Relatório:

Alunos: Guilherme Salim Monteiro de Castro Paes, 21.1.4109
Felipe Camargos Cotta, 21.1.4007

Implementação:

Typedefs:

```
typedef struct celula Celula; →TAD célula
typedef int boolean;→ uso de tipo booleano não presente em C
enum { false, true };
```

Principais Funcionalidades:

*Ler arquivo:

```
void TabuleiroInicializa(char * arquivo, int **tabuleiro);
```

Lê usa funções como fopen e fscanf para ler arquivo.

*Preparações para cálculos:

```
int contaCelulas (int **tabuleiro);
Celula* defineVazias(int **tabuleiro, Celula *cheias, int nro_vazias)
Celula *vazias = alocaCelulas(nro_vazias);
Celula *cheias = alocaCelulas(nro_cheias);
```

Contamos quantas células de cada tipo (cheias e vazias) tem no tabuleiro, alocamos dinamicamente vetores de TAD Celula para armazenamento das duas e na função defineVazias nós preenchemos os vetores com as posições de cada célula.

*Validação do Tabuleiro:

```
void validador(Celula *cheias, int nro_cheias, int **tabuleiro, int **invalidas);
boolean EhValido(int **tabuleiro,Celula cheia, int **invalidas);
```

A função EhValido recebe uma célula (que é passada através da função validador) e percorre a toda a linha, a coluna e a região desta célula, procurando por valores inválidos, caso ache um valor inválido, a célula testada é adicionada a uma matriz alternativa chamada “Invalidas”(que possui todos os valores iniciais = 0). Na matriz inválidas, na posição que fica a célula invalidada, é marcado um número de 1 a 7 indicando o caso de invalidez de acordo com a seguinte tabela:

```
/*
1=Apenas linha
2=Apenas Coluna
3=Linha e Coluna
4=Linha e Regiao
5=Coluna e Regiao
6= Linha Coluna e Regiao
7=Apenas Regiao*/
```

Os casos de invalidez da célula são marcados desta forma com o objetivo de tornar a impressão dos erros mais simples e limpa para o usuário.

*Pós validação:

Após ser feita a validação de todas as células cheias, nós rodamos um for, que passa por todas as células cheias e identifica o número de inválidas:

```
for(int i = 0; i < 9; i++){
for(int j = 0; j < 9; j++){
if(invalidas[i][j] != 0)
nro_invalidas++;
}
}
```

Caso o número de vazias seja 0 e o número de inválidas também seja 0, é impressa na tela a mensagem de que o usuário ganhou o jogo e o programa se encerra. Porém, caso o jogo seja válido, mas incompleto, é rodada a função:

```
void imprimeSugestoes(int nro_vazias, int **tabuleiro, Celula *vazias, int **invalidas);
```

Esta função pega todas as células vazias e passa pela função:

```
int* valoresValidos(int **tabuleiro, Celula vazia,int **invalidas);
```

Que por sua vez tem o objetivo de testar todos os valores possíveis (1 a 9 de acordo com as regras do Sudoku) e armazenar em um vetor de valores válidos. Após ser feito isto, a função ainda preenche o resto do vetor (que tem tamanho máximo de 9 inteiros) com o valor 10, uma vez que este valor é impossivel nós utilizamos disso para definir um ponto de parada na impressão do vetor. Ou seja, se o vetor é retornado apenas com as duas primeiras posições preenchidas, por exemplo, as outras 7 posições são atribuidas como 10 e a função “imprimeSugestões” irá imprimir todos os valores do vetor até chegar no primeiro 10.

Caso haja alguma célula inválida, é rodada a função:

```
void imprimeErros(int **invalidas);
```

Esta função passa por toda a matriz de células inválidas (após esta matriz ter sido preenchida anteriormente) e imprime os erros de acordo com seu código (se em invalidas[i][j] estiver o valor 1, por exemplo, a função irá imprimir o erro de linha no sudoku original).

Comentário sobre a função imprimeErros: o uso da variável int g se deu para podermos formatar e imprimir melhor a saída da função. Antes de sua implementação, o programa imprimia:
“Linha 1: (1, 3)
Linha 1: (1, 6)”
Então atribuímos o valor da linha para a variável, e colocamos um if antes da impressão da string “Linha %d”, desta forma, quando a variável g fosse igual ao valor da linha/coluna, era impresso a string e logo após era feito um incremento no valor da variável (g++), desta forma, da próxima vez que a função passasse pelo if, ela não entraria e seriam impressos apenas os valores das celulas inválidas, não a linha na qual estavam. Como ficou a impressão: “Linha 1: (1,3) (1,6)”.

Impressões gerais, Análise e Conclusão:

O processo de implementação deste trabalho foi definitivamente um dos mais cansativos que tivemos. Falhas de segmentação, erros de execução, etc eram mais comuns do que acertos. As partes mais difíceis e cansativas na nossa opinião foram a modularização e a formatação da impressão.

Trabalhar com a main presente em um arquivo e as outras funções presentes em outro arquivo foi desafiador, uma vez que o nível de abstração para compreendermos o que iria acontecer

na memória da máquina foi extremamente alto. No entanto, esta dificuldade se mostrou benéfica uma vez que nos ajudou a entender e conhecer melhor sobre a programação e estrutura de dados.

A impressão de saídas também teve seu nível de dificuldade, uma vez que os métodos que escolhemos para resolução dos problemas nos ajudavam muito no sentido de armazenamento de valores (matriz invalidas, vetor valores validos, etc) porém estes métodos não eram de extrema ajuda no que tange à impressão deste valores. Assim, tivemos muita dificuldade mas criamos níveis e métodos alternativos para termos uma impressão de saídas bem feita (preenchimento da matriz com valor 10, por exemplo, ou uso do int g para sabermos quando uma linha tinha ou não sido impressa, como explicados acima).

No entanto, apesar das dificuldades achamos o TP divertido e didático e sentimos que aprendemos muito durante sua implementação e isso, foi o que mais nos motivou a continuar por horas seguidas na implementação, pois sabíamos que quanto mais avançássemos, mais conhecimentos nos estaríamos adquirindo.