Universidade do Minho

Métodos de Programação 1

MiETI :: 1º Ano

2018/2019

- Eduardo Paiva, 85312

eduardopaiva@live.com.pt

- Tiago Ferreira, 85392

tiago.ferreira.19@hotmail.com

Universidade do Minho

Universidade do Minho

ÍNDICE

INTRODUÇÃO

Este projeto realizado no âmbito da cadeira de Métodos de Programação I visa criar um programa que obtenha um conjunto de comandos que operem o braço de um robot sob a forma de manipular blocos dispostos numa mesa.

Neste relatório iremos apresentar a nossa solução para o problema proposto, a descrição e também o algoritmo refinado e não refinado da mesma.

No final encontra-se o código em linguagem C, devidamente comentado em inglês.

DESCRIÇÃO DO PROBLEMA

O problema consiste em operar um braço robot com o intuito de manipular blocos dispostos numa mesa.

O número de blocos dispostos na mesa é dado pelo utilizador, bem como os comandos para operar o braço robot.

Existem no total 4 comandos para controlar o robot e um comando para parar a manipulação do mesmo.

Comandos válidos:

Move a onto b – a e b são números de blocos. Coloca o bloco a em cima do bloco b mas apenas depois de retornar os blocos por cima do a e do b às suas posições originais;

Move a over b - a e b são números de blocos. Coloca o bloco a no topo da pilha que contem o bloco b depois de retirar todos os blocos por cima do a para as suas posições originais;

Pile a onto b - a e b são números de blocos. Move o bloco a e todos os blocos por cima deste para cima do bloco b depois de retornar todos os blocos por cima do b à sua posição original;

Pile a over b – a e b são números de blocos. Move o o bloco a e todos os blocos acima deste para cima do bloco b;

Quit – Terminar a manipulação.

Qualquer comando em que a == b e a e b estejam na mesma linha devem ser ignorados.

DESCRIÇÃO DA SOLUÇÃO

Algoritmo não refinado:

1. Ler o número de blocos a ser utilizados;

2. Ler e analisar os comandos inseridos pelo utilizador. Realizar na matriz que contem os blocos as devidas alterações que os comandos fazem. Se o comando inserido for “quit”, avançar para o passo 3;

3. Imprimir a matriz.

Algoritmo refinado:

1.[main]

1.1[ler o número de blocos(n)]

1.2[inicializar a matriz de tamanho n linhas por n colunas(matrix[n][n])

1.2.1[preencher a primeira posição de cada coluna com o número 0 a n]

2.[Ler o comando]

2.1 Enquanto (a!=3) fazer

2.1.1. Ler move1

Se (move1==”quit”)

Sair do ciclo

Ler block, move2, block2

2.1.2[Ciclo para determinar em que linhas se encontram o block e o block2]

2.1.2.1. Guardar em b a linha que contem o block

´ 2.1.2.2. Guardar em c a linha que contem o block2

2.1.3[Comparar o número do block com o block2 e o b com o c]

2.1.3.1 Se (block!=block2 e b!=c)

2.1.3.1.1 Se (move1==”move” e move2==”onto”)

Chamar as seguintes funções:

ReturnBlock(n, matrix, block)

ReturnBlock2(n, matrix, block2)

BlockOnTopOfBlock2(n, matrix, block, block2)

2.1.3.1.2 Se (move1==”move” e move2==”over”)

Chamar as seguintes funções:

ReturnBlock(n, matrix, block)

BlockOnTopOfBlock2(n, matrix, block, block2)

2.1.3.1.3 Se (move1==”pile” e move2==”onto”)

Chamar as seguintes funções:

ReturnBlock2(n, matrix, block2)

BlockOnTopOfBlock2(n, matrix, block, block2)

2.1.3.1.4 Se (move1==”pile” e move2==”over”)

Chamar as seguintes funções:

BlockOnTopOfBlock2(n, matrix, block, block2)

3.[Apresentar a matriz final]

3.1. Se (posição da matriz == -1)

Não imprimir

Funcionamento das funções:

4.[Função ReturnBlock(n, matrix, block)]

4.1[Ciclo que percorre a matriz]

Se (matrix[i][j]==block)

4.1.1[Ciclo que percorre a linha onde se encontra o block]

Se (matrix[i][j]!=-1)

a toma o valor de o numero que esta na matrix[i][b]

matrix[a][0]= matrix[i][b]

matrix[i][b]=-1

5.[Função ReturnBlock(n, matrix, block)}]

5.1[Ciclo que percorre a matriz]

Se (matrix[i][j]==block2)

5.1.1[Ciclo que percorre a linha onde se encontra o block2]

Se (matrix[i][j]!=-1)

a toma o valor de o número que esta na matrix[i][b]

matrix[a][0]= matrix[i][b]

matrix[i][b]=-1

6.[FunçãoBlockOnTopOfBlock2(n, matrix, block, block2)]

6.1[Ciclo que determina a linha e a coluna onde se encontra o block e block2]

6.1.1 Se(matrix[i][j] == block)

m=i

o=j

6.1.2 Se(matrix[i][j] == block)

k=i;

l=j;

6.2.[Ciclo que corre a linha onde se encontra o block2]

6.2.1 Se(matrix[k][c]== -1)

Se ((matrix[m][o] == -1)

Sai do ciclo

matrix[k][c]= matrix[m][o]

matrix[m][o]=-1

Descrição do algoritmo:

Para descrever o algoritmo vamos utilizar um pequeno exemplo.

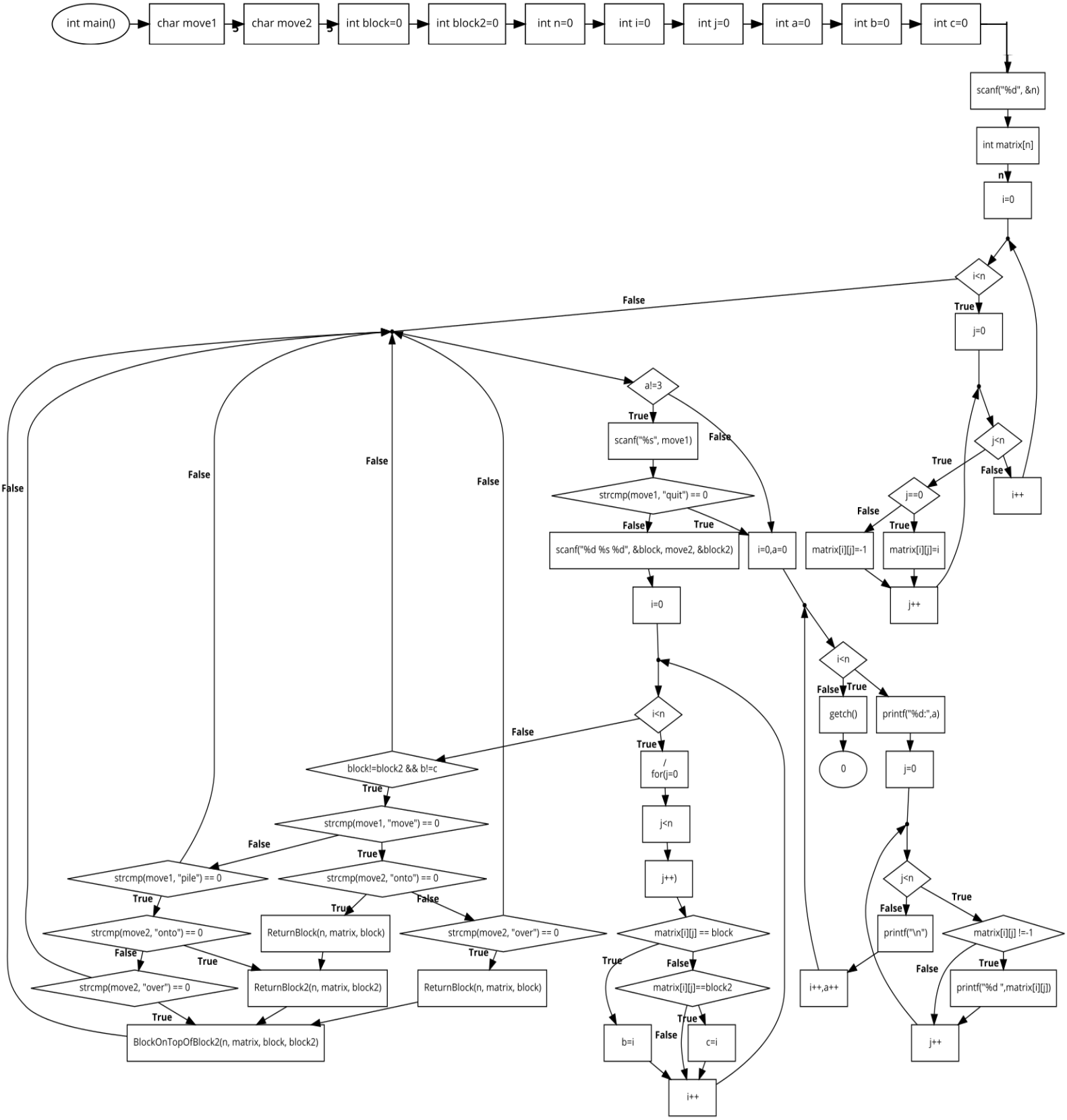
O utilizador inseriu o número 10 e depois os seguintes comandos: move 9 onto 1, quit.

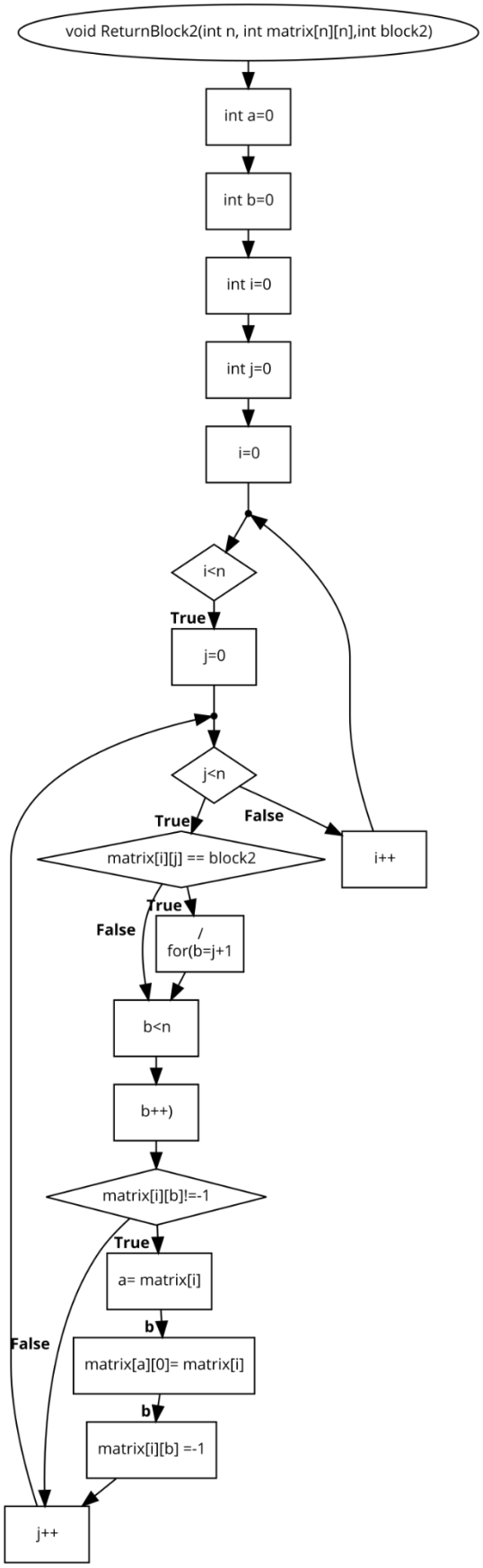
Neste caso irão ser usados 10 blocos, a matriz é preenchida na primeira coluna com o número da linha em que se encontra e o resto das posições com o numero -1. Analisa-se o comando e verifica-se se 9 é diferente de 1 e se estes não estão na mesma linha. Se ambas das condições forem verdade verifica-se que o primeiro tipo que é “move” e o segundo “onto”, chama-se a função que retorna os blocos por cima de 9 e a função que retorna os blocos por cima de 1, por último chama-se a função que coloca o 9 em cima de 1.

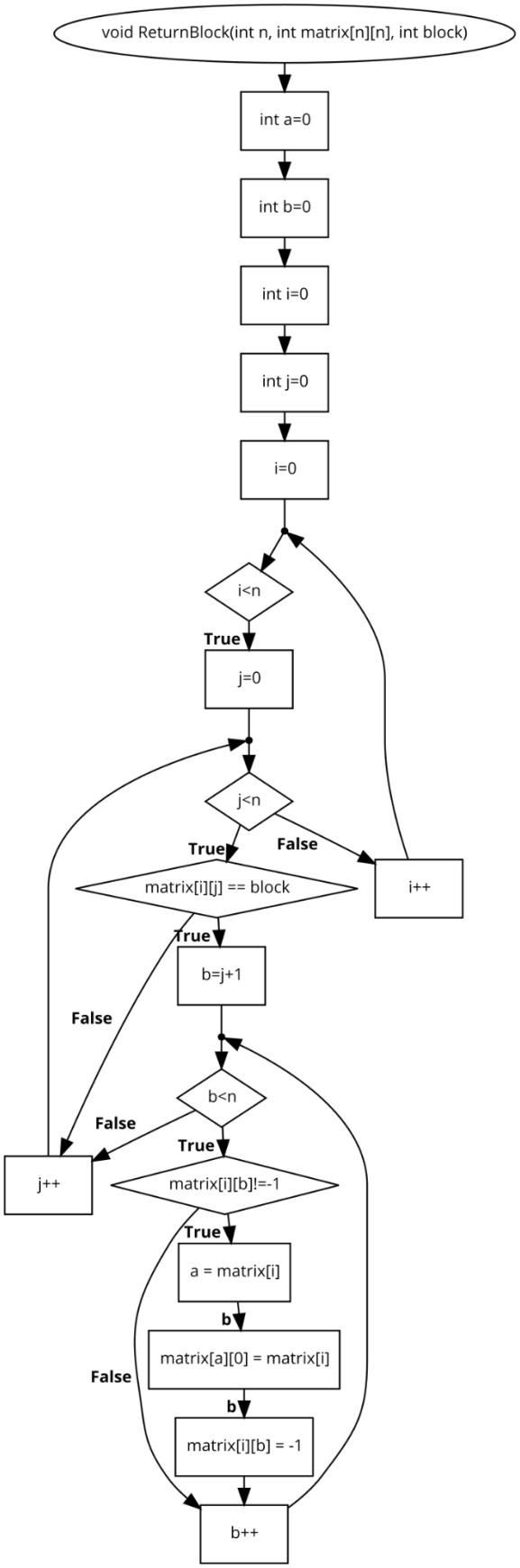
O segundo comando inserido foi “quit” que faz com que seja imprimida a matriz com os blocos.

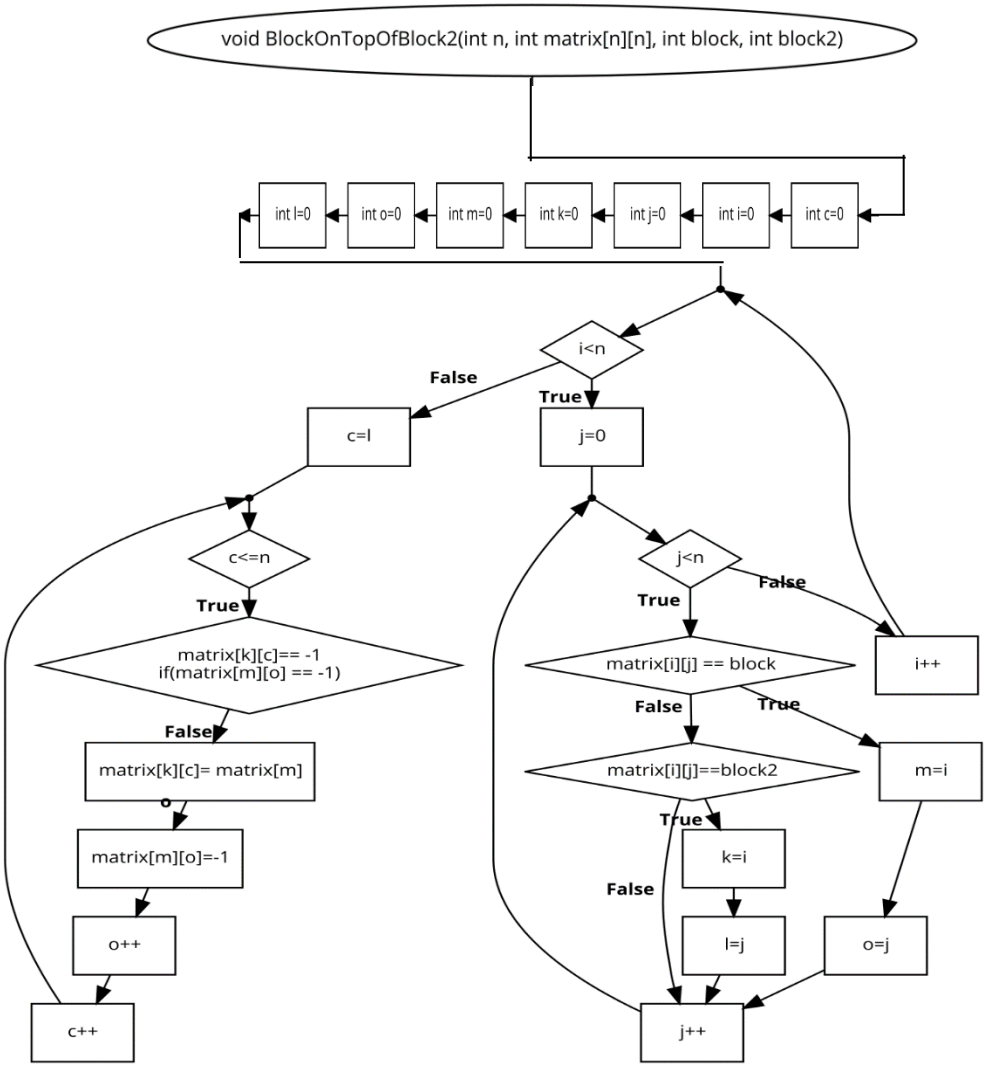
Universidade do Minho

FLUXOGRAMA




void ReturnBlock2(int n, int matrix[n][n],int block2)
int a=0
int b=0
int i=0
int j=0
i=0
i<n
True
j=0
j<n
True
False
matrix[i][j] == block2
i++
True
False
/
for(b=j+1
b<n
b++)
matrix[i][b]!=-1
True
a= matrix[i]
b
False
matrix[a][0]= matrix[i]
b
matrix[i][b] =-1
j++


void ReturnBlock(int n, int matrix[n][n], int block)
int a=0
int b=0
int i=0
int j=0
i=0
i<n
True
j=0
j<n
True
False
matrix[i][j] == block
i++
True
b=j+1
False
b<n
False
True
j++
matrix[i][b]!=-1
True
a = matrix[i]
b
False
matrix[a][0] = matrix[i]
b
matrix[i][b] = -1
b++


void BlockOnTopOfBlock2(int n, int matrix[n][n], int block, int block2)
int l=0
int o=0
int m=0
int k=0
int j=0
int i=0
int c=0
i<n
False
True
c=l
j=0
c<=n
True
matrix[k][c]== -1
if(matrix[m][o] == -1)
False
matrix[k][c]= matrix[m]
matrix[m][o]=-1
o++
c++
j<n
True
False
matrix[i][j] == block
False
True
matrix[i][j]==block2
True
False
k=i
l=j
m=i
o=j
i++
j++

CONCLUSÃO

Através da solução que desenvolvemos foi possível resolver o problema apresentado de forma eficaz e precisa.

O processo mais trabalhoso foi a elaboração do algoritmo, uma vez que tentamos que este fosse simples e fácil de entender.

A transformação do algoritmo para linguagem C foi acessível visto que desenvolvemos ao longo do semestre as capacidades necessárias para o mesmo.

ANEXOS

Código em linguagem C:

```
/*Pratical Project MP1 2018/2019*/
/*Made by Tiago Ferreira(A85392) and Eduardo Paiva(A85312)*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*function that returns the numbers above block to its original positions*/
void ReturnBlock(int n, int matrix[n][n], int block){

   int a=0;
   int b=0;
   int i=0;
   int j=0;

      for(i=0; i<n; i++){ /*cycle that runs through the matrix*/

            for(j=0; j<n; j++){

                  if(matrix[i][j] == block){/*checks whether the matrix number is equal to the block*/

                     for(b=j+1; b<n; b++){

                           if(matrix[i][b]!=-1){/*checks if the number on the matrix[i][b] is different from
-1*/

                              a = matrix[i][b]; /*store in a the number that is in position [i][b]of the
matrix*/
```

```
                        matrix[a][0] = matrix[i][b]; /*places in position [a][0] of the matrix the number in position [i][b] of the matrix*/
                        matrix[i][b] = -1;

                    }

                }

            }

        }

    }
}
/*function that returns the numbers above block2 to its original positions*/
void ReturnBlock2(int n, int matrix[n][n],int block2){

    int a=0;
    int b=0;
    int i=0;
    int j=0;

    for(i=0; i<n; i++){/*cycle that runs through the matrix*/

        for(j=0; j<n; j++){

            if(matrix[i][j] == block2){/*checks whether the matrix number is equal to the block2*/

                for(b=j+1; b<n; b++){

                    if(matrix[i][b]!=-1){/*checks is the number on the matrix[i][b] is different from -1*/
```

```
                    a= matrix[i][b]; /*store in a the number that is in position [i][b]of the matrix*/
                    matrix[a][0]= matrix[i][b]; /*places in position [a][0] of the matrix the number in position [i][b] of the matrix*/
                    matrix[i][b] =-1;

                }

            }

        }

    }

}
/*function that places the pile that contain block on the top of block2*/
void BlockOnTopOfBlock2(int n, int matrix[n][n], int block, int block2){

    int c=0;
    int i=0;
    int j=0;
    int k=0;
    int m=0;
    int o=0;
    int l=0;

     for(i=0; i<n; i++){ /*cycle that determines the row and column of the block and saves the line in [m] and the column in [o]. does the same for block2, line[k] and column[l]*/

        for(j=0; j<n; j++){

            if(matrix[i][j] == block){

                m=i;
```

```
                o=j;

                }else if(matrix[i][j]==block2){

                        k=i;
                        l=j;

                }

            }

    }


    for(c=l; c<=n; c++){/*cycle that runs the line where the block2 is*/

        if( matrix[k][c]== -1){ /*checks is the number on the matrix[k][c] is different from -1*/

            if(matrix[m][o] == -1){/*if the number on the matrix[m][o] is equal to -1 exit the cycle*/

                break;

            }

            matrix[k][c]= matrix[m][o]; /*the position [k] [c] takes the value of position [i] [j]*/
            matrix[m][o]=-1; /*the position [m] [o]  takes the value of -1*/
            o++; /*increase o*/

        }

    }

}
/*Main function*/
int main(){
```

```
char move1[5]; /*string for the first type of move*/
char move2[5]; /*string for the second type of move*/
int block=0; /*variable for the first block*/
int block2=0; /*variable for the second block*/
int n=0; /*variable that stores the number of blocks*/
        int i=0;
int j=0;
int a=0;
int b=0;
int c=0;

scanf("%d", &n); /*scan how many blocks*/
int matrix[n][n]; /*matrix which contain the blocks*/

/*fills the first position of each column of the matrix with the numbers 0 to n-1 and the rest with the number -1*/
for(i=0; i<n; i++){

      for(j=0; j<n; j++){

           if(j==0){

              matrix[i][j]=i;

           }else{

           matrix[i][j]=-1;

           }

      }

}
```

```
/*cycle to read the moves*/
while(a!=3){ /*the cycle runs while "a" is different from 3*/

        scanf("%s", move1); /*read the type of move*/

        if(strcmp(move1, "quit") == 0){ /*if the movement is "quit" it leaves the cycle*/

           break;

           }

        scanf("%d %s %d", &block, move2, &block2); /*read  the first number, second type of move and the second number*/

        for(i=0; i<n; i++){ /*cycle that determines which line of the matrix is the block and block and store it in b and c */

            for(j=0; j<n; j++){

                if(matrix[i][j] == block){

                   b=i;

                }else if(matrix[i][j]==block2){

                   c=i;

                }

            }

        }

      if(block!=block2 && b!=c  ){ /*if the block is different from block2 and b is different form c*/
```

```
if(strcmp(move1, "move") == 0){ /*verifies that move1 is equal to "move"*/

    if(strcmp(move2, "onto") == 0){ /*verifies that move2 is equal to "onto"*/

        ReturnBlock(n, matrix, block); /*call the function that returns the numbers above block to its original positions*/

        ReturnBlock2(n, matrix, block2);/*call the function that returns the numbers above block2 to its original positions*/

        BlockOnTopOfBlock2(n, matrix, block, block2);/*call the function that places the pile that contain block on the top of block2*/

    }else if(strcmp(move2, "over") == 0){ /*verifies that move2 is equal to "over"*/

        ReturnBlock(n, matrix, block);/*call the function that returns the numbers above block to its original positions*/

        BlockOnTopOfBlock2(n, matrix, block, block2);/*call the function that places the pile that contain block on the top of block2*/

    }

}else if(strcmp(move1, "pile") == 0){ /*verifies that move1 is equal to "pile"*/

    if(strcmp(move2, "onto") == 0){ /*verifies that move2 is equal to "onto"*/

        ReturnBlock2(n, matrix, block2);/*call the function that returns the numbers above block2 to its original positions*/

        BlockOnTopOfBlock2(n, matrix, block, block2);/*call the function that places the block on the top of block2*/

    }else if(strcmp(move2, "over") == 0){ /*verifies that move2 is equal to "onto"*/

        BlockOnTopOfBlock2(n, matrix, block, block2);/*call the function that places the pile that contain block on the top of block2*/

    }

}
```

```
            }
        }
        for(i=0,a=0; i<n; i++,a++){ /*prints the final matrix*/
            printf("%d:",a);
           for(j=0; j<n; j++){
               if(matrix[i][j] !=-1){
                   printf("%d ",matrix[i][j]);
               }
            }
             printf("\n");
        }
        getch();
        return 0;
}
```