Relatório Fase 2

Estrutura de Dados Avançadas

Alunos:
18450 – Rui Pinto

Professor: João Carlos Silva;

Licenciatura em Engenharia de Sistemas Informaticos

Barcelos, Junho, 2022

Índice

Índice das Imagens

Introdução

O trabalho desenvolvido no âmbito da unidade curricular “Estrutura de Dados Avançadas”, tem como objetivo desenvolvimento de uma solução digital para o problema de escalonamento denominado Flexible Job Shop Problem. O programa permite gerar uma proposta de escalonamento para a produção de um produto envolvendo várias operações e a utilização de várias máquinas, minimizando o tempo as unidades de tempo necessário na sua produção.

Este projeto tem ainda a finalidade de aplicar conteúdos lecionados anteriormente da linguagem C, aprofundando-os, e ainda aplicar novos conteúdos como algoritmos de procura e de ordenação, listas dinâmicas e apontadores.

Propósitos e objetivos

Tal como foi referido, o objetivo do programa é gerir a produção de um determinado produto que é tratado como um Job, com as várias operações necessárias e as máquinas que serão utilizadas.

Na fase 1 do projeto foi-nos proposto gerir a produção de apenas um produto, podendo adicionar, editar ou remover as operações e determinar o tempo máximo, mínimo e médio necessário para completar o Job, apresentando as respetivas operações.

O programa também terá de abrir o ficheiro de texto que terá guardado toda a informação necessária para depois caso seja preciso modificar alguma informação ou até mesmo adicionar operações. No fim a informação será guardada no mesmo ficheiro.

A fase 2 do projeto gere a produção de vários produtos, tendo como objetivo adicionar, editar ou remover os Jobs. No fim do programa, toda a nova informação será armazenada/atualizada no documento de texto dados.txt.

Estrutura de Dados

A estrutura do projeto é composta por 3 ficheiros (Main.c , Header.h , Functions.c) todos realizados com a linguagem C.

Main.c

Este é o programa principal porque é onde corre o programa.

Primeiramente é constituído pelas bibliotecas necessárias para o funcionamento do programa e os restantes ficheiros do projeto (“Header.h” e “functions.c”).

```c
/**
 * @file Main.c
 * @author Rui Pinto (a18450@alunos.ipca.pt)
 * @brief
 * @version 1.1
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "header.h"
#include "functions.c"
```

Figura 1 - Bibliotecas e ficheiros Main.c

Depois de incluir as bibliotecas e os restantes ficheiros começa a função main (a função que principal). No início da função contém todas as variáveis necessárias e a utilização de funções para criar uma lista para os jobs e as operações e abrir o ficheiro de texto para recolher os dados.

```c
int main() {

    job *jb = NULL, *procuraJob = NULL;
    int opcao, idContJob = 1, idCont = 0;

    system("cls");
    jb = novoNoJob(idContJob);
    verificaDadosNoFich(jb,&idContJob);


    job *listaJob;
    operacao *listaOperacao;

    listaJob = inicializarJob();
    verificarDadosNoFicheiro(listaJob->op,&idCont,&(listaJob->numOperacoes));

```

Figura 2 - Variáveis e inicio do main

Por fim irá aparecer o menu e depois da recolha da opção do utilizador irá efetuar a função pretendida. No fim do programa os dados serão todos atualizados.

```c
main.c > ...
    do {
        menu(&opcao);

        switch(opcao) {
            case 1: inserirNovaOperacao(listaJob->op, &idCont);
                    system("pause");
                    break;
            case 2: listarOperacao(listaJob->op, listaJob->numOperacoes);
                    system("pause");
                    break;
            case 3: procuraJob = descobreJob(jb);
                    removeOperacao(procuraJob->op,&(procuraJob->numOperacoes));
                    system("pause");
                    break;
            case 4: procuraJob = descobreJob(jb);
                    editaOperacao(procuraJob->op);
                    system("pause");
                    break;
            case 5: determinaTempoCurto(listaJob->op);
                    system("pause");
                    break;
            case 6: determinaTempoLongo(listaJob->op);
                    system("pause");
                    break;
            case 7: system("cls");
                    listarJobEmListaOrdenada(jb);
                    system("pause");
                    break;
            case 8: inserirJob(jb,&idContJob);
                    system("pause");
                    break;
            case 9: removeJob(jb);
                    system("pause");
                    break;
            case 10:procuraJob = descobreJob(jb);
                    insereNoOperacao(procuraJob->op,&(procuraJob->numOperacoes));
                    system("pause");
                    break;
            case 11:procuraJob = descobreJob(jb);
                    removeOperacao(procuraJob->op,&(procuraJob->numOperacoes));
                    system("pause");
                    break;
            case 12:procuraJob = descobreJob(jb);
                    editaOperacao(procuraJob->op);
                    system("pause");
                    break;
            case 13:printf("Ate a proxima!\n\n");
                    break;
        }
    }while(opcao != 0);

    guardaDadosFile(jb,"dados.txt");

    return 0;
}
```

Figura 3 - Opções do utilizador e fim do main

Header.h

O ficheiro Header.h é utilizado como biblioteca para guardar as structs e as assinaturas das funções utilizadas durante todo o projeto. Tal como no main.c, no início terá de se colocar todas as bibliotecas necessárias para a execução do projeto.

Figura 4 - Bibliotecas Header.h

De seguida, estão as struct utilizadas no projeto (uma para a representação dos jobs e outra para as operações).

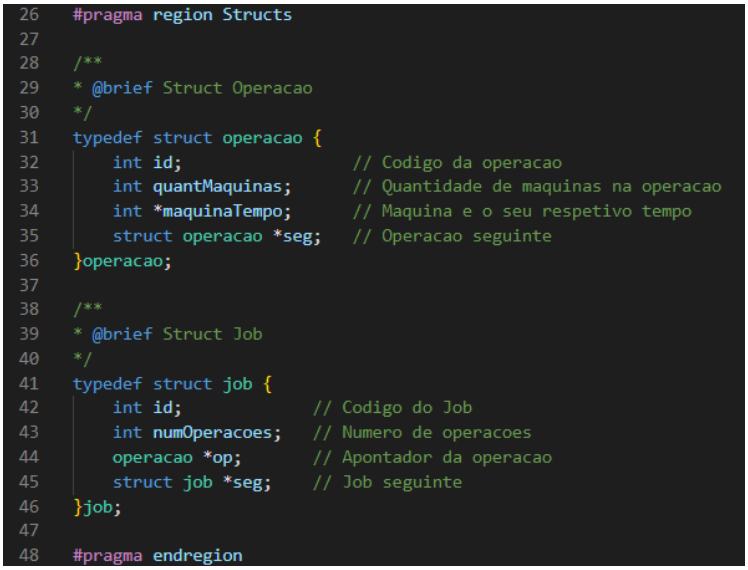

Figura 5 - Structs

Para finalizar estão colocadas todas as funções utilizadas no projeto divididas por categorias (Operações, Jobs, Ficheiros e outras funções).

header.h > ...

```
#pragma region Assinaturas Operacao

    operacao *inicializarOperacao();
    operacao *novoNoOperacao();
    void inserirNovaOperacao(operacao *op, int *idCont);
    void listaOperacao(operacao *op, int numOperacoes);
    void editaOperacao(operacao *op);
    void determinaTempoCurto(operacao *op);
    void determinaTempoLongo(operacao *op);

    void listarNoOperacao(operacao *op);
    void leOperacao(operacao *op);
    void removeOperacao(operacao *op, int *numOperacoes);
    void insereNoOperacao(operacao *op, int *numOperacoes);

#pragma endregion

#pragma region Assinaturas Job

    job *inicializarJob();
    job *novoNoJob(int key);

    void criaOperacaoApartirFich(operacao *op, int idOp, int cont, int arrM[], int arrT[]);

    void inserirJob(job *jb, int *idContJob);
    void removeJob(job *jb);

    void listarNoJob(operacao *auxOp, job *jb);
    void listarJobEmListaOrdenada(job *j);

    job *descobreJobLista(job *root, int key) ;
    job *descobreJob(job *auxjb);

#pragma endregion

#pragma region Assinaturas Ficheiro


    void verificarDadosNoFicheiro(operacao *op, int *idCont, int *numOperacoes);
    void criaOperacaoApartirFich(operacao *op, int idOp, int cont, int arrM[], int arrT[]);
    void escreveNoFich(operacao *op, int id, FILE *f_JOB);
    void guardaDadosFile(job *jb, char *nomeFich);
    job *verificaDadosNoFich(job *jb, int *idContJob);

#pragma endregion
 #pragma region Outras Assinaturas

     void menu(int *opcao);
     int procuraMaquinaIgual(operacao *maq, int procElem, int posAtual);
     int simNao();

 #pragma endregion Outras Assinaturas

 #endif
```

Figura 6 - Assinaturas

Functions.c

No último ficheiro a apresentar trata-se do functions.c, que não é nada mais nada menos que o ficheiro utilizado para guardar todas as funções existentes neste projeto

Iniciar Operação / Iniciar Job

Estas funções são utilizadas para abrir/criar uma lista para as operações e os Jobs.

```
20  /**
21  * @brief Inicia a lista das operações
22  */
23  operacao *inicializarOperacao() {
24      operacao *op;
25      op = (operacao*) malloc(sizeof(operacao));
26
27      if(op==NULL){
28          system("cls");
29          printf("Nao foi possivel criar a lista.\n\n");
30          system("pause");
31          return(NULL);
32      }
33      else {
34          (*op).id=0;
35          (*op).seg=NULL;
36          return(op);
37      }
38  }
```

Figura 7 - InicializarOperação

```
533 /**
534 * @brief Inicia a lista dos jobs
535 */
536 job *inicializarJob() {
537     job *p;
538     p = (job *) malloc( sizeof(job));
539
540     if(p==NULL) {
541         system("cls");
542         printf("E impossivel criar a estrutura\n\n");
543         system("pause");
544         return(NULL);
545     }
546     else {
547         (*p).numOperacoes = 0;
548         (*p).op = inicializarOperacao();
549         return(p);
550     }
551 }
```

Figura 8 - InicializarJob

Listar Jobs

Para listar os Jobs existentes são utilizadas as seguintes funções:

- **ListarJobEmListaOrdenada:** Esta função é puxada no main e percorre Job a Job para listar.

```
664   /**
665   * @brief Listagem dos jobs ordenados
666   */
667   void listarJobEmListaOrdenada(job *j) {
668       while(j->seg != NULL) {
669           listarNoJob(j->op,j);
670           j = j->seg;
671       }
672   }
```

Figura 9 - ListarJobEmListaOrdenada

- **ListarNoJob:** A função seguinte conta as operações existentes do job.

```
643   /**
644   * @brief  Lista os Jobs da lista
645   */
646   void listarNoJob(operacao *auxOp, job *jb) {
647       int i = 0;
648
649       for(i = 0; i < 25; i++) printf("%c",205);
650       printf("\nJOB N%c %d\n",167,jb->id);
651
652       if(jb->numOperacoes > 0) {
653           printf("Contagem de operacoes: %d\n",jb->numOperacoes);
654           while(auxOp->seg != NULL) {
655               listarNoOperacao(auxOp);
656               auxOp = auxOp->seg;
657           }
658       }
659       else {
660           printf("O job nao tem operacoes\n\n");
661       }
662   }
```

Figura 10 - ListarNoJob

- **ListarNoOperação:** A seguinte função lista todas as operações do Job selecionado.

```
413  /**
414  * @brief Listar a operação selecionada
415  */
416  void listarNoOperacao(operacao *op) {
417      int j = 0;
418
419      printf("Id - (%d)\n",op->id);
420      printf("Maquina - (");
421      for (j = 0; j < op->quantMaquinas; ++j) {
422          if((op->quantMaquinas - j) == 1)
423              printf("%d",op->maquinaTempo[0*op->quantMaquinas + j]);
424          else
425              printf("%d,",op->maquinaTempo[0*op->quantMaquinas + j]);
426      }
427      printf(")\nTempo - [");
428      for (j = 0; j < op->quantMaquinas; ++j) {
429          if((op->quantMaquinas - j) == 1)
430              printf("%d",op->maquinaTempo[1*op->quantMaquinas + j]);
431          else
432              printf("%d,",op->maquinaTempo[1*op->quantMaquinas + j]);
433      }
434      printf("]\n\n");
435
436  }
```

Figura 11 - ListarNoOperação

Inserir Job

Para inserir um Job são utilizadas as seguintes funções:

- **InserirJob:** A seguinte função começa com incrementar mais um Job e depois de o criar questiona o utilizador se pretende já adicionar as operações aos Jobs.

```
574  /**
575  * @brief Inserir um novo job
576  */
577  void inserirJob(job *jb, int *idContJob) {
578      job *auxjb = NULL, *atravesLista = NULL;
579      int SN;
580
581      system("cls");
582      (*idContJob)++;
583      auxjb = novoNoJob((*idContJob));
584      atravesLista = jb;
585      while (atravesLista->seg != NULL) {
586          atravesLista = atravesLista->seg;
587      }
588      atravesLista->seg = auxjb;
589
590      printf("Quer adicionar operacoes ao novo job?\n");
591      while (SN = simNao(), SN == 1) {
592          insereNoOperacao(atravesLista->op,&(atravesLista->numOperacoes));
593      }
594  }
```

Figura 12 - InserirJob

- **NovoNoJob:** Esta função cria o Job na lista.

```
553   /**
554   * @brief criação de um job na lista
555   */
556   job *novoNoJob(int key) {
557       job *no = (job *) malloc(sizeof(job));
558
559       if(no == NULL) {
560           system("cls");
561           printf("Nao foi possivel criar a lista\n\n");
562           system("pause");
563           return NULL;
564       }
565       else {
566           no->id = key;
567           no->numOperacoes = 0;
568           no->op = novoNoOperacao();
569           no->seg = NULL;
570           return no;
571       }
572   }
```

Figura 13 - NovoNoJob

- **NovoNoOperacao:** Tal como foi utilizado para criar um job na lista, também será necessário criar uma operação na lista, e é essa a funcionalidade da função seguinte.

```
40   /**
41   * @brief Inicializa o Job na lista
42   */
43   operacao *novoNoOperacao() {
44       operacao *p = (operacao*) malloc(sizeof(operacao));
45
46       if(p == NULL) {
47           system("cls");
48           printf("Nao foi possivel criar a lista.""\n\n");
49           system("pause");
50           return NULL;
51       }
52       else {
53           p->id = 0;
54           p->quantMaquinas = 0;
55           p->maquinaTempo = NULL;
56           p->seg = NULL;
57           return p;
58       }
59   }
```

Figura 14 - NovoNoOperacao

- **InsereNoOperacao:** Caso o utilizador pretenda adicionar as operações, a seguinte função criará operações ao job.

```c
/**
* @brief Inserir uma seguinte operação
*/
void insereNoOperacao(operacao *op, int *numOperacoes) {
    operacao *auxOp;
    int id = 1;

    auxOp = novoNoOperacao();
    system("cls");

    if(auxOp==NULL) {
        printf("Não há mais espaço de memória. É impossível inserir na operacao.\n\n");
        system("pause");
    }
    else {
        while((op->seg) != NULL) {
            id = op->id + 1;
            op = op->seg;
        }

        (*numOperacoes)++;
        op->id = id;
        leOperacao(op);
        op->seg = auxOp;
    }
}
```

Figura 15 - InsereNoOperacao

- **LeOperacao:** Por último esta função irá perguntar ao utilizador as informações acerca das operações.

```c
/**
* @brief Recolha de informação de uma operação
*/
void leOperacao(operacao *op) {
    int i, j, verificar;

    printf("Quantas máquinas podem ser utilizadas para esta operação: ");
    scanf("%d",&op->quantMaquinas);

    op->maquinaTempo = (int *)malloc(sizeof(int[2][op->quantMaquinas]));
    for (i = 0; i < 2; i++) {
        for (j = 0; j < op->quantMaquinas; j++) {
            if(i == 0) {
                do {
                    if( j > 0) {
                        verificar = 0;
                        printf("Qual é o id da máquina que pretende utilizar: ");
                        scanf("%d",&op->maquinaTempo[i*op->quantMaquinas + j]);

                        verificar = procuraMaquinaIgual(op,op->maquinaTempo[i*op->quantMaquinas + j],j);
                        if(verificar == 0) {
                            printf("\nA maquina %d ja esta em uso. Escolha outra maquina\n\n", op->maquinaTempo[i*op->quantMaquinas + j]);
                        }
                    }
                    else {
                        verificar = 1;
                        printf("Qual é o id da máquina que pretende utilizar: ");
                        scanf("%d",&op->maquinaTempo[i*op->quantMaquinas + j]);
                    }
                }while(verificar == 0);
            }
            else {
                printf("Quanto tempo demorará a maquina %d: ", op->maquinaTempo[0*op->quantMaquinas + j]);
                scanf("%d",&op->maquinaTempo[i*op->quantMaquinas + j]);
            }
        }
    }
}
```

Figura 16 - LeOperacao

Remover Job

Para remover um Job é utilizada a seguinte função:

- **RemoveJob:** A função irá pedir primeiramente o código do Job que o utilizador pretende remover, e de seguida irá mostra os dados do Job que foi removido. Caso dê um erro a função avisa que o código é inexistente.

```
596  /**
597  * @brief Remocao de um job
598  */
599  void removeJob(job *jb) {
600      job *j, *atras, *frente, *auxOp;
601      int ElemRetirar;
602
603      system("cls");
604      j = jb;
605
606      if((j->seg)==NULL) {
607          printf("Nao tem operacoes na lista\n");
608      }
609      else {
610          printf("Qual é o codigo do Job que deseja remover\n");
611          scanf("%d",&ElemRetirar);
612
613          if(ElemRetirar == jb->id) {
614              system ("cls");
615              printf("O job foi removido\n");
616              listarNoJob(jb->op,jb);
617              jb = jb->seg;
618              free(j);
619          }
620          else {
621              auxOp = jb;
622              while((ElemRetirar != auxOp->id) && (auxOp->seg != NULL)) {
623                  atras = auxOp;
624                  auxOp = auxOp->seg;
625                  frente = auxOp->seg;
626              }
627
628              if(ElemRetirar == auxOp->id) {
629                  atras->seg = frente;
630                  system ("cls");
631                  printf("O job foi removido\n");
632                  listarNoJob(jb->op,jb);
633                  free(auxOp);
634              }
635              else {
636                  system("cls");
637                  printf("O job com o numero %d nao existe na lista", ElemRetirar);
638              }
639          }
640      }
641  }
```

Figura 17 - RemoveJob

Inserir operação num determinado Job

Para inserir uma operação num determinado job são utilizadas as seguintes funções:

DescobreJob: Esta função é utilizada para pedir o código ao utilizador e verificar se o job existe.

```
688  /**
689  * @brief Procura o job que precisa
690  */
691  job *descobreJob(job *auxjb) {
692      job *jobADescobrir;
693      int ElemDescobrir;
694
695      system("cls");
696
697      if(auxjb == NULL) {
698          printf("Não há jobs na lista\n");
699          system("cls");
700          return NULL;
701      }
702      else {
703          printf("Qual é o id que pretende encontrar?\n");
704          scanf("%d",&ElemDescobrir);
705
706          jobADescobrir = descobreJobLista(auxjb,ElemDescobrir);
707          if(jobADescobrir != NULL)
708              return jobADescobrir;
709
710          printf("O Job com o codigo %d não existe!\n", ElemDescobrir);
711          return NULL;
712      }
713  }
```

Figura 18 - DescobreJob

De seguida são utilizadas as funções **InsereNoOperacao** e **LeOperacao** (já mencionadas em cima).

Remover operação de um determinado Job

Primeiramente é usada a função **DescobreJob** (mencionada acima) para descobrir se existe o job, e de seguida para remover uma operação desse job é utilizada a seguinte função:

- **RemoveOperacao:** Esta função irá pedir ao utilizador o código da operação que o utilizador deseja remover e depois de indicar o código irá mostrar a operação que foi removida.

```
245  /**
246  * @brief Remove uma operação
247  */
248  void removeOperacao(operacao *op, int *numOperacoes) {
249      operacao *j, *atras, *frente, *auxOp;
250      int ElemRetirar;
251
252      system("cls");
253      j = op;
254
255      if((j->seg)==NULL) {
256          printf("Não existe operações na lista.\n");
257      }
258      else {
259          printf("Qual é o código cuja operação pretende remover?\n");
260          scanf("%d",&ElemRetirar);
261
262          if(ElemRetirar == op->id) {
263              system ("cls");
264
265              (*numOperacoes)--;
266              printf("O elemento foi removido\n");
267              listarNoOperacao(op);
268              op = op->seg;
269              free(j);
270          }
271          else {
272              auxOp = op;
273              while((ElemRetirar != auxOp->id) && (auxOp->seg != NULL)) {
274                  atras = auxOp;
275                  auxOp = auxOp->seg;
276                  frente = auxOp->seg;
277              }
278
279              if(ElemRetirar == auxOp->id) {
280                  atras->seg = frente;
281                  system ("cls");
282
283                  (*numOperacoes)--;
284                  printf("O elemento foi removido!\n");
285                  listarNoOperacao(auxOp);
286                  free(auxOp);
287              }
288            else {
289                system("cls");
290                printf("O elemento com código %d não existe na lista.", ElemRetirar);
291            }
292        }
293    }
294  }
```

Figura 19 - RemoveOperacao

Editar Operação de um determinado Job

No começo é pedido o código do job em que o utilizador pretenda editar uma operação usando a função **DescobreJob**, de seguida usada a seguinte função:

- **EditaOperação:** A função pede o código da operação que pretenda editar e de seguida o utilizador atualiza os dados ao seu gosto.

```
205  /**
206  * @brief Edita uma operação
207  */
208  void editaOperacao(operacao *op) {
209      int ElemEditar;
210
211      system("cls");
212
213      if((op->seg) == NULL) {
214          printf("Nao existe operacoes na lista.\n");
215      }
216      else {
217          printf("Qual e o codigo da operacao que deseja editar?\n");
218          scanf("%d",&ElemEditar);
219
220          if(ElemEditar == op->id) {
221              system ("cls");
222              printf("O elemento esta a ser editado.\n");
223              listarNoOperacao(op);
224              leOperacao(op);
225          }
226          else {
227              while((ElemEditar != op->id) && (op->seg != NULL)) {
228                  op = op->seg;
229              }
230
231              if(ElemEditar == op->id) {
232                  system ("cls");
233                  printf("O elemento esta a ser editado.\n");
234                  listarNoOperacao(op);
235                  leOperacao(op);
236              }
237              else {
238                  system("cls");
239                  printf("O elemento com id %d não existe na lista", ElemEditar);
240              }
241          }
242      }
243  }
```

Figura 20 - EditaOperacao

Testes realizados

Durante o desenvolvimento do código foram feitos testes num ficheiro de texto. Nos testes da fase 1 foi adicionando primeiro as operações que seriam necessárias e de seguida as máquinas com o seu tempo.

Também foram feitos testes para as funções que determinam o máximo e o mínimo de tempo necessário para completar um Job, e por fim toda a informação é guardada no ficheiro de texto chamado dados.txt.

Para a fase 2 foram adicionados, removido e editado Jobs para além das operações que o programa já fazia.

Fazendo agora um teste, vamos adicionar um novo Job.

Primeiramente o programa abre o Menu, e será selecionado a opção 8:

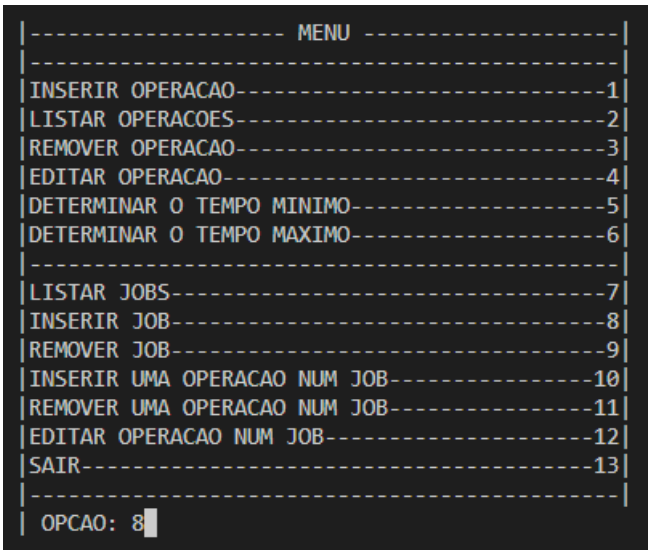

Figura 21 - Menu

De seguida pergunta se pretende adicionar as operações ao novo Job, vamos responder que sim.

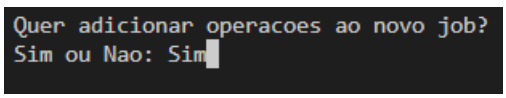

Figura 22 - Adicionar operações ao job

Depois de reponder que sim o programa pergunta quantas máquinas serão utilizadas na operação, respondemos 2.

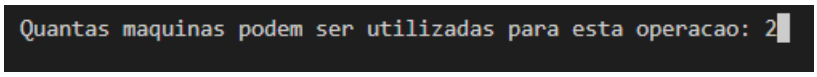

Figura 23 - Maquinas utilizadas na operação

Respondemos os dados que queríamos e quando terminamos o programa pergunta se pretendo adicionar mais uma operação, respondemos Não.

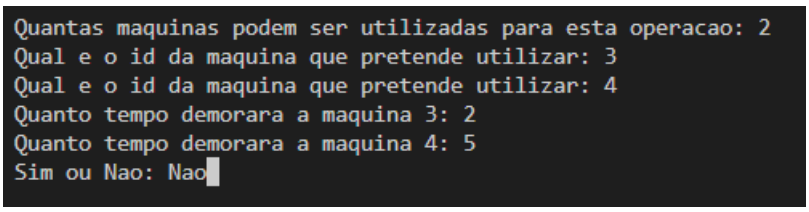

Figura 24 - Dados da operação

Adicionada a operação, fomos listar o Job criado e verificamos que este se encontra visível como o Job nº 9.

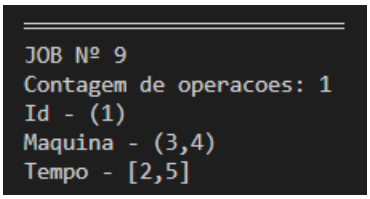

Figura 25 - Job adicionado

Agora voltamos ao menu e selecionamos a opção “Inserir uma operação num Job” e agora vamos inserir os dados necessários.

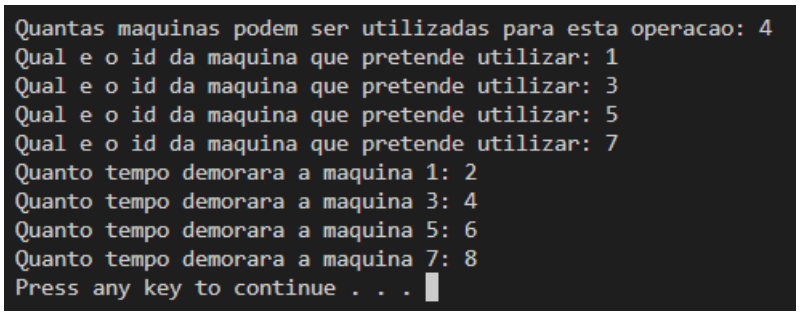

Figura 26 - Dados para atualizar

Como podemos verificar, a nova operação foi adicionada com sucesso!

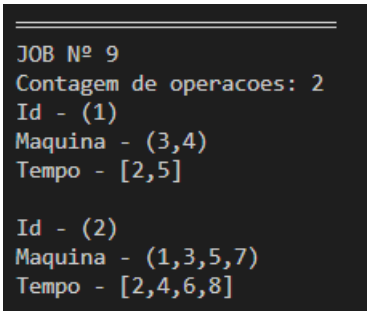

Figura 27 - Atualização de dados

De seguida vamos editar a operação nº 1, selecionamos no menu a opção 12 e o programa pergunta qual o id do Job em que pretendo alterar uma operação.

Selecionamos o Job nº 9 e a operação nº 1 tal como combinamos, e colocamos com os seguintes dados.

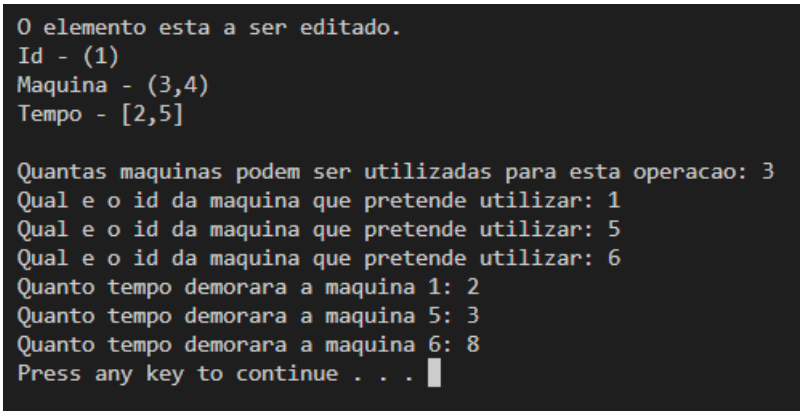

Figura 28 - Operação editada

Listamos os Jobs e como podemos verificar a operação foi editada com sucesso!

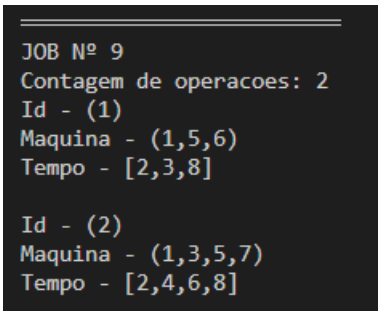

Figura 29 - Dados atualizados

De seguida vamos remover a operação Nº 2, no menu selecionamos a opção 11 e indicamos que queremos remover do job Nº 9 a operação Nº 2.

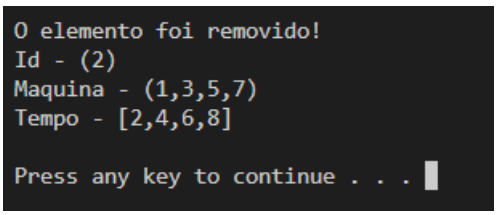

Figura 30 - Remoção de uma operação

Ao fazer a listagem verifica-se que a operação foi eliminada com sucesso.

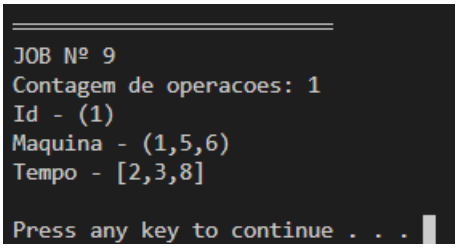

Figura 31 - Listagem atualizada

Por fim, e para finalizar a nossa fase de testes, iremos remover o Job no qual fizemos o teste a todas as funcionalidades, o Job nº 9.

Depois de selecionar a opção 9 do menu, indicamos que queríamos remover o Job Nº 9 e esta foi a listagem após a remoção.

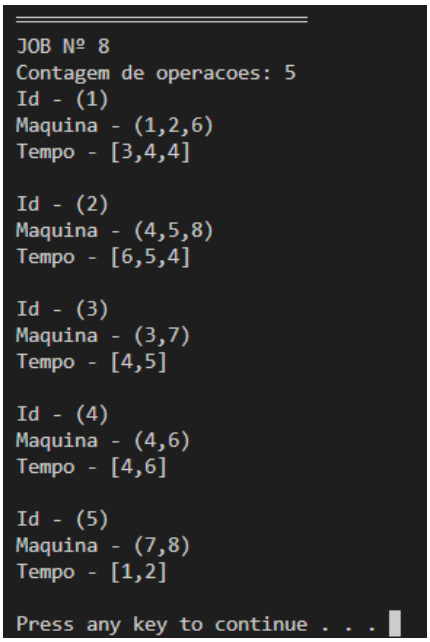

Figura 32 - Listagem final

Conclusão

Podemos assim dar o projeto concluído com sucesso, onde atingimos todos os objetivos propostos. Consideramos um projeto interessante e desafiante, uma vez que conseguimos aperfeiçoar as capacidades desenvolvidas nas aulas, mas também aprender novas técnicas de programação em C, desenvolvendo o nosso conhecimento da linguagem para além do que nos é lecionado.

Por fim, agradeço a dedicação e a entrega que senti por parte do professor João Carlos Silva que sempre se demonstrou disposto a esclarecer-me e a enriquecer o meu conhecimento para assim ser possível a realização deste projeto.

Bibliografia

Como bibliografia usei bastante os exercícios lecionados nas aulas e a internet, principalmente o StackOverFlow.