Algoritmos de Programação II - Trabalho 2

Prof. Lucas Reis November 8, 2021

Auxiliando a vacinação

A secretaria de saúde de Campo Grande teve um trabalho muito importante neste ano de 2021: a vacinação contra a COVID-19. Recentemente, atingimos a meta de 65% da população com esquema vacinal completo. Entretanto, o sistema que cataloga a vacinação encontrou sérios problemas: a constante necessidade de inserção e remoção de novos usuários estava sendo muito custosa, já que essas operações estavam sendo armazenadas em um vetor. Portanto, a SESAU recorreu a você para reimplementar o sistema de inserção de dados da vacinação utilizando uma estrutura de **Lista Encadeada**.

A pessoa

Os dados de cada pessoa capacitada a receber a vacina são armazenados em um registro (struct) que contém os seguintes campos:

```
struct Pessoa
{
    char nome[];
    int cpf;
    int nDoses;
    Pessoa *prox;
};
```

`nome` é um campo que contêm o nome de cada aluno, **que pode conter espaços**. Considere que nenhum aluno possui nome com mais de 100 caracteres. `cpf` é o campo que armazena o CPF de cada pessoa. `nDoses` é um campo que indica quantas doses da vacina a pessoa já recebeu. Para toda pessoa recém-cadastrada, esse valor é 0.

O Sistema

O sistema que você deve estruturar deve ser capaz de inserir, buscar e aplicar doses em diferentes pessoas armazenadas em uma estrutura de lista encadeda. Ele também deve ser capaz de imprimir informações sobre pessoas presentes nesse sistema, incluindo seu nome, CPF e a quantidade de doses recebidas da vacina.

As funções

O seu trabalho **deve** implementar as seguintes funções:

Busca

```
Pessoa *buscaNome(Pessoa *lst, char *nome);
Pessoa *buscaCPF(Pessoa *lst, int cpf);
```

Que realiza a busca na lista (cujo ponteiro para seu início é `lst`) por uma pessoa com o respectivo `nome` ou `CPF`. Essas funções devem retornar um ponteiro para a pessoa específica, ou `NULL` caso essa pessoa não exista na lista.

Inserção

- Sem cabeça

  ```
  void insere(Pessoa **lst, char *nome, int cpf);
  ```

- Com cabeça

  ```
  void insere(Pessoa *lst, char *nome, int cpf);
  ```

que recebe um ponteiro para o início da lista (`lst`), um `nome` e um `cpf` e cria uma nova **Pessoa** para ser inserida. Como estamos prezando por uma inserção em tempo constante, **Essa pessoa deve ser inserida no início da lista**.

Você **deve** criar todas as células (pessoas) usando **alocação dinâmica**).

Remoção

- Sem cabeça

  ```
  bool removePorNome(Pessoa **lst, char *nome);
  bool removePorCPF(Pessoa **lst, int cpf);
  ```

- Com cabeça

  ```
  bool removePorNome(Pessoa *lst, char *nome);
  bool removePorCPF(Pessoa *lst, int cpf);
  ```

que devem buscar na lista por uma pessoa com `nome` ou `cpf` específico e removê-la da lista. A função deve retornar `true` caso a remoção tenha ocorrido, e `false` caso contrário.

Você **deve liberar a memória** utilizada pela célula removida utilizando `free()`.

Incrementar Doses

```
bool IncrementaDosesPorNome(Pessoa *lst, char *nome);
bool incrementaDosesPorCPF(Pessoa *lst, int cpf);
```

Essas funções devem buscar por uma pessoa específica em sua lista encadeada (por `nome` ou `cpf`) e incrementar a quantidade de doses que essa pessoa tomou, se ela existir. Sua função deve retornar `true` caso tenha sido capaz de incrementar a dose da pessoa buscada, ou `false`

caso contrário (se a pessoa não existe ou se ela já possui duas doses). Você pode usar as funções `buscaNome` e `buscaCPF` para realizar a busca antes de incrementar a quantidade de doses.

Lembre-se de incrementar apenas pessoas que tem menos de duas doses.

Impressão

```
void imprimePessoa(Pessoa *p);
void imprimeLista(Pessoa *lst);
```

A primeira função deve imprimir as informações de uma única pessoa `p`.

A segunda função deve imprimir todas as pessoas presentes na lista encadeada enviada, na ordem em que elas se encontram na estrutura.

A impressão de cada pessoa deve seguir o seguinte formato:

```
------------
Pessoa de nome: nome
CPF: cpf
Num. de doses aplicadas: nDoses
------------
```

onde `nome`, `cpf` e `nDoses` são o nome, CPF e número de doses aplicadas na pessoa citada, respectivamente.

Caso a pessoa informada não exista (valor `NULL`) (pessoa não encontrada), sua função deve imprimir:

```
------------
Pessoa de nome: null
CPF: null
Num. de doses aplicadas: null
------------
```

Limpeza da lista

```
void limpaLista(Pessoa **lst);
```

Essa função deve percorrer a lista encadeada e limpar a memória de cada uma das células presentes. Ao final, ela deve atualizar o valor do ponteiro da lista (`lst`) para `NULL`.

A função main

A função `main` do seu programa deve, primeiramente, ler um inteiro $n \leq 10000$. Em seguida, você deve ler `n` pessoa, onde cada leitura de estudante deve ser feita em duas linhas. A primeira

delas deve informar o nome, e a segunda deve informar o CPF da n-ésima pessoa. Esses valores devem ser passados para `insere()` de modo a incrementar sua lista de pessoas.

Logo após, sua main deve ler repetidamente uma caractere `op`, que simboliza a ação que deve ser realiza na lista. As possíveis ações e o respectivo caratere de cada uma são:

- *imprimir* - `i`: Essa opção requisitar a impressão de uma pessoa
- *remover*: - `r`: Essa opção sinaliza a remoção de uma pessoa da lista
- *incrementar doses* - `d`: Essa opção sinaliza o incremento de número de doses de uma pessoa da lista

Cada opção lida deve ser acompanhada de um segundo caractere `op2`, sinalizando se a operação realizada deve ser baseada em um `nome` ou `cpf`. Assim, seu código deve esperar a leitura de um nome ou cpf específico para encontrar a pessoa buscada e depois realizar a impressão, remoção ou incremento de dose dessa pessoa.

Para opreações de incremento de dose ou remoção, seu programa deve imprimir alguma informação como feedback sinalizando se a operação anterior foi um sucesso ou não. Caso a operação tenha sucedido (indicado pelo retorno de `true` da função chamada), você deve imprimir `"OK!"`. Caso contrário, deverá imprimir `"FAIL!"`.

Seu programa deve continuar esperando novas opções `op` até que seja lido o caractere `x`, que indica o fim da leitura de ações. Por fim, seu programa deve imprimir a lista encadeada em ordem, com informações de todas as pessoas existentes.

Como último passo, seu programa **deve esvaziar a lista, limpando a memória alocada para cada pessoa presente** (use a função `limpaLista()`!!).

Abaixo seguem exemplos de possíveis casos de teste e a resposta esperada para seu programa para cada um deles.

Exemplo 1

Entrada

```
5
wilson
123456
joana
78910
felipe c
112
giovana
654
gerson
123547
d c
123647
i n
gerson
r n
giovana
d n
felipe c
d n
ana
d c
123456
x
```

Saída

```
FAIL!
------------
Pessoa de nome: gerson
CPF: 123547
Num. de doses aplicadas: 0
------------
OK!
OK!
FAIL!
OK!
------------
Pessoa de nome: gerson
CPF: 123547
```

```
Num. de doses aplicadas: 0
------------
------------
Pessoa de nome: felipe c
CPF: 112
Num. de doses aplicadas: 1
------------
------------
Pessoa de nome: joana
CPF: 78910
Num. de doses aplicadas: 0
------------
------------
Pessoa de nome: wilson
CPF: 123456
Num. de doses aplicadas: 1
------------
```

Exemplo 2

Entrada

```
5
wilson
123456
joana
78910
felipe c
112
cleiton
654
gerson
123547
d c
78910
d n
joana
i n
joana
d c
123547
d n
felipe c
d n
cleiton
```

```
d c
654
i n
ana
x
```

Saída

```
OK!
OK!
------------
Pessoa de nome: joana
CPF: 78910
Num. de doses aplicadas: 2
------------
OK!
OK!
OK!
OK!
------------
Pessoa de nome: null
CPF: null
Num. de doses aplicadas: null
------------
------------
Pessoa de nome: gerson
CPF: 123547
Num. de doses aplicadas: 1
------------
------------
Pessoa de nome: cleiton
CPF: 654
Num. de doses aplicadas: 2
------------
------------
Pessoa de nome: felipe c
CPF: 112
Num. de doses aplicadas: 1
------------
------------
Pessoa de nome: joana
CPF: 78910
Num. de doses aplicadas: 2
------------
```

```
------------
Pessoa de nome: wilson
CPF: 123456
Num. de doses aplicadas: 0
------------
```

Limpeza de memória

Saber organizar e limpar memória não utilizada é parte integral de programar em C++ quando se utiliza memória alocada dinamicamente. É `sua obrigação` alocar e limpar memória corretamente, de modo que seu programa **não tenha** memória alocada no heap no momento em que ele é encerrado (o fim da main).

Uma ótima ferramenta que pode te auxiliar para averiguar se você está realizando essas operações corretamente é o *valgrind*, uma ferramenta de linha de comando muito poderosa que pode ser usada para te informar sobre o andamento do seu código, sinalizando se você alocou e desalocou memória da maneira correta.

Nota

A nota do trabalho será contabilizada da seguinte maneira:

1. Inserção e remoção - 2 pontos
2. Busca - 1 ponto
3. Impressão - 1 ponto
4. Estrutura da main - 1 ponto
5. Alocação e limpeza de memória - 2 pontos
6. Casos de teste - 3 pontos

Atenção: Não esqueça de **comentar seu código** de maneira concisa e clara!

Entrega

A entrega deve ser feita até o último minuto (23:59) do dia 28/11.