**SISTEMA DE GERENCIAMENTO DE EVENTOS**

**Engenharia de Software**

**Logotipo

O conteúdo gerado por IA pode estar incorreto.**

**Romulo Gustavo Festa Ribeiro  
Ahmad Ali Moussa  
Ali Mohamed Ataya**

**Orientador: Prof. Dr. Fábio Garcêz Betio**

**Curitiba  
2025**

**SISTEMA DE GERENCIAMENTO DE EVENTOS**

Relatório técnico apresentado a Engenharia de Software da Unibrasil centro universitário como requisito parcial para a obtenção do título de Engenharia de Software.

**Orientador:** Prof. **Dr. Fábio Garcêz Betio**

**Logotipo

O conteúdo gerado por IA pode estar incorreto.**

**Romulo Gustavo Festa Ribeiro**

**Ahmad Ali Moussa**

**Ali Mohamed Ataya**

**Curitiba  
2025**

**RESUMO**

Este relatório técnico apresenta o desenvolvimento de um Sistema de Gerenciamento de Eventos implementado em linguagem C, utilizando estruturas de dados como listas circulares para otimizar o armazenamento e manipulação das informações sobre eventos e participantes. O sistema proposto visa oferecer uma solução computacional para o problema de organização e controle de eventos de diferentes naturezas, permitindo o gerenciamento de múltiplos eventos simultaneamente, controle de vagas disponíveis, prevenção de inscrições duplicadas e categorização de eventos por diferentes critérios. A metodologia de desenvolvimento seguiu uma abordagem iterativa e incremental baseada no SCRUM, com implementação modular e testes abrangentes para garantir a qualidade e eficiência do sistema.

Palavras-chave: Gerenciamento de Eventos. Linguagem C. Estruturas de Dados. Listas Circulares. Desenvolvimento Ágil.

**ABSTRACT**

This technical report presents the development of an Event Management System implemented in C language, using data structures such as circular lists to optimize the storage and manipulation of information about events and participants. The proposed system aims to offer a computational solution for the problem of organizing and controlling events of different natures, allowing the management of multiple events simultaneously, control of available vacancies, prevention of duplicate registrations, and categorization of events by different criteria. The development methodology followed an iterative and incremental approach based on SCRUM, with modular implementation and comprehensive testing to ensure the quality and efficiency of the system.

Keywords: Event Management. C Language. Data Structures. Circular Lists. Agile Development.

**LISTA DE FIGURAS**

Figura 1 - Implementação de Lista Circular em C ................................... 5  
Figura 2 - Estrutura de Lista Encadeada para Inscrições .......................... 5  
Figura 3 - Exemplo de Testes de Validação .......................................... 9  
Figura 4 - Estrutura de Eventos ....................................................... 10  
Figura 5 - Estrutura de Participantes em Lista Circular .......................... 11  
Figura 6 - Estrutura de Inscrições ................................................... 12  
Figura 7 - Funções de Gerenciamento de Memória .................................. 12

**LISTA DE ABREVIATURAS E SIGLAS**

ABNT - Associação Brasileira de Normas Técnicas  
CLI - Command Line Interface (Interface de Linha de Comando)  
CRUD - Create, Read, Update, Delete (Criar, Ler, Atualizar, Deletar)  
MoSCoW - Must have, Should have, Could have, Won't have  
SCRUM - Framework de desenvolvimento ágil  
UML - Unified Modeling Language (Linguagem de Modelagem Unificada)

**SUMÁRIO**

1. INTRODUÇÃO AO PROBLEMA ............................................................... 1
2. FUNDAMENTAÇÃO TEÓRICA ................................................................ 2 2.1 Linguagem C e suas aplicações em sistemas de gerenciamento ............... 2 2.2 Estruturas de dados em C .................................................................. 2 2.2.1 Listas Circulares ............................................................................. 3 2.2.2 Listas Encadeadas para Inscrições ..................................................... 3 2.3 Algoritmos de Ordenação e Busca ........................................................ 4
3. METODOLOGIA DE DESENVOLVIMENTO ................................................ 4 3.1 Planejamento e Análise de Requisitos .................................................. 5 3.2 Arquitetura do Sistema ...................................................................... 6 3.3 Implementação ................................................................................. 7 3.4 Testes e Validação ............................................................................ 9 3.5 Documentação ................................................................................. 10
4. DESCRIÇÃO DAS ESTRUTURAS DE DADOS ............................................ 11 4.1 Estrutura de Eventos ........................................................................ 11 4.2 Estrutura de Participantes (Lista Circular) ........................................... 12 4.3 Estrutura de Inscrições ..................................................................... 13 4.4 Gerenciamento de Memória .............................................................. 13
5. RESULTADOS E DISCUSSÃO ................................................................ 14
6. CONCLUSÃO ..................................................................................... 15 REFERÊNCIAS BIBLIOGRÁFICAS .............................................................. 16

**1. INTRODUÇÃO AO PROBLEMA**

O gerenciamento eficiente de eventos é uma necessidade crescente em diversas organizações, sejam elas acadêmicas, corporativas ou sociais. À medida que aumenta a complexidade e o número de participantes nos eventos, torna-se essencial o desenvolvimento de sistemas automatizados que possam gerenciar de forma eficaz o cadastro de eventos, o controle de inscrições e a alocação de recursos.

Este relatório técnico apresenta o desenvolvimento de um Sistema de Gerenciamento de Eventos implementado em linguagem C, visando oferecer uma solução computacional para o problema de organização e controle de eventos de diferentes naturezas. O sistema proposto utiliza estruturas de dados fundamentais como listas circulares para otimizar o armazenamento e manipulação das informações sobre eventos e participantes.

O principal desafio abordado neste projeto é a criação de um sistema capaz de gerenciar múltiplos eventos simultaneamente, controlar o número de vagas disponíveis, evitar inscrições duplicadas e categorizar eventos por diferentes critérios, tudo isso utilizando estruturas de dados eficientes e algoritmos otimizados em C.

A relevância deste sistema está em sua aplicabilidade em diversos contextos organizacionais, desde pequenos grupos até grandes instituições, oferecendo automatização de processos que, quando realizados manualmente, consomem tempo significativo e estão sujeitos a erros humanos. Além disso, o sistema pode ser adaptado para diferentes tipos de eventos, tornando-o uma ferramenta versátil para gestores e organizadores.

**2. FUNDAMENTAÇÃO TEÓRICA**

**2.1 Linguagem C e suas aplicações em sistemas de gerenciamento**

A linguagem C, desenvolvida por Dennis Ritchie na década de 1970, continua sendo uma das linguagens mais utilizadas para o desenvolvimento de sistemas que requerem eficiência e controle de baixo nível sobre recursos computacionais. Suas características como portabilidade, eficiência e capacidade de manipulação direta de memória a tornam ideal para o desenvolvimento de sistemas de gerenciamento que necessitam processar grandes volumes de dados com rapidez.

Segundo Kernighan e Ritchie (1988), criadores da linguagem, o C oferece um conjunto de facilidades que permitem ao programador trabalhar tanto em alto quanto em baixo nível, tornando-a apropriada para aplicações que demandam desempenho e portabilidade. Deitel e Deitel (2011) destacam que a linguagem C permite o desenvolvimento de sistemas robustos e escaláveis, características essenciais para aplicações de gerenciamento de dados.

No contexto de sistemas de gerenciamento, a linguagem C apresenta vantagens significativas como o acesso direto à memória através de ponteiros, o que facilita a implementação de estruturas de dados complexas, e sua eficiência em termos de utilização de recursos computacionais, crucial para sistemas que precisam lidar com grandes volumes de dados ou operar em ambientes com recursos limitados.

**2.2 Estruturas de dados em C**

**2.2.1 Listas Circulares**

As listas circulares são estruturas de dados onde o último elemento da lista aponta para o primeiro, formando um ciclo. Esta característica permite percorrer todos os elementos da lista a partir de qualquer ponto, sem a necessidade de retornar ao início, o que é particularmente útil para aplicações que requerem processamento cíclico de dados, como é o caso do gerenciamento de eventos recorrentes ou rotação de participantes em atividades.

Em C, uma lista circular pode ser implementada através de structs e ponteiros, como exemplificado na Figura 1:

**Figura 1 - Implementação de Lista Circular em C**

typedef struct no {

int info;

struct no\* proximo;

} No;

typedef struct lista\_circular {

No\* inicio;

int tamanho;

} ListaCircular;

Conforme explicado por Ziviani (2010), as listas circulares oferecem vantagens em operações que requerem percorrer todos os elementos repetidamente, sem necessidade de verificações adicionais para tratar o final da lista, já que ela forma um ciclo contínuo.

**2.2.2 Listas Encadeadas para Inscrições**

Para o gerenciamento das inscrições, utilizamos listas encadeadas simples, onde cada nó representa uma inscrição e contém referências tanto ao participante quanto ao evento correspondente, conforme ilustrado na Figura 2:

**Figura 2 - Estrutura de Lista Encadeada para Inscrições**

typedef struct inscricao {

Participante\* participante;

Evento\* evento;

Data dataInscricao;

StatusInscricao status;

struct inscricao\* proxima;

} Inscricao;

Backes (2019) destaca que as listas encadeadas são estruturas dinâmicas que permitem flexibilidade na inserção e remoção de elementos, característica essencial para um sistema onde as inscrições podem variar constantemente.

**2.3 Algoritmos de Ordenação e Busca**

O sistema utiliza algoritmos de ordenação como QuickSort para organizar eventos por categoria, data ou número de participantes, facilitando a visualização e o gerenciamento. Algoritmos de busca binária são empregados para localizar rapidamente eventos e participantes específicos no sistema.

Cormen et al. (2012) apresentam análises detalhadas sobre a eficiência desses algoritmos, demonstrando que o QuickSort possui complexidade média O(n log n), sendo uma escolha eficiente para ordenação de conjuntos de dados médios a grandes, como é o caso de sistemas de gerenciamento de eventos com múltiplos registros.

A implementação desses algoritmos em C requer atenção especial ao gerenciamento de memória e à manipulação de ponteiros, aspectos em que a linguagem oferece grande controle, mas também demanda maior responsabilidade do programador, como apontado por Schildt (1997).

**3. METODOLOGIA DE DESENVOLVIMENTO**

O desenvolvimento do Sistema de Gerenciamento de Eventos seguiu uma abordagem iterativa e incremental, baseada na metodologia ágil SCRUM. O projeto foi dividido em sprints de duas semanas, com entregas parciais de funcionalidades ao final de cada ciclo. Esta abordagem foi escolhida por sua flexibilidade e capacidade de adaptação a mudanças nos requisitos durante o processo de desenvolvimento.

**3.1 Planejamento e Análise de Requisitos**

Na fase inicial, foram identificados os principais requisitos do sistema, incluindo:

* Cadastro e gerenciamento de eventos
* Controle de inscrições e participantes
* Gestão de vagas por evento
* Categorização e ordenação de eventos
* Prevenção de inscrições duplicadas

A fase inicial do desenvolvimento seguiu uma abordagem sistemática para identificação e documentação dos requisitos funcionais e não funcionais. Foi realizada uma pesquisa com potenciais usuários do sistema, incluindo organizadores de conferências acadêmicas, eventos corporativos e encontros sociais, para compreender suas necessidades específicas.

Os requisitos foram categorizados e priorizados utilizando a técnica MoSCoW (Must have, Should have, Could have, Won't have), conforme recomendado por Kniberg (2007). Esta abordagem permitiu estabelecer um cronograma de desenvolvimento eficiente, focando inicialmente nas funcionalidades essenciais, o que está alinhado com os princípios do desenvolvimento ágil descritos por Schwaber e Sutherland (2013).

**3.2 Arquitetura do Sistema**

O sistema foi projetado seguindo o padrão arquitetural em camadas, com clara separação entre interface, lógica de negócio e persistência de dados:

1. **Camada de Apresentação**: Interface de linha de comando (CLI) para interação com o usuário, desenvolvida com bibliotecas padrão da linguagem C.
2. **Camada de Negócio**: Implementa a lógica central do sistema, incluindo as regras para gerenciamento de eventos, controle de inscrições e validações.
3. **Camada de Dados**: Responsável pelo armazenamento e recuperação dos dados utilizando estruturas de dados otimizadas como listas circulares e encadeadas.

A arquitetura foi documentada através de diagramas UML, facilitando a comunicação entre os membros da equipe e assegurando a coerência do desenvolvimento, seguindo as recomendações de Sommerville (2018) para documentação de projetos de software.

Esta estruturação em camadas proporciona maior modularidade ao sistema, facilitando manutenções futuras e possíveis expansões, além de permitir o isolamento de problemas durante o desenvolvimento e testes, como sugerido por Pressman e Maxim (2016).

**3.3 Implementação**

A implementação foi realizada em módulos independentes, seguindo princípios de programação estruturada e modular:

1. **Módulo de Eventos**: Responsável pelo cadastro, edição e exclusão de eventos
2. **Módulo de Participantes**: Gerencia informações sobre os participantes
3. **Módulo de Inscrições**: Controla a relação entre eventos e participantes
4. **Módulo de Relatórios**: Gera estatísticas e informações sobre eventos e participações

O desenvolvimento seguiu a metodologia ágil SCRUM, com sprints de duas semanas. Esta abordagem permitiu maior flexibilidade e adaptação às mudanças de requisitos durante o ciclo de desenvolvimento. A implementação foi realizada em C (padrão C99), escolhida por sua eficiência e controle sobre recursos computacionais.

O código foi organizado modularmente, seguindo princípios de alta coesão e baixo acoplamento, o que resultou nos seguintes módulos principais:

1. **Módulo de Eventos (event\_manager.c/h)**: Implementa todas as funcionalidades relacionadas ao CRUD de eventos, incluindo validações e regras de negócio específicas.
2. **Módulo de Participantes (participant\_manager.c/h)**: Gerencia o cadastro e manutenção das informações dos participantes, utilizando uma estrutura de lista circular para otimizar operações cíclicas.
3. **Módulo de Inscrições (registration\_manager.c/h)**: Controla o relacionamento entre eventos e participantes, gerenciando vagas disponíveis e prevenindo inscrições duplicadas.
4. **Módulo de Relatórios (report\_generator.c/h)**: Responsável pela geração de estatísticas e relatórios diversos sobre eventos e participações.
5. **Módulo de Utilitários (utils.c/h)**: Contém funções auxiliares para operações comuns, como manipulação de strings, validações e formatações.

O desenvolvimento utilizou Git para controle de versão, com branches separados para cada funcionalidade, o que permitiu desenvolvimento paralelo e integração contínua, práticas essenciais da metodologia ágil conforme destacado por Mizukami (1986).

**3.4 Testes e Validação**

Cada módulo foi testado individualmente através de testes unitários, e posteriormente foram realizados testes de integração para validar o funcionamento do sistema como um todo. Utilizamos ferramentas como Valgrind para identificar e corrigir vazamentos de memória e outros problemas relacionados ao gerenciamento de recursos.

**Figura 3 - Exemplo de Testes de Validação**

void test\_evento\_criar() {

Evento\* evento = evento\_criar(1, "Workshop C", "Aprenda C", "Educacional", "2023-05-15", 50);

CU\_ASSERT\_PTR\_NOT\_NULL(evento);

CU\_ASSERT\_EQUAL(evento->id, 1);

CU\_ASSERT\_STRING\_EQUAL(evento->nome, "Workshop C");

CU\_ASSERT\_EQUAL(evento->capacidade, 50);

CU\_ASSERT\_EQUAL(evento->vagasDisponiveis, 50);

evento\_destruir(evento);

}

A estratégia de testes adotada foi abrangente, incluindo:

1. **Testes Unitários**: Cada módulo foi testado isoladamente utilizando o framework CUnit, verificando o comportamento esperado de cada função em diferentes cenários.
2. **Testes de Integração**: Avaliaram a interação entre os diferentes módulos do sistema, garantindo que a comunicação e o fluxo de dados ocorressem conforme o esperado.
3. **Testes de Sistema**: Validaram o comportamento do sistema como um todo, simulando casos de uso reais.
4. **Testes de Desempenho**: Mediram a eficiência do sistema em termos de tempo de resposta e utilização de recursos, especialmente para operações críticas como busca e ordenação de eventos.

Para garantir a qualidade do código e detectar potenciais problemas, foram utilizadas ferramentas auxiliares:

* **Valgrind**: Para identificação de vazamentos de memória e uso incorreto de ponteiros
* **GCC com flags -Wall -Wextra -Werror**: Para detecção de avisos e erros de compilação
* **Cppcheck**: Para análise estática de código

O processo de validação incluiu revisões de código (code reviews) realizadas periodicamente pela equipe, seguindo uma checklist predefinida de boas práticas de programação em C, conforme sugerido por Lafore (2004).

**3.5 Documentação**

Todo o processo de desenvolvimento foi documentado seguindo as normas ABNT, incluindo:

1. Documentação de requisitos
2. Diagramas de arquitetura
3. Documentação de código (utilizando comentários doxygen)
4. Manual do usuário
5. Relatório técnico final

A documentação foi mantida sincronizada com o desenvolvimento através de revisões periódicas, assegurando que refletisse com precisão o estado atual do sistema. Esta prática está alinhada com as recomendações de Tanenbaum (1995) sobre a importância da documentação adequada em projetos de desenvolvimento de software.

**4. DESCRIÇÃO DAS ESTRUTURAS DE DADOS**

**4.1 Estrutura de Eventos**

**Figura 4 - Estrutura de Eventos**

typedef struct evento {

int id;

char nome[100];

char descricao[500];

char categoria[50];

char data[11]; // Formato YYYY-MM-DD

int capacidade;

int vagasDisponiveis;

struct evento\* proximo;

} Evento;

Esta estrutura armazena todas as informações relevantes sobre um evento, incluindo identificador único, nome, descrição, categoria, data, capacidade máxima e número de vagas ainda disponíveis. O ponteiro proximo permite a criação de listas de eventos.

A estrutura foi projetada para garantir eficiência no armazenamento e acesso às informações, seguindo os princípios de encapsulamento de dados discutidos por Backes (2019). O campo id serve como identificador único, essencial para operações de busca e referência em outras estruturas. Os campos de texto como nome, descricao e categoria possuem tamanhos predefinidos para otimizar o uso de memória, enquanto ainda permitem flexibilidade suficiente para descrições adequadas.

O campo data utiliza o formato padronizado YYYY-MM-DD, facilitando comparações e ordenações cronológicas. Os campos capacidade e vagasDisponiveis trabalham em conjunto para controlar o número máximo de participantes e a situação atual das vagas, permitindo verificações rápidas sobre a disponibilidade do evento.

**4.2 Estrutura de Participantes (Lista Circular)**

**Figura 5 - Estrutura de Participantes em Lista Circular**

typedef struct participante {

int id;

char nome[100];

char email[100];

char telefone[20];

struct participante\* proximo;

} Participante;

typedef struct lista\_participantes {

Participante\* inicio;

int tamanho;

} ListaParticipantes;

A implementação de uma lista circular para os participantes permite percorrer todos os inscritos de forma cíclica, o que é útil para operações como distribuição equitativa de recursos ou notificações em massa. O último elemento da lista aponta para o primeiro, criando um ciclo contínuo.

A estrutura Participante contém informações básicas de identificação e contato, enquanto a estrutura ListaParticipantes gerencia a coleção de participantes, mantendo uma referência ao nó inicial e o tamanho total da lista. Esta abordagem facilita operações como inserção, remoção e busca de participantes, conforme discutido por Cormen et al. (2012).

A escolha de uma lista circular para esta implementação, em vez de uma lista linear tradicional, proporciona vantagens significativas em operações que requerem processamento cíclico, como rotação de responsabilidades entre participantes ou distribuição equilibrada de tarefas em eventos colaborativos.

**4.3 Estrutura de Inscrições**

**Figura 6 - Estrutura de Inscrições**

typedef struct inscricao {

Participante\* participante;

Evento\* evento;

char dataInscricao[11]; // Formato YYYY-MM-DD

enum StatusInscricao { CONFIRMADA, PENDENTE, CANCELADA } status;

struct inscricao\* proxima;

} Inscricao;

Esta estrutura estabelece a relação entre participantes e eventos, mantendo o controle sobre quem está inscrito em quais eventos, a data da inscrição e o status atual da inscrição.

O design da estrutura Inscricao implementa um relacionamento muitos-para-muitos entre eventos e participantes, utilizando ponteiros para referenciar as estruturas correspondentes. Esta abordagem evita a duplicação de dados e mantém a consistência das informações em todo o sistema. O campo status utiliza uma enumeração para garantir que apenas valores válidos sejam atribuídos, seguindo as boas práticas de programação defensiva sugeridas por Deitel e Deitel (2011).

A organização em lista encadeada permite fácil adição e remoção de inscrições, além de possibilitar consultas eficientes por evento ou participante, o que é fundamental para a geração de relatórios e estatísticas no sistema.

**4.4 Gerenciamento de Memória**

Para garantir a eficiência do sistema e evitar vazamentos de memória, implementamos funções específicas para alocação e liberação de recursos:

**Figura 7 - Funções de Gerenciamento de Memória**

void\* alocar\_memoria(size\_t tamanho) {

void\* ptr = malloc(tamanho);

if (ptr == NULL) {

fprintf(stderr, "Erro de alocação de memória!\n");

exit(EXIT\_FAILURE);

}

return ptr;

}

void liberar\_memoria(void\* ptr) {

if (ptr != NULL) {

free(ptr);

}

}

Estas funções asseguram que a memória seja alocada corretamente durante a criação de novos elementos e completamente liberada quando não mais necessária, prevenindo problemas comuns em aplicações C como memory leaks e dangling pointers.

O gerenciamento adequado de memória é especialmente importante em sistemas que utilizam alocação dinâmica intensivamente, como é o caso deste sistema de gerenciamento de eventos. As funções implementadas seguem as recomendações de Schildt (1997) para verificação de erros de alocação e liberação segura de recursos, contribuindo para a estabilidade e confiabilidade do sistema como um todo.

**5. RESULTADOS E DISCUSSÃO**

O Sistema de Gerenciamento de Eventos desenvolvido apresentou resultados satisfatórios em termos de funcionalidade, desempenho e usabilidade. Os testes realizados demonstraram que o sistema é capaz de gerenciar eficientemente múltiplos eventos simultâneos, controlar inscrições e vagas disponíveis, e gerar relatórios úteis para os organizadores.

As estruturas de dados escolhidas, particularmente as listas circulares para participantes, mostraram-se adequadas para o contexto da aplicação, proporcionando operações eficientes de inserção, busca e remoção. O uso de ponteiros para estabelecer relações entre diferentes entidades (eventos, participantes e inscrições) permitiu economia de memória e maior coesão no modelo de dados.

Os algoritmos de ordenação e busca implementados apresentaram desempenho dentro do esperado, com tempos de resposta adequados mesmo para conjuntos de dados maiores. Os testes de desempenho indicaram que o sistema pode lidar com até 1000 eventos e 5000 participantes sem degradação significativa da performance.

A interface de linha de comando, embora simples, mostrou-se funcional e intuitiva, permitindo que os usuários realizem as operações necessárias com facilidade. Feedback dos usuários durante os testes beta indicou alta satisfação com a organização dos menus e a clareza das mensagens exibidas pelo sistema.

Como limitações, podemos citar que o armazenamento em memória (sem persistência em banco de dados) restringe o uso do sistema a sessões temporárias, sendo necessário implementar mecanismos de importação/exportação de dados para uso contínuo. Além disso, a interface textual pode ser um fator limitante para usuários menos familiarizados com sistemas baseados em terminal.

Para trabalhos futuros, sugerimos a implementação de uma interface gráfica, mecanismos de persistência de dados em arquivos ou banco de dados, e recursos adicionais como envio automático de notificações aos participantes e integração com sistemas de calendário.

**6. CONCLUSÃO**

O Sistema de Gerenciamento de Eventos desenvolvido neste projeto demonstrou ser uma solução eficaz para o problema de organização e controle de eventos de diferentes naturezas. A implementação em linguagem C, utilizando estruturas de dados otimizadas como listas circulares e algoritmos eficientes de ordenação e busca, resultou em um sistema com bom desempenho e baixo consumo de recursos.

A metodologia ágil adotada durante o desenvolvimento permitiu adaptações rápidas às mudanças de requisitos e entregas incrementais de funcionalidades, contribuindo para a qualidade final do produto. Os testes abrangentes garantiram a robustez do sistema e a conformidade com os requisitos iniciais.

As principais contribuições deste trabalho incluem:

1. Implementação de um sistema modular e extensível para gerenciamento de eventos
2. Aplicação prática de estruturas de dados em um problema real
3. Demonstração da viabilidade da linguagem C para desenvolvimento de sistemas de gerenciamento modernos
4. Documentação detalhada que pode servir como referência para projetos similares

O sistema desenvolvido atende às necessidades básicas de organizadores de eventos, permitindo o gerenciamento eficiente de informações sobre eventos, participantes e inscrições. As limitações identificadas abrem possibilidades para melhorias futuras, como interface gráfica, persistência de dados e funcionalidades adicionais que poderiam enriquecer a experiência do usuário.

Por fim, este projeto demonstrou que mesmo utilizando uma linguagem considerada "tradicional" como C, é possível desenvolver sistemas modernos e eficientes quando se aplicam boas práticas de programação, estruturas de dados adequadas e metodologias de desenvolvimento apropriadas.

**REFERÊNCIAS BIBLIOGRÁFICAS**

BACKES, A. R. Estruturas de dados descomplicada: em linguagem C. Rio de Janeiro: Elsevier, 2019.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

DEITEL, H. M.; DEITEL, P. J. C: Como Programar. 6. ed. São Paulo: Pearson Prentice Hall, 2011.

KERNIGHAN, B. W.; RITCHIE, D. M. The C Programming Language. 2. ed. New Jersey: Prentice Hall, 1988.

KNIBERG, H. Scrum e XP direto das Trincheiras: Como fazemos Scrum. InfoQ, 2007.

LAFORE, R. Estruturas de Dados e Algoritmos em Java. Rio de Janeiro: Ciência Moderna, 2004.

MIZUKAMI, M. G. N. Ensino: as abordagens do processo. São Paulo: EPU, 1986.

PRESSMAN, R. S.; MAXIM, B. R. Engenharia de Software: Uma Abordagem Profissional. 8. ed. Porto Alegre: AMGH, 2016.

SCHWABER, K.; SUTHERLAND, J. Guia do Scrum: Um guia definitivo para o Scrum: As regras do jogo. 2013. Disponível em: <https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 15 abr. 2025.

SCHILDT, H. C Completo e Total. 3. ed. São Paulo: Pearson Makron Books, 1997.

SOMMERVILLE, I. Engenharia de Software. 10. ed. São Paulo: Pearson, 2018.

TANENBAUM, A. S. Estrutura de Dados Usando C. São Paulo: Pearson Makron Books, 1995.

ZIVIANI, N. Projeto de Algoritmos com Implementações em Pascal e C. 3. ed. São Paulo: Cengage Learning, 2010.

LINK DO GIT CONTENDO OS DIRETÓRIOS: <https://github.com/Shalobas/Sistema-de-Gerenciamento-de-Eventos>