Trabalho Prático 1
Algoritmos e Estruturas de Dados II

Lucas Caetano Lopes Rodrigues

Belo Horizonte
2017

1. Introdução

A abstração de problemas tratando-os em domínios menos complexos que a realidade é fundamental para desenvolver várias áreas dentro da computação, como a pesquisa sobre inteligência artificial nas áreas de planejamento e robótica. Neste âmbito, uma das abstrações possíveis é trabalhar utilizando o “mundo dos blocos”, onde há um cenário com *n* blocos dispostos de tal forma que se possa fazer certas operações pré-definidas sobre eles.

O objetivo deste trabalho é utilizar a base teórica que nos foi ensinada sobre alocação dinâmica de memória, apontadores e tipos abstratos de dados, tais como listas e pilhas, para implementar a manipulação dos blocos nesse domínio.

Para isso, além dos Tipos Abstratos de Dados (TADs) necessários para a implementação do cenário dos blocos, deverão ser desenvolvidas as funções de manipulação dos blocos: **moveOnto**, **moveOver**, **pileOnto**, **pileOver**.

Com isso, espera-se praticar conceitos de manipulação de apontadores e espaços dinâmicos da memória, assim como desenvolver a lógica por trás das funcionalidades.

2. Implementação

2.1 Estruturas de dados

- Blocks

A estrutura Blocks descreve o cenário do mundo de blocos. Ela contém um vetor dinamicamente alocado de Pilhas. Cada uma das pilhas representa uma posição da qual se pode colocar ou retirar blocos.

```
typedef struct{
    Pile *ground;
    int size;
} Blocks;
```

- Funções e procedimentos:

1. createScenario(Blocks *scenario, int n): cria o cenário correspondente ao mundo de blocos com *n* blocos (e, consequentemente, *n* pilhas) no total. Não tem retorno.

2. findBlock(Blocks scenario, int key): encontra um item específico dentro do cenário, percorrendo todos os itens no fundo de todas as pilhas e retornando o índice do item quando encontrado.

- Pile

A estrutura Pile é uma pilha, que contém as propriedades *bottom* e *top*, que indicam o fundo e o topo da pilha, assim como o tamanho *size* da pilha. As propriedades *bottom* e *top* são apontadores para células, que encapsulam os itens.

```
typedef struct {
    Pointer bottom, top;
    int size;
} Pile;
```

    - Funções e procedimentos:

1. createEmptyPile(Pile *p): cria uma pilha vazia.
2. isPileEmpty(const Pile *p): checa se a pilha recebida por parâmetro está vazia.
3. pile(Item i, Pile *p): empilha o item *i* na pilha *p*.
4. unpile(Pile *p, Item *i): desempilha o item no topo da pilha *p*, armazenando seu valor no item *i*.
5. sizeOfPile(const Pile *p): retorna o tamanho da pilha.

- Cell

A estrutura Cell é uma implementação da Célula que contém um Item (com os dados efetivos) e um Pointer (que é um apontador para Cell) para a próxima célula na pilha.

```
typedef struct Cell_str *Pointer;

typedef struct Cell_str{
    Item content;
    Pointer next;
} Cell;
```

- Item

A estrutura Item contém uma chave primária que, além de ser a chave do item, é também seu conteúdo. Essa chave é um número inteiro.

```
typedef int primaryKey;

typedef struct{
    primaryKey key;
} Item;
```

2.2 Programa principal

O programa principal cria uma variável do tipo Blocks, que será o cenário utilizado durante todo o programa. Além disso, ele recebe os argumentos que correspondem aos arquivos de entrada e saída e checa se o número de parâmetros está correto.

Então, ele chama a função **_generateScenario(argv[1], &scenario)** que lê a primeira linha do arquivo de entrada (**argv[1]**) e cria o cenário de acordo com o número de blocos contido no arquivo.

Em seguida, o programa chama a função **_executeCommands**, que continua a leitura do arquivo para chamar as funções sequencialmente de acordo com como elas são chamadas no arquivo.

Após a chamada das funções correspondentes para cada linha do arquivo, o cenário encontra-se configurado e o programa principal chama a função **_writeOutput(scenario, argv[2])** para escrever o TAD Blocks no estado final no arquivo de saída.

3. Análise de complexidade

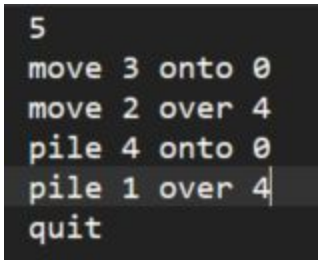
- createEmptyPile: Não faz nenhum loop, apenas aloca dinamicamente a pilha. Complexidade O(1).
- isPileEmpty: compara o topo da pilha com o fundo da pilha para saber se ela está vazia ou não. Complexidade O(1).
- pile: Faz operações simples de alocação de memória e mudança de apontadores. Complexidade O(1).
- unpile: Faz operações simples de liberação de memória e mudança de apontadores. Complexidade O(1).
- sizeOfPile: Retorna a propriedade size da pilha. Complexidade O(1).

- createScenario: Aloca dinamicamente um espaço correspondente a (*n* * *tamanho do TAD Pilha*), roda um loop de 0 a *n* chamando as funções **createEmptyPile** e **pile**. Como as funções citadas possuem complexidade O(1), a função **createScenario** possui complexidade O(n).
- findBlock: Roda dois loops aninhados procurando por itens em cada uma das pilhas do cenário. Complexidade O(n²), sendo *n* o tamanho do cenário.

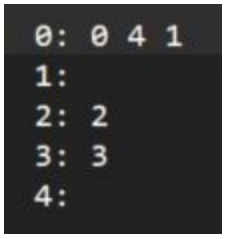
- _generateScenario: Faz operações básicas de leitura de arquivo e chama a função **createScenario**. Complexidade O(n).
- _moveOnto: Possui loops aninhados para encontrar os itens no cenário e posicioná-los no lugar certo. Além disso, chama as funções **sizeOfPile**, **pile** e **unpile**. Como estas não alteram a complexidade da função, por serem O(1), a função tem complexidade O(n²).
- _moveOver: No melhor caso (item *A* não possui blocos em cima, a função possui complexidade O(n). No pior caso, quando *A* possui itens em cima, a função possui complexidade O(n²).
- _pileOnto: A função possui loops aninhados para descobrir os elementos e modificá-los. Complexidade O(n²).
- _pileOver: No melhor caso, quando não há itens em cima de *A*, a função tem complexidade O(n). Já quando há itens em cima de *A*, a função possui complexidade O(n²).
- _executeCommands: A função possui um loop que roda até chegar no comando quit. Ela possui complexidade O(k), onde k é o número de comandos no arquivo de entrada.
- _writeOutput: Faz operações para escrever no arquivo de saída a configuração final do TAD Blocks. Possui loops aninhados para percorrer o TAD por completo. Complexidade O(n²).

4. Testes

Arquivo de entrada:



Arquivo de saída:



5. Conclusão

A implementação do trabalho ocorreu como esperado. As maiores dificuldades foram: abstrair os Tipos Abstratos de Dados para se adequarem ao domínio do mundo dos blocos, mantendo a fidelidade às operações possíveis; tratar as funções que poderiam ser feitas sobre os blocos para operarem de modo correto; trabalhar com ponteiros para alterar o cenário sempre que necessário e não alterá-lo quando não necessário.

6. Referências

https://stackoverflow.com/

Anexos

Código no GitHub:

https://github.com/lucasclopesr/tp1

Listagem dos programas:

- Blocks.h
- Cell.h
- Item.h
- Pile.h
- Blocks.c
- Pile.c
- TP1.c

```
#ifndef BLOCKS_H
#define BLOCKS_H

typedef struct{
     Pile *ground;
     int size;
} Blocks;

void createScenario(Blocks*, int);
int findBlock(Blocks, int);

#endif
```

```
#ifndef CELL_H
#define CELL_H

typedef struct Cell_str *Pointer;

typedef struct Cell_str{
     Item content;
     Pointer next;
} Cell;

#endif
```

```
#ifndef ITEM_H
#define ITEM_H

typedef int primaryKey;

typedef struct{
      primaryKey key;
} Item;

#endif
```

```
#ifndef PILE_H
#define PILE_H

typedef struct {
      Pointer bottom, top;
      int size;
} Pile;

void createEmptyPile(Pile*);
int isPileEmpty(const Pile*);
void pile(Item, Pile*);
int unpile(Pile*, Item*);
int sizeOfPile(const Pile*);

#endif
```

```
//Blocks.c
#include "../lib/Item.h"
#include "../lib/Blocks.h"
#include "../lib/Pile.h"

void createScenario(Blocks *scenario, int n){
     int i;
     Item deafultItem;

     scenario->ground = (Pile *) malloc(n*sizeof(Pile));

     for(i = 0; i < n; i++){
          createEmptyPile(&(scenario->ground[i]));
          deafultItem.key = i;

          pile(deafultItem, &(scenario->ground[i]));
     }

     scenario->size = n;

     //Loop to check if scenario was created correctly.
     /*for(i = 0; i < n; i++){
          printf("Bottom: %d\n",
scenario->ground[i].bottom->content.key);
          printf("Top: %d\n\n", scenario->ground[i].top->content.key);
     }*/
}

int findBlock(Blocks scenario, int key){
     int i, j;
     Pointer temporaryItem;


     for(i = 0; i < scenario.size; i++){
          if(isPileEmpty(&(scenario.ground[i])) == 0){
               temporaryItem = scenario.ground[i].top->next;
               for(j = 1; j < sizeOfPile(&(scenario.ground[i]));
j++){
                    if(temporaryItem->content.key == key){
```

```
                        return i;
                    }
                    temporaryItem = temporaryItem->next;
                }
            }
        }

        return -1;
    }
```

```
//Pile.c
#include "../lib/Item.h"
#include "../lib/Cell.h"
#include "../lib/Pile.h"
#include <stdlib.h>
#include <stdio.h>

void createEmptyPile(Pile *p){
     p->top = (Pointer) malloc(sizeof(Cell));
     p->bottom = p->top;
     p->top->next = NULL;
     p->size = 0;
}

int isPileEmpty(const Pile *p){
     return (p->top == p->bottom);
}

void pile(Item i, Pile *p){
     Pointer aux;
     aux = (Pointer) malloc(sizeof(Cell));

     p->top->content = i;
     aux->next = p->top;
     p->top = aux;
     p->size++;
}

int unpile(Pile *p, Item *i){
     Pointer aux;

     if(isPileEmpty(p)){
          return -1;
     }

     aux = p->top;
     p->top = aux->next;
     free(aux);
```

```
      p->size--;
      *i = p->top->content;
      return 0;
}

int sizeOfPile(const Pile *p){
      return p->size;
}
```

```
/* ************************************************
*   Autor: Lucas Caetano Lopes Rodrigues           *
*   Disciplina: AEDS 2                             *
*   Curso: Ciência da Computação                   *
*   Where there is ruin there is hope for a treasure.  *
************************************************** */

#include "Pile.c"
#include "Blocks.c"
#include <string.h>

FILE* _generateScenario(char *input, Blocks *scenario){
      FILE *inputFile = fopen(input, "r");
      int c;

      if(inputFile == NULL){
            //File not found
            printf("Arquivo não encontrado.");
      } else {
            fscanf(inputFile, "%d", &c);
            createScenario(scenario, c);
      }

      return inputFile;
}

void _moveOnto(Blocks *scenario, int a, int b){
      //Iterators
      int i; //Finds pile corresponding to a's index.
      int j; //Finds pile corresponding to b's index.
      int k; //Aux iterator to unpile and pile itens around.
      int unpiledIndex;

      int sizeOfAStack;
      int sizeOfBStack;

      Item temporaryUnpiledItem;

      for(i = 0; i < scenario->size; i++){
```

```
          if(scenario->ground[i].bottom->content.key == a){
               if(sizeOfPile(&(scenario->ground[i])) == 1){
                    //No blocks above a
                    for(j = 0; j < scenario->size; j++){
                         if(scenario->ground[j].bottom->content.key
== b){
                              if(sizeOfPile(&(scenario->ground[j]))
== 1){
                                   //No blocks above b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                   unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                              } else {
                                   //Some blocks above b
                                   //Unpile blocks above b,
returning them to original location
                                   sizeOfBStack =
sizeOfPile(&(scenario->ground[j]));
                                   for(k = 1; k < sizeOfBStack;
k++){

unpile(&(scenario->ground[j]), &temporaryUnpiledItem);
                                        unpiledIndex =
temporaryUnpiledItem.key;
                                        pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                   }

                                   //Move a onto b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                   unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                              }
                         }
                    }
               } else {
                    //Some blocks above a
                    for(j = 0; j < scenario->size; j++){
                         if(scenario->ground[j].bottom->content.key
== b){
```

```
                                    if(sizeOfPile(&(scenario->ground[j]))
== 1){
                                          //No blocks above b
                                          //Unpile blocks above a,
returning them to original location
                                          sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
                                          for(k = 1; k < sizeOfAStack;
k++){

unpile(&(scenario->ground[i]), &temporaryUnpiledItem);
                                                unpiledIndex =
temporaryUnpiledItem.key;
                                                pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                          }

                                          //Move a onto b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                          unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                                    } else {
                                          //Some blocks above b
                                          //Unpile blocks above b,
returning them to original location
                                          sizeOfBStack =
sizeOfPile(&(scenario->ground[j]));
                                          for(k = 1; k < sizeOfBStack;
k++){

unpile(&(scenario->ground[j]), &temporaryUnpiledItem);
                                                unpiledIndex =
temporaryUnpiledItem.key;
                                                pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                          }

                                          //Unpile blocks above a,
returning them to original location
                                          sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
```

```
                                        for(k = 1; k < sizeOfAStack;
k++){

unpile(&(scenario->ground[i]), &temporaryUnpiledItem);
                                             unpiledIndex =
temporaryUnpiledItem.key;
                                             pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                        }

                                        //Move a onto b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                        unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                                   }
                              }
                         }
                    }
               } else {
                    //Element is not on the "ground". Figure out if I have
to treat this.
               }
     }

     /*for(i = 0; i < scenario->size; i++){
          printf("Pilha %d: %d\n", i,
sizeOfPile(&(scenario->ground[i])));
     }
     printf("\n");*/
}

void _moveOver(Blocks *scenario, int a, int b){
     //Iterators
     int i; //Finds pile corresponding to a's index.
     int j; //Finds pile corresponding to b's index.
     int k; //Aux iterator to unpile and pile itens around.
     int pileOfB;
     int unpiledIndex;

     int sizeOfAStack;
     int sizeOfBStack;
```

```
      Item temporaryUnpiledItem;

      for(i = 0; i < scenario->size; i++){
            if(scenario->ground[i].bottom->content.key == a){
                  if(sizeOfPile(&(scenario->ground[i])) == 1){
                        //No blocks above a
                        pileOfB = findBlock(*scenario, b);

                        if(pileOfB != -1){
                              //Move a over b
                              pile(scenario->ground[i].bottom->content,
&(scenario->ground[pileOfB]));
                              unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                        }
                  } else {
                        //Some blocks above a
                        //Unpile blocks above a, returning them to
original location
                        sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
                        for(k = 1; k < sizeOfAStack; k++){
                              unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                              unpiledIndex = temporaryUnpiledItem.key;
                              pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                        }

                        pileOfB = findBlock(*scenario, b);

                        if(pileOfB != -1){
                              //Move a over b
                              pile(scenario->ground[i].bottom->content,
&(scenario->ground[pileOfB]));
                              unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                        }
                  }
            } else {
                  //Element is not on the "ground". Figure out if I have
to treat this.
            }
```

```
     }

     /*for(i = 0; i < scenario->size; i++){
          printf("Pilha %d: %d\n", i,
sizeOfPile(&(scenario->ground[i])));
     }
     printf("\n");*/
}

void _pileOnto(Blocks *scenario, int a, int b){
     //Iterators
     int i; //Finds pile corresponding to a's index.
     int j; //Finds pile corresponding to b's index.
     int k; //Aux iterator to unpile and pile itens around.
     int unpiledIndex;

     int sizeOfAStack;
     int sizeOfBStack;
     int sizeOfTemporaryStack;

     Item temporaryUnpiledItem;
     Pile temporaryPile;

     createEmptyPile(&temporaryPile);

     for(i = 0; i < scenario->size; i++){
          if(scenario->ground[i].bottom->content.key == a){
               if(sizeOfPile(&(scenario->ground[i])) == 1){
                    //No blocks above a
                    for(j = 0; j < scenario->size; j++){
                         if(scenario->ground[j].bottom->content.key
== b){
                              if(sizeOfPile(&(scenario->ground[j]))
== 1){
                                   //No blocks above b
                                   //Pile a onto b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                   unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                              } else {
                                   //Some blocks above b
```

```
                                        //Unpile blocks above b,
returning them to original location
                                        for(k = 1; k <
sizeOfPile(&(scenario->ground[j])); k++){

unpile(&(scenario->ground[j]), &temporaryUnpiledItem);
                                             unpiledIndex =
temporaryUnpiledItem.key;
                                             pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                        }

                                        //Pile a onto b

pile(scenario->ground[i].bottom->content, &(scenario->ground[j]));
                                        unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                                   }
                              }
                         }
                    } else {
                         //Some blocks above a
                         for(j = 0; j < scenario->size; j++){
                              if(scenario->ground[j].bottom->content.key
== b){
                                   if(sizeOfPile(&(scenario->ground[j]))
== 1){
                                        //No blocks above b
                                        //Unpile blocks above a
                                        sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
                                        for(k = 0; k < sizeOfAStack;
k++){

unpile(&(scenario->ground[i]), &temporaryUnpiledItem);
                                             unpiledIndex =
temporaryUnpiledItem.key;
                                             pile(temporaryUnpiledItem,
&(temporaryPile));
                                        }

                                        //Pile blocks above a in right
order onto b
```

```
                                        sizeOfTemporaryStack =
sizeOfPile(&temporaryPile);
                                        for(k = 0; k <
sizeOfTemporaryStack; k++){
                                             unpile(&temporaryPile,
&temporaryUnpiledItem);
                                             pile(temporaryUnpiledItem,
&(scenario->ground[j]));
                                        }
                                   } else {
                                        //Some blocks above b
                                        //Unpile blocks above b,
returning them to original location
                                        sizeOfBStack =
sizeOfPile(&(scenario->ground[j]));
                                        for(k = 1; k < sizeOfBStack;
k++){

unpile(&(scenario->ground[j]), &temporaryUnpiledItem);
                                             unpiledIndex =
temporaryUnpiledItem.key;
                                             pile(temporaryUnpiledItem,
&(scenario->ground[unpiledIndex]));
                                        }

                                        //Unpile blocks above a
                                        sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
                                        for(k = 0; k < sizeOfAStack;
k++){

unpile(&(scenario->ground[i]), &temporaryUnpiledItem);
                                             unpiledIndex =
temporaryUnpiledItem.key;
                                             pile(temporaryUnpiledItem,
&(temporaryPile));
                                        }

                                        //Pile blocks above a in right
order onto b
                                        sizeOfTemporaryStack =
sizeOfPile(&temporaryPile);
```

```
                                        for(k = 0; k <
sizeOfTemporaryStack; k++){
                                             unpile(&temporaryPile,
&temporaryUnpiledItem);
                                             pile(temporaryUnpiledItem,
&(scenario->ground[j]));
                                        }
                                   }
                              }
                         }
                    }
          } else {
               //Element is not on the "ground". Figure out if I have
to treat this.
          }
     }

     /*for(i = 0; i < scenario->size; i++){
          printf("Pilha %d: %d\n", i,
sizeOfPile(&(scenario->ground[i])));
     }
     printf("\n");*/
}

void _pileOver(Blocks *scenario, int a, int b){
     //Iterators
     int i; //Finds pile corresponding to a's index.
     int j; //Finds pile corresponding to b's index.
     int k; //Aux iterator to unpile and pile itens around.
     int unpiledIndex;
     int pileOfB;

     int sizeOfAStack;
     int sizeOfBStack;
     int sizeOfTemporaryStack;

     Item temporaryUnpiledItem;
     Pile temporaryPile;

     createEmptyPile(&temporaryPile);

     for(i = 0; i < scenario->size; i++){
          if(scenario->ground[i].bottom->content.key == a){
```

```
                if(sizeOfPile(&(scenario->ground[i])) == 1){
                      //No blocks above a
                      pileOfB = findBlock(*scenario, b);

                      if(pileOfB != -1){
                            //Pile a over b
                            pile(scenario->ground[i].bottom->content,
&(scenario->ground[pileOfB]));
                            unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                      }
                } else {
                      //Some blocks above a
                      //Unpile blocks above a
                      sizeOfAStack =
sizeOfPile(&(scenario->ground[i]));
                      for(k = 0; k < sizeOfAStack; k++){
                            unpile(&(scenario->ground[i]),
&temporaryUnpiledItem);
                            unpiledIndex = temporaryUnpiledItem.key;
                            pile(temporaryUnpiledItem,
&(temporaryPile));
                      }

                      pileOfB = findBlock(*scenario, b);

                      if(pileOfB != -1){
                            //Pile blocks above a in right order onto b
                            sizeOfTemporaryStack =
sizeOfPile(&temporaryPile);
                            for(k = 0; k < sizeOfTemporaryStack; k++){
                                  unpile(&temporaryPile,
&temporaryUnpiledItem);
                                  pile(temporaryUnpiledItem,
&(scenario->ground[pileOfB]));
                            }
                      }
                }
           } else {
                //Element is not on the "ground". Figure out if I have
to treat this.
           }
     }
```

```
	/*for(i = 0; i < scenario->size; i++){
		printf("Pilha %d: %d\n", i,
sizeOfPile(&(scenario->ground[i])));
	}
	printf("\n");*/
}

void _executeCommands(FILE *input, Blocks *scenario){
	char cmd1[5], cmd2[5];
	int item1, item2;
	int scanReturn;

	while(scanReturn != 1){
		scanReturn = fscanf(input, "%s %d %s %d", cmd1, &item1,
cmd2, &item2);
		if(strcmp(cmd1, "quit") != 0){
			if(strcmp(cmd1, "move") == 0){
				if(strcmp(cmd2, "onto") == 0){
					//Call moveOnto
					_moveOnto(scenario, item1, item2);
				}
				if(strcmp(cmd2, "over") == 0){
					//Call moveOver
					_moveOver(scenario, item1, item2);
				}
			}

			if(strcmp(cmd1, "pile") == 0){
				if(strcmp(cmd2, "onto") == 0){
					//Call pileOnto
					_pileOnto(scenario, item1, item2);
				}
				if(strcmp(cmd2, "over") == 0){
					//Call pileOver
					_pileOver(scenario, item1, item2);
				}
			}
		}
	}
}

void _writeOutput(Blocks scenario, char *output){
```

```
	FILE *outputFile = fopen(output, "w");

	int i;
	int j;
	int k;
	int sizeOfStack;
	int sizeOfTemporaryStack;

	Item temporaryUnpiledItem;
	Pile temporaryPile;

	createEmptyPile(&temporaryPile);

	for(i = 0; i < scenario.size; i++){
		fprintf(outputFile, "%d: ", i);
		sizeOfStack = sizeOfPile(&(scenario.ground[i]));

		for(j = 0; j < sizeOfStack; j++){
			unpile(&(scenario.ground[i]), &temporaryUnpiledItem);
			pile(temporaryUnpiledItem, &(temporaryPile));
		}

		sizeOfTemporaryStack = sizeOfPile(&(temporaryPile));
		for(k = 0; k < sizeOfTemporaryStack; k++){
			unpile(&(temporaryPile), &temporaryUnpiledItem);
			fprintf(outputFile, "%d ", temporaryUnpiledItem);
		}

		fprintf(outputFile, "\n");
	}
}

void main(int argc, char **argv){
	Blocks scenario;
	FILE *input;

	int i;

	if(argc == 3){
		input = _generateScenario(argv[1], &scenario);

		if(input != NULL){
			_executeCommands(input, &scenario);
```

```
                _writeOutput(scenario, argv[2]);
            }
        } else {
            printf("Numero incorreto de parametros.\n");
        }
}
```