ESTGOH
Escola Superior de Tecnologia e Gestão de Oliveira do Hospital
Instituto Politécnico de Coimbra

Licenciatura em Engenharia Informática

Estruturas de dados

2019/2020

Relatório

Ficha de Listas Ligadas

David Ferreira nrº 2019141281

António Pereira nrº 2019141051

Índice

1 Introdução

Neste relatório iremos explicar a resolução da ficha de trabalho número 5, relativamente à matéria de listas ligadas, que é um tipo de armazenamento volátil na memória. Iremos tentar fazer uma breve explicação de cada exercício proposto pela professora e tentar explicar como se adequa com as listas ligadas com prints de código e exemplos de output do mesmo, falando sobre as nossas maiores dificuldades ao longo da resolução da ficha.

2 Resolução dos exercícios

2.1 Exercício 4 ficha de Structs (reformulado)

Neste exercício, teríamos que ir ver à ficha anterior o exercício 4, que constava numa pequena empresa familiar que pretendia organizar os dados dos seus funcionários. Neste exercício 1 da ficha de listas ligadas, em vez de definirmos uma tabela, definimos um pointer para n funcionários indicados pelo utilizador.

Assim, a estrutura ficou exatamente a mesma, a diferença é que, em cada subprograma, o argumento seria um pointer do tipo struct, como vamos ver nos próximos programas.

Nota: A resolução deste exercício é bastante parecida com a da ficha anterior.

i) Introduzir os dados de um funcionário na lista de funcionários

Para o utilizador conseguir introduzir os dados dos funcionários, tivemos que utilizar vários scanf e gets (para strings). Assim, o código ficou deste modo:

```
void IntroduzirDados(func * pt_func)
{
        printf("\nNome: ");
        fflush(stdin);
        gets(pt_func->nome);
        printf("\nNumero: ");
        scanf("%d", &pt_func->numero);
        printf("\nTarefa: ");
        fflush(stdin);
        gets(pt_func->tarefa);
        printf("\nSalario: ");
        scanf("%f", &pt_func->salario);
}
```

Dá para reparar que, ao contrário do exercício 4 da ficha 4, esta função não contém nenhum ciclo (algo que vai ser explicado mais para a frente), aliás, nenhuma das funções explicadas neste exercício terão ciclos. Além disso, dá para notar que no argumento temos um pointer do tipo struct, em vez da variável normal do tipo struct.

Exemplo de output (usado para os restantes exemplos):

```
Nome:Nuno Mendes

Numero:111

Tarefa:Limpezas

Salario:1000

Nome:Jose Soares

Numero:222

Tarefa:Web Dev

Salario:2000
```

ii) Listar todos os funcionários e respetivos dados

Mais uma vez, esta função é muito parecida com a do exercício 4, que consiste em printfs:

```
void ApresentarDados(func * pt_func)
{
        printf("Nome: %s\n", pt_func->nome);
        printf("Numero: %d\n", pt_func->numero);
        printf("Tarefa: %s\n", pt_func->tarefa);
        printf("Salario: %.1f\n\n", pt_func->salario);
}
```

Output:

```
 Nome: Nuno Mendes
Numero: 111
Tarefa: Limpezas
Salario: 1000.0

Nome: Jose Soares
Numero: 222
Tarefa: Web Dev
Salario: 2000.0
```

iii) Listar os funcionários com salário superior a 500€

Para esta função, para conseguirmos provar que funciona, acrescentámos um terceiro funcionário:

```
 Nome: Nuno Mendes
Numero: 111
Tarefa: Limpezas
Salario: 1000.0

Nome: Jose Soares
Numero: 222
Tarefa: Web Dev
Salario: 2000.0

Nome: Rui Mata
Numero: 333
Tarefa: RP
Salario: 499.0
```

Para esta função, basta utilizar um if para verificar se o campo do salário do funcionário é superior a 500. Caso seja, então irá apresentar esse funcionário:

```c
void ListarAcima500(func * pt_func)
{
        if(pt_func->salario > 500)
        {
            printf("Nome: %s\n", pt_func->nome);
            printf("Numero: %d\n", pt_func->numero);
            printf("Tarefa: %s\n", pt_func->tarefa);
            printf("Salario: %.1f\n\n", pt_func->salario);
        }
}
```

Exemplo de output:

```
Funcionarios com salario superior a 500 euros:

Nome: Nuno Mendes
Numero: 111
Tarefa: Limpezas
Salario: 1000.0

Nome: Jose Soares
Numero: 222
Tarefa: Web Dev
Salario: 2000.0
```

iv) Procurar e apresentar os dados de um funcionário utilizando o seu nome

Em semelhança à função anterior, basta utilizar um if e verificar se a strcmp do nome a procurar com o nome na lista é igual. Caso seja, o programa irá apresentar os dados todos do funcionário encontrado.

Sendo assim, a função ficou com o seguinte aspeto:

```c
void ProcurarFuncionario(func * pt_func, char procurar[])
{
        if(strcmp(pt_func->nome, procurar)==0)
        {
            printf("Nome: %s\n", pt_func->nome);
            printf("Numero: %d\n", pt_func->numero);
            printf("Tarefa: %s\n", pt_func->tarefa);
            printf("Salario: %.1f\n\n", pt_func->salario);
        }
}
```

Exemplo de output:

```
Introduza o nome do funcionario que deseja encontrar:Jose Soares
 Nome: Jose Soares
Numero: 222
Tarefa: Web Dev
Salario: 2000.0
```

v) Atualizar os dados de um funcionário utilizando o seu número

Neste subprograma, à semelhança dos anteriores, bastou utilizar um if para verificar se o número que o utilizador introduziu existe na lista e, caso exista, o utlizador insere todos os dados de novo desse funcionário:

```
void AtualizarDados(func * pt_func, int numerofunc)
{

        if (numerofunc == pt_func->numero)
        {
            printf("\nInsira de novo os dados: ");
            printf("\nNome: ");
            fflush(stdin);
            gets(pt_func->nome);
            printf("\nNumero: ");
            scanf("%d", &pt_func->numero);
            printf("\nTarefa: ");
            fflush(stdin);
            gets(pt_func->tarefa);
            printf("\nSalario: ");
            scanf("%f", &pt_func->salario);
        }
```

Exemplo de output:

```
Insira o numero do funcionario cujos dados quer atualizar:333

Insira de novo os dados:
Nome:Francisca Ferreira

Numero:333

Tarefa:Java Dev

Salario:2500
```

Funcionários após a alteração:

```
 Nome: Nuno Mendes
Numero: 111
Tarefa: Limpezas
Salario: 1000.0

Nome: Jose Soares
Numero: 222
Tarefa: Web Dev
Salario: 2000.0

Nome: Francisca Ferreira
Numero: 333
Tarefa: Java Dev
Salario: 2500.0
```

Agora que as funções estão todas explicadas, está na hora de explicar como é que foram chamadas no main. Para conseguirem funcionar corretamente, tiveram que ser chamadas da seguinte maneira:

```
void main()
{
    int nmrfunc, numerofunc;
    func * pt_func;
    char procurar[100];

    printf("Introduza o numero de funcionarios: ");
    scanf("%d", &nmrfunc);

    pt_func = (func *)malloc(nmrfunc*sizeof(func));

    for(int i=0;i<nmrfunc;i++)
        IntroduzirDados(&pt_func[i]);

    for(int i=0;i<nmrfunc;i++)
        ApresentarDados(&pt_func[i]);

    printf("\nFuncionarios com salario superior a 500 euros: \n\n");
    for(int i=0;i<nmrfunc;i++)
        ListarAcima500(&pt_func[i]);

    printf("\nIntroduza o nome do funcionario que deseja encontrar: ");
    fflush(stdin);
    gets(procurar);
    for(int i=0;i<nmrfunc;i++)
        ProcurarFuncionario(&pt_func[i], procurar);
```

```
printf("\nInsira o numero do funcionario cujos dados quer atualizar: ");
scanf("%d", &numerofunc);
    for(int i=0;i<nmrfunc;i++)
        AtualizarDados(&pt_func[i], numerofunc);
    for(int i=0;i<nmrfunc;i++)
        ApresentarDados(&pt_func[i]);
    free(pt_func);
}
```

Como se pode notar, os argumentos das funções, no main, têm o endereço de memória estrutura “pt_func”. Contrariamente ao exercício 4 da ficha 4, os ciclos neste exercício eram feitos no main. Além disso, como tivemos que alocar memória para arranjar espaço para o “pt_func”, tivemos que utilizar a função malloc, que serve precisamente para isso. No fim do programa, para libertar o espaço ocupado por esse malloc, tivemos que utilizar a função free.

2.2 Escreva subprogramas que desempenhem cada uma das seguintes tarefas:

Os exercícios a, b, c e d já tinham os respetivos subprogramas criados pela professora, portanto vamos apenas mostrar exemplos de output e explicar como funcionam.

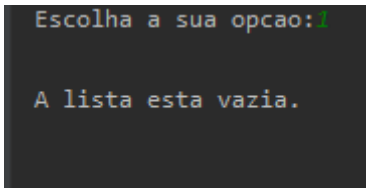
a) Criar uma lista vazia

Para criar uma lista vazia, bastava criar uma variável do tipo plista e igualá-la a NULL. A função não tem argumentos, pois no main foi chamada da seguinte forma criando 2 listas:

```
void main() {
    float x, v, procura;
    int n, opcao;
    char verifica;
    plista a = CriaLista();
    plista b = CriaLista();
```

b) Indicar se uma lista é vazia

Para esta função, bastou fazer um if e retornar qualquer coisa dependendo da lista estar vazia ou não. No nosso caso, alterámos ligeiramente o subprograma. Caso a lista esteja vazia, retorna o char 'y', caso contrário, retorna 'n'. Exemplo de output dessa função para uma lista vazia:

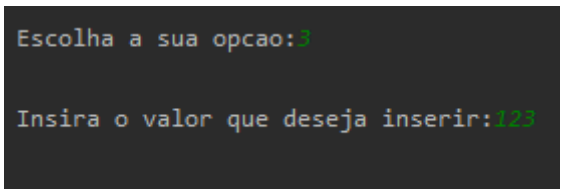


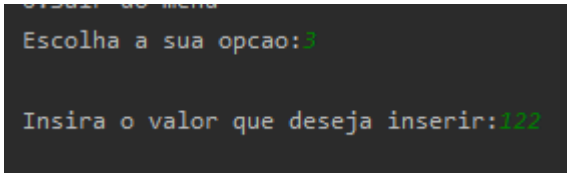
c) Mostrar uma lista

Esta função permite ao utilizador ver a lista que pretende, precisando apenas de mudar o argumento na chamada no main (caso tenha mais que uma lista criada). Uma função muito útil pois permite-nos, no fim de alguma alteração à lista, verificar se correu tudo bem e se está tudo em ordem.

d) Juntar um elemento no início da lista

Com este subprograma, temos a capacidade de inserir um elemento no início da lista pretendida. É bastante simples. Simplesmente criamos um nó que aponte para o início da lista e inserimos o valor que pretendemos. Um exemplo de output é o seguinte:





Tendo em conta que a ordem de inserção é a que está utilizando a função MostraLista conseguimos ver como ficou a nossa lista:

```
Escolha a sua opcao:2

[122.0, 123.0, ]
```

e) Inserir um elemento x no fim de uma lista

Para inserir um elemento no fim da lista, a lógica é muito semelhante à de inserir no início. Simplesmente temos que percorrer a lista toda até encontrarmos o último elemento, igualar esse último elemento ao que nós queremos colocar lá e passar o valor do último elemento para o lado esquerdo. Em código, é algo deste género:

```c
//alinea e
void JuntaFim(float x, plista * lst)
{
    plista no = (plista) malloc(sizeof(listano));    plista paux = *lst;
    if (no == NULL)
        printf("\nErro, nao ha memoria disponivel.");
    else {
        no->valor = x;
        no->prox = NULL;
        if (*lst == NULL) {
            *lst = no;
        } else {
            while (paux->prox != NULL) {
                paux = paux->prox;
            }
            paux->prox = no;
        }
    }
}
```

Temos que pensar em todas as alternativas. Ou seja, caso a lista esteja vazia, caso tenha apenas um elemento, etc. Caso a lista esteja vazia, o primeiro elemento será, logicamente, também o último. Assim, nesse caso basta igualar o nó ao primeiro elemento da lista e fazer com que o ponteiro passe a apontar para NULL, para definitivamente ser o último elemento. Caso a lista não esteja vazia, temos que percorrer a mesma até encontrarmos o último elemento e, quando encontrado, fazer o que fizemos em cima para juntar no início só que com uma particularidade: o ponteiro terá que apontar para NULL, para o elemento ser, sem dúvida, o último. Para evitar perder a lista ou estragar a mesma, criámos um ponteiro auxiliar que aponta para o primeiro elemento da lista. Assim, minimizamos riscos com a mesma eficácia.

Exemplo de output (seguindo o exemplo de output do exercício anterior):

```
Escolha a sua opcao:4

Insira o valor que deseja inserir no fim:121
```

```
Escolha a sua opcao:2

[122.0, 123.0, 121.0, ]
```

f) Remover o primeiro elemento de uma lista

Para conseguirmos remover qualquer elemento de uma lista, temos que usar a função free. Além disso, temos que também usar um ponteiro auxiliar para que não corramos, de novo, o risco de perder a lista. Assim, para remover o primeiro elemento de uma lista, basta apontar com o ponteiro auxiliar para o início da mesma e usar a função free:

```
void RemovePrimeiro(plista * lst)
{
    plista paux = *lst;
    *lst = (*lst)->prox;
    free(paux);
}
```

Output:

```
Escolha a sua opcao:2

[123.0, 121.0, ]
```

g) Remover o último elemento de uma lista

Em semelhança à função para acrescentar um elemento à lista, esta função também terá que percorrer a lista toda até chegar ao último elemento e, aí, utilizar a função free de novo. Desta vez, tivemos que utilizar dois ponteiros auxiliares:

```
void RemoveUltimo(plista * lst)
{
    plista paux = *lst;
    plista aux = NULL;
    while(paux->prox!=NULL)
    {
        aux = paux;
        paux = paux->prox;
    }
    free(paux);
    if(aux!=NULL)
        aux->prox = NULL;
    else
        *lst = NULL;
}
```

O ponteiro paux aponta para o início da lista, enquanto o ponteiro aux é uma lista vazia. Assim, caso a lista não esteja vazia, o ponteiro aux irá andar “atrás” do ponteiro paux, de modo a que, quando é encontrado o último elemento, se possa eliminar o paux e, caso a lista tenha mais que um elemento, o aux passa a apontar para NULL, ou seja, passa a ser o último elemento, caso só tenha um elemento, a lista passa a ser vazia.

Output:

```
Escolha a sua opcao:2

[123.0, ]
```

h) Remover o n-ésimo elemento de uma lista

Para podermos remover o elemento que está na posição escolhida pelo utilizador, teremos que percorrer o ciclo o número de vezes do número que o utilizador inserir. Ou seja, supondo que o utilizador decide eliminar o elemento da posição 3. A função terá que percorrer 2 ponteiros até chegar ao terceiro. Para o programa ter noção de como vai a contagem de posições, tivemos que utilizar um count. Basicamente, cada vez que ele entra no ciclo, o count é incrementado e no while verifica se já é igual ao número que o utilizador inseriu ou não. Depois é apenas usar a função free:

```
void RemoveNelemento(plista * lst, int n)
{
    plista paux = *lst;
    plista aux = NULL;
    int count = 1;
    while(paux->prox != NULL && count != n )
    {
        count++;
        aux = paux;
        paux = paux->prox;
    }
    if(count<n)
        printf("\nA lista tem apenas %d elemento(s)", count);
    if(aux!=NULL)
        aux->prox = paux->prox;
    else
        *lst = (*lst)->prox;
    free(paux);
}
```

Exemplo de output:

Antes:

```
Escolha a sua opcao:2

[122.0, 123.0, ]
```

Depois (removendo a segunda posição):

```
Escolha a sua opcao:2

[122.0, ]
```

i) Remover a primeira ocorrência do elemento x de uma lista

Esta função é muito semelhante à de cima, mas simplesmente em vez de termos um count, verificamos se o lst->valor é igual ao elemento que desejamos remover:

```
void RemovePrimeiraOcorrencia(plista * lst, float v)
{
    plista paux = *lst;
    plista aux = NULL;
    while(paux->valor != v && paux->prox != NULL)
    {
        aux = paux;
        paux = paux->prox;
    }
    if(paux->valor == v)
    {
        if(aux!=NULL)               //caso tenha mais que um valor
            aux->prox = paux->prox;
        else
            *lst = (*lst)->prox;    //caso tenha apenas um valor
        free(paux);
    }
}
```

Exemplo de output:

```
Escolha a sua opcao:2

[122.0, 122.0, ]
```

```
Escolha a sua opcao:8

Indique o valor que deseja remover:122
```

```
Escolha a sua opcao:2

[122.0, ]
```

j) Remover todas as ocorrências de um elemento x da lista

Usámos como auxílio as funções para remover no fim e também para verificar o tamanho da lista (que irá ser abordada a seguir), pois tem o mesmo resultado e poupa trabalho e memória (explicado nos comentários):

```c
//alínea j
void RemoveTodasOcorrencias(plista * lst, float v)
{
    plista paux = *lst;
    int count = 51238;
    int size = 0;
    while(paux!=NULL && count != size) //a partir do momento em que count = size, não há mais nada a remover porque a função
                                       // RemovePrimeiraOcorrencia não removeu nada, logo, o count = size.
    {
      count = ComprimentoLista(*lst);  //count passa a ser o comprimento da lista
      RemovePrimeiraOcorrencia(lst, v); //funcao para remover a ocorrencia do valor
      size = ComprimentoLista(*lst);    // size passa a ser o novo comprimento (total-1)
      paux = *lst;                      //paux passa a ser o proximo elemeto e tudo se repete
    }
}
```

Exemplo de output:

```
Escolha a sua opcao:2

[122.0, 122.0, 125.0, 144.0, 123.0, 122.0, ]
```

```
Escolha a sua opcao:9

Indique o valor que deseja remover da lista:122
```

```
Escolha a sua opcao:2

[125.0, 144.0, 123.0, ]
```

k) Determinar o comprimento de uma lista

Este subprograma não foi complicado de se fazer. Simplesmente, enquanto a lista não acabava, incrementámos um count e devolvemos esse valor:

```c
int ComprimentoLista(plista lst)
{
    int count = 0;
    while(lst != NULL)
    {
        count++;
        lst = lst->prox;
    }
    return count;
}
```

Exemplo de output:

```
0.Sair do menu
Escolha a sua opcao:10

A lista tem 1 elemento(s).
```

l) Procurar um valor x numa lista e devolver um ponteiro para esse elemento

Este subprograma foi relativamente simples de se fazer. Simplesmente fizemos um ciclo while que fazia andar o ponteiro até que o campo valor da lista fosse igual ao valor que o utilizador pretendia. Caso a lista estivesse vazia, retornava 0, caso encontrasse o valor, retornava um ponteiro para o próprio valor:

```
float ProcuraValor(plista lst, float x)
{
    printf("\nInsira o valor que deseja procurar: ");
    scanf("%f", &x);
    while(lst!=NULL && lst->valor != x)
    {
        lst = lst->prox;
    }
    if(lst == NULL)
        return 0;
    else
    return lst->valor;
}
```

Exemplo de output:

```
Insira o valor que deseja procurar:122

Valor 122.0 encontrado.
```

m) Ler os valores de uma lista, construindo-a

Para conseguirmos montar uma lista, aproveitámos a função de juntar valores no fim. Ou seja, pedimos ao utilizador o número de elementos da lista e fazemos um ciclo for com o subprograma JuntaFim:

```
void MontaLista(plista * lst, int max)
{
        float valor;
        for(int i=1;i<=max;i++)
        {
            printf("\nInsira o valor que deseja inserir na posicao %d da lista: ", i);
        scanf("%f", &valor);
        JuntaFim(valor, lst);
    }
}
```

Exemplo de output:

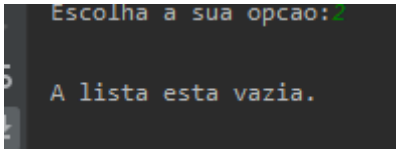
```
Escolha a sua opcao:12

Insira o numero de elementos da lista:2

Insira o valor que deseja inserir na posicao 1 da lista:111

Insira o valor que deseja inserir na posicao 2 da lista:222
```

n) Limpar uma lista

Mais uma vez, auxiliámos de uma função anterior para nos poupar trabalho e tempo. Fizemos, dentro do ciclo while (enquanto *lst != NULL) utilizamos o subprograma RemoveUltimo:

```
//alínea n
void LimpaLista(plista * lst)
{
    while(*lst!=NULL)
        RemoveUltimo(lst);
}
```

Output:

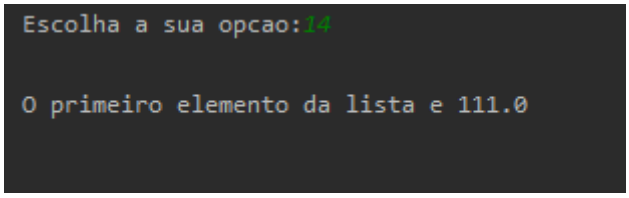


o) Mostrar o primeiro elemento da lista

Simplesmente retornamos o valor que está no início da lista:

```
float MostraPrimeiro(plista lst)
{
    if(ListaVazia(lst)=='y')
        printf("\nLista esta vazia.");
    return lst->valor;
}
```

Output:



p) Mostrar o último elemento de uma lista

Muito semelhante ao de cima, mas teremos que fazer um ciclo while até chegar ao último elemento e retornar o valor contido no mesmo:

```
float MostraUltimo(plista lst)
{
    if(ListaVazia(lst)=='y')
        printf("\nLista esta vazia.");
    else {
        while (lst->prox != NULL)
            lst = lst->prox;
    }
    return lst->valor;
}
```

Output:

```
Escolha a sua opcao:15

O ultimo elemento da lista e 222.0
```

q) Mostrar a cauda de uma lista

Para mostrar a cauda de uma lista, ou seja, todos os elementos excetuando o primeiro, teremos que ter o pointer a apontar para o elemento seguinte da lista. Ou seja, algo assim:

```
void MostraCauda(plista lst)
{
    if(ListaVazia(lst)=='y')
        printf("\nLista esta vazia.");
    else
    {
        lst = lst->prox;
        while(lst->prox!=NULL)
        {
            lst = lst->prox;
        }
        MostraLista(lst);
    }
}
```

Output:

```
Escolha a sua opcao:16

[333.0, 222.0, ]
```

r) Juntar duas listas numa só

Para juntarmos duas listas numa apenas, precisámos, obviamente, de criar outra lista, que demos o nome de lst2. De seguida, bastou fazer um ciclo até chegar ao último elemento da lista lst1 e meter o pointer que aponta para o próximo valor em vez de NULL, a apontar para lst2:

```
//alínea r
void ConcatenaListas(plista * lst1, plista * lst2) {

    plista aux = *lst1;
    if (*lst1 == NULL)
    {
        *lst1 = *lst2;
    } else
        {
        while (aux->prox != NULL){
            aux = aux->prox;
        }
            aux->prox = *lst2;
        }
}
//alínea s
```

Output:

```
Insira o numero de elementos da segunda lista:2

Insira o valor que deseja inserir na posicao 1 da lista:444

Insira o valor que deseja inserir na posicao 2 da lista:555

[111.0, 333.0, 222.0, 444.0, 555.0, ]
```

s) Copiar de uma lista para a outra

Para conseguir fazer com que uma lista fique com os mesmos valores da outra auxiliámo-nos, de novo, num exercício anterior, de novo no de juntar no fim da lista. Assim, bastou fazermos um ciclo e, dentro desse ciclo, chamarmos a função JuntaFim e avançarmos com o ponteiro:

```
void CopiaLista(plista * lst1, plista * lst2)
{
    plista paux = *lst1;
        while (paux != NULL) {
            JuntaFim(paux->valor, &(*lst2));
            paux = paux->prox;
    }
}
```

Output:

```
Escolha a sua opcao:18

[111.0, 333.0, 222.0, 444.0, 555.0, ]

[111.0, 333.0, 222.0, 444.0, 555.0, ]
```

t) Trocar dois elementos de uma lista

Pedindo ao utilizador os elementos que ele pretende trocar, foi apenas preciso fazer dois ciclos while na mesma lista até chegarem aos valores pretendidos. Depois, foi apenas preciso igualar os respetivos campos de valores da lista aos valores introduzidos pelo utilizador:

```c
//alínea t
void TrocaElementos(plista * lst, float x, float y)
{
    plista paux = *lst;
    plista aux = *lst;
    printf("\nInsira o valor que quer trocar: ");
    scanf("%f", &x);
    printf("\nInsira o proximo valor a trocar: ");
    scanf("%f", &y);

    if(paux == NULL || aux == NULL)
        printf("\nUma das listas esta vazia!");
    else {
        while (paux != NULL && paux->valor != x)
            paux = paux->prox;

        while (aux != NULL && aux->valor != y)
            aux = aux->prox;

        paux->valor = y;
        aux->valor = x;
    }
}
```

Output:

```
Escolha a sua opcao:19

Insira o valor que quer trocar:555

Insira o proximo valor a trocar:444

[111.0, 333.0, 222.0, 555.0, 444.0, ]
```

u) Ordenar uma lista por ordem crescente dos seus valores

Para conseguirmos ordenar uma lista, seja de que forma for, necessitamos de trocar os valores dos pointers que apontam para o próximo valor/valor atual. Além disso, necessitamos do género de uma flag para o subprograma ir fazendo os ciclos vezes sem conta até estar tudo ordenado. Deste modo, tivemos que fazer desta maneira para ordenar de forma **crescente** os números:

```
//alínea u
void OrdenaCrescente(plista * lst)
{
    plista paux = NULL;
    float auxiliar;
    int flag = 0;
    while(flag == 0)
    {
        flag = -1;
        paux = *lst;
        while(paux->prox != NULL)
        {
            if (paux->valor > (paux->prox)->valor)
            {
                auxiliar = (paux->prox)->valor;
                (paux->prox)->valor = paux->valor;
                paux->valor = auxiliar;
                flag = 0;
            }
            paux = paux->prox;
        }
    }
}
f MostraCauda
```

Output:

```
Escolha a sua opcao:20

[111.0, 222.0, 333.0, 444.0, 555.0, ]
```

v) Ordenar por ordem descrescente os valores de uma lista

Exatamente a mesma lógica do subprograma anterior mas por ordem decrescente dos valores:

```c
//alinea v
void OrdenaDecrescente(plista * lst)
{
    plista paux = NULL;
    float auxiliar;
    int flag = 0;
    while(flag == 0)
    {
        flag = -1;
        paux = *lst;
        while(paux->prox != NULL)
        {
            if (paux->valor < (paux->prox)->valor)
            {
                auxiliar = (paux->prox)->valor;
                (paux->prox)->valor = paux->valor;
                paux->valor = auxiliar;
                flag = 0;
            }
            paux = paux->prox;
        }
    }
}
```

Output:

```
Escolha a sua opcao:21

[555.0, 444.0, 333.0, 222.0, 111.0, ]
```

2.3 Considerar a representação de uma lista

a) Crie uma nova estrutura lista ajustada a esta forma (enunciado):

Para fazer este exercício, criámos duas estruturas: uma para os pointers e outra para a lista em si:

```c
#include <stdio.h>
#include <stdlib.h>
//alínea a

typedef struct lno * plista;
typedef struct lno
{
    float valor;
    struct lno *prox;
}listano;

struct ponteiros
{
    plista head;
    plista tail;
};typedef struct ponteiros pt;
```

b) Altere as alíneas a), b) e), g), j), n), p) e t) do exercício anterior de modo a que possam ser usadas na nova representação

As alíneas a), b), n) e p) são exatamente iguais às do exercício anterior. Já as seguintes tivemos que alterar consoante a nova lista.

Alínea e):

Para conseguirmos acrescentar elementos no fim das listas com esta nova metodologia, usámos o subprograma que fizemos anteriormente mas adaptámos a esta nova ideia. Assim, em vez de termos um pointer a apontar para o início da lista e ao mesmo tempo para o fim, temos um pointer no início da lista que percorre a lista e um pointer que aponta para o fim da lista. Então ficou deste modo:

```
void JuntaFim(float x, plista * head, plista * tail)
{
    char c = ListaVazia(*head);
    plista no = (plista) malloc(sizeof(listano));
    if(no==NULL)
        printf("\nNao ha memoria disponivel!");

    if(c=='n') {

        no->valor = x;
        no->prox = NULL;

        (*tail)->prox = no;
        (*tail) = no;
    } else {
        no->valor = x;
        no->prox = NULL;

        *head = no;
        *tail = *head;
    }
}
```

Alínea g):

Para esta alínea, depois de percorrermos a lista, caso tenhamos apenas um valor na lista, basta meter os valores da head e da tail a NULL, para a lista ficar vazia. Caso tenha mais que um elemento, quando encontramos o último elemento, metemos a tail a apontar para o elemento anterior e o pointer do tail a apontar para o último, algo deste género (página seguinte):

```c
void RemoveUltimo(plista * tail, plista * head)
{
    char c = ListaVazia(*head);
    plista paux = NULL;
    plista aux = NULL;
    if(c=='n')
    {
        paux = *head;
        if(paux->prox == NULL)
        {
            *tail = NULL;
            *head = NULL;
        } else
            {
            while(paux->prox != NULL)
            {
                aux = paux;
                paux = paux->prox;
            }
            *tail = aux;
            (*tail)->prox = NULL;
            }
    } else printf("\nLista esta vazia!");
}
```

Alínea j):

Para esta função, depois de encontrarmos o valor pretendido, caso o valor pretendido se encontrasse no início da lista, o paux passa a apontar para o próximo e o head ia apontar para paux. Caso fosse o último, a tail aponta para esse valor e, caso esteja no meio da lista, o ponteiro anterior passa a apontar para o pointeiro atual para ficar “normal”.

```c
void RemoveTodasOcorrencias(plista * tail, plista * head, float x)
{
    char c = ListaVazia(*head);
    plista paux, aux;
    if(c=='n')
    {
        paux = *head;
        if(paux->valor == x)
        {
            paux = paux->prox;
            *head = paux;
        }
        else
        {
            while(paux->prox != NULL)
            {
                aux = paux;
                paux = paux->prox;
                if(paux->valor == x)
                {
                    if(paux->prox == NULL)
                    {
                        *tail = aux;
                        aux->prox = paux->prox;
                    } else{
                            aux->prox = paux->prox;
                    }
                }
            }
```

RemoveTodasOcorrencias

Alínea t)

Muito semelhante ao do exercício anterior, simplesmente apontámos os ponteiros auxiliares para o head:

```
void TrocaListas(plista * head, float x, float y)
{
    plista paux = *head;
    plista aux = *head;
    printf("\nInsira o valor que quer trocar: ");
    scanf("%f", &x);
    printf("\nInsira o proximo valor a trocar: ");
    scanf("%f", &y);

    if(paux == NULL || aux == NULL)
        printf("\nUma das listas esta vazia!");
    else {
        while (paux != NULL && paux->valor != x)
            paux = paux->prox;

        while (aux != NULL && aux->valor != y)
            aux = aux->prox;

        paux->valor = y;
        aux->valor = x;
    }
}
```

2.4 Considere a estrutura de dados Pilha

Os exercícios a) e b) são iguais aos do enunciado da professora do exercício 2. A alínea e) também é igual à função para mostrar o primeiro elemento da lista do exercício 2. Para os exercícios 4 e 5, simplesmente pegámos nas funções utilizadas anteriormente e adaptámos a estas novas funções. Por exemplo, neste caso, o último elemento a entrar tem que ser o primeiro a sair. Assim, podemos utilizar a função JuntaInicio, pois o elemento entra em último e fica à frente dos elementos que já lá estavam.

Alínea c) (Empilhar um elemento da lista):

```
void AcrescentaFila(float x, pfila * lst)
{
    pfila no = (pfila) malloc(sizeof(fila));
    pfila paux = *lst;
    if (no == NULL)
        printf("\nErro, nao ha memoria disponivel.");
    else {
        no->valor = x;
        no->prox = NULL;
        if (*lst == NULL) {
            *lst = no;
        } else {
            while (paux->prox != NULL) {
                paux = paux->prox;
            }
            paux->prox = no;
        }
    }
}
```

Alínea d) (Retirar um elemento da lista):

```
void RemoveFila(pfila * lst)
{
    pfila paux = *lst;
    (*lst) = (*lst)->prox;
    free(paux);
}
```

De novo a mesma lógica de funções anteriores, neste caso do subprograma RemoveInicio, para remover o elemento do início para dar seguimento à lógica do LastInFirstOut.

2.5 Considere a estrutura de dados Fila

Para estes subprogramas, a lógica foi exatamente a mesma dos do exercício 4, mas em vez de utilizar a função para juntar no início, utilizei a de juntar no fim. De resto é exatamente igual.

2.6 Refazer o exercício 4 da ficha 4 mas com uma lista ligada em vez de tabela

a) Defina uma estrutura de dados que, para além dos dados de um funcionário, tenha um ponteiro

A estrutura que nós utilizámos foi a seguinte:

```
//alínea a
typedef struct lno * plista;
typedef struct lno
{
    int numero;
    char nome[200];
    char tarefa[200];
    float salario;
    struct lno *prox;
}listano;
```

b) Elabore subprogramas para:

i. Introduzir os dados de um funcionário na lista

Para conseguirmos introduzir os dados dos funcionários, bastou pedirmos ao utilizador o número de funcionários (no main) e usar scanf e gets. Depois, caso a lista esteja vazia, temos o pointer do lst a apontar para o nó para este ser o primeiro e único valor da lista. Caso tenha elementos já, irá percorrer a lista até ao fim e o pointer do último elemento passa a apontar para o "novo funcionário". O código é demasiado grande para meter num print, portanto dividimos em dois:

```
void EscreveDados(float x, int nmr, char nome[], char tarefa[], plista * lst) {
        plista no = (plista) malloc(sizeof(listano));
        plista paux = *lst;
        if (no == NULL)
            printf("\nErro, nao ha memoria disponivel.");
        else {
            printf("\nIndique o numero do funcionario: ");
            scanf("%d", &nmr);
            printf("\nInsira o nome do funcionario: ");
            fflush(stdin);
            gets(nome);
            printf("\nInsira a tarefa do funcionario: ");
            fflush(stdin);
            gets(tarefa);
            printf("\nInsira o salario do funcionario: ");
            scanf("%f", &x);
```

```
            no->salario = x;
            no->numero = nmr;
            strcpy(no->tarefa, tarefa);
            strcpy(no->nome, nome);
            no->prox = NULL;
            if (*lst == NULL) {
                *lst = no;
            } else {
                while (paux->prox != NULL) {
                    paux = paux->prox;
                }
                paux->prox = no;
            }
        }
```

Exemplo de output:

```
Insira o numero de funcionarios:2

Indique o numero do funcionario:121

Insira o nome do funcionario:Antonio

Insira a tarefa do funcionario:Estudar

Insira o salario do funcionario:1234

Indique o numero do funcionario:122

Insira o nome do funcionario:Joana

Insira a tarefa do funcionario:Estudar

Insira o salario do funcionario:1234
```

ii. Apresentar os dados de todos os funcionários da lista

Bastou fazermos printf para cada campo da lista dentro de um ciclo while e andar com o ponteiro da mesma:

```
void ApresentarDados(plista lst)
{
    if(lst == NULL)
        printf("\nNao existem funcionarios!!");
    else{
        while (lst != NULL)
        {
            printf("\nNome do funcionario: %s", lst->nome);
            printf("\nNumero do funcionario: %d", lst->numero);
            printf("\nTarefa do funcionario: %s", lst->tarefa);
            printf("\nSalario do funcionario: %.1f", lst->salario);
            printf("\n\n");
            lst = lst->prox;
        }
```

Output:

```
Nome do funcionario: Antonio
Numero do funcionario: 121
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0


Nome do funcionario: Joana
Numero do funcionario: 122
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0
```

iii. Apresentar todos os dados de funcionários com salário acima de 500€

Bastou utilizar o método do programa anterior mas com um if para distinguir os que têm salário acima de 500 e os que não:

```
void SalarioAcima500(plista lst)
{
    if(lst == NULL)
        printf("\nNao existem funcionarios!");
    else {
        while (lst != NULL) {
            if (lst->salario > 500) {
                printf("\nNome do funcionario: %s", lst->nome);
                printf("\nNumero do funcionario: %d", lst->numero);
                printf("\nTarefa do funcionario: %s", lst->tarefa);
                printf("\nSalario do funcionario: %.1f", lst->salario);
                printf("\n");
            }
            lst = lst->prox;
        }
    }
}
```

Output:

```
Funcionarios com salario acima de 500:


Nome do funcionario: Antonio
Numero do funcionario: 121
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0
```

Nota: Neste output eu alterei o salário da Joana para 499, logo, não aparece aqui.

iv. Procurar e devolver uma estrutura funcionário usando o seu nome e, caso não exista, devolver NULL

Para esta função bastou-nos fazer, de novo, outro ciclo while e andar com o ponteiro até encontrar o nome pretendido. Depois, bastou retornar a lista:

```
plista ProcuraFuncionario(plista lst)
{
    char procurar[200];
    printf("\nInsira o nome do funcionario que deseja encontrar: ");
    fflush(stdin);
    gets(procurar);
            while(lst!=NULL)
            {
                if(strcmp(lst->nome, procurar)==0)
                {
                    return lst;
                    break;
                }
                lst = lst->prox;
            }
            return NULL;
}
```

Output:

```
Insira o nome do funcionario que deseja encontrar:Joana

Funcionario Joana encontrado(a)!
```

v. Atualizar os dados de um funcionário utilizando o seu número

Muito parecido com a função de escrever os dados, mas acrescentamos um ciclo while para que, quando encontrado o funcionário pretendido, o programa peça para inserir de novo os dados do funcionário:

```
void AtualizarDados(plista lst)
{
    int numero, nmr;
    char nome[200], tarefa[200];
    float x;
    printf("\nInsira o numero do funcionario cujos dados pretende atualizar: ");
    scanf("%d", &numero);
    if(lst == NULL)
        printf("\nNao existem funcionarios!");
    else {
        while(lst != NULL)
        {
            if (lst->numero == numero)
```

```
            if (lst->numero == numero)
            {
                printf("\nIndique o numero do funcionario: ");
                scanf("%d", &nmr);
                printf("\nInsira o nome do funcionario: ");
                fflush(stdin);
                gets(nome);
                printf("\nInsira a tarefa do funcionario: ");
                fflush(stdin);
                gets(tarefa);
                printf("\nInsira o salario do funcionario: ");
                scanf("%f", &x);
                lst->salario = x;
                lst->numero = nmr;
                strcpy(lst->tarefa, tarefa);
                strcpy(lst->nome, nome);
            }
            lst = lst->prox;
        }
    }
```

Output:

```
Insira o numero do funcionario cujos dados pretende atualizar:122

Indique o numero do funcionario:122

Insira o nome do funcionario:Joana

Insira a tarefa do funcionario:Estudar

Insira o salario do funcionario:1234

Lista atualizada de funcionarios:
```

```
Nome do funcionario: Antonio
Numero do funcionario: 121
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0


Nome do funcionario: Joana
Numero do funcionario: 122
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0
```

vi. Ordenar a lista por ordem alfabética dos funcionários

Por fim, para conseguirmos fazer esta função, baseámo-nos no subprograma da alínea u) no exercício 2. Assim, ficou algo deste tipo:

```
void OrdenaNomes(plista * lst)
{
    plista paux = NULL;
    char auxiliar[200];
    int flag = 0;
    while(flag == 0)
    {
        flag = -1;
        paux = *lst;
        while(paux->prox != NULL)
        {
            if (strcmp(paux->nome, (paux->prox)->nome)>0)
            {
                strcpy(auxiliar, (paux->prox)->nome);
                strcpy((paux->prox)->nome, paux->nome);
                strcpy(paux->nome, auxiliar);
                flag = 0;
            }
            paux = paux->prox;
        }
    }
}
```

Output:

```
Nome do funcionario: Joana
Numero do funcionario: 121
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0


Nome do funcionario: Antonio
Numero do funcionario: 122
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0
```

```
Lista apos ordenada alfabeticamente:
Nome do funcionario: Antonio
Numero do funcionario: 121
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0


Nome do funcionario: Joana
Numero do funcionario: 122
Tarefa do funcionario: Estudar
Salario do funcionario: 1234.0
```

3 Conclusão

Com esta ficha de trabalho, ficámos a ter uma maior noção de como funcionam as listas ligadas na linguagem C. Uma ficha que ao início parecia muito mais complicada do que realmente foi, apesar de não ter sido simples também, especialmente em algumas alíneas do exercício 2. Resumidamente, foi um trabalho bem conseguido e achamos que fizemos tudo o que a professora pretendia.

4 Referências

PDFs e explicações da professora

5 Anexos

Listar ficheiros com código criado...