16 DE MAIO DE 2022

RELATÓRIO DO TRABALHO DE PROJETO 1

CADEIRA DE PROGRAMAÇÃO

ANDRÉ CASTRO (22768), GABRIEL RAMOS (23531), GONÇALO MENDES (24012), LARA MARQUES (23535), MARIANA GOMES (23539), GABRIELA NETO (24011)

1º ANO DA LICENCIATURA EM ENGENHARIA E GESTÃO INSDUSTRIAL

IPCA

Índice

Introdução

A realização deste projeto, que se irá perlongar ao longo de todo o semestre, foi proposta no âmbito da disciplina de Programação, que integra o 1º ano da licenciatura de engenharia e gestão industrial na instituição superior do Cávado e do Ave (IPCA), tendo como objetivo e propósito sedimentar e aprofundar os conhecimentos adquiridos na linguagem C, principalmente os conteúdos interligados com os conceitos de listas.

No mesmo foi nos colocado o desafio de desenvolver uma proposta digital para o problema de escalonamento- Flexible Job Shop Problem- sendo que a mesma deverá gerar uma solução para o mesmo num produto dependente de diversas máquinas e operações, de modo a minimizar o tempo fundamental para o seu desenvolvimento e finalização.

Para o desenvolvimento do projeto foram utilizadas diversas ferramentas de aprendizagem, como vídeos intuitivos, auxílio do docente, documentos disponibilizados no github e documentos de aulas lecionadas.

Desenvolvimento do projeto

Como organizamos o projeto

No Visual Studio foi criado um projeto, subdividido em 8 ficheiros, dos quais 6 são arquivos de origem (.c), 1 é arquivo de cabeçalho e um 1 é arquivo de recurso (como ilustra a figura 1).

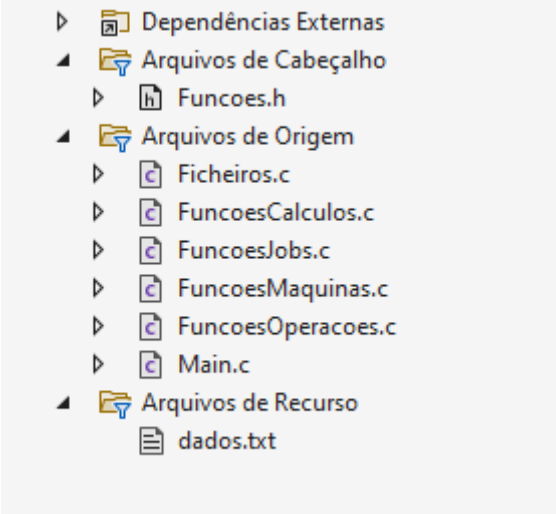

Figura 1

No arquivo de cabeçalho (.h) denominado *Funcoes.h* foram colocadas as structs destinadas às máquinas, às operações e aos jobs.

Nos arquivos de origem foram então colocados 6 arquivos, sendo que nas *FuncoesMaquina.c* se encontram desenvolvidas as funções que irão ser necessárias para criar, inserir, remover, procurar, alterar e testar (mostrar as máquinas disponível na consola), este raciocínio / organização aplica-se igualmente aos arquivos *FuncoesJobs.c* e *FuncoesOperacoes.c*, mas para jobs e operações, respetivamente. Nas *FuncoesCalculo.c* foram desenvolvidas as funções necessárias ao cálculo do tempo mínimo, máximo e médio para o desenvolver de um jobe nos *Ficheiros.c* foram desenvolvidas as funções necessárias para ler e guardar os dados dal listas de jobs, operações e máquinas.

Posteriormente foi criado um arquivo *Main.c,* onde foram criadas as listas associadas aos parâmetros das máquinas, operações e jobs, para tal ser possível, foram chamadas as funções criadas e definidas nos ficheiros *FuncoesJobs.c*, *FuncoesOperacoes.c* e *FuncoesMaquinas.c*. Neste está também presente funções definidas em *FuncoesCalculo,* de modo a ser, posteriormente, visível na consola o objetivo final de todo o projeto- o mínimo, o máximo e a média do tempo necessário para o desenvolvimento de um job.

Como pensamos e estruturamos o problema

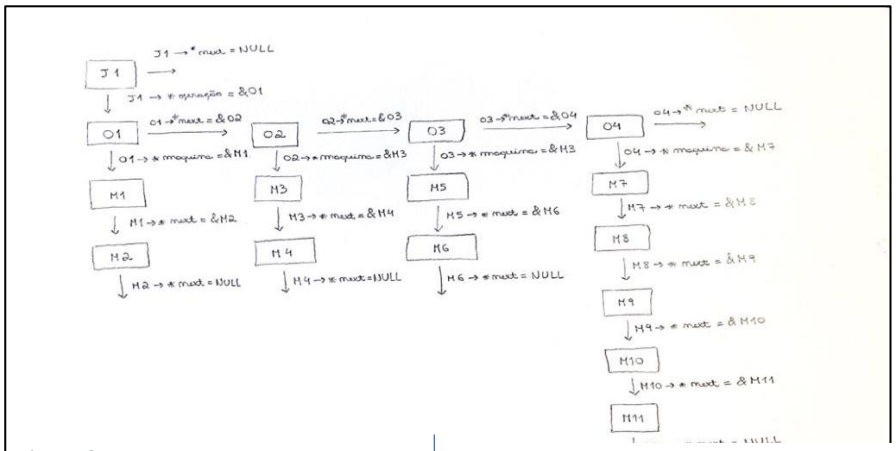

Figura 2

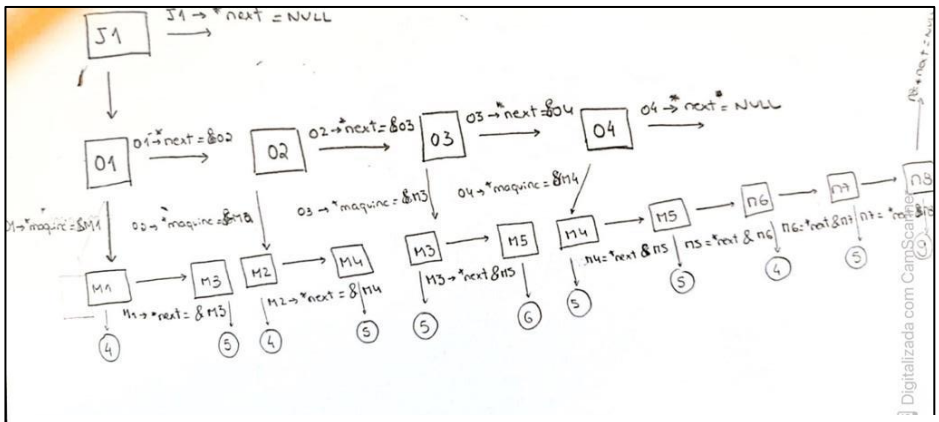

Figura 3

Num primeiro momento, foi desenvolvido um esquema, como mostra a *Figura 2*, composto por 1 job que se efetuava passando por 4 operações, e cada uma dessas operações poderia ser desenvolvida por variadas máquinas, que tinham um código e um tempo específico. No entanto foi percetível um erro cometido, o facto de máquinas iguais, mas com tempos de desenvolvimento diferentes serem diferentemente denominadas, e tal não poderia acontecer, pois assim teríamos 11 máquinas e não 8 como era pressuposto.

Então depois de ter sido desenvolvido um longo trabalho de acordo com o esquema da figura 2, o mesmo foi restruturado de modo a coincidir com as características propostas pelo projeto.

Depois de repensar a estrutura, como é possível pela observação da *Figura 3* , colocou-se igualmente as operações necessárias ao desenvolvimento de um job e as possíveis máquinas com as quais cada operação pode ser desenvolvida, sendo que cada uma das máquinas possui o tempo necessário que a mesma necessita para efetuar a operação.

Estruturas de dados

Arquivo *Funcoes.h*

```c
/**
 * @brief Informacao de uma maquina.
 */
typedef struct Maquina {

    int codigoMaquina; /**< codigo para identificar a maquina*/
    int tempo;/**<tempo que a maquina demora a executar a sua operacao*/

}Maquina;

/**
 * @brief Lista de maquinas
 * Informaçao sobre as maquinase apontador para a seguinte
 */
typedef struct ListaMaquinas {

    Maquina maquina;
    struct ListaMaquinas* next;

}ListaMaquinas;

/**
 * @brief Informaçao sobre uma operacao.
 */
typedef struct Operacao {

    int codigoOperacao; /**<codigo para identificar a operacao*/
    char nome[N];

}Operacao;

/**
 * @brief Lista de operçoes.
 * Informaçoes sobre a operações, as maquinas e apontador para a seguinte
 */

typedef struct ListaOperacoes {

    Operacao operacao;
    ListaMaquinas* listaMaquinas;
    Operacao* next; /*<apontador do sitio onde é armazenado a proxima operacao*/

/**
 * @brief Informação de um Job
 */
typedef struct Job {

   int codigoJob; /*<codigo para identificar o job*/
   char nome[N];

}Job;


/**
 * @brief Lista de jobs.
 * Informação de um job, operçoes, maquinas e apontador para o seguite
 */
typedef struct ListaJobs {

    Job job;
    ListaOperacoes* listaOperacoes;/*<lista de operacoes que sao uzadas para realizar um job*/
    struct job* next;/*<apontador do sitio onde é armazenado o proximo job*/
```

Figuras 4.1,4.2,4.3

Como mostram as *Figuras 4.1, 4.2, 4.3* foram criadas estruturas (structs) para as máquinas, operações e jobs, que serão posteriormente utilizadas no programa. Como tal nas estruturas da máquina foram definidos dois parâmetros que a mesma tinha de conter, o seu código e tempo. Para as operações e jobs foi lhes atribuída uma denominação e um código.

Para além das estruturas criadas anteriormente, foram desenvolvidas outras estruturas de modo a estabelecer ligações entre as mesmas. Na struct *ListaMaquinas,* foram introduzidos dois campos, sendo que no primeiro consta informação relativa a uma máquina e, no segundo o apontador para a seguinte máquina. Na struct *ListaOperacoes,* estão definidos três campos, sendo que no primeiro é colocada toda informação relativa a uma operação, no segundo o apontador para a lista de máquinas que a operação possui e no terceiro o apontador para a operação seguinte. Na struct *ListaJobs,* foram estruturados três campos, no primeiro é definida a informação relativa a um job, no segundo o apontador para a lista de operações que o job possui e no terceiro o apontador para o seguinte job.

```
100  /**
101   * @brief Preservar dados em ficheiro.
102   */
103  typedef struct TodaInformacao {
104
105      int codigoJob;
106      int codigoOperacao;
107      int codigoMaquina;
108      int tempo;
109
110  }TodaInformacao;
111
```

Figuras 4.4

Para além das estruturas criadas anteriormente, foi também gerada uma struct independente constituída por quatro campos que possuem um código de um job, de uma operação, de uma máquina e um tempo relativo à operação, com o objetivo de preservar toda a informação, de modo que a leitura e a preservação das informações relativas aos jobs sejam possíveis.

```
#pragma region Funcoes
//funções para as maquinas
Maquina* CriaMaquina(int novoCod, int novoTempo);
ListaMaquinas* CriaNodoListaMaquinas(Maquina* c);
ListaMaquinas* InsereMaquinaListaMaquina(ListaMaquinas* h, Maquina* m);
ListaMaquinas* ProcuraMaquina(ListaMaquinas* inicio, int cod);
void MostraTodasMaquinas(ListaMaquinas* h);
void MostraUmaMaquina(ListaMaquinas* inicio, int cod);
ListaMaquinas* RemoveMaquina(ListaMaquinas* h, int cod);
ListaMaquinas* RemoveTodasMaquinas(ListaMaquinas* h);
ListaMaquinas* AlteraMaquina(ListaMaquinas* h, int cod, int novoTempo, int novoCod);

//funções para as operações
Operacao* CriaOperacao(int novoCod, char nome[N]);
ListaOperacoes* CriaNodoListaOperacoes(Operacao* o);
ListaOperacoes* InsereOperacaoListaOperacao(ListaOperacoes* h, Operacao* o);
ListaOperacoes* ProcuraOperacao(ListaOperacoes* inicio, int cod);
ListaOperacoes* InsereMaquinaOperacao(ListaOperacoes* h, Operacao* o, int cod);
void MostraTodasOperacoes(ListaOperacoes* h);
void MostraUmaOperacao(ListaOperacoes* inicio, int cod);
void MostraMaquinasOperacao(ListaOperacoes* inicio, int codOperacao);
void MostraMaquinasOperacoes(ListaOperacoes* inicio);
ListaOperacoes* RemoveOperacao(ListaOperacoes* h, int cod);
ListaOperacoes* RemoveOperacao2(ListaOperacoes* h, int cod);
ListaOperacoes* RemoveTodasOperacoes(ListaOperacoes* h);
ListaOperacoes* AlteraOperacao(ListaOperacoes* h, int cod, int novoCod, char* novoNome);

//funções para os Jobs
Job* CriaJobs(int novoCod, char nome[N]);
ListaJobs* CriaNodoListaJobs(Job* j);
ListaJobs* InsereJobsListaJobs(ListaJobs* h, Job* j);
ListaJobs* ProcuraJob(ListaJobs* inicio, int cod);
ListaJobs* InsereOperacaoJobs(ListaJobs* h, Job* j, int cod);
void MostraTodosJobs(ListaJobs* h);
void MostraUmJob(ListaJobs* inicio, int cod);
void MostraMaquinasOperacoesJob(ListaJobs* inicio, int codJob);
void MostraMaquinasOperacoesJobs(ListaJobs* inicio);
ListaJobs* RemoveJob(ListaJobs* h, int cod);
ListaJobs* RemoveJob2(ListaJobs* h, int cod);
ListaJobs* RemoveTodosJobs(ListaJobs* h);
ListaJobs* AlteraJob(ListaJobs* h, int cod, int novoCod, char* novoNome);

//funções calculo
int TempoMinimo(ListaOperacoes* InicioOperacao);
int TempoMaximo(ListaOperacoes* InicioOperacao);
float TempoMedioDeUmaOperacao(ListaOperacoes* InicioOperacao, int Cod);

//extra
float TempoMedioDeUmJob(ListaOperacoes* InicioOperacao);

//funções para guardar e receber dados
bool SaveMaquina(ListaMaquinas* h, char* fileName);
bool SaveOperacao(ListaOperacoes* h, char* fileName);
bool SaveJobs(ListaJobs* h, char* fileName);
bool SaveAll(ListaJobs* h, char* fileName);
ListaJobs* GetAllMaquinas(char* fileName);
ListaOperacoes* GetAllOperacoes(char* fileName);
ListaJobs* GetAllJobs(char* fileName);
ListaJobs* GetAll(char* fileName, ListaJobs* h);
ListaJobs* GetData(char* fileName);
```

Figura 5

Posteriormente no arquivo *Funcoes.h*, foram chamadas todas as funções desenvolvidas e utilizadas nos ficheiros ponto c, de modo a interligar todos estes arquivos (*Figura 5*).

Arquivo *FuncoesMaquinas.c (FuncoesOperacoes.c e FuncoesJobs.c)*

Neste arquivo foram colocadas todas as funções destinadas à criação (*CriaMaquinas e CriaNodoListaMaquinas*), ligação (*InsereMaquinas*), remoção (RemoveMaquina), procura (ProcuraMaquina), alteração (AlteraMaquina) e apresentação (TesteMaquina) de máquinas.

```
12  /**
13   * .
14   * @brief Cria nova Maquina
15   * @param novoCod codigo da maquina
16   * @param novoTempo tempo da maquina
17   * @return
18   */
19  Maquina* CriaMaquina(int novoCod, int novoTempo) {
20      Maquina* nova = (Maquina*)malloc(sizeof(Maquina));
21      nova->codigoMaquina = novoCod;
22      nova->tempo = novoTempo;
23      return nova;
24  }
28  /**
29   * .
30   * @brief Cria novo nodo para a Lista de Maquinas
31   * Copia para o nodo da lista a informação de uma Maquina
32   * @param c nova maquina
33   * @return
34   */
35  ListaMaquinas* CriaNodoListaMaquinas(Maquina* c) {
36      ListaMaquinas* nova = (ListaMaquinas*)calloc(sizeof(ListaMaquinas));
37      nova->maquina.codigoMaquina = c->codigoMaquina;
38      nova->maquina.tempo = c->tempo;
39      nova->next = NULL;
40      return nova;
41  }
```

Figura 6,1 e 6,2

```
43  /**
44   * .
45   * @brief Insere Maquina na Lista de Maquinas
46   * @param [in] Apontador para o inicio da lista de maquinas
47   * @param m maquina criada
48   * @return
49   */
50  ListaMaquinas* InsereMaquinaListaMaquina(ListaMaquinas* h, Maquina* m) {
51      if (m == NULL) return h;    //se nova não tem dados
52      //Cria novo nodo da lista de pessoas
53      ListaMaquinas* nova = CriaNodoListaMaquinas(m);
54      if (h == NULL) h = nova; //se lista é vazia
55      else {                  //insere ordenado pelo m
56          nova->next = h;
57          h = nova;
58      }
59      return h;
60  }
```

Figura 7

Na função *CriaMaquina (Figura 6,1),* é introduzida a informação numa máquina e posteriormente na função *CriaNodoListaMaquinas (Figura 6.2)* é criado um local para uma máquina específica e após esta criação a função *InsereMaquina* (*Figura 7)* permite fazer a ligação entre as diversas máquinas.

```c
/**
 * .
 * @brief Procura Maquina
 * @param [in] Apontador para o inicio da lista de maquinas
 * @param cod codigo da maquina a procurar
 * @return
 */
ListaMaquinas* ProcuraMaquina(ListaMaquinas* inicio, int cod) {
    ListaMaquinas* aux = inicio;
    while (aux) {
        if (aux->maquina.codigoMaquina == cod) return aux; //se encontrou
        aux = aux->next;
    }
    return NULL;            //se não encontrou
}
```

Figura 8

```c
ListaMaquinas* RemoveMaquina(ListaMaquinas* h, int cod) {

    if (h == NULL) return NULL;    //Lista vazia
    else
    {
        ListaMaquinas* aux = h;
        ListaMaquinas* auxAnt = NULL;

        //caso seja a primeira maquina a ser removida
        if (aux && aux->maquina.codigoMaquina != cod) {
            h = aux->next;
            free(aux);
            return h;
        }
        else
        {
            auxAnt = aux;
            aux = aux->next;
        }

        while (aux && aux->maquina.codigoMaquina != cod) {   //procura para remover
            auxAnt = aux;
            aux = aux->next;
        }
        if (aux != NULL) {        //se encontrou, remove
            auxAnt->next = aux->next;
            free(aux); //limpar o espaco do armazenamento da maquina removida
        }
    }
    return h;
}
/**
 * .
 * @brief Remove todas as maquinas
 * @param [in] Apontador para inicio da Lista
 * @param [out] Apontador para inicio da Lista
 */

ListaMaquinas* RemoveTodasMaquinas(ListaMaquinas* h) {

    if (h == NULL) return NULL;    //Lista vazia

    else {

        ListaMaquinas* aux = h;

        while (aux != NULL) {

            //caso seja a primeira maquina a ser removida
            if (aux) {
                h = aux->next;
                free(aux);
                aux = h;
            }
        }
    }
    return h;
}
```

Figura 9,1 e 9,2

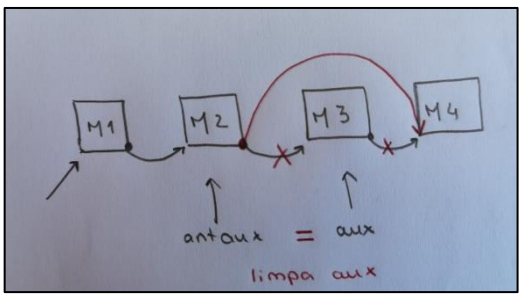

Figura 9,3

Na função *ProcuraMaquina (Figura 8)* é introduzido um código respetivo à máquina que pretendemos encontrar percorrendo esta a lista das máquinas até encontrar caso ela exista. Esta função é usada na função *RemoveMaquina (Figura 9,1)* que nos permite localizar e eliminar a máquina pretendida. Para além desta última função "remover" também criamos uma função que nos permite eliminar todas as máquinas de uma lista que posteriormente será útil, aquando na remoção de uma operação e também no processo de remoção de uma job.

Na função *RemoveMaquinas (Figura 9,1)* é criado um apontador *aux* do tipo máquina que aponta para o início da lista, após tal, este apontador vai percorrer a lista e compara se o código a procurar é igual ao código da máquina para o qual *aux* está a apontar, isto é possível devido ao ciclo criado pelo while. Caso tal se verifique a máquina a remover é eliminada.

```
152 /**
153  * .
154  * @brief Alterar tempo e codigo da Maquina
155  * @param [in] Apontador para inicio da Lista
156  * @param cod Codigo da maquina a alterar
157  * @param novoTempo novo tempo da maquina
158  * @param novoCod novo codigo da maquina
159  * @param [out] Apontador para inicio da Lista
160  */
161 ListaMaquinas* AlteraMaquina(ListaMaquinas* h, int cod, int novoTempo, int novoCod) {
162     ListaMaquinas* aux = ProcuraMaquina(h, cod);
163     if (aux != NULL)        //se encontrou a maquina, returnou o aux
164     {
165         aux->maquina.tempo = novoTempo;
166         aux->maquina.codigoMaquina = novoCod;
167     }
168     return h;
169 }
```

Figura 10

Na função *AlteraMaquina (Figura 10)* é criado um apontador *aux* do tipo máquina que percorre toda a lista até encontrar a máquina desejada, e quando encontra os dados inseridos na função é possível alterar os campos da máquina caso esta exista.

```
/**
 * .
 * @brief Mostra todas as maquinas existentes
 * @param [in]  Apontador para inicio da Lista
 */

void MostraTodasMaquinas(ListaMaquinas* h) {
    ListaMaquinas* aux = h;
    if (h != NULL) {

        if (aux) {
            while (aux) {
                printf("\n Maquina numero: %d", aux->maquina.codigoMaquina);
                printf("\n Demora %d para realizar a operacao", aux->maquina.tempo);
                printf("\n-----------------------------------------");
                aux = aux->next;
            }

        }
        else
        {
            printf("\n Não existem maquinas");
        }
    }
    else
    {
        printf("\n Lista vazia");
    }
}
```

Figura 11

A função *MostraTodasMaquinas (Figura 11)* possibilita a visualização das máquinas que foram criadas na consola, bem como os seus parâmetros, enquanto na função *MostraUmaMaquina* apenas é executada a demonstração de uma máquina.

Arquivo *FuncoesJobs.c e FuncoesOperacoes.c*

Nos arquivos de origem *FuncoesJobs.c* e *FuncoesOperacoes.c* foram seguidos os mesmos critérios e metodologias do que as aplicadas no arquivo *FuncoesMaquinas.c*, que foi supra explicado função por função, alterando apenas as variáveis e parâmetros requeridos pelas structs.

Arquivo *FuncoesCalculo.c*

Neste arquivo estão as 3 funções requeridas no projeto no que diz respeito ao cálculo do tempo mínimo e máximo de um job, e à média de cada operação desenvolvida. Durante a realização do programa, por falha da correta leitura do enunciado, supôs-se que o objetivo seria calcular também a média de um job, desse modo, quando o erro foi detetado decidiu-se deixar o extra no programa.

Arquivo *Main.c*

O arquivo *Main.c* permite a inserção dos dados da tabela fornecida e a posterior aplicação dos mesmos às funções criadas nos restantes arquivos.

Testes e resultados

A apresentação de testes e resultados não foi possível, uma vez que, apesar da não identificação de erros por parte do programa e dos membros do grupo, a execução do projeto não expõem resultados, como é possível observar na Figura.

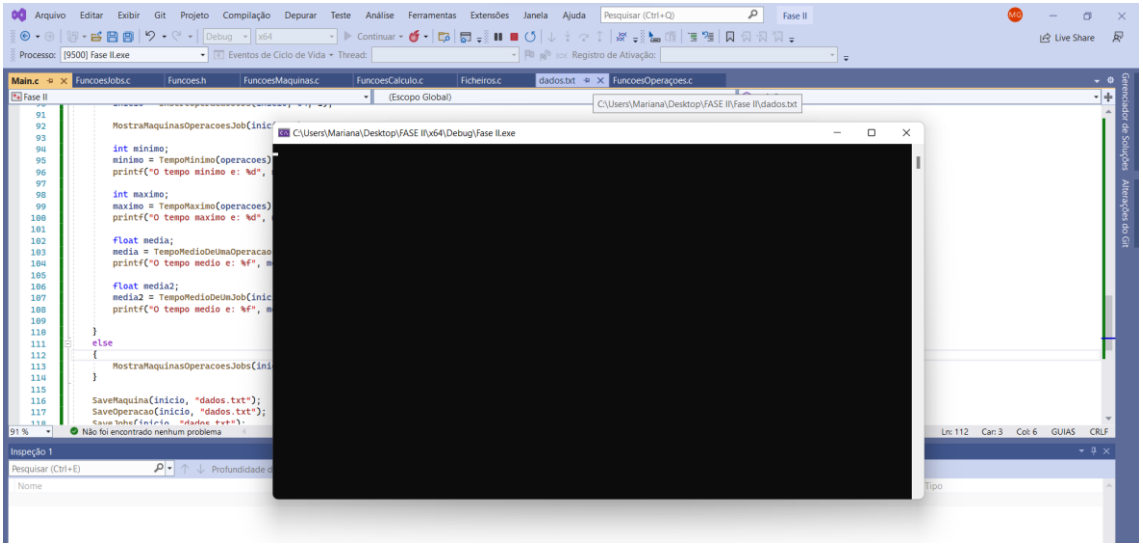

Figura 12

Sendo que o projeto foi revisto diversas vezes, pelos diferentes elementos constituintes do grupo, e não foram detetados defeitos que permitissem o correto funcionamento e o decorrer do programa desenvolvido que a equipa não consegue solucionar.

Ficheiros.c

No arquivo dos *Ficheiros.c* construímos funções com o intuito de preservar e ler informações sobre a lista de jobs. Primeiramente, criamos três funções base *SaveMaquina, SaveOperacao* e *SaveJobs* que guardam as informações no formato binário, tal como na função *SaveAll* onde a função percorre toda a lista de jobs guardando as suas componentes.

```
89  bool SaveAll(ListaJobs* h, char* fileName) {
90      if (h == NULL) return false;
91      FILE* fp;
92
93      if ((fp = fopen(fileName, "wb")) == NULL) return false;
94
95      //grava n registos no ficheiro
96      ListaJobs* auxJobs = h;
97      TodaInformacao auxFile; //struct para gravar em ficheiro!
98      while (auxJobs) {
99          //Colocar no registo de ficheiro a inf que está no registo de memória
100         auxFile.codigoJob = auxJobs->job.codigoJob;
101         ListaOperacoes* auxOperacoes = auxJobs->listaOperacoes;
102         ListaMaquinas* auxMaquinas = auxOperacoes->listaMaquinas;
103         //percorre a lista das operacoes
104         while (auxOperacoes) {
105             auxFile.codigoOperacao = auxOperacoes->operacao.codigoOperacao;
106             fwrite(&auxFile, sizeof(auxFile), 1, fp);
107             while (auxMaquinas) {
108                 auxFile.codigoMaquina = auxMaquinas->maquina.codigoMaquina;
109                 auxFile.tempo = auxMaquinas->maquina.tempo;
110                 fwrite(&auxFile, sizeof(auxFile), 1, fp);
111                 auxMaquinas = auxMaquinas->next;
112             }
113             auxOperacoes = auxOperacoes->next;
114         }
115         auxJobs = auxJobs->next;
116     }
117
118     fclose(fp);
119     return true;
120 }
```

Figura 13

```
207  ListaJobs* GetAll(char* fileName, ListaJobs* h) {
208
209      FILE* fp;
210      ListaJobs* aux = h;
211      Job j;
212
213      if (h == NULL) return NULL;
214
215      if ((fp = fopen(fileName, "rb")) == NULL) return NULL;
216
217      TodaInformacao auxFile;
218
219      while (fread(&auxFile, sizeof(auxFile), 1, fp)) {
220          //aux = ProcuraJobs(h, auxFile.job.codigoJob);
221          //ListaJobs* InsereOperacoesJobs(ListaJobs * h, Operacao * o, int codJob);
222          h = InsereOperacaoJobs(h, &auxFile.codigoOperacao, auxFile.codigoJob);
223          aux->listaOperacoes = InsereMaquinaOperacao(aux->listaOperacoes, &auxFile.codigoMaquina, auxFile.codigoJob);
224      }
225      fclose(fp);
226      return h;
227  }
```

Figura 14

Seguidamente, criamos funções para realizar o processo inverso, ou seja, ler os dados armazenados através das funções *GetAllJobs*, *GetAllOperacoes* e *GetAllMaquinas,* que leem os dados binários armazenados. Na função *GetAll* são estabelecidas as ligações entre os jobs e as operações, as operações e as máquinas obtidas nas funções anteriores. Relativamente à função *GetData* procedemos à leitura de um documento txt, para a formação da lista de jobs.

Conclusão

Este trabalho teve como propósito consolidar e praticar todos os conhecimentos adquiridos ao longo das aulas, tendo a funcionalidade de tentar interligar todo esse conhecimento de modo a ser possível desenvolver todas as funções requeridas pelo projeto.

Além de tudo, foi um trabalho que nos exigiu uma grande quantidade tempo, sendo que foram surgindo diversas adversidades, que com esforço e dedicação foram devidamente ultrapassadas e compreendidas sem nunca serem retardadas, procurando obter sempre os resultados e conhecimentos desejados.

A conclusão deste projeto apenas foi possível devido à interação e entreajuda dos diferentes elementos do grupo, a partilha dos diversos conhecimentos que cada integrante permitiu que todos evoluíssem de uma forma mais consciente e intuitiva, sendo que o recurso a lápis e papel para a explicação de diferentes ideias, era um ponto fulcral durante todas as sessões.

Desta forma, com a execução deste trabalho conseguimos adquirir uma melhor perceção do que são as estruturas de dados dinâmicas na linguagem c, bem como consolidar a matéria da cadeira anterior.