**UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL**

GABRIEL DE CAMPOS GONSALVES
RUAN SARTÓRIO THOMAZ
HUGO KUSHI

# RELATÓRIO ALGORITMOS E PROGRAMAÇÃO 2:

Programa Show do Milhão

**CAMPO GRANDE, MS**
**OUTUBRO, 2025**

**UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL**

GABRIEL DE CAMPOS GONSALVES
RUAN SARTÓRIO THOMAZ
HUGO KUSHI

# RELATÓRIO ALGORITMOS E PROGRAMAÇÃO 2:

Programa Show do Milhão

Trabalho referente ao segundo semestre do curso de Sistemas de Informação da disciplina de Algoritmos e Programação 2 ministrada pelo profº Dr. Anderson Bessa da Costa.

**CAMPO GRANDE, MS**
**OUTUBRO, 2025**

## Introdução

O grupo foi formado por 3 membros, cada membro ficou responsável por implementar um trecho do código “Show do Milhão” na linguagem C. Inicialmente, o grupo se reuniu presencialmente no laboratório da FACOM para dividir as tarefas, planejar o calendário de metas a serem cumpridas até a data da entrega do trabalho e discutir, em um primeiro levantamento de requisitos, as funções que deveriam ser utilizadas no programa.

A comunicação entre o grupo era feita de forma presencial e online, por meio das plataformas Whatsapp, Discord e Google Meet. O código era de livre acesso entre todos os membros e estava disponível em um repositório no GitHub. Para o desenvolvimento do código, o grupo utilizou como embasamento teórico os fóruns Stack Overflow e Cplusplus além do livro Algoritmos em Linguagem C do autor Paulo Feofillof.

## Descrição do Código

Nos tópicos a seguir, são apresentados os principais aspectos do algoritmo, com ênfase nos pontos que requerem maior atenção.

**1) Inclusão de bibliotecas:**

```c
#include <stdio.h>   // entrada e saída (printf, scanf, fopen, fread etc)
#include <stdlib.h>  // funções utilitárias (malloc, rand, srand)
#include <time.h>    // para usar time() com srand()
#include <ctype.h>   // para funções de manipulação de caracteres
```

**A Biblioteca <ctype.h>** foi utilizada para a função `tolower`, que converte as letras maiúsculas para minúsculas. As referências seguiram o padrão da documentação do guia Cplusplus - tolower.

**2) Definindo a estrutura da pergunta (struct):**

```c
struct pergunta{
    char nivel;
    char descricao[200];
    char alt[4][30];
    char alt_correta;
};

typedef struct pergunta Pergunta;
```

**char nivel**: nível da pergunta (fácil, médio, difícil, super difícil).

**char descricao[200]**: enunciado da pergunta.

**alt [4][30]**: array de char, define quatro alternativas, cada uma com até 30 bytes.

**alt_correta**: tipo char que guarda a letra correta (‘a’, ‘b’, ‘c’ ou ‘d’).

**typedef**: serve pra poder usar `‘Pergunta’` como nome de tipo em vez de `‘struct Pergunta’`.

**3) Função perguntar( ):**

A função exibe uma pergunta para o jogador, mostra o total em dinheiro acumulado e as ajudas disponíveis, recebe a resposta e retorna um código indicando o resultado da jogada.

```
int perguntar(Pergunta p, int *pulos, int *ajuda_plateia, int
*ajuda_universitarios, int *ajuda_cartas, int valor_ganho, int
contador_pergunta)
```

Os retornos - `return` possíveis são:

- `1 -> resposta correta;`
- `0 -> resposta errada;`
- `2 -> usou uma ajuda (repete a pergunta);`
- `4 -> jogador decidiu parar;`
- `5 -> jogador pulou a pergunta.`

**4) Controle de ajudas:**

Cada ajuda consome uma unidade do seu respectivo contador que foi passado para a função por referência, com *.

```
if (*ajuda_plateia > 0) {
    *ajuda_plateia = *ajuda_plateia - 1;
```

Assim, as ajudas não são infinitas e sempre serão decrementadas quando utilizadas!

**5) Verificação da resposta:**

```
if (resposta == p.alt_correta) {
    return 1;
}
return 0;
```

Se a alternativa digitada for igual à correta `-> p.alt_correta`, o jogador acerta. Caso contrário, o jogo termina e o jogador não leva nenhum prêmio.

**6) Abrir as perguntas do arquivo:**

Esse trecho do código abre o arquivo “`perguntas.dat`” no modo de leitura - `read` para binários.

```
ptr_f = fopen("perguntas.dat", "rb");
total_perguntas = fread(perguntas, sizeof(Pergunta), 70, ptr_f);
fclose(ptr_f);
```

Aqui as 70 perguntas são lidas, cada uma com um tamanho em bytes (tamanho esse definido na descrição do trabalho). A variável `Perguntas` é um vetor que disponibiliza as questões na memória para ser usada no jogo. A função `fclose( )` fecha o arquivo, otimizando o programa.

**7) Escolha aleatória de perguntas:**

```
do {
    ind_pergunta = inicio + rand( ) % (fim - inicio + 1);
} while (repete[ind_pergunta]);
```

Esse laço garante que a pergunta sorteada será de dentro do intervalo correto `(inicio e fim)` e que nenhuma pergunta se repita durante a execução. Após usada, a posição é marcada com: `repete[ind_pergunta] = 1;`

**8) Estrutura dos níveis:**

O jogo possui 4 níveis:

- `Nível 1 (fácil) -> perguntas de 0-19`
- `Nível 2 (médio) -> perguntas de 20-39`
- `Nível 3 (difícil) -> perguntas de 40-59`
- `Nível 4 (super-difícil) -> perguntas 60-69`

```
for (int nivel = 1; nivel <= 4; nivel++) {
    if (nivel == 1) { inicio = 0; fim = 19; }
    else if (nivel == 2) { inicio = 20; fim = 39; }
    ...
```

Cada nível tem 5 perguntas, exceto o último, que tem a grande pergunta final.

**9) Sistema de pontuação e recompensa:**

Após cada acerto os valores aumentam progressivamente conforme o nível, se o jogador acertar todas as perguntas recebe R$ 1.000.000.

```
switch (nivel) {
    case 1:
        if (i == 0) valor_ganho = 1000;
        else if (i == 1) valor_ganho = 2000;
        ...
}
```

Cada `case(1, 2, 3, …)` representa o nível ao qual a pergunta pertence e as condicionais `if e else if` definem o valor ganho por acerto, o `break` foi utilizado para sair da condicional. O valor ganho na premiação por pergunta segue a ordem da seguinte tabela:

| | Acertar | Parar |
|---|---|---|
| 1 | R$ 1 mil | R$ 0 |
| 2 | R$ 2 mil | R$ 1 mil |
| 3 | R$ 3 mil | R$ 2 mil |
| 4 | R$ 4 mil | R$ 3 mil |
| 5 | R$ 5 mil | R$ 4 mil |
| 6 | R$ 10 mil | R$ 5 mil |
| 7 | R$ 20 mil | R$ 10 mil |
| 8 | R$ 30 mil | R$ 20 mil |
| 9 | R$ 40 mil | R$ 30 mil |
| 10 | R$ 50 mil | R$ 40 mil |
| 11 | R$ 100 mil | R$ 50 mil |
| 12 | R$ 200 mil | R$ 100 mil |
| 13 | R$ 300 mil | R$ 200 mil |
| 14 | R$ 400 mil | R$ 300 mil |
| 15 | R$ 500 mil | R$ 400 mil |
| 16 | R$ 1 milhão | R$ 500 mil |

**10) Encerramento do jogo e do algoritmo:**

O jogador tem 3 opções para finalizar o jogo e consequentemente encerrar o nosso algoritmo:

- `Errar a pergunta e perder tudo :-(`
- `Parar o jogo e levar o prêmio acumulado :-/`

- Acertar a última pergunta e se tornar um milionário :-)

Quando ocorre um desses casos o algoritmo imprime uma mensagem correspondente a escolha do jogador, como no exemplo abaixo no qual a pergunta do milhão foi respondida corretamente.

```
printf("\n\n===== Parabens! Voce venceu o jogo! =====\n");
printf("Total premio: R$ %d\n\n", valor_ganho);
```

## Ambiente de Desenvolvimento

O código fonte foi compartilhado por meio do GitHub para a interação de todos os membros do grupo. O algoritmo foi escrito em computadores com os sistemas operacionais Windows e Linux Mint, através das IDE's Visual Studio Code e Embarcadero Dev-C++ com a utilização do compilador GCC - (GNU Compiler Collection).

```
O Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\Users\gabri> gcc --version
gcc.exe (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

A versão do GCC no Windows é a 6.3.0 obtida através do MinGW.org e 13.3.0 no ambiente do Linux Mint. O código foi implementado na linguagem C em um arquivo com a extensão ".c" e compilado por meio do terminal, gerando um arquivo executável com a extensão ".exe" no sistema Windows e ".out" no ambiente Linux.

## Decisões de Implementação

**1) Passagem das variáveis por referência:** Os contadores de ajuda e pulos são passados como ponteiros para que a função `perguntar( )` possa modificar os seus valores

.

```
int perguntar(Pergunta p, int *pulos, int *ajuda_plateia, int
*ajuda_universitarios, int *ajuda_cartas, ...)
```

Decidimos utilizar desse modo, pois a função precisa diminuir o número de ajudas e pulos disponíveis a cada uso, passar por valor impediria essa atualização no escopo principal.

**2) Uso de inteiros para o retorno da função `perguntar( )`:** Simplificou o gerenciamento do jogo no loop principal e tornou o programa mais organizado e legível. Isso evitou a

criação de múltiplas funções para cada tipo de resposta, facilitou o controle do fluxo na `main( )` e manteve o código mais modular e fácil de entender

**3) Uso de `srand(time(NULL))`:** O grupo decidiu utilizar a função `srand( )` junto a biblioteca `<time.h>` para gerar os números de forma aleatória a cada execução, evitando que o jogo apresente as mesmas perguntas ao jogador. A função `time,` segundo o fórum cplusplus, retorna o número de segundos desde 00:00 horas de 1° de janeiro de 1970 UTC (ou seja, o carimbo de data/hora unix atual).

**4) Verificando os erros na abertura do arquivo:**

```
if (ptr_f == NULL) {
    perror("Erro ao abrir o arquivo");
    return 1;
}
```

Isso evita que o programa tente ler um arquivo inexistente e prosseguir, o uso da função `perror( )` serve para mostrar uma mensagem mais clara sobre o erro ocorrido, aumentando a confiabilidade no programa.

**5) Uso de vetores para representar as alternativas e as ajudas:** Decidimos representar as alternativas e a contagem dos votos com vetores.

```
char letras[] = {'a', 'b', 'c', 'd'};
int votos[4] = {0};
```

Isso simplificou o acesso às alternativas em loops, facilitou as operações matemáticas (ex: contagem de votos) e substituiu múltiplas variáveis separadas por uma estrutura mais compacta.

**6) Controle dos limites de cada nível:** Ao definir faixas específicas de dificuldade garantimos que cada nível utilize apenas perguntas do seu conjunto correspondente.

```
if (nivel == 1) { inicio = 0; fim = 19; }
else if (nivel == 2) { inicio = 20; fim = 39; }
```

Além disso, esse método permite expandir facilmente o banco de perguntas e facilita a manutenção no jogo.

**7) Manutenção da quantidade ao pular uma pergunta:**

```
else if (resultado == 5) {
    i--;
}
```

Isso decrementa o contador do laço `(i-)`, garantindo que o jogador não perca uma questão ao pular, mantendo a quantidade total de perguntas por nível.

**8) Implementação de um sistema de pontuação fixa por pergunta:** Atribui prêmios fixos para cada pergunta, sem cálculos dinâmicos com **+=**.

```
if (i == 0) valor_ganho = 1000;
else if (i == 1) valor_ganho = 2000;
...
```

Além de simplificar a implementação, tornou mais fácil o entendimento do código para o grupo.

**9) Uso de variáveis auxiliares para o controle do jogo:**

```
int repete[70] = {0};
int contador_pergunta = 1;
```

Empregar as variáveis auxiliares garantiu que:

- `repete[ ]`: marca as perguntas já feitas, evitando repetições;
- `contador_pergunta`: mantém uma sequência coerente para o jogador, mesmo após os pulos;

**10) Armazenar o valor do prêmio como inteiro:** O grupo decidiu armazenar a variável do valor do prêmio como um inteiro - `int`, pois os valores dentro do programa não necessitam de pontos flutuantes após a vírgula, sendo exatos: `1000, 10000, 100000, 1000000`. Além disso, a representação feita na memória por um inteiro é em binário puro (0 e 1), isso torna o nosso algoritmo mais eficiente para o propósito do trabalho otimizando a velocidade na execução.

| Tipo de dado | Tamanho comum (bytes) | Economia ao converter para inteiro |
|---|---|---|
| int | 4 bytes | -------- |
| float | 4 bytes | Economia em performance |
| double | 8 bytes | 4 bytes |

**11) Encerramento imediato após “derrota”, “pare” ou “vitória”:**

```
else {
    printf("===== Resposta errada! =====\n");
    return 1;
}
```

Encerrar o jogo imediatamente após o erro, pare ou vitória garante a otimização dos recursos e mantém a dinâmica real do jogo, como no exemplo acima -> `uma resposta errada = eliminação`.

## Problemas enfrentados

**1) Erro ao abrir as perguntas:** Na primeira reunião o grupo buscou entender os elementos do jogo e os conhecimentos necessários para a realização do trabalho. Com isso, ao testar a abertura do arquivo "`perguntas.dat`" encontramos o primeiro desafio.

```
        printf("%s\n", perguntas[inc].descricao);

        printf("%s\n", perguntas[inc].alt[0][inc]);
        printf("%s\n", perguntas[inc].alt[1][inc]);
        printf("%s\n", perguntas[inc].alt[2][inc]);
        printf("%s\n", perguntas[inc].alt[3][inc]);
```

Adicionamos a variável "`alt`" como uma matriz "`[n][inc]`" para todas as 70 perguntas, porém ao realizar a compilação a saída não retornou nenhum valor. Para solucionar esse problema, retiramos a variável "`[inc]`" deixando somente os inteiros de 0 até 3 para cada um dos níveis, desse modo resolvendo o problema.

```
        printf("%s\n", perguntas[inc].descricao);

        printf("%s\n", perguntas[inc].alt[0]);
        printf("%s\n", perguntas[inc].alt[1]);
        printf("%s\n", perguntas[inc].alt[2]);
        printf("%s\n\n", perguntas[inc].alt[3]);
```

**2) Nível da pergunta e a incrementação no valor do prêmio "bugados":** Ao começar o jogo o nível das perguntas não era compatível com a etapa, perguntas do nível 4 eram utilizadas no nível 1, 2 ou 3. Além disso, o valor acrescido no saldo do prêmio não era correspondente ao definido para cada etapa, exemplo: `Pergunta 1 -> Saldo de R$1.000, Pergunta 2 -> Saldo de R$ 1.767, diferente do saldo correto que deveria ser de R$2.000.`

`Código antigo:`

```
                printf("\nDigite a alternativa: ");
                scanf("%c", &resposta);
```

`Código novo:`

```
                printf("\nDigite a alternativa: ");
                scanf(" %c", &resposta);
```

Bastou adicionar um espaço após o “ %” para solucionar o problema, isso fez com que os “Enter” dos scanf anteriores fossem desconsiderados, seguindo a lógica presente no fórum stack overflow.

**3) Repete[70] não era inicializado:** Isso fez com que o conteúdo inicial do vetor fosse lixo de memória, então durante as checagens de verificação os resultados eram aleatórios.

```
int repete[70];
```

Como o vetor não foi inicializado com um valor específico, as suas posições retêm os dados pré-existentes na memória, exemplo `repete = {0, 0, 1, 73287, -15, 1, 0, …}`. Está ocorrência provoca um erro nas linhas seguintes do código, devido a incerteza do valor correto das posições. Para solucionar o problema, inicializamos as posições do vetor sempre com o `valor zero (0)`, sendo uma forma rápida e segura de garantir que o array comece limpo.

```
int repete[70] = {0};
```

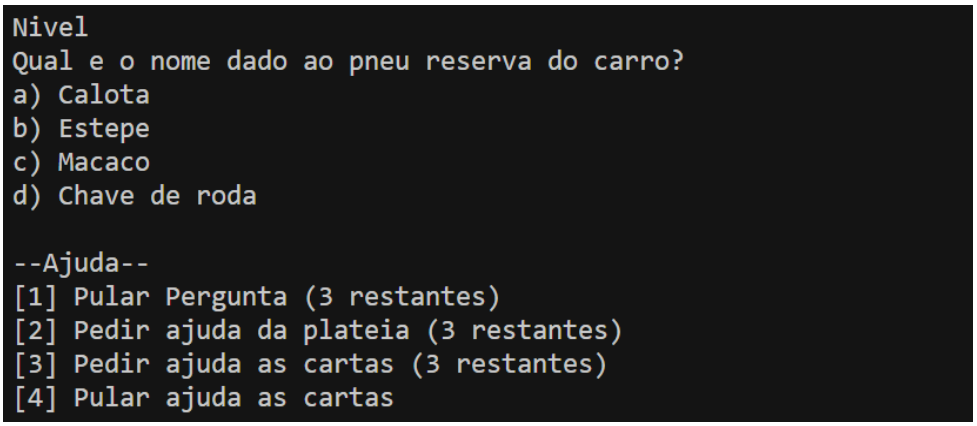
**4) Repetição de perguntas já sorteadas:** Algumas perguntas apareciam mais de uma vez durante o jogo.

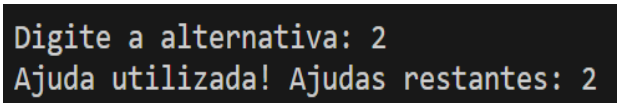
```
Pergunta 1:
Quando e comemorado o dia da independencia do Brasil?
a) 21 de abril
b) 12 de outubro
c) 7 de setembro
d) 25 de dezembro

--Ajuda--
[1] Pular Pergunta (3 restantes)
[2] Pedir ajuda da plateia (3 restantes)
[3] Pedir ajuda as cartas (3 restantes)
[4] Pular ajuda as cartas

Pergunta 5:
Quando e comemorado o dia da independencia do Brasil?
```

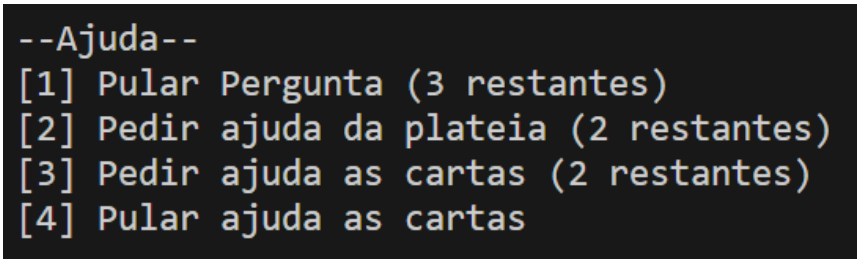
```
a) 21 de abril
b) 12 de outubro
c) 7 de setembro
d) 25 de dezembro

--Ajuda--
[1] Pular Pergunta (3 restantes)
[2] Pedir ajuda da plateia (3 restantes)
[3] Pedir ajuda as cartas (3 restantes)
[4] Pular ajuda as cartas
```

**Solução:** foi criado o vetor `repete[]` para registrar perguntas já utilizadas, evitando repetições.

**5) Ajudas sendo decrementadas simultaneamente:** Todas as ajudas diminuíam juntas após o uso de uma delas.




Como pode ser observado, ao utilizar a ajuda `[2] Plateia`, o contador da ajuda `[3] Cartas` também foi decrementado indevidamente de três para duas utilizações restantes.

**Solução:** As variáveis foram passadas por referência (`int *ajuda_plateia`, `*pulos`, `int *ajuda_carta`, etc.), permitindo o controle individual de cada ajuda através dos ponteiros.

**6) Erro nos níveis das perguntas:** Outro erro relevante estava relacionado à distribuição incorreta dos níveis de dificuldade. Algumas perguntas de nível elevado apareciam logo nas

primeiras rodadas. Isso aconteceu porque os intervalos de índices usados para o sorteio não estavam definidos corretamente. O problema foi resolvido ao ajustar as faixas de aleatoriedade para cada nível:

- 0 a 19 (fácil);
- 20 a 39 (médio);
- 40 a 59 (difícil);
- 60 a 69 (final).

## Conclusões

O desenvolvimento do trabalho de Algoritmos e Programação 2 consolidou diversos conceitos fundamentais de programação, como manipulação de arquivos binários, uso de estruturas (`struct`), controle de fluxo, funções com passagem por referência e geração de números aleatórios.

Durante a implementação do algoritmo, o grupo enfrentou e solucionou diversos problemas relacionados à leitura de dados, gerenciamento de ponteiros e controle lógico do jogo, o que contribuiu significativamente para o aprimoramento das habilidades de depuração e análise de código.

Diante desse cenário, a equipe considera que o algoritmo atingiu plenamente seus objetivos de forma clara e didática. Conclui-se, portanto, que a experiência reforçou a importância da lógica de programação, do trabalho em equipe e da atenção aos detalhes no desenvolvimento de futuros projetos.

## Referências Bibliográficas


- PAULO FEOFILLOF. **Algoritmos em Linguagem C**. [s.l.] Elsevier Brasil, 2013.
- **CPLUSPLUS.COM** - The C++ Resources Network. Disponível em: <https://cplusplus.com/>.
- **STACKOVERFLOW**. Newest questions. Disponível em: <https://stackoverflow.com/questions>.
- **Wikipedia GNU Compiler Collection**. Disponível em: <https://pt.wikipedia.org/wiki/GNU_Compiler_Collection>.
- **Git - git-commit Documentation**. Disponível em: <https://git-scm.com/docs/git-commit>.
- **GITHUB**. GitHub. Disponível em: <https://github.com/>.
- SARTHO-DEV. **GitHub - sartho-dev/trabalho.alg.prog2**. Disponível em: <https://github.com/sartho-dev/trabalho.alg.prog2>.
- **Departamento de Ciência da Computação - Universidade de São Paulo (USP)**. Disponível em: <https://www.ime.usp.br/dcc/>.
- **Introdução à Programação C - PUCRS**. Disponível em: <https://www.inf.pucrs.br/~pinho/LaproI/IntroC/IntroC.htm>.
- **Microsoft Learn - Ponteiros (C++)**. Disponível em: <https://learn.microsoft.com/pt-br/cpp/cpp/pointers-cpp?view=msvc-170>.