Desenvolvimento de Quiz em Linguagem C: Uma Abordagem Estruturada

Bruno Salvador, Paulo Lucas
Dr. Aldo Henrique Mendes (Orientador)
Centro Universitário Euro-Americano (Unieuro), Brasília, DF, Brasil

Resumo—Este trabalho apresenta o desenvolvimento de um jogo interativo do tipo quiz em linguagem C, voltado a testar conhecimentos sobre a própria linguagem. O sistema mantém um banco de 30 questões e seleciona aleatoriamente 10 a cada rodada, garantindo dinamismo. A arquitetura aplica encapsulamento via struct, alocação dinâmica com malloc e free, passagem por referência por ponteiros e tratamento rigoroso do buffer de teclado. O programa demonstrou robustez nos testes, validando entradas em maiúsculas e minúsculas pela aritmética da Tabela ASCII e prevenindo vazamentos de memória com a liberação explícita ao final da execução. O projeto consolida boas práticas no gerenciamento manual de memória.

Palavras-chave—Linguagem C; Quiz; Structs; Alocação dinâmica; Buffer de teclado.

I. Introdução

A linguagem C é amplamente reconhecida pelo alto desempenho e pelo controle granular oferecido sobre os recursos do sistema [1]. Em disciplinas de Estrutura de Dados, compreender como agrupar informações heterogêneas e manipular endereços de memória é fundamental para o desenvolvimento de softwares eficientes [2].

O presente projeto tem como objetivo desenvolver um jogo interativo do tipo quiz, focado em testar conhecimentos sobre a linguagem C. O programa visa não apenas oferecer uma interface funcional em terminal, mas aplicar na prática conceitos de programação estruturada, evitando o uso de variáveis globais soltas.

O sistema abriga um banco de 30 questões, embaralhado a cada rodada, das quais apenas 10 são selecionadas. O desenvolvimento exigiu alocação dinâmica de memória, passagem de parâmetros por referência e tratamento do fluxo de entrada de dados do teclado.

II. Metodologia

A arquitetura do código foi construída sobre quatro pilares principais: estruturação de dados, alocação dinâmica, ponteiros com passagem por referência, e tratamento do buffer de entrada.

A. Estruturação de Dados

Os dados foram encapsulados via typedef struct em uma estrutura denominada Pergunta, contendo um vetor para o enunciado, uma matriz para três alternativas de resposta e um caractere para o gabarito [2].

B. Alocação Dinâmica

O programa não instancia todas as questões simultaneamente. Aloca espaço estritamente para as 10 perguntas da rodada com malloc e o operador sizeof(Pergunta). Ao final da execução, free é acionado para devolver a memória ao sistema operacional, prevenindo vazamentos [1].

C. Ponteiros e Passagem por Referência

A função modular fazerPergunta exibe e corrige as questões. Para evitar a cópia de matrizes, utilizou-se passagem por referência recebendo o ponteiro Pergunta *p. Os membros são acessados pelo operador seta (->), e a chamada na main envia o endereço com o operador & [1].

D. Entrada e Limpeza de Buffer

A interação utiliza scanf para leitura de um caractere, com subtração de 32 do valor ASCII para validar tanto respostas em maiúsculas quanto em minúsculas. A função limparBuffer descarta caracteres residuais e quebras de linha por meio de laço while com getchar, prevenindo que o fluxo do programa pule leituras [2].

III. Resultados

O sistema executou corretamente todas as funcionalidades projetadas. A Figura 1 apresenta a tela inicial e a primeira pergunta embaralhada. A Figura 2 ilustra a entrada de dados e o feedback de acerto/erro com validação de caractere. A Figura 3 mostra a saída final do programa, com pontuação obtida e menu de repetição.

Os testes de execução verificaram a alocação correta para 10 perguntas por rodada, a liberação de memória sem vazamentos e a robustez do tratamento de buffer, que evitou saltos indevidos no fluxo de leitura.

IV. Discussão

A implementação evidenciou que o controle manual de memória oferece eficiência superior a métodos puramente estáticos. O uso de structs garantiu escalabilidade do banco de questões, alinhando-se aos princípios organizacionais discutidos por Schildt [2]. A alocação dinâmica reservou espaço estrito para as 10 perguntas da rodada, otimizando o consumo de memória.

A passagem por referência via ponteiros evitou cópias desnecessárias de dados e reduziu o consumo da pilha, prática defendida por Kernighan e Ritchie [1]. A função limparBuffer mitigou falhas clássicas de leitura do scanf. O

controle manual de memória, embora exija rigor para evitar vazamentos (garantido pelo free), resultou em software estável e robusto.

V. Conclusão

O projeto demonstrou na prática a importância do gerenciamento consciente da memória e do fluxo de dados em C. O uso de structs tornou o código legível e escalável, enquanto ponteiros e alocação dinâmica mostraram-se indispensáveis para criar aplicações eficientes. O tratamento do buffer de teclado reforçou a necessidade de projetar softwares tolerantes a falhas na interação humana, resultando em uma aplicação robusta e funcional.

Referências

[1] B. W. Kernighan e D. M. Ritchie, C: A Linguagem de Programação, 2ª ed. Rio de Janeiro: Campus, 1989.

[2] H. Schildt, C Completo e Total, 3ª ed. São Paulo: Makron Books, 1997.

[3] P. Deitel e H. Deitel, C: Como Programar, 6ª ed. São Paulo: Pearson, 2011.