Programação 2023/24

LEI, LEI-PL, LEI-CE

Aula Laboratorial 8

Bibliografia:
K. N. King. *C programming: A Modern Approach* (2nd Edition). W. W. Norton: capítulo 17.

Código de apoio para a aula: https://gist.github.com/FranciscoBPereira

Listas Ligadas Simples

Exercícios Obrigatórios

1. Considere a definição da estrutura *struct local* que permite criar uma lista ligada simples, em que cada nó tem uma string armazenada.

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

typedef struct local no, *pno;

struct local{
    char nome[50];
    pno prox;
};
```

A função *addNo_1( )* adiciona um nó à lista ligada. Recebe como parâmetros um ponteiro para o início da lista e a *string* a adicionar. Devolve um ponteiro para o início da lista, depois de criar um novo nó, preencher com a *string* recebida e efetuar a inserção na lista.

```
pno addNo_1(pno lista, char *st){
    pno novo;

    novo = malloc(sizeof(no));
    if(novo == NULL) return lista;
    strcpy(novo->nome, st);
    novo->prox = lista;
    return novo;
}
```

a) Qual a estratégia de inserção na lista seguida por esta função? Em que posição da lista são colocados os nós que são criados e adicionados?

b) Após a execução desta função *main()* qual é a constituição da lista ligada, assumindo que não existem erros de gestão de memória?

```
int main() {
    pno lista = NULL;

    lista = addNo_1(lista, "AAA");
    lista = addNo_1(lista, "BBB");
    lista = addNo_1(lista, "CCC");
    ...

}
```

A função addNo_2( ) adiciona um nó à lista ligada. Recebe como parâmetros um ponteiro para o início da lista e a string a adicionar. Devolve um ponteiro para o início da lista, depois de criar um novo nó, preencher com a string recebida e efetuar a inserção na lista.

```
pno addNo_2(pno lista, char *st){
    pno novo, aux=lista;

    novo = malloc(sizeof(no));
    if(novo == NULL) return lista;
    strcpy(novo->nome, st);
    novo->prox = NULL;
    if(lista == NULL)
        lista = novo;
    else{
        while(aux->prox != NULL)
            aux = aux->prox;
        aux->prox = novo;
    }
    return lista;
}
```

c) Qual a estratégia de inserção na lista seguida por esta função? Em que posição da lista são colocados os nós que são criados e adicionados?

d) Após a execução desta função *main()* qual é a constituição da lista ligada, assumindo que não existem erros de gestão de memória?

```
int main() {
    pno lista = NULL;

    lista = addNo_2(lista, "AAA");
    lista = addNo_2(lista, "BBB");
    lista = addNo_2(lista, "CCC");
    ...

}
```

A função addNo_3( ) adiciona um nó à lista ligada. Recebe como parâmetros um ponteiro para o início da lista e a string a adicionar. Devolve um ponteiro para o início da lista, depois de criar um novo nó, preencher com a string recebida e efetuar a inserção na lista.

```
pno addNo_3(pno lista, char *st){
    pno novo;

    novo = malloc(sizeof(no));
    if(novo == NULL) return lista;
    strcpy(novo->nome, st);
    if(lista == NULL) {
        novo->prox = lista;
        lista = novo;
    }
    else {
        novo->prox = lista->prox;
        lista->prox = novo;
    }
    return lista;
}
```

e) Qual a estratégia de inserção na lista seguida por esta função? Em que posição da lista são colocados os nós que são criados e adicionados?

f) Após a execução desta função *main()* qual é a constituição da lista ligada, assumindo que não existem erros de gestão de memória?

```
int main() {
    pno lista = NULL;

    lista = addNo_3(lista, "AAA");
    lista = addNo_3(lista, "BBB");
    lista = addNo_3(lista, "CCC");
    ...
}
```

g) Após a execução desta função *main()* qual é a constituição da lista ligada, assumindo que não existem erros de gestão de memória?

```
int main() {
    pno lista = NULL;

    lista = addNo_3(lista, "AAA");
    lista = addNo_1(lista, "BBB");
    lista = addNo_1(lista, "CCC");
    lista = addNo_2(lista, "DDD");
    lista = addNo_3(lista, "EEE");
    ...
}
```

2. Pretende-se utilizar uma lista ligada simples para armazenar informação sobre o índice de massa corporal de um conjunto de indivíduos. A estrutura a utilizar para armazenar a informação de cada pessoa é a seguinte:

```
typedef struct pessoa no, *pno;
struct pessoa{
      char nome[100];
      int id;
      float peso, altura, imc;
      pno prox;
};
```

O índice de massa corporal (imc) é obtido através da seguinte expressão: $imc = peso/altura^2$, em que o peso é especificado em kg e a altura em metros.

a) Crie um ponteiro de lista na função *main()*.

b) Escreva uma função em C que adicione um nó à lista, contendo os dados de uma nova pessoa. O nome, identificador numérico, peso e altura dessa pessoa devem ser obtidos do utilizador. A função deve garantir que o id indicado é único e não se encontra atribuído a outra pessoa que já esteja na lista. A nova pessoa deve ser adicionada ao final da lista ligada.

c) Escreva uma função em C que mostre na consola a informação completa de todas as pessoas que se encontram na lista ligada, incluindo o seu imc.

d) Escreva uma função em C que atualize o peso de uma das pessoas que se encontra na lista. O id da pessoa a considerar e o seu novo peso são 2 dos parâmetros da função.

e) Escreva uma função em C que elimine uma pessoa da lista ligada. O id da pessoa a eliminar é um dos parâmetros da função. O nó retirado da lista deve ser libertado.

f) Escreva uma função em C que elimine todos os nós da lista ligada.

g) Escreva uma função em C que elimine da lista todas as pessoas com um imc superior a um determinado limite. O valor limite a considerar é um dos parâmetros da função. Todos os nós retirados da lista devem ser libertados.

h) Escreva uma função em C que elimine os K últimos nós da lista ligada. O valor K é um dos parâmetros da função. Se a lista tiver menos do que K elementos, a função não efetua nenhuma alteração.

3. Pretende-se alterar a gestão da lista ligada definida na questão 2, por forma a que as pessoas na lista passem a estar ordenadas por imc crescente. Altere o que for necessário nas funções implementadas na questão anterior, para garantir que a informação passa a estar organizada desta forma.

Exercícios Complementares

4. Um consultório médico pretende controlar o atendimento de sucessivos pacientes de forma a respeitar a ordem de chegada. Pretende-se implementar um programa que faça a gestão de uma lista ligada em que cada nó corresponde a um dos pacientes que se encontra à espera. A lista ligada

funcionará como uma fila de espera, em que os pacientes, ao chegarem ao consultório, vão para o final da lista. O atendimento é sempre efetuado ao primeiro paciente da lista.

a) Considerando que cada paciente é completamente identificado pelo seu nome, defina a(s) estrutura(s) que considerar necessária(s) para a resolução do problema.

b) Crie um ponteiro de lista na função *main()*.

c) Escreva uma função em C que adicione um novo paciente ao final da fila de espera. O nome do paciente que acabou de chegar deve ser introduzido pelo utilizador.

d) Escreva uma função em C que permita visualizar toda a fila de espera.

e) Escreva uma função em C que implemente o atendimento de um novo paciente. A função deve retirar o elemento que está no início da fila de espera, escrever o seu nome no monitor e libertar o espaço ocupado por este nó.

f) Escreva uma função em C que permita retirar da fila um paciente que tenha desistido de esperar. A função deve obter o nome do paciente através do utilizador. Esta operação corresponde a eliminar um dos nós da lista, devendo o espaço ser libertado.

5. Considerando o exercício anterior, nem todos os pacientes necessitam de ser atendidos com a mesma urgência. Considere que existem três graus de urgência: grávidas (nível 1), idosos e crianças (nível 2) e restantes pacientes (nível 3). Só serão atendidos pacientes do nível 2 se não existirem pacientes do nível 1 e só serão atendidos pacientes do nível 3 se não existirem pacientes dos níveis 1 e 2.
Pretende-se que modifique a função de inserção de um novo paciente de modo a ter em conta estes três níveis de prioridade. A fila de espera deve continuar a ser única e a função de atendimento não deve ser alterada (o próximo paciente a ser atendido é o que se encontra no início da fila). Para responder a esta questão deve efetuar as alterações necessárias na definição da(s) estrutura(s) de dados.

Sugestão: Para encontrar o ponto de inserção de um novo paciente, a função deve procurar o primeiro nó da lista que tenha urgência menor (por exemplo, se o novo paciente for de nível 1, então a função deve procurar o primeiro nó que tenha nível 2 ou superior). A inserção do novo paciente é feita imediatamente antes deste ponto.

Trabalho Autónomo

6. As listas ligadas podem servir como suporte para a implementação de várias estruturas de dados fundamentais. Algumas destas estruturas foram já informalmente utilizadas nas questões anteriores, mas nesta secção serão definidas de uma forma mais concreta. Os alunos interessados podem consultar o livro *Algorithms in C* de Robert Sedgewick (disponível na biblioteca do ISEC), para uma visão mais completa das estruturas de dados fundamentais.

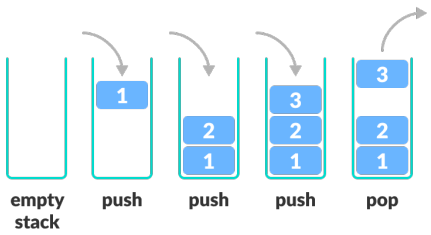
O tipo de dados a utilizar nesta secção é o seguinte:

```
typedef struct dados no, *pno;

struct dados{
      int v;
      char c;
      pno prox;
};
```

a) **Pilha (*Stack*)**: A pilha é uma estrutura de dados fundamental muito utilizada na área da computação. Armazena um conjunto de elementos, em que a inserção/eliminação é efetuada da forma LIFO (last-in-first-out), ou seja, os últimos elementos a chegar são os primeiros a sair. As operações fundamentais são:

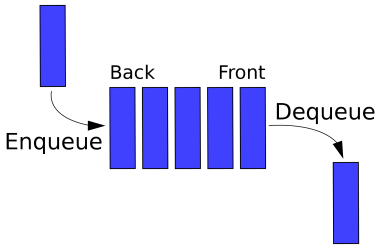
i. Adicionar um elemento ao topo da pilha (*Push*).
ii. Retirar um elemento do topo da pilha (*Pop*).
iii. Verificar se a pilha está vazia (*Is_Empty*).
iv. - Mostrar o conteúdo da pilha.



Implemente as operações básicas que permitam gerir uma pilha.

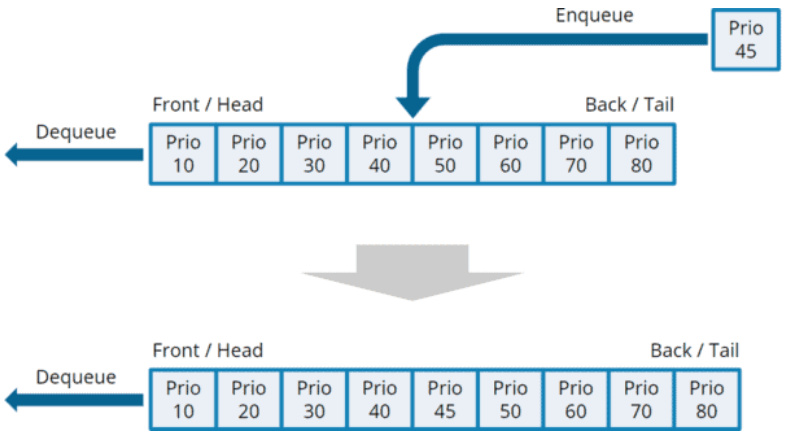
b) **Fila de Espera (*Queue*)**: As filas de espera são estruturas de dados do tipo FIFO (first-in-first-out), em que vários elementos se encontram armazenados de uma forma sequencial. As operações fundamentais são:

i. Adicionar um elemento ao final da fila de espera (*Enqueue*).
ii. Retirar um elemento do início da fila de espera (*Dequeue*).
iii. Verificar se a fila está vazia (*Is_Empty*).
iv. Mostrar o conteúdo da fila.



Implemente as operações básicas que permitam gerir uma fila de espera.

c) **Fila de Espera com Prioridade (*Priority Queue*)**: As filas de espera com prioridade são estruturas de dados do tipo fila em que cada elemento tem uma prioridade associada. Elementos com prioridade superior são atendidos antes dos elementos com prioridade inferior. A inserção na fila é efetuada de forma a manter a prioridade. Desta forma, garante-se que os elementos são sempre retirados do início da fila. No exemplo desta questão, considere que a prioridade corresponde ao valor do campo *v*. As operações fundamentais são:
   i. Adicionar um elemento à fila de espera, mantendo a prioridade (*Enqueue_with_Priority*).
   ii. Retirar um elemento do início da fila de espera (*Dequeue_Highest_Priority*).
   iii. Verificar se a fila está vazia (*Is_Empty*).
   iv. Mostrar o conteúdo da fila.
   v. Verificar prioridade máxima (*Peek*): Devolve a prioridade máxima existente na fila (não altera a estrutura de dados).



Implemente as operações básicas que permitam gerir uma fila de espera com prioridade.