motivação.

Desta forma, observa-se que é necessário uma abordagem diferente para o ensino de lógicas e linguagens de programação [45], de forma que o aluno esteja sempre sendo motivado para programar e assim consolidar a lógica e estruturação de um programa. Dependendo do problema a ser solucionado, a forma textual pode não ser a melhor maneira de apresentar um programa computacional, então um dos recursos utilizados é apresentá-los de forma visual. Forçando aos alunos a questionarem *o que aconteceria se* na análise do algoritmo, a visualização do programa funciona como catalisador de aprendizagem [46], transformando a programação em um ato lúdico.

Lúdico, do latim *ludus*, significa brincar. Uma atividade lúdica, no contexto pedagógico, significa flexibilidade e oferece ao aluno a experiência de situação-problema usando regras de um jogo, o qual deve ser resolvido a partir de lógica e raciocínio. Atividades lúdicas funcionam com sucesso no contexto de educação infantil [47], porém também possuem resultados satisfatórios no contexto do ensino superior [45].

O ensino de programação envolve, além de apresentar uma linguagem de programação ao aluno, consolidar o conceito de algoritmo. O termo algoritmo é usado em computação para descrever passos finitos e efetivos para a resolução de problemas que são adequados para serem implementados em um computador. Quando escrevemos um programa, nós descrevemos um modo de resolver um determinado problema. Este modo independe da linguagem de programação usada para escrever o código [7].

Entre os diversos cursos existentes de computação, alguns tópicos costumam ser abordados no mesmo período (Anexo III), sendo os principais:

- Tipos de dados;
- Estruturas de controle;
- Subalgoritmos;
- Estruturas de dados n-dimensionais e;
- Recursão.

### **2.1.3.1 Tipo de dados**

Todas as variáveis que serão utilizadas ao longo do programa possuem propriedades de manipulação, como propriedades aritméticas e propriedades lógicas. Na linguagem C, a declaração de variáveis indica essa propriedade, podendo ser numérica, como inteiros ou pontos-flutuantes, e literais, como caracter [48].

### **2.1.3.2 Estruturas de controle**

#### **2.1.3.2.1 Estruturas condicionais**

Estruturas condicionais provocam um desvio no fluxo de execução do programa. [7]. Dada uma expressão, ele executará um código dentro de um bloco se a expressão for verdadeira, caso contrário, não executará e irá pular este bloco, continuando o fluxo do programa. Por exemplo, na linguagem C as estruturas como *if*, *else* e *switch* executam essa função.

#### **2.1.3.2.2 Estruturas de repetição**

Estruturas de repetição também provocam desvio no fluxo de execução do programa, entretanto executará o bloco enquanto a condição for verdadeira [7]. Por exemplo, na linguagem C as estruturas como *for*, *while* e *do while* executam essa função.

### **2.1.3.3 Estruturas de dados multidimensionais**

#### **2.1.3.3.1 Unidimensional**

Vetores, estrutura de dado unidimensional, armazenam uma sequência de valores de mesmo tipo, sendo estes valores acessados de forma indexada. Se o vetor possui tamanho  $N$ , então seus índices variam de 0 à  $N - 1$  [7].

#### **2.1.3.3.2 N-dimensional**

Matrizes, estrutura de dado n-dimensional, armazenam uma sequência de vetores de mesmo tipo, também acessando os vetores de forma indexada [7]. Por exemplo, uma matriz bi-dimensional é um vetor de vetor, uma matriz tri-dimensional é um vetor de vetor de vetor, e assim sucessivamente. Para uma matriz bi-dimensional, faz-se uma abstração de armazenar os dados em linhas e colunas [48].

#### 2.1.3.4 Subalgoritmos

Subalgoritmos são algoritmos que executam uma atividade específica. Se há tarefas repetidas durante todo o algoritmo, esses trechos podem ser substituídos por chamadas de subalgoritmos, diminuindo o tamanho do código fonte e aumentando legibilidade do código [49]. Por exemplo, em C existem funções, que recebem parâmetros e retornam uma informação ao trecho que o chamou. Estas funções são de tipos conhecidos de dados ou pode ser do tipo *void*, que não retorna nada.

#### 2.1.3.5 Recursão

Funções podem ser usadas de modo recursivamente, ou seja, que chamam a si mesmas. A cada chamada de função obtém um novo conjunto de variáveis, independente das atribuições anteriores [48].

## 2.2 Trabalhos Correlatos

No atual currículo do Bacharelado em Ciência da Computação da **UnB**, há somente duas matérias optativas que utilizam alguma modelagem gráfica para o desenvolvimento de aplicações. São elas: **Introdução a Desenvolvimento de Jogos (IDJ)** e **Princípios de Computação Gráfica (PCG)**; a primeira utiliza a **API SDL** e a segunda a **API OpenGL**.

Baseado nas ementas destas disciplinas disponibilizadas pelo Matrícula Web, **IDJ** exige que o aluno esteja pelo menos no 4º semestre e **PCG**, no 3º semestre, devido aos pré-requisitos de cada disciplina. Tanto **IDJ** quanto **PCG** requerem o domínio de conteúdos que são apresentados em **Programação Orientada a Objetos (POO)**, **Estrutura de Dados (ED)** e Álgebra Linear, inviabilizando o uso em **APC** de modelagem gráfica do modo que são apresentados em **IDJ** e **PCG**.

Pesquisando ementas de outras universidades que oferecem o curso de computação, foram encontradas cinco que, em sua primeira matéria de programação, permitem ao aluno a visualização gráfica de seus programas, seja durante todo o semestre ou só em parte dele. A Tabela 2.1 ilustra as universidades encontradas.

Buscando alternativas para o uso de modelagem gráfica nos dois primeiros semestres de curso que a linguagem de programação seja C/C++, foram encontradas quatro soluções gráficas:

- PlayLib, desenvolvida na PUC/Rio [50];

Tabela 2.1: Universidades que utilizam recursos de computação gráfica para auxílio no ensino.

<b>Universidade</b>	<b>Curso</b>	<b>Linguagem</b>	<b>Quando é usado</b>	<b>Como é usado</b>
The University of New Mexico	CS 105L: Introduction to Computer Programming using Javascript	Javascript	Do início ao fim do semestre	Utiliza o canvas do HTML5 para desenhar figuras geométricas com Javascript
Standford University	CS106A: Programming Methodology	Java	Do início ao fim do semestre	Utiliza a linguagem Karel, baseada em Java, para auxiliar alunos na resolução de problemas gráficos
Carmegie Mellon University	15110: Principles of Computing	Python	No final do semestre	Utiliza biblioteca tkinder no laboratório de recursão e de criação de jogos simples
University of California, Berkeley (UCB)	COMPSCI W10 The Beauty and Joy of Computing	Scratch	Do início ao fim do semestre	Utiliza a linguagem BYOB, baseada em Scratch, para ensinar algoritmos básicos e até criação de jogos simples
ETH Zurich (Swiss Federal Institute of Technology)	252-0835-00L, Computer Science I	C++	Da metade ao fim do semestre	Utiliza biblioteca própria Turtle-Grafik Library para auxiliar manipulação de imagens, recursão e resolução de problemas gráficos

- WinBGIm [51];
- API SDL [52];
- Turtle Grafik, desenvolvida na ETH Zürich[53].

## 2.2.1 Soluções alternativas

### 2.2.1.1 PlayLib

A PlayLib é uma biblioteca gráfica que tem como objetivo simplificar o processo de criação de aplicativos gráficos, bem como auxiliar a aprendizagem dos fundamentos da linguagem C e solidificar técnicas de programação aplicadas em desenvolvimento de jogos. A biblioteca foi criada utilizando a [API OpenGL](#) e toda sua manipulação se refere ao plano 2D. Em seu desenvolvimento, foi utilizado a linguagem C++ e seu uso está restrito ao sistema operacional Windows.

Listagem 2.1: Abrir uma janela simples usando PlayLib

```

1 #include "Graphics.h"
2 Graphics graphics;
3
4 void MainLoop() { }
5
6 int main(void) {
7     graphics.CreateMainWindow(800, 600, "Teste");
8     graphics.SetMainLoop(MainLoop);
9     graphics.StartMainLoop();
10    return 0;
11 }
```

### 2.2.1.2 WinBGIm

BGI, *Borland Graphics Interface*, é uma biblioteca gráfica fornecida com compilador Borland C [54], o qual foi projetado para utilizar sistemas operacionais baseados em DOS. Foi criado inicialmente com intuito de plotar gráficos. A WinBGIm é uma extensão da BGI que, além de usar as funções da própria BGI, também utiliza funções do Windows para facilitar a criação destes gráficos [55].

Listagem 2.2: Abrir uma janela simples usando WinBGIm

```

1 #include "stdio"
2 #include "iostream"
3 #include "graphics"
4
```

```

5 using namespace std;
6
7 int main( ){
8     initwindow( 800 , 600 , "Teste" );
9     while( !kbhit() );
10    closegraph( );
11    return 0 ;
12 }
```

### 2.2.1.3 SDL

A SDL é uma biblioteca gráfica criada visando portabilidade de código [52], sendo ela multiplataforma. Ela é dividida em cinco subsistemas: vídeo, joystick, áudio, CD e temporizador. Cada um desses sistemas pode ser trabalhado independentemente, o que torna a integração com outras bibliotecas fácil de ser realizado. Isso permite, por exemplo, usar a OpenGL para renderizar gráficos 3D e a SDL para lidar com outros tipos de eventos, já que seu subsistema de vídeo é exclusivamente 2D [56].

Listagem 2.3: Abrir uma janela simples usando SDL

```

1 #include <SDL.h>
2
3 const int WINDOW_WIDTH = 800;
4 const int WINDOW_HEIGHT = 600;
5 const char* WINDOW_TITLE = "Teste";
6
7 int main(int argc, char **argv){
8     SDL_Init( SDL_INIT_VIDEO );
9
10    SDL_Surface* screen = SDL_SetVideoMode( WINDOW_WIDTH, WINDOW_HEIGHT, 0,
11        SDL_HWSURFACE | SDL_DOUBLEBUF );
12    SDL_WM_SetCaption( WINDOW_TITLE, 0 );
13
14    SDL_Event event;
15    bool gameRunning = true;
16
17    while (gameRunning) {
18        if (SDL_PollEvent(&event)){
19            if (event.type == SDL_QUIT){
20                gameRunning = false;
21            }
22        }
23    }
24
25    SDL_Quit();
26
27    return 0;
28 }
```

### 2.2.1.4 Turtle Grafik

O Turtle Grafik é um código em C++ criado pelo professor Felix Friedrich, da Universidade ETH Zürich. Ao adicionar o arquivo *turtle.cpp* no projeto e executar umas das funções dele, tendo como resultado dessa adição um código multiplataforma, a cada chamada de função ele salva um arquivo bmp chamado *turtle\_out.bmp* com o resultado gráfico da chamada. Não há como abrir janela de contexto.

Como o nome sugere, o Turtle Grafik utiliza o método de desenho computacional *Turtle Graphics*, que é preservado o estado inicial da tartaruga e, a partir dele, move-se a tartaruga em um sentido, direção e magnitude num plano cartesiano [57].

## 2.2.2 Comparação entre as alternativas e a proposta

Para exemplo de comparação, as Listagens 2.1, 2.2, 2.3 e 2.4 mostram como abrir uma janela de contexto gráfico usando a PlayLib, a WinBGI, a SDL e a PLAYAPC, respectivamente.

Listagem 2.4: Abrir uma janela simples usando PLAYAPC

```
1 #include <playAPC/playapc.h>
2
3 int main( ){
4     AbreJanela(800, 600, "Teste");
5     Desenha();
6     return 0;
7 }
```

Apesar de tanto a PlayLib quanto a PLAYAPC serem baseadas na OpenGL e ambas serem 2D, além de terem quase a mesma motivação, há algumas diferenças cruciais:

- A PLAYAPC:
  - possui guia de instalação para qualquer sistema operacional, enquanto a PlayLib só disponibilizou a instalação para Windows;
  - não exige do aluno o conceito de *callbacks*, ou sequer o conceito de objetos ou de classes. Apesar de também utilizar C++ em seu desenvolvimento e Orientação a Objetos, sua interface foi criada visando sua utilização de forma estruturada, paradigma usado em APC na UnB.
- A PlayLib possui manipulação de texto e áudio, tornando-a conveniente para o desenvolvimento de pequenos jogos, diferente da PLAYAPC, a qual é utilizada somente para renderização de figuras geométricas.

Entre a biblioteca WinBGIm e a PLAYAPC, as diferenças basicamente se resumem ao desempenho

- Uma vez que WinBGIm utiliza o DOS, resgatando do disco drivers gráficos e as fontes de texto, ele gera uma janela renderizada de forma totalmente independente. A PLAYAPC exige que o computador tenha instalado alguma placa gráfica para poder renderizar a cena, e que tenha suporte a OpenGL.
- Como o WinBGIm foi feita com o propósito inicial de plotar gráficos, ela não possui suporte nativo para animações, sendo uma alternativa fazer a animação frame a frame limpando toda a cena e criado novos objetos em novos lugares, dando a sensação de animação. A PLAYAPC possui suporte nativo a animações, uma vez que é possível utilizar funções de transformações lineares como *Move*, *Gira* e *Redimensiona* dentro de um laço criado pelo aluno, não tendo a necessidade de recriar todas as geometrias da cena a cada iteração.

Entre a SDL e a PLAYAPC, as diferenças são diversas:

- A PLAYAPC esconde o laço de renderização e o tratamento de eventos, ambos encapsulados pelas funções *Desenha* e *Desenha1Frame*.
- Ao contrário da SDL, a PLAYAPC não exige que o aluno saiba, nos primeiros contatos com a biblioteca, o uso de ponteiros ou de objetos.
- Apesar de não estar implementado, a extensão para manipulação de objetos 3D é possível usando a PLAYAPC, diferente da SDL.
- A SDL pode ser utilizada tanto em C quanto em C++. Usando a PLAYAPC em um programa C, a compilação tem que usar a *toolchain* do C++.
- Apesar da SDL ser uma API gráfica exclusivamente 2D, seu desempenho de renderização é inferior ao da OpenGL, usada na PLAYAPC [56].

Entre a biblioteca Turtle Grafik e a PLAYAPC, as diferenças são suas restrições:

- A Turle Grafik:
  - por utilizar a metodologia *Trutle Graphic*, não possui a liberdade de desenhar duas ou mais geometrias separadas;
  - é limitada para desenhar figuras geométricas e salvá-las em uma imagem, não tendo a possibilidade de animação;

- apenas renderiza figuras geométricas, já a **PLAYAPC**, além de renderizar figuras geométricas, também lê imagens e possui entrada de teclado e mouse;
- Por ter sido disponibilizado o código fonte para ser adicionado ao projeto e usar somente bibliotecas padrão do C++, ela não possui problemas de compatibilidade entre diversos as diversas plataformas existentes, bastando tão somente compilá-lo junto com o projeto. A **PLAYAPC**, mesmo oferecendo o código fonte, não é tão compacta a ponto de ser adicionada num projeto do aluno e ser compilada, sendo preferível utilizar os binários já compilados de acordo com o sistema operacional.

# **Capítulo 3**

## **Solução Proposta**

### **3.1 Projeto da Biblioteca**

#### **3.1.1 Levantamento de requisitos**

Realizando reuniões semanais com professores envolvidos no projeto, além de apresentar à outros professores que possivelmente estariam interessados no projeto, foi utilizado a técnica de *brainstorm* para o levantamento de requisitos da biblioteca.

A princípio, ela deveria ser aplicada em cada tópico abordado em **APC**, sendo utilizada de forma lúdica e didática do início ao final do semestre. A biblioteca deve ser capaz de acompanhar a evolução gradativa de complexidade dos programas desenvolvidos pelos alunos.

Foram reunidos cinco grandes funcionalidades que a biblioteca deveria ter inicialmente, sendo elas:

- Renderização de formas geométricas;
- Animação;
- Renderização de imagens;
- Entrada de mouse e teclado e;
- Outras, no intuito de auxiliar o aluno a manipular as demais funções da biblioteca;

Sendo todas as funcionalidades capazes de serem utilizadas em qualquer período ao longo do semestre.

Com o avanço das reuniões, viu-se necessário pesquisar quais os tópicos da primeira matéria da computação em outras universidades eram abordados para a elaboração de uma apostila de exercícios (Apêndice A) genérica, não apenas focada em APC da UnB. A Figura 3.1 ilustra uma pesquisa realizada com 25 universidades, das quais foram analisados as ementas e planos de aulas (Anexo III), fazendo uma relação entre os conteúdos abordados com a quantidade de universidades que os abordam. A Tabela 3.1 indica as universidades que foram pesquisadas.

Dentre os tópicos práticos, foram selecionados os seis mais recorrentes para a elaboração de exercícios, os quais são:

- Tipos de dados;
- Estruturas de controle;
- Subalgoritmos;
- Vetores;
- Matrizes e;
- Recursão.

Apesar da PLAYAPC poder potencialmente atender a qualquer um dos tópicos da disciplina APC, no presente trabalho restringiu-se seu uso a estes seis tópicos mais relevantes.

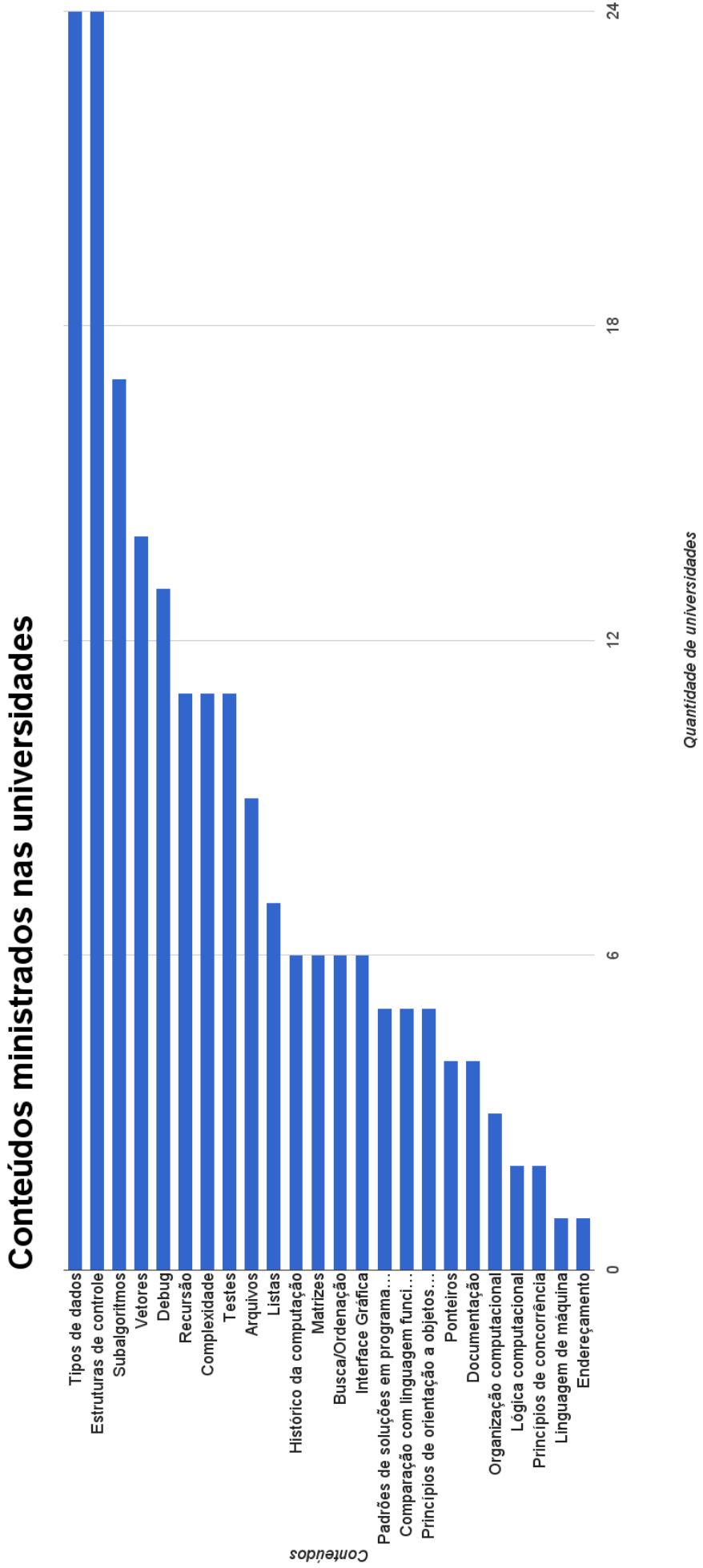


Figura 3.1: Conteúdos ministrados em relação a quantidade de universidades que os abordam.

Tabela 3.1: Universidades pesquisadas.

<b>Universidade</b>	<b>Curso</b>
UnB	Algoritmos e programação de computadores
USP	MAC0110 - Introdução à Computação
Unicamp	MC102 - Algoritmos e Programação de Computadores
Massachusetts Institute of Technology (MIT)	6.00 Introduction to Computer Science and Programming
University of Waterloo	CS 137 Programming Principles
University of Michigan	EECS 183. Elementary Programming Concepts
Vanderbilt University	CS 1101 [Formerly CS 101] Programming and Problem Solving
PUC-Rio	Programação para Informática 1
Hiroshima University	Introduction to Computer Programming
The University of New Mexico	CS 105L: Introduction to Computer Programming using Javascript
Stanford University	CS106A: Programming Methodology (Java)
University of Oxford	Imperative Programming
Carnegie Mellon University	15110 Principles of Computing
Havard University	Introduction to Computer Science Programming in Python (6.0001)
University of California, Berkeley (UCB)	COMPSCI W10 The Beauty and Joy of Computing
University of Cambridge	Paper 1: Foundations of Computer Science
The Hong Kong University of Science and Technology (HKUST)	COMP 1022P Introduction to Computing with Java
ETH Zurich (Swiss Federal Institute of Technology)	252-0835-00L Computer Science I
Princeton University	COS 126 / EGR 126 (QR) General Computer Science
National University of Singapore (NUS)	CS1101S - Programming Methodology
The University of Hong Kong	ENGG1111 Computer Programming and Applications
The University of Melbourne	COMP10001 Foundations of Computing
Imperial College London	CO120.1-Programming I
University of Toronto	CSC108H1: Introduction to Computer Programming

### 3.1.2 Requisitos

#### 3.1.2.1 Requisitos não-funcionais

1. Tecnologia usada deve ser a OpenGL;
2. Linguagem de desenvolvimento deverá ser C++;
3. As plataformas que deverão ser compatíveis com a biblioteca deverão ser Windows, Linux (baseados em sistemas Debian) e Mac e;
4. O sistema de referência deverá ser exclusivamente o sistema de coordenadas cartesiano.

Devido a tecnologia escolhida ser a OpenGL, é obrigatório para o funcionamento da biblioteca que tenha uma função que inicialize a janela de contexto e outra que renderize o contexto.

##### 3.1.2.1.1 Inicializar janela de contexto

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

A renderização da `PLAYAPC` utiliza unidades do plano cartesiano, não unidades em *pixel*. Desta forma, todos os cálculos referentes a posicionamento de geometrias não dependem do tamanho da janela em *pixel*, mas sim da posição no plano cartesiano, onde o centro sempre estará no meio da janela.

A função *AbreJanela* 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 *MudaLimitesJanela* antes de chamar *AbreJanela*. Seu primeiro argumento se refere a largura em *pixels* da janela, o segundo a altura em *pixels* e o terceiro se refere ao nome que a janela terá.

O funcionamento desta função dá-se pela seguinte maneira: dado a largura e altura da janela em *pixels*, é necessário descobrir a proporção desta janela para que o sistema respeite as coordenadas de um plano limitado de -100 à 100 unidades, tanto em largura quanto em altura. Esta proporção é encontrada realizando uma regra de três simples como indica a Equação 3.1, sendo  $\frac{tam}{2}$  é a menor dimensão da janela em *pixels* dividido por 2, 100 é a quantidade de unidades total de um dos lados do plano e *x* é a quantidade de *pixels* de uma unidade do plano de renderização.

$$\begin{array}{rcl} \frac{tam}{2} & - & 100 \\ x & - & 1 \end{array} \quad (3.1)$$

Como exemplo, considere o caso particular ilustrado na Figura 3.2. A janela tem  $800\ pixels$  de largura e  $600\ pixels$  de altura e deseja-se renderizar um ponto no lado mais a direita da janela, com  $y = 0$ . A Figura 3.3 ilustra este resultado.

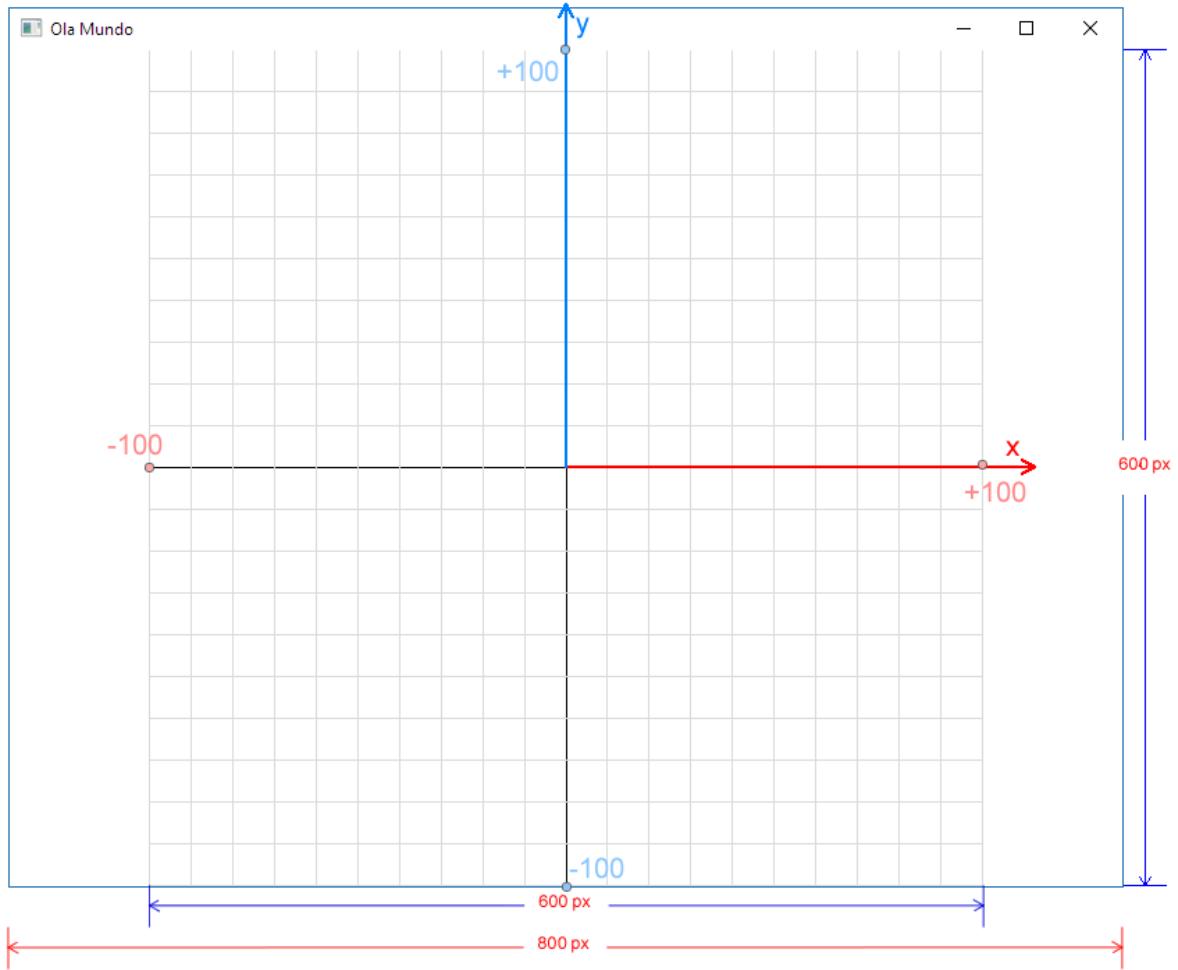


Figura 3.2: Caso quando  $largura \neq altura$ . A exibição plano cartesiano está limitado de  $-100$  unidades à  $100$  unidades, tanto em  $x$  quanto em  $y$ , onde  $100$  unidades equivalem à  $300\ pixels$ , ou  $200$  unidades equivalem à  $600\ pixels$ .

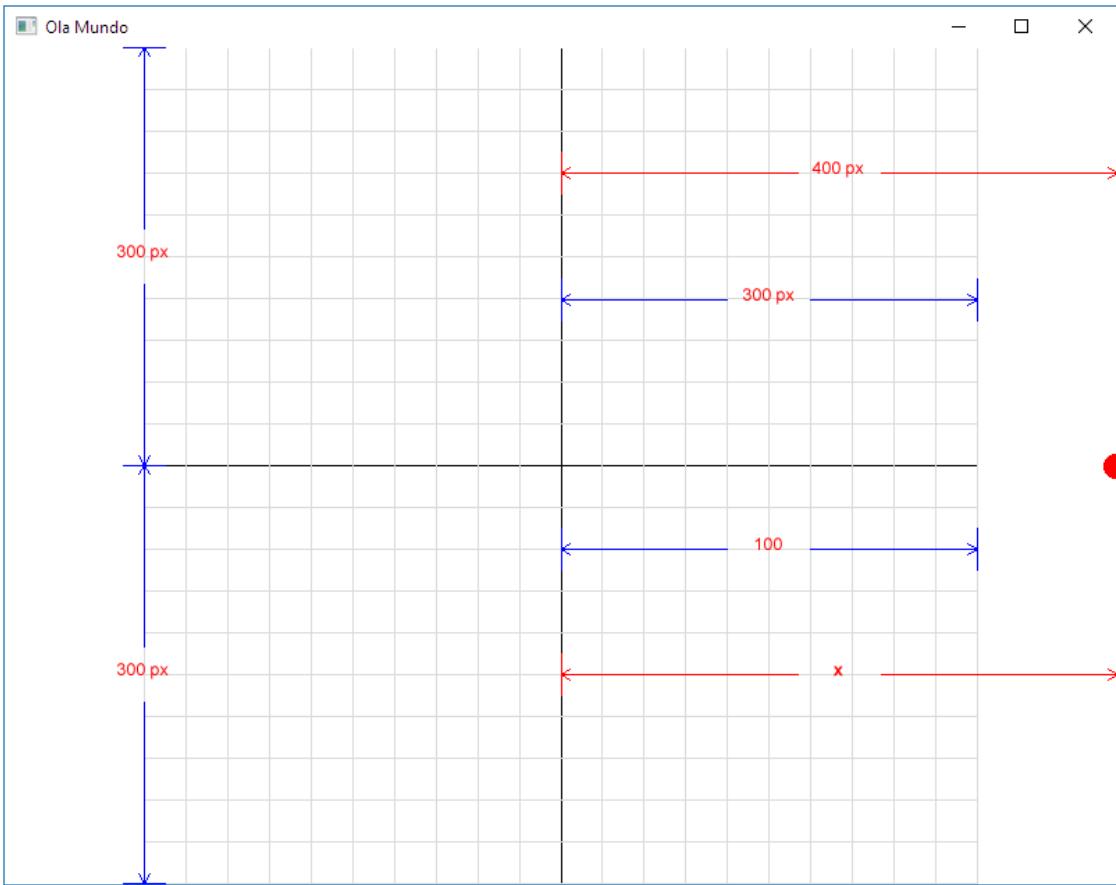


Figura 3.3: Renderização de um ponto vermelho em  $(133, 0)$ . O ponto está exagerado para melhor visualização.

Queremos, então, que o ponto fique à *400 pixels* de distância à partir do centro no eixo *x*. É necessário obter essa posição em unidades do plano cartesiano. Este cálculo é feito com uso da Equação 3.2, derivada da Equação 3.1, onde 300 é a distância ao longo do eixo *x* ou *y* em *pixels* do centro até a borda do plano cartesiano, 400 é a posição, também em *pixels*, no eixo *x* onde se deseja renderizar o ponto e 100 é a medida em unidades do centro até a borda do plano cartesiano.

$$\begin{array}{rcl} 300 & - & 100 \\ 400 & - & x \end{array} \quad (3.2)$$

Desta forma, encontramos que o valor desejado de  $x \cong 133$ .

### 3.1.2.1.2 Renderizar contexto

```
void Desenha()
```

A função *Desenha* realiza o laço 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 laço 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.

```
int Desenha1Frame()
```

A função *Desenha1Frame* renderiza pelo menos  $\frac{1}{60}$  segundos da cena. 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.

### 3.1.2.2 Requisitos funcionais para renderização de formas geométricas

1. O usuário deverá ser capaz de manipular o tipo de dado *ponto*, que possui coordenada *x* e coordenada *y*;
2. O usuário deve ser capaz de criar formas geométricas como ponto, reta, polígono, círculo, circunferência, elipse, triângulo, quadrado e retângulo, respeitando definições matemáticas;
3. O usuário, ao passar um vetor de pontos para uma função específica, deverá conseguir exibir um gráfico;
4. O usuário deve ser capaz de colorir todas as formas geométricas;
5. O usuário deve ser capaz de aplicar texturas nas formas geométricas;

#### 3.1.2.2.1 Tipo de dado Ponto

```
struct Ponto{  
    float x;  
    float y;  
}
```

Ponto 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  $!=$ .

•  $=$

```
Ponto p1, p2;  
(...)
```

```
p1 = p2;
```

•  $+$  (ou  $-$ )

```
Ponto p1, p2, p3;  
(...)
```

```
p1 = p2 + p3;
```

•  $+ =$  (ou  $- =$ )

```
Ponto p1, p2;  
(...)
```

```
p1 += p2;
```

•  $==$  (ou  $!=$ )

```
Ponto p1, p2;  
(...)
```

```
if (p1 == p2){  
(...)  
}
```

### 3.1.2.2.2 Forma geométrica Ponto

```
int CriaPonto(Ponto p)
```

A função *CriaPonto* 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*.

### **3.1.2.2.3 Forma geométrica Reta**

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

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

### **3.1.2.2.4 Forma geométrica Polígono**

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

A função *CriaPoligono*, 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 por conta da OpenGL. Caso haja a necessidade de criação de figuras não-convexas, será necessário "quebrar" a geometria não-convexa em duas ou mais geometrias convexas.

```
int CriaPoligonoVetor(short int index, Ponto *p)
```

A função *CriaPoligonoVetor* cria e retorna o mesmo tipo de geometria que a *CriaPoligono*. Entretanto, seu diferencial são seus argumentos: seu primeiro argumento é o tamanho do vetor e o segundo argumento o vetor propriamente dito.

### **3.1.2.2.5 Forma geométrica Círculo**

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

A função *CriaCirculo*, 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.

### **3.1.2.2.6 Forma geométrica Circunferência**

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

A função *CriaCircunferencia*, cria uma geometria do tipo *CIRCUNFERENCIA*, retornando um índice deste tipo de geometria. Seu primeiro argumento é o tamanho do raio e o segundo argumento é onde estará centrada a circunferência.

### **3.1.2.2.7 Forma geométrica Elipse**

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

A função *CriaElipse*, 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.

### **3.1.2.2.8 Forma geométrica Triângulo**

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

A função *CriaTriangulo* 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.

### **3.1.2.2.9 Forma geométrica Quadrado**

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

A função *CriaQuadrado* 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.

### **3.1.2.2.10 Forma geométrica Retângulo**

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

A função *CriaRetangulo* 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.

### **3.1.2.2.11 Renderização de gráficos**

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

Tabela 3.2: 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

A função *CriaGrafico* 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 3.2.

### 3.1.2.2.12 Colorir formas geométricas

A função *Pintar* pode ser utilizada de duas formas diferentes.

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

A última geometria criada receberá a cor definida utilizando o sistema de cores RGB. O primeiro argumento é o componente vermelho, o segundo o componente verde e o terceiro o componente azul, todos variando de 0 à 255.

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

Apenas a geometria do tipo *nome* com índice definida por *index* será pintada com as cores especificadas pelos valores *red*, *green* e *blue*. O quarto argumento é o tipo de geometria, podendo variar de acordo com a função com prefixo *Cria* utilizado, o quinto argumento é o retorno das funções com prefixo *Cria*. De forma resumida, o argumento *nome*, do tipo *geometrias\_validas*, pode receber os valores descritos na Tabela 3.3.

Tabela 3.3: Valor de *nome* da função *Pintar* e *AssociaImagem*.

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

### 3.1.2.2.13 Mostrar imagens em formas geométricas

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

A função *AbreImagem* abre uma imagem e a deixa-a pronta para ser vinculada a alguma geometria. Seu primeiro parâmetro é o caminho da imagem e ela retorna o índice de textura para ser vinculado.

Para vincular uma imagem a uma geometria, existem duas formas de usar a função *AssociarImagem*

```
void AssociaImagem( int textura )
```

A última geometria criada receberá a imagem associada àquela textura. Seu único argumento é o índice de textura, retorno da função *AbreImagem*.

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

Apenas a geometria do tipo *nome* com índice definida por *index* receberá a imagem associada àquela textura. O segundo argumento é o tipo de geometria, podendo variar de acordo com a função com prefixo *Cria* utilizado, o terceiro argumento é o retorno das funções com prefixo *Cria*. O argumento *nome*, do tipo *geometrias\_validas*, pode receber os valores descritos na Tabela 3.3.

### 3.1.2.3 Requisitos funcionais para animação

1. O usuário deve ser capaz de realizar animações dentro de um laço de repetição;

2. O usuário deverá ser capaz de movimentar, girar e redimensionar uma ou mais geometrias e;
3. O usuário deverá ser capaz de limpar cenas ou geometrias criadas;

### **3.1.2.3.1 Renderizar cena dentro de um laço de repetição**

Utilizando a função *Desenha1Frame* dentro de um laço de repetição, ela irá renderizar  $\frac{1}{60}$  segundos da cena. Sendo assim, pode ter definições de novas geometrias ou transformações antes ou depois da função ser chamada.

### **3.1.2.3.2 Agrupando mais de uma geometria**

Para a animação de um conjunto de geometrias, se faz necessário a criação de um grupo.

```
int CriaGrupo ( )
```

A função *CriaGrupo* agrupa todo um conjunto de geometrias, associando todas a um único índice, em um único conjunto. Desta forma, é possível transformar um conjunto de geometrias de forma independente, apenas referenciando o índice do grupo. Todas as geometrias criadas após a chamada desta função pertencerão ao mesmo grupo.

Caso o programa não utilize esta função para criação de um grupo, todas as geometrias definidas pertencerão ao mesmo grupo.

### **3.1.2.3.3 Movimentação de uma ou mais geometrias**

Há duas maneiras de utilizar a função *Move*: ou ela translada o último grupo criado ou ela translada um grupo específico. Independente de qual grupo que sofrerá a ação de movimentar, todas as geometrias do grupo serão transladas até que o ponto de referência da primeira geometria deste grupo esteja no novo ponto desejado.

```
void Move( Ponto p)
```

Seu argumento é o ponto no plano cartesiano que se deseja mover as geometrias, onde ele irá movimentar o último grupo definido.

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

Seu primeiro argumento é o ponto no plano cartesiano que se deseja mover as geometrias e o segundo argumento o índice do grupo para ser movimentado, índice criado pela função *CriaGrupo*.

### **3.1.2.3.4 Rotação de uma ou mais geometrias**

Há duas maneiras de utilizar a função *Gira*: ou ela rotaciona o último grupo criado ou ela rotaciona um grupo específico. Independente de qual grupo que sofrerá a ação de girar, todas as geometrias do grupo serão rotacionados até que o ponto de referência da primeira geometria deste grupo esteja no novo ponto desejado.

```
void Gira( float theta )
```

Seu argumento é o ângulo em graus que o último grupo criado deverá ser rotacionada. O grupo irá rotacionar em relação ao plano cartesiano, não ao seu próprio eixo.

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

Seu primeiro argumento é o ângulo em graus que o grupo com o índice *index* será rotacionado. O grupo irá rotacionar em relação ao plano cartesiano, não ao seu próprio eixo.

### **3.1.2.3.5 Redimensionar uma ou mais geometrias**

Há duas maneiras de utilizar a função *Redimensiona*: ou ela redimensiona o último grupo criado ou ela redimensiona um grupo específico. Independente de qual grupo que sofrerá a ação de redimensionar, todas as geometrias do grupo serão redimensionadas com o fator passado em *x* e em *y*.

```
void Redimensiona( float x , float y )
```

Seu primeiro argumento é o fator em *x* e seu segundo argumento é o fator em *y* de distorção que as geometrias do último grupo criado sofrerão.

```
void Redimensiona( float x , float y , int index )
```

Seu primeiro argumento é o fator em *x*, seu segundo argumento é o fator em *y* e seu terceiro argumento é o grupo que as geometrias serão distorcidas.

### **3.1.2.3.6 Limpar a cena**

```
void ApagaGrupo( int index )
```

A função *ApagaGrupo* destrói todas as geometrias do grupo *index*. Seu argumento é o grupo que deverá ser apagado, que é retorno da função *CriaGrupo*.

```
void LimpaDesenho()
```

Limpa toda a cena, destruindo grupos e geometrias. Porém, não altera cor de fundo nem limites do eixo de coordenadas.

### **3.1.2.4 Requisitos funcionais para entrada de mouse e teclado**

1. O usuário deve informar uma tecla ou botão do mouse para uma função e esta deve retornar se ela foi pressionada e;
2. O usuário deve ser informado qual a posição do mouse em relação ao plano cartesiano;

#### **3.1.2.4.1 Entrada de teclado**

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

A função *ApertouTecla* retorna 1 se o usuário pressionou ao menos uma vez a tecla *tecla* e retorna 0 caso contrário. Seu argumento é uma das possíveis teclas do teclado, listados na Tabela 3.4.

```
int RetornaTecla()
```

Ao usuário digitar qualquer tecla reconhecível pela PLAYAPC, esta função retorna qual tecla foi essa, listada na Tabela 3.4.

#### **3.1.2.4.2 Entrada de mouse**

```
int ApertouBotaoMouse( int bt )
```

A função *ApertouBotaoMouse* retorna 1 se o usuário pressionou ao menos uma vez o botão *bt* e retorna 0 caso contrário. Seu argumento é um dos possíveis botões do mouse, listados na Tabela 3.5.

```
int RetornaBotaoMouse()
```

Ao usuário apertar qualquer botão reconhecível pela PLAYAPC, esta função retorna qual botão foi esse, listado na Tabela 3.5.

```
void PosicaoMouse( int *x, int *y )
```

A função *PosicaoMouse* retorna para *x* e *y* atual posição do mouse em relação ao plano cartesiano. Se a janela de contexto não estiver em foco ou se o mouse não estiver sobre a janela, a função irá retornará a última posição registrada. No começo da execução, é necessário que as variáveis *x* e *y* estejam inicializadas. Seus argumentos são as variáveis *x* e *y* que receberão a posição atual do mouse.

Tabela 3.4: Teclas reconhecidas pela PLAYAPC.

<b>Valor</b>	<b>Descrição</b>
'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

Tabela 3.5: Botões de mouse reconhecidos pela PLAYAPC.

<b>Valor</b>	<b>Descrição</b>
GLFW_MOUSE_BUTTON_n	Botões possíveis do mouse ( $n \in (0..8)$ )
GLFW_MOUSE_BUTTON_LEFT	Botão esquerdo ( $n \leftarrow 1$ )
GLFW_MOUSE_BUTTON_RIGHT	Botão direito ( $n \leftarrow 2$ )
GLFW_MOUSE_BUTTON_MIDDLE	Botão do meio ( $n \leftarrow 3$ )

### **3.1.2.5 Requisitos funcionais para outros tipos de funções**

1. O usuário deve ser capaz de colorir o fundo da janela
2. O usuário deve ter noção dos limites do plano cartesiano;
3. O usuário deve ser capaz de alterar os limites de renderização;
4. O usuário deve ser capaz de, dado uma imagem, extrair os componentes RGB da mesma na forma de uma matriz R, G e B e;
5. O usuário deve ser capaz de aumentar a espessura da borda de uma geometria;

#### **3.1.2.5.1 Colorir plano de fundo da janela**

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

Colore o plano de fundo da janela com a cor passada pelos parâmetros *red*, *green* e *blue*, que respeitam o sistemas de cores RGB. Seu primeiro argumento é o componente vermelho, o segundo é o verde e o terceiro o azul.

#### **3.1.2.5.2 Plano de renderização**

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

Esta é a única função da **PLAYAPC** que, se for utilizada, deve ser chamada antes da função *AbreJanela*. Ela altera os limites das coordenadas da playAPC. Caso esta função não seja chamada, os limites são por padrão de  $-100$  à  $100$ , tanto no eixo horizontal quanto no eixo vertical. Seu argumento é o limite positivo de até onde a janela de visualização deve renderizar, que será utilizado tanto no eixo vertical quanto no original indo de  $-limite$  até *limite*.

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

Exibe linhas das coordenadas da **PLAYAPC** de  $-100$  à  $100$  no eixo horizontal e vertical, caso os limites não sejam alterados. Seu centro é localizado em  $(0, 0)$  e é identificado pelas linhas pretas. As linhas cinzas indentificam o intervalo das coordenadas. Seu argumento são os espaçamentos entre as linhas em cinza. Esta função serve para auxiliar o usuário a identificar onde serão renderizadas suas geometrias.

### 3.1.2.5.3 Componentes RGB de uma imagem

```
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 ] )
```

Dado uma imagem BMP de 24 bits, a função lê a imagem e extrai os componentes vermelho, verde e azul dela. Seu primeiro argumento é o caminho da imagem, seu segundo e terceiro argumentos são a largura e altura da imagem, seu quarto, quinto e sexto argumento são as matrizes que irão receber os componentes vermelho, verde e azul, respectivamente. As matrizes precisarão ter necessariamente a quantidade de linhas e colunas maior ou igual aos parâmetros *largura* e *altura*.

### 3.1.2.5.4 Alterar espessura da borda

```
void Grafite( int espessura )
```

Rasterização, em computação gráfica, é o processo que determina quais *pixels* serão utilizados para a renderização de uma geometria.

A função *Grafite* aumenta a quantidade de linhas de rasterização da última geometria criada, variando de 1 a  $\infty$ . Seu parâmetro é quanto se deseja aumentar as linhas. 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.

## 3.1.3 Arquitetura da biblioteca

A arquitetura em camadas foi escolhida por conta de sua manutenibilidade, apesar que, dependendo do caso, a arquitetura acaba resultando em uma queda de desempenho por ter camadas que chamam um método de uma camada anterior. Um exemplo é a função *PintarFundo*, a qual está na camada mais superior, a interface da *PLAYAPC*, que chama um método da camada *Evento* que chama uma única função da *OpenGL*, a *glClear*.

Porém, suponha que fosse necessário mudar a API *GLFW* 2.7 para a *GLFW* 3.0, ou mudar para a biblioteca *GLUT*. Para isso, bastaria apenas modificar a classe *OpenGLBase*, mantendo os métodos ou adicionando novos, se fosse o caso.

A Fig. 3.4 ilustra esta arquitetura é utilizada e como cada camada se comunica.

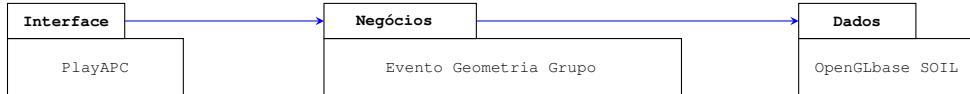


Figura 3.4: Organização da arquitetura da PLAYAPC.

- A camada de interface é a comunicação entre o usuário e a biblioteca gráfica, que trata desde a criação de objetos a serem exibidos na tela até ao laço de renderização, todos estruturados de forma sequencial, em que o usuário possa escrever um programa sem a necessidade de ter o conhecimento de orientação a objeto. Esta camada abrange o seguinte código fonte e o arquivo de cabeçalho, ambos de mesmo nome:
  - playapc
- A camada de dados é responsável pela manipulação da biblioteca GLFW, responsável pela criação de janelas com o contexto da OpenGL e gerencia os eventos para renderização, sendo chamada somente pela camada de negócios. É também nesta camada que se encontra bibliotecas de terceiros que foram incorporados à PLAYAPC. Ela possui os seguintes arquivos:
  - OpenglBase
  - SOIL
- A camada de negócios é a camada que constrói objetos, realiza transformações algébricas e gerencia chamadas para controle de renderização. Grande parte do código fonte da PLAYAPC se concentra nessa camada, pois esta abrange os seguintes códigos fontes e seus respectivos arquivos:
  - Grupo
  - Evento
  - PlanoCartesiano
  - Geometria
    - \* Circulo
    - \* Circunferencia
    - \* Elipse
    - \* Poligono
    - \* Pontinho
    - \* Reta
    - \* Retangulo

- \* Triangulo
- \* Grafico

Toda a camada de interface da PLAYAPC utiliza a metodologia de encapsulamento para simplificar o processo de criação de geometrias e sua renderização, tendo o melhor exemplo nesta camada a função *Desenha*, que esconde a informação de que esta função é responsável pelo laço de renderização.

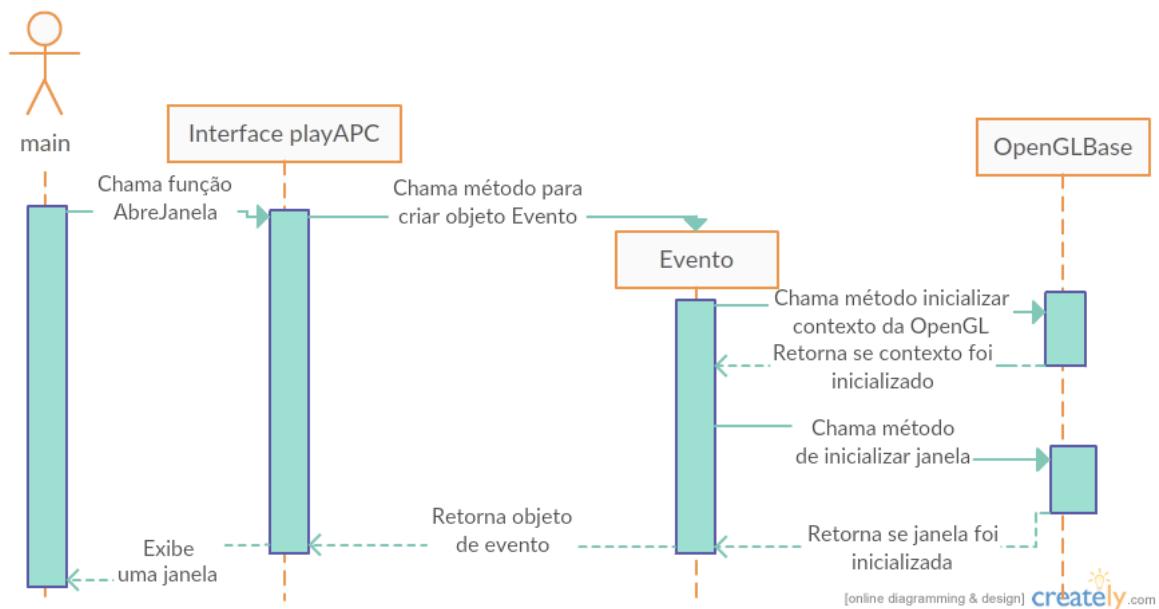


Figura 3.5: Diagrama de Sequência da função *AbreJanela*.

Como exemplo particular, ilustrado na Figura 3.5, para o funcionamento desta chamada de função *AbreJanela*, a interface da PLAYAPC cria um objeto do tipo *Evento*, sendo esta classe responsável por gerenciar todos os eventos de entrada e renderização. Note que não há a necessidade de saber como é implementado este objeto para a *main*, tornando o próprio objeto *Evento* independente de código, uma vez que sua funcionalidade foi abstraída para somente devolver ao usuário uma janela de contexto.

A PLAYAPC utiliza o método Herança com a classe Geometria, que possui métodos e atributos em comum com as subclasses Círculo, Circunferência, Elipse, Polígono, Reta, Retângulo, Ponto, Gráfico e Triângulo.

Listagem 3.1: Descrição da classe Geometria.

```

class Geometria{
    protected:
        GLfloat red, green, blue;
    
```

```

    GLfloat grafite;
    Geometria *prox;
    GLuint textura;

public:
    //Métodos virtuais
    virtual void RenderizaPontos() = 0; //Método para desenhar as
        geometrias, com glBegin() e glEnd()
    virtual Ponto CalculaDeslocamento(Ponto p) = 0;

    //Métodos em comum de todas as geometrias
    void Cor(GLfloat red = 0, GLfloat green = 0, GLfloat blue =
        0); //Define cores
    void Grafite(GLfloat grafite = 1.0); //Define grossura da
        borda
    void Agrupa(Geometria *prox); //Cria lista de geometrias
    void Desagrupa(Geometria **atual, Geometria *primeiro); ////
        Tira da lista algum elemento
    Geometria *getProx(); //Pega o próximo elemento da lista
    GLuint getTextura(); //Retorna área alocada para textura
    void setTextura(GLuint data); //Seta área alocada para
        textura
} ;

```

As classes referentes às geometrias que podem ser criadas com a PLAYAPC se diferenciam no método *RenderizaPontos* e no método de *CalculaDeslocamento*, visto que estes métodos são virtuais. Para o método de *RenderizaPontos*, cada um dos herdeiros utilizam a função da OpenGL *glBegin()* e *glEnd()* passando diferentes argumentos dependendo do tipo de geometria.

- A classe *Pontinho* é criada a partir de um único *Ponto p* passado pelo usuário e utiliza como argumento para *glBegin()* o *GL\_POINTS*.
- A classe *Reta* é criada a partir de dois pontos *Ponto p1* e *Ponto p2* passado pelo usuário e utiliza como argumento para *glBegin()* o *GL\_LINES*.
- A classe *Polígono* é criada a partir de  $n$  pontos, agrupados em uma lista, passado pelo usuário e utiliza como argumento para *glBegin()* o *GL\_POLYGON*.
- A classe *Triangulo* é criada a partir de um único ponto *Ponto p* de referência, que é o ponto do canto esquerdo inferior, um valor para a base e um valor para a altura passados pelo usuário e utiliza como argumento para *glBegin()* o *GL\_TRIANGLES*.

- A classe *Retangulo*, que é a mesma utilizada para criar quadrados com *CriaQuadrado*, é criada a partir de um único ponto *Ponto p* de referência, que é o ponto do canto esquerdo inferior, um valor para a base e um valor para a altura passados pelo usuário e utiliza como argumento para *glBegin()* o *GL\_QUADS*.
- A classe *Circulo* é criada a partir de um único ponto *Ponto p* de referência, que é o ponto do centro do círculo e um valor o raio passados pelo usuário e utiliza como argumento para *glBegin()* o *GL\_TRIANGLE\_FAN*.
- A classe *Circunferencia* é criada a partir de um único ponto *Ponto p* de referência, que é o ponto do centro da circunferência e um valor o raio passados pelo usuário e utiliza como argumento para *glBegin()* o *GL\_LINE\_LOOP*.
- A classe *Elipse* é criada a partir de um único ponto *Ponto p* de referência, que é o ponto do centro da elipse, e dois valores que correspondem a metade do lado menor e do lado maior passados pelo usuário e utiliza como argumento para *glBegin()* o *GL\_TRIANGLE\_FAN*.
- A classe *Grafico* é criada a partir de  $n$  pontos passados pelo vetor *Ponto p\** passado pelo usuário e utiliza como argumento para *glBegin()* o *GL\_LINES*.

O método *CalculaDeslocamento* é um método que, para um ponto *p* passado como argumento deste método, cada herdeiro calcula a distância que este ponto está em relação ao seu ponto de referência, seja ele o ponto esquerdo, o ponto central ou até mesmo o primeiro ponto que foi passado como argumento na construção de um objeto dele.

## 3.2 Apresentação da biblioteca

Para a apresentação da biblioteca, vamos apresentar o clássico problema da Torre de Hanói. Para melhor visualização do problema, a quantidade de discos foi limitada para 5. A Listagem 3.2 apresenta um código em C, com saída textual, com uma das possíveis soluções da torre de Hanói.

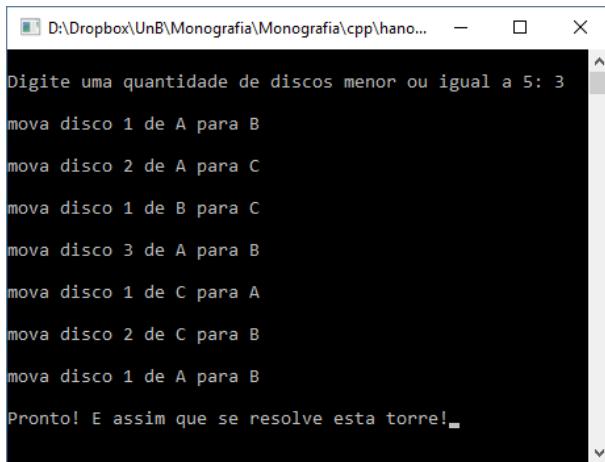


Figura 3.6: Resolução da torre de Hanói para 3 discos.

Listagem 3.2: Código fonte da Torre de Hanói.

```

1 #include <stdio.h>
2
3 void hanoi(int n, char a, char b, char c){
4     /* mova n discos do pino a para o pino b usando
5        o pino c como intermediario */
6
7     if (n == 1){
8         printf("\nmova disco %d de %c para %c\n", n, a, b);
9     }
10    else
11    {
12        hanoi(n - 1, a, c, b); // H1
13        printf("\nmova disco %d de %c para %c\n", n, a, b);
14
15        hanoi(n - 1, c, b, a); // H2
16    }
17 }
18
19 int main(void){
20     int numDiscos;
21
22     do{
23         printf("\nDigite uma quantidade de discos menor ou igual a 5: ");
24         scanf("%d", &numDiscos);
25     }while(numDiscos > 5 || numDiscos <= 0);
26
27     hanoi(numDiscos, 'A', 'B', 'C');
28
29     printf("\nPronto! E assim que se resolve esta torre!");
30
31     return 0;
32 }
```

### 3.2.1 Criação da janela de contexto da PLAYAPC

A primeira instrução necessária para usar a PLAYAPC, depois de ter instalado e linkado a biblioteca, é incluir o seu header no código do programa.

Listagem 3.3: Incluir PLAYAPC no código.

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

Agora podemos criar uma janela de contexto com a função *AbreJanela*. Também podemos pintar o fundo desta janela de branco e exibir nosso plano de renderização, o plano cartesiano.

Listagem 3.4: Abrir uma janela branca, mostrando o plano cartesiano.

```
(...)
do {
    printf("\nDigite uma quantidade de discos menor ou igual a 5:");
    scanf("%d", &numDiscos);
} while (numDiscos > 5 || numDiscos <= 0);

AbreJanela(400, 400, "Torre de Hanoi");
PintarFundo(255, 255, 255);
MostraPlanoCartesiano(10);
(...)
```

Ao final da execução, queremos que a janela continua aberta e a cena se mantenha estática, então utilizamos a função *Desenha* para isso.

Listagem 3.5: Mantendo cena estática.

```
(...)
hanoi(numDiscos, 'A', 'B', 'C');

printf("\nPronto! E assim que se resolve esta torre!");

Desenha();

return 0;
}
(...)
```

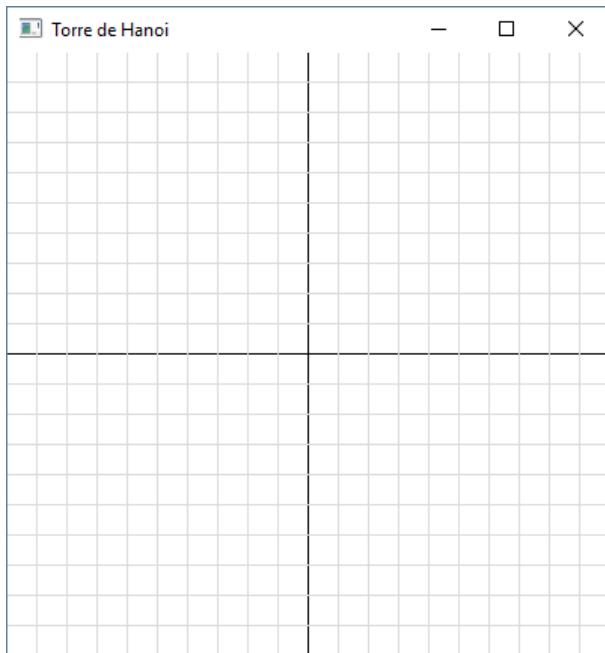


Figura 3.7: Janela de contexto.

```

1 #include <stdio.h>
2 #include <playAPC/playapc.h>
3
4 void hanoi(int n, char a, char b, char c){
5     /* mova n discos do pino a para o pino b usando
6         o pino c como intermediario */
```

- 7
- 8 if (n == 1) {
- 9 printf("\nmova disco %d de %c para %c\n", n, a, b);
- 10 }
- 11 else
- 12 {
- 13 hanoi(n - 1, a, c, b); // H1
- 14 printf("\nmova disco %d de %c para %c\n", n, a, b);
- 15
- 16 hanoi(n - 1, c, b, a); // H2
- 17 }
- 18 }
- 19
- 20 int main(void) {
- 21 int numDiscos;
- 22
- 23 do{
- 24 printf("\nDigite uma quantidade de discos menor ou igual a 5: ");
- 25 scanf("%d", &numDiscos);
- 26 }while(numDiscos > 5 || numDiscos <= 0);
- 27
- 28 AbreJanela(400, 400, "Torre de Hanoi");
- 29 PintarFundo(255, 255, 255);
- 30 MostraPlanoCartesiano(10);
- 31 }

```

32     hanoi(numDiscos, 'A', 'B', 'C');
33
34     printf("\nPronto! E assim que se resolve esta torre!");
35
36     Desenha();
37
38     return 0;
39 }

```

### 3.2.2 Criação dos discos de Hanoi

Vamos agora desenhar o fundo estático da torre de Hanói. Por praticidade, vamos criar uma função *geraFundo*, onde ela irá criar uma espécie de mesa e as torres A, B e C.

Como o programa terá animação complexa, onde cada disco se move independente e há um fundo estático que não se move, se faz necessário a utilização de grupos. Por isso, vamos declarar um variável para ser o índice do grupo *fundo*.

Listagem 3.6: Variável para o grupo *fundo*.

```

(...)
int main(void) {

    int numDiscos;
    int grupofundo;
}

(...)

```

Agora podemos simplificar nosso desenho reduzindo tudo a retângulos. Um retângulo precisa de uma base, altura e um ponto de referência, que é o inferior esquerdo. Como queremos criar uma mesa que ocupe toda parte inferior da janela, e sabendo que os limites de renderização é  $-100$  à  $100$ , o canto inferior esquerdo da mesa deve estar posicionado na posição  $(-100, -100)$ , com a mesa tendo 200 unidades de largura.

Listagem 3.7: Mesa.

```

(...
int geraFundo () {
    Ponto p;
    int fundo;

    fundo = Criagrupo();
}

```

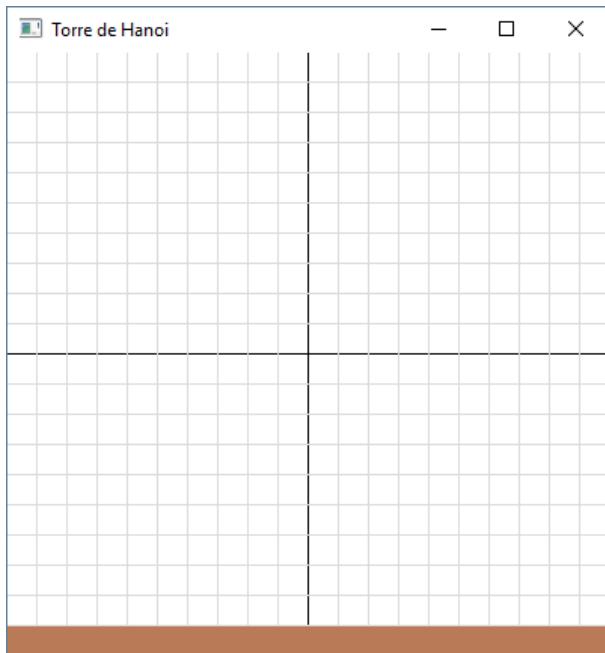


Figura 3.8: Mesa.

```
p.x = -100;  
p.y = -100;  
CriaRetangulo(200, 10, p);  
Pintar(185, 122, 87);  
(...)
```

Agora podemos criar nossas torres. Também simplificando as torres como retângulos, impraticamente vamos distribuí-las ao longo do plano. Porém, queremos que as torres fiquem acima da mesa. Como a mesa tem altura 10, então o canto esquerdo inferior em  $y$  das torres deve ser  $-90$ . Vamos definir que a torre A esteja em azul, a torre B em vermelho e a torre C em amarelo.

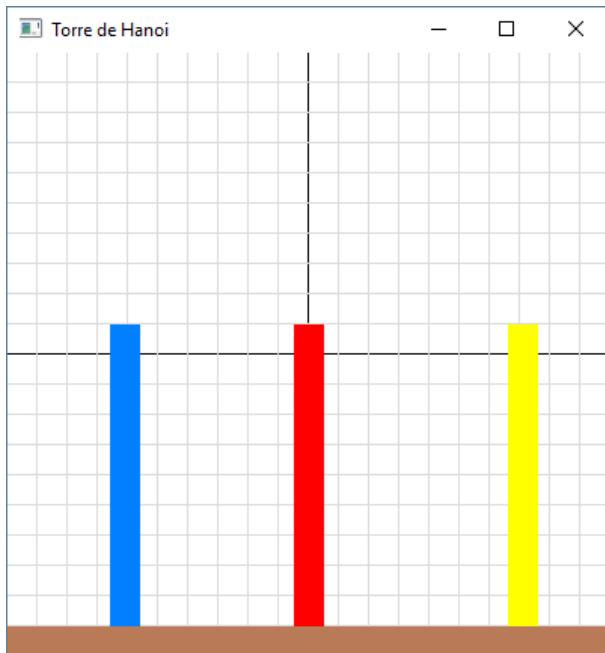


Figura 3.9: Torres.

Listagem 3.8: Torres.

```
(...)
p.y = -90;
//Torre A
p.x = -66;
CriaRetangulo(10, 100, p);
Pintar(0, 128, 255);

//Torre B
p.x = -5;
CriaRetangulo(10, 100, p);
Pintar(255, 0, 0);

//Torre C
p.x = 66;
CriaRetangulo(10, 100, p);
Pintar(255, 255, 0);
(...)
```

Agora que o fundo está criado, podemos retornar o valor do índice deste grupo para a variável *grupo fundo*.

Listagem 3.9: Função *geraFundo*.

```
(...)
int geraFundo() {
    Ponto p;
    int fundo;

    fundo = CriaGrupo();

    p.x = -100;
    p.y = -100;
    CriaRetangulo(200, 10, p);
    Pintar(185, 122, 87);

    p.y = -90;
    //Torre A
    p.x = -66;
    CriaRetangulo(10, 100, p);
    Pintar(0, 128, 255);

    //Torre B
    p.x = -5;
    CriaRetangulo(10, 100, p);
    Pintar(255, 0, 0);

    //Torre C
    p.x = 66;
    CriaRetangulo(10, 100, p);
    Pintar(255, 255, 0);

    return fundo;
}
(...)

AbreJanela(400, 400, "Torre de Hanoi");
PintarFundo(255, 255, 255);
MostraPlanoCartesiano(10);

grupofundo = geraFundo();
```

```
hanoi(numDiscos, 'A', 'B', 'C');  
  
(...)
```

Como vamos mover independentemente cada disco de torre em torre, precisamos criar um grupo para cada disco, saber onde cada disco está e saber qual sua largura. Desta forma, vamos criar uma estrutura com estas informações e criar um vetor de 5 posições desta estrutura.

Listagem 3.10: Torres.

```
(...)  
typedef struct{  
    int grupo, largura;  
    char torre;  
}tipoDisco;  
(...)  
  
int main(void) {  
    int numDiscos;  
    int grupofundo;  
    tipoDisco discos[5];  
(...)
```

Agora, por organização, vamos criar uma função que irá criar os discos, passando o vetor *grupodiscos* e a quantidade de discos como argumento. A cor de cada disco da torre de Hanói será definido aleatoriamente. Todos os discos definidos nesta função começarão na torre A.

Antes de começarmos, para não lidarmos com lixo de memória, vamos zerar os valores dos discos.

O último disco, o maior de todos, ele estará na altura da mesa, então  $p.y \leftarrow -90$ . Impiricamente, como a maior quantidade de discos é 5, vamos definir que ele esteja posicionado em  $-100$  em  $x$  e que sua base seja igual a  $80$ . Vamos definir também que o maior disco seja a última posição do vetor, uma vez que ele será um dos últimos a ser movimentado.

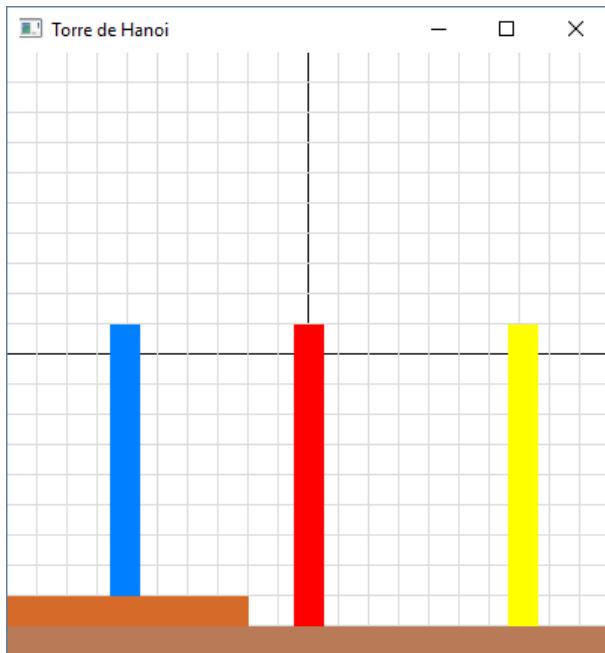


Figura 3.10: Último disco.

Listagem 3.11: Último disco.

```
(...)
void geraDiscos(tipoDisco discos[], int numDiscos) {
    Ponto p;
    int base, altura;

    for(int i = 0; i < 5; i++) {
        discos[i].torre = '0';
        discos[i].grupo = -1;
        discos[i].largura = -1;
    }

    p.y = -90;
    p.x = -100;

    base = 80;
    altura = 10;
    for(int i = numDiscos - 1; i >= 0; i--) {
        discos[i].torre = 'A';
        discos[i].grupo = CriaGrupo();
        discos[i].largura = base;
```

```

CriaRetangulo(base, altura, p);
Pintar(rand()%255, rand()%255, rand()%255);

( . . . )

```

Agora precisamos definir a posição dos outros discos. Em  $y$  sempre será incrementado de 10, porque queremos que as alturas sejam constantes. O próximo disco sempre será menor que o anterior, então a largura também irá diminuir. Por isso, em  $x$ , precisamos manter o cuidado de deixar os discos sempre centralizados.

Como serão no máximo 5 discos, com o primeiro tendo largura igual a 80, vamos definir que o próximo disco seja 16 unidades menor ( $\frac{80}{5} = 16$ ), garantindo que todos os cinco discos aparecerão na tela. Empiricamente, vamos centralizá-los com uma distância em  $x$  em 8 unidades de diferença.

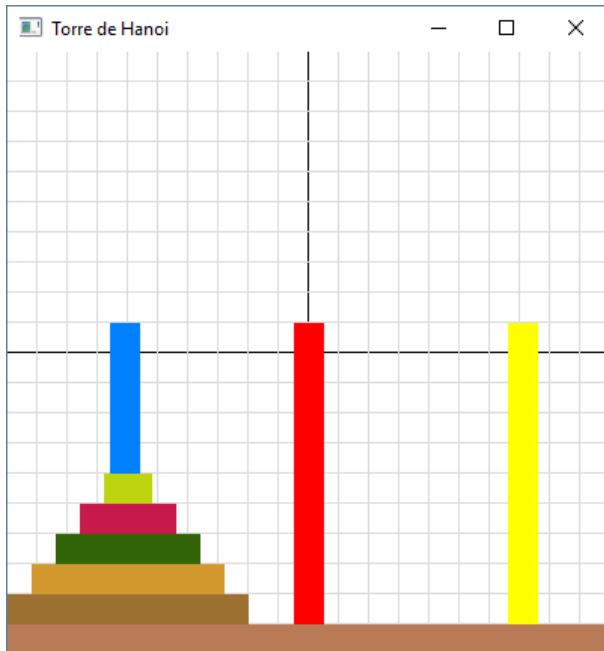


Figura 3.11: Todos os discos.

```

1 #include <stdio.h>
2 #include <playAPC/playapc.h>
3 #include <math.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 typedef struct{
8     int grupo, largura;
9     char torre;
10 }tipoDisco;

```

```

11
12 void geraDiscos(tipoDisco discos[], int numDiscos) {
13     Ponto p;
14     int base, altura;
15
16     for(int i = 0; i < 5; i++) {
17         discos[i].torre = '0';
18         discos[i].grupo = -1;
19         discos[i].largura = -1;
20     }
21
22     p.y = -90;
23     p.x = -100;
24
25     base = 80;
26     altura = 10;
27     for(int i = numDiscos - 1; i >= 0; i--) {
28         discos[i].torre = 'A';
29         discos[i].grupo = CriaGrupo();
30         discos[i].largura = base;
31
32         CriaRetangulo(base, altura, p);
33         Pintar(rand()%255, rand()%255, rand()%255);
34
35         base -= 16;
36         p.x += 8;
37         p.y += 10;
38
39     }
40 }
41
42 int geraFundo() {
43     Ponto p;
44     int fundo;
45
46     fundo = CriaGrupo();
47
48     p.x = -100;
49     p.y = -100;
50     CriaRetangulo(200, 10, p);
51     Pintar(185, 122, 87);
52
53     p.y = -90;
54     //Torre A
55     p.x = -66;
56     CriaRetangulo(10, 100, p);
57     Pintar(0, 128, 255);
58
59     //Torre B
60     p.x = -5;
61     CriaRetangulo(10, 100, p);
62     Pintar(255, 0, 0);
63
64     //Torre C
65     p.x = 66;
66     CriaRetangulo(10, 100, p);
67     Pintar(255, 255, 0);

```

```

68
69     return fundo;
70 }
71
72 void hanoi(int n, char a, char b, char c){
73     /* mova n discos do pino a para o pino b usando
74     o pino c como intermediario */
75
76     if (n == 1){
77         printf("\nmova disco %d de %c para %c\n", n, a, b);
78     }
79     else
80     {
81         hanoi(n - 1, a, c, b);                                // H1
82         printf("\nmova disco %d de %c para %c\n", n, a, b);
83
84         hanoi(n - 1, c, b, a);                                // H2
85     }
86 }
87
88 int main(void) {
89     int numDiscos;
90     int grupofundo;
91     tipoDisco discos[5];
92
93     srand(time(NULL));
94
95     do{
96         printf("\nDigite uma quantidade de discos menor ou igual a 5: ");
97         scanf("%d", &numDiscos);
98     }while(numDiscos > 5 || numDiscos <= 0);
99
100    AbreJanela(400, 400, "Torre de Hanoi");
101    PintarFundo(255, 255, 255);
102    MostraPlanoCartesiano(10);
103
104    grupofundo = geraFundo();
105    geraDiscos(discos, numDiscos);
106
107    hanoi(numDiscos, 'A', 'B', 'C');
108
109    printf("\nPronto! E assim que se resolve esta torre!");
110
111    Desenha();
112
113    return 0;
114 }
```

### 3.2.3 Movimentação dos discos

Para movimentar os discos, nós precisaremos passar para a função *hanoi* o nosso vetor de discos e, para cada mensagem de movimentação do disco, precisaremos de fato mover o disco. Vamos criar então uma função *moveDisco* e chamá-la dentro da função *hanoi*. Esta função precisará

ter a referência dos discos, ou seja, o vetor de discos, também precisará saber de qual torre estaremos mexendo o disco e para qual torre iremos mover o disco.

Listagem 3.12: Como a função *moveDisco* será chamada.

```
(...)
void hanoi(int n, char a, char b, char c, tipoDisco discos[5]) {
    /* mova n discos do pino a para o pino b usando
     o pino c como intermediario */ 

    if (n == 1) {
        printf("\nmova disco %d de %c para %c\n", n, a, b);
        moveDisco(discos, a, b);
    }
    else
    {
        hanoi(n - 1, a, c, b, discos); // H1
        printf("\nmova disco %d de %c para %c\n", n, a, b);
        moveDisco(discos, a, b);

        hanoi(n - 1, c, b, a, discos); // H2
    }
}
(...)
```

Agora, dado o nosso vetor, precisamos achar o último disco da torre A e movê-la para torre B. Porém, se já tiver um disco na torre B, a altura que colocaremos esse novo disco não será a da mesa, mas sim da altura da mesa vezes a altura da quantidade de discos que existem naquela torre.

Listagem 3.13: Determinando a posição em *y* do disco.

```
(...)
void moveDisco(tipoDisco discos[5], char a, char b) {
    int grupodisco, qtddiscos = 0;
    int i = 0, index;
    Ponto p;

    while(discos[i].torre != a) {
        i++;
        ...
```

```

}

index = i; //Se saiu do loop, entao encontramos o disco que
precisaremos mover

for(i = 0; i < 5; i++) {
    if(discos[i].torre == b)
        qtddiscos++;
}

p.y = -90 + qtddiscos * 10; //altura da mesa + quantidade de
discos

(...)
```

Agora precisamos definir a coordenada  $x$  do ponto esquerdo inferior do disco para onde queremos mover. Nós sabemos que o canto esquerdo inferior da torre A está em  $-66$ , da torre B em  $-5$  e da torre C em  $66$ . Então, dada o centro da torre em  $x$  (uma vez que as torres tem largura 10, seu centro será sua posição do canto inferior mais 5), subtraímos com a largura do disco dividido por 2. Após estes cálculo, poderemos mover o disco e definir que agora este disco está na torre B.

Listagem 3.14: Determinando a posição em  $x$  do disco.

```

(...)

switch(b) {
    case 'A':
        p.x = -66 + 5; //mais 5 do centro da torre
        break;
    case 'B':
        p.x = -5 + 5; //mais 5 do centro da torre
        break;
    case 'C':
        p.x = 66 + 5; //mais 5 do centro da torre
        break;
}

p.x -= discos[index].largura/2;

discos[index].torre = b;

Move(p, discos[index].grupo);

(...)
```

Como a execução do programa acontece muito rápido, gostaríamos que o próprio usuário, ao digitar a tecla *enter*, pulasse de um passo para o outro. Então, enquanto o usuário não apertar *enter*, o programa deve continuar naquele passo da solução da torre de Hanói.

Listagem 3.15: Laço para pegar entrada do usuário.

```
(...)
while (!ApertouTecla (GLFW_KEY_ENTER) )
    DesenhalFrame ();
```

(...)

Como o programa já começa chamando a função *hanoi*, que já exibe o primeiro passo da solução, colocaremos este mesmo laço também na função *main* para o usuário determinar quando o programa deve começar a solucionar a torre.

Listagem 3.16: Laço para esperar o usuário querer começar a solução.

```
(...)
AbreJanela (400, 400, "Torre de Hanoi");
PintarFundo (255, 255, 255);
MostraPlanoCartesiano (10);

grupofundo = geraFundo ();
geraDiscos (discos, numDiscos);

while (!ApertouTecla (GLFW_KEY_ENTER) )
    DesenhalFrame ();
```

hanoi (numDiscos, 'A', 'B', 'C', discos);

(...)

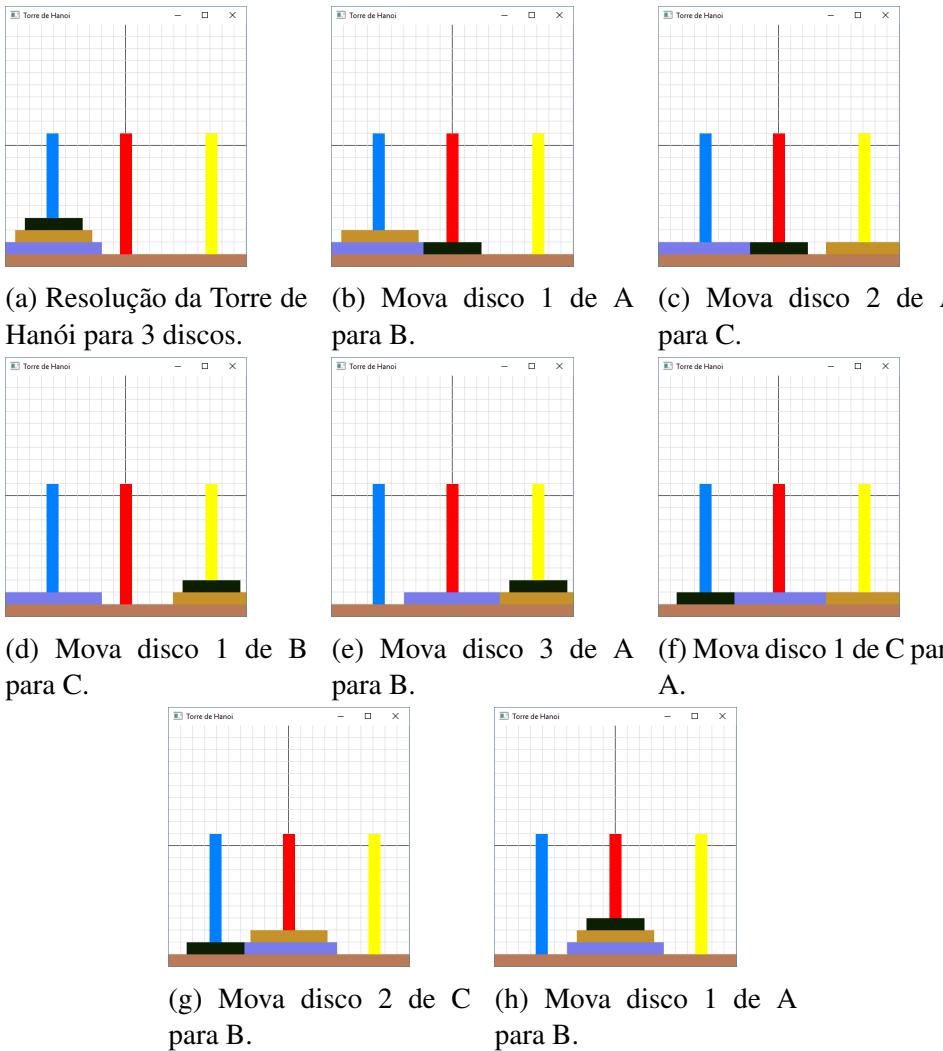


Figura 3.12: Solução da torre de Hanói para 3 discos: a) Início; b) Passo 1; c) Passo 2; d) Passo 3; e) Passo 4; f) Passo 5; g) Passo 6; h) Torre solucionada.

```

1 #include <stdio.h>
2 #include <playAPC/playapc.h>
3 #include <math.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 typedef struct{
8     int grupo, largura;
9     char torre;
10 }tipoDisco;
11
12 void geraDiscos(tipoDisco discos[], int numDiscos){
13     Ponto p;
14     int base, altura;
15
16     for(int i = 0; i < 5; i++){
17         discos[i].torre = '0';
18         discos[i].grupo = -1;
19         discos[i].largura = -1;

```

```

20     }
21
22     p.y = -90;
23     p.x = -100;
24
25     base = 80;
26     altura = 10;
27     for(int i = numDiscos - 1; i >= 0; i--) {
28         discos[i].torre = 'A';
29         discos[i].grupo = CriaGrupo();
30         discos[i].largura = base;
31
32         CriaRetangulo(base, altura, p);
33         Pintar(rand()%255, rand()%255, rand()%255);
34
35         base -= 16;
36         p.x += 8;
37         p.y += 10;
38
39     }
40 }
41
42 int geraFundo() {
43     Ponto p;
44     int fundo;
45
46     fundo = CriaGrupo();
47
48     p.x = -100;
49     p.y = -100;
50     CriaRetangulo(200, 10, p);
51     Pintar(185, 122, 87);
52
53     p.y = -90;
54     //Torre A
55     p.x = -66;
56     CriaRetangulo(10, 100, p);
57     Pintar(0, 128, 255);
58
59     //Torre B
60     p.x = -5;
61     CriaRetangulo(10, 100, p);
62     Pintar(255, 0, 0);
63
64     //Torre C
65     p.x = 66;
66     CriaRetangulo(10, 100, p);
67     Pintar(255, 255, 0);
68
69     return fundo;
70 }
71
72 void moveDisco(tipoDisco discos[5], char a, char b) {
73     int grupodisco, qtddiscos = 0;
74     int i = 0, index;
75     Ponto p;
76

```

```

77     while(discos[i].torre != a) {
78         i++;
79     }
80     index = i; //Se saiu do loop, entao encontramos o disco que precisaremos mover
81
82     for(i = 0; i < 5; i++) {
83         if(discos[i].torre == b)
84             qtddiscos++;
85     }
86
87     p.y = -90 + qtddiscos * 10; //altura da mesa + quantidade de discos
88
89     switch(b) {
90         case 'A':
91             p.x = -66 + 5; //mais 5 do centro da pilastra
92             break;
93         case 'B':
94             p.x = -5 + 5; //mais 5 do centro da pilastra
95             break;
96         case 'C':
97             p.x = 66 + 5; //mais 5 do centro da pilastra
98             break;
99     }
100
101    p.x -= discos[index].largura/2;
102
103    discos[index].torre = b;
104
105    Move(p, discos[index].grupo);
106
107    while(!ApertouTecla(GLFW_KEY_ENTER))
108        DesenhaFrame();
109
110 }
111
112 void hanoi(int n, char a, char b, char c, tipoDisco discos[5]){
113     /* mova n discos do pino a para o pino b usando
114     o pino c como intermediario */ 
115
116     if (n == 1){
117         printf("\nmova disco %d de %c para %c\n", n, a, b);
118         moveDisco(discos, a, b);
119     }
120     else
121     {
122         hanoi(n - 1, a, c, b, discos); // H1
123         printf("\nmova disco %d de %c para %c\n", n, a, b);
124         moveDisco(discos, a, b);
125
126         hanoi(n - 1, c, b, a, discos); // H2
127     }
128 }
129
130 int main(void){
131     int numDiscos;
132     int grupofundo;
133     tipoDisco discos[5];

```

```

134
135     srand(time(NULL));
136
137     do{
138         printf("\nDigite uma quantidade de discos menor ou igual a 5: ");
139         scanf("%d", &numDiscos);
140     }while(numDiscos > 5 || numDiscos <= 0);
141
142     AbreJanela(400, 400, "Torre de Hanoi");
143     PintarFundo(255, 255, 255);
144     MostraPlanoCartesiano(10);
145
146     grupoFundoo = geraFundo();
147     geraDiscos(discos, numDiscos);
148
149     while (!ApertouTecla(GLFW_KEY_ENTER))
150         DesenhaFrame();
151
152     hanoi(numDiscos, 'A', 'B', 'C', discos);
153
154     printf("\nPronto! E assim que se resolve esta torre!");
155
156     Desenho();
157
158     return 0;
159 }
```

# Capítulo 4

## Resultados Experimentais

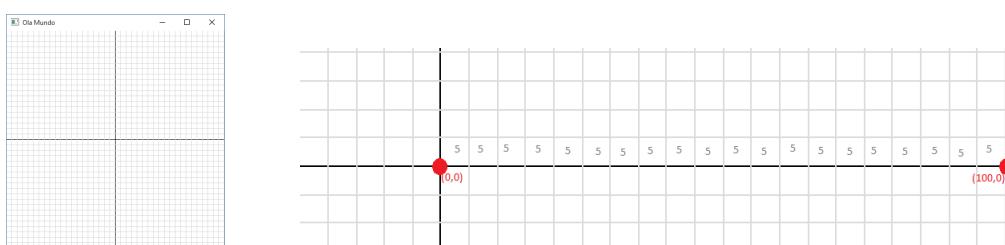
### 4.1 Práticas sugeridas pela PLAYAPC

Baseado nos tópicos mais recorrentes da matéria inicial dos cursos de computação, explicado na Seção 3.1, foram feitos 24 exercícios no total, separados por tópicos. Todos os exercícios, juntamente com sua solução e comentários, podem ser consultados na apostila de exercício (Apêndice A). Nesta aposila, ainda são encontrados mais 3 exercícios extras, que abordam todos os tópicos juntos.

Para melhor visualização, os códigos-fontes de todos os exercícios propostos nesta Seção também podem ser encontrados no Apêndice E.

#### 4.1.1 Tipo de dados e estruturas de controle

Exercício 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 4.1: Plano cartesiano de -100 à 100: a) Todo plano; b) Parte do plano.

Comentário: 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 instalada.

Exercício 1.2) Exiba um boneco palito.



Figura 4.2: Boneco Palito.

Comentário: 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<sup>1</sup>.

Exercício 1.3) Exiba a estrela de Davi.

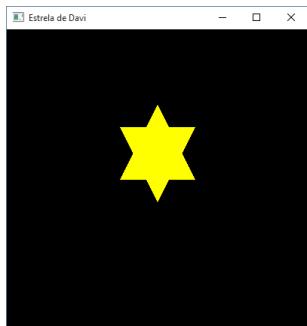


Figura 4.3: Estrela de Davi.

Comentário: 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*.

---

<sup>1</sup><http://playapc.zaghetto.com/category/funcoes/geometrias>

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

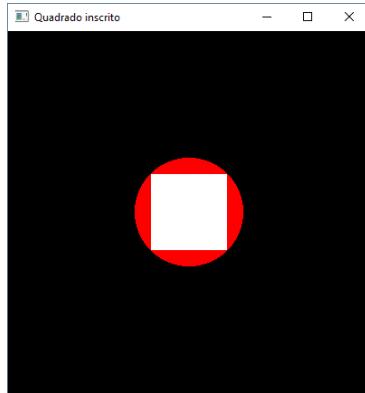
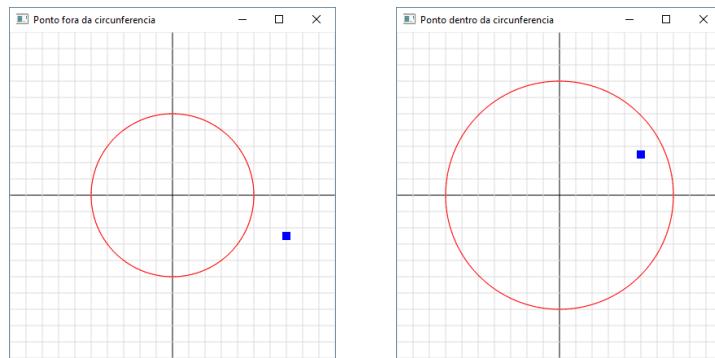


Figura 4.4: Quadrado inscrito em um círculo.

Comentário: 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.

Exercício 1.5) 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.



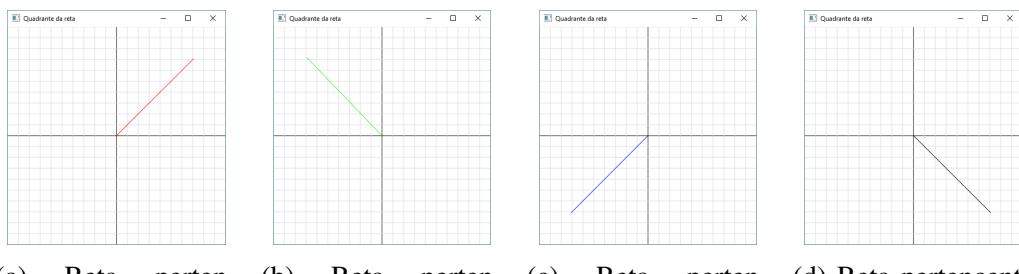
(a) Ponto está fora da circunferência. (b) Ponto está dentro da circunferência.

Figura 4.5: Verificação se ponto está dentro ou fora da circunferência: a) Fora; b) Dentro.

Comentário: 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*.

Exercício 1.6) 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

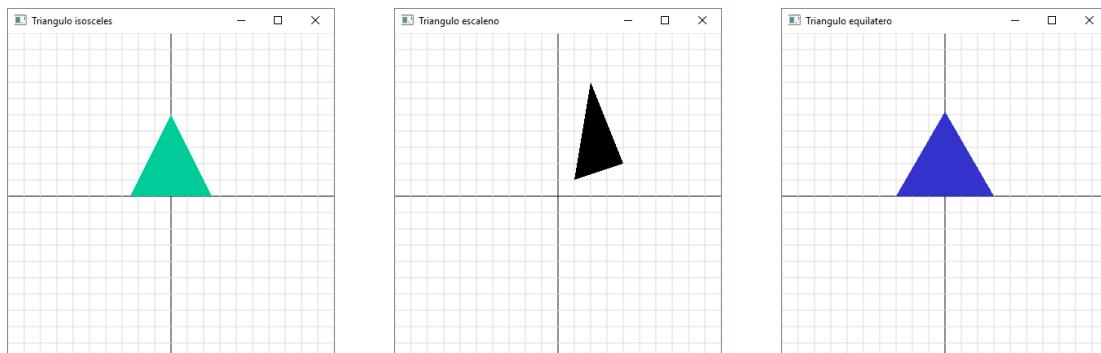


(a) Reta pertencente ao primeiro quadrante. (b) Reta pertencente ao segundo quadrante. (c) Reta pertencente ao terceiro quadrante. (d) Reta pertencente ao quarto quadrante.

Figura 4.6: Identificação de quadrante: a) Primeiro quadrante; b) Segundo quadrante; c) Terceiro quadrante; d) Quarto quadrante.

Comentário: 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.

Exercício 1.7) 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.

Figura 4.7: Existência de triângulos: a) Isósceles; b) Escaleno; c) Equilátero.

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

Exercício 1.8) Exiba um carrinho se movendo de  $-100$  à  $100$ .

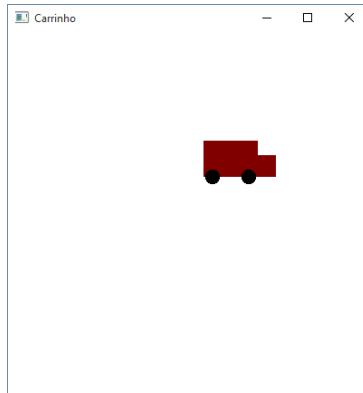


Figura 4.8: Carro se movendo da posição  $-100$  até a posição  $100$ .

Comentário: 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*.

Exercício 1.9) Construa um moinho de vento e coloque apenas as hélices para girar.

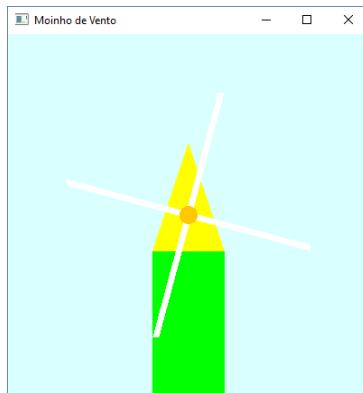


Figura 4.9: Moinho de vento.

Comentário: 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.

Exercício 1.10) Escreva um programa utilizando a playAPC que simule 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.

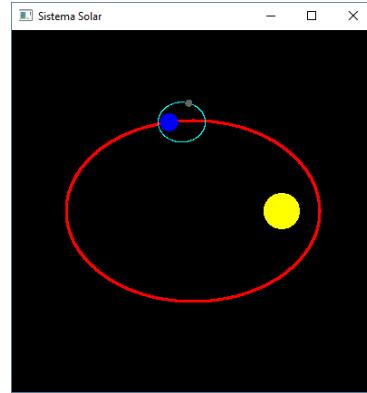


Figura 4.10: Sistema solar.

Comentário: Esta prática ilustra, de maneira exagerada, o movimento elíptico de um sistema solar, reforçando que a lua deve girar em torno da terra de maneira mais rápida que a terra deve girar ao redor da lua.

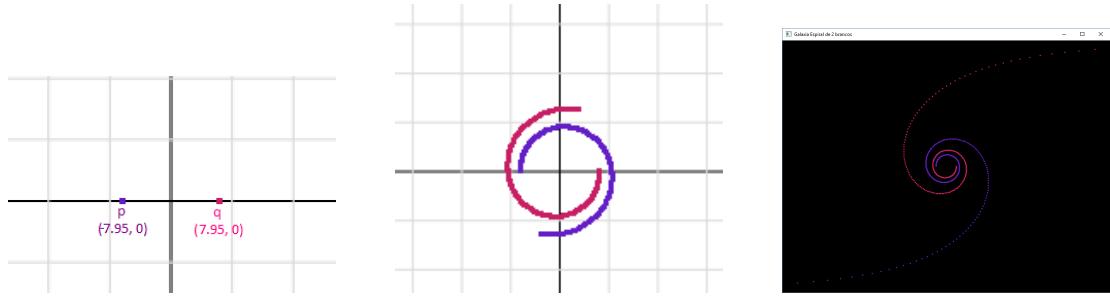
Exercício 1.11) Sabe-se que a equação da espiral hiperbólica pode ser definida por

$$\begin{aligned} x &= a \cos(\theta) \\ y &= a \sin(\theta) \end{aligned} \quad (4.1)$$

onde  $a$  é a assíntota para  $y$  e  $\theta$  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 4.1. Para ponto  $p$  em  $(x, y)$  de uma das hiperbólicas, o ponto  $p$  da outra espiral deve estar posicionado em  $(-x, -y)$ .

Comentário: Esta prática ilustra como a função *Desenha1Frame* pode ser utilizada.

Exercício 1.12) Crie uma animação em que 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.



(a) Posição inicial aproximada de ambas as espirais.  
 (b) 100º iteração no processo de criação das espirais.  
 (c) Espirais completas.

Figura 4.11: Espiral hiperbólica: a) Primeira iteração; b) Centésima iteração; c) Espirais.

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

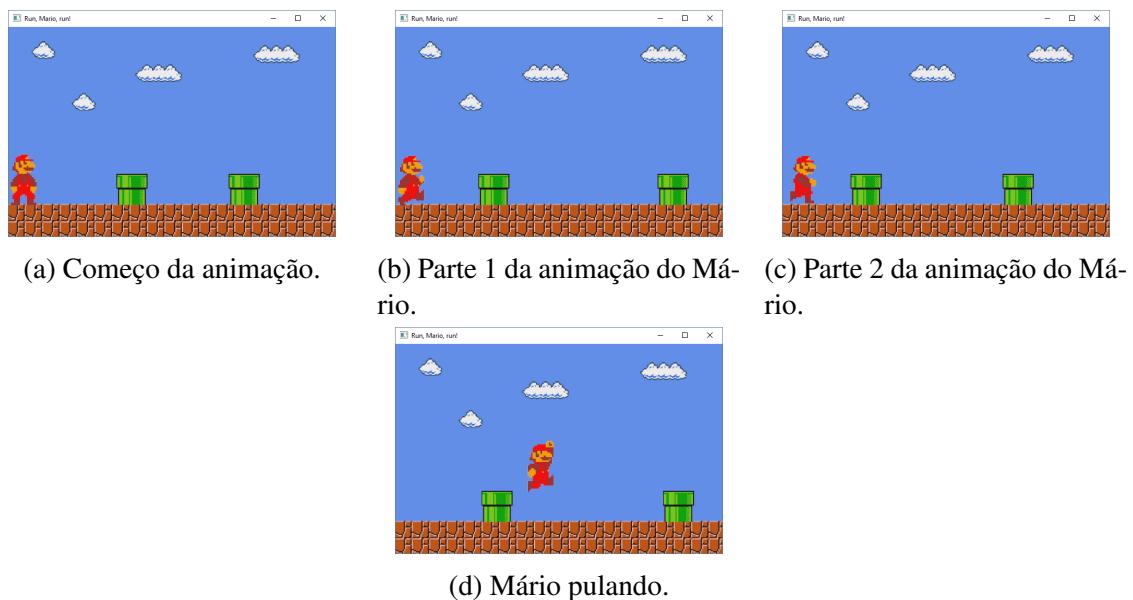


Figura 4.12: Animação do Mário: a) Início; b) e c) Mário andando; d) Mário pulando.

Comentário: 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.

### 4.1.2 Vetores e Matrizes

Exercício 2.1) Exiba o gráfico do polinômio  $-x^3$  para  $-50 \leq x \leq 50$ .

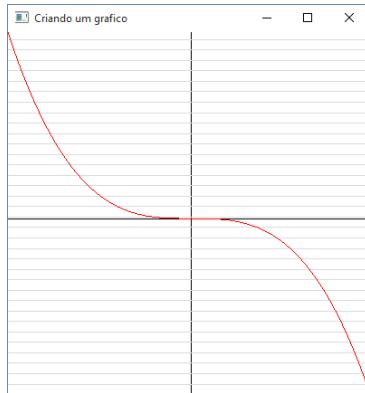


Figura 4.13: Gráfico do polinômio  $-x^3$ .

Comentário: 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 PLAYAPC 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)`.

Exercício 2.2) 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 4.2 e o resultado é armazenado em três novos vetores Rf, Gf, Bf. O tamanho da janela de filtragem é fixo e igual a 3.

$$\begin{aligned} Rf[i] &= \frac{R[i-1]+R[i]+R[i+1]}{3} \\ Gf[i] &= \frac{G[i-1]+G[i]+G[i+1]}{3} \\ Bf[i] &= \frac{B[i-1]+B[i]+B[i+1]}{3} \end{aligned} \quad (4.2)$$

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 4.3.

$$\frac{1}{n} \sum_{i=0}^n \sqrt{(R[i] - Rf[i])^2 + (G[i] - Gf[i])^2 + (B[i] - Bf[i])^2} \quad (4.3)$$

Equação 2: Cálculo da distância euclidiana média

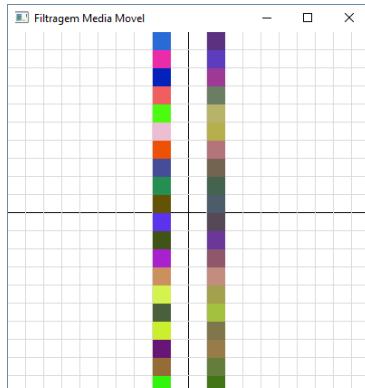


Figura 4.14: À esquerda, 20 quadrados com componentes RGB fornecidos pelo usuário. À direita, aplicação do filtro de média móvel central em cada quadrado.

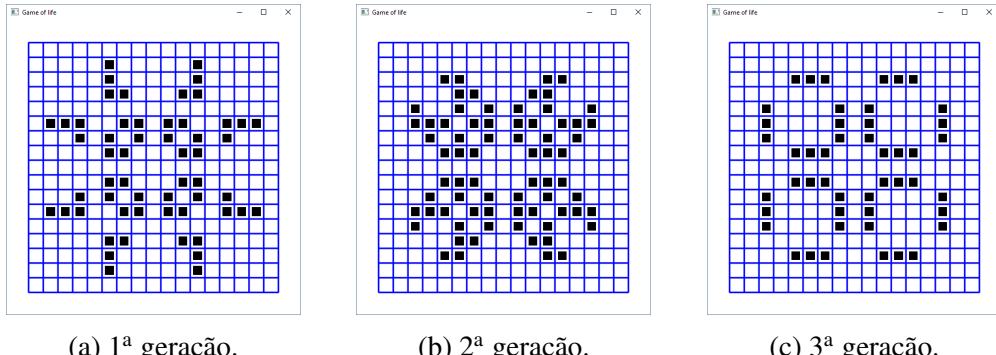
Comentário: Esta prática exercita conceitos de processamento de imagens, aplicando um filtro dado os componentes RGB da figura, além de exercitar conceitos teóricos de matemática, como o funcionando do operador  $\Sigma$ .

Exercício 2.3) O jogo da vida é um autômato celular que simula a alteração e mudanças de grupos de seres vivos. Foi criado pelo matemático John Horton Conway na década de 70. Extremamente útil para diversas áreas da ciência, este autômato possui quatro regras simples:

1. Qualquer célula viva com menos de dois vizinhos vivos morre de solidão
2. Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação
3. Qualquer célula morta com exatamente três vizinhos vivos se torna uma célula viva
4. Qualquer célula viva com dois ou três vizinhos vivos continua no mesmo estado para a próxima geração

Mostre graficamente o jogo da vida para uma matriz com 17 linhas e 17 colunas 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) 1<sup>a</sup> geração.

(b) 2<sup>a</sup> geração.

(c) 3<sup>a</sup> geração.

Figura 4.15: Jogo da vida: a) Primeira geração; b) Segunda geração; c) Terceira geração.

Comentário: 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.

Exercício 2.4) 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.

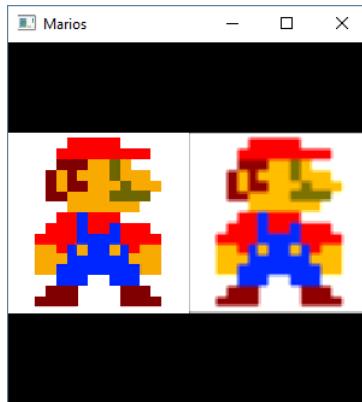


Figura 4.16: Filtro Motion Blur.

Comentário: Esta prática 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.

Exercício 2.5) O jogo batalha naval é um jogo de tabuleiro no qual os jogadores devem adivinhar em quais posições do tabuleiro estão localizados os navios do adversário. Ganhá aquele que

derrubar todos os navios. Implemente o jogo batalha naval utilizando a PLAYAPC, utilizando as seguintes premissas:

- O tabuleiro será 9x9;
- Posicione arbitrariamente 10 navios sobre este tabuleiro, sendo 3 navios com tamanho igual a 3, 3 navios com tamanho igual a 2 e 4 navios com tamanho igual a 1;
- Não pode posicionar um navio onde já existir outro navio e não pode posicionar o navio imediatamente do lado de outro navio: deve existir pelo menos um espaço vazio entre os dois navios;
- Os navios podem estar nas verticais e nas horizontais, não nas diagonais;
- Se um navio de tamanho igual a 1 for atingido, fica indicado no lugar que o navio foi derrubado;
- Se um navio de tamanho igual a 2 ou 3 for atingido, deve ficar indicado que o navio foi atingido, porém não foi derrubado. Somente quando todas as peças deste navio forem atingidas é que deve-se indicar o navio foi derrubado;
- Se o usuário escolheu uma posição do tabuleiro onde não há navio, deve ficar indicado que o usuário não atingiu água e;
- O jogo deve avisar ao usuário quando ele conseguir derrubar todos os 10 navios.

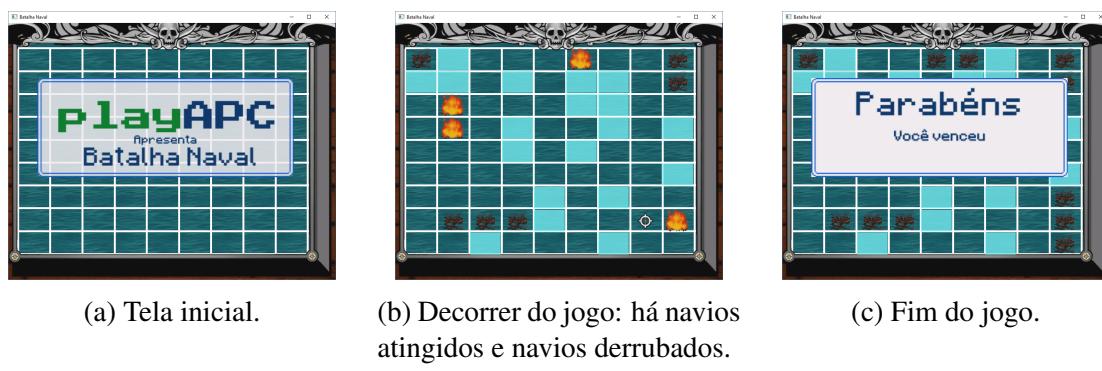


Figura 4.17: Batalha Naval: a) Início; b) Decorrer do jogo; c) Fim do jogo.

Comentário: Esta prática ilustra como as funções *PosicaoMouse*, *ApertouBotaoMouse* e *AssociaImagem* podem ser utilizadas. O aluno deve realizar cálculos para descobrir se o mouse está dentro de uma determinada região da tela e trocar imagens de acordo com a situação do jogo: se o usuário só está posicionando a mira, se o usuário atingiu um navio ou se derrubou um navio.

### 4.1.3 Subalgoritmos

Exercício 3.1) 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.

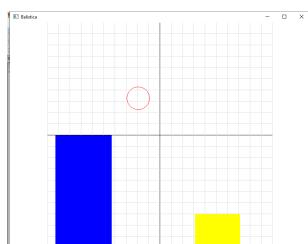


Figura 4.18: Lançador Balístico.

Comentário: Esta prática ilustra como a função *Limpadesenho* é usada para poder redesenhar outras cenas.

Exercício 3.2) Baseado no exercício anterior, 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 retilínea na direção contrária ao ataque.

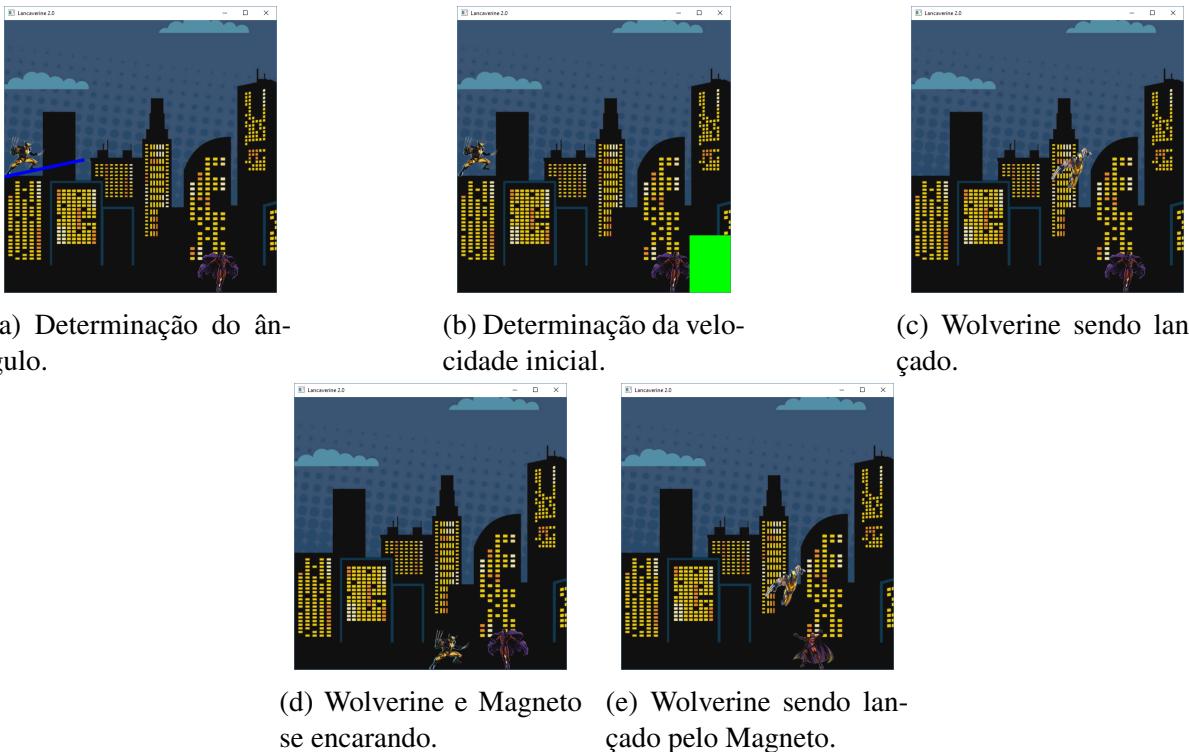


Figura 4.19: Lançaverine: a) Ângulo de lançamento; b) Velocidade inicial; c) Wolverine percorrendo trajetória; d) Caso quando Wolverine não acerta o Magneto; e) Caso quando Wolverine acerta o Magneto.

Comentário: Esta prática exercita o conceito de animação. 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 *lançar*, na Listagem E.19.

Exercício 3.3) Snake foi um jogo que começou no console Atari, em 1976, e foi popularizado nos anos 90 quando a Nokia começou a colocar o jogo em seus celulares.

O jogador controla uma cobra, que é representada por um quadrado no começo, e, a cada novo alimento que a cobra ingere, ela aumenta em um quadrado o seu tamanho, aumentando gradativamente a dificuldade do jogo. Se a cobra atingir a parede, ela morre. Se a cobra atingir o próprio corpo, ela também morre.

Dado estas regras, crie uma versão do jogo Snake utilizando a PLAYAPC.

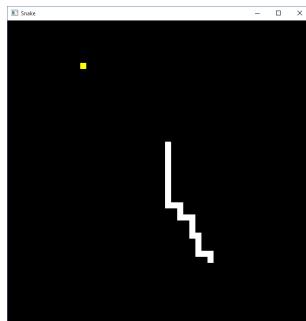


Figura 4.20: Jogo Snake.

Comentário: Esta prática ilustra como a função *ApertaTecla* e *MudaLimitesJanela* podem ser utilizadas: a primeira para lidar com input de teclado<sup>2</sup> e a segunda para ajustar o plano onde as geometrias serão desenhadas.

#### 4.1.4 Recursão

Exercício 4.1) 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  $\theta$  fornecida pelo usuário, quanto para o lado direito, com ângulo  $-\theta$ .

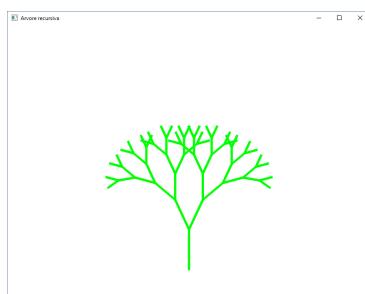


Figura 4.21: Árvore Binária com altura 30, ângulo  $25^\circ$  e profundidade 6.

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

Exercício 4.2) 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:

1. Só pode mover um disco de cada vez
2. Só pode mover o disco que estiver mais acima de uma torre e deve-se colocar no topo de outra torre

---

<sup>2</sup><http://playapc.zaghetto.com/funcoes/extras/input/tecla-pressionada>

3.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.

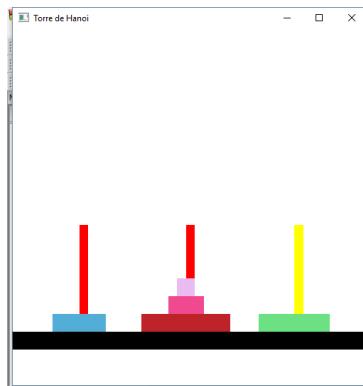


Figura 4.22: Solucionador da torre de Hanói.

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

Exercício 4.3) 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  $60^\circ$  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  $120^\circ$  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.

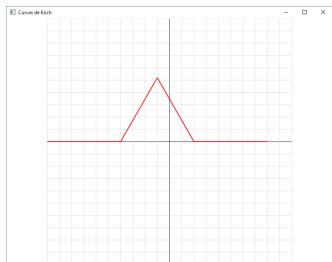
(a)Exiba a curva de Koch de ordem  $i$  com tamanho  $n$  igual à 180 unidades centrada na posição  $(-100, 0)$ .

(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)

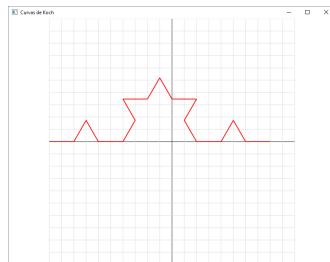
Como sugestão, a Listagem 4.1 exemplifica um cabeçalho da função *koch*, onde seu primeiro argumento é o ponto de origem, seu segundo argumento é o tamanho do segmento de reta, seu terceiro argumento é a inclinação do segmento e seu quarto argumento é a ordem da curva. A função retorna o segundo ponto da reta naquele nível de recursão que deverá ser criada.

Listagem 4.1: Header da função *koch*

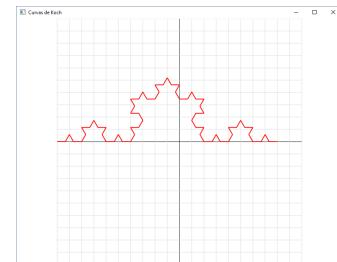
```
Ponto koch (Ponto from, double len, double theta, int order)
```



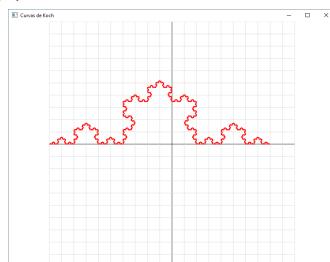
(a) Curva de Koch de ordem 1.



(b) Curva de Koch de ordem 2.



(c) Curva de Koch de ordem 3.



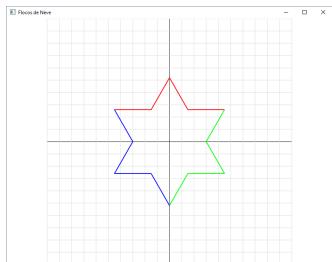
(d) Curva de Koch de ordem 6.

Figura 4.23: Curva de Koch: a) Ordem 1; b) Ordem 2; c) Ordem 3; d) Ordem 6.

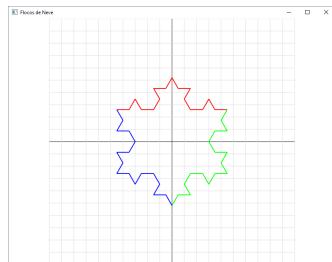
Comentário: Esta prática reforça o modo de utilização da função *CriaReta*, usando a função *movaCaneta*.

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

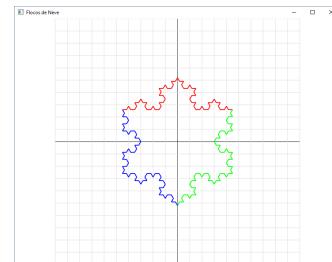
As Figuras 4.24 ilustram as três retas, onde a primeira curva, com inclinação  $0^\circ$ , está representada em vermelho, a segunda, com inclinação de  $120^\circ$  está representada em verde e a terceira curva, com inclinação  $-120^\circ$  está representada em azul.



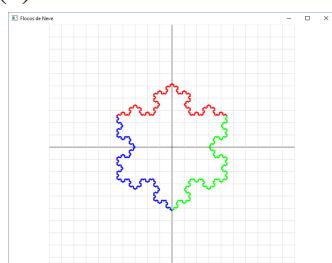
(a) Floco de neve de ordem 1.



(b) Floco de neve de ordem 2.



(c) Floco de neve de ordem 3.



(d) Floco de neve de ordem 6.

Figura 4.24: Floco de neve: a) Ordem 1; b) Ordem 2; c) Ordem 3; d) Ordem 6.

Comentário: Esta prática é uma evolução do exercício anterior.

Exercício 4.4) 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 4.25 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.

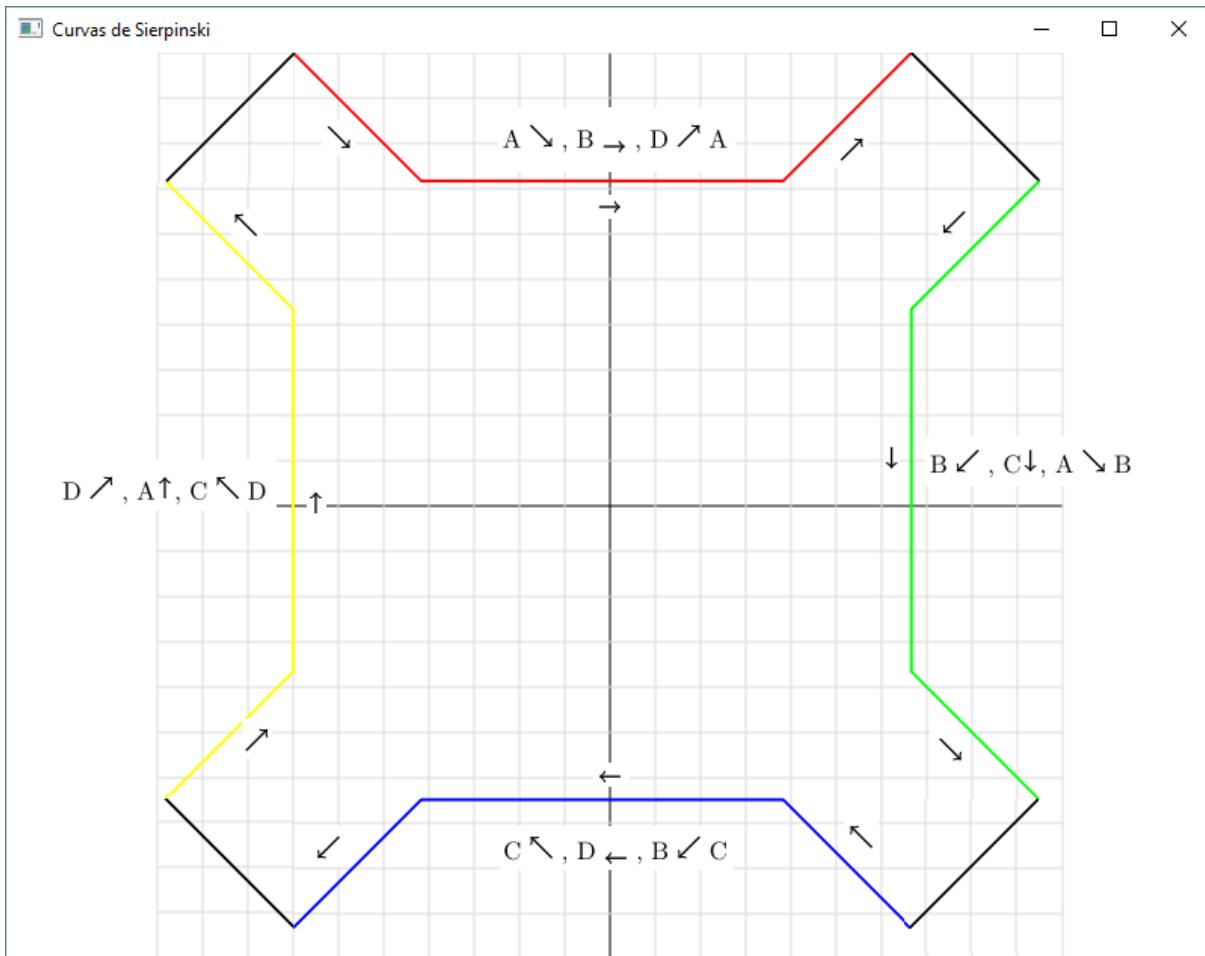


Figura 4.25: Curva de Sierpiński de ordem 1 com ângulo de  $45^\circ$ .

O programa inicia com uma curva básica  $S$  dada pelo padrão  $A \searrow, B \swarrow, C \nwarrow$  e  $D \nearrow$ . As setas indicam a virada do ângulo que as retas *fecham* os desenhos das funções.

- A curva A é dada pelo padrão  $A \searrow, B \rightarrow, D \nearrow A$
- A curva B é dada pelo padrão  $B \swarrow, C \downarrow, A \searrow B$
- A curva C é dada pelo padrão  $C \nwarrow, D \leftarrow, B \swarrow C$
- A curva D é dada pelo padrão  $D \nearrow, A \uparrow, C \nwarrow D$

Considere o ângulo de inclinação da curva de Sierpiński como  $45^\circ$ , ou seja, as retas desenhadas tem uma inclinação de  $45^\circ$  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 4.2 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 4.3.

Listagem 4.2: Função para conectar próxima curva

```
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 4.3: 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 Sierpinsk");
    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);

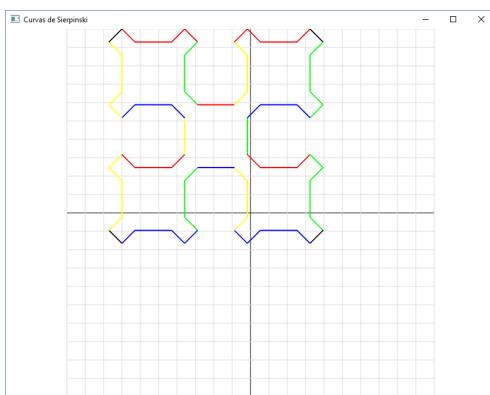
    p = D(k, h, p);
    p = reta(1, h, p);

Desenha();

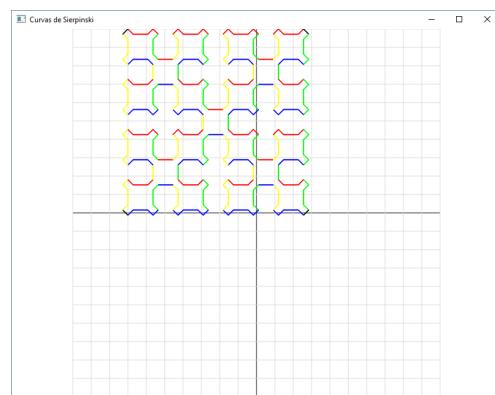
return 0;
}

```

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 4.26: Curva de Sierpiński: a) Ordem 2; b) Ordem 3.

Comentário: Esta prática reforça o modo de utilizar a função *CriaReta*. Além disso, exercita fundamentalmente recursões mútuas entre 4 funções.

## 4.2 Aceitação por parte dos professores em participar do experimento de aplicar a PLAYAPC

Assim que a PLAYAPC passou a ter versões estáveis para o uso, professores envolvidos com o projeto passaram a criar seus próprios materiais (Anexo I) e a desenvolver suas próprias práticas. Todos os exercícios de recursão foram desenvolvidos pelo Professor Ralha, exceto o Exercício 4.2. Além destes, outros diversos exercícios também foram propostos e testados pelo professor, como o da espiral hiperbólica, Exercício 1.11, o do moinho de vento, Exercício 1.9 e grande parte dos exercícios extras disponíveis na Apostila (Apêndice A). Nas versões mais recentes da

PLAYAPC, o professor Alexandre Zaghetto utilizou a biblioteca para programar exemplos do autômato Jogo da Vida<sup>3</sup><sup>4</sup> e também criou uma adaptação da obra *Circles in a Circles* do pintor Wassily Kandinsky. Sua adaptação é conhecida como *Sub-Kandisy Generator*<sup>5</sup>.

Utilizando a última versão da PLAYAPC, versão 1.7.2, um exercício criado pelo Professor Zaghetto foi o Certificado de Bravura (Anexo II). Neste exercício, os alunos tiveram até a Semana Universitária da UnB para implementar a própria versão do jogo *Freeway*, jogo criado em 1981 e publicado pela empresa Atari.

### 4.3 Aceitação por parte dos alunos

A PLAYAPC passou a ser aplicada a partir do semestre 2/2014, na turma de CB orientada pelo Professor Jóse Carlos Loureiro Ralha. Nos semestres 1/2015 e 2/2015, foram aplicadas nas turmas de CB do Professor Alexandre Zaghetto e da Professora Carla Castanho, ambos utilizando a biblioteca em somente uma prática.

Com a versão 1.7.2 da PLAYAPC, foi aplicada um questionário de satisfação de uso (Anexo C) na turma de 2/2016 ministrada pelo Prof. Alexandre Zaghetto, no qual cinco das dez práticas utilizaram a PLAYAPC. O questionário avalia o Guia de Referência, os recursos da biblioteca e se ela cumpriu seu objetivo de auxiliar o ensino.

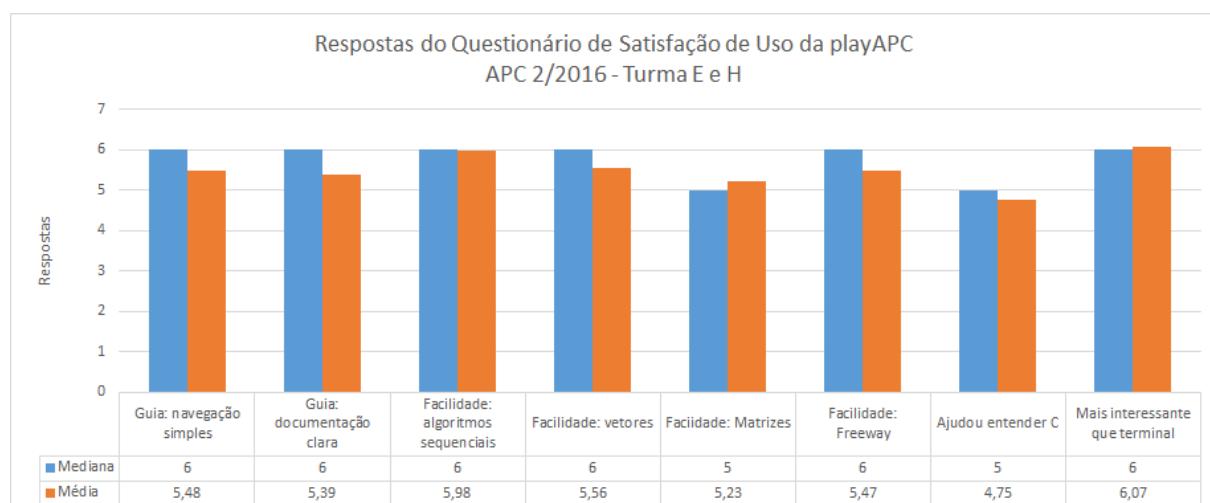


Figura 4.27: Gráfico com o balanço geral das respostas dos alunos de APC das turmas E e H. Respostas em branco foram consideradas com valor 0.

A partir Figura 4.27, é possível inferir que a biblioteca possui uma ótima aceitação, pois:

<sup>3</sup><https://www.youtube.com/watch?v=he11VwfLJDM>

<sup>4</sup><https://www.youtube.com/watch?v=96KXQERnsis>

<sup>5</sup><https://www.youtube.com/watch?v=UlVh9Dw0Sgo>

- Os alunos concordam que:
  - O Guia possui uma navegação simples e sua documentação está clara;
  - Foi fácil utilizar a PLAYAPC nas práticas de algoritmos sequencias, vetores e a realização de um jogo e;
  - Utilizar a PLAYAPC é mais interessante que utilizar somente a tela do terminal para saída do programa.
- Os alunos concordam em parte que:
  - Foi fácil utilizar a PLAYAPC na prática de matrizes e;
  - Utilizar a PLAYAPC ajudou no entendimento da linguagem C.

Para a prática de algoritmos sequenciais, os exercícios aplicados foram os de:

1. Exibir um boneco palito, [Exercício 1.2](#);
2. Exibir a estrela de davi, [Exercício 1.3](#);
3. Exibir um quadrado inscrito, [Exercício 1.4](#);
4. Verificar se ponto está dentro ou fora de uma circunferência, [Exercício 1.5](#) e;
5. Verificar existência de triângulo, [Exercício 1.7](#).

Para a prática de vetores, o exercício aplicado foi o de aplicar filtro de média móvel em um vetor RGB, [Exercício 2.2](#). Para a prática de matrizes, os exercícios aplicados foram os de:

1. Implementar o Jogo da vida, [Exercício 2.3](#);
2. Aplicar filtro de média móvel em uma imagem, [Exercício 2.4](#).

Para a prática de realizar um jogo, o exercício aplicado foi o Certificado de Bravura (Anexo II).

### **4.3.1 Instalação da biblioteca**

Anteriormente, o código fonte da PLAYAPC era anexado ao projeto do aluno e somente a partir daí era usado as funções da biblioteca. Atualmente, o Guia de Referência (Anexo B) oferece passo a passo da instalação, com vídeo explicando onde colocar a biblioteca de vínculo dinâmico e como linkar a biblioteca utilizando a IDE Code::Blocks.

O principal desafio encontrado pela PLAYAPC é a instalação em computadores que possuem apenas versões da OpenGL 1.0, sendo que a PLAYAPC utiliza funções a partir da versão 1.3. Este problema é contornado atualizando os drivers da placa de vídeo, porém nem todos os alunos possuem facilidade neste processo de atualização.

Outro grande desafio é a instalação da biblioteca em plataformas Mac: devido a falta de hardware compatível e falta de suporte das máquinas virtuais utilizadas, não foi possível a geração de binários. Por conta disso, são entregues ao aluno dois arquivos makefile, um para instalação das dependências e outro para instalação da biblioteca propriamente dita. Porém, não é todo aluno de primeiro semestre que possui um Mac que sabe como utilizar um makefile ou até mesmo utilizar o terminal.

### 4.3.2 Utilização da biblioteca

Uma das consequências observadas ao utilizar a PLAYAPC por parte dos alunos foi a liberdade criativa que a biblioteca proporciona. Os enunciados, apesar de serem auto-explicativos, não restringem a utilização dos recursos. Todos possuem acesso ao Guia de Referência (Apêndice B) e, uma vez que suas funções podem ser usadas em qualquer momento do curso, depende somente do grau de criatividade do aluno para deixar a prática mais interessante.

Por exemplo, uma das práticas de algoritmos sequenciais foi a renderização de um boneco palito, [Exercício 1.2](#), na qual foram sugeridas utilização das geometrias como reta e circunferência para serem utilizadas. Um dos alunos teve a liberdade criativa de fazer, ao invés de um boneco palito padrão, um boneco palito dançando, como ilustra a Figura 4.28.

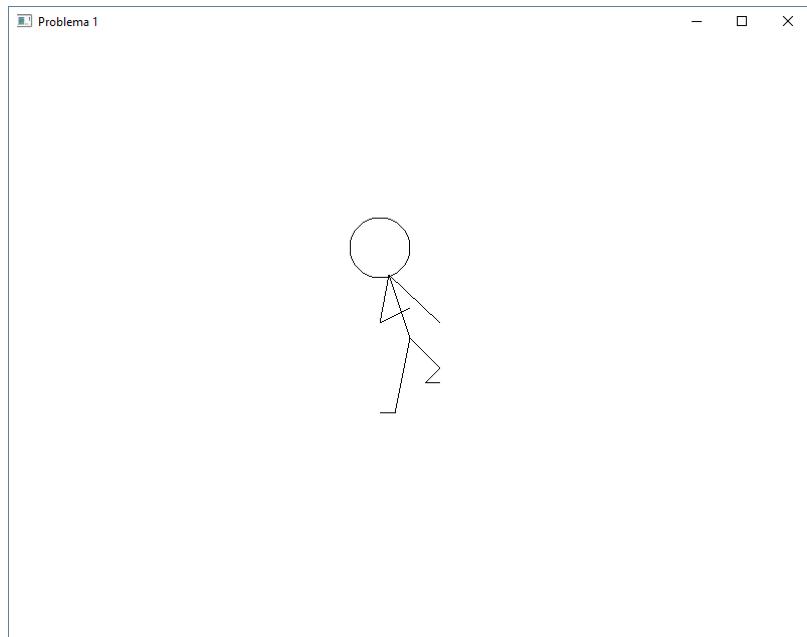


Figura 4.28: Prática 3 de Algoritmos Sequenciais, realizada pelo aluno Bernardo Ferreira Santos Ximbre, **APC** turma E de 2/2016.

Outro exemplo foi a prática de algoritmo de repetição foi a renderização de hélices girando em torno do centro do plano cartesiano, dando a impressão de um ventilador, [Exercício 1.9](#). Um dos alunos adicionou um outro contexto na cena e criou um moinho de vento, como ilustra a Figura 4.29. Mais tarde, esta prática foi reformulada respeitando o contexto criado pelo aluno.

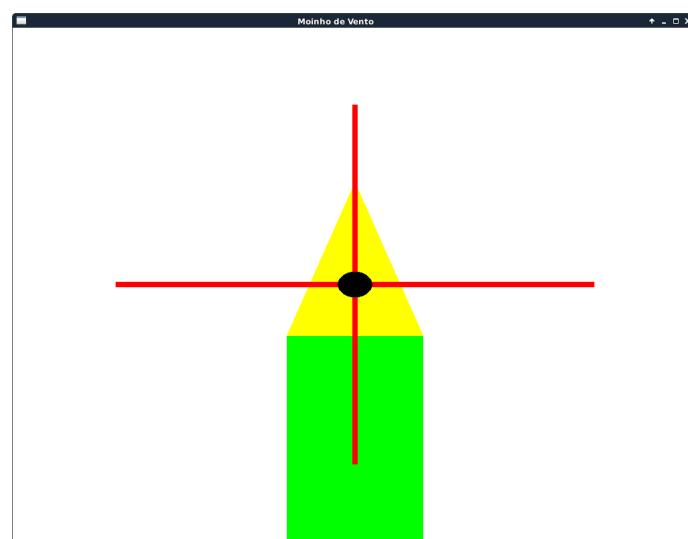


Figura 4.29: Prática 1 de Algoritmos com Repetição, realizada pelo aluno Pedro Paulo de Pinho Matos, **CB** turma B de 2/2014.

Outro exemplo é a prática de vetores, em que dada a entrada de 20 componentes RGB, pedia-se para renderizar os componentes originais e os componentes filtrados, [Exercício 2.2](#). Foi sugerido a utilização de quadrados alinhados verticalmente, mas não era regra. Um dos alunos exibiu os vetores originais e os filtrados em forma de um círculo, onde o círculo externo são os vetores originais e o círculo interno representa os vetores filtrados.



Figura 4.30: Prática 4 de Vetores, realizada pelo aluno Pedro Augusto Coelho Nunes, [APC](#) turma E de 2/2016.

# Capítulo 5

## Conclusão

Com este trabalho, no qual foi proposto o desenvolvimento de uma biblioteca gráfica para ser aplicada na aulas de laboratório, observou-se que houve uma grande aceitação tanto por partes dos professores quanto por partes dos alunos. Os professores propuseram exercícios não listados na apostila, além de experimentarem a biblioteca em projetos pessoais que não estão na ementa de **APC**, e os alunos utilizaram todos os recursos da biblioteca para resolver os exercícios de forma criativa. Sendo assim, concluímos com êxito o objeto de desenvolver uma biblioteca gráfica que pode ser utilizada por programadores com pouca ou nenhuma experiência em computação gráfica, inclusive de forma didática para os alunos do primeiro semestre da disciplina **APC**.

As principais dificuldades encontradas por parte dos alunos que a experimentaram foi em relação a sua instalação, uma vez que os mesmos dificilmente tiveram contato com instalação de bibliotecas e **API**. Portanto, a necessidade de um instalador é evidente, especialmente para usuários Windows e Mac. Após esse processo de instalação, a utilização da biblioteca se torna simples, já que é oferecido o guia de referência das funções disponibilizadas pela **PLAYAPC**. Como sugestão de trabalho com a **PLAYAPC**, uma pesquisa mais detalhada sobre o desempenho de turmas seria interessante, avaliando se a biblioteca aumenta o interesse dos alunos na matéria e, consequentemente, se diminui a evasão no curso.

Por parte dos professores, a maior queixa foi em relação ao desempenho, pois programas que necessitam níveis profundos de recursão ficam prejudicados por conta de sua arquitetura. Uma requisição também por parte dos professores envolvidos com o desenvolvimento da biblioteca foi a manipulação de objetos 3D, uma vez que OpenGL foi desenvolvida visando manipular objetos 3D de forma eficiente. Apesar de ter se iniciado o desenvolvimento de uma biblioteca 3D, a **Simple User Graphics Object Interface 2 (SUGOI)**, o projeto não foi concluído. A **SUGOI** deveria ser o paralelo da **PLAYAPC** envolvendo geometrias em 3D, porém a **PLAYAPC**

evoluía em um ritmo mais acelerado do que era possível implementar funcionalidades para a **SUGOI**. Apesar da ausência 3D, acreditamos que a PLAYAPC possa ajudar no desenvolvimento acadêmico dos discentes na UnB.

# Referências

- [1] Luísa Behrens Palmeira and Matheus Parreira Santos. Mineração de dados para estudar evasão no curso de bacharelado em ciência da computação da univerdasidade de brasília. Monografia. Departamento de Ciência da Computação/UnB. [1](#)
- [2] Ayla Débora Dantas S. Rebouças, Diego Lopez Marques, Luís Feliphe Silva Costa, and Max André de Azevedo Silva. Aprendendo a ensinar programação combinando jogos e python. [2](#)
- [3] Guilherme Germoglio. Arquitetura de software. Eletronicamente, <http://cnx.org/content/col10722/1.9/>, 2012. Acessado em outubro 2016. [3](#)
- [4] Ian Sommerville. *SOFTWARE ENGINEERING*. Addison-Wesley, 2011. [3](#), [4](#), [5](#), [6](#)
- [5] SEBRAE. Manual de ferramentas da qualidade. Eletronicamente, <http://www.dequi.eel.usp.br/~barcza/FerramentasDaQualidadeSEBRAE.pdf>, 2014. Acessado em 14 de setembro de 2016. [4](#)
- [6] Dongping Huang. *An analysis of how static and shared libraries affect memory usage on an IP-STB*, 2010. Master of Science Thesis in the Programme Networks and Distributed Systems. [4](#), [5](#)
- [7] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, 2011. [4](#), [18](#), [19](#)
- [8] Bertrand Meyer. *Object-Oriented - Software Construction*. ISE Inc, 1997. [5](#)
- [9] Edward Angel and Dave Shreiner. *Interactive Computer Graphics*. Addison-Wesley, 2012. [5](#), [13](#), [14](#), [15](#)
- [10] Muhammad Mobeen Movania. *OpenGL Development Cookbook*. Packt Publishing, 2013. [5](#)
- [11] Graham Sellers, Richard S. Wright Jr, and Nicholas Haemel. *OpenGL SuperBible*. Pearson Education Inc, 6 edition, 2014. [5](#)
- [12] Dave Shreiner, Graham Sellers, John Kessenich, and Bill Licea-Kane. *OpenGL Programming Guide*. Addison-Wesley, 2013. [5](#), [7](#), [14](#), [16](#)
- [13] Anton Gerdelen. Anton's Opengl 4 Tutorials. Kindle Edition, 2014. [5](#)

- [14] Henrique Freitas, Mírian Oliveira, Amarolinda Zanel Zaccal, and Jean Moscarola. Utilização da pesquisa de satisfação de clientes como ferramenta para decisões gerenciais e melhoria contínua, 2000. Monografia. Escola de Engenharia/Universidade Federal do Rio Grande do Sul. [6](#)
- [15] Severino Domingos da Silva Júnior and Francisco José Costa. Mensuração e escalas de verificação: uma análise comparativa das escalas de likert e phrase completion, 2014. Revista Brasileira de Pesquisas de Marketing, Opinião e Mídia. [6](#)
- [16] John F. Hughes, Andries Van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics Principles and Practice*. Addison Wesley, 2014. [7](#)
- [17] IVAN EDWARD SUTHERLAND. *SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM*, 1963. [7](#)
- [18] Acm siggraph. Eletronicamente, <http://www.siggraph.org/about/about-acm-siggraph>. Acessado em outubro 2016. [7](#)
- [19] Esteban Walter Gonzalez Clua and João Ricardo Bittencourt. *Desenvolvimento de Jogos 3D: Concepção, Design e Programação*, 2005. Artigo. Centro de Ciências Exatas e Tecnológicas/UNISINOS. [7](#)
- [20] Scott Morrison and Nicholas Harrington. *PONG in Analog*, 2009. Acessado em outubro de 2016 <http://web.mit.edu/sdmorr/Public/6.101/labreport-final.pdf>. [7](#)
- [21] Frans Mäyrä. *An introduction to Game Studies - Games in Culture*. SAGE Publications Ltd, 2008. [7, 9](#)
- [22] Santosh Kumar, Shashi Bhushan Jha, and Rupesh Kumar Singh. *Graphical Process Unit A New Era*, 2014. Journal of Emerging Technologies and Innovative Research. [7, 9](#)
- [23] John E. Warnock. *A HIDDEN SURFACE ALGORITHM FOR COMPUTER GENERATED HALFTONE PICTURES*, 1969. Technical Report. [9](#)
- [24] John Lasseter and Pixar. *MODELS OF LIQUID REFLECTION FOR COMPUTER SYNTHESIZED PICTURES*, 1977. Article ACM SIGGRAPH Computer Graphics. [9](#)
- [25] BRAD MYERS, SCOTT E. HUDSON, and RANDY PAUSCH. *Past, Present, and Future of User Interface Software Tools*, março 2000. ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1. [9](#)
- [26] Richard J. Radke. *Computer Vision for Visual Effects*. Cambridge University Press, 2012. [9](#)
- [27] Mark J. P. Wolf. *The Video Game Explosion - A history from Pong to Playstation and Beyond*. Greenwood, 2008. [9, 10](#)
- [28] James F. Blinn. *PRINCIPLES OF TRADITIONAL ANIMATION APPLIED TO 3D COMPUTER ANIMATION*, 1987. Computer Graphics, Volume 21. [9](#)

- [29] Gordon Bell. *Bell's Law for the Birth and Death of Computer Classes: A theory of the Computer's Evolution*, 2008. IEEE SSCS NEWS. 10
- [30] Chris McClanahan. *History and Evolution of GPU Architecture*, 2010. A Paper Survey. 10, 11, 13
- [31] John Peddie. *The History of Visual Magic in Computers*. Spring Verlag, 2013. 11
- [32] Kevin Hawkins and Dave Astle. *OpenGL Game Programming*. Prima Tech, 2001. 11
- [33] Johnny Ahrens. *Shader Não Fotorrealístico para Desenho de Mangás*, 2009. Monografia. Departamento de Ciência da Computação/UnB. 11
- [34] MARCELLO MARINHO RIBEIRO. *UM ESTUDO DE COMPUTAÇÃO GRÁFICA 2D: MODELAGEM GEOMÉTRICA*, 2012. Monografia. Departamento de Computação/Universidade Católica de Goiás. 11, 17
- [35] Khronos Group. *Vulkan Overview*, 2016. Eletronicamente <https://www.khronos.org/assets/uploads/developers/library/overview/vulkan-overview.pdf>. 11
- [36] Luis Valente. Opengl - um tutorial, 2004. Dissertação (Mestrado). Instituto de Computação/Universidade Federal Fluminense. 13
- [37] Joey de Vries. *Learn OpenGL - An offline transcript of learnopengl.com*. Joey de Vries, 2015. 13
- [38] Manson Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley, 2003. 13
- [39] Norman Chin, Chris Frazier, Paul Ho, Zicheng Liu, and Kevin P. Smith. The opengl graphics system utility library (version 1.3). Disponível em <https://www.opengl.org/registry/doc/glul1.3.pdf>. Acessado em 07 de outubro de 2016. 13
- [40] Jonathan Leigh Dummer. Simple opengl image library. Disponível em <http://www.lonesock.net/soil.html>. Acessado em 16 de agosto de 2016. 13
- [41] Marco Cabral and Paulo Goldfeld. Curso de Álgebra linear - fundamentos e aplicações. Disponível em <http://www.labma.ufrj.br/~mcabral/textos/alglin/CursoAlgLin-livro.pdf>, Outubro 2012. Acessado em 31 de novembro de 2014. 14
- [42] Viviane Cristina Almada de Oliveira. Sobre a produção de significados para a noção de transformação linear em álgebra linear. Dissertação (mestrado). Instituto de Geociências e Ciências Exatas/UNIVERSIDADE ESTADUAL PAULISTA. 14
- [43] Andreia Sartori and Sérgio Tibiriça. *HISTÓRICO SOBRE O DIREITO DA EDUCAÇÃO NAS CONSTITUIÇÕES BRASILEIRAS*, 2014. ETIC 2014 - Encontro de Iniciação Científica. 17
- [44] Douglas Tavares Borges Leal and Edgard Cornachione Jr. *A Aula Expositiva no Ensino da Contabilidade*, 2006. Contab. Vista & Rev., v. 17, n. 3, p. 91-113. 17

- [45] Adriana da Silva Nogueira and Luiz Alexandre da Silva Rosado. *FILOSOFIA LOGO E APRENDIZAGEM DE LÓGICA DE PROGRAMAÇÃO EM CURSO SUPERIOR DE INFORMÁTICA*, 2014. Pôster. Programa de Pós-Graduação em Educação (PPGE / UNESA Rio de Janeiro) Eixo 9. Pesquisa, Artes, Mídias e Educação. 17, 18
- [46] Anabela de Jesus Gomes. *Dificuldades de aprendizagem de programação de computadores: contributos para sua compreensão e resolução*, 2010. Monografia. Departamento de Engenharia Informática/Universidade de Coimbra. 18
- [47] Élia Amaral do Carmo Santos. *O LÚDICO NO PROCESSO ENSINO-APRENDIZAGEM*, 2010. 18
- [48] Brian W.Kernighan and Dennis M.Ritche. *The C Programming Language*. Prentice Hall, 1988. 19, 20
- [49] Bruno Tonet and Cristian Koliver. *Introdução aos algoritmos algoritmos*, 2015. Apostila do Núcleo de Aprendizagem de Programação da Universidade de Caxias do Sul. 20
- [50] Edirlei Soares de Lima and Bruno Feijó. Aprendendo a programar em c com playlib. Disponível em [http://www-di.inf.puc-rio.br/~bfeijo/PlayLib\\_Doc\\_v\\_1\\_3\\_0.pdf](http://www-di.inf.puc-rio.br/~bfeijo/PlayLib_Doc_v_1_3_0.pdf), 2012. Acessado em 29 de setembro de 2016. 20
- [51] Michael Main. Borland graphics interface (bgi) for windows. Disponível em <http://winbgim.codecutter.org/>. Acessado em 02 de outubro de 2014. 22
- [52] Lorraine Figueiroa. Beginner's guide to sdl. Disponível em <http://www.brainycode.com/downloads/LearningSDL011.pdf>, Novembro 2011. Acessado em 30 de agosto de 2014. 22, 23
- [53] Felix Friedrich. Turtle grafik. Disponível em <http://lec.inf.ethz.ch/itet/informatik1/2015/>. Acessado em 18 de agosto de 2016. 22
- [54] Borland Software Corporation. Borlandc. Disponível em <http://borlandc.org/>. Acessado em 02 de outubro de 2014. 22
- [55] Michael Main. Guide to borland graphics interface (bgi) for windows. Disponível em <http://www.cs.colorado.edu/~main/cs1300/doc/bgi/index.html>. Acessado em 02 de outubro de 2014. 22
- [56] Reno Medeiros Dantas. *Um Teste Comparativo entre OpenGL e SDL*, 2012. Monografia. Departamento de Ciência da Computação/UnB. 23, 25
- [57] John B. Schneider, Shira Lynn Broschat, and Jess Dahmen. *Algorithmic Problem Solving with Python*, maio 2015. <http://www.eecs.wsu.edu/~schneidj/PyBook/swan.pdf>. 24

## **Apêndice A**

# **Apostila de exercícios da PLAYAPC**

A apostila de exercícios foi organizada como uma sugestão de práticas de laboratórios para os professores que queiram utilizar a biblioteca gráfica PLAYAPC. Os enunciados estão listados na Seção 3.2 e a resolução dos mesmos exercícios na Seção ??, com exceção do último capítulo da apostila, que contém exercícios extras.

A estrutura da apostila está organizada em 8 capítulos.

- O capítulo 1 possui 4 exercícios de algoritmos sequenciais;
- O capítulo 2 possui 3 exercícios de algoritmos condicionais;
- O capítulo 3 possui 5 exercícios de algoritmos com repetição;
- O capítulo 4 possui 2 exercícios de vetores;
- O capítulo 5 possui 3 exercícios de matrizes;
- O capítulo 6 possui 3 exercícios de função;
- O capítulo 7 possui 4 exercícios de recursão;
- O capítulo 8 possui 3 exercícios extras;

O capítulo 8 possui exercícios que referenciam os demais capítulos, envolvendo todos os tópicos abordados do 1 ao 7.

# Apêndice B

## Guia de referência da PLAYAPC

**APC play**

**HOME** **INSTALAÇÃO** ▾ **DOCUMENTAÇÃO** ▾ **LISTA DE FUNÇÕES** **EXEMPLOS** ▾

### GUIA DE REFERÊNCIA DA PLAYAPC

A playAPC é uma biblioteca educacional de programação voltada para os alunos de Algoritmos e Programação de Computadores (APC) da UnB que tem como objetivo ilustrar, de maneira gráfica, os conceitos aprendidos durante o curso com aplicações simples. Com ela, é possível criar animações simples e desenhos estáticos.

A biblioteca consiste em um conjunto de funções para criação e manipulação de formas 2D e utilização de imagens. Utiliza a API OpenGL e GLFW.

A playAPC foi desenvolvida usando o conceito de Orientação a Objetos de C++, por praticidade de desenvolvimento e facilidade de encapsulamento. Entretanto, a playapc.h entrega ao aluno uma interface amigável com chamadas de funções simples para a criação de todos os objetos, utilizando o paradigma Imperativo.

Neste guia, você encontrará todas as funções da playAPC explicadas e exemplificadas, a fim de facilitar o uso da biblioteca

### REQUISITOS

Esta biblioteca está configurada para executar juntamente com a GLFW 2.7 e OpenGL a partir da

**CATEGORIAS**

- Como instalar
- Exemplos
  - 2/2016
- Básicos
- Funções
  - Essencias
  - Extras
    - Auxílio
    - Cor
    - Grupo
    - Imagem
    - Input
  - Geometrias
  - Transformações

Figura B.1: Tela principal do Guia de Referência.

O Guia de Referência, disponibilizado em plataforma online, está separado em quatro guias de navegação:

1. Instalação
2. Documentação
3. Lista de Funções
4. Exemplos

## B.1 Instalação

O Guia ilustra passo a passo o processo de instalação da biblioteca, separado por sistema operacional. Há a aba para Windows, Linux e Mac. Para os três sistemas operacionais, há pelo menos quatro seções para o processo de instalação:

1. Verificação se o computador tem suporte à OpenGL
2. Instalação de dependência
3. Instalação dos binários da biblioteca
4. Verificação de a biblioteca foi corretamente instalada

Para o sistema operacional Windows, há ainda um vídeo-tutorial explicando como realizar a instalação da biblioteca, englobando os passos de instalação de dependências até a verificação se foi instalada corretamente.

## B.2 Documentação

O Guia de Referência apresenta a documentação das funções da interface da PLAYAPC, assim como está indicado na Seção 3.1.2. Detalhes de implementação não foram descritos por não ser interessante no ponto de vista do aluno, o qual apenas precisa saber como utilizar as funções. Cada função documentada apresenta um resumo da sua ação e uma explicação de seus argumentos e retorno quando existirem. Além disso, cada função possui, pelo menos, um exemplo de como utilizá-la, com capturas de tela e código fonte.

## **B.3 Lista de funções**

O Guia disponibiliza uma navegação rápida para busca de funções, listando todas as funções criadas em ordem alfabética e listando-as também por versões da biblioteca.

## **B.4 Exemplos**

O Guia disponibiliza cinco exemplos criados pelos desenvolvedores da biblioteca, mostrando como utilizar um conjunto diferenciado de funções da **PLAYAPC** em programas mais contextualizados. Há ainda uma aba de exemplos criados pelos alunos que utilizaram a biblioteca, sendo disponibilizado apenas os executáveis de seus programas. Os programas atualmente disponíveis foram cedidos pelos alunos das turmas E e H do semestre 2/2016 , ministrado pelos professores Alexandre Zaghetto e Carla Castanho.

## **Apêndice C**

# **Questionário qualitativo sobre o uso da PLAYAPC**

## Questionário sobre o uso da playAPC

Agradecemos a sua participação em nossa pesquisa, fundamental para desenvolvermos mais recursos para a biblioteca. A análise dos dados obtidos terá finalidade exclusivamente acadêmica.

Turma: \_\_\_\_\_ Seu curso: \_\_\_\_\_

Selecione o sistema operacional onde você instalou e utilizou a biblioteca:

( ) Windows ( ) Linux ( ) Mac ( ) Outro ( ) Não instalei

**Orientação:** Avalie o seu grau de satisfação sobre cada item da pesquisa, usando uma escala de 1 a 7, onde cada valor significa:

- |  |   |
|--|---|
| (1) Concordo completamente                                 | (2) Concordo                              |
| (3) Concordo em parte (discordo <i>mais que concordo</i> ) | (4) <i>Nem discordo nem concordo; +/-</i> |
| (5) Concordo em parte (Concordo <i>mais que discordo</i> ) | (6) Concordo                              |
| (7) Concordo completamente                                 |   |

### 1. Como você avalia o **Guia de Referência**:

- A navegação era simples e não tive dificuldades de encontrar uma funcionalidade específica:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

- A documentação estava clara e não tive dificuldades para entender como a função funcionava:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

### 2. Como você avalia a sua **facilidade de usar funções da biblioteca** nas práticas de laboratório. Leve em consideração o nome da função, a ação da função e o contexto onde ela estava sendo utilizada; *cuidado, avalie somente o uso da biblioteca, não a complexidade da prática.*

- Algoritmos sequencias e com alternativas utilizando a playAPC :  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

- Vetores e playAPC:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

- Matrizes e playAPC:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

- Atividade Livre:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

### 3. Como você avalia o **uso da biblioteca no geral**:

- Entendi melhor o funcionamento da linguagem C graças a biblioteca:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

- É mais interessante exibir o programa com a playAPC do que exibir com o terminal:  
1 ( )      2 ( )      3 ( )      4 ( )      5 ( )      6 ( )      7 ( )

### 4. Que melhores você sugeriria para a playAPC para os próximos semestres? Use o verso da folha, se necessário.

## Apêndice D

### Logo da PLAYAPC



(a) Colorido.



(b) Tons de cinza.

Figura D.1: Logo oficial da PLAYAPC: a)Colorido; b) Tons de cinza.

O logo da PLAYAPC, criado pelo *designer* Antônio Cândido Neto<sup>1</sup>, foi pensado para trazer a ideia de diversão, onde o aluno abre uma janela de contexto e se diverte desenhando nessa janela, e trazer junto a identidade da UnB, onde foi desenvolvida.

---

<sup>1</sup>[antonio.cdasn@gmail.com](mailto:antonio.cdasn@gmail.com)

# Apêndice E

## Códigos fontes dos práticas sugeridas

O enunciado das práticas deste Apêndice podem ser encontradas na Seção 4.1.

### E.1 Tipo de dados e estruturas de controle

Exercício 1.1.

Listagem E.1: Código fonte de Plano Cartesiano

```
1 #include <playAPC/playapc.h>
2 int main(){
3
4     Ponto p;
5     AbreJanela(800, 600, "Ola Mundo");
6
7     PintarFundo(255, 255, 255);
8     MostraPlanoCartesiano(10);
9
10    p.x = 133;
11    p.y = 0;
12
13    CriaCirculo(3, p);
14    Pintar(255, 0, 0);
15
16    Desenha();
17 }
```

Exercício 1.2.

Listagem E.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;
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,
15    ponto central)
16  Pintar(255, 233, 234);
17
18  p.x = -40;
19  p.y = 30;
20  CriaRetangulo(80, 10, p); //(base, altura, ponto esquerdo inferior)
21  Pintar(255, 233, 234);
22
23  p.x = -40;
24  p.y = -30;
25  CriaTriangulo(80, 40, p); //(base, altura, ponto esquerdo inferior)
26  Pintar(255, 233, 234);
27
28  p.x = -15;
29  p.y = -40;
30  CriaQuadrado(10, p); //(lado, ponto esquerdo inferior)
31  Pintar(255, 233, 234);
32
33  p.x = 5;
34  p.y = -40;
35  CriaQuadrado(10, p);
36  Pintar(255, 233, 234);
37
38  Desenha();
39 }

```

### Exercício 1.3.

Listagem E.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 }
```

### Exercício 1.4.

Listagem E.4: Código fonte do quadrado inscrito

```

1 #include <playAPC/playapc.h>
2 #include <stdio.h>
3 #include <math.h>
4 int main(){
5     float raio, lado, apotema;
6     Ponto p1, p2;
7
8     printf("Digite o valor do raio do circulo: ");
9     scanf("%f", &raio);
10
11    lado = raio * sqrt(2);
12    apotema = lado/2;
13
14    p1.x = 0;
15    p1.y = 0;
16
17    p2.x = -apotema;
18    p2.y = -apotema;
19
20    AbreJanela(400, 400, "Quadrado inscrito");
21    PintarFundo(0, 0, 0);
22
23    CriaCirculo(raio, p1);
24    Pintar(255, 0, 0);
25
26    CriaQuadrado(lado, p2);
27    Pintar(255, 255, 255);
28
29    Desenha();
30
31    return 0;
32 }
```

### Exercício 1.5.

Listagem E.5: 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 }

```

### Exercício 1.6.

#### Listagem E.6: 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 }
```

### Exercício 1.7.

Listagem E.7: 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 }
```

### Exercício 1.8.

Listagem E.8: 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        DesenhaFrame();
35    }
36    Desenha();
37 }
```

### Exercício 1.9.

Listagem E.9: 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  p1.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     DesenhaFrame();
62     Gira(angulo, helices); @\label{line:Giram1}@
63     angulo++;
64 }
65
66 Desenha();
67 return 0;
68 }
```

### Exercício 1.10.

#### Listagem E.10: Código fonte do sistema solar

```

1 #include <playAPC/playapc.h>
2 #include <math.h>
3
4 int main () {
5     Ponto p1, p2, p3;
6     int orbitalua, terra, lua, sol;
7     double ang, focoSol, focoTerra;
8
9     AbreJanela (400,400, "Sistema Solar");
10    PintarFundo (0, 0, 0);
11
12
13 //CriaElipse(70, 50 , p1); //Orbita da Terra
14 for(ang = 0; ang < 2*PI; ang += 0.01){
15     p1.x = 70*cos(ang);
16     p1.y = 50*sin(ang);
17
18     CriaPonto(p1);
19     Pintar (255,0,0); Grafite(3);
20 }
21
22 focoSol = sqrt(pow(70, 2) - pow(50, 2)); //a^2 = b^2 + c^2
23
24 orbitalua = CriaGrupo();
25 //CriaElipse(15, 10, p1); //Orbita da Lua
26 for(ang = 0; ang < 2*PI; ang+= 0.01){
27     p1.x = 13*cos(ang);
28     p1.y = 11*sin(ang);
29
30     CriaPonto(p1);
31     Pintar (0,255,255);
32 }
33
34 focoTerra = sqrt(pow(13, 2) - pow(11, 2)); //a^2 = b^2 + c^2
35
36 p1.y = 0;
37
38 p1.x = focoTerra;
39 terra = CriaGrupo();
40 CriaCirculo(5, p1); Pintar (0,0,255); //Terra
41
42 p1.x = 0;
43 lua = CriaGrupo();
44 CriaCirculo(2, p1); Pintar (100, 100, 100); //Lua
```

```

45
46
47 p1.x = focoSol;
48 sol = CriaGrupo();
49 CriaCirculo(10, p1); Pintar (255,255,0); //Sol
50
51
52 for (ang = 0; ; ang +=.01) {
53     p2.x = 70*cos(ang);
54     p2.y = 50*sin(ang);
55     Move(p2,terra); //translada Terra para sua orbita
56
57     p3.x = 13*cos(ang*15)+ p2.x + focoTerra;
58     p3.y = 11*sin(ang*15)+ p2.y;
59     Move(p3,lua); //translada Lua para sua orbita
60
61     p2.x += 13 + focoTerra;
62     Move(p2,orbitalua); //em torno da Terra
63
64     DesenhaFrame();
65 }
66
67 Desenha(); // (desnecessario) mantem janela aberta
68 return 0;
69 }
```

### Exercício 1.11.

#### Listagem E.11: 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     /*
9      Espiral Hiperbólica: equação em coordenadas cartesianas
10     x = a*cos(t)/t
11     y = a*sin(t)/t
12
13     a é a assintota para y (reta paralela ao eixo x)
14     t equivalente ao angulo em coordenadas polares
15     */
16     Ponto p, q, r;
17
18     // for (double t = 0; t < 4*PI; t += .01){
19
20     /* espiral hiperbolica, caminhando do "fim" pro centro (0,0)
21     p.x = 100*cos(t)/t;
22     p.y = 100*sin(t)/t;
23     */
24     int k = 0;
25     for (double t = 4*PI; t > 0 ; t -= .05) {
26         p.x = 100*cos(t)/t;           q.x = -p.x;
27         p.y = 100*sin(t)/t;           q.y = -p.y;
```

```

28
29     printf("p.x: %f\t p.y: %f\n", p.x, p.y);
30     printf("q.x: %f\t q.y: %f\n", q.x, q.y);
31
32     CriaPonto (p);
33     Pintar (200, 30, 100);
34     Grafite(3);
35
36     CriaPonto (q);
37     Pintar (100, 30, 200);
38     Grafite(3);
39
40     Desenha1Frame();
41     k++;
42 }
43
44
45 //A massive Black Hole in the very centre
46 //If you want to see (the unseeable) black hole
47 //paint the background on a different colour
48 r.x =0;      r.y = 0;
49 CriaCirculo(8, r);
50 Pintar (0, 0, 0);
51
52 for (double t=0; ; t += .5) {
53     Gira(t);
54     Desenha1Frame();
55
56     //Depois de um tempinho, pinta o fundo de branco pra mostrar
57     //o buraco negro
58     if ( t > 200 ) PintarFundo (255, 255, 255);
59
60     //quebra o loop e encerra o programa
61     if(ApertouTecla(GLFW_KEY_ENTER)) return 0;
62 }
63
64 }

```

### Exercício 1.12.

#### Listagem E.12: Código fonte do Mario animado

```

1 #include <playAPC/playapc.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <math.h>
5
6 int main(){
7     Ponto p, pcano1, pcano2, pmario;
8     int imgbackground, imgcano,
9         imgmario_caminhando_1, imgmario_caminhando_2, imgmario_caminhando_3,
10        imgmario_pre_caminhada, imgmario_pre_salto, imgmario_salto;
11     int grupocano, grupomario, grupoback;
12     float t, x, y;
13     int raio_pulo = 40;
14
15     srand (time(NULL));

```

```

15  AbreJanela(600, 385, "Run, Mario, run!");
16
17  imgbackground = AbreImagem("Mario_Run/background.png"); @{ \label{line:AbreImagem}@
18  imgcano = AbreImagem("Mario_Run/cano.png");
19  imgmario_caminhando_1 = AbreImagem("Mario_Run/mario_caminhando_1.png");
20  imgmario_caminhando_2 = AbreImagem("Mario_Run/mario_caminhando_2.png");
21  imgmario_caminhando_3 = AbreImagem("Mario_Run/mario_caminhando_3.png");
22  imgmario_pre_caminhada = AbreImagem("Mario_Run/mario_pre_caminhada.png");
23  imgmario_pre_salto = AbreImagem("Mario_Run/mario_pre_salto.png");
24  imgmario_salto = AbreImagem("Mario_Run/mario_salto.png");
25
26 //Background
27 grupoback = CriaGrupo();
28     p.x = -156;
29     p.y = -100;
30     CriaRetangulo(312, 200, p);
31     Pintar(255, 255, 255);
32     AssociaImagem(imgbackground); @{ \label{line:AssociaImagemm1}@
33
34 //Canos
35 grupocano = CriaGrupo();
36     pcano1.x = -(100 - rand()%50);
37     pcano1.y = -70;
38     CriaQuadrado(30, pcano1);
39     Pintar(255, 255, 255);
40     AssociaImagem(imgcano);
41
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); @{ \label{line:ultimoRetangulo}@
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
77             porque ta indo de 0 a PI)
78         y = 50 * sin(t);
79
80         p.x = pmario.x + raio_pulo + x; //posicao do deslocamento x + raio 30 +
81             posicao inicial do Mario
82         p.y = pmario.y + y;
83
84         Move(p, grupomario);
85         Desenha1Frame();
86         Desenha1Frame();
87         Desenha1Frame();
88
89         t+= 0.5;
90     }
91     p.y = -75;
92 }
93
94 if(p.x + raio_pulo > pcano2.x && p.x < pcano2.x + raio_pulo){
95     AssociaImagem(imgmario_pre_salto);
96     t = 0;
97     while(t < 50){
98         t++;
99         Desenha1Frame();
100    }
101
102    AssociaImagem(imgmario_salto);
103    t = 0;
104    pmario = p;
105    while(t <= PI){
106        x = -(raio_pulo * cos(t)); //circulo (0,0) com raio 30 (x esta invertido
107            porque ta indo de 0 a PI)
108        y = 50 * sin(t);
109
110        p.x = pmario.x + raio_pulo + x; //posicao do deslocamento x + raio 30 +
111            posicao inicial do Mario
112        p.y = pmario.y + y;
113
114        Move(p, grupomario);
115        Desenha1Frame();
116        Desenha1Frame();
117        Desenha1Frame();
118
119        t+= 0.5;
120    }
121    p.y = -75;
122 }
123
124 AssociaImagem(imgmario_caminhando_1);
125 Desenha1Frame();
126 Desenha1Frame();
127 Desenha1Frame();
128 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 }
```

## E.2 Vetores e Matrizes

### Exercício 2.1.

Listagem E.13: 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 }

```

### Exercício 2.2.

Listagem E.14: 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 dist = 0;
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 %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         dist += sqrt(pow(R[i] - Rf[i], 2) + pow(G[i] - Gf[i], 2) + pow(B[i] - Bf[i], 2));
65     }
66     dist = dist/(QTD-2);
67
68     printf("A distancia euclidiana media e %.2f\n", dist);
69
70     Desenha();
71 }

```

### Exercício 2.3.

Listagem E.15: 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][
15         COLUNAS];
16     int l, c, i, j, soma, r, s, n, ESTADO, deltaQuad=10, largQuad = 6, marg = 2;
17     Ponto p1, p2, p3;
18
19     /////////////////
20     // Define o Janela          //
21     /////////////////
22     AbreJanela(550, 550, "Game of life");
23     PintarFundo(255, 255, 255);
24
25     ///////////////
26     // Desehma Grid //
27     ///////////////

```

```

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, 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;
76 geracaoAtual[5][6] = VIVO; geracaoAtual[5][7] = VIVO;
77 geracaoAtual[6][3] = VIVO; geracaoAtual[6][5] = VIVO; geracaoAtual[6][7] = VIVO;
78 geracaoAtual[7][5] = VIVO; geracaoAtual[7][6] = VIVO;
79
80 // Realiza a reflexao do padrao
81 for(l = 0; l < LINHAS/2; l++) {
82 for(c = 0; c < COLUNAS/2; c++) {
83 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)-1][(COLUNAS-1)-c] = geracaoAtual[l][c]; // Inferior
85             direito
86     }
87 }
88
89 while(1){
90
91     // Faz a copia da geracao atual para a geracao anterior
92     for(l = 0; l < LINHAS; l++){
93         for(c = 0; c < COLUNAS; c++) {
94             proximaGeracao[l][c] = geracaoAtual[l][c];
95         }
96     }
97
98     for(l=0; l<LINHAS; l++) {
99         for(c=0; c<COLUNAS; c++) {
100             if(proximaGeracao[l][c]==1) {
101                 printf("%c", 219);
102                 Pintar(0, 0, 0, QUADRADO, geoIndex[l][c]);
103             } else{
104                 printf("%c", proximaGeracao[l][c], ' ');
105                 Pintar(255, 255, 255, QUADRADO, geoIndex[l][c]);
106             }
107             printf("\n");
108     }
109     DesenhalFrame();
110
111     while(!ApertouTecla(GLFW_KEY_ENTER))
112         DesenhalFrame();
113
114     // Conta a quantidade de vizinhos de uma celula
115     for (l = 1; l < LINHAS-1; l++){
116         for(c = 1; c < COLUNAS-1; c++) {
117             ESTADO = proximaGeracao[l][c];
118             soma = 0;
119             for(r = l-1; r < l+2; r++)
120                 for(s = c-1; s < c+2; s++)
121                     soma+=proximaGeracao[r][s];
122
123             if(proximaGeracao[l][c] == VIVO) soma--;
124
125             //Define se uma celula deve morrer, permanecer viva ou nascer
126             if ((ESTADO == VIVO && soma < 2) || (ESTADO == VIVO && soma > 3)){
127                 geracaoAtual[l][c] = MORTO;
128             } else if ((ESTADO == VIVO && (soma > 2 || soma <= 3)) || (ESTADO == MORTO &&
129                 soma == 3)){
130                 geracaoAtual[l][c] = VIVO;
131             }
132         }
133         waitFor (1);
134         system("cls");
135     }
136
137     Desenha();

```

```

138     return 0;
139 }
140
141 void waitFor (unsigned int secs) {
142     unsigned int retTime;
143     retTime = time(0) + secs;      // Get finishing time.
144     while (time(0) < retTime);    // Loop until it arrives.
145 }
```

### Exercício 2.4.

Listagem E.16: 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); @{ \label{line:ExtraiRGBdeBMP} @
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         }
46     }
47 }
```

```

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 }

```

## Exercício 2.5.

Listagem E.17: Código fonte da batalha naval

```

1 #include <playAPC/playapc.h>
2 #include <math.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 typedef struct{
7     Ponto esq;
8     int index;
9 }tipoVisu;

```

```

10
11 //Legenda para mLog
12 // -2: água atingida
13 // -1: navio afundado
14 // 0 : água
15 // 1 : navio escondido
16 // 2 : navio atingido, mas não afundado
17 // 3: água ao redor do navio (auxiliar, depois será substituído por 0)
18
19 int main(){
20     int gfundo, glicavel, gbalao, aberta = 1, contnavio = 0, pedaco;
21     Ponto p, p2;
22     int mLog[9][9];
23     tipoVisu mVisu[9][9];
24     int xtab = 0, ytab = 0, xpos, ypos, x, y;
25     int imgponteiro, imgaviso, imgtransparente, imgagua, imgfogo, imgdestrocos, imgmar,
26         imgtelabatalha;
27     int tam, qtd, sentido, valido;
28
29     srand(time(NULL));
30
31     AbreJanela(850, 666, "Batalha Naval");
32     MostraPlanoCartesiano(10);
33
34     imgponteiro = AbreImagem("batalha/selecao.png");
35     imgtransparente = AbreImagem("batalha/areanada.png");
36     imgagua = AbreImagem("batalha/agua.png");
37     imgfogo = AbreImagem("batalha/Fogo.png");
38     imgdestrocos = AbreImagem("batalha/destrocos.png");
39
40     //cria fundo
41     gfundo = CriaGrupo();
42
43     imgmar = AbreImagem("batalha/mar12x12.png");
44     imgtelabatalha = AbreImagem("batalha/TelaBatalha.png");
45
46     p.x = -120;
47     p.y = -80;
48     CriaRetangulo(230, 170, p);
49     Pintar(255, 255, 255);
50     AssociaImagem(imgmar);
51
52     p.y = 80;
53     p2.y = -80;
54     p.x = -120;
55     p2.x = -120;
56     for(int i = 0; i < 10; i++) {
57         CriaReta(p, p2);
58         Pintar(255, 255, 255);
59         Grafite(5);
60
61         p.x += 25.1;
62         p2.x += 25.1;
63     }
64
65     p.y = -80;

```

```

66     p2.y = -80;
67     p.x = 107;
68     p2.x = -120;
69     for(int i = 0; i < 10; i++) {
70         CriaReta(p, p2);
71         Pintar(255, 255, 255);
72         Grafite(5);
73
74         p.y += 17.8;
75         p2.y += 17.8;
76     }
77
78     p.x = -127.62;
79     p.y = -100;
80     CriaRetangulo(255, 200, p);
81     Pintar(255, 255, 255);
82     AssociaImagem(imgtelabatalha);
83     //fim de criafundo
84
85     //cria tabuleiro
86     glicavel = CriaGrupo();
87     p.y = 63.5;
88     for(int i = 0; i < 9; i++) {
89         p.x = -119.5;
90         for(int j = 0; j < 9; j++) {
91
92             mLog[i][j] = 0;
93             mVisu[i][j].index = CriaRetangulo(24, 17, p);
94             mVisu[i][j].esq = p;
95             Pintar(255, 255, 255);
96             AssociaImagem(imgtransparente);
97
98             p.x += 25;
99
100        }
101        p.y -= 18;
102    }
103
104    //comeco de sorteiaBarcos
105    for(tam = 3; tam > 0; tam--) {
106
107        qtd = (tam == 1 ? 4 : 3); //se barco for de tamanho 1, então há 4 barcos. Se não, há 3
108        barcos
109
110        for(int q = 0; q < qtd; q++) {
111            do{
112                x = rand()%9;
113                y = rand()%9;
114                sentido = rand()%2;
115
116                switch(sentido){
117                    case 0: //horizontal
118                        if(x > 9 - tam)
119                            x = 9 - tam;
120                        break;
121                    case 1: //vertical
122                        if(y > 9 - tam)

```

```

122             y = 9 - tam;
123             break;
124         }
125         valido = 1;
126         //inicio de posição válida
127         switch(sentido){
128             case 0: //horizontal
129                 for(int i = 0; i < tam; i++){
130                     //verifica os vizinhos
131                     for(int j = -1; j <= 1; j++){
132                         for(int k = -1; k <= 1; k++){
133                             if(((x + i + j < 9) && (x + i + j >= 0)) && ((y + k) < 9) && ((y + k) >= 0))){
134                                 if(mLog[y + k][x + i + j] == 1) //se tiver um barco
135                                     valido = 0; //nao é posição válida
136                             }
137                         }
138                     }
139                 }
140             }
141             break;
142             case 1: //vertical
143                 for(int i = 0; i < tam; i++){
144                     //verifica os vizinhos
145                     for(int j = -1; j <= 1; j++){
146                         for(int k = -1; k <= 1; k++){
147                             if(((x + j < 9) && (x + j >= 0)) && ((y + i + k) < 9) && ((y + i + k) >= 0))){
148                                 if(mLog[y + i + k][x + j] == 1) //se tiver um barco
149                                     valido = 0; //nao é posição válida
150                             }
151                         }
152                     }
153                 }
154             }
155             break;
156         //fim de posição válida
157     }while(!valido);
158
159     switch(sentido){
160         case 0: //horizontal
161             for(int i = 0; i < tam; i++){
162                 mLog[y][x + i] = 1; //barco
163
164                 for(int j = -1; j <= 1; j++){
165                     for(int k = -1; k <= 1; k++){
166                         if(((x + i + j < 9) && (x + i + j >= 0)) && ((y + k) < 9) && ((y + k) >= 0)){
167                             if(mLog[y + k][x + i + j] == 0)
168                                 mLog[y + k][x + i + j] = 3; //água espaçada
169                         }
170                     }
171                 }
172             }
173             break;
174
175         case 1: //vertical

```

```

176         for(int i = 0; i < tam; i++) {
177             mLog[y + i][x] = 1; //barco
178
179             for(int j = -1; j <= 1; j++) { //x
180                 for(int k = -1; k <= 1; k++) { //y
181                     if((((x + j < 9) && (x + j >= 0)) && ((y + i + k) < 9) && (y +
182                         i + k >= 0)) {
183                         if(mLog[y + i + k][x + j] == 0)
184                             mLog[y + i + k][x + j] = 3; //agua espaçada
185                     }
186                 }
187             }
188             break;
189         }
190     }
191 }
192 //fim de sorteia barcos
193 //inicio de limpa água Aux
194 for(int i = 0; i < 9; i++) {
195     for(int j = 0; j < 9; j++) {
196         if(mLog[i][j] == 3)
197             mLog[i][j] = 0;
198     }
199 }
200 //fim de limpa água Aux
201
202 //início de imprimeMatriz da resposta
203 for(int i = 0; i < 9; i++) {
204     for(int j = 0; j < 9; j++) {
205
206         switch(mLog[i][j]) {
207             case 0:
208                 printf("%c ", 176);
209                 break;
210
211             case 1:
212                 printf(".");
213                 break;
214             }
215         }
216     }
217     printf("\n");
218 }
219 //fim de imprimeMatriz da resposta
220 //fim de cria tabuleiro
221
222 gbalao = CriaGrupo();
223 imgaviso = AbreImagem("batalha/playapc.png");
224 p.x = -105;
225 p.y = -20;
226 CriaRetangulo(200, 77, p);
227 Pintar(255, 255, 255);
228 AssociaImagem(imgaviso);
229
230 while(!ApertouBotaoMouse(GLFW_MOUSE_BUTTON_LEFT))
231     DesenhaFrame();

```

```

232
233     ApagaGrupo(gbalao);
234
235     while(aberta && contnavio != 19){
236         //pega posição do tabuleiro
237         PosicaoMouse(&xpos, &ypos);
238
239         //descobrindo posição ytab
240         for(int i = 0; i < 9; i++){
241
242             if(mVisu[i][0].esq.y <= ypos && mVisu[i][0].esq.y + 18 >= ypos) {
243                 ytab = i;
244                 break;
245             }
246         }
247
248         //descobrindo posição xtab
249         for(int j = 0; j < 9; j++){
250
251             if(mVisu[0][j].esq.x <= xpos && mVisu[0][j].esq.x + 25 >= xpos) {
252                 xtab = j;
253                 break;
254             }
255         }
256         //fim de pega posição do tabuleiro
257
258         //inicio de muda cursor
259         pedaco = 0;
260
261         if(mLog[ytab][xtab] == 0 || mLog[ytab][xtab] == 1) //Se não tiver preenchido
262             AssociaImagem(imgponteiro, RETANGULO, mVisu[ytab][xtab].index);
263
264         if(ApertouBotaoMouse(GLFW_MOUSE_BUTTON_LEFT)) {
265             if(mLog[ytab][xtab] == 0) {
266                 mLog[ytab][xtab] = -2;
267                 AssociaImagem(imgagua, RETANGULO, mVisu[ytab][xtab].index);
268             }
269             else if(mLog[ytab][xtab] == 1) {
270
271                 int vizinho = ytab - 1;
272
273                 //esquerda
274                 while(vizinho >= 0 && mLog[vizinho][xtab] > 0) {
275                     if(mLog[vizinho][xtab] == 1)
276                         pedaco++;
277                     vizinho--;
278                 }
279
280                 vizinho = ytab + 1;
281                 //direita
282                 while(vizinho < 10 && mLog[vizinho][xtab] > 0) {
283                     if(mLog[vizinho][xtab] == 1)
284                         pedaco++;
285                     vizinho++;
286                 }
287
288                 vizinho = xtab - 1;

```

```

289 //cima
290 while(vizinho >= 0 && mLog[ytab][vizinho] > 0){
291     if(mLog[ytab][vizinho] == 1)
292         pedaco++;
293     vizinho--;
294 }
295
296 vizinho = xtab + 1;
297 //baixo
298 while(vizinho < 10 && mLog[ytab][vizinho] > 0){
299     if(mLog[ytab][vizinho] == 1)
300         pedaco++;
301     vizinho++;
302 }
303
304 if(pedaco > 0){
305     AssociaImagem(imgfogo, RETANGULO, mVisu[ytab][xtab].index);
306     mLog[ytab][xtab] = 2;
307 }
308 else{
309     AssociaImagem(imgdestrocos, RETANGULO, mVisu[ytab][xtab].index);
310     mLog[ytab][xtab] = -1;
311
312 //esquerda
313 vizinho = ytab - 1;
314 while(vizinho >= 0 && mLog[vizinho][xtab] == 2){
315     AssociaImagem(imgdestrocos, RETANGULO, mVisu[vizinho][xtab].index);
316     mLog[vizinho][xtab] = -1;
317
318     vizinho--;
319 }
320
321 vizinho = ytab + 1;
322 //direita
323 while(vizinho < 10 && mLog[vizinho][xtab] == 2){
324     AssociaImagem(imgdestrocos, RETANGULO, mVisu[vizinho][xtab].index);
325     mLog[vizinho][xtab] = -1;
326
327     vizinho++;
328 }
329
330 vizinho = xtab - 1;
331 //cima
332 while(vizinho >= 0 && mLog[ytab][vizinho] == 2){
333     AssociaImagem(imgdestrocos, RETANGULO, mVisu[ytab][vizinho].index);
334     mLog[ytab][vizinho] = -1;
335
336     vizinho--;
337 }
338
339 vizinho = xtab + 1;
340 //baixo
341 while(vizinho < 10 && mLog[ytab][vizinho] == 2){
342     AssociaImagem(imgdestrocos, RETANGULO, mVisu[ytab][vizinho].index);
343     mLog[ytab][vizinho] = -1;
344
345     vizinho++;

```

```

346                     }
347
348                 }
349             }
350         }
351         //fim de muda cursor
352
353     contnavio = 0;
354     for(int i = 0; i < 9; i++) {
355         for(int j = 0; j < 9; j++) {
356             if(mLog[i][j] == -1)
357                 contnavio++;
358         }
359     }
360     aberta = DesenhalFrame();
361
362     //inicio de muda cursor
363     pedaco = 0;
364
365     if(mLog[ytab][xtab] == 0 || mLog[ytab][xtab] == 1) //Se não tiver preenchido
366         AssociaImagem(imgtransparente, RETANGULO, mVisu[ytab][xtab].index);
367
368     //fim de muda cursor
369 }
370
371     gbalao = CriaGrupo();
372     imgaviso = AbreImagem("batalha/playapc_youwin.png");
373     p.x = -105;
374     p.y = -20;
375     CriaRetangulo(200, 77, p);
376     Pintar(255, 255, 255);
377     AssociaImagem(imgaviso);
378
379     Desenha();
380
381     return 0;
382 }
```

## E.3 Subalgoritmos

### Exercício 3.1.

Listagem E.18: Código fonte do lançador balístico

```

1 #include <playAPC/playapc.h>
2 #include <time.h>
3
4 int colisaoBolinhaRetangulo(int raio, int largurapredio, Ponto alvo, Ponto partida, Ponto
    movimentacao) {
5
6     if( movimentacao.y - raio <= alvo.y
7         && movimentacao.y - raio > alvo.y - 1           //tolerancia para pouso no eixo y
8         && movimentacao.x + raio < alvo.x + largurapredio
```

```

9         && movimentacao.x >= alvo.x
10        )
11        return 1;
12    else if(fabs(movimentacao.x + raio) > 100 || fabs(movimentacao.y + raio) > 100)
13        return 2;
14    else
15        return 0;
16
17 }
18
19 int main(){
20     Ponto p, alvo, partida;
21     float theta, v0, t;
22     int colisao = 1, resposta, bolinha;
23
24     AbreJanela(800, 600, "Balistica");
25     srand(time(NULL));
26     do{
27         MostraPlanoCartesiano(10);
28         PintarFundo(255, 255, 255);
29         //Predio Azul///
30         CriaGrupo(); //Separa os grupos - este e o grupo dos predios
31         if(colisao == 1) //Comeca uma nova fase
32             p.x = rand()%20 - 100; // -100 para comecar do lado esquerdo da tela
33         else //Repete fase anterior
34             p.x = partida.x - 25;
35         p.y = 0;
36         CriaRetangulo(50, -100, p);
37         Pintar(0, 0, 255);
38         /////////////
39
40         //Predio Amarelo//
41
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         DesenhaFrame();
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 }
```

(a).

Listagem E.19: 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
19 imgmagneto_standing, int imgwolverine_standing, int colisao){
20     Ponto p;
```

```

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
64         //exclui-la depois
65     p.x = 0;
66     p.y = 0;
67     p2.x = 60;
68     p2.y = 0;
69     CriaReta(p , p2); //Cria-lo na origem para nao dar problema com as coordenadas
70     //Materia de algebra linear
71
72     Pintar(0, 0, 255);
73     Grafite(10);
74     p = partida;
75
76     Move(p); //Move-lo para o lugar certo
    do{

```

```

77     DesenhaFrame();
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
102        inicial\n");
103
104    grupo_lancar = CriaGrupo();
105    p.x = 70;
106    p.y = 0; //Y como 0 para nao confundir as coordenadas
107    //Materia de algebra linear
108
109    CriaRetangulo(30, 1, p); //Altura como 1 para poder redimensiona-lo depois
110    Pintar(0, 255, 0);
111    p.x = 70;
112    p.y = -100;
113    Move(p); //Move-lo para o lugar certo
114
115    do{
116        DesenhaFrame();
117
118        if(ApertouTecla(GLFW_KEY_UP)) {
119            v0 += 10;
120            Redimensiona(1, v0);
121            printf("\nVelocidade: %f", v0);
122        }
123        else if(ApertouTecla(GLFW_KEY_DOWN)) {
124            v0 -= 10;
125            Redimensiona(1, v0);
126            printf("\nVelocidade: %f", v0);
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 comeclar uma nova fase
159         defineCenario(&alvo, &partida, &grupo_wolverine, imgbackground, imgmagneto_standing,
160                     imgwolverine_standing, colisao);
161
162         Desenha1Frame();
163
164         theta = setaAngulo(partida);
165         v0 = setaVelocidade();
166
167         ApagaGrupo(grupo_wolverine); //vamos resetar o wolverine para ataque
168
169         grupo_wolverine = CriaGrupo(); //Separa os grupos - este e o grupo da bolinha
170         CriaQuadrado(TAMANHOQUADRADO, partida);
171         Pintar(255, 255, 255);
172         AssociaImagem(imgwolverine_flying);
173
174         t = 0;
175         theta = theta * PI/180;
176         colisao = 0;
177         do{
178             if(!colisao){
179                 p.x = partida.x + v0 * cos(theta) * t;
180                 p.y = partida.y + v0 * sin(theta) * t - ((1/2.0) * (9.8) * (t*t));
181                 Move(p, grupo_wolverine);
182                 t += 0.1; //tempo
183             }
184
185             colisao = colisaoBolinhaRetangulo(TAMANHOQUADRADO, TAMANHOQUADRADO, alvo, p);
186
187             Desenha1Frame();
188         }while(!colisao);

```

```

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             DesenhalFrame();
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             DesenhalFrame();
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         DesenhalFrame();
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             DesenhalFrame();
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 }
```

### Exercício 3.3.

#### Listagem E.20: Código fonte de Snake

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <playAPC/playapc.h>
4
5 #define ESQ GLFW_KEY_LEFT
6 #define DIR GLFW_KEY_RIGHT
7 #define CIMA GLFW_KEY_UP
8 #define BAIXO GLFW_KEY_DOWN
9
10 #define TAM 50
11
12 typedef enum{
13     CABECA,
14     CORPO,
15     COMIDA,
16     VAZIO
17 }tipoCobra;
18
19 typedef struct{
20     int direcao; //se tipo nao for vazio, indica direcao
21     tipoCobra tipo; //como sera pintado
22     int index; //indice do quadrado a ser pintado
23 }tipoCelula;
24
25 void inicializaMatriz(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
26     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         }
39     }
40 }
```

```

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
149         cabeca foi
150         if(!bateu(m, npos_i, npos_j)){
151             int comeu;

```

```

151     comeu = updateCabeca(m, npos_i, npos_j, direcao);
152     if(comeu)
153         sorteiaComida(m);
154     else{
155         updateRastro(m, rpos_i, rpos_j);
156         atualizaPosicao(m[rpos_i][rpos_j].direcao, &rpos_i, &rpos_j);
157         m[rpos_i][rpos_j].tipo = CORPO;
158     }
159
160     pos_i = npos_i;
161     pos_j = npos_j;
162
163     printf("C(%d, %d)\t R(%d, %d)\n", pos_i, pos_j, rpos_i, rpos_j);
164 }
165
166 else
167     break;
168
169 aberto = DesenhalFrame();
170 if(!aberto)
171     break;
172 }
173
174 printf("O jogo acabou!\n");
175
176 if(aberto)
177     Desenha();
178
179 return 0;
180 }
```

## E.4 Recursão

### Exercício 4.1.

Listagem E.21: Árvore recursiva

```

1 #include <playAPC/playapc.h>
2
3 #define RAD 0.01745
4
5 void arvore(Ponto p1, float altural, float angulo1, float angulo_diff, float profundidade){
6     Ponto p2;
7     float altura2, angulo2;
8
9     if(0 < profundidade){
10         p2.x = p1.x + altural * cos(angulo1 * RAD);
11         p2.y = p1.y + altural * sin(angulo1 * RAD);
12         CriaReta(p1, p2);
13         Pintar(0, 255, 0);
14         Grafite(5.0);
15
16         altura2 = altural * 0.8;
```

```

17     angulo2 = angulo1 + angulo_diff;
18     arvore(p2, altura2, angulo2, angulo_diff, profundidade - 1);
19
20     angulo2 = angulo1 - angulo_diff;
21     arvore(p2, altura2, angulo2, angulo_diff, profundidade - 1);
22 }
23
24 }
25
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(800,800, "Arvore recursiva");
43     PintarFundo(255, 255, 255);
44
45     arvore(p, altura, 90, angulo, profundidade);
46
47     Desenha();
48
49     return 0;
50 }
```

## Exercício 4.2.

### Listagem E.22: Torre de Hanói

```

1 #include <stdio.h>
2 #include <playAPC/playapc.h>
3 #include <math.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 typedef struct{
8     int grupo, largura;
9     char torre;
10 }tipoDisco;
11
12 void geraDiscos(tipoDisco discos[], int numDiscos){
13     Ponto p;
14     int base, altura;
15
16     for(int i = 0; i < 5; i++){
17         discos[i].torre = '0';
18         discos[i].grupo = -1;
```

```

19         discos[i].largura = -1;
20     }
21
22     p.y = -90;
23     p.x = -100;
24
25     base = 80;
26     altura = 10;
27     for(int i = numDiscos - 1; i >= 0; i--) {
28         discos[i].torre = 'A';
29         discos[i].grupo = CriaGrupo();
30         discos[i].largura = base;
31
32         CriaRetangulo(base, altura, p);
33         Pintar(rand()%255, rand()%255, rand()%255);
34
35         base -= 16;
36         p.x += 8;
37         p.y += 10;
38
39     }
40 }
41
42 int geraFundo() {
43     Ponto p;
44     int fundo;
45
46     fundo = CriaGrupo();
47
48     p.x = -100;
49     p.y = -100;
50     CriaRetangulo(200, 10, p);
51     Pintar(185, 122, 87);
52
53     p.y = -90;
54     //Torre A
55     p.x = -66;
56     CriaRetangulo(10, 100, p);
57     Pintar(0, 128, 255);
58
59     //Torre B
60     p.x = -5;
61     CriaRetangulo(10, 100, p);
62     Pintar(255, 0, 0);
63
64     //Torre C
65     p.x = 66;
66     CriaRetangulo(10, 100, p);
67     Pintar(255, 255, 0);
68
69     return fundo;
70 }
71
72 void moveDisco(tipoDisco discos[5], char a, char b) {
73     int grupodisco, qtddiscos = 0;
74     int i = 0, index;
75     Ponto p;

```

```

76
77     while(discos[i].torre != a) {
78         i++;
79     }
80     index = i; //Se saiu do loop, entao encontramos o disco que precisaremos mover
81
82     for(i = 0; i < 5; i++) {
83         if(discos[i].torre == b)
84             qtddiscos++;
85     }
86
87     p.y = -90 + qtddiscos * 10; //altura da mesa + quantidade de discos
88
89     switch(b) {
90         case 'A':
91             p.x = -66 + 5; //mais 5 do centro da pilastra
92             break;
93         case 'B':
94             p.x = -5 + 5; //mais 5 do centro da pilastra
95             break;
96         case 'C':
97             p.x = 66 + 5; //mais 5 do centro da pilastra
98             break;
99     }
100
101    p.x -= discos[index].largura/2;
102
103    discos[index].torre = b;
104
105    Move(p, discos[index].grupo);
106
107    while(!ApertouTecla(GLFW_KEY_ENTER))
108        DesenhaFrame();
109
110 }
111
112 void hanoi(int n, char a, char b, char c, tipoDisco discos[5]){
113     /* mova n discos do pino a para o pino b usando
114     o pino c como intermediario */  

115
116     if (n == 1){
117         printf("\nmova disco %d de %c para %c\n", n, a, b);
118         moveDisco(discos, a, b);
119     }
120     else
121     {
122         hanoi(n - 1, a, c, b, discos); // H1
123         printf("\nmova disco %d de %c para %c\n", n, a, b);
124         moveDisco(discos, a, b);
125
126         hanoi(n - 1, c, b, a, discos); // H2
127     }
128 }
129
130 int main(void) {
131     int numDiscos;
132     int grupofundo;

```

```

133     tipoDisco discos[5];
134
135     srand(time(NULL));
136
137     do{
138         printf("\nDigite uma quantidade de discos menor ou igual a 5: ");
139         scanf("%d", &numDiscos);
140     }while(numDiscos > 5 || numDiscos <= 0);
141
142     AbreJanela(400, 400, "Torre de Hanoi");
143     PintarFundo(255, 255, 255);
144     MostraPlanoCartesiano(10);
145
146     grupofundo = geraFundo();
147     geraDiscos(discos, numDiscos);
148
149     while(!ApertouTecla(GLFW_KEY_ENTER))
150         DesenhaFrame();
151
152     hanoi(numDiscos, 'A', 'B', 'C', discos);
153
154     printf("\nPronto! E assim que se resolve esta torre!");
155
156     Desenha();
157
158     return 0;
159 }
```

(a).

Listagem E.23: 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 800
7 #define ALTURA 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 = 6;
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 }
```

(b).

Listagem E.24: 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 4.4.

#### Listagem E.25: 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 800
9 #define ALTURA 600
10 #define SIZE 180
11
12
13 typedef unsigned int Cardinal;
14
15 Ponto a (Cardinal k, float h, Ponto p, int R, int B, int G);
16 Ponto b (Cardinal k, float h, Ponto p, int R, int B, int G);
17 Ponto c (Cardinal k, float h, Ponto p, int R, int B, int G);
18 Ponto d (Cardinal k, float h, Ponto p, int R, int B, int G);
19
20
21 Ponto reta(Cardinal fator, Cardinal ext, Ponto p, int R, int B, int G) {
22     // ang é para multiplicar por 45 graus
23     Ponto p1;
24     double ar = fator * 45.0 * PI/180;
25
26     p1.x = p.x + ext*cos(ar);
27     p1.y = p.y + ext*sin(ar);
28     CriaReta(p,p1);
29     Grafite(2);
30     Pintar(R,G,B);
31     return p1;
32 }
33
34
35
36 Ponto a (Cardinal k, float h, Ponto p, int R, int B, int G) {
37     if ( k > 0 ) {
38         p=a(k-1,h, p, 255, 0, 0); p=reta(7,h, p, R, G, B);
39         p=b(k-1,h, p, 0, 255, 0); p=reta(0,2*h, p, R, G, B);
40         p=d(k-1,h, p, 255, 255, 0); p=reta(1,h, p, R, G, B);
41         p=a(k-1,h, p, 255, 0, 0);
42     }
43     return p;
44 }
45
46 Ponto b (Cardinal k, float h, Ponto p, int R, int B, int G) {
47     if ( k > 0 ) {
48         p=b(k-1,h, p, 0, 255, 0); p=reta(5,h, p, R, G, B);
49         p=c(k-1,h, p, 0, 0, 255); p=reta(6,2*h, p, R, G, B);
50         p=a(k-1,h, p, 255, 0, 0); p=reta(7,h, p, R, G, B);
```

```

51     p=b(k-1,h, p, 0, 255, 0);
52 }
53 return p;
54 }
55
56 Ponto c (Cardinal k, float h, Ponto p, int R, int B, int G) {
57     if ( k > 0 ) {
58         p=c(k-1,h, p, 0, 255); p=reta(3,h, p, R, G, B);
59         p=d(k-1,h, p, 255, 255, 0); p=reta(4,2*h, p, R, G, B);
60         p=b(k-1,h, p, 0, 255, 0); p=reta(5,h, p, R, G, B);
61         p=c(k-1,h, p, 0, 0, 255);
62     }
63     return p;
64 }
65
66 Ponto d (Cardinal k, float h, Ponto p, int R, int B, int G) {
67     if ( k > 0 ) {
68         p=d(k-1,h, p, 255, 255, 0); p=reta(1,h, p, R, G, B);
69         p=a(k-1,h, p, 255, 0, 0); p=reta(2,2*h, p, R, G, B);
70         p=c(k-1,h, p, 0, 0, 255); p=reta(3,h, p, R, G, B);
71         p=d(k-1,h, p, 255, 255, 0);
72     }
73     return p;
74 }
75
76
77 int main(void) {
78
79     Ponto p;
80     int i = 2; //dai pra cima a coisa fica grande e mais lenta
81     float h = 40;
82
83     p.x = -70; p.y = 100; //ordem=1 cabe todo na tela
84
85
86     MostraPlanoCartesiano(10);
87     AbreJanela(LARGURA,ALTURA, "Curvas de Sierpinski");
88     PintarFundo(255, 255, 255);
89
90     if (i>0) h /= i*i;
91     p=a(i,h, p, 255, 0, 0); p=reta(7,h, p, 0, 0, 0);
92     p=b(i,h, p, 0, 255, 0); p=reta(5,h, p, 0, 0, 0);
93     p=c(i,h, p, 0, 0, 255); p=reta(3,h, p, 0, 0, 0);
94     p=d(i,h, p, 255, 255, 0); p=reta(1,h, p, 0, 0, 0);
95
96
97     Desenha();
98
99
100    return 0;
101 }
```

## **Anexo I**

**Material sobre recursão usando PLAYAPC  
criado pelo professor Ralha**

# Recursão Revisitada

usando a **PLAYCB**

**JOSÉ CARLOS LOUREIRO RALHA**

## 1 Recursão Revisitada

Em documento anterior apresentamos alguns exemplos de problemas cuja solução foi obtida utilizando procedimentos recursivos. Esse foi o caso para problemas como

- *quicksort*
- *mergesort*
- *torres de hanói*

Neste módulo, vamos mostrar recursão através de exemplos cuja saída será na forma gráfica. Para isso vamos empregar a **PLAYCB**.



 Nossa primeiro problema será traçar uma curva famosa denominada *curva de Koch* em homenagem ao seu propositor. A curva de Koch básica é construída tomando como base um segmento de reta. Vamos fazer o traçado usando uma caneta sem tirá-la do papel. Do ponto localizado na extremidade esquerda traçamos um segmento de reta de comprimento  $n$  paralelo ao eixo  $x$ . Nesse ponto mudamos a direção do traçado; vamos agora traçar um segmento de reta de mesmo comprimento mas em um ângulo de  $60^\circ$ . Na extremidade direita do segmento traçado voltamos a mudar a direção em  $120^\circ$ . A partir daí traçamos um segmento de mesmo comprimento. Por fim, mudamos de novo o ângulo, desta vez em  $60^\circ$ , e traçamos um novo segmento. A Fig. 1 mostra o padrão descrito.<sup>1</sup> Um ponto importante diz respeito aos ângulos: *a medida é tomada sobre a reta que acabamos de traçar*.

Se repetirmos o mesmo padrão em cada segmento de reta da Fig. 1 obtemos a curva de Koch de ordem 2, exibida na Fig. 2. Esse padrão pode ser repetido indefinidamente. No limite, *quando a ordem tende a infinito*, obtemos uma *curva fractal*. A Fig. 3 mostra o caso para a ordem 5.

---

<sup>1</sup>Este tipo de traçado tem uma representação gráfica especial denominada *Turtle Graphics* e é a base da linguagem **LOGO**. A linguagem **LOGO** foi projetada como parte de uma experiência cognitiva relacionada a capacidade de aprendizagem de crianças de tenra idade.

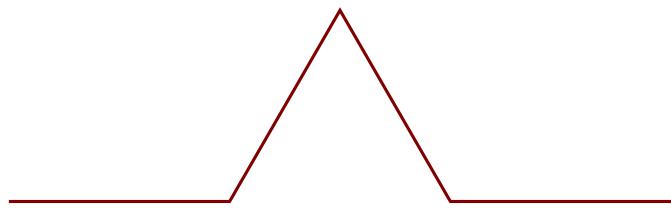


Figura 1: Curva de Koch de ordem 1.

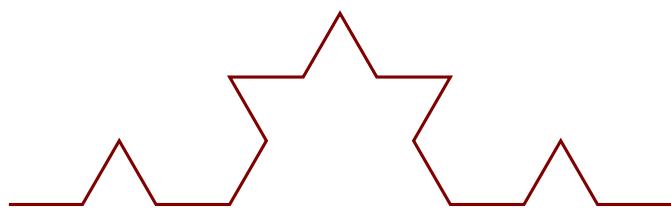


Figura 2: Curva de Koch de ordem 2.

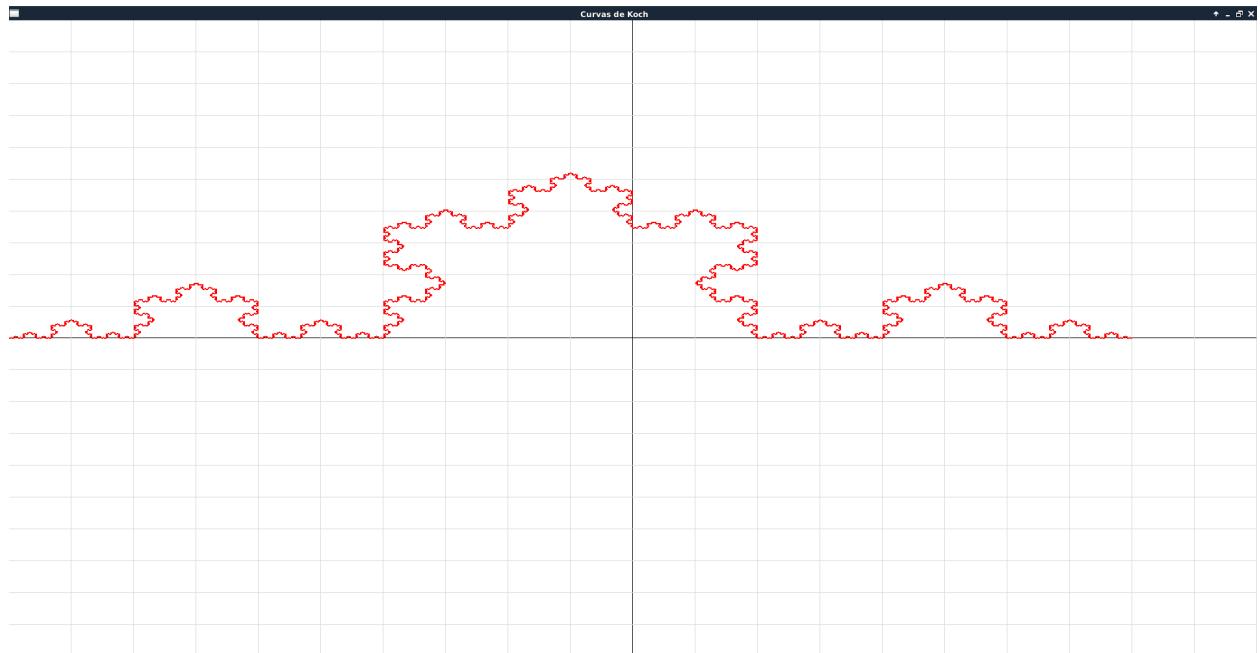


Figura 3: Curva de Koch de ordem 5. Tela produzida pelo programa `koch.c` usando a `PLAYCB`.

*A repetição da aplicação do padrão de subdivisão é naturalmente realizado usando recursão.* A Listagem 1 mostra o programa recursivo completo usando a `PLAYCB`.

Listagem 1: Programa `koch.c`

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <playcb.h>

#define LARGURA 1962//800
#define ALTURA 1280//600
#define SIZE 180

Ponto movaCaneta(Ponto from, double theta, double len) {
    Ponto to;
    double rad = theta*PI/180;
    to.x = from.x + len*cos(rad);
    to.y = from.y + len*sin(rad);
    return to;
}

void koch (Ponto from, double len, double theta, int order) {
    Ponto to;
    double rad = theta*PI/180;

    if (order==0) {
        to.x = from.x + len*cos(rad);
        to.y = from.y + len*sin(rad);
        CriaReta(from, to); Grafite(2); Pintar(255,0,0);
    }
    else {
        koch(from, len/3, theta, order-1);

        to = movaCaneta(from, theta, len/3);
        from = to;//dispensável.
        // seu uso é para tornar intuitiva a chamada abaixo
        koch(from, len/3, theta+60, order-1);

        to = movaCaneta(from, theta+60, len/3);
        from = to;
        koch(from, len/3, theta-60, order-1);

        to = movaCaneta(to, theta-60, len/3);
        from = to;
        koch(from, len/3, theta, order-1);
    }
}
```

```

int main(void) {
    int ordem=5;
    Ponto p0;

    MostraPlanoCartesiano(10);
    AbreJanela(LARGURA , ALTURA , "Curvas de Koch");
    PintarFundo(255 , 255 , 255);

    p0.x = -100; p0.y = 0.0;

    koch(p0, SIZE, 0, ordem);

    Desenha();

    return 0;
}

```



**COMENTÁRIOS** logo no início da listagem encontramos as diretivas de inclusão e entre estas a da `playcb.h` e fa `math`. A biblioteca `math` disponibiliza as funções seno—`sin`—e cosseno—`cos`— além da constante `PI`. Os `defines` caracterizam a largura, altura da janela a ser aberta. Ajuste esses valores a capacidade de sua placa gráfica. A constante `SIZE` especifica a extensão da reta base, se esta fosse desenhada em sua totalidade.

A função `movaCaneta` é a alma *Turtle Graphics* do programa. Ela movimenta a caneta de desenho sem tocar o papel. O que isso quer dizer é que a partir do Ponto `from`, do ângulo `theta` e da extensão `len` passadas como entrada, a rotina calcula o ponto de destino usando transformações lineares simples. O ponto de destino `to` é retornado como valor da função. Notem que funções podem devolver `structs`.

Vamos deixar a função `koch` para o final dado que ela é a alma do traçado.

A função `main` é bem simples. Ela define uma variável `ordem` a qual especifica a ordem da curva de Koch. Experimentem mudar os valores: quanto mais baixo mais simples a curva será—menos detalhes serão apresentados. O papel da variável `p0` é definir o ponto referencial da curva, ou seja, onde a curva começa. Usando o grid de fundo produzido pela função `PLAYCB MostraPlanoCartesiano(10)`—a qual quadricula o plano em quadrados de 10 unidades de medida—determinamos os valores iniciais para as componentes `x` e `y` do Ponto `p0`.

Chegamos agora a chamada inicial da função `koch`. Para essa primeira chamada passamos os valores

- do ponto referencial `p0`,
- a extensão `SIZE` da linha reta *virtual de suporte*<sup>2</sup> da curva,
- o ângulo—`0`—em que devemos virar a cabeça da tartaruga,

---

<sup>2</sup>Parece evidente que a extensão de uma curva fractal tende a infinito enquanto a da reta virtual suporte se mantém fixo.

- e a **ordem** da curva.

Agora é a vez de discutirmos a função recursiva **koch**.

A função recebe como valores de entrada

- um **Ponto from** que especifica o ponto incial do (sub)traçado,
- a extensão **double len** do segmento de reta a traçar,
- o ângulo **theta** em que devemos virar a cabeça da tartaruga.

A cabeça aponta para onde o segmento de reta deve ser traçado. E por analogia os ângulos podem virar a *esquerda* ou a *direita* da cabeça da tartaruga,

- a **ordem** da curva. A ordem controla o número de chamadas recursivas e consequentemente o nível de “rendilhado” da curva.

Fora as declarações das variáveis locais **Ponto to;** e **double rad = theta\*PI/180;** a função tem apenas uma única instrução, a saber: **if else**.

O **if else** controla as chamadas recursivas. Os traçados de retas só são feitos quando a **ordem** for igual a zero. Caso contrário—**ordem>0**—o ramo **else** se encarrega de realizar 4 chamadas recursivas.

O ramo “**then**” da instrução **if** quando alcançado se responsabiliza pelo desenho dos segmentos de reta. Para isso é necessário calcular o ponto de destino já que a **PLAYCB** trabalha com a função **CriaReta** entre dois pontos. Por essa razão, declaramos localmente a variável **to** a qual vai armazenar a posição do outro ponto do segmento de reta. O cálculo das coordenadas **to.x** e **to.y** são feitos usando trigonometria básica e não requerem maiores explicações. Agora que estamos de posse dos dois pontos extremos do segmento de reta podemos usar a função **PLAYCB CriaReta(from, to);**, mudar a expressura do traço **Grafite(2)**; e designar a cor do traçado **Pintar(255, 0, 0);**. É importante lembrar que a **PLAYCB** não adere ao sistema *Turtle*. Isso significa que o ponto de referência não é automaticamente mudado ao final do traçado. Essa é a razão para a existência da função **movaCaneta** no ramo **else**.

No ramo **else** encontramos 4 chamadas recursivas para a função **koch**. A ideia básica é que cada chamada se encarrega de substituir cada um dos quatro segmentos da curva por réplicas de toda a curva em uma escala de 1/3. Como devemos respeitar a direção em que a tartaruga anda, temos que atualizar **theta** de acordo. Esse é o motivo para a soma e subtração de 60° ao ângulo **theta**. Como já havia dito, a **PLAYCB**<sup>3</sup> não adere ao formato *Turtle* e por isso temos que calcular onde a caneta deveria estar ao término da execução de cada chamada recursiva. Esse é o papel das duas instruções: **to = movaCaneta...** e **from = to;**.

Agora que sabemos desenhar as curvas de Koch, fica fácil desenhar um *floco de neve*.

## 2 Floco de Neve

Para desenhar um floco de neve junta três curvas de Koch usando um triângulo como inspiração. A Listagem 2 mostra como realizar essa proeza.

---

<sup>3</sup>A **PLAYCB** usa a API **OPENGL** que por adotar uma visão mais genérica e 3D não adere ao sistema *Turtle Graphics*.

Listagem 2: Programa `snowflake.c`

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <playcb.h>

#define LARGURA 800 //1962
#define ALTURA 600 //1280
#define SIZE 180

Ponto movaCaneta(Ponto from, double theta, double len) {
    Ponto to;
    double rad = theta*PI/180;
    to.x = from.x + len*cos(rad);
    to.y = from.y + len*sin(rad);
    return to;
}

Ponto koch (Ponto from, double len, double theta, int order) {
    Ponto to;
    double rad = theta*PI/180;

    if (order==0) {
        to.x = from.x + len*cos(rad);
        to.y = from.y + len*sin(rad);

        CriaReta(from, to); Grafite(2); Pintar(255,0,0);
    }
    else {
        koch(from, len/3, theta, order-1);

        to = movaCaneta(from, theta, len/3);
        from = to;//dispensável.
        // seu uso é para tornar intuitiva a chamada abaixo
        koch(from, len/3, theta+60, order-1);

        to = movaCaneta(from, theta+60, len/3);
        from = to;
        koch(from, len/3, theta-60, order-1);

        to = movaCaneta(to, theta-60, len/3);
        from = to;
        koch(from, len/3, theta, order-1);
    }
    return movaCaneta(from, theta, len/3);
}
```

```

void floco_de_neve(int ordem) {
    Ponto p0;
    p0.x = -45.0; p0.y = 26.0; //centro do floco de neve

    p0 = koch(p0, SIZE/2, 0, ordem);
    p0 = koch(p0, SIZE/2, -120, ordem);
    p0 = koch(p0, SIZE/2, 120, ordem);

}

int main(void) {
    int ordem=6;

    MostraPlanoCartesiano(10);
    AbreJanela(LARGURA, ALTURA, "Flocos_de_Neve");
    PintarFundo(255, 255, 255);

    floco_de_neve(ordem);

    Desenha();

    return 0;
}

```

 **COMENTÁRIOS** O código é basicamente uma cópia melhorada de [koch.c](#). Note que no presente código a função `koch` tem 2 instruções e se encarrega de calcular e devolver o ponto extremo direito do traçado realizado. Isso simplifica o projeto da função `floco_de_neve`.

### 3 Curvas de Sierpiński

Sim, o nome tem um acento agudo na letra n.

Essa é uma curva mais sofisticada, do tipo *space-filling curve* que como o nome diz tendem a preencher todo o espaço disponível (e sem nunca se cortar). As figuras Fig.4 – Fig. 7 mostram as curvas de ordem 1, 2, 3 e 4 resp..

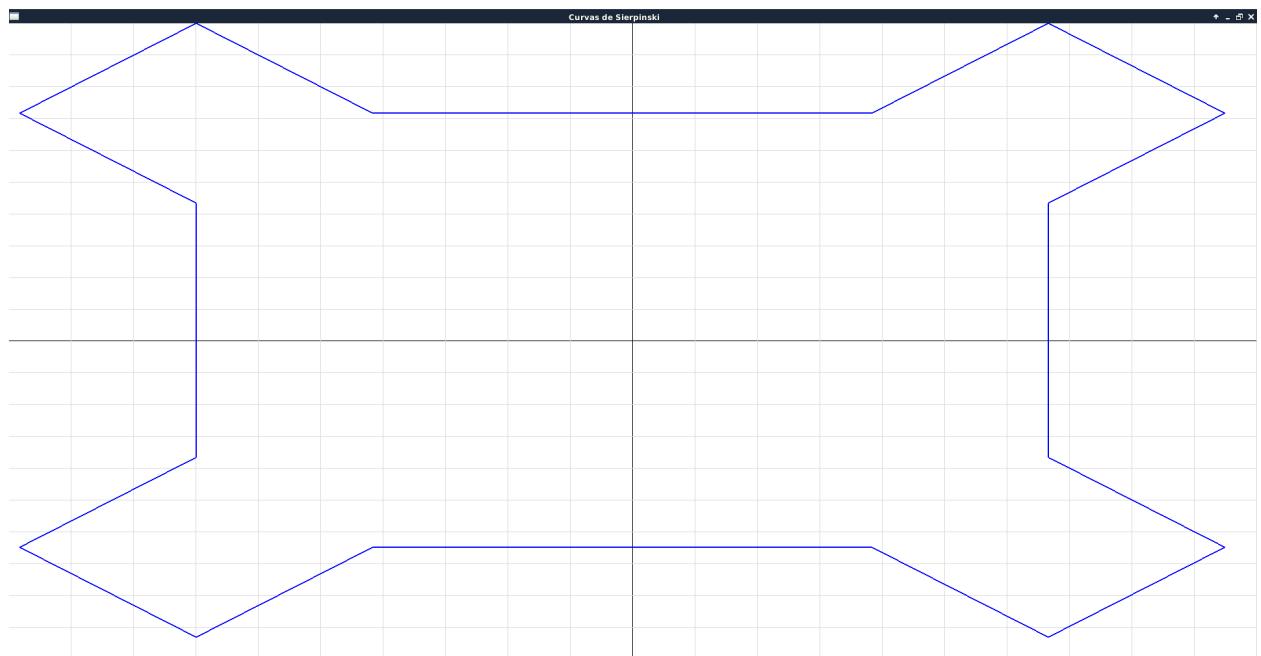


Figura 4: Curva de Sierpiński de ordem 1

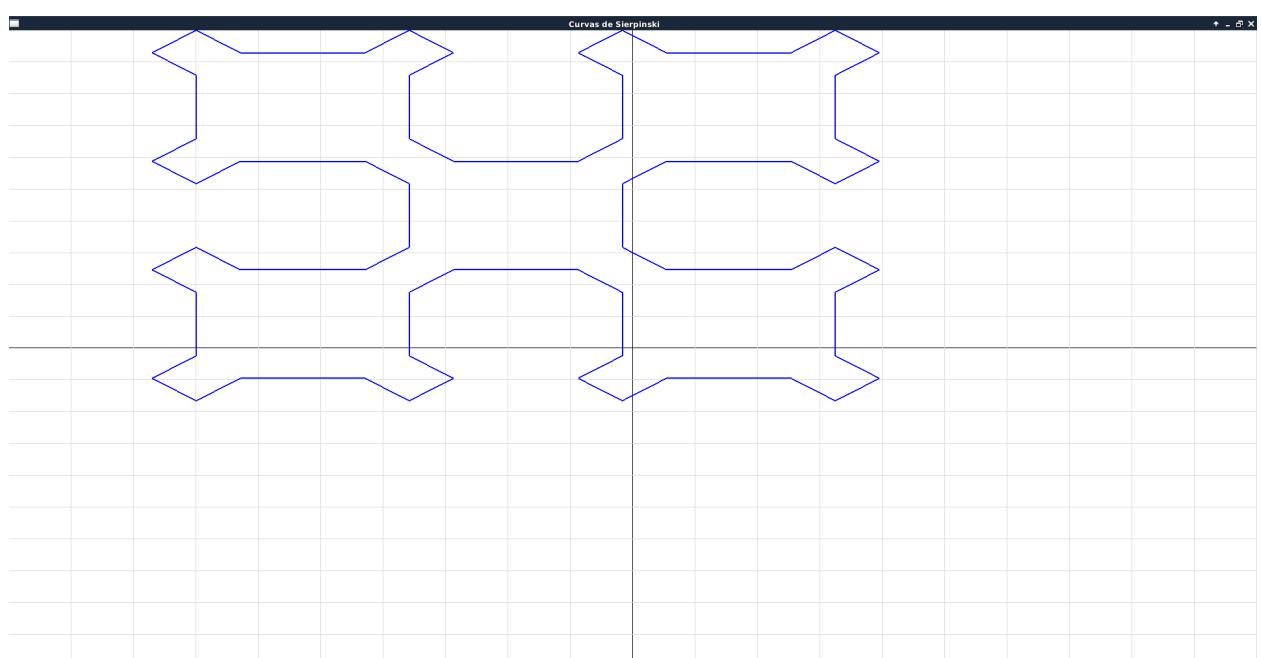


Figura 5: Curva de Sierpiński de ordem 2

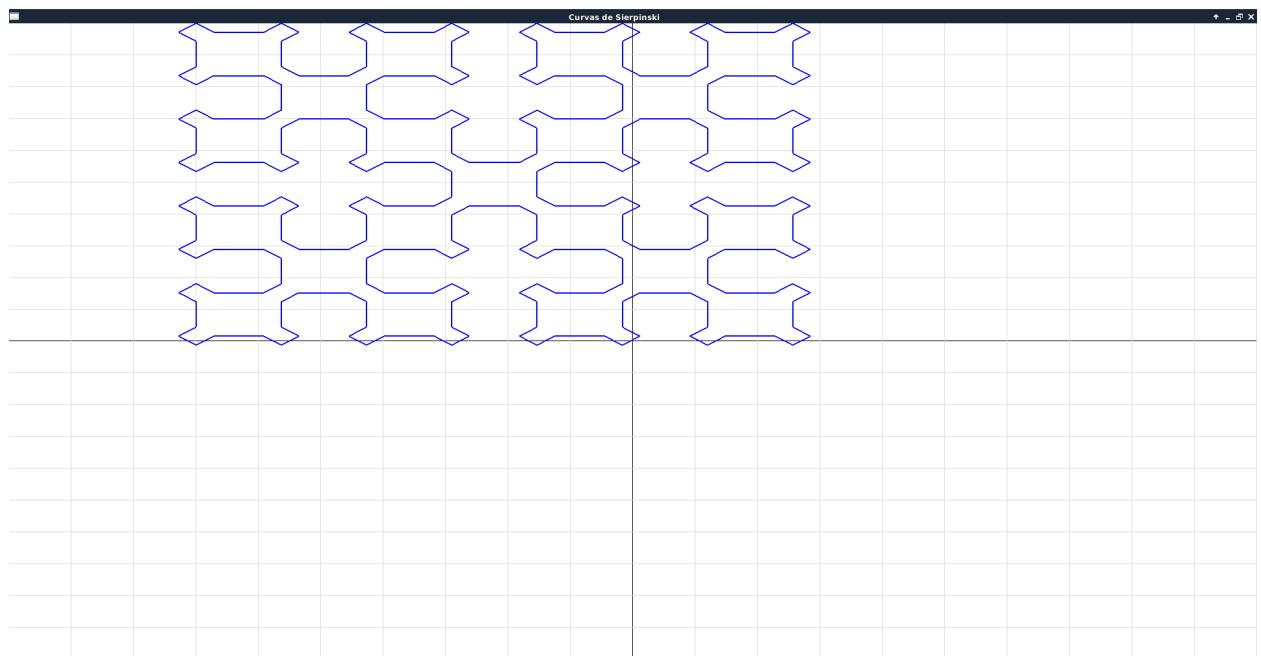


Figura 6: Curva de Sierpiński de ordem 3

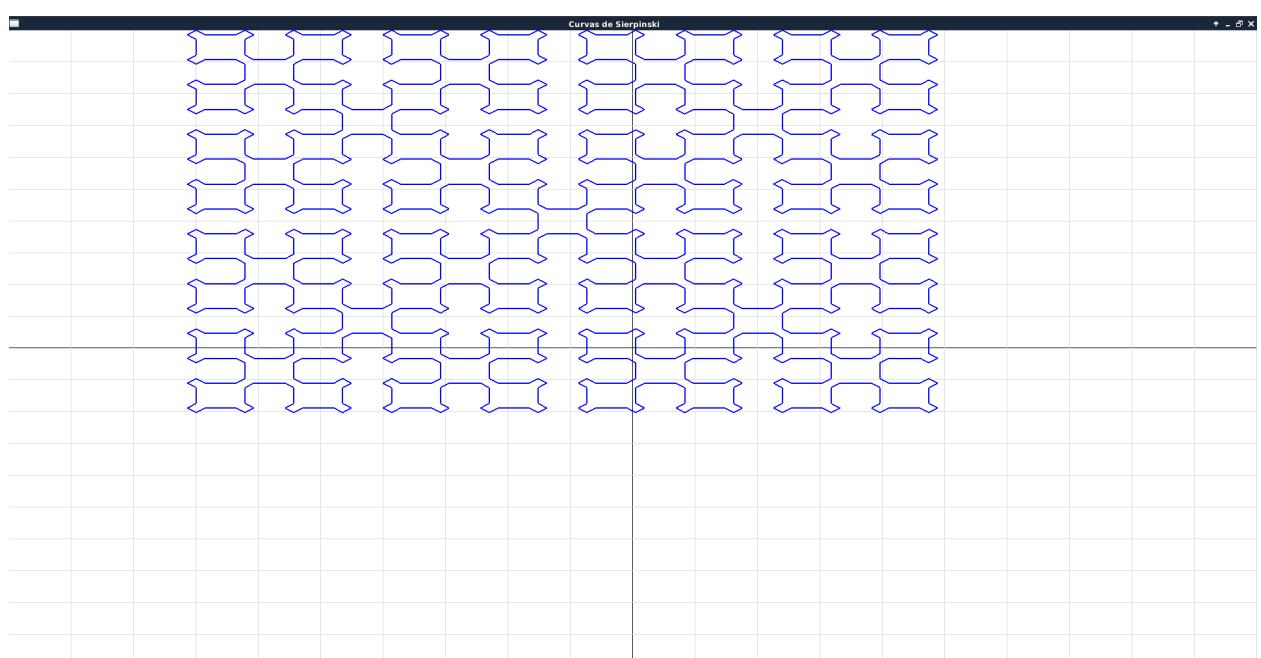


Figura 7: Curva de Sierpiński de ordem 4

Essa é uma curva que assusta a primeira vista pois ela é mutuamente recursiva entre 4 funções! Não se assuste, a coisa não é tão complicada.

Para entender o padrão recursivo vamos usar a seguinte descrição:<sup>4</sup>

- A curva básica **S** é dada pelo padrão **A ↘ B ↙ C ↖ D ↗**

As setas servem para indicar a virada de ângulo das retas que “fecham” o desenho das funções **A**, **B**, **C** e **D**.

- A curva **A** é dada pelo padrão **A ↘ B → D ↗ A**

- A curva **B** é dada pelo padrão **B ↙ C ↓ A ↘ B**

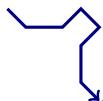
- A curva **C** é dada pelo padrão **C ↖ D ← B ↙ C**

- A curva **D** é dada pelo padrão **D ↗ A ↑ C ↖ D**

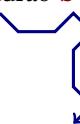
Os padrões podem ser acompanhados na Fig. 4. Vamos iniciar a observação da figura começando na parte de cima, aquela junto a identificação da janela, e seguindo um trajeto em sentido horário. A curva **A** desenha a parte de cima da figura. A curva **B** desenha as três retas na parte lateral direita. A curva **C** desenha as três retas na parte horizontal inferior da figura. A curva **D** desenha as três retas na parte vertical esquerda da figura. O padrão **S** desenha as retas que ligam essas quatro curvas abertas. O resultado disso é uma curva fechada, a curva de Sierpiński de ordem 1.

Se ainda assim você não visualizou a sequência de traçado, vamos ao mundo das tartarugas. Em um *gráfico de tartaruga* não podemos tirar o lápis do papel enquanto desenhamos. Comece com **A**; o lápis desenha uma reta que emenda na outra que emenda na outra. Qual é o resultado disso? Só pode ser . Quando **A** termina a caneta está posicionada onde desenhamos a ponta da seta. Agora olhe no padrão **S** e veja que após o término de **A** vem o traçado de uma reta ↘. De onde paramos acrescente essa reta e teremos . Agora vem a execução da rotina

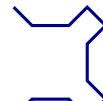
**B** a qual traça a curva começando de onde paramos. E o seu traçado é . O resultado dessa junção só pode ser



. Como acabou a execução de **B** entra em ação a ligação no padrão **S** antes da chamada de **C**; essa ligação



desenha ↙. Some essa reta a figura produzida até agora e obteremos



acrescenta ↖ ao traçado produzindo .

O resto fica como exercício mental.

O programa completo é apresentado na Listagem 3.

Listagem 3: Programa **sierpinskி.c**

```
#include <stdlib.h>
#include <stdio.h>
```

<sup>4</sup>Essa descrição está em [1].

```

#include <math.h>
#include <playcb.h>

// desenho realizado no estilo conhecido como      Turtle Graphics
#define LARGURA 1962 //800
#define ALTURA   1280 //600
#define SIZE    180

typedef unsigned int Cardinal;

Ponto A (Cardinal k, float h, Ponto p);
Ponto B (Cardinal k, float h, Ponto p);
Ponto C (Cardinal k, float h, Ponto p);
Ponto D (Cardinal k, float h, Ponto p);

Ponto reta(Cardinal fator, Cardinal ext, Ponto p) {
    // fator é para multiplicar por 45 graus
    Ponto p1;
    double ar = fator * 45.0 * PI/180; //ângulo em radianos

    p1.x = p.x + ext*cos(ar); //coordenada x do ponto de destino
    p1.y = p.y + ext*sin(ar); //coordenada y do ponto de destino
    CriaReta(p,p1);    Grafite(2);    Pintar(0,0,255);
    return p1;
}

Ponto A (Cardinal k, float h, Ponto p) {
    if ( k > 0 ) {
        p = A(k-1,h, p); p = reta(7,h, p);
        p = B(k-1,h, p); p = reta(0,2*h, p);
        p = D(k-1,h, p); p = reta(1,h, p);
        p = A(k-1,h, p);
    }
    return p;
}

Ponto B (Cardinal k, float h, Ponto p) {
    if ( k > 0 ) {
        p = B(k-1,h, p); p = reta(5,h, p);
        p = C(k-1,h, p); p = reta(6,2*h, p);
        p = A(k-1,h, p); p = reta(7,h, p);
        p = B(k-1,h, p);
    }
    return p;
}

```

```

Ponto C (Cardinal k, float h, Ponto p) {
    if ( k > 0 ) {
        p = C(k-1,h, p); p = reta(3,h, p);
        p = D(k-1,h, p); p = reta(4,2*h, p);
        p = B(k-1,h, p); p = reta(5,h, p);
        p = C(k-1,h, p);
    }
    return p;
}

Ponto D (Cardinal k, float h, Ponto p) {
    if ( k > 0 ) {
        p = D(k-1,h, p); p = reta(1,h, p);
        p = A(k-1,h, p); p = reta(2,2*h, p);
        p = C(k-1,h, p); p = reta(3,h, p);
        p = D(k-1,h, p);
    }
    return p;
}

int main(void) {

    Ponto p;
    int i = 5; //dai pra cima a coisa fica grande e mais lenta
    float h = 40;

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

    MostraPlanoCartesiano(10);
    AbreJanela(LARGURA, ALTURA, "Curvas de Sierpinski");
    PintarFundo(255, 255, 255);

    if (i>0) h /= i*i;
    p = A(i,h, p); p = reta(7,h, p);
    p = B(i,h, p); p = reta(5,h, p);
    p = C(i,h, p); p = reta(3,h, p);
    p = D(i,h, p); p = reta(1,h, p);

    Desenha();

    return 0;
}

```

No código usamos as inclinações das retas designadas por valores entre 0 e 7, inclusive. Esses são fatores multiplicativos para um ângulo base de  $45^\circ$  e são passados como o 1º argumento das funções. A extensão das retas é passada como 2º parâmetro. Quando as retas são horizontais ou verticais suas extensões são maiores que as demais.

O terceiro parâmetro corresponde ao ponto de referência inicial do traçado. Note que as funções se encarregam de devolver esse valor quando terminam suas execuções.

Hora de descansar. Que tal um joguinho de tabuleiro?

## 4 Problema das oito rainhas

Este problema pede para colocarmos de forma segura oito rainhas num tabuleiro de xadrez. Em outras palavras, nenhuma rainha pode estar sob ataque de outra.

Para os curiosos: antes de iniciar a discussão sobre a solução apresentada, este problema tem 92 soluções das quais 12 são independentes entre si. As demais soluções podem ser obtidas por operações de reflexão, simetria, etc. a partir de outra solução.

A Listagem 4 mostra um programa que apresenta *apenas uma* das muitas soluções desse problema.

Notem que o tabuleiro é representado de forma implícita através de um único vetor de tamanho 8. Observando o programa podemos ver que a estratégia usada para “percorrer” o tabuleiro em busca de uma casa segura consiste em usar uma variável que representa a coluna corrente, um vetor que representa a linha onde poderíamos colocar uma rainha. So far, so good. Mas o problema são as diagonais. Como descobrir se podemos ou não colocar uma rainha na linha—vetor **tabul**—aparentemente disponível? A resposta é dada por uma relação que envolve linha e coluna. Esses fatos são usados pela função **ta\_livre**, linhas 68–74.

O program foi originalmente escrito objetivando apresentar **TODAS** as soluções do problema. Isso se consegue com o uso de *retrocesso*—*backtracking*. A implementação apresentada na Listagem 4 mostra como podemos interromper o processo de retrocesso, linhas 90 e 91.

O tabuleiro  $8 \times 8$  é desenhado pela função **tabuleiro**, linhas 22–40. Essa função pinta cada casa do tabuleiro alternando as cores, linhas 30–35. A função **casa**, linhas 14–20, é quem pinta um quadrado na cor especificada.

O programa representa uma rainha através do desenho de um círculo azul. Isso é feito pela função **rainha**, linhas 9–12.

Listagem 4: Programa **rainhas.c**

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <playcb.h>
4
5 int tabul[8]; //armazena o índice da rainha na linha i
6 //Se tabul[5] for 3, então na linha 5 a rainha está na coluna 3
7
8 /* Início das rotinas gráficas do tabuleiro */
9 void rainha (Ponto p) {
10    CriaCirculo(4,p); //raio, ponto
11    Pintar(0,0,255);
12 }
13
```

```

14 void casa (Ponto p, int tipo) {
15     CriaRetangulo(10,10,p);
16     if (tipo == 0)
17         Pintar(0,255,0);
18     else
19         Pintar(255,255,0);
20 }
21
22 void tabuleiro (void) {
23     int i, j, cor = 1;
24
25     Ponto p;
26
27     p.x = -40;
28     p.y = -40;
29
30     for (i=0; i<8; i++) {
31         for (j=0; j<8; j++) {//pinta as 8 colunas de uma linha
32             casa(p, cor%2);
33             cor++;
34             p.x += 10;;
35         }
36         p.x = -40;
37         p.y += 10;
38         cor++; //próxima linha tem que alternar a cor inicial
39     }
40 }
41 /* Fim das Rotinas Gráficas do tabuleiro */
42
43
44 void print (){//rotina de desenho que está associada ao problema 8 rainhas
45     Ponto p;
46
47     int i,j;
48
49     p.x = -35;
50     p.y = -35;
51
52     for (i=0; i<8; i++) {
53         for (j=0; j<8; j++) {
54             if (j==tabul[i]) {
55                 if (j==0) p.x = p.x + j*10;
56                 else p.x = p.x + j*10 + 5;
57                 rainha (p);
58             }
59             p.x = -40;
60             p.y = p.y + 10;
61         }
}

```

```

62 }
63 }
64
65 //Verifica se a posição (indx,indy) está sob ataque de uma das
66 // rainhas 0...(indy-1)
67
68 int ta_livre (int indx, int indy) {
69     int i;
70
71     for (i=0; i<indy; i++)
72         if ((tabul[i]==indx) || (abs(tabul[i]-indx)==abs(i-indy))) return 0;
73     return 1;
74 }
75
76 //Tenta por a rainha n na linha i
77 void tenta (int n) {
78     static int pare = 0; //variável responsável pelo corte no retrocesso
79
80     int i;
81
82     if (n==8) {
83         pare = 1; //sinaliza que encontrou a 1ª solução
84         print(); //imprime o tabuleiro
85     }
86     else
87         for (i=0; i<8; i++)
88             if (ta_livre(i,n)) {
89                 tabul[n]=i;
90                 if (!pare) tenta (n+1);
91                 else break; //corta o backtracking. Só uma solução é impressa!
92             }
93 }
94
95
96 int main (void) {
97     Ponto p;
98
99     MostraPlanoCartesiano(10);
100    AbreJanela(800,600, "Tabuleiro de Xadrez");
101    PintarFundo(255, 255, 255);
102
103    tabuleiro();
104    tenta(0);
105
106    Desenha();
107
108    return 0;
109 }
```

A função `tenta` é responsável pela colocação das rainhas no tabuleiro. Para isso, ela se vale da função `ta_livre`. `ta_livre` verifica se a casa corrente está ou não livre. Se estiver, `ta_livre` retorna 1. Com isso, na linha 88, entramos no ramo “then” e armazenamos em `tabul` a coordenada da rainha. A próxima instrução, linha 90, verifica se já encontramos uma solução. Se esse for o caso, linha 91, quebramos o `for` da linha 87 e com isso encerramos a função `tenta`.

**Um aspecto novo para vocês é o uso da variável `pare`.**

Essa variável é **local** a função `tenta` mas é declarada como `static`. *Variáveis static tem a capacidade de guardar o valor entre chamadas da função em que foram declaradas!* Por essa razão, quando uma das chamadas recursivas de `tenta` consegue colocar a oitava rainha no tabuleiro—linha 82—atribuimos a essa variável o valor 1—linha 83. Dessa forma, as demais instâncias de `tenta` ficam sabendo que é hora de parar—linhas 90 e 91.

## 5 Torres de Hanói

Trabalho em progresso. Neste caso eu vou implementar em 3D usando a OpenGL. O resultado final será bonito mas o código será bem menos legível para iniciantes.

## Referências

- [1] Niklaus Wirth. *Algoritmos e Estruturas de Dados*. Prentice Hall do Brasil, 1989.

## **Anexo II**

### **Enunciado do Certificado de Bravura**



**Algoritmos e  
Programação de Computadores**  
**Disciplina 113476**

**Prof. Alexandre Zaghetto**  
zaghetto@unb.br

Universidade de Brasília  
Instituto de Ciências Exatas  
Departamento de Ciência da Computação

<http://www.nickgentry.com/>

**Certificado de Bravura  
Freeway**

**1. Certificado de Bravura**

Neste desafio você deve implementar uma versão do jogo de Atari **FREEWAY** de 1981.



## 1. Certificado de Bravura

Para isso, considere os elementos básicos de cenário mostrados abaixo (pista, carros, caminhões e galinha).



29/09/2016

4

## 1. Certificado de Bravura

Antes de começar, o jogo deve apresentar uma tela de abertura. Quando o usuário pressionar a tecla enter, o jogo deve ser iniciado.

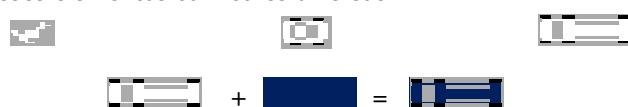


29/09/2016

5

## 1. Certificado de Bravura

Os caminhões, carros e galinha são imagens PNG com *background* transparente. Isso permite a renderização desses elementos com cores diversas.



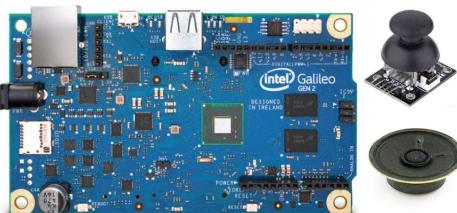
Não é obrigatória a competição entre dois jogadores. Em seu modo mínimo, o jogo deve ser jogado por um único jogador até que um limite de tempo seja alcançado. Ao final, será contada a quantidade de vezes que o jogador conseguiu atravessar a pista.

Aqui você pode encontrar um *gameplay* da versão original do jogo: <http://bit.ly/2duOgkY>

## 1. Certificado de Bravura

Sejam criativos! Você é livre para incluir características adicionais.

Você pode tentar incluir elementos sonoros e o controle do jogo utilizando o Intel Galileu Gen 2 (mais difícil) ou um telefone celular (mais fácil).



Interessados em usar o Galileu para essa finalidade, favor informar. Posso emprestar o equipamento.

## 1. Certificado de Bravura

Aqueles que tiverem interesse, podem participar da competição que irá premiar a melhor implementação do Freeway com um **Certificado de Bravura**, um **pendrive do Darth Vader** e **1.0 ponto** na média da disciplina.

O segundo receberá o **Certificado de Bravura** e **0.5 ponto** na média da disciplina.

O terceiro lugar receberá o **Certificado de Bravura**.

Os trabalhos serão avaliados pela equipe de monitores, junto com o professor da disciplina.

## 1. Certificado de Bravura

Os códigos fonte das implementações devem ser submetidos via Moodle por equipes de no máximo dois competidores.

As inscrições serão validadas apenas se junto com os códigos, os competidores enviarem um link no youtube com um *gameplay* que demonstra todas as funcionalidades implementadas. O vídeo pode conter áudio para melhor explicação da implementação.

## **Anexo III**

### **Ementas dos cursos iniciais de computação**

Quick Links: [FAQs](#) | [Meeting Times](#) | [Labs \(labs.html\)](#) | [Textbook](#) | [Grading Policy](#) | [Academic Integrity](#) | [Students with Disabilities](#)

---

## Course Overview

This is a course in fundamental computing principles for students with little to no computing background. It covers the following topics:

- Programming constructs involving sequencing, selection, iteration, and recursion.
- Data organization such as arrays and lists.
- Use of abstraction in computing for data representation, computer organization, computer networks, functional decomposition, and application programming interfaces.
- Use of computational principles in problem-solving like divide and conquer, randomness, and concurrency.
- Classification of computational problems based on complexity, non-computable functions, and using heuristics to find reasonable solutions to complex problems.
- Social, ethical and legal issues associated with the development of new computational artifacts will also be discussed.

## Meeting Times

Lecture 1 MWF 2:30 - 3:20, WEH 7500, Margaret Reid-Miller.

Lecture 2 MWF 3:30 - 4:20, WEH 7500, Margaret Reid-Miller.

Labs (also known as recitations), held by course TAs in GHC 5208 and GHC 5210.

You are **required** to go to your assigned lecture and lab. Since part of your course grade depends on lab participation, you must go to your assigned section to get lab credit.

## Textbook

There is **no required textbook** for this course. We will provide lecture slides and draft lecture notes as needed. We will additionally assign some readings from the book *Blown To Bits: Your Life, Liberty, and Happiness after the Digital Explosion* by Hal Abelson, Ken Ledeen, and Harry Lewis. Publisher: Addison-Wesley, 2008 (ISBN: 978-0137135592). This book is available **for free** from [bitsbook.com](http://www.bitsbook.com/) (<http://www.bitsbook.com/>). You can buy a hard copy of this book if you wish from any major bookseller.

The following textbooks are recommended as supplementary resources:

- *Explorations in Computing: An Introduction to Computer Science and Python Programming* by John Conery. September 2014 edition. Publisher: Chapman and Hall/CRC. This book is available as an e-book or e-book rental (ISBN 9781498718349) and as a hardback (ISBN 9781466572447).
- *Introduction to Computation and Programming Using Python* by John V. Guttag. August 2013 edition. Publisher: MIT Press. This book is available as an e-book (ISBN 9780262316644) and as a paperback (ISBN 9780262525008).

## Assignments

There are two types of assignments you will complete in this course: problems sets (PS) and programming assignments (PA). Problem sets are written assignments that help you test your understanding of conceptual parts in this course. Programming assignments help you test your programming skills or use of other online tools presented in class. Problem sets are handed in in class and programming assignments are handed in using the tool Autolab (<https://milkshark.ics.cs.cmu.edu>). You will be assigned about 11 problem sets and 9-10 programming assignments throughout the semester, *but* these numbers are subject to change.

## Grading Policy

**All assignments must be handed in on time** (unless you are given instructions otherwise). **Late or missing work will receive zero credit.** The reason this is done is so that we can get feedback to you as quickly as possible so you can learn from your mistakes and prepare for the exams. Additionally, it allows us to post a sample solution as soon as possible for the benefit of all students. **We will drop 1 written assignment and 1 programming assignment with the lowest grade (except where noted).** You are required to provide documentation if you could not submit an assignment due to a legitimate reason (e.g. major illness, death in immediate family, university-sanctioned event with verification from advisor or coach, etc.). You must take all exams (written and lab exams) at the times they are given. There will be **NO MAKEUPS FOR EXAMS** allowed except for acceptable documented circumstances such as the ones listed above.

Your course grade will be calculated based on the following:

Homework Assignments: 30%

Lab Participation: 5%

Two Lab Exams: 10% (5% each)

Two Written Exams: 30% (15% each)

Final Exam: 25%

Grades from all assignments and exams may be reviewed for up to 5 days after they are returned and posted. After this period, the grade is considered final and cannot be changed. We reserve the right to review an entire assignment or exam if it is submitted for re-grading.

We use Autolab (<https://milkshark.ics.cs.cmu.edu>) for releasing all of your grades and we expect you to use it to keep track of your grade status.

## Discussion Forum

We will use Piazza (<https://piazza.com>) for course announcements and online discussions.

## Academic Integrity

The value of your degree depends on the academic integrity of yourself and your peers in each of your classes. Please read the University Policy on Academic Integrity (<http://www.cmu.edu/policies/documents/Academic%20Integrity.htm>) carefully to understand the penalties associated with academic dishonesty at Carnegie Mellon University.

*Academic integrity* means that any work you submit for this course is your own. This is critical to your learning. The policy's intention is that you never hand in something you don't understand. Your understanding must be deep enough that, if necessary, you could re-do the work completely on your own. In short, do your own work.

We want you to collaborate with other students *only if the collaboration improves your understanding*. Therefore, you can talk about the assignments, but no one may take notes or record the discussion. When you write your solution, it should be *yours*. Go to a separate area and write your own code or answers. Do

this *individually* so that you don't end up copying someone else's work. Your own solution, even if it is *incorrect*, is much better than someone else's that you don't understand.

When working on programming assignments, do not look at other students' code or show them your own. If you need that kind of help, get it from the course staff. You may discuss your code at a conceptual level; for example, "do we need a loop for this purpose or just an if statement?". You may collaborate on code at a whiteboard, but you may not take notes or photographs; the purpose of the collaboration is to develop your understanding so that you can then solve the problem yourself, on your own.

If the course staff sees similarities between your work and that of another student, we will attempt to understand what happened. Usually this involves asking you to explain your work and how you did it, and to re-create the work or solve a related problem during our meeting.

For exams, your work must be your own with no communication between you and others (except course staff), and you may use only authorized materials.

Often students have trouble keeping up with the workload due to personal issues. If this happens to you, your best action is to see your instructors. We can help you work toward a solution and will be happy to assist.

In this class, *cheating, copying, or plagiarism* means copying all or part of a program or homework solution from another student or unauthorized source, or knowingly giving such information to another student, or handing in a copy of work that you and another student did together, or giving or receiving unauthorized information during an examination. If you use information from another authoritative resource, you must cite the source of this information (and receive permission if required).

Students who violate this policy will be charged with academic dishonesty that can result in failure in this course and possible expulsion from Carnegie Mellon University. Review the official University Code for more information.

Every student is required to sign and return the Academic Integrity Form ([other/academic\\_integrity\\_form.pdf](#)) within the first week of classes.

## Students with Disabilities

Individuals with documented disabilities may be eligible to receive services and accommodations from CMU's Equal Opportunity Services (EOS) office. For more information, please contact Larry Powell, Manager of Disability Services at (412) 268-2013 (voice/TTY).

## Frequently Asked Questions

- **I have *questions about my assignment that is due soon*. How do I get help?**

You can go to one of the office hours to receive help from the TAs or the instructors. You can also post your question to Piazza (<https://piazza.com/class#fall2016/15110>) to get an online response.

- **How can I *find my grades*?**

All your grades are available on Autolab (<https://autolab.andrew.cmu.edu/courses/15110-f16/assessments>) upon being graded. We make every effort to make your grades available within a week of their due date.

- **Some of my grades are missing on Autolab. What do I need to do?**

First, contact your TA ([labs.html](#)) and ask why all or some grades are missing. Next, if the issue is not resolved within a day or two, send an e-mail to the instructors (copy your TA) detailing which grades are missing and your lab section.

- **Can I *switch my recitation section*?**

Yes, as long as we have enough space in the new recitation. You must make an official change of recitations through SIO.

- **I must be out of town for university related event (e.g. member of a team). What should I do about my assignments?**

If you have an *official* excuse we will make special arrangements for you to submit the assignment; please contact the instructors.

- **I missed one of the lab sessions. Can I still submit my lab assignment late?**

No. However, you can drop two labs and we will count this as one of them.

- **I am out of town attending a family/important event . How can I submit my assignments due for the week?**

The programming assignment must be submitted online using Autolab if the submission link is active. Otherwise, you should alert the instructors and your TA that the link is not active and that you need to submit your work. The written assignment must be scanned and sent as an attachment to your TA before the due date. However, we can only allow you to do this one time during the semester.

- **I missed the in class exam because I fell sick. What should I do?**

You must immediately seek medical treatment and receive an official medical excuse. You must also contact the instructors *prior* to the exam or as soon as possible. In this case, once we see the excuse, we can make arrangements to give you a makeup test. Otherwise, we will be unable to make any exceptions.

- **I need to discuss my performance in the course with someone. What should I do?**

You must immediately contact one of the instructors of the course. They will be able to assist you in dealing with the situation.

- **What is the best way to prepare for an in class exam?**

If you are attending lectures and doing homeworks, you are well prepared. All you need to do is to review all lectures and class assignments. We also regularly offer help sessions before the exam. Plan to attend one of them.

- **I am failing the course. Is there any extra work I can do to get a passing grade?**

Unfortunately, in a large class like ours, we cannot make exceptions. The best way to avoid this situation is to talk to one of the instructors as soon as possible to find out what you need to do. Do not wait until the last few weeks of classes to discuss your performance.

- **I want to add this course. Is it possible to do it?**

You can only add a course during the first two weeks of classes. We do not accept any new students after the second week. However, you are welcome to audit the course, provided we have enough space. Please consult an instructor.

# 120\_1

---

## CO120.1-Programming I

### Course Aims

#### Academic aims

This is the students' first programming course. It aims to introduce some of the fundamentals of programming for beginners using a strongly-typed functional programming language (Haskell) that most students will have little or no experience of. The emphasis is on writing succinct, beautiful code, without being bogged down in the syntax and semantics of a conventional procedural or object-oriented language. The course is taught using a problem-solving approach with students being encouraged to use the various language features to solve well-specified problems independently and explore some fundamental algorithms and data structures in computer science.

### Learning Outcomes

Students will be able to:

- \* Use static types as a partial specification of what a function is intended to do.
- \* Use type error messages as an aid to debugging.
- \* Devise recursive solutions to problems that involve iteration.
- \* Reason about the complexity of basic recursive functions and functions defined over linear and tree-like data structures.
- \* Design and use test suites for functional testing.
- \* Use the knowledge and experience gained to develop succinct and efficient solutions to unseen, but well-specified, problems of small to medium scale.

### Course syllabus

#### Syllabus

Expressions, basic types, product types, arithmetic sequences, list comprehensions, function types and user-defined function definitions, recursion, polymorphism, list processing, enumerated types, higher-order functions, algebraic data types, type classes and overloading.

### Pre-requisites

None, other than basic pre-university mathematics.

## Teaching methods

Weekly lectures, catch-up tutorials, small-group tutorials, timetabled laboratory sessions, supervised catch-up laboratory sessions.

There is also a series of optional lectures on Advanced Programming in Haskell.

## Assessments

There is an unassessed practice test (formative assessment only), a 'driving test' (20%) and a final 'main test' (80%), all of which are taken in the laboratory under exam conditions using the Lexis test administration system.

Students can also undertake independent self assessment through unassessed exercises, for which model answers are made available.

## Reading list

S. Thompson, "Haskell: The Craft of Functional Programming" (Third Edition), Addison Wesley, 2011.

P. Hudak, "The Haskell School of Expression", Cambridge University Press, 2000.

R. Bird, "Introduction to Functional Programming using Haskell", Prentice Hall, 1988.

S. S. Skiena and M. A. Revilla, "Programming Challenges -- the Programming Contest Training Manual", Springer, 2003.

B. O'Sullivan, J. and D. B. Stewart, "Real World Haskell", O'Reilly Media, 2008.

The Haskell Wiki: <http://haskell.org/>

## Course leaders

Dr Anthony Field

---

### Imperial College London

South Kensington Campus  
London SW7 2AZ, UK  
tel: +44 (0)20 7589 5111

[Campuses & maps >](#)

© 2016 Imperial College  
London

[Skip to Main Content](#)

# Course Description

## CIS\*1500 Introduction to Programming F,W (3-2) [0.50]

---

Introductory problem-solving, programming and data organization techniques required for applications using a general purpose programming language. Topics include control structures, data representation and manipulation, program logic, development and testing. For students who require a good understanding of programming or are planning on taking additional specialist Computing and Information Science courses. This is the entry point to all CIS programs.

*Offering(s):* Also offered through Distance Education format.

*Restriction(s):* [CIS\\*1650](#)

*Department(s):* School of Computer Science

# Handbook

---

## COMP10001 Foundations of Computing

On this page:

- [Subject Overview](#) - #overviewId
- [Breadth options](#) - #breadthId
- [Related Program\(s\)](#) - #relatedDocumentId

[Print](#) - /view/2016/COMP10001?output=PDF

**Credit Points:** 12.5

**Level:** 1 (Undergraduate)

This subject has the following teaching availabilities in 2016:

Semester 1, Parkville - Taught on campus.[Show/hide details](#) - #  
Pre-teaching Period Start not applicable  
Teaching Period 29-Feb-2016 to 29-May-2016  
Assessment Period End 24-Jun-2016  
Last date to Self-Enrol 11-Mar-2016  
Census Date 31-Mar-2016  
Last date to Withdraw without fail 06-May-2016

**Dates & Locations:**

Semester 2, Parkville - Taught on campus.[Show/hide details](#) - #  
Pre-teaching Period Start not applicable  
Teaching Period 25-Jul-2016 to 23-Oct-2016  
Assessment Period End 18-Nov-2016  
Last date to Self-Enrol 05-Aug-2016  
Census Date 31-Aug-2016  
Last date to Withdraw without fail 23-Sep-2016

Timetable can be viewed [here](https://sws.unimelb.edu.au/2016/Reports>List.aspx?objects=COMP10001&weeks=1-52&days=1-7&periods=1-56&template=module_by_group_list) - https://sws.unimelb.edu.au/2016/Reports>List.aspx?objects=COMP10001&weeks=1-52&days=1-7&periods=1-56&template=module\_by\_group\_list.

For information about these dates, click [here](http://ask.unimelb.edu.au/app/answers/detail/a_id/6032) - http://ask.unimelb.edu.au/app/answers/detail/a\_id/6032.

Contact Hours: 60 hours, comprised of three 1-hour lectures and one 2-hour workshop per week

Total Time Commitment:

**Time Commitment:** 170 hours

**Prerequisites:** None

**Corequisites:** None

**Recommended**

**Background** None

**Knowledge:**

Subject

[INFO10001 Informatics 1: Data on the Web](#) - /view/2016/INFO10001

## INFO10001 Informatics-1:Practical Computing (prior to 2011)

<b>Non Allowed Subjects:</b>	615-145 Concepts of Software Development 1 433-151 Introduction to Programming (Advanced) 433-171 Introduction to Programming 600-151 Informatics-1: Practical Computing
------------------------------	---

For the purposes of considering request for Reasonable Adjustments under the Disability Standards for Education (Cwth 2005), and Student Support and Engagement Policy, academic requirements for this subject are articulated in the Subject Overview, Learning Outcomes, Assessment and Generic Skills sections of this entry.

<b>Core Participation Requirements:</b>	It is University policy to take all reasonable steps to minimise the impact of disability upon academic study, and reasonable adjustments will be made to enhance a student's participation in the University's programs. Students who feel their disability may impact on meeting the requirements of this subject are encouraged to discuss this matter with a Faculty Student Adviser and Student Equity and Disability Support: <a href="http://services.unimelb.edu.au/disability">http://services.unimelb.edu.au/disability</a> - <a href="http://services.unimelb.edu.au/disability">http://services.unimelb.edu.au/disability</a>
---	---

**Coordinator**

Prof Christopher Leckie, Prof Tim Baldwin

**Contact**

Semester 1: Professor Tim Baldwin

email: [tbaldwin@unimelb.edu.au](mailto:tbaldwin@unimelb.edu.au) - <mailto:tbaldwin@unimelb.edu.au>

Semester 2: A/Prof Chris Leckie

email: [cleckie@unimelb.edu.au](mailto:cleckie@unimelb.edu.au) - <mailto:bjpope@unimelb.edu.au>

<b>Subject Overview:</b>	<b>AIMS</b>  Solving problems in areas such as business, biology, physics, chemistry, engineering, humanities, and social sciences often requires manipulating, analysing, and visualising data through computer programming. This subject teaches students with little or no background in computer programming how to design and write basic programs using a high-level procedural programming language, and to solve simple problems using these skills.  This subject is the first subject in the Computing & Software Systems and the Informatics majors, and introduces students to programming and the basics of algorithmic thinking.
	<b>INDICATIVE CONTENT</b>  Fundamental programming constructs; fundamental data structures; abstraction; basic program structures; algorithmic problem solving, testing and debugging; introduction to the Web, multimedia and visualisation.  Examples of projects that students complete are: <ul style="list-style-type: none"><li>• A text analytics “library” consisting of a series of independent functions to calculate/extract different things given a document/document collection as input</li><li>• A video recommender system, broken down into a series of functions</li><li>• An AI player for an online card game, designed such that students play off against each other (and against the class) at the end of semester.</li></ul>

<b>INTENDED LEARNING OUTCOMES (ILO)</b>	
<b>Learning Outcomes:</b>	<p>On completion of this subject the student is expected to:</p> <ol style="list-style-type: none"> <li>1. Use the fundamental programming constructs (sequence, alternation, selection)</li> <li>2. Use the fundamental data structures (arrays, records, lists, associative arrays)</li> <li>3. Use abstraction constructs such as functions</li> <li>4. Understand and employ some basic program structures</li> <li>5. Understand and employ some basic algorithmic problem solving techniques</li> <li>6. Read, write, and debug simple, small programs</li> </ol>
<b>Assessment:</b>	<ul style="list-style-type: none"> <li>• A three-stage project, requiring approximately 30 - 35 hours of work, with stages due at the end of each third of the semester - approximately weeks 4, 8, and 12 (30%)</li> <li>• One 1-hour mid-semester test (10%)</li> <li>• A workshop assignment to demonstrate programming competency, due two thirds of the way through semester (10%), requiring approximately 10 - 13 hours of work per student</li> <li>• One 2-hour end-of-semester examination (50%).</li> </ul> <p><b>Hurdle requirement:</b> To pass the subject, students must obtain at least:</p> <ul style="list-style-type: none"> <li>• 50% overall, 20/40 for the project and assignment work</li> <li>• And 30/60 for the mid-semester test and end-of-semester written examination combined.</li> </ul> <p>Intended Learning Outcomes (ILOs) 1-6 are addressed in the projects, the mid-semester test, and the workshop assignment and the final exam.</p>
<b>Prescribed Texts:</b>	None
<b>Breadth Options:</b>	<p>This subject potentially can be taken as a breadth subject component for the following courses:</p> <ul style="list-style-type: none"> <li>• Bachelor of Arts - <a href="https://handbook.unimelb.edu.au/view/2016/B-ARTS">https://handbook.unimelb.edu.au/view/2016/B-ARTS</a></li> <li>• Bachelor of Commerce - <a href="https://handbook.unimelb.edu.au/view/2016/B-COM">https://handbook.unimelb.edu.au/view/2016/B-COM</a></li> <li>• Bachelor of Environments - <a href="https://handbook.unimelb.edu.au/view/2016/B-ENVS">https://handbook.unimelb.edu.au/view/2016/B-ENVS</a></li> <li>• Bachelor of Music - <a href="https://handbook.unimelb.edu.au/view/2016/B-MUS">https://handbook.unimelb.edu.au/view/2016/B-MUS</a></li> </ul> <p>You should visit <b>learn more about breadth subjects</b> - <a href="http://breadth.unimelb.edu.au/breadth/info/index.html">http://breadth.unimelb.edu.au/breadth/info/index.html</a> and read the breadth requirements for your degree, and should discuss your choice with your student adviser, before deciding on your subjects.</p>
<b>Fees Information:</b>	<a href="http://enrolment.unimelb.edu.au/fees">Subject EFTSL, Level, Discipline &amp; Census Date</a> - <a href="http://enrolment.unimelb.edu.au/fees">http://enrolment.unimelb.edu.au/fees</a>
<b>Generic Skills:</b>	<p>On completion of this subject, students should have developed the following generic skills:</p> <ul style="list-style-type: none"> <li>• An ability to apply knowledge of basic science and engineering fundamentals</li> <li>• An ability to undertake problem identification, formulation and solution</li> <li>• The capacity to solve problems, including the collection and evaluation of information</li> <li>• The capacity for critical and independent thought and reflection</li> <li>• An expectation of the need to undertake lifelong learning, and the capacity to do so.</li> </ul>
<b>LEARNING AND TEACHING METHODS</b>	
The subject is delivered through a combination of lectures and workshops (combination of tutorial and individual/group work in a computer lab). Students get	

	<p>a hands-on introduction to Python through a series of online worksheets with embedded programming tasks/automatic assessment, and then go on to complete three projects.</p>
<b>INDICATIVE KEY LEARNING RESOURCES</b>	
<b>Notes:</b>	Students have access to lecture notes, lecture slides, tutorial worksheets, which houses the interactive worksheets as well as a programming environment. The subject LMS site also contains links to recommended resources relating to basic programming, and advanced problems for students who want to extend themselves.
<b>CAREERS / INDUSTRY LINKS</b>	
	As an introductory programming subject, this is relevant to all aspects of the IT industry. Exemplar companies/organisations which have been involved in the delivery of the subject (through guest lectures etc.) are: Palantir Technologies (software engineering, intelligent systems), AURIN (Australian Urban Research Infrastructure Network: geomatics, distributed computing, web development), VLSCI (Victorian Life Sciences Computing Initiative; computational biology, bioinformatics, distributed computing, big data). There have also been guest lecturers from within the university in fields including computational ophthalmology, electronic voting, and social media analysis.
<b>Related Course(s):</b>	<a href="#">Bachelor of Biomedicine - /view/2016/B-BMED</a> <a href="#">Diploma in Informatics - /view/2016/D-INFO</a>
<b>Related Majors/Minors/Specialisations:</b>	<a href="#">Science-credited subjects - new generation B-SCI and B-ENG. - /view/2016/%21B-SCI-SPC%2B1021</a> <a href="#">Selective subjects for B-BMED - /view/2016/%21B-BMED-SPC%2B1000</a>

Date created: 11 October 2007 Last modified: 22 January 2009 Authoriser: Vice-Principal and Academic Registrar, Office of the Vice-Principal and Academic Registrar Maintainer: [Handbook Team](#), Applications Management, Academic Services Applications Development Help: [for Future Students](#) [for Current Students](#)

# Michaelmas Term 2016: Part IA lectures

## Paper 1: Foundations of Computer Science

*Lecturer: Professor L.C. Paulson*

*No. of lectures and practicals:* 12 + 5 (NST and PBST students will take 4 practicals)

*Suggested hours of supervisions:* 3 to 4

*This course is a prerequisite for Programming in Java and Prolog (Part IB).*

### Aims

The main aim of this course is to present the basic principles of programming. As the introductory course of the Computer Science Tripos, it caters for students from all backgrounds. To those who have had no programming experience, it will be comprehensible; to those experienced in languages such as C, it will attempt to correct any bad habits that they have learnt.

A further aim is to introduce the principles of data structures and algorithms. The course will emphasise the algorithmic side of programming, focusing on problem-solving rather than on hardware-level bits and bytes. Accordingly it will present basic algorithms for sorting, searching, etc., and discuss their efficiency using  $O$ -notation. Worked examples (such as polynomial arithmetic) will demonstrate how algorithmic ideas can be used to build efficient applications.

The course will use a functional language (ML). ML is particularly appropriate for inexperienced programmers, since a faulty program cannot crash. The course will present the elements of functional programming, such as curried and higher-order functions. But it will also introduce traditional (procedural) programming, such as assignments, arrays and references.

### Lectures

- **Introduction to Programming.** The role of abstraction and representation. Introduction to integer and floating-point arithmetic. Declaring functions. Decisions and booleans. Example: integer exponentiation.
- **Recursion and Efficiency.** Examples: Exponentiation and summing integers. Overloading. Iteration *versus* recursion. Examples of growth rates. Dominance and  $O$ -Notation. The costs of some representative functions. Cost estimation.
- **Lists.** Basic list operations. Append. Naïve *versus* efficient functions for length and reverse. Strings.
- **More on lists.** The utilities take and drop. Pattern-matching: zip, unzip. A word on polymorphism. The “making change” example.
- **Sorting.** A random number generator. Insertion sort, mergesort, quicksort. Their efficiency.

- **Datatypes and trees.** Pattern-matching and case expressions. Exceptions. Binary tree traversal (conversion to lists): preorder, inorder, postorder.
- **Dictionaries and functional arrays.** Functional arrays. Dictionaries: association lists (*slow*) *versus* binary search trees. Problems with unbalanced trees.
- **Functions as values.** Nameless functions. Currying. The “apply to all” functional map. *Examples:* matrix transpose and product. The predicate functionals filter and exists.
- **Sequences, or lazy lists.** Non-strict functions such as *IF*. Call-by-need *versus* call-by-name. Lazy lists. Their implementation in ML. Applications, for example Newton-Raphson square roots.
- **Queues and search strategies.** Depth-first search and its limitations. Breadth-first search (BFS). Implementing BFS using lists. An efficient representation of queues. Importance of efficient data representation.
- **Polynomial arithmetic.** Addition, multiplication of polynomials using ideas from sorting, etc.
- **Elements of procedural programming.** Address *versus* contents. Assignment *versus* binding. Own variables. Arrays, mutable or not. Introduction to linked lists.

## Objectives

At the end of the course, students should

- be able to write simple ML programs;
- understand the fundamentals of using a data structure to represent some mathematical abstraction;
- be able to estimate the efficiency of simple algorithms, using the notions of average-case, worse-case and amortised costs;
- know the comparative advantages of insertion sort, quick sort and merge sort;
- understand binary search and binary search trees;
- know how to use currying and higher-order functions;
- understand how ML combines imperative and functional programming in a single language.

## Recommended reading

\* Paulson, L.C. (1996). *ML for the working programmer*. Cambridge University Press (2nd ed.).

Okasaki, C. (1998). *Purely functional data structures*. Cambridge University Press.

For reference only:

Gansner, E.R. & Reppy, J.H. (2004). *The Standard ML Basis Library*. Cambridge University Press. ISBN: 0521794781

---

## Paper 1: Object-Oriented Programming

*Lecturer: Dr R.K. Harle, Dr A.C. Rice and Dr S. Cummins*

*No. of lectures and practicals: 12 + 5*

*Suggested hours of supervisions: 3 to 4*

## Aims

The goal of this course is to provide students with the ability to write programs in Java and make use of the concepts of Object-Oriented Programming. Examples and discussions will use Java primarily, but other languages may be used to illustrate specific points where appropriate. The course is designed to accommodate students with diverse programming backgrounds; it is taught with a mixture of lectures and practical sessions where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

## Lecture syllabus

- **Types, Objects and Classes** Moving from functional to imperative. Distinguishing state and behaviour. Primitive types. Function prototypes. Objects and classes as custom types. Introduction to parameterised types (templates/Generics).
- **Pointers, References and Memory** Pointers and references. The call stack and heap. Iteration and recursion. Pass-by-value and pass-by-reference. Objects as reference types in Java.
- **Creating Classes** Modularity. Encapsulation. Information hiding. Access modifiers. Advantages of immutability. Creating Generic types in Java. Static data.
- **Inheritance** Inheritance. Casting. Shadowing. Overloading. Overriding. Abstract Methods and Classes.
- **Polymorphism and Multiple Inheritance** Polymorphism in ML and Java. Multiple inheritance. Interfaces in Java.



## Department of **COMPUTER SCIENCE** (/)

Menu Search

# COS 126: General Computer Science: An Interdisciplinary Approach

An introduction to computer science in the context of scientific, engineering, and commercial applications. The course will teach basic principles and practical issues, and will prepare students to use computers effectively for applications in computer science, physics, biology, chemistry, engineering, and other disciplines. Topics include: hardware and software systems; programming in Java; algorithms and data structures; fundamental principles of computation; and scientific computing, including simulation, optimization, and data analysis. No prior programming experience required. Video lectures, one or two classes, two preceptorials.

**Semester:** Fall16

**Lectures:** Tuesday, Thursday, 10:00-10:50

**Location:** TBD

## Faculty

Kevin Wayne

**Office:** Computer Science 207

**Extension:** 4455

**Email:** wayne

## Additional Information

Registrar's Fall16 COS offerings ([http://registrar.princeton.edu/course-offerings/search\\_results.xml?  
subject=COS&term=1172](http://registrar.princeton.edu/course-offerings/search_results.xml?subject=COS&term=1172))

CS Course Schedule (/courses/schedule)

The Undergraduate Coordinator is Colleen Kenny-McGinley

**Email:** ckenny

**Office:** Computer Science 210

**Extension:** 1746

---

## Events (/general/newsevents/events)

15  
Sep

Dan Boneh: New research directions in cryptography (/events/25422)

26  
Sep

Frank Pfenning: Colloquium Speaker (/events/25423)

5  
Oct

Tandy Warnow: Colloquium Speaker (/events/25424)

---

► Full calendar and archive (/general/newsevents/events)

---

## News (/general/newsevents/news)

### AUGUST 16TH, 2016

Prof. Zhandry awarded Best Young Researcher at CRYPTO 2016 (/news/prof-zhandry-awarded-best-young-researcher-crypto-2016)

### AUGUST 11TH, 2016

Strong showing of Princeton researchers at CCS (/news/strong-showing-princeton-researchers-ccs)

### AUGUST 5TH, 2016

Prof. Martonosi named Cornell's Andrew D. White Professor-at-Large (/news/prof-martonosi-named-cornells-andrew-d-white-professor-large)

---

► News archive (/general/newsevents/news)

Follow us: 

(<https://www.facebook.com/PrincetonCS>)



(<https://twitter.com/princetoncs>)



(<http://www.princeton.edu>)

© 2015 The Trustees of Princeton University.

Terms of Use (/general/policies) | Privacy Policy (/general/policies#privacy) | Site Map (/sitemap)

(<http://www.princeton.edu/engineering>)

# CSC108H1: Introduction to Computer Programming

**Division**

Faculty of Arts and Science

**Course Description**

Programming in a language such as Python. Elementary data types, lists, maps. Program structure: control flow, functions, classes, objects, methods. Algorithms and problem solving. Searching, sorting, and complexity. Unit testing. No prior programming experience required.

NOTE: You may not take this course concurrently with CSC120H1/CSC148H1, but you may take CSC148H1 after CSC108H1.

**Department**

Computer Science

**Exclusion**

CSC120H1, CSC121H1, CSC148H1

**Course Level**

100/A

**Arts and Science Breadth**

(5) The Physical and Mathematical Universes

**Arts and Science Distribution**

Science

**Campus**

St. George

**Term**

2016 Fall

**Course Meeting Sections**

[A&S Timetable Information](#)

Activity	Day and Time	Instructor	Location	Class Size	Current Enrolment	Option to Waitlist	Delivery Mode
Lec 0101	FRIDAY 10:00-11:00 WEDNESDAY 10:00-11:00 MONDAY 10:00-11:00	E De Lara	WB 116 WB 116 WB 116	220	220	✓	IN-CLASS
Lec 0102	MONDAY 10:00-11:00 WEDNESDAY 10:00-11:00 FRIDAY 10:00-11:00	J Smith	HS 610 AH 100 AH 100	220	220	✓	IN-CLASS
Lec 0201	FRIDAY 13:00-14:00 WEDNESDAY 13:00-14:00 MONDAY 13:00-14:00	E De Lara	MB 128 MB 128 MB 128	220	220	✓	IN-CLASS
Lec 5101	WEDNESDAY 18:00-21:00	J Smith	MS 3154	250	250	✓	IN-CLASS
Lec 9901		P Gries		250	250	✓	ONLINE

**Last updated**

2016-08-17 09:15:16.0


[Home](#) > [Programme](#) > [Courses Offered](#)

## Course Information

### ENGG1111 Computer Programming and Applications

2012-13						
Instructor(s):	Wong Kenneth	(Class A)	No. of credit(s):	6		
	Chui C K	(Class B)				
	Mitcheson George	(Class C)				
Recommended Learning Hours:	Lecture: 27 Lab Session: 12					
Pre-requisite(s):						
Co-requisite(s):						
Remarks:	This course may not be taken with CSIS1117 or ENGG1013 or ENGG1014.					

#### Course Learning Outcomes

- Analyze simple problems and derive solutions, providing a logical flow of instructions
- Use and construct functions for structured computer programs
- Demonstrate competency in the use of various data structures in program writing
- Identify and correct different types of programming errors, including data validation
- Demonstrate competency in program testing and debugging

#### Syllabus

##### Calendar Entry:

This course covers both the basic and advanced features of the C/C++ programming languages, including syntax, identifiers, data types, control statements, functions, arrays, file access, objects and classes, class string, structures and pointers. It introduces programming techniques such as recursion, linked lists and dynamic data structures. The concept and skills of program design, implementation and debugging, with emphasis on problem-solving, will also be covered.

Target students are those who wish to complete the programming course in a more intensive mode in 1 semester. Students with some programming knowledge are encouraged to take this course.

##### Detailed Description:

Weeks 1-6 Basic programming.	Mapped to Outcomes
C++ syntax, identifiers, data types, control statements, functions, arrays, file access	1, 2
Program design, implementation and debugging, problem solving	1, 4, 5
Weeks 7-13: Advance features.	Mapped to Outcomes
Class and objects, string, structures and pointers, recursion, linked list, dynamic data structures	1, 2, 3

##### Assessment:

Continuous Assessment: 50%  
Written Examination: 50%

**Teaching Plan**

[Class ENGG1111A](#)  
[Class ENGG1111B](#)  
[Class ENGG1111C](#)

 [PAGE TOP](#)

# WATERLOO | CHERITON SCHOOL OF COMPUTER SCIENCE

## CS 137 Programming Principles

### Objectives

This course introduces software engineering students to elementary data structures, and to the functional programming paradigm.

### Intended Audience

Level 1A Software Engineering undergraduates. It is assumed that students have experience developing well-structured, modular programs.

### Related Courses

Antirequisites: CS 115, 135, 136, 145, CHE 121, CIVE 121, ECE 150, GENE 121, PHYS 239, SYDE 121

Successor: CS 138.

### Hardware and Software

Used in course: An integrated development environment (IDE) for C++ (such as xCode for Mac OS X).

### References

C Programming: A Modern Approach 2nd ed. Author: K.N. King

### Schedule

Three hours of lecture per week, plus a two-hour lab and a one-hour tutorial. Normally available in Fall only.

### Outline

#### Introduction and review (6 hours)

Review of hardware and software, problem solving and programming, including declarations, selection and looping constructs, I/O.

#### Subprograms (6 hours)

Functions. Scoping. Parameter passing, by value and by reference. Top-down design. Programming with stubs.

#### Structured Data (7.5 hours)

Arrays. Structures. Arrays of structures. Multidimensional arrays. Appropriate choice of data structures. String processing and tokenization.

#### Recursion (3 hours)

Introduction to recursive procedures and functions.

#### Analysis (1.5 hours)

Introduction to O-notation, space and time efficiency, and complexity.

#### Sorting Algorithms (4.5 hours)

Algorithm design and efficiency, through discussion of elementary sorting algorithms (insertion sort, selection sort) and advanced sorting algorithms (mergesort, quicksort, heapsort).

#### Pointers and Dynamic Structures (6 hours)

Introduction to pointers, memory allocation and deallocation, singly and doubly-linked lists.

## History of Computer Science (1.5 hours)

Babbage, Hilbert, Godel, Church, Turing. Early development of electronic computers and programming languages. History of concepts covered in this course.

David R. Cheriton School of Computer Science  
University of Waterloo  
Waterloo, Ontario, Canada N2L 3G1

MAKING THE FUTURE  
SUPPORT WATERLOO

Tel: 519-888-4567 x33293  
Fax: 519-885-1208

[Contact](#) | Feedback: [cs-webmaster@cs.uwaterloo.ca](mailto:cs-webmaster@cs.uwaterloo.ca) | [David R. Cheriton School of Computer Science](#) | [Faculty of Mathematics](#)



Last modified: Tuesday, 04-Sep-2012 10:33:47 EDT

## [U-M Engineering \(<http://www.engin.umich.edu/>\)](http://www.engin.umich.edu/)

search

- [About \(<http://www.engin.umich.edu/college/about>\)](http://www.engin.umich.edu/college/about) +
- [Research \(<http://www.engin.umich.edu/college/research>\)](http://www.engin.umich.edu/college/research) +
- [Academics \(<http://www.engin.umich.edu/college/academics>\)](http://www.engin.umich.edu/college/academics) +
- [Admissions \(<http://www.engin.umich.edu/college/admissions>\)](http://www.engin.umich.edu/college/admissions) +
- [Departments \(<http://www.engin.umich.edu/college/departments>\)](http://www.engin.umich.edu/college/departments) +
- [Giving \(<http://www.engin.umich.edu/college/giving>\)](http://www.engin.umich.edu/college/giving) +
- [Info for You \(<http://www.engin.umich.edu/college/info>\)](http://www.engin.umich.edu/college/info) +

(<https://facebook.com/michigan.engineering>) (<https://twitter.com/UMengineering>)  
 (<https://instagram.com/michiganengineering>) (<https://youtube.com/michiganengineering>)



[\(<http://umich.edu>\)](http://umich.edu)

University of Michigan

© 2015 THE REGENTS OF THE UNIVERSITY OF MICHIGAN

MICHIGAN ENGINEERING | COLLEGE ADMINISTRATION, 1221 BEAL AVENUE, ANN

ARBOR, MI 48109-2102

+1 (734) 647-7000

[CONTACT THE COLLEGE \(<http://www.engin.umich.edu/>\)](http://www.engin.umich.edu/)  
[SAFETY INFORMATION \(<http://safety.engin.umich.edu/>\)](http://safety.engin.umich.edu/)



# Electrical Engineering and Computer Science Courses

## 100 Level Courses

### **EECS 101. Thriving in a Digital World**

*Prerequisite: none. (4 credits)*

From mobile apps to bitmaps, this course explores computational technologies and how they impact society and our everyday lives. Topics include: social networks, creative computing, algorithms, security and digital privacy. Traditional computer programming is not a primary focus. Instead, mobile applications will be created using a novel visual programming environment.

### **EECS 183. Elementary Programming Concepts**

*Prerequisite: none. (Credit for only one: EECS 183, ENGR 101) (4 credits)*

Fundamental concepts and skills of programming in a high-level

language. Flow of control: selection, iteration, subprograms. Data structures: strings, arrays, records, lists, tables. Algorithms using selection and iteration (decision making, finding maxima/minima, searching, sorting, simulation, etc.) Good program design, structure and style are emphasized. Testing and debugging. Not intended for Engineering students (who should take ENGR 101), nor for CS majors in LSA who qualify to enter EECS 280.

## 200 Level Courses

### EECS 203 (CS 203). Discrete Mathematics

*Prerequisite: MATH 115. (4 credits)*

Introduction to the mathematical foundations of computer science. Topics covered include: propositional and predicate logic, set theory, function and relations, growth of functions and asymptotic notation, introduction to algorithms, elementary combinatorics and graph theory and discrete probability theory.

### EECS 215. Introduction to Electronic Circuits

*Prerequisite: MATH 116, ENGR 101, Corequisite PHYSICS 240 (or 260). Cannot receive credit for both EECS 314 and EECS 215. (4 credits)*

Introduction to electronic circuits. Basic Concepts of voltage and current; Kirchhoff's voltage and current laws; Ohm's law; voltage and current sources; Thevenin and Norton equivalent circuits; DC and low frequency active circuits using operational amplifiers, diodes, and transistors; small signal analysis; energy and power. Time- and frequency-domain analysis of RLC circuits. Basic passive and active electronic filters. Laboratory experience with electrical signals and circuits.

### EECS 216. Introduction to Signals and Systems

*Prerequisite: EECS 215 or EECS 314 or BIOMEDE 211, preceded or accompanied by MATH 216. (4 credits).*

Theory and practice of signals and systems engineering in continuous and discrete time. Continuous-time linear time-invariant systems, impulse response, convolution. Fourier series, Fourier transforms, spectrum, frequency response and filtering. Sampling leading to basic digital signal processing using the discrete-time Fourier and the discrete Fourier transform. Laplace transforms, transfer functions, poles and zeros, stability. Applications of Laplace transform theory to RLC circuit analysis. Introduction to communications, control and signal processing. Weekly recitations and hardware/Matlab software laboratories.

### EECS 230. Electromagnetics I

*Prerequisite: MATH 215, PHYS 240 (or 260), EECS 215. (4 credits)*

Vector calculus. Electrostatics. Magnetostatics. Time-varying fields: Faraday's Law and displacement current. Maxwell's equations in differential form. Traveling waves and phasors. Uniform plane waves. Reflection and transmission at normal incidence. Transmission lines. Laboratory segment may include experiments with transmission lines, the use of computer-simulation exercises, and classroom demonstrations.

FECHAR X

# Ementa de Disciplina

**INF1004****PROGRAMACAO P/ INFORMATICA I****4 créditos****E m e n t a**

Introdução à computação; programando com funções; condicionais e operadores lógicos; solução conceitual; introdução a iteração; modelo de computador; tipagem de dados, variáveis e operadores em uma linguagem procedural; entrada e saída; controle de fluxo procedimental; funções; iteração; vetores e matrizes; desenvolvimento de programas. Vinculação de laboratório: INF 1006.

**B i b l i o g r a f i a**

CELES FILHO, WALDEMAR; CERQUEIRA, RENATO FONTOURA DE GUSMÃO; RANGEL NETTO, JOSÉ LUCAS MOURÃO. INTRODUÇÃO À ESTRUTURAS DE DADOS: COM TÉCNICAS DE PROGRAMAÇÃO EM C; RIO DE JANEIRO: CAMPUS, 2004.

KERNIGHAN, BRIAN W.; RITCHIE, DENNIS M. C, A LINGUAGEM DE PROGRAMAÇÃO PADRÃO ANSI; RIO DE JANEIRO: CAMPUS, 1989.

SCHILD, HERBERT. C COMPLETO E TOTAL; SÃO PAULO: MAKRON, 1991.

**B i b l i o g r a f i a C o m p l e m e n t a r**

Nenhuma bibliografia complementar encontrada para INF1004

**P r é - r e q u i s i t o s**

Nenhum pre-requisito encontrado para INF1004

Última atualização: 29/01/2014

# Course Catalogue

Courses

Lecturers

Time and Place

## 252-0835-00L Computer Science I

Semester	Autumn Semester 2014
Lecturers	F. O. Friedrich
Periodicity	yearly course
Language of instruction	German

 Tabs

### Catalogue data

Abstract	The course covers the fundamental concepts of computer programming with a focus on systematic algorithmic problem solving. Taught language is C++. No programming experience is required.
Objective	Primary educational objective is to learn programming with C++. When successfully attended the course, students have a good command of the mechanisms to construct a program. They know the fundamental control and data structures and understand how an algorithmic problem is mapped to a computer program. They have an idea of what happens "behind the scenes" when a program is translated and executed. Secondary goals are an algorithmic computational thinking, understanding the possibilities and limits of programming and to impart the way of thinking of a computer scientist.
Content	The course covers fundamental data types, expressions and statements, (Limits of) computer arithmetic, control statements, functions, arrays, structural types and pointers. The part on object orientation deals with classes, inheritance and polymorphism, simple dynamic data types are introduced as examples. In general, the concepts provided in the course are motivated and illustrated with algorithms and applications.
Lecture notes	A script written in English will be provided during the semester. The script and slides will be made available for download on the course web page.
Literature	Bjarne Stroustrup: Einführung in die Programmierung mit C++, Pearson Studium, 2010 Stephen Prata, C++ Primer Plus, Sixth Edition, Addison Wesley, 2012 Andrew Koenig and Barbara E. Moo: Accelerated C++, Addison-Wesley, 2000.
Prerequisites / Notice	From AS 2013, an admission to the exam does not any more formally require an attending of the recitation sessions. Handing in solutions to the weekly exercise sheets is thus not mandatory, but we strongly recommend it.  Examination is a one hour-long written test.

### Performance assessment

<b>Performance assessment information (valid until the course unit is held again)</b>	
► Performance assessment as a two-semester course together with 252-0836-00L Computer Science II (next semester)	
For programme regulations (Examination block)	Bachelor Programme in Computational Science and Engineering 2008; Version 01.08.2014 (Examination Block) Bachelor's Programme in Electrical Engineering and Information Technology 2004; Version 19.06.2012 (Examination Block) Bachelor's Programme in Electrical Engineering and Information Technology 2012; Version 24.02.2016 (Examination Block)
ECTS credits	8 credits
► Performance assessment as a semester course (other programmes)	

**In examination block for** Bachelor's Programme in Computational Science and Engineering 2010; Version 01.08.2014 (Examination Block)  
Bachelor's Programme in Computational Science and Engineering 2012; Version 24.02.2016 (Examination Block)

**ECTS credits** 4 credits

**Examiners** F. O. Friedrich

**Type** session examination

**Language of examination** German

**Course attendance confirmation required** No

**Repetition** The performance assessment is offered every session. Repetition possible without re-enrolling for the course unit.

**Mode of examination** written 60 minutes

**Additional information on mode of examination** Wir offerieren im Semester zwei freiwillige Programmierübungen, welche korrigiert und bewertet werden. Die dabei erzielten Punkte werden in die spätere Prüfungsklausur als Bonus mitgenommen. Maximal erreichbarer Bonus entspricht 1/4 Note. Dieser Bonus kann nicht in spätere Repetitionsklausuren mitgenommen werden.

**Written aids** keine

If the course unit is part of an examination block, the credits are allocated for the successful completion of the whole block. This information can be updated until the beginning of the semester; information on the examination timetable is binding.

## Learning materials

Main link	Information
Only public learning materials are listed.	

## Courses

Number	Title	Hours				Lecturers
252-0835-00 V	Informatik I	2 hrs	Wed	08-10	ETF E 1 »	F. O. Friedrich
252-0835-00 U	Informatik I	2 hrs	Mon	13-15	CHN F 42 »	F. O. Friedrich
				13-15	ETZ E 6 »	
				13-15	ETZ F 91 »	
				13-15	ETZ G 91 »	
				13-15	ETZ H 91 »	
				13-15	IFW A 34 »	
				13-15	ML F 34 »	
				13-16	HG E 26.1 »	
				15-17	CHN G 46 »	
				15-17	ETZ F 91 »	
				15-17	ETZ G 91 »	
				15-17	ETZ H 91 »	
				15-17	ETZ K 91 »	
				15-17	HG D 5.1 »	
				16-19	HG E 26.1 »	
				17-19	ETZ F 91 »	
				17-19	ETZ G 91 »	
				17-19	ETZ H 91 »	
				17-19	ETZ K 91 »	
				17-19	IFW D 42 »	
			27.10.	13-15	HG E 23 »	

## Restrictions

There are no additional restrictions for the registration.

## Offered in

Programme	Section	Type
Electrical Engineering and Information Technology Bachelor	First Year Examinations	O 
Computer Science (General Courses)	Computer Science for Non-Computer Scientists	Z 



[Home](#) | [Subject Search](#) | [Help](#) | [Symbols Help](#) | [Pre-Reg Help](#) | [Final Exam Schedule](#) | [My Selections](#)

## Course 6: Electrical Engineering and Computer Science

### Fall 2016

[Course 6 Home](#)   [CI-M Subjects for Undergraduate Majors](#)   [Evaluations \(Certificates Required\)](#)

[6.00-6.299](#)
[6.30-6.799](#)
[6.80-6.ZZZ](#)


### Basic Undergraduate Subjects

#### 6.00 Introduction to Computer Science and Programming



Prereq: None

Units: 3-7-2

 **Lecture:** MW3 (26-100) **Lab:** TBA +final

Introduction to computer science and programming for students with little or no programming experience. Students learn how to program and how to use computational techniques to solve problems. Topics include software design, algorithms, data analysis, and simulation techniques. Assignments are done using the Python programming language. Meets with 6.0001 first half of term and 6.0002 second half of term. Credit cannot also be received for 6.0001 or 6.0002. Final given during final exam week.

J. V. Guttag

No textbook information available

#### 6.0001 Introduction to Computer Science Programming in Python



Prereq: None

Units: 2-3-1

 Ends Oct 21. **Lecture:** MW3 (26-100) **Recitation:** F10 (36-112, 36-153) or F11 (36-112, 36-153) or F12 (36-153) or F1 (36-153) or F2 (36-153) or F3 (36-153) or F12 (36-112) or F1 (36-112) or F2 (36-112)

Introduction to computer science and programming for students with little or no programming experience. Students develop skills to program and use computational techniques to solve problems. Topics include the notion of computation, Python, simple algorithms and data structures, testing and debugging, and algorithmic complexity. Combination of 6.0001 and 6.0002 counts as REST subject. Final given in the seventh week of the term.

J. V. Guttag

[Textbooks \(Fall 2016\)](#)

#### 6.0002 Introduction to Computational Thinking and Data Science

Prereq: [6.0001](#) or permission of instructor

Units: 2-3-1

 Begins Oct 24. **Lecture:** MW3 (26-100) **Recitation:** F10 (36-112, 36-153) or F11 (36-112, 36-153) or F12 (36-153) or F1 (36-153) or F2 (36-153) or F3 (36-153) or F12 (36-112) or F1 (36-112) or F2 (36-112) +final

Provides an introduction to using computation to understand real-world phenomena. Topics include plotting, stochastic programs, probability and statistics, random walks, Monte Carlo simulations, modeling data, optimization problems, and clustering. Combination of 6.0001 and 6.0002 counts as REST subject. Final given during final exam week.

J. V. Guttag

[Textbooks \(Fall 2016\)](#)

#### 6.002 Circuits and Electronics

Prereq: [Physics II \(GIR\)](#); Coreq: [18.03](#) or [2.087](#)

Units: 4-1-7

 **Lecture:** TR11 (34-101) **Lab:** TBA **Recitation:** WF11 (26-310) or WF12 (26-310) or WF1 (26-310) or WF2 (26-310) +final

Fundamentals of the lumped circuit abstraction. Resistive elements and networks, independent and dependent sources,

[Harvard Course Catalog](#)  
Cross-Registration[My Cross-Registration List](#)[Calendars and Information](#)[Credit Conversion](#)**Introduction to Computer Science Programming in Python** or [return to Course Catalog Search](#)**6.0001 MIT****School**

Massachusetts Institute of Technology

**Department**

Electrical Eng &amp; Computer Sci

**Faculty**

Guttag, John V.

**Term**Spring 2016 ([show academic calendar](#))**Day and Time**

Contact host school for schedule

**Credits**6.00 ([show credit conversion for other schools](#))**Credit Level**

Undergraduate

**Description**

Introduction to computer science and programming for students with little or no programming experience. Students develop skills to program and use computational techniques to solve problems. Topics include the notion of computation, Python, simple algorithms and data structures, testing and debugging, and algorithmic complexity. Combination of 6.0001 and 6.0002 counts as REST subject.

**Prerequisite(s)**

None

**Textbook Information** Currently no textbook information is available for this course. Please check back or visit the [Harvard Coop](#).**Cross Registration**

Crossregistration for MIT courses requires instructor approval.

Please [login](#) to create your cross registration course list.

If you are a non-Harvard student, please log in with your XID.

Courses from the Massachusetts Institute of Technology are included to facilitate cross-registration by eligible Harvard students into MIT. These courses are taught at MIT, by MIT instructors, and are subject to MIT policies.

Copyright 2015 President & Fellows of Harvard College, All Rights Reserved | [Privacy Policy](#)

...

# Hiroshima University Syllabus

[Back to syllabus main page](#)

[Japanese](#)

<b>Academic Year</b>	2016Year	<b>School/Graduate School</b>	Liberal Arts Education Program					
<b>Lecture Code</b>	63130001	<b>Subject Classification</b>	Foundation Courses					
<b>Subject Name</b>	プログラミング序説[1工二]							
<b>Subject Name (Katakana)</b>	プログラミングジョセツ							
<b>Subject Name in English</b>	Introduction to Computer Programming							
<b>Instructor</b>	MORIKAWA KATSUMI,TAKAFUJI DAISUKE,KAMEI SAYAKA							
<b>Instructor (Katakana)</b>	モリカワ カツミ,タカフジ ダイスケ,カメイ サヤカ							
<b>Campus</b>	HigashiHiroshima	<b>Semester/Term</b>	1st-Year, First Semester, First Semester					
<b>Days, Periods, and Classrooms</b>	(1st) Mon9-10: 工103, メセ本端							
<b>Lesson Style</b>	Lecture	<b>Lesson Style</b> 【More Details】	Lecture & practice (fifty-fifty)					
<b>Credits</b>	2	<b>Class Hours/Week</b>	2	<b>Language on Instruction</b>	J : Japanese			
<b>Course Level</b>	1 : Undergraduate Introductory							
<b>Course Area (Area)</b>	02 : Informatics							
<b>Course Area (Discipline)</b>	02 : Computing Systems and Information Infrastructures							
<b>Eligible Students</b>	1st year students							
<b>Keywords</b>	Computer programming, C programming language.							
<b>Special Subject for Teacher Education</b>		<b>Special Subject</b>						
<b>Related Programs</b>								
<b>Class Status within General Education /Integrated Courses</b>								
<b>Expected Outcome</b>								
<b>Class Objectives</b>	Acquire basic programming techniques using C language, i.e., (1) learn how to write computer programs, (2) understand the behavior of							

<b>/Class Outline</b>	programs written in C, and (3) write and run C programs for simple processing requests.
<b>Class Schedule</b>	<p>Lesson 1: Guidance. Introduction to programming. Linux and C programming Language. Steps of writing and running a program. (Morikawa)</p> <p>Lesson 2: Programming practice. (Takafuji)</p> <p>Lesson 3: Display "hello, world". Declare variables and assign values. Comments in code. (Morikawa)</p> <p>Lesson 4: Programming practice. (Takafuji)</p> <p>Lesson 5: Data types for integer, floating-point, and character. Formatted output using printf(). Error messages and their solutions. (Morikawa)</p> <p>Lesson 6: Programming practice. (Takafuji)</p> <p>Lesson 7: Test statement. if statement, if-else statement, Increment operator. for statement. (Morikawa)</p> <p>Lesson 8: Programming practice. (Takafuji)</p> <p>Lesson 9: while statement. do statement. Interchangeability among for, while, and do statements. Nested loops. Reading keyboard input. (Morikawa)</p> <p>Lesson 10: Programming practice. (Kamei)</p> <p>Lesson 11: Operations involving integer and floating-point types. Type conversions. Array. (Morikawa)</p> <p>Lesson 12: Programming practice. (Kamei)</p> <p>Lesson 13: Array of arrays. Array initialization. Practical problems using arrays. (Morikawa)</p> <p>Lesson 14: Programming practice. (Kamei)</p> <p>Lesson 15 Summary &amp; final test. (Morikawa)</p>
<b>Text/Reference Books,etc.</b>	<p>(Textbook) C の絵本, (株) アンク著, 翔泳社</p> <p>(Reference) 新・C言語入門 シニア編, 林 晴比古(著), ソフトバンククリエイティブ</p>
<b>PC or AV used in Class,etc.</b>	Textbook, handout, PowerPoint
<b>Suggestions on Preparation and Review</b>	For lessons 1 to 14: Topics studied in the classroom will be covered by the programming practice at ICE room scheduled next week. Try to input sample programs, compile & run these programs. Reading and understanding many good programs is a best way of mastering programming skills.
<b>Requirements</b>	(1) The rooms of this class are (i) lecture room for lessons, and (ii) ICE room for practice. Go to the appropriate room based on the schedule. (2) The contents of practice are how to write C programs, how to use tools for programming, and practices using several examples. (3) Each student should write your programs, obtain the results of these programs, write reports and submit them by the end of the given deadline.
<b>Grading Method</b>	The maximum score of all practices at ICE is about 35. The maximum score of final test is about 65. A pass grade is (i) 60 or more of the sum of them, and (ii) 50% or more of the final test.
<b>Message</b>	Each student should join the class positively to overcome difficulties and solve problems when mastering programming skills. Inactive students may fail to accomplish the class objectives.
	1. Assemble in lecture room 103 (Faculty of Engineering) on the first

**Other**

- day of the class, i.e., 11 April (Mon).  
2. This class accepts students of the Cluster II, Faculty of Engineering.

Please fill in the term-end class evaluation questionnaire.

Instructors will reflect on your feedback and utilize the information for improving their teaching.

Small classes will not conduct the questionnaire to prevent your identity from being uncovered.

[Back to syllabus main page](#)

## **Computer Science and Engineering**

### **COMP 1001 Exploring Multimedia and Internet Computing [3 Credit(s)]**

This course is an introduction to computers and computing tools. It introduces the organization and basic working mechanism of a computer system, including the development of the trend of modern computer system. It covers the fundamentals of computer hardware design and software application development. The course emphasizes the application of the state-of-the-art software tools to solve problems and present solutions via a range of skills related to multimedia and internet computing tools such as internet, e-mail, WWW, webpage design, computer animation, spread sheet charts/figures, presentations with graphics and animations, etc. The course also covers business, accessibility, and relevant security issues in the use of computers and Internet. *Exclusion(s): ISOM 2010, any COMP courses of 2000-level or above*

### **COMP 1002 Computer and Programming Fundamentals I [3 Credit(s)]**

Introduction to computers and programming. Computer hardware and software. Problem solving. Program design. Procedural abstraction. Debugging and testing. Simple and structured data types. Recursive programming. Introduction to searching and sorting. *Exclusion(s): COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022P, COMP 1022Q*

### **COMP 1021 Introduction to Computer Science [3 Credit(s)]**

This course introduces students to the world of Computer Science. Students will experience a range of fun and interesting areas from the world of computing, such as game programming, web programming, user interface design and computer graphics. These will be explored largely by programming in the Python language. *Exclusion(s): COMP 1002, COMP 1004 (prior to 2013-14), COMP 1022P, COMP 1022Q, COMP 2011*

### **COMP 1022P Introduction to Computing with Java [3 Credit(s)]**

This course is designed to equip students with the fundamental concepts of programming elements and data abstraction using Java. Students will learn how to write procedural programs using variables, arrays, control statements, loops, recursion, data abstraction and objects using an integrated development environment. *Exclusion(s): COMP 1002, COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022Q, COMP 2011, ISOM 3320*

### **COMP 1022Q Introduction to Computing with Excel VBA [3 Credit(s)]**

This course is designed to equip students with the fundamental concepts of programming using the VBA programming language, within the context of the Microsoft Excel program. Students will first learn how to use Excel to analyze and present data, and will then learn how to use VBA code to build powerful programs. *Exclusion(s): COMP 1002, COMP 1004 (prior to 2013-14), COMP 1021, COMP 1022P, COMP 2011, ISOM 3230*

### **COMP 1029A Introduction to Mobile Application Development Using Android [1 Credit(s)]**

*[Previous Course Code(s): COMP 4901C]* This course provides a basic introduction to mobile application development using the Android platform. It is intended for students who have some prior programming experience, but wish to learn the basics of mobile application development. The course will introduce them to the Android SDK and development environment, Android application components: Activities and their lifecycle, UI design, Multimedia, and 2D graphics support in Android. Students explore these concepts through self-learning course materials and guided laboratory exercises. Graded P or F. *Prerequisite(s): COMP 1002 OR COMP 1004 (prior to 2013-14) OR COMP 1021 OR COMP 1022P OR COMP1022Q*

### **COMP 1029C C Programming Bridging Course [1 Credit(s)]**

This course introduces the C programming language. It is intended for students who already have some experience in computer programming but wish to learn how to apply those programming skills to the C language. The course covers basic programming topics, such as variables, control, loops, and functions, to more advanced topics. Students explore these by self-learning of course materials together with guided programming exercises. Students without the prerequisites but possess relevant programming knowledge may seek instructor's approval for enrolling in the course. Graded P or F. *Exclusion(s): COMP 1002, COMP 1004 (prior to 2013-14), COMP 2011* *Prerequisite(s): COMP 1021 OR COMP 1022P OR COMP 1022Q OR ISOM 3230 OR ISOM 3320*

### **COMP 1029J Java Programming Bridging Course [1 Credit(s)]**

This course introduces the Java programming language. It is intended for students who already have some experience in computer programming but wish to learn how to apply those programming skills to the Java language. The course covers basic programming topics such as variables, control statements, loops, functions, and object-oriented programming concepts. Students explore these by self-learning of course materials together with guided programming exercises. Students without the prerequisites but possess relevant programming knowledge may seek instructor's approval for enrolling in the course. Graded P or F. *Exclusion(s): COMP 1022P, COMP 3021, ISOM 3320* *Prerequisite(s): COMP 1002 OR COMP 1004 (prior to 2013-14) OR COMP 1021 OR COMP 1022Q OR ISOM 3230*

We use cookies on this website. If you continue to access our website, we'll assume that you consent to receiving cookies in accordance with our [Privacy Policy](http://www.cs.ox.ac.uk/privacy-policy.html) (<http://www.cs.ox.ac.uk/privacy-policy.html>).



## University of Oxford Department of Computer Science

HOME > OUR STUDENTS > COURSES > IMPERATIVE PROGRAMMING I

# Imperative Programming I: 2016-2017

Lecturer Geraint Jones (</people/Geraint.Jones>)

Degrees Preliminary Examinations (</teaching/csp/PreliminaryExaminations>) – Computer Science and Philosophy (</teaching/csp>)

Preliminary Examinations (</teaching/bacompisci/PreliminaryExaminations>) – Computer Science (</teaching/bacompisci>)

Preliminary Examinations (</teaching/mcs/PreliminaryExaminations>) – Mathematics and Computer Science (</teaching/mcs>)

Term Hilary Term 2017 (16 lectures)

## Overview

This course applies lessons that have been learnt in Functional Programming to the design of programs written in an imperative style. By studying a sequence of programming examples, each a useful software tool in its own right, students learn to construct programs in a systematic way, structuring them as a collection of modules with well-defined interfaces.

The course introduces the idea of loop invariants for understanding and reasoning about loops. The course also introduces the idea of modularising larger programs, capturing the functionality of a component of the program using an abstract mathematical specification, and describing formally the relationship between that specification and the implementation.

Through lab exercises, students learn to create, debug and maintain programs of a non-trivial but moderate size.

## Learning outcomes

After studying this course, undergraduates will be able to:

1. Translate basic functional idioms into imperative ones.
2. Design simple loops, using invariants to explain why they work correctly.
3. Use subroutines and modules to structure more complex programs.
4. Specify a module as an abstract datatype, and formalise the relationship between that specification and an implementation.
5. Design simple data structures.
6. Understand the imperative implementation of some common algorithms.

## Synopsis

- Basic imperative programming constructs: assignments, conditionals, procedures and loops. Comparison of imperative and functional programming. Examples.
- Method of invariants: correctness rules for **while** loops; proof of termination. Examples including summing an array, slow and fast exponentiation. Unit testing; debugging.
- Examples: string comparison, printing numbers in decimal.

- Binary search.
- Quicksort.
- Programming with abstract datatypes.
- Objects and classes as modules; specification; data abstraction.
- Reference-linked data structures: linked lists, binary trees.

## Syllabus

Imperative programming constructs, with informal treatment of invariants. Procedures and modules; their use in the design of large programs; specification and implementation of abstract datatypes. Data structures: arrays, reference-linked data structures. Basic tools for program development. Case studies in design of medium-sized programs.

## Reading list

There is no set text for the course, in the sense of a book that is followed by the lectures. I shall be keeping in mind a book about Oberon:

- Reiser & Wirth, Programming in Oberon, Addison--Wesley, 1992.

That book is out of print, but there is a **scanned version** (<http://spivey.ox.ac.uk/wiki/files/rd/ProgInOberonWR.pdf>) available.

Mike Spivey's Oberon compiler implements the Oberon-2 dialect, with a few minor differences that are explained in the Lab Manual. The language itself is described in **the Oberon--2 report** (<ftp://ftp.ethoberon.ethz.ch/Oberon/OberonV4/Docu/Oberon2.Report.ps>) from **ETHZ** (<https://www.ethz.ch/en.html>) (**local copy in PDF** (<http://spivey.ox.ac.uk/wiki/files/rd/o2report.pdf>)).

There are many adequate treatments of the use of logic and invariants in the development of imperative programs; one reasonably pitched one is

- Gries, The Science of Programming, Springer, 1981.

Useful additional cultural reading, recommended for reading after the course, perhaps during the Easter vacation:

- Jon Bentley, *Programming Pearls*, Dorling Kindersley, 2006.

Calendars (/calendars.html)  
Internal (<http://intranet.cs.ox.ac.uk/home2/>)  
RSS Feeds (/rssfeeds.html)  
Sitemap (/sitemap.html)  
Privacy & Cookies (/privacy-policy.html)



(<https://www.linkedin.com/company/department-of-computer-science-university-of-oxford>)



(<https://twitter.com/CompSciOxford>)



(<https://www.facebook.com/DepartmentOfComputerScienceUniversityOfOxford>)



(<https://www.flickr.com/photos/computerscienceoxford/>)



**Público**

Calendário Escolar

Disciplina

Editais

FAQ

**Acesso Restrito**

Entrar

Esqueci a Senha

Primeiro Acesso

**Informações da Disciplina**

Preparar para impressão

**Júpiter - Sistema de Graduação****Instituto de Matemática e Estatística****Ciência da Computação****Disciplina: MAC0110 - Introdução à Computação**

Introduction to Computer Science

**Créditos Aula:** 4**Créditos Trabalho:** 0**Carga Horária Total:** 60 h**Tipo:** Semestral**Ativação:** 01/01/2003**Objetivos**

Introduzir a programação de computadores através do estudo de uma linguagem algorítmica e de exercícios práticos.

*Introduce computer programming by studying an algorithmic language and practical exercises.*

**Docente(s) Responsável(eis)**

3039753 - Mauro Cesar Bernardes

**Programa Resumido**

*A brief history of computing. Algorithms: characterization, notation, basic structures. Computers: basic unities, instructions, stored program, addressing, programs in machine language. Concepts of algorithmic languages: expressions; sequential, selective and repetitive commands; input/output; structured variables; functions. Development and documentation of programs. Samples of non-numeric processing. Extense programming and debugging practices.*

**Programa**

Breve história da computação.

Algoritmos: caracterização, notação, estruturas básicas.

Computadores: unidades básicas, instruções, programa armazenado, endereçamento, programas em linguagem de máquina.

Conceitos de linguagens algorítmicas: expressões; comandos seqüenciais, seletivos e repetitivos; entrada/saída; variáveis estruturadas; funções.

Desenvolvimento e documentação de programas.

Exemplos de processamento não numérico.

Extensa prática de programação e depuração de programas.

*A brief history of computing. Algorithms: characterization, notation, basic structures. Computers: basic unities, instructions, stored program, addressing, programs in machine language. Concepts of algorithmic languages: expressions; sequential, selective and repetitive commands; input/output; structured variables; functions. Development and documentation of programs. Samples of non-numeric processing. Extense programming and debugging practices.*

**Avaliação****Método****Crítério**

Média ponderada de provas e exercícios de programação.

**Norma de Recuperação**

**Bibliografia**

- "Material didático para disciplinas de Introdução à Computação", Projeto MAC Multimídia, «<http://www.ime.usp.br/~macmulti/>».
- V. Setzer, R. Terada, "Introdução à Computação e à Construção de Algoritmos", McGraw-Hill, 1991.
- E. Roberts, "The Art and Science of C", Addison-Wesley, 1995.
- H.M. Deitel, P.J. Deitel, "Como Programar em C", 2a ed., Livros Técnicos e Científicos, 1999.
- J-P. Tremblay, R.B. Bunt, "Ciência dos Computadores", McGraw-Hill, 1983.
- B.W. Kernighan, D.M. Ritchie, "A Linguagem de Programação C, padrão ANSI", Campus, 1990.

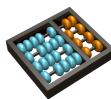
[Clique para consultar os requisitos para MAC0110](#)

[Clique para consultar o oferecimento para MAC0110](#)

---

[Créditos](#) | [Fale conosco](#)

© 1999 - 2016 - Superintendência de Tecnologia da Informação/USP



## MC102 - Algoritmos e Programação de Computadores

A partir de 2011

PRÉ-REQUISITO: não há

### EMENTA

Conceitos básicos de organização de computadores. Construção de algoritmos e sua representação em pseudocódigo e linguagens de alto nível. Desenvolvimento sistemático e implementação de programas. Estruturação, depuração, testes e documentação de programas. Resolução de problemas.

Programa:

Tópicos a serem estudados (preferencialmente nesta ordem):

- 1 - Organização Básica de um Ambiente Computacional
- 2 - Variáveis, Constantes e Atribuições
- 3 - Entrada e Saída de Dados
- 4 - Expressões Aritméticas, Lógicas e Relacionais
- 5 - Comandos Condicionais
- 6 - Comandos de Repetição
- 7 - Vetores e Strings
- 8 - Matrizes
- 9 - Funções
- 10 - Escopo de Variáveis
- 11 - Ponteiros e Alocação Dinâmica de Vetores
- 12 - Algoritmos de Ordenação
- 13 - Algoritmos de Busca
- 14 - Tipos Enumerados e Registros
- 15 - Arquivos Textos e Binários
- 16 - Recursão

### Bibliografia:

- P. Feofiloff. Algoritmos em Linguagem C. Campus-Elsevier, 1ª. edição, 2009  
 H. M. Deitel, P. J. Deitel. C - Como Programar, 6ª. edição, Pearson Education, 2011.  
 B. W. Kernighan, D. M. Ritchie. The C Programming Language, 2ª. edição, Prentice-Hall, 1988 [Tradução: C - A Linguagem de Programação. Editora Campus, 1989]  
 J. L. Szwarcfiter, L. Markenzon. Estruturas de Dados e seus Algoritmos, 3ª. edição, Editora LTC, 2010  
 W. Celes, R. Cerqueira, J.L. Rangel. Introdução a Estruturas de Dados, 1ª. edição, Editora Campus, 2004  
 N. Ziviani. Projeto de Algoritmos com Implementações em Pascal e C, 3ª. edição, Editora Cengage Learning, 2011  
 T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos - Teoria e Prática, 3ª. edição, Editora Campus, 2012  
 R. Sedgewick, K. Wayne. Algorithms, 4ª. edição, Addison-Wesley, 2011  
 A. Kelley, I. Pohl. A Book on C, 4ª. edição, Addison Wesley, 1998

# CS 106A – General Information and Syllabus

---

## Instructors

### Alisha Adam

Email: [aadam@stanford.edu](mailto:aadam@stanford.edu)  
Office: Gates B28  
Office Hours: M/W 2-4pm & by appointment

### Rohit Talreja

Email: [rtalreja@stanford.edu](mailto:rtalreja@stanford.edu)  
Office: Gates B28  
Office Hours: M/W 2-4pm & by appointment

## Class Web Page

The web page for the class is: <http://cs106a.stanford.edu>. All course announcement and materials (lecture slides, handouts, assignments, etc.) will be posted to the website so make sure to check it regularly!

## Course Overview

This course serves as an introduction to the fundamental techniques, programming constructs, and design strategies that form the basis of modern software. You will learn a new approach towards problem solving from a computational perspective and gain an understanding of how programming is applied across many domains. Specifically, at the end of the course, you should feel comfortable writing programs that interact with the user, manipulate graphical objects on the screen, and process and manipulate data.

## Lectures and Discussion Sections

There will be four 50-minute lectures every Monday through Thursday, from 11:30am - 12:20pm in NVIDIA Auditorium. You will also attend a mandatory 50-minute discussion section each week led by one of our awesome section leaders. Section signups are handled online at <http://cs198.stanford.edu/>. Section signups will open on the first day of class (June 20<sup>th</sup>) and close at noon the next day (June 21<sup>st</sup>). We will send you an email with your section assignment by the evening of June 21<sup>st</sup>. Sections will begin the first week of class.

## Text and Handouts

There are two textbooks for the course, both of which are available at the Stanford Bookstore. The main textbook is also available at the university library. Recommended

readings for each lecture will be posted on the lecture calendar. We will also post additional handouts and reference materials on the course website.

- 1) *The Art & Science of Java*. Eric Roberts (ISBN: 978-0321486127).

This is the main textbook for the class. Starting in the second week, most of the readings will be assigned from this book. We definitely recommend having a copy for both the assignments and exams.

- 2) *Karel the Robot Learns Java*. Eric Roberts.

This is a short (35-page) tutorial that introduces major concepts in programming through Karel the Robot. We will use the course reader for the first week of the course, and it will be useful for your first assignment. The reader is available electronically on the course website and in hardcopy from the Stanford Bookstore.

Note: Our exams are open-book, but you will **not** be able to use any digital materials (e.g. PDF version of the course reader) on the exam.

## Assignments

There will be 6 programming assignments throughout the quarter, roughly one for each major topic we will cover. Assignments will require more time as the quarter progresses, so later assignments will be weighted higher (see section below regarding grading). With the exception of the last assignment, you will receive feedback on each assignment during an interactive, one-on-one session with your section leader, who will provide a rating on the following scale for both functionality and programming style:

- ++ *Plus-Plus*: An absolutely mind-blowing solution, the likes of which we see only a few times each quarter. If a section leader thinks an assignment is worthy of this rating, they will pass it along to the instructors for approval.
- + *Plus*: A “perfect” submission or one that exceeds expectations for the assignment. To receive this rating, a submission often reflects additional work beyond the stated requirements, or gets the job done in a particularly elegant way.
- ✓ + *Check-Plus*: A submission that satisfies all the requirements for the assignment, showing solid functionality as well as style.
- ✓ *Check*: A submission that satisfied all the requirements for the assignment, possibly with a few small problems.
- ✓ - *Check-Minus*: A submission that has problems serious enough to fall short of the requirements for the assignment.
- *Minus*: A submission that has extremely serious problems, but demonstrates some effort and understanding of the concepts.

- *Minus-Minus*: A submission that shows little effort and does not represent passing work.

From previous quarters we expect most grades to be in the ✓ or ✓+ range. We have also found that using buckets for scores allows your section leader to spend more time talking about what you need to learn from the assignment without worrying about justifying each point.

After each assignment, your section leader will send out a sign-up form for an interactive grading session. We will explain the interactive grading process in more detail during the first lecture.

**Assignments must be completed individually**, though you are allowed to discuss high-level ideas with each other. Whenever you obtain help, you should credit those who helped you directly in the program (i.e. by adding a comment to the .java file). See the section on the Honor Code below.

## Late Policy

Each assignment will be due at **5:00pm** on the day it is due, as specified on the assignment handout as well as the class calendar. Assignments must be submitted electronically as described on the course website. If you are having technical difficulties with the submission process, email your submission to your section leader **before** the deadline for the assignment.

We understand that things come up and you may need a little extra time to complete an assignment, so every student will be granted **three free “late days”** to be used during the quarter. Each late day allows you to turn in an assignment up to 24 hours late without any penalty. You can use late days individually, or combine two for a 48-hour extension on a single assignment. You may use your late days on any assignment and do not need to request permission (though it may be a good idea to email your section leader so they know that they should expect a late submission). After you have exhausted your three free late days, assignments turned in late will receive a penalty of one grade bucket per day (for example, a ✓+ will turn into a ✓, and so on). **Since this is a compressed quarter, assignments received later than 48 hours following the due date will not be graded.** This also means that you can't use all 3 free late days on the same assignment.

You can think about “late days” as pre-approved extensions to use at your discretion. As a result, getting an extension beyond the three free late days will generally not be

approved. Exceptions are made for very special circumstances, primarily extended medical problems or emergencies. **Extensions beyond the free late days can only be granted by the instructors (specifically, do not ask your section leader about extensions). To request an extension after exhausting your free late days, you must email the instructors at least 24 hours before the assignment deadline.**

## **LaIR Helper Hours**

For most of you, this is your very first experience with computer programming. As a result, we provide extensive support and assistance throughout the quarter. Section leaders are available from 7-11pm from Sunday through Wednesday evenings in the basement of the Gates Computer Science building (room B08) to help with assignments or review course material. The latest schedule for helper hours can be found at [\*\*http://cs198.stanford.edu\*\*](http://cs198.stanford.edu) under the “Helper Schedule” link.

## **Computer Facilities**

The assignments in 106A will require extensive use of a computer. The preferred software is the Eclipse development environment that runs on both Mac OSX and Windows. Instructions for downloading and installing Eclipse are available on the course website. Eclipse should also be installed on all cluster and library computers across campus.

## **Exams**

This course will have two written exams. The exams are **open-book but closed-notes**. This means that you may bring physical copies of both course texts to the exam. However, you may not use any additional typed or handwritten notes, including anything that was posted on the course website. Additionally, electronic devices are not allowed.

The midterm exam will take place **from 7pm – 9pm on Monday, July 18<sup>th</sup> (location TBD)**. If you absolutely cannot make the regularly scheduled midterm, you must send a request via email to the instructors by 5:00pm on Monday, July 10<sup>th</sup> for an alternate exam time. Please include in your email all the possible times that you’re available to take the exam from Monday, July 18<sup>th</sup> to Wednesday, July 20<sup>th</sup>.

**The final exam is scheduled for 12:15pm to 3:15pm on August 12th (location TBD)**. Since this time is set by the University Registrar, **there will be no alternate time for the final exam (SCPD students see below)**. Please make sure that you can attend the final exam before enrolling in the course.

SCPD students can choose to take the exams on campus during the scheduled time or make arrangements through the SCPD office to take them remotely. In the latter case, exams should be sent back through the SCPD office by end of day PST on August 12<sup>th</sup>.

## Grading

Your overall course grade will be calculated as a weighted average of the following categories:

- 50% programming assignment (weighted toward the later assignments)
- 30% final exam
- 15% midterm exam
- 5% section attendance and participation

## Honor Code

Although you are allowed (and encouraged) to discuss high-level assignment ideas with other students, you should write your programs individually. Discussing assignments in such detail that another student could duplicate your code is not permitted. As mentioned earlier, when you receive help from other sources, you must credit them directly in the code by adding comments. **Any assistance used without proper citation may be considered plagiarism.**

Additionally, you are not allowed to copy code or code snippets from any sources. The only exceptions to this are the course textbook, handouts, and lecture examples, which you should cite when used. It is also a violation of the Honor Code to look at another student's code (including students from previous quarters) or show another student your code.

You may not upload your code to a public repository (such as [github.com](https://github.com) or [bitbucket.com](https://bitbucket.com)) or any other website where it can be publicly discovered. Posting code where it can be publicly discovered can be considered a violation of the Honor Code. If you would like to upload your code to a **private** repository, you may do so.

The full text of the Stanford Honor Code can be found at:

<https://communitystandards.stanford.edu>

**Office of Accessible Education**

Students who need an academic accommodation based on the impact of a disability must initiate the request with the Student Disability Resource Center (SDRC) located within the Office of Accessible Education (OAE). SDRC staff will evaluate the request with required documentation, recommend reasonable accommodations, and prepare an Accommodation Letter for faculty dated in the current quarter in which the request is being made. Students should contact the SDRC as soon as possible since timely notice is needed to coordinate accommodations. The OAE is located at 563 Salvatierra Walk, and their phone number is 650-723-1066.

*© Alisha Adam and Rohit Talreja. Thanks to Mehran Sahami, Marty Stepp, Keith Schwarz and Jerry Cain for components of this handout.*

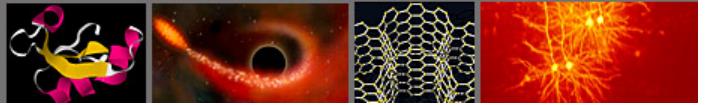


<http://www.vanderbilt.edu>

## Scientific Computing

# Scientific Computing

AT VANDERBILT



[/scientific\\_computing/](/scientific_computing/)

Home [/scientific\\_computing/](/scientific_computing/) > Course Descriptions

## Finding Eligible Scientific Computing Courses

On YES (<https://webapp.mis.vanderbilt.edu/more/SearchClasses!input.action>), to select all courses approved for credit in the Scientific Computing minor, select the “Advanced” link next to the search box, select the “Class Attributes” drop-down box on the bottom right of the advanced search page, and then select “Eligible for Scientific Computing” to find all courses.

## Core Programming Courses

**CS 1101 [Formerly CS 101] Programming and Problem Solving.** An intensive introduction to algorithm development and problem solving on the computer. Structured problem definition, top down and modular algorithm design. Running, debugging, and testing programs. Program documentation. [3] - or -

**CS 1103 [Formerly CS 103] Introductory Programming for Engineers and Scientists.** An introduction to problem solving on the computer. Intended for students other than computer science and computer engineering majors. Methods for designing programs to solve engineering and science problems. Generic programming concepts. [3]

**CS 2201 [Formerly CS 201] Program Design and Data Structures.** The study of elementary data structures, their associated algorithms and their application in problems; rigorous development of programming techniques and style; design and implementation of programs with multiple modules, using good data structures and good programming style. Prerequisite: CS 1101. [3]

- or -

**CS 2204 [Formerly CS 204]** Program Design and Data Structures for Scientific Computing. Data structures and their associated algorithms in application to computational problems in science and engineering. Time and memory complexity; dynamic memory structures; sorting and searching; advanced programming and problem-solving strategies; efficient software library use. Prerequisite: CS 1101 or 1103. [3]

## Scientific Computing Courses

**SC 3250 [Formerly SC 250] Scientific Computing Toolbox.** Team taught course with topics illustrating use of computational tools in multiple science and engineering domains. Topics may include simulations of complex physical, biological, social, and engineering systems, optimization and evaluation of simulation models, Monte Carlo methods, scientific visualization, high performance computing, or data mining. Prerequisite: CS 1101 or 1103; Math 1200. [3]

**SC 3260 [Formerly SC 260] High Performance Computing.** Introduction to concepts and practice of high performance computing. Parallel computing, grid computing, GPU computing, data communication, high performance security issues, performance tuning on shared-memory-architectures. Prerequisite: CS 2201 or CS 2204. SPRING. [3]

**SC 3841 [Formerly SC 293A] Directed Study in Scientific Computing.** Participation in ongoing research projects under direction of a faculty sponsor. Project must combine scientific computing tools and techniques with a substantive scientific or engineering problem. Consent of both the faculty sponsor and one Director of the SC minor is required. Prerequisite: SC 3250. [1-3 each semester]

**SC 3842 [Formerly SC 293B] Directed Study in Scientific Computing.** Continuation of SC 3841 under the direction of the same or different faculty sponsor. Same requirements as for SC 3841 [1-3 each semester]

**SC 3843 [Formerly SC 293C] Directed Study in Scientific Computing.** Continuation of SC 3842 under the direction of the same or different faculty sponsor. Same requirements as for SC 3841. [1-3 each semester]

**SC 3851 [Formerly SC 295A] Independent Study in Scientific Computing.** Development of a research project by the individual student under direction of a faculty sponsor. Project must combine scientific computing tools and techniques with a substantive scientific or engineering problem. Consent of both the faculty sponsor and one Director of the SC minor is required. Prerequisite: SC 3250. [1-3 each semester]

**SC 3852 [Formerly SC 295B] Independent Study in Scientific Computing.** Continuation of SC 3851 under the direction of the same or different faculty sponsor. Same requirements as for SC 3851. [1-3 each semester]

**SC 3853 [Formerly SC 295C] Independent Study in Scientific Computing.** Continuation of SC 3852 under the direction of the same or different faculty sponsor. Same requirements as for SC 3851. [1-3 each semester]

**SC 3890 [Formerly SC 290] Special Topics in Scientific Computing.** Special topics course. [3]

These pages contain selected sections of [Structure and Interpretation of Computer Programs](#) by Harold Abelson and Gerald Sussman with Julie Sussman.

The programs are given in JavaScript, translated from Scheme by Martin Henz. The JavaScript programs can be executed (evaluated) by clicking on them.

If you are interested, here is [some information](#) on how these pages were generated, and what tools are used “under the hood”.

## Contents

### [1 Building Abstractions with Functions](#)

#### [1.1 The Elements of Programming](#)

##### [1.1.1 Expressions](#)

##### [1.1.2 Naming and the Environment](#)

##### [1.1.3 Evaluating Operator Combinations](#)

##### [1.1.4 Functions](#)

##### [1.1.5 The Substitution Model for Function Application](#)

##### [1.1.6 Conditional Statements and Predicates](#)

##### [1.1.7 Example: Square Roots by Newton’s Method](#)

##### [1.1.8 Functions as Black-Box Abstractions](#)

### [1.2 Functions and the Processes They Generate](#)

#### [1.2.1 Linear Recursion and Iteration](#)

#### [1.2.2 Tree Recursion](#)

#### [1.2.3 Orders of Growth](#)

#### [1.2.4 Exponentiation](#)

#### [1.2.5 Greatest Common Divisors](#)

#### [1.2.6 Example: Testing for Primality](#)

### [1.3 Formulating Abstractions with Higher-Order Functions](#)

#### [1.3.1 Functions as Arguments](#)

#### [1.3.2 Function Definition Expressions](#)

#### [1.3.3 Functions as General Methods](#)

#### [1.3.4 Functions as Returned Values](#)

### [2 Building Abstractions with Data](#)

#### [2.1 Introduction to Data Abstraction](#)

##### [2.1.1 Example: Arithmetic Operations for Rational Numbers](#)

##### [2.1.2 Abstraction Barriers](#)

##### [2.1.3 What Is Meant by Data?](#)

##### [2.1.4 Extended Exercise: Interval Arithmetic](#)

#### [2.2 Hierarchical Data and the Closure Property](#)

##### [2.2.1 Representing Sequences](#)

[2.2.2 Hierarchical Structures](#)[2.2.3 Sequences as Conventional Interfaces](#)[2.2.4 Example: A Picture Language](#)[2.3 Symbolic Data](#)[2.3.1 Strings](#)[2.3.2 Example: Symbolic Differentiation](#)[2.3.3 Example: Representing Sets](#)[2.3.4 Example: Huffman Encoding Trees](#)[2.4 Multiple Representations for Abstract Data](#)[2.4.1 Representations for Complex Numbers](#)[2.4.2 Tagged data](#)[2.4.3 Data-Directed Programming and Additivity](#)[2.5 Systems with Generic Operations](#)[2.5.1 Generic Arithmetic Operations](#)[2.5.2 Combining Data of Different Types](#)[3 Modularity, Objects, and State](#)[3.1 Assignment and Local State](#)[3.1.1 Local State Variables](#)[3.1.2 The Benefits of Introducing Assignment](#)[3.1.3 The Costs of Introducing Assignment](#)[3.2 The Environment Model of Evaluation](#)[3.2.1 The Rules for Evaluation](#)[3.2.2 Applying Simple Functions](#)[3.2.3 Frames as the Repository of Local State](#)[3.2.4 Internal Definitions](#)[3.3 Modeling with Mutable Data](#)[3.3.1 Mutable List Structure](#)[3.3.2 Representing Queues](#)[3.3.3 Representing Tables](#)[3.4 Concurrency: Time Is of the Essence](#)[3.5 Streams](#)[3.5.1 Streams Are Delayed Lists](#)[3.5.2 Infinite Streams](#)[4 Metalinguistic Abstraction](#)[4.1 The Metacircular Evaluator](#)[4.1.1 The Core of the Evaluator](#)[4.1.2 Representing Statements and Expressions](#)[4.1.3 Evaluator Data Structures](#)[4.1.4 Running the Evaluator as a Program](#)

## 5 Computing with Register Machines



# UC Berkeley EECS

## CS10 : The Beauty and Joy of Computing

### Spring 2012



## QuickLinks

[UC-WISE \(labs\)](#)

[Google+ \(#cs10news\)](#)

[Piazza](#)

[BYOB](#)

[bSpace](#)

[Webcasts](#)

## Overview

**CS10**, *The Beauty and Joy of Computing*, is an exciting new course offered by the [UC Berkeley EECS Dept](#). Computing has changed the world in profound ways. It has opened up wonderful new ways for people to connect, design, research, play, create, and express themselves. However, just using a computer is only a small part of the picture. The real transformative and empowering experience comes when one learns how to program the computer, to translate ideas into code. This course will teach students how to do exactly that, using [BYOB](#) (based on [Scratch](#)), one of the friendliest programming languages ever invented. It's purely graphical, which means programming involves simply dragging blocks around, and building bigger blocks out of smaller blocks.



Our labs are held in the Apple Orchard, which is not only the newest lab on campus with the fastest machines, but also has the most natural light!

But this course is far more than just learning to program. We'll focus on some of the "Big Ideas" of computing, such as abstraction, design, recursion, concurrency, simulations, and the limits of computation. We'll show some beautiful applications of computing that have changed the world, talk about the history of computing, and where it will go in the future. Throughout the course, relevance will be emphasized: relevance to the student and to society. As an example, the final project will be completely of the students' choosing, on a topic most interesting to them. The overarching theme is to expose students to the beauty and joy of computing. This course is designed for computing non-majors, although interested majors are certainly welcome to take the class as well! We are especially excited about bringing computing (through this course) to traditionally under-represented groups in computing, i.e., women and ethnic minorities.



Fall 2009 students work together using pair programming

Fall 2009 students [pair programming](#) in Scratch.

Some context: in the Fall of 2009, we piloted a 2-unit version of this course as the freshman/sophomore seminar [CS39N: The Beauty and Joy of Computing](#) to 20 students. [It was such a success](#) that we decided to move ahead to make this course our new computing course for non-majors, replacing the venerable [CS3L](#) and [CS3S](#). Last fall (2010) was a 90-person pilot and we're continuing to grow the course as word spreads to more students. We're continually replacing the weakest parts of the curriculum and hope you'll enjoy!

We will be using Pair Programming, described best by Laurie Williams, a computer science professor at North Carolina State University: "Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software."

## UC Berkeley CS10 (The Beauty and Joy of Computing) Testimonial #01 [720p HD]



18 student testimonials about CS10!

## News

- 03-08**      **Midterm Project Progress Report Submission Form**  
Now available [Here](#). Only submit ONE per group.
- 02-29**      **Midterm Project Proposal Submission Form**  
Now available [Here](#). Only submit ONE per group.
- 01-29**      **bSpace set up**  
You should now have access to the bSpace course: "CS10 Sp12." If you don't see it, please let us know right away!
- 01-24**      **Section #8 - it's official!**  
Please see the weekly schedule below for the finalized days and times.
- 01-23**      **Enrollment key restriction removed for UC-WISE / sage.cs / The labs**  
The Spring 2012 (online) lab site was set up with an enrollment key (as many of you found out when trying to "enrol" [sic] in the course). The key has been removed, so be sure to click the "enrol [sic] me in this course" link on the left-hand side of the page (after you log in).
- 01-19**      **New section, first lab tomorrow (Friday the 19th at 9am)**  
As the new section's (8's) first lab has already passed (Wednesday morning), it's important to go to tomorrow's (Friday's) lab at 9am if you want to attend the 8th section and/or haven't been to lab yet this week.
- 01-18**      **Welcome to CS10, everyone!**  
Those of us on staff are really excited about sharing the Beauty and Joy of Computing with you all, and are looking forward to a great semester! -The Staff

## Webcasts

Webcasts of our lectures are freely available online!

- [Spring 2012](#)
- [Fall 2011](#)
- [Spring 2011](#)
- [Fall 2010](#)

## Calendar

### Weekly Schedule

**NOTE: Discussion time for section #8 (108) is still TBD, so if you are wait-listed, please go to any of the existing Friday sections for the first week or two.**

Hour	Monday	Tuesday	Wednesday	Thursday	Friday
------	--------	---------	-----------	----------	--------

9:00am

10:00am

11:00am

12:00pm

1:00pm

2:00pm

3:00pm

4:00pm

5:00pm

6:00pm

7:00pm

8:00pm

9:00pm

Lab 014  
(Navin Eluthesen)  
[200 Sutardja Dai](#)

Lab 018  
(Samir Makhani)  
[200 Sutardja Dai](#)

Lab 014  
(Navin Eluthesen)  
[200 Sutardja Dai](#)

Lab 018  
(Samir Makhani)  
[200 Sutardja Dai](#)

Discussion 101  
(Aijia Yan)  
[320 SODA](#)

Lab 018  
(Samir Makhani)  
[200 Sutardja Dai](#)

Lab 015  
(Luke Segars)  
[200 Sutardja Dai](#)

Lab 015  
(Luke Segars)  
[200 Sutardja Dai](#)

Discussion 102  
(Yaniv (Rabbit))  
[320 SODA](#)

Discussion 103  
(Yaniv (Rabbit))  
[320 SODA](#)

Lab 016  
(Pierce Vollucci)  
[200 Sutardja Dai](#)

Lab 016  
(Pierce Vollucci)  
[200 Sutardja Dai](#)

Discussion 104  
(Navin Eluthesen)  
[320 SODA](#)

Discussion 108  
(Samir Makhani)  
[7 Evans](#)

Lecture 10 Evans

Lab 017  
(Navin Eluthesen)  
[200 Sutardja Dai](#)

Lecture 10 Evans

Lab 017  
(Navin Eluthesen)  
[200 Sutardja Dai](#)

Discussion 106  
(Pierce Vollucci)  
[320 SODA](#)

Lab 011  
(Aijia Yan)  
[200 Sutardja Dai](#)

Lab 011  
(Aijia Yan)  
[200 Sutardja Dai](#)

Discussion 107  
(Navin Eluthesen)  
[320 SODA](#)

[200 Sutardja Dai](#)

[200 Sutardja Dai](#)

Lab 012  
(Yaniv (Rabbit))  
[200 Sutardja Dai](#)

Lab 012  
(Yaniv (Rabbit))  
[200 Sutardja Dai](#)

Lab 013  
(Yaniv (Rabbit))  
[200 Sutardja Dai](#)

Lab 013  
(Yaniv (Rabbit))  
[200 Sutardja Dai](#)

## Semester Schedule (subject to change)

- **These readings are required, but are challenging - you should understand the "big idea" concepts, rather than the technical details.**
- **These readings are optional (but recommended).**

Week	Days in 2012	Readings (Sa/Su)	Lecture 1 (M)	Lab 1 (M/Tu)	Lecture 2 (W)	Lab 2 (W/Th)	Discussion (F)	HW & Projects Due
1	Jan 16 - Jan 21	<ul style="list-style-type: none"> <li>• <a href="#">Prof. Harvey's Intro to Abstraction</a></li> <li>• <a href="#">Why Software is Eating the World</a></li> <li>• <a href="#">Is Abstraction the Key to Computing?</a></li> <li>• <a href="#">Designing Games with a Purpose (GWAP)</a></li> <li>• <a href="#">Justices Split on Violent Games</a></li> <li>• <a href="#">Kinect's Future a Game Controller in Everything</a></li> </ul>		MLK Holiday (extended)	MLK Holiday (extended)	<a href="#">Abstraction</a>	<a href="#">Broadcast Animations, &amp; Music</a>	Welcome, Introductions, and Expectations
2	Jan 23 - Jan 28	<ul style="list-style-type: none"> <li>• <a href="#">Animating a Blockbuster</a></li> <li>• <a href="#">More readings on video games</a></li> <li>• <a href="#">Program or Be Programmed (Video: Author</a></li> </ul>	<a href="#">3D Graphics</a>	<a href="#">Loops and Variables</a>	<a href="#">Video Games</a>	<a href="#">Random, If, &amp; Input</a>	<a href="#">Anatomy of a Computer &amp; The Power of Binary</a>	<a href="#">Homework 0</a> Friday at 11:59pm
	Jan 30 -				<a href="#">Programming</a>			<a href="#">Homework 1</a>

3	Feb 04	<a href="#">Speech</a> ) • <a href="#">Scratch: Programming for All (CACM)</a> • <a href="#">BtB Chapter 1</a>	<a href="#">Functions</a>	<a href="#">BYOB</a>	<a href="#">Paradigms</a>	<a href="#">Lists I</a>	Video games	Friday at 11:59pm
4	Feb 06 - Feb 11	Guest Lecturer TA Luke Segars: <a href="#">Algorithms I</a>	<a href="#">Lists II</a>	<a href="#">Algorithms II, Order of Growth</a>	<a href="#">Algorithms</a>	<a href="#">Lists &amp; Algorithms</a>	<a href="#">Homework 2</a>	Friday at 11:59pm
5	Feb 13 - Feb 18	No Reading (QUEST) <b>Quest Review:</b> <a href="#">4-8 go to 390 Hearst Mining</a> Sunday, 6-9pm, 2050 VLSB	<a href="#">Quest (in-class exam -- lab sections</a> <a href="#">Algorithm Complexity</a>	Guest Lecturer TA Yaniv Assaf: <a href="#">Concurrency</a>	<a href="#">Concurrency</a>	Algorithms & Algorithmic Complexity		
6	Feb 20 - Feb 25	• <a href="#">How Moore's Law Works</a> • <a href="#">Free Lunch is Over</a> • <a href="#">Spending Moore's dividend (CACM)</a>	Presidents Day (extended)	Presidents Day (extended)	<a href="#">Recursion I</a>	<a href="#">Recursion I</a>	<a href="#">Recursion, Project Introduction, and Homework 3</a>	<a href="#">Homework 3</a> Friday at 11:59pm
7	Feb 27 - Mar 03	• <a href="#">BtB Chapter 2</a> • <a href="#">Computing as Social Science</a> • <a href="#">BtB Chapter 3</a> • <a href="#">BtB Chapter 4</a>	<a href="#">Social Implications I</a>	<a href="#">Project Work</a>	Guest Lecturer TA Aijia Yan: <a href="#">Recursion II</a>	<a href="#">Recursion II</a>	Recursion Revisited	
8	Mar 05 - Mar 10	Reading Segment 1 • <a href="#">BtB Chapter 4</a> Reading Segment 2 • <a href="#">Living in a Digital World</a> • <a href="#">BtB Chapter 5</a> Reading Segment 1 • <a href="#">BtB Chapter 5</a> Reading Segment 2 • <a href="#">BtB Chapter 5</a> Reading Segment 3 • <a href="#">Data Explosion Creates Revolution</a>	Guest Lecturer Gerald Friedland: <a href="#">Social Implications II</a>	<a href="#">Project Work</a>	Guest Lecturer Bjoern Hartmann: <a href="#">HCI</a>	<a href="#">Recursion III</a>	Social Implications of Computing	
9	Mar 12 - Mar 17	Reading Segment 3 • <a href="#">BtB Chapter 6 (27-37)</a> • <a href="#">Data Explosion Creates Revolution</a>	<a href="#">Game Theory</a>	<a href="#">Project Work</a>	Guest Lecturer Raffi Krikorian: <a href="#">'Twitter'</a>	<a href="#">Applications That Changed The World</a>	Midterm Prep	<a href="#">Midterm Project (and some general tips)</a> Friday at 11:59pm
10	Mar 19 - Mar 24	No Reading (MIDTERM) <b>Midterm Review:</b> Sunday, 6-9pm, 2050 VLSB	Guest Lecturer Anna Rafferty: <a href="#">Artificial Intelligence</a>	<a href="#">Online Midterm Exam</a>	Guest Lecturer TA Luke Segars: <a href="#">Applications that Changed the World</a>	<a href="#">Study for Paper Midterm Exam, 3/22 155 Dwinelle, 6-8pm (Info) (Histogram)</a>	Artificial Intelligence	
11	Mar 26 - Mar 31	No Reading (Break)	Spring Break	Spring Break	Spring Break	Spring Break	Spring Break	
12	Apr 02 - Apr 07	• <a href="#">BtB Chapter 7</a> <a href="#">Lambda + HOFs I</a> <a href="#">Lambda + HOFs I</a> <a href="#">Lambda + HOFs II</a> <a href="#">Lambda + HOFs II</a>					<a href="#">HOFs &amp; Lambdas</a>	<a href="#">Blog Entry</a> Sunday at 11:59pm
13	Apr 09 - Apr 14	• <a href="#">What is Cloud Computing? (Video)</a> • <a href="#">A Berkeley View of Cloud Computing</a> • <a href="#">What is IBM's Watson?</a> • <a href="#">The Great Robot Race (Video)</a> • <a href="#">Computers Solve Checkers – It's a Draw</a> • <a href="#">Brian Harvey's AI notes</a> • <a href="#">The First</a>	<a href="#">Distributed Computing</a>	<a href="#">Distributed Computing</a>	Guest Lecturer Prof Kathy Yellick: <a href="#">'Saving the World with Computing (CS + X)'</a>	<a href="#">Project Work</a>	<a href="#">HOFs &amp; Lambdas Revisited</a>	<a href="#">Blog Comments</a> Friday at 11:59pm
14	Apr 16 - Apr 21	<a href="#">Limits of Computing</a>	<a href="#">Project Work</a>	<a href="#">Future of Computing</a>	<a href="#">Project Work and Peer Review</a>	Open Topic		<a href="#">Project Peer Review Write-up</a> Friday at 11:59pm



15	Apr 23 - Apr 28	<p><a href="#">Church of Robotics</a></p> <ul style="list-style-type: none"> <li>• <a href="#">The Thinking Machine (Video)</a></li> <li>• <a href="#">Computer Pioneer Alan Turing</a></li> <li>• <a href="#">Why is Quantum Different?</a></li> <li>• <a href="#">Quantum Leap</a></li> <li>• <a href="#">Twenty Top Predictions for life 100 years from now</a></li> </ul> <ul style="list-style-type: none"> <li>• <a href="#">Apple's 1987 Knowledge Navigator (Video)</a></li> <li>• <a href="#">Microsoft's view of productivity in 2019 (Video)</a></li> <li>• <a href="#">The Future of Augmented Reality</a></li> <li>• <a href="#">Apple's Siri</a></li> <li>• <a href="#">BtB: Conclusion</a></li> </ul>	Guest Lecturer Prof Armando Fox: ' <a href="#">Cloud Computing</a> '	<a href="#">Project Work</a>	<a href="#">Summary and Farewell</a>	Online Final Exam	Final Thoughts
16	Apr 30 - May 05	No Reading (RRR)  No Reading (Final)	RRR Week	RRR Week	RRR Week	RRR Week	<a href="#">RRR Week</a>
17	May 07 - May 12	<p><b>Final Review:</b> Sunday, 6-9pm, 2050 VLSB</p> <ul style="list-style-type: none"> <li>• <a href="#">Reading Slides</a></li> <li>• <a href="#">Programming Slides</a></li> </ul>	Finals Week	Finals Week	<p><b>Paper Final Exam May 9th, 7-10 pm (10 Evans)</b></p> <p><a href="#">Supplementary Handout</a></p>	Finals Week	Finals Week

## Staff

### Lecturer



[Dan Garcia \( bio \)](#)  
[ddgarcia@cs.berkeley.edu](mailto:ddgarcia@cs.berkeley.edu)  
 777 Soda, (510) 517-4041  
 OH: F 2-3pm in 777 Soda

(but please first check his [Travel Schedule](#) to be sure he's here that week!)

### Teaching Assistants



**Pierce Vollucci** ([bio](#))  
[cs10-ra@inst.eecs.berkeley.edu](mailto:cs10-ra@inst.eecs.berkeley.edu)

**OH** : Th 3-4pm, Tables Outside Lab



**Luke Segars** ([bio](#))  
[cs10-tc@inst.eecs.berkeley.edu](mailto:cs10-tc@inst.eecs.berkeley.edu)

360 Hearst Mining

**OH** : W 4-5pm, Tables Outside Lab



**Navin Eluthesen** ([bio](#))  
[cs10-ta@inst.eecs.berkeley.edu](mailto:cs10-ta@inst.eecs.berkeley.edu)

**OH** : M 2-3pm, 283E Soda  
& F 3-4pm, 611 Soda



**Yaniv Assaf, aka Rabbit** ([bio](#))  
[rab6bit@gmail.com](mailto:rab6bit@gmail.com)

**OH** : Th 5-6pm, 411 Soda  
& F 1-2pm, 200 SD Lab



**Aijia Yan** ([bio](#))  
[cs10-tf@inst.eecs.berkeley.edu](mailto:cs10-tf@inst.eecs.berkeley.edu)

**OH** : F 11am-12pm, Tables Outside Lab



**Samir Makhani** ([bio](#))  
[makhani@berkeley.edu](mailto:makhani@berkeley.edu)

**OH**: F 2-3pm, 200 SD Lab

## Readers



**Shreya Lakhan-Pal** ([bio](#))  
[cs10-rd@inst](mailto:cs10-rd@inst)



**Kylan Nieh** ([bio](#))  
[cs10-re@inst](mailto:cs10-re@inst)



**Head Grader Max Dougherty** ([bio](#))  
[cs10-rf@inst](mailto:cs10-rf@inst)



**Christian Pedersen** ([bio](#))  
[cs10-rc@inst](mailto:cs10-rc@inst)



**Ian Birnam** ([bio](#))  
[cs10-rb@inst](mailto:cs10-rb@inst)

## Grading

For the most part, we would prefer to teach this course *without* grades. What a wonderful concept, learning for learning sake! However, even though we can't change the "system" overnight, we can create grading policies that support learning as much as possible. The various course activities will contribute to your grade as follows:

Activity	Course Points	Percentage of Total Grade
----------	---------------	---------------------------

Weekly Quizzes and Homework	60	15%
Paper	60	15%
Midterm Project	60	15%
Final Project	60	15%
Quest	20	5%
Midterm	60	15%
Final Exam	80	20%

## How We'll Calculate Your Grade

Your letter grade will be determined by total course points, as shown in the table below. There is no curve; your grade will depend only on how well you do, not on how well everyone else does. Incomplete grades will be granted only for dire medical or personal emergencies that cause you to miss the final exam, and only if your work up to that point is satisfactory.

Points	Grade
390-400	A+
370-389	A
360-369	A-
350-359	B+
330-349	B
320-329	B-
310-319	C+
290-309	C
280-289	C-
240-279	D
< 240	F

## Resources

- ▲ [BYOB : Build Your Own Blocks](#)
- ▲ [Scratch Forums](#)
- ▲ [Blown to Bits](#)
- ▲ [Debugging Rules!](#)
- ▲ [UC Berkeley](#)
- ▲ [College of Engineering](#)
- ▲ [Department of Electrical Engineering & Computer Sciences](#)
- ▲ [Webcast archive of 2010Fa lectures](#)
- ▲ [Solutions to Lab Exercises](#)

Contact [Webmaster](#) 2014-06-28@01:20:40 PDT

This work is licensed under a [Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License](#)





**Disciplina: Ementa/Programa**  
[FECHAR JANELA](#)

<b>Órgão:</b>	CIC - Departamento de Ciência da Computação
<b>Código:</b>	113476
<b>Denominação:</b>	Algoritmos e Programação de Computadores
<b>Nível:</b>	Graduação
<b>Vigência:</b>	1971/2
<b>Pré-req:</b>	Disciplina sem pré-requisitos
<b>Ementa:</b>	Princípios fundamentais de construção de programas. Construção de algoritmos e sua representação em pseudocódigo e linguagens de alto nível. Noções de abstração. Especificação de variáveis e funções. Testes e depuração. Padrões de soluções em programação. Noções de programação estruturada. Identificadores e tipos. Operadores e expressões. Estruturas de controle: condicional e repetição. Entrada e saída de dados. Estruturas de dados estáticas: agregados homogêneos e heterogêneos. Iteração e recursão. Noções de análise de custo e complexidade. Desenvolvimento sistemático e implementação de programas. Estruturação, depuração, testes e documentação de programas. Resolução de problemas. Aplicações em casos reais e questões ambientais.
<b>Bibliografia:</b>	<p>Básica</p> <p>Cormen, T. et al., Algoritmos: Teoria e Prática. 3a ed., Elsevier - Campus, Rio de Janeiro, 2012</p> <p>Ziviani, N., Projeto de Algoritmos com implementação em Pascal e C, 3a ed., Cengage Learning, 2010.</p> <p>Felleisen, M. et al., How to design programs: an introduction to computing and programming, MIT Press, EUA, 2001.</p> <p>Complementar</p> <p>Evans, D., Introduction to Computing: explorations in Language, Logic, and Machines, CreatSpace, 2011.</p> <p>Harel, D., Algorithmics: the spirit of computing, Addison-Wesley, 1978.</p> <p>Manber, U., Introduction to algorithms: a creative approach, Addison-Wesley, 1989.</p> <p>Kernighan, Brian W; Ritchie, Dennis M., C, a linguagem de programação: Padrão ansi. Rio de Janeiro: Campus</p> <p>Farrer, Harry. Programação estruturada de computadores: algoritmos estruturados. Rio de Janeiro: Guanabara Dois, 2002.</p>

**© 2004-2016 CPD - Centro de Informática  
UnB - Universidade de Brasília**

# **CS 105L: Introduction to Computer Programming using JavaScript**

## **Spring 2016**

[Department of Computer Science](#)  
[University of New Mexico](#)



### ***Instructor:***

[Joel Castellanos](#)  
[Department of Computer Science](#)

### ***Course Description***

CS-105L, Introduction to Computer Programming, is a gentle and fun introduction. Students will use a simple Integrated Development Environment to author small programs in a high level language that do interesting things.

Pre-Requisites: None.

Co-Requisites: None.

This semester, we will be programming in JavaScript to make small, but fun programs that run in Web Browsers. In addition to programming, the course also covers some of the great ideas in computer science such as modeling, visualization, emergence, search engine page ranking systems, and complex adaptive systems. Throughout the course, students will author many short programs that perform two-dimensional graphics, animations, customized image manipulations and some simple games.

CS-105 is designed as a first course in computer programming for:

1. Pre-CS majors who do not have previous programming experience and are not yet ready for the fast pace and rigor of CS-152 (Computer Programming Fundamentals).
2. Students without programming experience who want to learn the basics of programming, an introduction to the JavaScript, HTML 5, and CSS to gain practical skills in Web design, how to create customized multi-media effects and other tasks.

---

### ***Syllabus & Lab Schedule:***

[CS-105 Syllabus for Spring 2016](#)

[CS-105 Schedule of lab, office hours and tutor times and e-mails.](#)

---

### ***Textbooks and Web Resources:***



### Eloquent JavaScript, 2nd Edition

by Marijn Haverbeke

ISBN-13: 978-1593275846

**w3schools.com** [w3schools.com: HTML 5 Tutorial](#)

**w3schools.com** [w3schools.com: JavaScript Tutorial](#)

**w3schools.com** [w3schools.com: CSS Tutorial](#)

## Software:



[WebStorm: JavaScript Integrated Development Environment \(IDE\)](#) Note: with your UNM e-mail, you can get a free student license.



[putty.exe](#) Windows client for remote terminal access to linux.unm.edu.

## Lecture Notes, Videos, Labs and Source Code

### Week 1:

1. Video: [Khan Academy: What is Programming?](#)
2. Video: [Khan Academy: Learning Programming on Khan Academy](#)
3. Video: [Khan Academy: Drawing Basics](#)
4. Notes: [Welcome to CS-105](#)
5. Lab: [Lab 1: I Love to Draw](#)
6. Code: [CS105\\_Lab1\\_MyFirstName\\_MyLastName.html](#) This is the Template HTML file I showed in class.

### Week 2:

1. Video: [Khan Academy: Coloring](#)
2. Video: [Khan Academy: Variables](#)
3. Video: [Khan Academy: Animation Basics](#)
4. Read: [w3Schools on if statements](#) (and check out the "Try it Yourself" button).
5. Notes: [Variables and the Draw Loop](#)
6. Code: [CS105\\_Lab2\\_BouncingBall.html](#) This is the starting bouncing ball code for lab
  2. Start with this and change it.
7. Code: [processing.min.js](#) JavaScript file NEEDED by to include Processing code in your JavaScript program.
8. Lab: [Lab 2: Bumper Ball](#)

**Week 3:**

1. Video: [Khan Academy: Functions](#)
2. Video: [Khan Academy: Logic and if Statements](#)
3. code: [Lab 2 Solution by Zach Ward](#) Zach has portals! - he also some ellipses collisions - not perfect, but what he has works better than the unmodified circle collision.
4. code: [Lab 2 Solution by Alex Booher](#) A simple Pong game with a ***very simple*** AI - but since an interactive game and AI were not required, this is extra credit.

**Week 4 & 5:**

1. Read: Eloquent JavaScript, Chapter 1: Values, Types, and Operators
2. Video: [Khan Academy: Interactive Programs](#): Responding to Mouse clicks and Mouse movements.
3. Video: [Khan Academy: Resizing with Variables](#)
4. Lab: [Lab 3: Bumper Pool: Using mouse, collisions and physics Part 1: Board setup and cue ball shot.](#)
5. Code: [processing.min.js](#) JavaScript file NEEDED by to include Processing code in the same folder as your JavaScript program.
6. Code: [CS105\\_Lab3\\_MouseDrag.html](#)
7. Code: [CS105\\_Lab3\\_ChangeInTime.html](#)
8. Code: [CS105\\_Lab3\\_DragBall.html](#) This is a partial solution to lab 3. It is the code we developed in class on Tuesday, Feb 16. It combines the mouse drag with the change in time code. As we left it in class, it sets the X speed after click-and-drag, but does not yet set the y speed. To restart the ball on the right edge, reload the page.

**Week 6:**

1. Video: [Khan Academy: Looping](#)
2. Video: [Khan Academy: Intro to Arrays](#)
3. Read: Eloquent JavaScript, Chapter 2: Program Structure
4. Lab: [Lab 4: Bumper Pool: Using mouse, collisions and physics Part 2: Wall collision, friction, and speed vector.](#)
5. Code: [processing.min.js](#) JavaScript file NEEDED by to include Processing code in the same folder as your JavaScript program.
6. Code: [CS105\\_Lab3\\_Pool\\_Part1\\_Joel.html](#)
7. Notes: [Loops](#)
8. Code: [Factors.html](#) Example how to get input from a web page and a while loop inside the draw loop.
9. Code: [PrimeFactors.html](#) Modification of Factors.html that displays the input number's prime factors.
10. Code: [Lines1\\_LinesInALoop.html](#) Drawing Lines Example 1: Use a loop to draw many lines.
11. Code: [Lines2\\_DrawingAndErasingLines.html](#) Drawing Lines Example 2: Cycles between drawing and erasing lines.
12. Code: [Lines3\\_CrossingSetsOfLines.html](#) Drawing Lines Example 3: Draws a crossing set of lines in different colors. Then erases the lines and starts over.
13. Code: [Lines4\\_100sOfColors.html](#) Drawing Lines Example 4: Drawing lines smooth color changes.
14. Code: [Lines5\\_StringArt\\_SingleCorner.html](#) Drawing Lines Example 5: Line art that draws straight lines that have the illusion of curving.

15. Code: [Lines6\\_StringArt\\_FourCorners.html](#) Drawing Lines Example 6: Cool line art that draws 8 lines each frame.
16. Lab: [Lab 5: Line Art](#)
17. Code: [LineArt\\_Anjuli\\_Kvamme.html](#) Lab 5 Line Art solution by Anjuli Kvamme

## Week 7 & 8:

1. Midterm exam on Thursday, March 10.
2. Lab: [Lab 6: Publishing a Webpage.](#)
3. Code: [Lines7\\_StringArt\\_Slider.html](#) Example JavaScript animation that responds to slider movements.
4. Notes: [Midterm Exam Review](#)

## Week 9:

1. Exam: [Midterm Exam: 2016 Spring](#)
2. Code: [MidtermExam\\_2016\\_Spring.html](#) Exam questions in a single program.
3. Code: [ColorChooser.html](#) Example showing 3 sliders used to set the Red, Green and Blue value of the canvas background.
4. Code: [CS105\\_Lab7\\_BouncingBalls.html](#) This sample program combines much of what we did in the past to draw two balls that move with friction. They bounce off the walls and off each other. Also, moving the mouse over a ball, gives the ball a kick. This is a great starting place for lab 7. All you need to do is to add 98 balls and you are done!
5. Lab: [Lab 7:A Hundred Bouncing Balls on the Web.](#)
6. Code: <http://www.unm.edu/~botelloc/home.html> Cin's Milkyway: JavaScript Website by Cindy Botello.
7. Code: <http://www.unm.edu/~boohera/home.html> Art from Music: JavaScript Website by Alexander Booher.

## Week 10:

1. Code: [WalkAndTurn.html](#) Starting point for lab 8.
2. Notes: [Computer Tour](#)

## Week 11 & 12:

1. Code: [CryptogramGame.html](#) Starting point for lab 9.
2. Lab: [Cryptogram Game: Requirements and Algorithms](#)
3. w3schools: [Creating a drop-down menu using the HTML <select> Tag](#)
4. w3schools: [How to get the index and value selected in a drop-down menu](#)
5. Code: [ElasticCollision.html](#) Elastic Collision Example

## Week 13 & 14:

1. Web: [Khan Academy: Index of all Khan Acamemty's Computer Programming Courses](#)
2. Web: [Khan Academy: Intro to HTML/CSS: Making Web Pages.](#) Text emphasis, HTML Images, HTML Tables, CSS font familiew and font sizes HTML connection to JavaScript (Selecting by id),
3. Web: [Khan Academy: Advanced JS: Games & Visualizations.](#) Buttons, 3D Shapes, Making a Memory Game, Objects, Object Constructors, Object member fields (this.) and Object member functions (.prototype.)
4. Code: [MemoryGame.html](#) This version has a getImage(filename) function that looks in the current directory for the file and it also has a getKhanImage(filename) function that gets the image form the Khan Academy webpage.
5. image: [ladybug.png](#)

**Week 15:**

1. Code: [Flapper.html](#) Image Rotation Example: Flapper.
2. image: [flapperLeft.png](#)