**Relatório do Problema 2: Concurso sobre Inovação Tecnológica**

**Esther de Santana Araújo**

Engenharia de Computação - Universidade Estadual de Feira de Santana

Feira de Santana, 11 de junho de 2018.

esthersss09@gmail.com

**1. Introdução**

Um concurso de inovação tecnológica fora organizado e realizado a partir da iniciativa de um grupo de professores e empresas da cidade de Feira de Santana, que visavam incentivar e fortalecer a área de tecnologia da informação da região. As equipes participantes deveriam ser formadas por universitários e apresentariam seus projetos, estes que poderiam ser da área de Gestão, Educação ou Projeto Social. Um dos objetivos dos idealizadores do concurso era também fornecer oportunidades aos participantes, já que o prêmio era uma consultoria de professores e empresas da área para que futuramente fosse possível a concretização e comercialização do produto.

Para que o concurso ocorresse, era necessário um software que realizasse o gerenciamento da competição. Então, foi solicitado aos estudantes de Engenharia de computação da UEFS a construção do programa, também como forma de dar visibilidade ao curso e à Universidade.

A resolução do problema foi elaborada através da linguagem de programação C, que é uma linguagem estruturada e compilada. O código fonte é composto por funções para facilitar a modularização e uma **struct** que permite armazenar diferentes tipos de dados num mesmo índice. Utilizou-se também um arquivo de texto, para salvar o ranking do concurso.

**2. Metodologia**

**2.1 Sessões tutoriais**

Nas primeiras sessões foram discutidas formas de armazenar os dados das equipes. Optei por vetores de **struct**, onde cada índice do vetor representa uma equipe, e tratando-se de uma **struct** já estaria armazenando os dados de cada uma de forma eficiente e fácil de manusear dentro do código.

Foram bastante discutidas nas sessões formas de tratar os espaços na leitura do arquivo, e sugerido o uso do **fgets** para ler uma linha inteira além da função **str tok** para ignorar as lacunas, porém resolvi por utilizar a função **fscanf** para coletar as informações, pois ela já ignora os espaços naturalmente e, dessa forma, eu não precisaria me preocupar com o tamanho de cada espaço entre as palavras e números.

Em sessão fora decidido que os nomes dos arquivos de entrada e saída fossem no padrão dos exemplos do descritivo, porém, o problema pede que se dê ao usuário a possibilidade de nomear o arquivo de saída. Dessa forma, deixei apenas o arquivo de entrada com o nome padrão.

**2.2 Requisitos do programa**

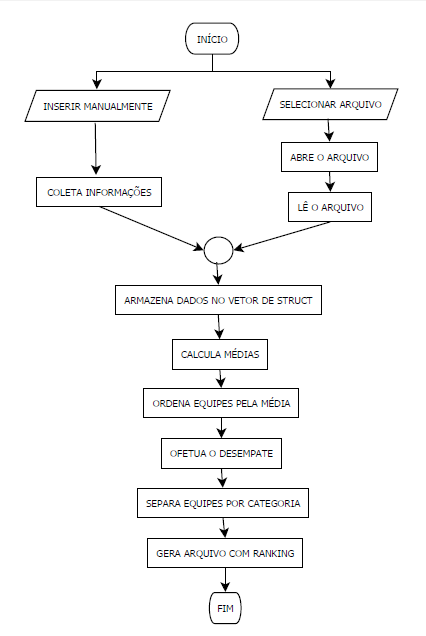
O programa deveria permitir a entrada de no máximo 10 equipes, que seriam identificadas por um código (o qual não poderia se repetir entre as equipes), por sua categoria e pelo seu nome. Além disso, o software deveria receber as notas de 5 jurados, destinadas a determinados critérios de avaliação, sendo o programa responsável por obter a média desses critérios e a média geral da equipe (levando em consideração o peso de cada critério para o cálculo da média). Sendo assim possível obter o resultado do concurso.

Os dados de cada equipe poderiam ser recebidos pelo programa através de uma entrada manual ou de um arquivo txt, e por fim deveria ser gerado um outro arquivo com o ranking do concurso. O arquivo de saída com o ranking deveria ter a media de cada critério, informar a equipe vencedora de cada categoria, além listar as equipes desclassificadas.

**2.3 Descrição do programa**

O programa oferece duas opções ao usuário, uma de colocar os dados de cada equipe manualmente e outra de ler um arquivo já com essas informações, assim como solicitado no problema. Para isso, criei funções diferentes para receber os dados, porém, o resto das funções responsáveis por calcular as médias, separar as equipes por categoria, ordenar as equipes pela média, gerar saída e etc. são comuns aos dois tipos diferentes de entrada (manual ou através do arquivo).

Depois que o vetor de **struct** recebe os dados de todas as equipes (qualquer que tenha sido a forma de entrada) é chamada a função de calcular as médias, e em seguida existe outra função para ordenar o vetor de **struct** de forma decrescente em relação à essas médias que foram calculadas. Depois disso, esse vetor é ordenado novamente caso existam equipes com médias iguais, e então ele é enviado para outra função responsável pelo desempate que leva em consideração a nota obtida pelo grupo no critério nível de maturidade. Com o vetor devidamente ordenado, outra função separa as equipes por categoria para formar o ranking, e por fim a função de gerar o arquivo de saída é chamada. O fluxo de execução sintetizado anteriormente pode ser observado conforme a **(Figura 1)** anexada abaixo**.**



**Fonte: Próprio autor.**

**Figura 1: Fluxograma.**

**2.4 Ordem de codificação**

As primeiras funções implementadas no programa foram: a de pegar os dados de forma manual, a de fazer os cálculos das médias gerais e a das médias dos critérios. Optei por começar com pela construção dessas funções, pois eram mais simples de fazer e também porque eram parte das primeiras metas estipuladas em sessão. Durante todo o processo de codificação tentei implementar a função de leitura do arquivo, porém, já que havia considerado esta parte como a mais difícil no problema imaginei que fosse demorar um pouco mais para concluí-la. Dessa forma, para não ficar estagnada no desenvolvimento do programa fui construindo paralelamente as outras funções de ordenação das equipes, separação das mesmas por categoria, de desempate e a de gerar a saída.

A função de ordenação recebe o vetor de **struct**, e utilizando o método **BubbleSort** reorganiza esse vetor de forma decrescente com relação às médias das equipes. Optei pelo uso desse método de ordenação, apesar de ser o mais lento, pois o vetor teria no máximo 10 posições, ou seja, não é muito grande e não atrasaria relevantemente a execução do programa. A função para desempatar compara as médias de duas em duas, caso sejam iguais, realiza uma troca de posição no vetor respeitando o critério de desempate. A função de separar as categorias divide o vetor principal já ordenado em três outros vetores (educação, gestão e projeto social).

No caso da entrada por arquivo, a lógica de leitura funciona da seguinte forma: Existe um laço de repetição do tipo **do while** que realiza a leitura até que o arquivo chegue ao final, e dentro deste laço existem mais 4. O primeiro é responsável por ler a parte destinada ao cadastro, que realiza a leitura até que a variável de categoria encontre o J1no arquivo (o que sinaliza ao programa que o cadastro acabou e a próxima parte a ser lida é a matriz com as notas). O segundo laço é do tipo **while**, responsável por armazenar os C’s e J’s do arquivo numa variável lixo, apenas para desconsiderar esta parte do arquivo que não será utilizada. O terceiro laço é do tipo **do while** (e tem um **while** dentro dele**)** os dois juntos realizam a tarefa de ler a matriz com o nome de cada equipe, a categoria da mesma e suas respectivas notas.

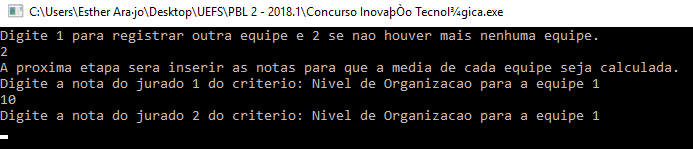
O código fonte do programa foi desenvolvido utilizando a IDE *Dev C++* de desenvolvimento em linguagem *C,* no sistema operacional *Windows 10.*

# Resultados e Discussões

Para utilizar o programa o usuário deve estar atento e seguir as instruções de uso do software para evitar possíveis erros. Sendo assim, deve-se seguir os seguintes passos:

* Escolher o modo de entrada de dados;
* Caso tenha escolhido o arquivo, coloca-lo no mesmo diretório do executável;
* Caso escolha a entrada manual, seguir as instruções na tela do programa;
* Nomear o arquivo de saída.

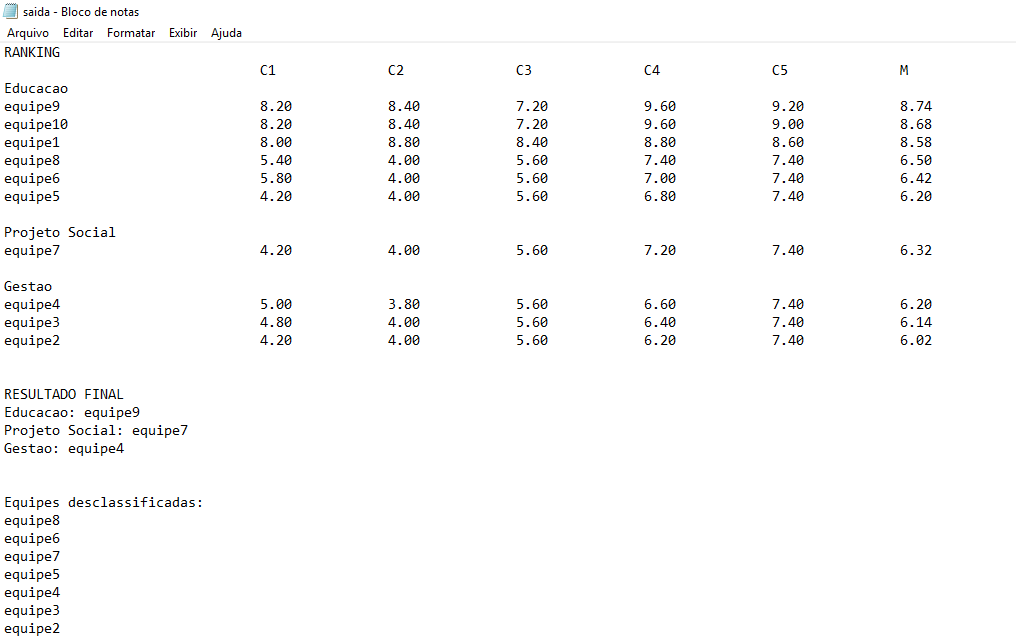
Em relação à escolha manual como forma de entrada, é importante salientar que o código da equipe deve ser um número inteiro (como combinado em sessão tutorial), e seguindo os exemplos do descritivo o nome da equipe deve ser composto por uma palavra. Depois do cadastro, o usuário irá colocar as notas de todas as equipes, sendo feita a leitura de um critério por vez. O critério em questão estará sinalizado na tela conforme é demonstrado na **(Figura 2)** logo abaixo.



**Fonte: Próprio autor.**

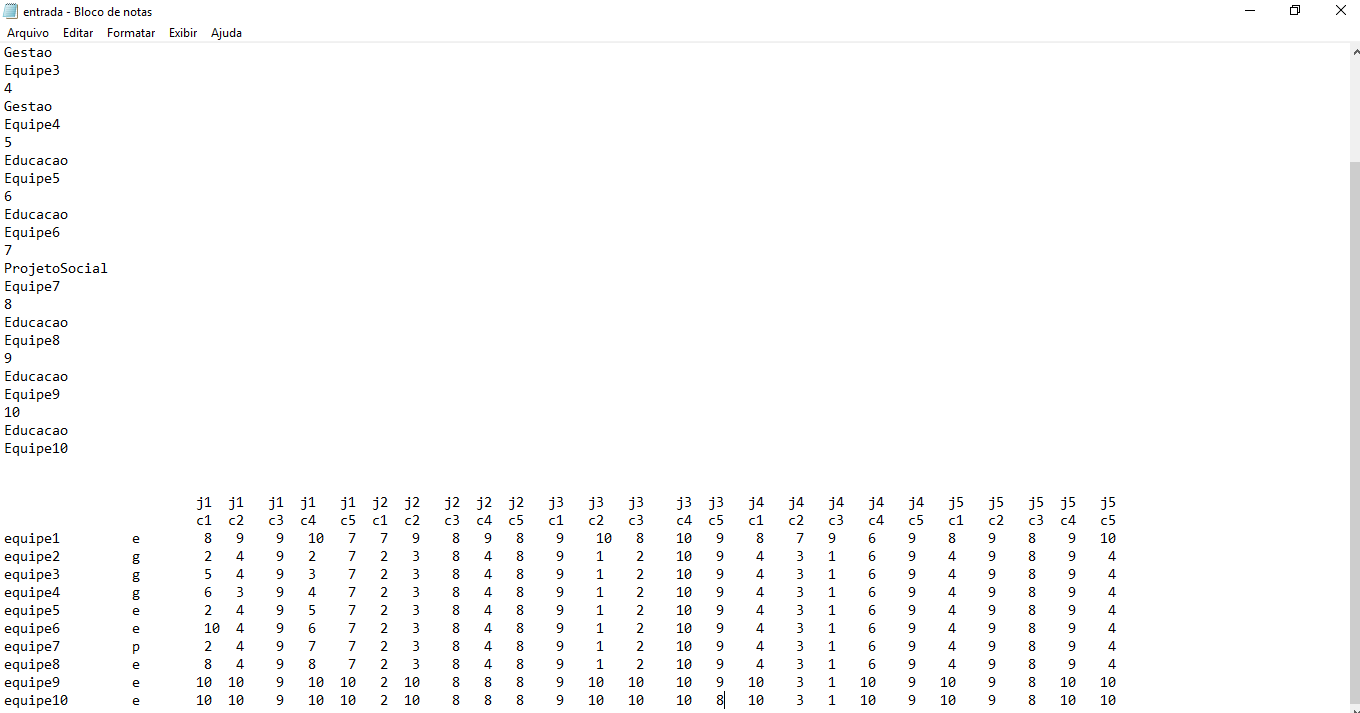
**Figura 2: Print de tela.**

No caso do arquivo, deve estar no formato txt, e formatado como o exemplo do descritivo (exceto na parte do cadastro, na qual a categoria também deve ser composta por um nome apenas). Tentei utilizar o **fgets** ao invés do **fscanf** para ler a linha toda e pegar dois nomes (se necessário) para a variável de categoria, porém, para isso eu teria que mudar a lógica de leitura de arquivo e não encontrei outra forma de faze-la. Realizei alguns testes para o arquivo, e se as instruções de uso forem corretamente seguidas o programa funciona de forma efetiva, como se pode conferir no arquivo de saída **(Figura 3)** gerado a partir do arquivo de entrada teste **(Figura 4)**, ambas anexadas logo abaixo.



**Fonte: Próprio autor.**

**Figura 3: Arquivo de saída.**



**Fonte: Próprio autor.**

**Figura 4: Arquivo de entrada.**

# Conclusão

O sistema desenvolvido soluciona o problema proposto de forma completa. Os requisitos solicitados como o de ordenar as equipes corretamente, pedir ao usuário para nomear o arquivo de saída, gerar o arquivo com o ranking tal qual o exemplo do descritivo, e etc. foram todos cumpridos.

O programa pode ter possíveis melhorias e extensões, como por exemplo a geração de um arquivo com os dados da entrada manual, para caso ocorra um erro de digitação o usuário não tenha que repetir o processo de colocar as notas e apenas modifique o arquivo, podendo utilizar o outro recurso do programa de ler o arquivo com as notas. Através do processo de construção deste programa, foi possível aprender a gerar arquivos, ler arquivos, e praticar modularização.

# Referências

SCHILDT, H. **C Completo e Total**. 3. ed. Makron B1ooks, 1996.