Relatório do Trabalho Prático de AEDII



Trabalho realizado por:

Rúben Gomes nº21118

Conteúdo

[Introdução 3](#_Toc71309316)

[Escolha das estruturas de dados 4](#_Toc71309317)

[Leitura e inserção dos dados nas listas 5](#_Toc71309318)

[Exercícios propostos 8](#_Toc71309319)

[Alínea 1 8](#_Toc71309320)

[Alínea 2: 9](#_Toc71309321)

[Alínea 3 10](#_Toc71309322)

[Alínea 4 11](#_Toc71309323)

[Alínea 5 11](#_Toc71309324)

[Alínea 6: 11](#_Toc71309325)

[Alínea 7: 12](#_Toc71309326)

[Opções extra 15](#_Toc71309327)

[Conclusão 16](#_Toc71309328)

# Introdução

Como trabalho prático este semestre foi pedido a realização de um trabalho na linguagem C. O objetivo deste trabalho era a gestão de ficheiros que continham várias informações acercas de conjuntos da marca LEGO e as suas peças, podendo desta forma fornecer ao utilizador várias informações sobre estes conjuntos, desde quais as peças necessárias para construir certos conjuntos, a quais os conjuntos que podemos construir com as peças existentes, etc.

No início do trabalho juntei-me a 2 colegas meus e iniciamos o trabalho em grupo (só tendo depois reparado que eram grupos de no máximo 2 elementos). Como foi sugerido pelo professor Alberto, eu continuei a fazer o trabalho individualmente e os meus colegas ficaram num grupo de 2, tendo ficado tanto eu como eles com uma parte do trabalho (leitura dos ficheiros e inserção nas listas e as alíneas 1 e 3 do enunciado) feita da mesma maneira como tínhamos feito quando estávamos em grupo.

Nas seguintes páginas deste relatório irei relatar a forma como abordei o problema em mão e quais os obstáculos que fui encontrando ao longo do trabalho e como os resolvi.

# Escolha das estruturas de dados

Tendo sido o início deste semestre focado em Listas na linguagem C, e dado o problema em mão, decidi que a melhor escolha para abordar o armazenamento destas listas são as listas duplamente ligadas.

Comecei então por criar as estruturas necessárias para todas as listas que previ serem necessárias para abordar este problema. Em baixo tenho um exemplo das três estruturas criadas sendo que foi criada uma para cada ficheiro:

//Estrutura para a lista do ficheiro sets.tsv

typedef struct \_sets

{

    char set\_num[20];

    char name[100];

    int year;

    char theme[100];

    struct \_sets\*next;

    struct \_sets\*previous;

}Sets;

(linhas 10 a 19 do código do projeto)

Como exemplificado em cima, decidi escolher as listas duplamente ligadas pois me permitem mover facilmente pelas listas com os apontadores next e previous.

Criei também uma estrutura extra para guardar alguns dados ao longo do trabalho que fossem necessários guardar algo que eu vou exemplificar mais à frente.

typedef struct

{

    char str[100];

    bool possible;

}Saver;

# Leitura e inserção dos dados nas listas

A abordagem para a leitura dos ficheiros foi uma abordagem básica, tendo sido praticamente igual para os 3 ficheiros, como exemplificado em baixo:

Sets\* read\_Sets(Sets\* lst)

{

    FILE\* sets=fopen("sets.tsv", "r");

    int i=0;

    char str[200], fLine[100];

    Sets v;

    if (sets==NULL)

    {

        perror("Unable to open the file requested!");

    }

    else

    {

        while (!feof(sets))

        {

            if (i==0)

            {

                fgets(fLine, sizeof(fLine), sets);

            }

            else

            {

                fflush(stdin);

                fgets(str, sizeof(str), sets);

                sscanf(str, "%s\t%[^\t]\t%d\t%[^\t]", v.set\_num, v.name, &v.year, v.theme);

                lst = head\_insert\_sets(lst, v);

            }

            i++;

        }

    }

    return lst;

}

(linhas 192 a 222)

Na inserção dos ficheiros decidi fazer a inserção à cabeça (sendo a mais simples de fazer e mais rápida) normal em 2 dos ficheiros os ficheiros “parts.tsv” e “parts\_sets.tsv” como exemplificado em baixo:

PartsSets\* head\_insert\_parts\_sets(PartsSets\* lst, PartsSets reg)

{

    PartsSets\* new= (PartsSets\*)malloc(sizeof(PartsSets));

    strncpy(new->set\_num,reg.set\_num,100);

    new->quantity=reg.quantity;

    strncpy(new->part\_num,reg.part\_num,100);

    new->next=lst;

    if (new->next)

    {

        new->next->previous=new;

    }

    new->previous=NULL;

    return new;

}

(linhas 52 a 67)

No ficheiro “sets.tsv” fiz uma inserção ordenada crescente pelo ano dos conjuntos que me serviu de ajuda para a alínea 1 onde tinha de indicar os conjuntos de cada tema ordenados pelo ano, como exemplificado em baixo:

Sets\* head\_insert\_sets(Sets\* lst, Sets reg)

{

    Sets \*new= (Sets\*)malloc(sizeof(Sets));

    strncpy(new->set\_num,reg.set\_num,20);

    strncpy(new->name,reg.name, 100);

    new->year=reg.year;

    strncpy(new->theme, reg.theme, 100);

    new->next=new->previous=NULL;

    if (!lst || lst->year > reg.year)

    {

        new->next=lst;

        lst=new;

        if (lst->next)

        {

            lst->next->previous=lst;

        }

    }

    else

    {

        Sets\* aux= lst;

        for (;aux->next && aux->next->year < reg.year; aux= aux->next)

        {

        }

        new->next=aux->next;

        new->previous=aux;

        aux->next=new;

        if (new->next)

        {

            new->next->previous=new;

        }

    }

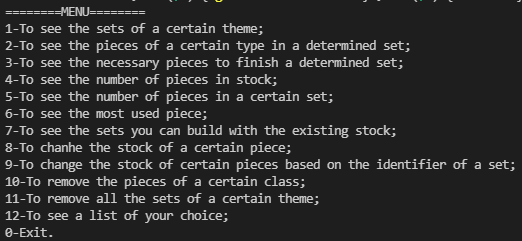
    return lst;

}

(linhas 88 a 123)

# Exercícios propostos

Para gerir todas as funções e procedimentos necessários à utilização do programa, criei um Menu de opções, como se pode ver a seguir:



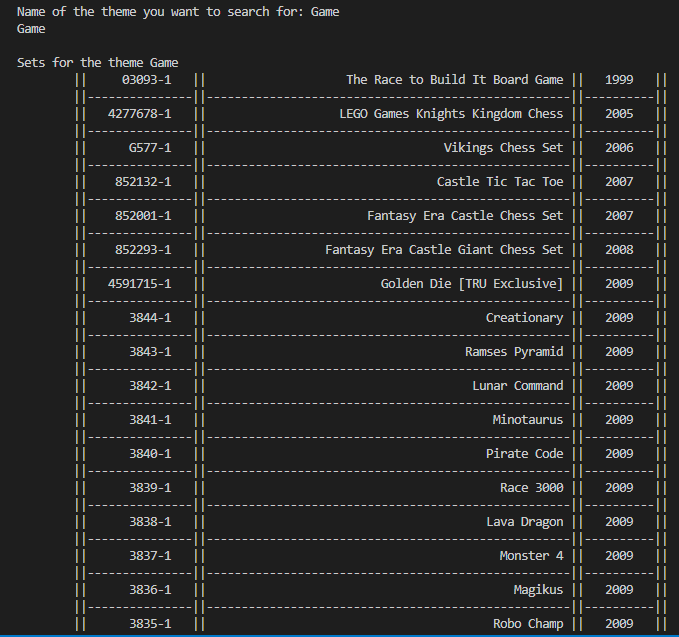
## Alínea 1

//1º: Procedimento para determinar os conjuntos de cada tema

void themeSets(Sets\*lst)

Na alínea 1 apenas percorri a lista do ficheiro “sets.tsv” e verifiquei quais eram os conjuntos do tema que pedi por input ao utilizador e listei-os.

Teste:



## Alínea 2:

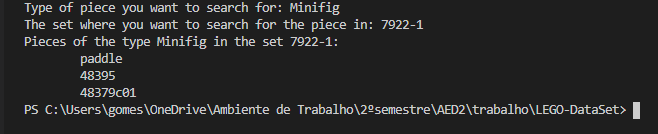
//2º-Procedimento para determinar as peças de determinado tipo em determinado conjunto

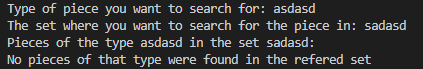
void searchPieceSet(Parts\* lstParts, PartsSets\* lstPartsSets)

Na alínea 2 foi onde dei uso pela 1º vez à estrutura extra que referi na página 4. Precisei desta estrutura para criar um array de strings, onde podia guardar todas as peças da classe que o utilizador pedia, quando percorria a lista do ficheiro “parts.tsv”.

De seguida percorri a lista do ficheiro “parts\_sets.tsv” e procurei o conjunto que foi dado por input do utilizador. Para cada linha do conjunto selecionado percorri o array de strings onde guardei as peças anteriormente e verifiquei que peças correspondiam às do array e listei-as.

Testes:





## Alínea 3

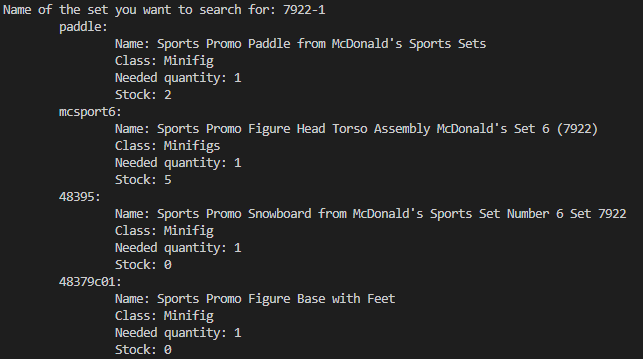
//3º: Procedimento para determinar as peças necessárias para acabar um determinado conjunto

void necessaryPieces(PartsSets\*lstPartsSets, Parts\*lstParts)

Na alínea 3, houve alguma confusão no meu raciocínio, sobre como procurar o resto da informação das peças na lista do ficheiro “Parts.tsv”, mas fiz o procedimento da seguinte maneira.

Percorri a lista do ficheiro “Parts\_Sets.tsv” e verifiquei todos os sets que eram iguais ao input do utilizador e listei as peças que estavam nesses sets. Depois percebi que para mostrar o resto das informações basta percorrer a lista do ficheiro “Parts.tsv” dentro deste mesmo ciclo e procurar as restantes informações das peças, usando a peça que estava a ser listada.

Teste:



## Alínea 4

//4º: Função para devolver o nº de peças em stock

int piecesInStock(Parts\* lst)

A alínea 4 foi simples, apenas percorri a lista do ficheiro “Parts.tsv” e somei o stock de todas as peças e devolvi-o.

Teste:



## Alínea 5

//5ª: Função para devolver o numero de peças num determinado num determinado conjunto

int piecesInSet(PartsSets\* lst)

Nesta alínea a função também foi simples, sendo que só pedi o set para analisar ao utilizador e percorrendo a lista do ficheiro “Parts\_Sets.tsv” somei a quantidade das peças do set pedido.

Teste:



## Alínea 6:

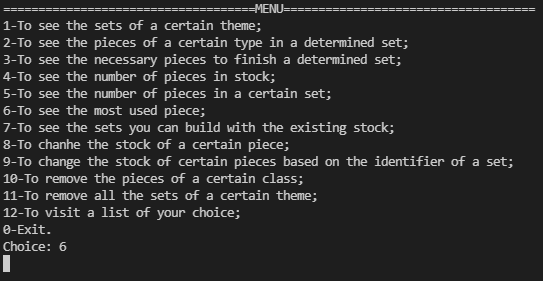
//6º: Procedimento para determinar a peça mais usada em conjuntos diferentes

void most\_used(PartsSets\* lstPartSets, Parts\* lstParts)

A alínea 6 foi onde comecei a ter dificuldade com o tempo de execução, algo que não consegui resolver. Apesar o código usado para este procedimento ser funcional, não parece ser o mais eficaz para o que preciso. O facto de ter de percorrer a lista do ficheiro “Parts\_Sets.tsv” de 580 mil linhas 25 mil vezes, foi o que me impediu de ter um tempo de execução tão pequeno quanto o necessário para o procedimento.

O tempo de execução acabou por ser de 20 a 30 minutos, sendo que a ideia do procedimento era correr a lista do ficheiro “Parts.tsv” e para cada peça contar as vezes que aparecem na lista do ficheiro “Parts\_Sets.tsv”.

Testes:





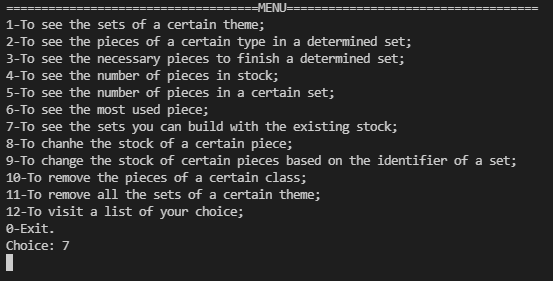
## Alínea 7:

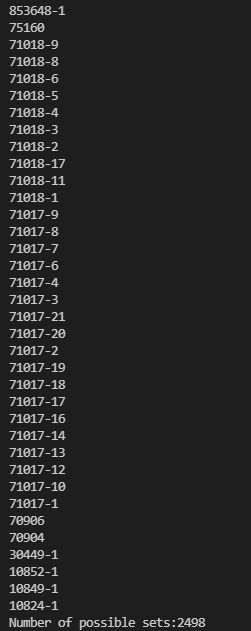
//7º: Procedimento para determinar os conjuntos que se podem construir com o stock existente

void BuildSet(PartsSets\* lstPartsSets, Parts\* lstParts, Sets\* lstSets)

A alínea 7 não fugiu à regra da alínea 6, o tempo de execução também foi à volta de 20 a 30 minutos, tendo corrido as mesmas listas que corri anteriormente. O procedimento foi percorrer todas as peças existentes na lista do ficheiro “Parts\_Sets.tsv” e verificar para cada uma se o seu stock na lista do ficheiro “Parts.tsv” é maior que a quantidade, se não for corre-se a lista do ficheiro “Sets.tsv” e procura-se o conjunto em causa, tornando a variável “possible” dessa posição da lista falsa.

Testes:





# Opções extra

Para além das alíneas anteriores foi-nos pedido para adicionar mais algumas opções para remover peças e conjuntos e gestão de stock.

Todas estas operações foram feitas de um modo simples, sendo que as de remoção foram exatamente iguais àquilo que fizemos nas aulas. Na gestão do stock foram também 2 operações simples em que se procurava a peça ou o conjunto em questão e se adicionava ao stock com a escolha do utilizador.

//Função para mudar o stock de uma determinada peça

Parts\* changeStock(Parts\*lst)

//Funçao para mudar o stock das peças de acordo com o conjunto escolhido

Parts\* changeStockBySet(Parts\* lstParts, PartsSets\* lstPartsSets)

//Função para remover todas as peças de uma determinada classe

Parts\* removeParts(Parts \* lst)

//Função para remover todos os sets de um determinado tema

Sets\* removeSets(Sets\* lst)

## Conclusão

Para concluir este relatório, devo dizer que foi um trabalho desafiante e extensivo, que me ajudou a desenvolver muitos novos conhecimentos de programação e consolidar aquilo que já tínhamos aprendido durante as aulas. Creio que apesar de não ter conseguido fazer tudo da maneira que queria, acabei por conseguir fazer o trabalho da melhor maneira que podia com as minhas limitadas capacidades de programação.