

Processo de escalonamento e *process plan* em linguagem C

Diana Alexandra da Costa Dinis, nº 23552

Licenciatura em Engenharia Informática Médica

Relatório Estruturas de Dados Avançadas

Professor Doutor João Carlos Cardoso da Silva

Barcelos, junho de 2022

Índice de conteúdos

Índice de figuras	iii
Índice de tabelas	v
Introdução.....	1
Propósitos e objetivos.....	2
Estruturas de dados	3
Funções de recolher informações	6
Funções de manipulação	6
Funções de ficheiro	9
Funções de interação	10
Funções de verificação.....	15
Função de listar.....	17
Funções de menu.....	17
Testes realizados.....	20
Conclusão	30
Bibliografia.....	31
Anexos.....	32

Índice de figuras

Figura 1 - Struct Job	3
Figura 2 - Process plan dentro do programa.	5
Figura 3 - Função ultimoJob()	6
Figura 4 - Função ultimaOp()	6
Figura 5 - Função inserirJob()	7
Figura 6 - Função removerJob()	8
Figura 7 - Função removerOperacao()	8
Figura 8 - Função removerMaquina()	9
Figura 9 - Função lerFicheiro()	9
Figura 10 - Função escreverFicheiro()	10
Figura 11 - Função inserirJ()	11
Figura 12 - Função removerJ()	12
Figura 13 - Função inserirO()	12
Figura 14 - Função removerO()	13
Figura 15 - Função inserirM()	13
Figura 16 - Função removerM()	14
Figura 17 - Função alterarM()	14
Figura 18 - Função qualJob()	15
Figura 19 - Função qualOp()	15
Figura 20 - Função verificarJobExistente()	16
Figura 21 - Função verificarOpExistente()	16
Figura 22 - Função verificarMaqExistente()	17
Figura 23 - Função listarJob()	17
Figura 24 - Função pressione()	18
Figura 25 - Função menu()	18
Figura 26 - Função alteracaoO()	18
Figura 27 - Função alteracaoJ()	18
Figura 28 - Função main()	19
Figura 29 - Teste à função lerFicheiro()	20
Figura 30 - Função para verificar existência de job.....	20
Figura 31 - Chamar a função verificarJobExistente()	21
Figura 32 - Terminal window do teste da função verificarJobExistente()	21
Figura 33 - Função verificarJobExistente() a funcionar.....	22
Figura 34 - Função verificarOpExistente a funcionar.....	22
Figura 35 - Primeira proposta para utilizar a função verificarMaqExistente()	23
Figura 36 - Terminal window do primeiro teste da função verMaqExistente()	23
Figura 37 - Função verMaqExistente() a funcionar.....	24
Figura 38 – Listagem do job 0.....	24
Figura 39 – Exemplo de inserção do process plan proposto.....	25

Figura 40 - Função inserirO() a funcionar	25
Figura 41 - Função inserirM() a funcionar.....	26
Figura 42 - Função reduzirJob() a funcionar.	26
Figura 43 - Debug da função removerJob()	27
Figura 44 - Função removerOperacao() a funcionar.	28
Figura 45 - Função removerMaquina() a apresentar erro.	28
Figura 46 - Função removerMaquina() a funcionar.	29
Figura 47 - Função alterarMaquina() a funcionar.....	29

Índice de tabelas

Tabela 1 - Jobshop	3
--------------------------	---

Introdução

Este projeto foi desenvolvido no âmbito da UC de Estruturas de Dados Avançadas, sendo este relatório referente à Fase 2 de entrega do projeto de avaliação à UC em questão, desenvolvido pelo aluno Diana Alexandra da Costa Dinis, de número de estudante 23552, com acompanhamento do Professor Doutor João Carlos Cardoso da Silva.

A estrutura deste relatório será primeiramente a descrição dos propósitos e objetivos desta segunda fase. Após está o capítulo das estruturas de dados, onde o código estará exposto com algumas explicações de como este encaixa no programa a ser desenvolvido. A parte dos testes estará descrito testes realizados ao código de modo a garantir o seu funcionamento dinâmico e eficaz. Conclusão com as últimas considerações. Bibliografia com alguns links e livros acessados ao longo do desenvolvimento do projeto e, por fim, os anexos, onde estarão disponíveis documentos escritos à mão que auxiliaram o raciocínio no desenvolvimento do código.

Permitirá avaliar conhecimentos de implementação de código na linguagem C e, mais a fundo, a eficiência destes programas a resolver os problemas a que estes estão pensados, tendo de passar, naturalmente, por uma série de testes de modo a ter-se a certeza que o programa é funcional.

Propósitos e objetivos

O propósito desta segunda fase do trabalho prático será a demonstração de conhecimento e capacidade de resolver os problemas de *process plan*, de modo a que os jobs estejam organizados de forma eficiente e a que todas as operações possam estar a ser desenvolvidas no menor tempo possível, tendo em conta as máquinas disponíveis a cada tempo t . Aproveitando também para melhorar funções recorrentes da fase 1 do trabalho e a otimização destas.

A implementação de um programa que tenha a capacidade de filtrar estas informações, escolhendo sempre o melhor caminho será um desafio. No entanto, com a orientação do professor e com os ficheiros de apoio disponibilizados pelo professor e alguns pela internet, este trabalho foi realizado.

Estruturas de dados

De início, a struct que define todo o *process plan* será diferente da primeira fase. Adicionou-se um campo que permite identificar o número do Job, o que está visível na figura 1, abaixo.

```
typedef struct Job
{
    int nmrJob;
    int nmrOperacao;
    int maquina;
    int unidadeTempo;
    struct Job* opSeguinte;
}job;
```

Figura 1 - Struct Job

As operações do ficheiro *main.c* ligadas aos menus de modo a tornar o programa funcional e acessível foram maioritariamente reutilizadas da primeira fase, com algumas alterações de texto e a inserção de mais algumas variantes.

Assim como na fase 1, neste trabalho as funções estão todas enunciadas no ficheiro *header.h* e estão organizadas em categorias também, separadas por comentários, que especificam onde as funções de encaixam.

A tabela 1 abaixo representa o *jobshop* que se tem de traduzir na lista ligada:

Process plan	Operação						
	1	2	3	4	5	6	7
pr_{1,2}	(1,3) [4,5]	(2,4) [4,5]	(3,5) [5,6]	(4,5,6,7,8) [5,5,4,5,9]	-	-	-
pr_{2,2}	(1,3,5) [1,5,7]	(4,8) [5,4]	(4,6) [1,6]	(4,7,8) [4,4,7]	(4,6) [1,2]	(1,6,8) [5,6,4]	(4) [4]
pr_{3,3}	(2,3,8) [7,6,8]	(4,8) [7,7]	(3,5,7) [7,8,7]	(4,6) [7,8]	(1,2) [1,4]	-	-
pr_{4,2}	(1,3,5) [4,3,7]	(2,8) [4,4]	(3,4,6,7) [4,5,6,7]	(5,6,8) [3,5,5]	-	-	-
pr_{5,1}	(1) [5]	(2,4) [4,5]	(3,8) [4,4]	(5,6,8) [3,3,3]	(4,6) [5,4]	-	-
pr_{6,3}	(1,2,3) [3,5,6]	(4,5) [7,8]	(3,6) [9,8]	-	-	-	-
pr_{7,2}	(3,5,6) [4,5,4]	(4,7,8) [4,6,4]	(1,3,4,5) [3,3,4,5]	(4,6,8) [4,6,5]	(1,3) [3,3]	-	-
pr_{8,1}	(1,2,6) [3,4,4]	(4,5,8) [6,5,4]	(3,7) [4,5]	(4,6) [4,6]	(7,8) [1,2]	-	-

Tabela 1 - Jobshop

Na figura 2 abaixo está representada a lista ligada listada, após ser inserida no ficheiro do projeto com recurso às funções desenvolvidas.

```

-----* Listar os jobs
J: 8 // O: 1 // M: 1 // T: 3
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 6 // T: 4

J: 8 // O: 2 // M: 4 // T: 6
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 8 // T: 4

J: 8 // O: 3 // M: 3 // T: 4
J: 8 // O: 3 // M: 7 // T: 5

J: 8 // O: 4 // M: 4 // T: 4
J: 8 // O: 4 // M: 6 // T: 6

J: 8 // O: 5 // M: 7 // T: 1
J: 8 // O: 5 // M: 8 // T: 2

J: 7 // O: 5 // M: 3 // T: 3
J: 7 // O: 5 // M: 1 // T: 3

J: 7 // O: 4 // M: 8 // T: 5
J: 7 // O: 4 // M: 6 // T: 6
J: 7 // O: 4 // M: 4 // T: 4

J: 7 // O: 3 // M: 5 // T: 5
J: 7 // O: 3 // M: 4 // T: 4
J: 7 // O: 3 // M: 3 // T: 3
J: 7 // O: 3 // M: 1 // T: 3

J: 7 // O: 2 // M: 8 // T: 4
J: 7 // O: 2 // M: 7 // T: 6
J: 7 // O: 2 // M: 4 // T: 4

J: 7 // O: 1 // M: 6 // T: 4
J: 7 // O: 1 // M: 5 // T: 5
J: 7 // O: 1 // M: 3 // T: 4

J: 6 // O: 1 // M: 1 // T: 3
J: 6 // O: 1 // M: 2 // T: 5
J: 6 // O: 1 // M: 3 // T: 6

J: 6 // O: 2 // M: 4 // T: 7
J: 6 // O: 2 // M: 5 // T: 8

J: 6 // O: 3 // M: 3 // T: 9
J: 6 // O: 3 // M: 6 // T: 8

J: 5 // O: 5 // M: 6 // T: 4
J: 5 // O: 5 // M: 4 // T: 5

J: 5 // O: 4 // M: 8 // T: 3
J: 5 // O: 4 // M: 6 // T: 3
J: 5 // O: 4 // M: 5 // T: 3

J: 5 // O: 3 // M: 8 // T: 4
J: 5 // O: 3 // M: 3 // T: 4

J: 5 // O: 2 // M: 4 // T: 5
J: 5 // O: 2 // M: 2 // T: 4

J: 5 // O: 1 // M: 1 // T: 5

J: 4 // O: 1 // M: 1 // T: 4
J: 4 // O: 1 // M: 3 // T: 3
J: 4 // O: 1 // M: 5 // T: 7

J: 4 // O: 2 // M: 2 // T: 4
J: 4 // O: 2 // M: 8 // T: 4

J: 4 // O: 3 // M: 3 // T: 4
J: 4 // O: 3 // M: 4 // T: 5
J: 4 // O: 3 // M: 6 // T: 6
J: 4 // O: 3 // M: 7 // T: 7

```

```

J: 4 // O: 4 // M: 5 // T: 3
J: 4 // O: 4 // M: 6 // T: 5
J: 4 // O: 4 // M: 8 // T: 5

J: 3 // O: 1 // M: 2 // T: 7
J: 3 // O: 1 // M: 3 // T: 6
J: 3 // O: 1 // M: 8 // T: 8

J: 3 // O: 2 // M: 4 // T: 7
J: 3 // O: 2 // M: 8 // T: 7

J: 3 // O: 3 // M: 3 // T: 7
J: 3 // O: 3 // M: 5 // T: 8
J: 3 // O: 3 // M: 7 // T: 7

J: 3 // O: 4 // M: 4 // T: 7
J: 3 // O: 4 // M: 6 // T: 8

J: 3 // O: 5 // M: 1 // T: 1
J: 3 // O: 5 // M: 2 // T: 4

J: 2 // O: 7 // M: 4 // T: 4

J: 2 // O: 6 // M: 8 // T: 4
J: 2 // O: 6 // M: 6 // T: 6
J: 2 // O: 6 // M: 1 // T: 5

J: 2 // O: 5 // M: 6 // T: 2
J: 2 // O: 5 // M: 4 // T: 1

J: 2 // O: 4 // M: 8 // T: 7
J: 2 // O: 4 // M: 7 // T: 4
J: 2 // O: 4 // M: 4 // T: 4

J: 2 // O: 3 // M: 6 // T: 6
J: 2 // O: 3 // M: 4 // T: 1

J: 2 // O: 2 // M: 4 // T: 5
J: 2 // O: 1 // M: 5 // T: 7
J: 2 // O: 1 // M: 3 // T: 5
J: 2 // O: 1 // M: 1 // T: 1

J: 1 // O: 1 // M: 1 // T: 4
J: 1 // O: 1 // M: 3 // T: 5

J: 1 // O: 2 // M: 2 // T: 4
J: 1 // O: 2 // M: 4 // T: 5

J: 1 // O: 3 // M: 3 // T: 5
J: 1 // O: 3 // M: 5 // T: 6

J: 1 // O: 4 // M: 4 // T: 5
J: 1 // O: 4 // M: 5 // T: 5
J: 1 // O: 4 // M: 6 // T: 4
J: 1 // O: 4 // M: 7 // T: 5
J: 1 // O: 4 // M: 8 // T: 9

Prima qualquer tecla para ir para o menu!

```

Figura 2 - Process plan dentro do programa.

Funções de recolher informações

Estas funções são funções que irão permitir recolher informação de forma automática, ao ler da lista ligada.

A função de inserir um Job nesta segunda fase será auxiliada por uma função que determina o último job, permitindo inserir um job consecutivo, permitindo haver uma ordem crescente, que não ande a saltar de passos em passos. Essa função de determinar o último job, `ultimoJob()`, está representada na figura 3 abaixo:

```
int ultimoJob(job* lista) //vai procurar qual é o ultimo valor de job.
{
    int ultimo = 0;

    while (lista != NULL)
    {
        if (ultimo <= lista->nmrJob) ultimo = lista->nmrJob;
        lista = lista->seguinte;
    }
    return ultimo;
}
```

Figura 3 - Função `ultimoJob()`

A função de inserir uma nova operação também será auxiliada por uma função `ultimaOp()` que irá avaliar o valor da última operação dentro do job em específico. Esta função está presente na figura 4 abaixo.

```
int ultimaOp(job* lista, int numJob)
{
    int ultimo = 0;

    while (lista != NULL)
    {
        if (ultimo <= lista->nmrOperacao && numJob == lista->nmrJob) ultimo = lista->nmrOperacao;
        lista = lista->seguinte;
    }
    return ultimo;
}
```

Figura 4 - Função `ultimaOp()`

Funções de manipulação

Estas funções são as descritas de modo a alterar os valores da lista ligada, como inserir novos elementos, removê-los e etc., como se verá ao longo deste subcapítulo.

A função de inserir o novo Job, `inserirJob()`, está representada, portanto, na figura 5. Esta função permite inserir os novos blocos seguindo uma ordem crescente de jobs.

```
job* inserirJob(job* inicio, int numeroJob, int numeroOp, int numeroMaq, int numeroTemp)
{
    int i = 0;
    job* novo = malloc(sizeof(job)), *ptr;

    if (novo != NULL)
    {
        novo->nmrJob = numeroJob;
        novo->nmrOperacao = numeroOp;
        novo->maquina = numeroMaq;
        novo->unidadeTempo = numeroTemp;
        novo->seguinte = inicio;
    }

    if (inicio == NULL)
    {
        inicio = novo;
    }

    else if (novo->nmrJob <= inicio->nmrJob)
    {
        novo->seguinte = inicio;
        inicio = novo;
    }

    else
    {
        ptr = inicio;

        while (ptr->seguinte != NULL && ptr->seguinte->nmrJob <= novo->nmrJob)
        {
            ptr = ptr->seguinte;
        }

        novo->seguinte = ptr->seguinte;
        ptr->seguinte = novo;
    }

    return (inicio);
}
```

Figura 5 - Função `inserirJob()`

A próxima função a ser desenvolvida foi a função `removerJob()` que, como o nome indica, irá percorrer toda a lista ligada, eliminando todos os jobs que possuam o número igual ao enviado por parâmetro. Esta função encontra-se na figura 6 **Erro! A origem da referência não foi encontrada.** na página seguinte.

Além de remover um job, esta função também permite reduzir o número dos jobs consequentes, de modo a manter uma sequência de jobs contínua.

```

job* removerJob(job* lista, int numJob)
{
    job* jobAnterior, *jobAtual = lista;

    while (jobAtual != NULL)
    {
        jobAtual = lista;

        if (numJob == lista->numJob)
        {
            lista = jobAtual->seguinte;
            free(jobAtual);
        }
        else
        {
            jobAnterior = lista;

            while ((jobAtual != NULL) && (jobAtual->numJob != numJob))
            {
                jobAnterior = jobAtual;

                if (jobAnterior->numJob > numJob)
                {
                    jobAnterior->numJob = jobAnterior->numJob - 1;
                }

                jobAtual = jobAtual->seguinte;
            }

            if (jobAtual != NULL) {
                jobAnterior->seguinte = jobAtual->seguinte;
                free(jobAtual);
            }
        }
    }

    return(lista);
}

```

Figura 6 - Função *removerJob()*.

A próxima função é a **removerOperacao()**, que se encontra na figura 7 abaixo. Esta função é parecida à anterior, apenas diferenciando nas condições de acesso aos ciclos de eliminação, tendo de incluir o número da operação. Um dos truques para que esta função funcione está dentro da condição do ciclo **while**, com aquele cálculo do **||**, sendo este o símbolo da operação **ou**. Esta função também reduz o número das operações seguintes à removida.

```

job* removerOperacao(job* inicio, int numJob, int numOp)
{
    job* jobAnterior, *jobAtual = inicio;

    while (jobAtual != NULL)
    {
        jobAtual = inicio;

        if (inicio->numJob == numJob && numOp == inicio->numOp)
        {
            inicio = jobAtual->seguinte;
            free(jobAtual);
        }
        else
        {
            jobAnterior = inicio;

            while ((jobAtual != NULL) && (jobAtual->numOperacao != numOp || jobAtual->numJob != numJob))
            {
                jobAnterior = jobAtual;

                if (jobAnterior->numOperacao > numOp && jobAnterior->numJob == numJob)
                {
                    jobAnterior->numOperacao = jobAnterior->numOperacao - 1;
                }

                jobAtual = jobAtual->seguinte;
            }

            if (jobAtual != NULL) {
                jobAnterior->seguinte = jobAtual->seguinte;
                free(jobAtual);
            }
        }
    }

    return(inicio);
}

```

Figura 7 - Função *removerOperacao()*.

Por último, a função `removerMaquina()` presente na figura 8 abaixo permite remover uma máquina dentro da operação de um job em específico. É uma variante mais simples das anteriores de remoção, visto que não é necessário reduzir as restantes máquinas, por estas não apresentarem uma sequência. É uma função parecida à função de remover operações presente na fase 1 deste trabalho.

```

job* removerMaquina(job* inicio, int numJob, int numOp, int numMaq)
{
    job* jobAnterior, * jobAtual = inicio;

    while (jobAtual != NULL)
    {
        jobAtual = inicio;

        if (inicio->numJob == numJob && numOp == inicio->numJob && inicio->maquina == numMaq)
        {
            inicio = jobAtual->seguirte;
            free(jobAtual);
        }
        else
        {
            jobAnterior = inicio;
            jobAtual = jobAtual->seguirte;

            while ((jobAtual != NULL) && (jobAtual->numOperacao != numOp || jobAtual->numJob != numJob || jobAtual->maquina != numMaq))
            {
                jobAnterior = jobAtual;
                jobAtual = jobAtual->seguirte;
            }

            if (jobAtual != NULL)
            {
                jobAnterior->seguirte = jobAtual->seguirte;
                free(jobAtual);
            }
        }
    }

    return(inicio);
}

```

Figura 8 - Função `removerMaquina()`.

Funções de ficheiro

Após as funções de inserir e listar estarem completas, as de escrever a lista ligada num ficheiro e a de ler a lista do ficheiro foram as próximas a serem desenvolvidas.

Na fase 1, a função de ler do ficheiro `lerFicheiro()` apresentava um bug que não permitia que a função executá-se corretamente. Esse erro foi, portanto, corrigido e já se encontra funcional para esta fase. Esta função está representada na figura 9 abaixo.

```

job* lerFicheiro()
{
    int numeroJob, numeroOp, numeroMaq, numeroTemp;
    job* lista = NULL;
    FILE* outfile = fopen("processplan.txt", "r");

    if (outfile != NULL)
    {
        while (!feof(outfile))
        {
            fscanf(outfile, "J: %d // O: %d // M: %d // T: %d\n", &numeroJob, &numeroOp, &numeroMaq, &numeroTemp);
            lista = inserirJob(lista, numeroJob, numeroOp, numeroMaq, numeroTemp);
        }
    }
    else printf("Ficheiro vazio!");

    fclose(outfile);

    return(lista);
}

```

Figura 9 - Função `lerFicheiro()`.

A função que pouco mudou entre estas duas fases do trabalho foi a função de escrever no ficheiro, que é chamada no fim do programa, se o utilizador utilizar o menu principal para fechar o programa. Esta função permite guardar todos os dados que estariam na lista ligada naquele momento no ficheiro, escrevendo por cima do que o que já estaria no ficheiro. Sendo assim, ao guardar uma lista ligada ao fechar o programa, perdemos toda a informação do ficheiro que lá estaria. Esta função `escreverFicheiro()` está presente na figura 10 abaixo:

```
job* escreverFicheiro(job* inicio)
{
    FILE* infile = fopen("processplan.txt", "w"); //abrir ficheiro para escrever

    for (; inicio != NULL; inicio = inicio->seguinte)
    {
        fprintf(infile, "J: %d // O: %d // M: %d // T: %d\n", inicio->nmrJob, inicio->nmrOperacao, inicio->maquina, inicio->unidadeTempo);
    }

    if (fprintf != 0) printf("Dados guardados com sucesso!\n\n");
    else
    {
        printf("Ocorreu um erro na escrita do ficheiro!");
        pressione();
    }

    fclose(infile);
}
```

Figura 10 - Função `escreverFicheiro()`

Funções de interação

Como na fase anterior, estas funções são as responsáveis por fazer o programa dinâmico, permitindo que haja a interação com o utilizador, que irá introduzir os valores ou afins de modo a que o programa corra as funções de manipulação da lista e que permita navegar perante os diferentes menus e opções.

A primeira função de interação desenvolvida terá sido a associada a inserir jobs no *process plan*, denominada por `inserirJ`, presente na figura 11 abaixo, permite ao programa recolher as informações que necessita para inserir um novo job na lista ligada.

Esta função permite incluir mais operações ao mesmo job, perguntando ao utilizador se o deseja fazer ou não. Há o mesmo processo para as máquinas nas operações. Esta funcionalidade permite que o programa seja mais user friendly, evitando a necessidade de recorrer sempre ao menu principal para adicionar operações ao mesmo job e máquinas às mesmas operações dentro do job.

Há também um sistema que permite controlar se já há uma máquina associada à operação daquele job em específico, como se vai falar mais para a frente neste relatório.


```

job* inserirJ(job* inicio)
{
    int novoJob, novaOperacao = 0, novaMaquina, novoTempo, maisJ = 1, mais0 = 1;

    novoJob = ultimoJob(inicio) + 1;

    printf("Job: %d", novoJob);

    do
    {
        novaOperacao = novaOperacao + 1;
        printf("\nOperação: %d", novaOperacao);

        do
        {
            printf("\nMáquina: ");
            scanf("%d", &novaMaquina);

            int verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMaquina);

            while (novaMaquina > MAXMACHINE)
            {
                printf("\nA máquina %d não existe. \nInsira outra máquina: ", novaMaquina);
                scanf("%d", &novaMaquina);
            }

            while (verMq == 1)
            {
                printf("\nA máquina %d já se encontra associada à operação %d. \nInsira outra máquina: ", novaMaquina, novaOperacao);
                scanf("%d", &novaMaquina);
                verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMaquina);
            }

            printf("Tempo de processamento: ");
            scanf("%d", &novoTempo);

            inicio = inserirJob(inicio, novoJob, novaOperacao, novaMaquina, novoTempo);

            printf("\n\nDeseja adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
            scanf("%d", &mais0);

            while (mais0 != 0 && mais0 != 1)
            {
                printf("\nOpção inválida. \nQuer adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
                scanf("%d", &mais0);
            }

        } while (mais0 == 1);

        printf("\n\nDeseja adicionar mais alguma operação ao job?\n1-sim ; 0 - Não\nOpção: ");
        scanf("%d", &maisJ);

        while (maisJ != 0 && maisJ != 1)
        {
            printf("\nOpção inválida. \nQuer adicionar mais alguma operação ao job atual?\n1 - Sim; 0 - Não\nOpção: ");
            scanf("%d", &maisJ);
        }

    } while (maisJ == 1);

    pressione();
    return(inicio);
}

```

Figura 11 - Função *inserirJ()*

Esta função *inserirJ()* é auxiliada por funções que permitem que haja uma sequência de jobs e operações, como um process plan terá de admitir. Estas funções irão avaliar os números máximos que terá os jobs, ou seja, o último job e outra função encarregue de fazer o mesmo processo para as operações. Após, a esse resultado será adicionado mais 1 para que siga essa sequência. Essas funções de recolher os dados já foram mencionadas anteriormente.

A próxima de interação será a função `removeJ()`, presente na figura 12 abaixo. Esta função permite recolher com o utilizador qual será o job que este pretenderá eliminar do process plan. Assim como a função de inserir, esta também permite que haja a remoção de mais do que um job, sem necessitar ir ao menu principal.

```

job* removeJ(job* lista)
{
    int jobRemover, mais = 1;

    do
    {
        printf("Job a remover: ");
        scanf("%d", &jobRemover);

        int ver = verificarJobExistente(lista, jobRemover);

        while (ver != 1)
        {
            printf("\nO job %d não existe na lista ligada.\nInsira um job válido para remover: ", jobRemover);
            scanf("%d", &jobRemover);
            ver = verificarJobExistente(lista, jobRemover);
        }

        lista = removerJob(lista, jobRemover);

        printf("\n\nRemover mais algum job?\n1 - Sim; 0 - Não\nOpção: ");
        scanf("%d", &mais);
    } while (mais == 1);

    pressione();
    return lista;
}
  
```

Figura 12 - Função `removeJ()`.

Assim como a de inserir, também é auxiliada pela função que irá avaliar a existência do job que o utilizador inseriu.

A função `inserirO()`, presente na figura 13 abaixo, permite ao utilizador escolher adicionar uma operação a um job já existente.

```

job* inserirO(job* inicio, int numJob)
{
    int novoJob = numJob, novaOperacao = ultimaOp(inicio, numJob) + 1, novaMaquina, novoTempo, maisO = 1;

    printf("\nJob: %d", novoJob);
    printf("\nOperação: %d", novaOperacao);

    do
    {
        printf("\nMáquina: ");
        scanf("%d", &novaMaquina);

        int verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMaquina);

        while (novaMaquina > MAXMACHINE)
        {
            printf("\nA máquina %d não existe. \nInsira outra máquina: ", novaMaquina);
            scanf("%d", &novaMaquina);
        }

        while (verMq == 1)
        {
            printf("\nA máquina %d já se encontra associada à operação %d. \nInsira outra máquina: ", novaMaquina, novaOperacao);
            scanf("%d", &novaMaquina);
            verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMaquina);
        }

        printf("Tempo de processamento: ");
        scanf("%d", &novoTempo);

        inicio = inserirJob(inicio, novoJob, novaOperacao, novaMaquina, novoTempo);

        printf("\n\nDeseja adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
        scanf("%d", &maisO);

        while (maisO != 0 && maisO != 1)
        {
            printf("\nOpção inválida. \nQuer adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
            scanf("%d", &maisO);
        }
    } while (maisO == 1);

    pressione();
    return inicio;
}
  
```

Figura 13 - Função `inserirO()`.

Esta função permite inserir mais que uma máquina na mesma operação e inserir mais do que uma operação de uma vez, de modo a que o utilizador não necessite de voltar ao menu principal de modo a inserir mais operações dentro do mesmo job.

A próxima função é a **removerO()**. Esta função permite eliminar uma operação dentro de um job, permitindo também eliminar mais do que uma operação. Esta função encontra-se na figura 14 abaixo.

```

job* removerO(job* inicio, int numJob)
{
    int opRemover, mais = 1;

    do
    {
        printf("Operação do job %d a remover: ", numJob);
        scanf("%d", &opRemover);

        int ver = verificarOpExistente(inicio, numJob, opRemover);

        while (ver != 1)
        {
            printf("\n\nA operação %d não existe no job %d.\nInsira uma operação válido para remover: ", opRemover, numJob);
            scanf("%d", &opRemover);
            ver = verificarOpExistente(inicio, numJob, opRemover);
        }

        inicio = removerOperacao(inicio, numJob, opRemover);

        printf("\n\nRemover mais alguma operação do job %d?\n1 - Sim; 0 - Não\nOpção: ", numJob);
        scanf("%d", &mais);
    } while (mais == 1);

    pressione();
    return inicio;
}

```

Figura 14 - Função *removerO()*.

A função **inserirM()** permite inserir uma nova máquina dentro de uma operação de um job em específico, havendo a possibilidade de inserir mais do que uma máquina sem ser necessário voltar ao menu principal. Esta função verifica se a máquina existe, ou seja, se o valor dela não é superior a 9 e se ainda não foi adicionada. Esta função encontra-se na figura 15 abaixo.

```

job* inserirM(job* inicio, int numJob, int numOp)
{
    int novoJob = numJob, novaOperacao = numOp, novaMachina, novoTempo, mais0 = 1;

    printf("\nJob: %d", novoJob);
    printf("\nOperação: %d", novaOperacao);

    do
    {
        printf("\nMáquina: ");
        scanf("%d", &novaMachina);

        int verMach = verificarMachExistente(inicio, novoJob, novaOperacao, novaMachina);

        while (novaMachina > MAXMACHINE)
        {
            printf("\nA máquina %d não existe. \nInsira outra máquina: ", novaMachina);
            scanf("%d", &novaMachina);
        }

        while (verMach == 1)
        {
            printf("\nA máquina %d já se encontra associada à operação %d. \nInsira outra máquina: ", novaMachina, novaOperacao);
            scanf("%d", &novaMachina);
            verMach = verificarMachExistente(inicio, novoJob, novaOperacao, novaMachina);
        }

        printf("Tempo de processamento: ");
        scanf("%d", &novoTempo);

        inicio = inserirJob(inicio, novoJob, novaOperacao, novaMachina, novoTempo);

        printf("\n\nDeseja adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
        scanf("%d", &mais0);

        while (mais0 != 0 && mais0 != 1)
        {
            printf("\nOpção inválida.\nQuer adicionar mais alguma máquina à operação atual?\n1 - Sim; 0 - Não\nOpção: ");
            scanf("%d", &mais0);
        }
    } while (mais0 == 1);

    pressione();
    return inicio;
}

```

Figura 15 - Função *inserirM()*.

A função `removerM()`, na figura 16, permite remover uma máquina de uma operação. Para tal, vai recolher os dados sobre qual máquina eliminar, mandando depois para a função `removerMaquina()`. Esta função também manda verificar a existência da máquina e permite que se remova mais do que uma máquina por vez.

```

job* removerM(job* inicio, int numJob, int numOp)
{
    int maqRemover, mais = 1;

    do
    {
        printf("Máquina da operação %d do job %d a remover: ", numOp, numJob);
        scanf("%d", &maqRemover);

        int ver = verificarMaqExistente(inicio, numJob, numOp, maqRemover);

        while (ver != 1)
        {
            printf("\nA máquina %d não existe na operação %d do job %d.\nInsira uma máquina válida para remover: ", maqRemover, numOp, numJob);
            scanf("%d", &maqRemover);
            ver = verificarMaqExistente(inicio, numJob, numOp, maqRemover);
        }

        inicio = removerMaquina(inicio, numJob, numOp, maqRemover);

        printf("\nRemover mais alguma máquina da operação %d do job %d?\n1 - Sim; 0 - Não\nOpção: ", numOp, numJob);
        scanf("%d", &mais);
    } while (mais == 1);

    pressione();
    return inicio;
}

```

Figura 16 - Função `removerM()`.

A função `alterarM()` permite ao utilizador alterar o tempo de processamento de uma máquina, sem precisar manualmente removê-la e inseri-la, sendo esse o processo que esta função faz, após questionar o utilizador sobre qual máquina quer alterar. Esta função encontra-se na figura 17 abaixo.

```

job* alterarM(job* inicio, int numJob, int numOp)
{
    int maqAlterar, mais = 1, temp;

    do
    {
        printf("Máquina da operação %d do job %d a alterar: ", numOp, numJob);
        scanf("%d", &maqAlterar);

        int ver = verificarMaqExistente(inicio, numJob, numOp, maqAlterar);

        while (ver != 1)
        {
            printf("\nA máquina %d não existe na operação %d do job %d.\nInsira uma máquina válida para alterar: ", maqAlterar, numOp, numJob);
            scanf("%d", &maqAlterar);
            ver = verificarMaqExistente(inicio, numJob, numOp, maqAlterar);
        }

        printf("\nTempo de processamento novo: ");
        scanf("%d", &temp);

        inicio = removerMaquina(inicio, numJob, numOp, maqAlterar);
        inicio = inserirJob(inicio, numJob, numOp, maqAlterar, temp);

        printf("\nAlterar mais alguma máquina da operação %d do job %d?\n1 - Sim; 0 - Não\nOpção: ", numOp, numJob);
        scanf("%d", &mais);
    } while (mais == 1);

    pressione();
    return inicio;
}

```

Figura 17 - Função `alterarM()`.

A função `qualJob()` presente na figura 18 abaixo tem o objetivo de apenas recolher qual job o utilizador irá querer alterar.

```
int qualJob()
{
    int job;

    printf("Qual é o job que pretende alterar?\n\nOpcao: ");
    scanf("%d", &job);

    return job;
}
```

Figura 18 - Função `qualJob()`.

A função `qualOp()` mostrada na figura 19 abaixo possui o mesmo propósito que a função `qualJob()`, com a diferença de que será qual a operação o utilizador querará modificar, retornando esse valor.

```
int qualOp(job*lista, int nmrJob)
{
    system("cls");
    listarJob(lista);
    int op;

    printf("\n\nQual é a operação dentro do job %d que pretende alterar?\n\nOpcao: ", nmrJob);
    scanf("%d", &op);

    return op;
}
```

Figura 19 - Função `qualOp()`.

Funções de verificação

Estas funções irão permitir verificar a existência de jobs, operações ou máquinas. Dependendo do contexto, poderá ser bom ou mau a pré existência destas operações.

Na figura abaixo temos presente a função `verificarExistenciaJob()`, presente na figura 20 abaixo, que tal como indica o nome, irá procurar na lista ligada um job pre existente com o número que será inserido no menu.

```
int verificarJobExistente(job*lista, int numJob)
{
    int sim = 0;
    while (lista != NULL && sim !=1 )
    {
        if (lista->nmrJob == numJob)
        {
            sim = 1;
        }
        else lista = lista->seguinte;
    }
    return sim;
}
```

Figura 20 - Função *verificarJobExistente()*.

Esta função irá retornar o valor 1 se a lista ligada contiver o job com o número analisado e 0 se não contiver. No caso, esta função é aplicada de modo a podermos modificar jobs, portanto é importante que retorne 1 para que o programa consiga correr como esperado.

Esta lógica também se aplica à procura da existência da operação dentro do job em específico, sendo que a função *verificarOpExistente()*, na figura 21 abaixo, é aplicada no mesmo contexto.

```
int verificarOpExistente(job* lista, int numJob, int numOp)
{
    int sim = 0;
    while (lista != NULL && sim != 1)
    {
        if (lista->nmrJob == numJob && lista->nmrOperacao == numOp)
        {
            sim = 1;
        }
        else lista = lista->seguinte;
    }
    return sim;
}
```

Figura 21 - Função *verificarOpExistente()*.

A última função de verificação é a de existência da máquina na mesma operação dentro do mesmo job, dada por *verificarMaqExistente()*, de modo a tornar impossível uma mesma máquina ser associada duas vezes à mesma operação dentro do mesmo job. Esta função não difere muito das anteriores, mas está presente na figura 22 abaixo.

```

int verificarMaqExistente(job* lista, int numJob, int numOp, int numMaq)
{
    int sim = 0;
    while (lista != NULL && sim != 1)
    {
        if (lista->nmrJob == numJob && lista->nmrOperacao == numOp && lista->maquina == numMaq)
        {
            sim = 1;
        }
        else lista = lista->seguinte;
    }
    return sim;
}

```

Figura 22 - Função *verificarMaqExistente()*.

Para esta função, igualmente às anteriores, se a operação do job tiver a máquina, vai retornar o valor 1 e se não tiver, vai retornar o valor 0. No contexto, é importante que a função retorne 0, de modo a confirmar a não existência da máquina naquela operação em específico.

Função de listar

Esta função *listarJob()*, na figura 23 abaixo, serve para poder mostrar na consola toda a lista ligada, separando com um `\n`, ou seja um *enter*, operações diferentes e com dois jobs diferentes, para ser mais fácil ao utilizador localizar o que necessita.

```

void listarJob(job* lista)
{
    job* inicio = lista;
    while (inicio != NULL)
    {
        printf("J: %d // O: %d // M: %d // T: %d\n", inicio->nmrJob, inicio->nmrOperacao, inicio->maquina, inicio->unidadeTempo);

        if (inicio->seguinte != NULL) if (inicio->nmrJob != inicio->seguinte->nmrJob) printf("\n\n");
        if (inicio->seguinte != NULL) if (inicio->nmrOperacao != inicio->seguinte->nmrOperacao) printf("\n");

        inicio = inicio->seguinte;
    }
}

```

Figura 23 - Função *listarJob()*.

Funções de menu

Por último, estas funções são responsáveis por dar a conhecer ao utilizador os ‘caminhos’ que terá de percorrer de modo a poder utilizar o programa da forma desejada.

A função *pressione()*, na figura 24 na pagina seguinte, é mencionada no fim das funções de interação mais extensas, como a *inserirJ()*. Esta função permite que se saia de dentro daquela função com a interação do utilizador, permitindo que haja o tempo que este achar necessário para avaliar os dados que lhe estão a ser apresentados, antes de retornar ao menu principal.

```

void pressione()
{
    printf("\n\nPrima qualquer tecla para ir para o menu!");
    getch();
    system("cls");
}

```

Figura 24 - Função *pressione()*.

A função *menu()* apresenta todas as opções do menu principal, estando representada na figura 25 abaixo.

```

int menu()
{
    int opcaoMenu;

    printf("-----* Menu principal *-----\n\nEscolha uma opção:\n\n1-Listar os jobs\n2-Inserir job\n3-Remover job\n4-Alterar job\n0-Sair\n\nOpção: ");
    scanf("%d", &opcaoMenu);

    return opcaoMenu;
}

```

Figura 25 - Função *menu()*.

A função *alteracaoO()* apresenta todas as opções referentes à alteração de operações, como mostrado na figura 26 abaixo:

```

int alteracaoO(job*lista,int job, int op)
{
    system("cls");
    listarJob(lista);
    int opcaoAlterar;

    printf("\n\n0 que pretende alterar dentro da operação %d do job %d?\n\n1-Adicionar máquina\n2-Remover máquina\n3-Alterar máquina\n0-Menu\n\nOpção: ", op, job);
    scanf("%d", &opcaoAlterar);

    return opcaoAlterar;
}

```

Figura 26 - Função *alteracaoO()*.

A função *alteracaoJ()* demonstra todas as opções quanto à alteração dos jobs. Podemos ver a função na figura 27 abaixo:

```

int alteracaoJ(job*lista,int nmrJob)
{
    system("cls");
    int opcaoAlterar;
    listarJob(lista);

    printf("\n\n0 que pretende alterar dentro do job %d?\n\n1-Adicionar operação\n2-Remover Operação\n3-Alterar Operação\n0-Menu\n\nOpção: ", nmrJob);
    scanf("%d", &opcaoAlterar);

    return opcaoAlterar;
}

```

Figura 27 - Função *alteracaoJ()*.

Por último, mas definitivamente não menos importante, temos a função *main()*. Esta função é logo a primeira a ser executada pelo compilador ao iniciar o programa e é ela o elo de ligação entre os menus e as funções de interação, que permite que haja a dinâmica do programa. Esta função está na figura na página seguinte.


```

void main()
{
    setlocale(LC_ALL, "Portuguese");

    jobs lista = lerFicheiro();
    printf("PROCESSO DE ESCALONAMENTO E PROCESS PLAN EM LINGUAGEM C\nTrabalho realizado por: \nDiana A.C. Dinis, número 23552\n\n");

    while (true)
    {
        int opcaoMenu = menu();

        switch (opcaoMenu)
        {
            case 1: //listar
                system("cls");
                printf("-----> Listar os jobs -----> \n\n");
                //lista = ordemCrescente(lista);
                listarJob(lista);
                pressione();
                break;

            case 2: //insserir
                system("cls");
                printf("-----> Inserir um job -----> \n\n");
                listarJob(lista);
                printf("\n\n");
                lista = inserirJ(lista);
                break;

            case 3: //remover
                system("cls");
                printf("-----> Remover um job -----> \n\n");
                listarJob(lista);
                printf("\n\n");
                lista = removerJ(lista);
                break;

            case 4: //alterar
                system("cls");
                printf("-----> Alterar um job -----> \n\n");
                listarJob(lista);
                printf("\n\n");

                int s = 0, i = 0;
                int qJob = qualJob();

                while (s == 0)
                {
                    int verJ = verificarJobExistente(lista, qJob);

                    if (verJ == 0)
                    {
                        printf("\n\nJob não existe. Escolha um job diferente para alterar\n\nOpção: ");
                        scanf("%d", &qJob);
                    }
                    else
                    {
                        int opcaoAlterarJ = alteracaoJ(lista, qJob);

                        switch (opcaoAlterarJ)
                        {
                            case 1:
                                s = 1;
                                system("cls");
                                printf("-----> Inserir uma operação em Jhd -----> \n\n", qJob);
                                listarJob(lista);
                                printf("\n\n");
                                lista = inserirO(lista, qJob);
                                break;

                            case 2:
                                s = 1;
                                system("cls");
                                printf("-----> Remover uma operação em Jhd -----> \n\n", qJob);
                                listarJob(lista);
                                printf("\n\n");
                                lista = removerO(lista, qJob);
                                break;

                            case 3: //alterar operação
                                s = 1;

                                int qOp = qualOp(lista, qJob);

                                while (i == 0)
                                {
                                    int verO = verificarOpExistente(lista, qJob, qOp);

                                    if (verO == 0)
                                    {
                                        printf("\n\nOperação não existe dentro do job hd. Escolha uma operação diferente para alterar\n\nOpção: ", qJob);
                                        scanf("%d", &qOp);
                                    }
                                    else
                                    {
                                        int opcaoAlterarO = alteracaoO(lista, qJob, qOp);

                                        switch (opcaoAlterarO)
                                        {
                                            case 1:
                                                i = 1;
                                                system("cls");
                                                printf("-----> Inserir uma máquina em Ohd do Jhd -----> \n\n", qOp, qJob);
                                                listarJob(lista);
                                                printf("\n\n");
                                                lista = inserirM(lista, qJob, qOp);
                                                break;

                                            case 2:
                                                i = 1;
                                                system("cls");
                                                printf("-----> Inserir uma máquina em Ohd do Jhd -----> \n\n", qOp, qJob);
                                                listarJob(lista);
                                                printf("\n\n");
                                                lista = removerM(lista, qJob, qOp);
                                                break;

                                            case 3:
                                                s = 1;
                                                system("cls");
                                                printf("-----> Alterar uma máquina em Ohd do Jhd -----> \n\n", qOp, qJob);
                                                listarJob(lista);
                                                printf("\n\n");
                                                lista = alterarM(lista, qJob, qOp);
                                                break;

                                            case 4:
                                                i = 1;
                                                s = 0;
                                                break;

                                            default:
                                                i = 0;
                                                system("cls");
                                                printf("Escolha inválida. Tente de novo.\n\n");
                                                continue;
                                            }
                                        }
                                    }
                                }

                                break;

                            default:
                                s = 0;
                                system("cls");
                                printf("Escolha inválida. Tente de novo.\n\n");
                                continue;
                            }
                        }
                    }
                }

                break;

            case 5: //exit
                system("cls");
                escreveFicheiro(lista);
                printf("Obrigado! Prima qualquer tecla para fechar o programa.\n\n\n");
                exit(0);
                break;

            default:
                system("cls");
                printf("Opção inválida. Por favor, tente de novo.\n\n");
                menu();
            }
        }
    }
}

```

Figura 28 - Função main().

Testes realizados

Neste capítulo será descrito os testes ao código desenvolvido ao longo do trabalho.

As primeiras funções a serem testadas foram as de escrever e ler do ficheiro. De modo a vê-las funcionar, é necessário adicionar jobs ao ficheiro através da função de inserir, que já foi desenvolvida e testada na primeira fase, com sucesso. Recordar-se que na primeira fase, a função de ler do ficheiro tinha um problema onde o programa fechava, caso se tentasse listar as funções na *terminal window*. Para se testar, primeiramente inseriu-se um job temporário no ficheiro. Após, fechou-se o programa e logo após ao iniciá-lo de novo, pediu-se para que o programa lista-se a lista ligada associada. Esta função foi feita com grande sucesso, como se pode observar na figura 29 a), onde se vê os conteúdos listados no programa iniciado e na figura 29 b), que mostra os conteúdos do ficheiro “processplan.txt”.

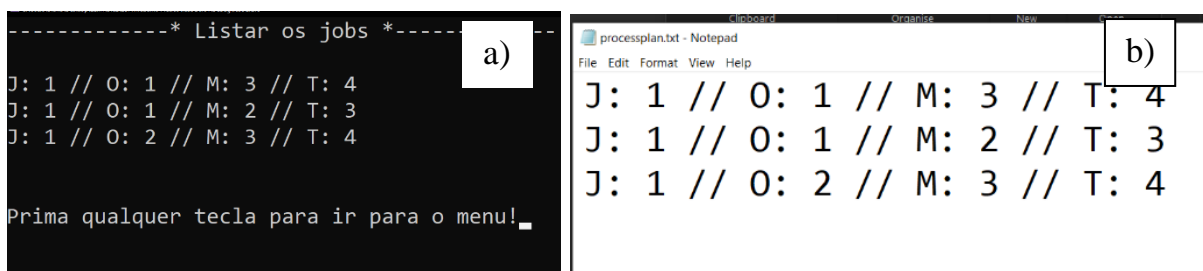


Figura 29 - Teste à função *lerFicheiro()*.

Uma nova função a ser inserida seria uma que iria verificar a existência de um job antes de o poder alterar, por exemplo, para adicionar ou remover alguma operação. Essa função está demonstrada na figura 30.

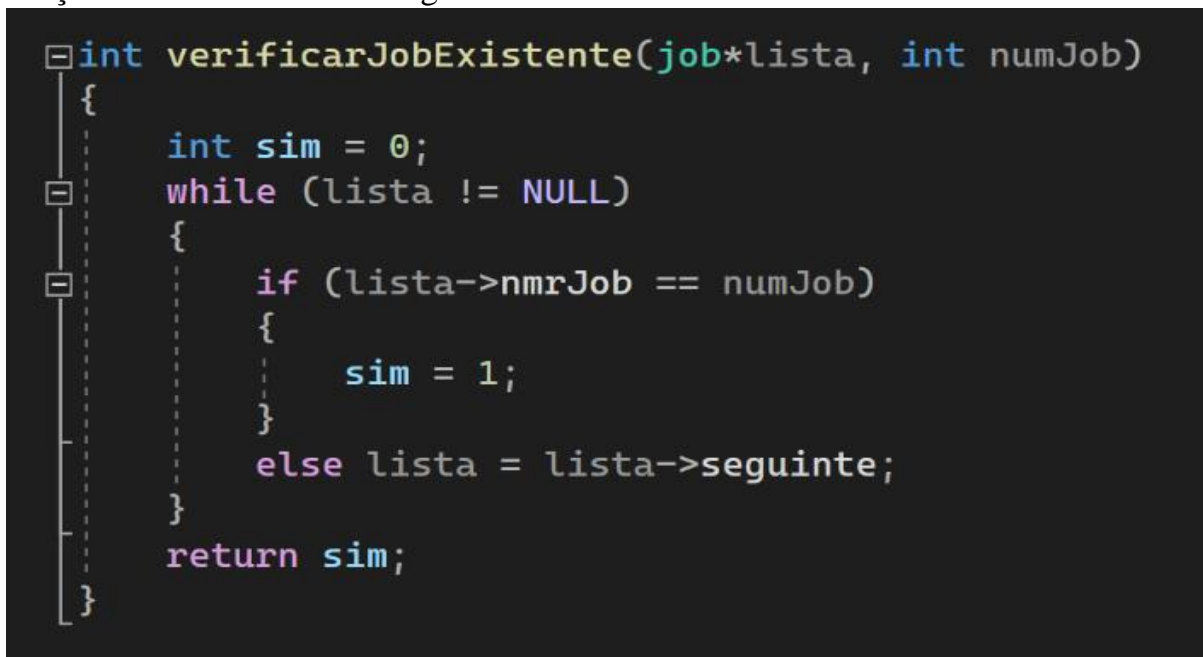


Figura 30 - Função para verificar existência de job

Esta função foi chamada no menu das alterações dos jobs, como se pode ver na figura 31 a seguir.

```
int qJob = qualJob();
int ver = verificarJobExistente(lista, qJob);

if (ver == 0)
{
    while (ver != 1)
    {
        printf("\nO job não existe. Escolha um job diferente para alterar\n\nOpção: ");
        scanf("%d", &qJob);
    }
}

else
{
    int opcaoAlterar = alterarJob(qJob);
}
```

Figura 31 - Chamar a função *verificarJobExistente()*.

Esta função vinha a ser chamada antes do menu geral das alterações, de modo a verificar a existência do job a ser alterado de modo a que o programa pudesse correr e realizar o esperado. No entanto, após a seleção do número do job a ser alterado, o programa encontra-se preso dentro de um loop, como se pode ver na *terminal window* presente na figura 32, onde primeiramente foi inserido um número de job que não existe e posteriormente, um que existe nessa lista ligada que se encontra listada em cima, sendo uma de testes enquanto o *process plan* do enunciado ainda não foi inserido na sua totalidade.

```
C:\Users\Diana\Desktop\LEIM\1s2\EDA\Trabalho Prático\fase 2\164\Debug\fase 2.exe
-----* Alterar um job *-----

J: 4 // O: 1 // M: 2 // T: 2
J: 3 // O: 1 // M: 1 // T: 2
J: 2 // O: 2 // M: 3 // T: 3
J: 2 // O: 1 // M: 4 // T: 2
J: 2 // O: 1 // M: 3 // T: 2
J: 1 // O: 1 // M: 2 // T: 3
J: 1 // O: 1 // M: 3 // T: 4
J: 1 // O: 2 // M: 3 // T: 4

Qual é o job que pretende alterar?
Opcao: 6

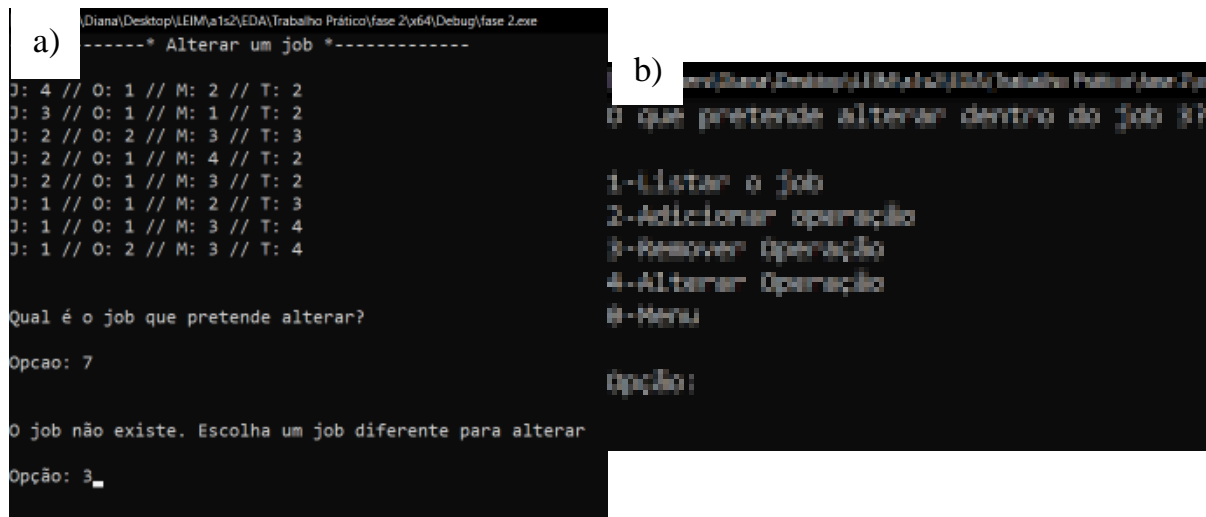
O job não existe. Escolha um job diferente para alterar

Opção: 2
Qual é o job que pretende alterar?
Opcao: 2
```

Figura 32 - *Terminal window* do teste da função *verificarJobExistente()*.

Como se pode verificar na figura 32 acima, há também uma repetição na pergunta do job a alterar. Ao chamar a função para ir buscar o número do job, *qualJob()*, fora do ciclo *while* do menu esse problema resolve-se. No entanto, o problema de ficar preso no ciclo não foi resolvido nesse instante.

De modo o programa a funcionar, verificou-se que o *bug* se encontrava na condição de paragem do ciclo **while** dentro da função de verificação. Este na função **verificarJobExistente()** foi atualizado para seguir a lista até ao fim e verificar se a variável **sim** é igual a 1, visto que se essa condição for verdadeira, existe aquele job na lista ligada. Isto traduz-se na condição **lista != NULL && sim != 1**. Podemos ver o código a funcionar na figura 33 abaixo, onde na figura 33 a) teremos a verificar a existência do job, testando um número que não existe e a seguir na figura 33 b) a confirmação através da função **alteracaoJ()**, que recebe por parâmetro o número do job a editar, que esta função de verificar concluiu-se.



```

a)
-----* Alterar um job *-----
J: 4 // O: 1 // M: 2 // T: 2
J: 3 // O: 1 // M: 1 // T: 2
J: 2 // O: 2 // M: 3 // T: 3
J: 2 // O: 1 // M: 4 // T: 2
J: 2 // O: 1 // M: 3 // T: 2
J: 1 // O: 1 // M: 2 // T: 3
J: 1 // O: 1 // M: 3 // T: 4
J: 1 // O: 2 // M: 3 // T: 4

Qual é o job que pretende alterar?

Opcao: 7

O job não existe. Escolha um job diferente para alterar

Opção: 3_

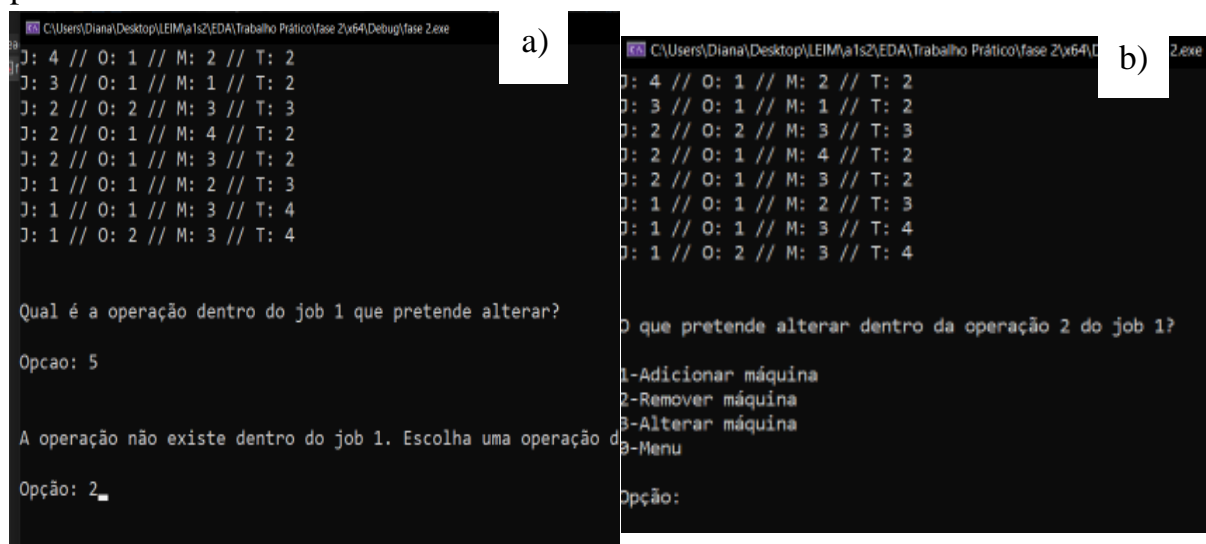
b)
-----* Alterar um job *-----
O que pretende alterar dentro do job 3?

1-Listar o job
2-Adicionar operação
3-Remover operação
4-Alterar operação
0-Menu

Opção:
  
```

Figura 33 - Função **verificarJobExistente()** a funcionar

A próxima função a ser testada seria a função **verificarOpExistente()**, que é uma variante da função anterior que irá aferir se existe ou não dentro daquele job em específico a operação que o utilizador inseriu para modificar. O teste está dividido na figura 34 abaixo, onde foi escolhido previamente o job 1 como o job a alterar, como se pode verificar.



```

a)
C:\Users\Diana\Desktop\LEIM\1s2\EDA\Trabalho Prático\fase 2\vx64\Debug\fase 2.exe
J: 4 // O: 1 // M: 2 // T: 2
J: 3 // O: 1 // M: 1 // T: 2
J: 2 // O: 2 // M: 3 // T: 3
J: 2 // O: 1 // M: 4 // T: 2
J: 2 // O: 1 // M: 3 // T: 2
J: 1 // O: 1 // M: 2 // T: 3
J: 1 // O: 1 // M: 3 // T: 4
J: 1 // O: 2 // M: 3 // T: 4

Qual é a operação dentro do job 1 que pretende alterar?

Opcao: 5

A operação não existe dentro do job 1. Escolha uma operação diferente

Opção: 2_

b)
C:\Users\Diana\Desktop\LEIM\1s2\EDA\Trabalho Prático\fase 2\vx64\Debug\fase 2.exe
J: 4 // O: 1 // M: 2 // T: 2
J: 3 // O: 1 // M: 1 // T: 2
J: 2 // O: 2 // M: 3 // T: 3
J: 2 // O: 1 // M: 4 // T: 2
J: 2 // O: 1 // M: 3 // T: 2
J: 1 // O: 1 // M: 2 // T: 3
J: 1 // O: 1 // M: 3 // T: 4
J: 1 // O: 2 // M: 3 // T: 4

O que pretende alterar dentro da operação 2 do job 1?

1-Adicionar máquina
2-Remover máquina
3-Alterar máquina
0-Menu

Opção:
  
```

Figura 34 - Função **verificarOpExistente()** a funcionar

A função a ser testada a seguir seria a função de verificar se uma máquina já estaria incluída no job. A primeira tentativa seria incluir a função quando estaria a incluir as máquinas na função de interação `inserirJ()` como se pode ver na figura 35.

```
do
{
    printf("\nMáquina: ");
    scanf("%d", &novaMáquina);

    int verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMáquina);

    while (novaMáquina > MAXMACHINE)
    {
        printf("\nA máquina %d não existe. \nInsira outra máquina: ", novaMáquina);
        scanf("%d", &novaMáquina);
    }

    while (verMq == 1)
    {
        printf("\nA máquina %d já se encontra associada à operação %d. \nInsira outra máquina: ", novaMáquina, novaOperacao);
        scanf("%d", &novaMáquina);
        int verMq = verificarMqExistente(inicio, novoJob, novaOperacao, novaMáquina);
    }
}
```

Figura 35 - Primeira proposta para utilizar a função `verificarMqExistente()`.

No entanto, ao correr o código, fica preso no ciclo `while(verMq ==1)`, como se pode verificar na figura 36 abaixo, referente à *terminal window*.

```
Job: 5
Operação: 1
Máquina: 3
Tempo de processamento: 1

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não

Opção: 1
Máquina: 3

A máquina 3 já se encontra associada à operação 1.
Insira outra máquina: 2

A máquina 2 já se encontra associada à operação 1.
Insira outra máquina: █
```

Figura 36 - Terminal window do primeiro teste da função `verMqExistente()`.

O bug foi encontrado na linha 33 do código. Retirando o `int` atrás do nome da variável, o programa já correu de forma a conferir corretamente a existência ou não da máquina na operação do job. Podemos ver o programa a correr na *terminal window* corretamente na figura 37 abaixo:

```

Job: 5
Operação: 1
Máquina: 2
Tempo de processamento: 1

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 1

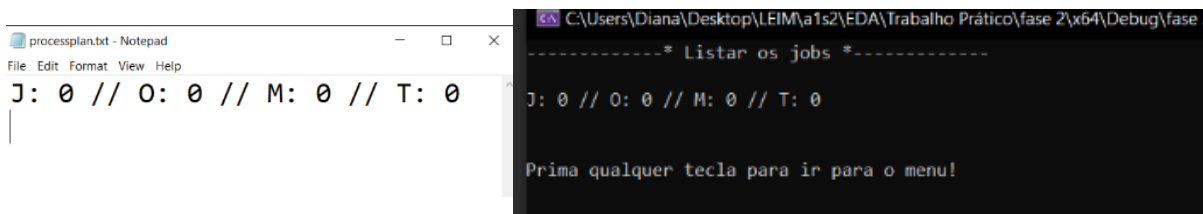
Máquina: 2

A máquina 2 já se encontra associada à operação 1.
Insira outra máquina: 4
Tempo de processamento: 1

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção:
  
```

Figura 37 - Função *verMaqExistente()* a funcionar.

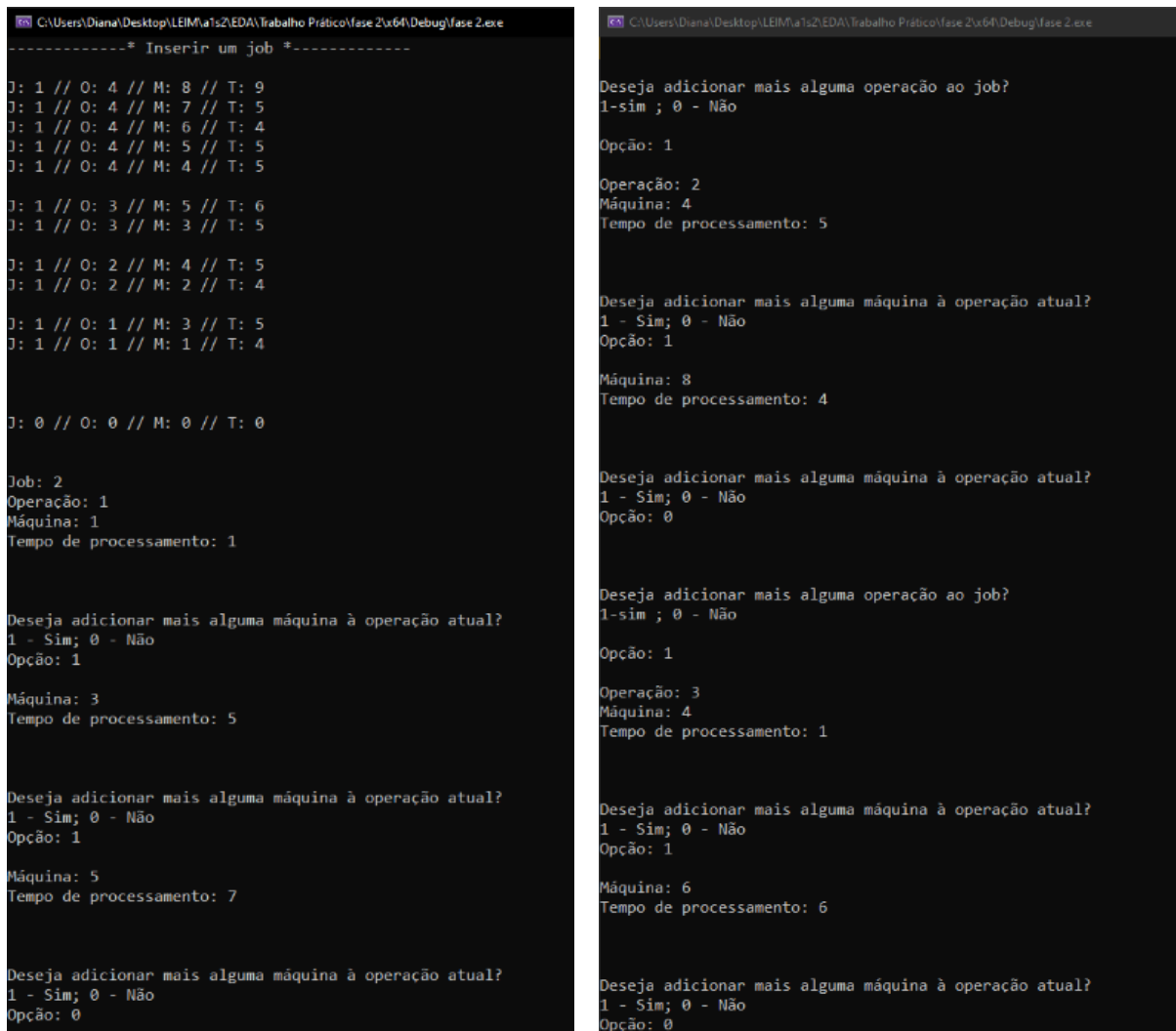
Apenas nesta altura que se foi adicionar o *process plan* proposto. Para tal, manualmente alterou-se no ficheiro de modo a apenas haver um job número 0, de modo a correr a função de inserir e esta começar em 1, visto que esta ao iniciar, vai somar mais 1 ao maior valor de job. Na figura 38 a) podemos observar o ficheiro *processplan.txt* contendo apenas o job 0 e na figura 38 b) a listagem desse mesmo job.



The figure consists of two side-by-side screenshots. The left screenshot shows a Notepad window titled 'processplan.txt - Notepad' with the following text: 'J: 0 // O: 0 // M: 0 // T: 0'. The right screenshot shows a command prompt window with the title 'C:\Users\Diana\Desktop\LEIM\152\EDA\Trabalho Prático\fase 2\x64\Debug\fase 2'. It displays a menu titled '* Listar os jobs *' with the same text 'J: 0 // O: 0 // M: 0 // T: 0' and a prompt 'Prima qualquer tecla para ir para o menu!'.

Figura 38 – Listagem do job 0.

Foi inserido o job 1 primeiro. Na figura 39 abaixo está representado um pouco do processo de inserção do job 2.



```

C:\Users\Diana\Desktop\LEIM\1a2\EDA\Trabalho Prático\Fase 2\64\Debug\Fase 2.exe
-----* Inserir um job *-----
J: 1 // O: 4 // M: 8 // T: 9
J: 1 // O: 4 // M: 7 // T: 5
J: 1 // O: 4 // M: 6 // T: 4
J: 1 // O: 4 // M: 5 // T: 5
J: 1 // O: 4 // M: 4 // T: 5

J: 1 // O: 3 // M: 5 // T: 6
J: 1 // O: 3 // M: 3 // T: 5

J: 1 // O: 2 // M: 4 // T: 5
J: 1 // O: 2 // M: 2 // T: 4

J: 1 // O: 1 // M: 3 // T: 5
J: 1 // O: 1 // M: 1 // T: 4

J: 0 // O: 0 // M: 0 // T: 0

Job: 2
Operação: 1
Máquina: 1
Tempo de processamento: 1

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 1

Máquina: 3
Tempo de processamento: 5

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 1

Máquina: 5
Tempo de processamento: 7

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 0

C:\Users\Diana\Desktop\LEIM\1a2\EDA\Trabalho Prático\Fase 2\64\Debug\Fase 2.exe
Deseja adicionar mais alguma operação ao job?
1-sim ; 0 - Não

Opção: 1

Operação: 2
Máquina: 4
Tempo de processamento: 5

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 1

Máquina: 8
Tempo de processamento: 4

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 0

Deseja adicionar mais alguma operação ao job?
1-sim ; 0 - Não

Opção: 1

Operação: 3
Máquina: 4
Tempo de processamento: 1

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 1

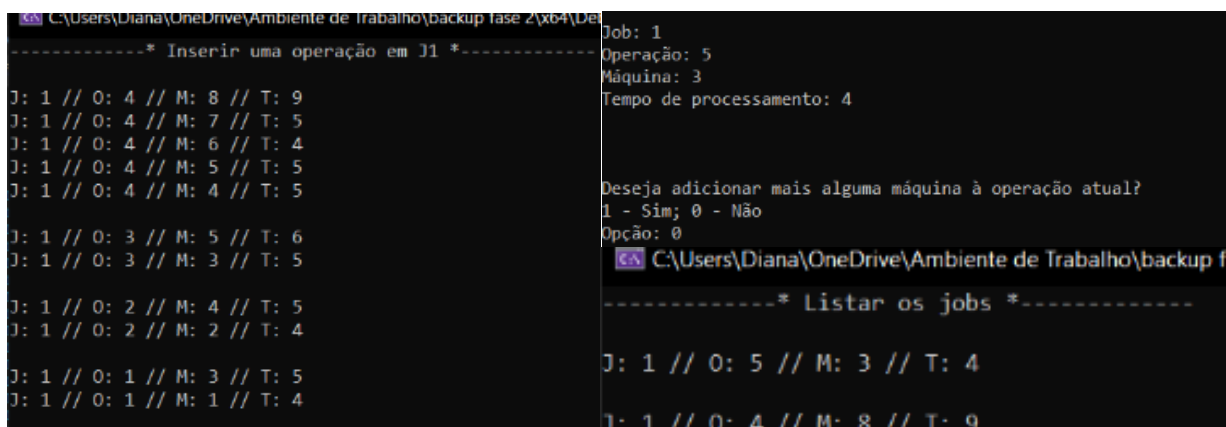
Máquina: 6
Tempo de processamento: 6

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 0

```

Figura 39 – Exemplo de inserção do process plan proposto

Foi a função `inserirO()`, de interação, a função seguinte na linha de testagem. Esta função contém código pertencente à função `inserirJ()`. Podemos observar o executamento desta função na figura 40 abaixo:



```

C:\Users\Diana\OneDrive\Ambiente de Trabalho\backup fase 2\64\De
-----* Inserir uma operação em J1 *-----
J: 1 // O: 4 // M: 8 // T: 9
J: 1 // O: 4 // M: 7 // T: 5
J: 1 // O: 4 // M: 6 // T: 4
J: 1 // O: 4 // M: 5 // T: 5
J: 1 // O: 4 // M: 4 // T: 5

J: 1 // O: 3 // M: 5 // T: 6
J: 1 // O: 3 // M: 3 // T: 5

J: 1 // O: 2 // M: 4 // T: 5
J: 1 // O: 2 // M: 2 // T: 4

J: 1 // O: 1 // M: 3 // T: 5
J: 1 // O: 1 // M: 1 // T: 4

Job: 1
Operação: 5
Máquina: 3
Tempo de processamento: 4

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 0

C:\Users\Diana\OneDrive\Ambiente de Trabalho\backup f
-----* Listar os jobs *-----

J: 1 // O: 5 // M: 3 // T: 4

J: 1 // O: 4 // M: 8 // T: 9

```

Figura 40 - Função `inserirO()` a funcionar

A próxima será a `inserirM()`. Esta função, igualmente à anterior, também possui código das últimas duas funções de interação referentes à inserção de novos dados na lista ligada. O funcionamento desta função encontra-se na figura 41 abaixo.

```
J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

Job: 8
Operação: 4
Máquina: 2
Tempo de processamento: 5

Deseja adicionar mais alguma máquina à operação atual?
1 - Sim; 0 - Não
Opção: 0

Prima qualquer tecla para ir para o menu!
```

```
C:\Users\Diana\OneDrive\Ambiente de Trabalho>
-----* Listar os jobs *-----

J: 8 // O: 4 // M: 3 // T: 5
```

Figura 41 - Função `inserirM()` a funcionar.

A próxima função a ser testada será a `removerJob()`. Primeiramente foi testada a função desenvolvida para a outra fase, com apenas algumas alterações para considerar o novo campo do número do job da lista ligada. No entanto, apesar de não apresentar erro no compilador, ao correr o programa este fechava.

Após analisar o código através do debug, apercebi-me que as condições de acesso ao ciclo `while` que permitia ler a lista completa, não era cumprido, de alguma forma, após a eliminação de um bloco da lista ligada.

Esta função foi alterada, portanto, de forma a embarcar também uma função que permitirá reduzir o número dos jobs seguintes ao job eliminado e o funcionamento desta está presente na figura 42 abaixo, onde a figura 42 a) apresenta a lista ligada antes de remover o job 1, a figura 42 b) o processo de requerimento de qual job pretendemos remover e, por fim, o resultado final na figura 42 c).

a)

```
-----* Listar os jobs *-----

J: 1 // O: 4 // M: 8 // T: 9
J: 1 // O: 4 // M: 7 // T: 5
J: 1 // O: 4 // M: 6 // T: 4
J: 1 // O: 4 // M: 5 // T: 5
J: 1 // O: 4 // M: 4 // T: 5

J: 1 // O: 3 // M: 5 // T: 6
J: 1 // O: 3 // M: 3 // T: 5

J: 1 // O: 2 // M: 4 // T: 5
J: 1 // O: 2 // M: 2 // T: 4

J: 1 // O: 1 // M: 3 // T: 5
J: 1 // O: 1 // M: 1 // T: 4

J: 2 // O: 1 // M: 1 // T: 1
J: 2 // O: 1 // M: 3 // T: 5
J: 2 // O: 1 // M: 5 // T: 7

J: 2 // O: 2 // M: 4 // T: 5
J: 2 // O: 2 // M: 8 // T: 4

J: 2 // O: 3 // M: 4 // T: 1
J: 2 // O: 3 // M: 6 // T: 6

J: 2 // O: 4 // M: 4 // T: 4
J: 2 // O: 4 // M: 7 // T: 4
```

b)

```
C:\Users\Diana\OneDrive\Ambiente de Trabalho>

Job a remover: 1

Remover mais algum job?
1 - Sim; 0 - Não
Opção: 0

Prima qualquer tecla para ir para o menu!
```

c)

```
-----* Listar os jobs *-----

J: 2 // O: 1 // M: 1 // T: 1
J: 1 // O: 1 // M: 3 // T: 5
J: 1 // O: 1 // M: 5 // T: 7
J: 1 // O: 2 // M: 4 // T: 5
J: 1 // O: 2 // M: 8 // T: 4
J: 1 // O: 3 // M: 4 // T: 1
J: 1 // O: 3 // M: 6 // T: 6
J: 1 // O: 4 // M: 4 // T: 4
J: 1 // O: 4 // M: 7 // T: 4
J: 1 // O: 4 // M: 8 // T: 7
J: 1 // O: 5 // M: 4 // T: 1
J: 1 // O: 5 // M: 6 // T: 2
J: 1 // O: 6 // M: 1 // T: 5
J: 1 // O: 6 // M: 6 // T: 6
J: 1 // O: 6 // M: 8 // T: 4
J: 1 // O: 7 // M: 4 // T: 4
J: 2 // O: 5 // M: 2 // T: 4
```

Figura 42 - Função `reduzirJob()` a funcionar.

Como se pode observar, comparando a figura 42 a) e a figura 42 c), a primeira ocorrência do job a ter o seu número reduzido, não funcionou. No entanto, todos os outros foram corrigidos para o valor anterior.

Ao fazer o debug, foi se apercebido que a expressão `jobAnterior = jobAtual` na linha 61 faz com que o programa salte a primeira ocorrência. Na figura 43 a), podemos observar os valores de `jobAtual` e `jobAnterior` antes de passar pela condição e na figura 43 b), os valores das mesmas variáveis depois de passar pela condição.

a)

```

55 }
56 else
57 { //Remove as seguintes ocorrências
58   jobAnterior = lista;
59   jobAtual = jobAnterior->seguinte;
60   while ((jobAtual != NULL) && (jobAtual->nmrJob != numJob))
61   {
62     jobAnterior = jobAtual;

```

Locals

Name	Value	Type
jobAnterior	0x000002955a529150 (nmrJob=2 nmrOperacao=1 maquina=1 ...)	Job *
jobAtual	0x000002955a529870 (nmrJob=2 nmrOperacao=1 maquina=3 ...)	Job *
lista	0x000002955a529150 (nmrJob=2 nmrOperacao=1 maquina=1 ...)	Job *
numJob	1	int

b)

```

55 }
56 else
57 { //Remove as seguintes ocorrências
58   jobAnterior = lista;
59   jobAtual = jobAnterior->seguinte;
60   while ((jobAtual != NULL) && (jobAtual->nmrJob != numJob))
61   {
62     jobAnterior = jobAtual;
63     if (jobAnterior->nmrJob > numJob)
64     {
65       jobAnterior->nmrJob = jobAnterior->nmrJob - 1;
66     }
67   }

```

Locals

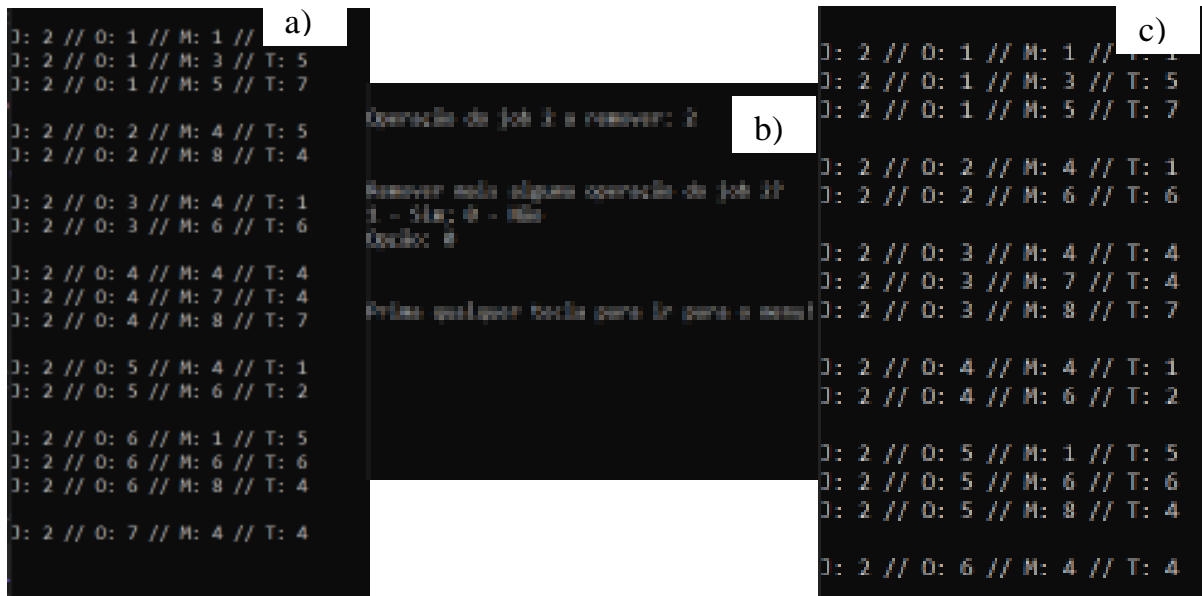
Name	Value	Type
jobAnterior	0x000002955a529870 (nmrJob=2 nmrOperacao=1 maquina=3 ...)	Job *
jobAtual	0x000002955a529870 (nmrJob=2 nmrOperacao=1 maquina=3 ...)	Job *
lista	0x000002955a529150 (nmrJob=2 nmrOperacao=1 maquina=1 ...)	Job *
numJob	1	int

Figura 43 - Debug da função `removerJob()`.

Ao saltar o bloco correspondente à máquina 1 na operação 1 do job 2, este não vai ser reduzido.

Ao remover a linha 57 da função, o código tornou-se operacional.

O próximo teste foi à função `removerOperacao()`. Na figura 44 a) vemos o job 2 listado, ao qual vamos remover a operação 2, como mostra a figura 44 b), onde o utilizador insere o valor da operação que quer remover. Na figura 44 c), temos o job 2 com a operação 2 removida e as restantes movidas para cima.



a)

```
J: 2 // O: 1 // M: 1 // T: 5
J: 2 // O: 1 // M: 3 // T: 5
J: 2 // O: 1 // M: 5 // T: 7

J: 2 // O: 2 // M: 4 // T: 5
J: 2 // O: 2 // M: 8 // T: 4

J: 2 // O: 3 // M: 4 // T: 1
J: 2 // O: 3 // M: 6 // T: 6

J: 2 // O: 4 // M: 4 // T: 4
J: 2 // O: 4 // M: 7 // T: 4
J: 2 // O: 4 // M: 8 // T: 7

J: 2 // O: 5 // M: 4 // T: 1
J: 2 // O: 5 // M: 6 // T: 2

J: 2 // O: 6 // M: 1 // T: 5
J: 2 // O: 6 // M: 6 // T: 6
J: 2 // O: 6 // M: 8 // T: 4

J: 2 // O: 7 // M: 4 // T: 4
```

b)

```
Operação da job 2 a remover: 2

Remover toda alguma operação da job 2?
1 - Sim; 0 - Não
Opção: 0
```

c)

```
J: 2 // O: 1 // M: 1 // T: 5
J: 2 // O: 1 // M: 3 // T: 5
J: 2 // O: 1 // M: 5 // T: 7

J: 2 // O: 2 // M: 4 // T: 1
J: 2 // O: 2 // M: 6 // T: 6

J: 2 // O: 3 // M: 4 // T: 4
J: 2 // O: 3 // M: 7 // T: 4
J: 2 // O: 3 // M: 8 // T: 7

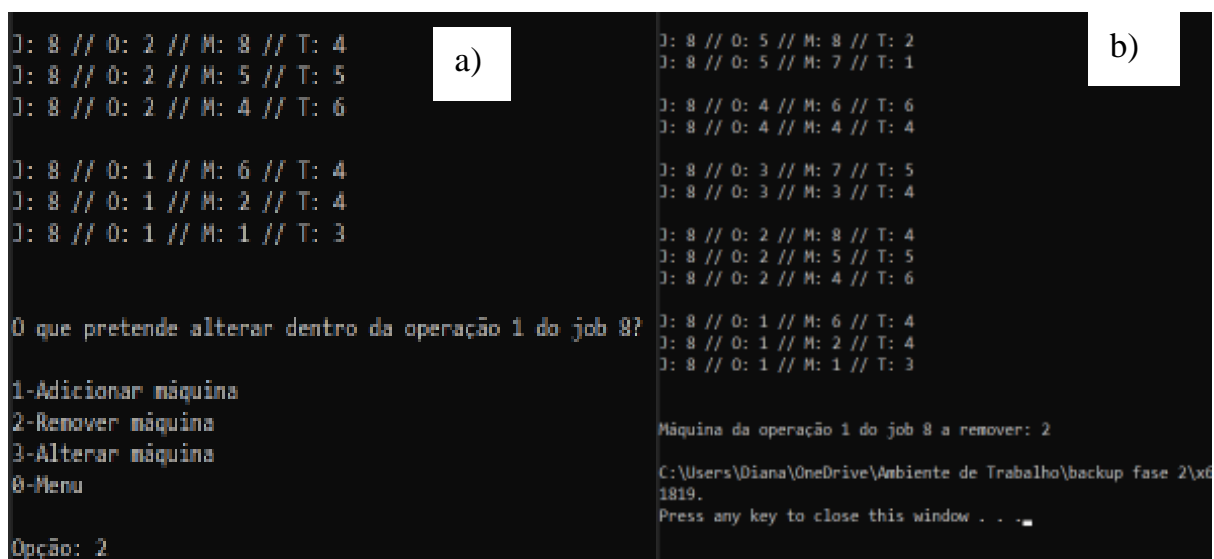
J: 2 // O: 4 // M: 4 // T: 1
J: 2 // O: 4 // M: 6 // T: 2

J: 2 // O: 5 // M: 1 // T: 5
J: 2 // O: 5 // M: 6 // T: 6
J: 2 // O: 5 // M: 8 // T: 4

J: 2 // O: 6 // M: 4 // T: 4
```

Figura 44 - Função `removerOperacao()` a funcionar.

A função `removerMaquina()` foi a próxima. Na figura 45 a) temos o job 8 listado, onde iremos remover a máquina 2 da operação 1. No entanto, na primeira tentativa, o programa parou de correr, como podemos observar na figura 45 b).



a)

```
J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

O que pretende alterar dentro da operação 1 do job 8?
1-Adicionar máquina
2-Remover máquina
3-Alterar máquina
0-Menu
Opção: 2
```

b)

```
J: 8 // O: 5 // M: 8 // T: 2
J: 8 // O: 5 // M: 7 // T: 1

J: 8 // O: 4 // M: 6 // T: 6
J: 8 // O: 4 // M: 4 // T: 4

J: 8 // O: 3 // M: 7 // T: 5
J: 8 // O: 3 // M: 3 // T: 4

J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

Máquina da operação 1 do job 8 a remover: 2
C:\Users\Diana\OneDrive\Ambiente de Trabalho\backup fase 2\xt
1819.
Press any key to close this window . . .
```

Figura 45 - Função `removerMaquina()` a apresentar erro.

Adicionando apenas uma linha de código a inicializar `jobAtual = inicio` no início do ciclo `while`, a função deixou de interromper o programa, como se pode ver na figura 46 a). Já na figura 46 b), comprova-se que foi removida a máquina daquela operação com sucesso, ao comparar com a figura 45 a).

a)

```

J: 8 // O: 5 // M: 8 // T: 2
J: 8 // O: 5 // M: 7 // T: 1

J: 8 // O: 4 // M: 6 // T: 6
J: 8 // O: 4 // M: 4 // T: 4

J: 8 // O: 3 // M: 7 // T: 5
J: 8 // O: 3 // M: 3 // T: 4

J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

Máquina da operação 1 do job 8 a remover: 2

Remover mais alguma máquina da operação 1 do job 8?
1 - Sim; 0 - Não
Opção: 0

Prima qualquer tecla para ir para o menu!
          
```

b)

```

J: 8 // O: 3 // M: 7 // T: 5
J: 8 // O: 3 // M: 3 // T: 4

J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

Prima qualquer tecla para ir para o menu!
          
```

Figura 46 - Função `removerMaquina()` a funcionar.

Por último, a última função a ser testada será a de `alterarMaquina()`. Vai ser alterada no job 8 a máquina 6 da operação 1, que podemos ver na figura 46 b), alterando o seu tempo de processamento de 4 para 2, como se pode ver na figura 47 a). Já na figura 47 b), podemos ver a máquina já alterada, realçada a amarelo.

a)

```

J: 8 // O: 4 // M: 6 // T: 6
J: 8 // O: 4 // M: 4 // T: 4

J: 8 // O: 3 // M: 7 // T: 5
J: 8 // O: 3 // M: 3 // T: 4

J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 6 // T: 4
J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3

Máquina da operação 1 do job 8 a alterar: 6
Tempo de processamento novo: 2

Alterar mais alguma máquina da operação 1 do job 8?
1 - Sim; 0 - Não
Opção: 0
          
```

b)

```

J: 8 // O: 5 // M: 8 // T: 2
J: 8 // O: 5 // M: 7 // T: 1

J: 8 // O: 4 // M: 6 // T: 6
J: 8 // O: 4 // M: 4 // T: 4

J: 8 // O: 3 // M: 7 // T: 5
J: 8 // O: 3 // M: 3 // T: 4

J: 8 // O: 2 // M: 8 // T: 4
J: 8 // O: 2 // M: 5 // T: 5
J: 8 // O: 2 // M: 4 // T: 6

J: 8 // O: 1 // M: 2 // T: 4
J: 8 // O: 1 // M: 1 // T: 3
J: 8 // O: 1 // M: 6 // T: 2

Prima qualquer tecla para ir para o menu!
          
```

Figura 47 - Função `alterarMaquina()` a funcionar.

Conclusão

A realização deste projeto veio se a revelar desafiante. Aspetos que foram mencionados como pontos a melhorar na primeira fase foram corrigidos, como a verificação da existência prévia de jobs ou operações que quiséssemos modificar, a inserção de forma sequencial de jobs e operações, permitindo haver uma sequência, que é requerido num *process plan* e a existência de um número máximo de máquinas disponíveis, entre outros.

Este trabalho não apresenta, no entanto, uma proposta de escalonamento do problema. Várias formas de o conseguir foram pensadas, porém nenhuma foi implementada.

O desenvolvimento deste trabalho prático permitiu o desenvolvimento de aspetos de programação em linguagem C que com certeza irão acompanhar toda a gente que seguir a profissão.

Bibliografia

Repositório GitHub: https://github.com/xalofal/fase_2

Anexos

menu - 1. listar jobs

2. inserir jobs

3. remover jobs

4. alterar jobs. - 2 adicionar operação

0. sair

3 remover operação

1 listar job

4 alterar operação -

0. menu

alterar 0

→ 1 adicionar máquina

2 remover máquina

3 alterar máquina. - 1. mudar o tempo

0. alterar j().

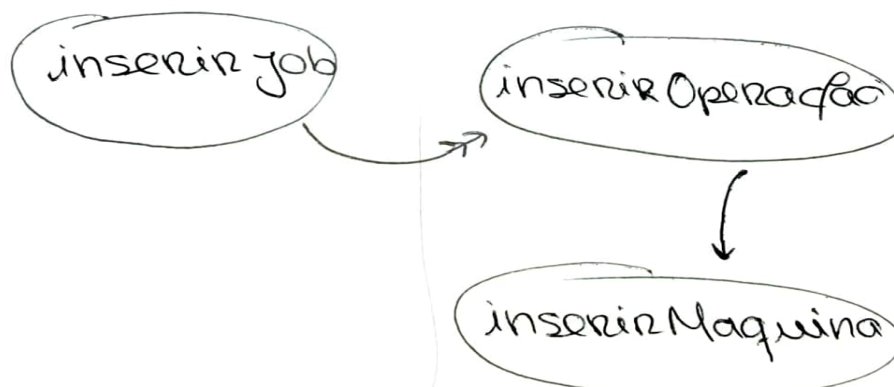
inserir jobs /

1. incluir num job existente as operações

2. incluir job novo

↓
 Vê qual foi o maior job e
 acrescenta (+1) ao valor.

mesmo processo para acrescentar as
 máquinas às operações.



remover job

remover o último: nada acontar

remover job do meio / início

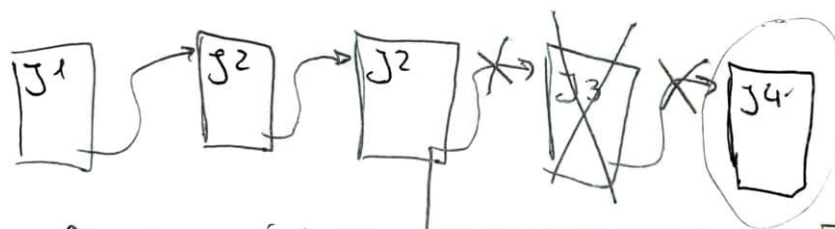


- percorrer a lista
- eliminar o job.
- alterar o número dos outros jobs para (-1) .



inserir tudo de novo?

- guardar numa lista Temp.



- elimina job 3

seguinte (j2) \rightarrow (j4)

Temp \rightarrow job = $\text{next job} - 1$

Para jobs não é preciso ~~alterar~~ o número, mas p/ não fazer uma função nova para inserir onde falta, mais vale.

É preciso para remover operações

3

Inserir Operações / máquinas

- Guardar qual job
- Guardar qual operação

① Adicionar no meio da lista
ou

① Adicionar no início

② Ordenar / juntar perto das operações/jobs
com o mesmo número

Utilizar função de inserir ~~para~~ jobs,
funções de interacção diferentes