Ramo Estudantil IEEE - UEL

Gabriel Henrique Navarro (gabriel.navarro7@uel.br)
Enzo Nakanishi (enzo.nakanishi@uel.br)
Matheus Nakanishi (matheus.nakanishi@uel.br)
Joao Henry Volpato (joao.henry.volpato@uel.br)

RELATÓRIO FINAL:

Projeto Resolvedor de Sudoku

Londrina
2023

Ramo Estudantil IEEE - UEL

Gabriel Henrique Navarro
Enzo Nakanishi
Joao Henry Volpato
Matheus Nakanishi

RELATÓRIO FINAL:

Projeto Resolvedor de Sudoku

Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina.

Diretor de Projetos: Nathan Andreani Netzel
Gestores de Projetos: Daniel Tresse Dourado, Levi Monteiro dos Santos

Londrina
2023

Contato do Ramo: sb.uel@ieee.org
Institute of Electrical and Electronics Engineers – IEEE
Universidade Estadual de Londrina - UEL • Paraná - Brasil

NAVARRO, Gabriel Henrique. NAKANISHI, Enzo. VOLPATO, João Henry. NAKANISHI, Matheus. **Relatório Final**: Projeto Resolvedor de Sudoku. 2023. Número total de folhas: 16. Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

O projeto Resolvedor de Sudoku consiste em um software capaz de ler um arquivo .txt que contenha um sudoku incompleto (uma matriz 9x9) e realizar a verificação por linha, coluna e quadrante para que não se repita nenhum número de 1 a 9 em ambas as conferências, assim completando-o e retornando a solução possível para o desafio.

Palavras-chave: Software. Resolvedor de Sudoku. Linguagem C. Programação.

OBJETIVOS

Realizar a programação de um software que possibilite a resolução de um Sudoku lido em formato .txt através da verificação de linhas, colunas e quadrante para que não seja repetido nenhum número de 1 a 9 em nenhuma destas verificações e se não houver algum erro apresentar a mensagem de que não é possível completar tal Sudoku.

Sumário

1. FERRAMENTAS UTILIZADAS

1.1. Linguagem C:

A **linguagem C** existe desde antes da internet e foi criada pelo cientista da computação Dennis Ritchie e Ken Thompson, em 1972. O propósito inicial era que fosse uma linguagem usada no desenvolvimento de uma nova versão do sistema operacional Unix, mas hoje é aplicada para criar *softwares*. É também muito usada em banco de dados para todos os tipos de sistemas: financeiro, governamental, mídia, entretenimento, telecomunicações, saúde, educação, varejo, redes sociais, etc. Grandes empresas como Apple, Microsoft, Oracle usam a linguagem C.

A linguagem C é chamada de **linguagem nível intermediária**, pois combina os elementos das linguagens de alto nível com o funcionalismo da linguagem nível máquina. Com C é possível ter controle exato de como um programa se comporta e dá acesso direto a partes do *hardwar*e, como o espaço na memória do computador.

Suas características e aplicações são:

- Portabilidade: significa que a linguagem pode ser compilada em diversas arquiteturas, em Mac ou PC, com Linux ou Windows. Uma característica que nem toda linguagem possui.
- Simplicidade: seguindo as regras, dificilmente vai cometer erros que possam comprometer o seu programa. O compilador (um programa de sistema que traduz o código legível para os seres humanos e converte para a linguagem binária que o processador “entende”). Também avisa a maior parte dos erros de sintaxe que você cometer.

1.2. Estruturas de Repetição (for)

Denominamos “laço” (loop em inglês) a uma estrutura de repetição. As estruturas de repetição, executam a repetição de um conjunto de instruções enquanto uma determinada condição seja verdadeira.

Em linguagem C, existem três estruturas de repetição, são elas: for, while e do…while. Cada uma destas estruturas tem a sua particularidade em termos de funcionamento.

O laço **for** é uma estrutura de repetição muito utilizada nos programas em C. É muito útil quando se sabe de antemão quantas vezes a repetição deverá ser executada. Este laço utiliza uma variável para controlar a contagem do loop, bem como seu incremento. Trata-se de um comando bem enxuto, já que própria estrutura faz a inicialização, incremento e encerramento do laço.

Sintaxe:

```
for(valor_inicial; condição_final; valor_incremento)
 {
    instruções;
 }
```

Figura 1 – FOR

1.3. Visual Studio Code

O **Visual Studio Code *(VS Code)*** é um editor de código-fonte gratuito e de código aberto desenvolvido pela Microsoft. Ele é amplamente utilizado por desenvolvedores de software para escrever, editar e depurar código em diversas linguagens de programação. O VS Code é conhecido por sua leveza, extensibilidade e suporte a uma ampla variedade de extensões, o que o torna uma ferramenta popular para desenvolvimento de software em muitos ambientes e plataformas.

Figura 2 – Visual Studio Code
FONTE: *https://miro.medium.com/v2/resize:fit:1400/0*SGbxc-dbU0gyaVWm.jpg*

2. METODOLOGIA

2.1 FUNÇÃO PARA LER A O ARQUIVO

Primeiramente foi criada uma função que lê o arquivo .txt e armazena cada valor ao seu local correspondente na matriz 9x9 (readMatrix);

```
5   void readMatrix(int mat[9][9], const char *file_name)
6   {
7       FILE *file = fopen(file_name,"r");
8
9       for (int i = 0; i < 9; i++)
10      {
11          for (int j = 0; j < 9; j++)
12              fscanf(file,"%d",&mat[i][j]);
13      }
14  }
```

Imagem 3 – Função readMatrix
Fonte: O próprio autor

2.2 FUNÇÃO PARA PRINTAR O SUDOKU

Para printarmos (exibirmos) o sudoku de uma forma visualmente agradável usamos a função *printSudoku.* Este código realiza as seguintes tarefas:

1. A função **printf** é usada para imprimir caracteres especiais que formarão a estrutura do tabuleiro de Sudoku. Isso cria uma aparência visual agradável no console.
2. Um loop **for** é iniciado, que itera 33 vezes. Isso é usado para imprimir a parte superior da estrutura do tabuleiro.
    - É verificado se **i** é igual a 11 ou 22. Se for verdade, o caractere especial **203** (╠) é impresso, que é usado para criar as divisões entre as regiões do Sudoku. Caso contrário, o caractere especial **205** (═) é impresso, que é usado para criar as linhas horizontais da estrutura.
3. O caractere especial **187** (╗) é impresso para completar a parte superior da estrutura.

4. É iniciado um loop **for** aninhado que percorre as linhas e colunas do tabuleiro de Sudoku.
    - É verificado se **j** é igual a 0. Se for verdade, o caractere especial **186** (╟) é impresso, que é usado para criar as linhas verticais da estrutura à esquerda do tabuleiro.
    - O valor da célula do Sudoku (**mat[i][j]**) é impresso com um formato de espaço em branco antes e depois para centralizar o número em sua célula.
    - É verificado se **j** é igual a 2, 5 ou 8. Se for verdade, o caractere especial **186** (╟) é impresso para criar as linhas verticais da estrutura dentro do tabuleiro. Caso contrário, o caractere especial **179** (│) é impresso para criar divisões verticais dentro do tabuleiro.
5. Após imprimir uma linha completa do tabuleiro, é verificado se **i** não é igual a 8 (ou seja, se não é a última linha do tabuleiro).
    - Se **i** não for igual a 8, é impressa uma linha horizontal que separa as linhas. O tipo de linha depende se **i** é igual a 2 ou 5, caso em que uma linha mais espessa é impressa.
6. O processo é repetido para cada linha e coluna do tabuleiro até que todo o tabuleiro tenha sido impresso.
7. Por fim, a parte inferior da estrutura é impressa usando caracteres especiais, de forma semelhante à parte superior da estrutura.

```c
void printSudoku(int mat[9][9])
{
    printf("\n\t\t\t\t\t\t%c",201);

    for (int i = 0; i < 33; i++)
    {
        if (i == 11 || i == 22)
            printf("%c",203);
        printf("%c",205);
    }
    printf("%c\n",187);

    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            if (j == 0)
                printf("\t\t\t\t\t\t%c",186);

            printf(" %d ",mat[i][j]);

            if (j == 2 || j == 5 || j == 8)
                printf("%c",186);
            else
                printf("%c",179);
        }
        if (i!= 8){
            if (i != 2 && i != 5){
                for (int k = 0; k < 36; k++)
                {
                    if (k == 0)
                        printf("\n\t\t\t\t\t\t%c",186);
                    else if (k == 12 || k == 24)
                        printf("%c",186);
                    else if (k % 4 == 0)
                        printf("%c",197);
                    else
                        printf("%c",196);
                }
                printf("%c\n",186);
            }
```

Imagem 4 – Função printSudoku - I
Fonte: O próprio autor

```
57                  else{
58                      printf("\n\t\t\t\t\t\t%c",204);
59                      for (int k = 0; k < 35; k++){
60                          if (k == 11 || k == 23)
61                          {
62                              printf("%c",206);
63                          }
64                          else{
65                              printf("%c",205);
66                          }
67                      }
68                      printf("%c\n",185);
69                  }
70              }
71          }
72          printf("\n\t\t\t\t\t\t%c",200);
73          for (int i = 0; i < 33; i++)
74          {
75              if (i == 11 || i == 22)
76                  printf("%c",202);
77              printf("%c",205);
78          }
79          printf("%c\n\n\n\n\n\n\n\n",188);
80      }
```

Imagem 4 – Função printSudoku - II
Fonte: O próprio autor

2.3. Funções de verificação

Para o programa completar o Sudoku corretamente é necessário que se faça a verificação por coluna, por linha e também por quadrante. Para realizarmos estas verificações utilizamos respectivamente as funções *scanColum, scanRow* e *scanSquare*;

```
82  bool scanRow(int mat[9][9], int i, int num)
83  {
84      for (int j = 0; j < 9; j++)
85      {
86          if (mat[i][j] == num)
87              return false;
88      }
89
90      return true;
91  }
```

Imagem 5 – Função scanRow

```
bool scanColumn(int mat[9][9], int j, int num)
{
    for (int i = 0; i < 9; i++)
    {
        if (mat[i][j] == num)
            return false;
    }

    return true;
}
```

Imagem 5 – Função scanColum

Fonte: O próprio autor

```
104     bool scanSquare(int mat[9][9], int x, int y, int num)
105     {
106         if (x < 3)
107             x = 0;
108         else if (x < 6)
109             x = 3;
110         else
111             x = 6;
112
113         if (y < 3)
114             y = 0;
115         else if (y < 6)
116             y = 3;
117         else
118             y = 6;
119
120         for (int i = x; i < x+3; i++)
121         {
122             for (int j = y; j < y+3; j++)
123                 if (mat[i][j] == num)
124                     return false;
125         }
126
127         return true;
128     }
```

Imagem 5 – Função scanSquare
Fonte: O próprio autor

2.4. Função principal de resolver o sudoku

Em resumo, esta função utiliza recursão e testa diferentes números em cada posição do tabuleiro até encontrar uma solução válida ou determinar que não há solução possível. É uma implementação típica de um algoritmo de resolução de Sudoku por força bruta. Chamamos está função de *sudokuResolver.*

```
149  bool sudokuSolver(int mat[9][9])
150  {
151      int row, column;
152
153      if (!findEmptyPosition(mat,&row,&column))
154          return true; //Todas posições completadas
155
156      for (int num = 1; num <= 9; num++) //Testa valores de 1 até 9
157      {
158          if (validPosition(mat,row,column,num)) //Verifica se o valor é válido na posição
159          {
160              mat[row][column] = num;
161
162              if (sudokuSolver(mat)) //Recursão
163                  return true;
164
165              mat[row][column] = 0;
166          }
167      }
168
169      return false;
170  }
```

Imagem 6 – Função sudokuResolve
Fonte: O próprio autor

2.5. Função main

A função main é nada mais nada menos que o nosso programa em si. No geral, este programa lê um tabuleiro de Sudoku de um arquivo, tenta resolvê-lo usando a função **sudokuSolver** e depois exibe a solução ou informa que não há solução possível. Na função main são utilizadas as funções que foram criadas previamente.

```
172  int main()
173  {
174      int mat[9][9];
175
176      char fileName[50] = "semSolucao.txt";
177      readMatrix(mat,fileName);
178
179      if (sudokuSolver(mat))
180      {
181          printf("\n\n\n\n\n\t\t\t\t\t\t  SUDOKU RESOLVIDO\n");
182          printSudoku(mat);
183      }
184      else
185      {
186          printf("\n\n\n\n\n\t\t\t\t\t\t  SUDOKU SEM SOLUCAO\n");
187          printSudoku(mat);
188      }
189
190      return 0;
191  }
```

Imagem 7 – Função Main
Fonte: O próprio autor

3. RESULTADOS E DISCUSSÕES

Ao finalizarmos o código foram realizados testes com aquivos de Sudoku que haviam a possibilidade de solução e também alguns testes com aqueles que seriam impossível a resolução, verificando assim o bom funcionamento do programa.

CONCLUSÕES

A partir dos testes de funcionamento do código, foi possível afirmar que tanto a resolução quanto as devidas conferências estavam funcionando e, logo, que a programação foi feita de maneira correta, assim, finalizando o projeto.