Universidade Federal de Mato Grosso do Sul
Faculdade de Computação (FACOM)
Algoritmos e Programação 2
Prof. Dr. Anderson Bessa da Costa

Lista 9 - Listas Encadeadas

Nesta primeira etapa, faremos um passo a passo para construirmos juntos uma lista encadeada. Resolva, em ordem, as questões abaixo.

1. Uma `lista encadeada` é uma sequência de registros que chamaremos de células. Implementaremos uma lista simplesmente encadeada, ou seja, uma lista onde cada célula possui um único ponteiro que aponta para a célula seguinte. Suporemos aqui que os objetos armazenados nas `células` são do tipo `int`. A estrutura das células pode, então, ser definida como:

```
// Definicao de tipo
struct cel {
  int conteudo;
  struct cel *seg;
};

// A partir de agora usaremos apenas o nome celula
typedef struct cel celula;
```

2. Uma lista encadeada pode ser vista de duas maneiras, dependendo do papel que sua primeira célula desempenha: i) lista com cabeça; e ii) lista sem cabeça. Trataremos preferencialmente listas com cabeça, pois são mais fáceis de manipular.

   Na lista com cabeça, a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante. A primeira célula é a cabeça da lista. Se `lst` é o endereço da cabeça então `lst->seg` vale `NULL` se e somente se a lista está vazia. Para criar uma lista vazia deste tipo, podemos fazer como a seguir. Adicione o código da função `main` abaixo da definição de tipo feita no exercício 1.

```
// Definicao de tipo
...

int main() {
  // Crie uma lista encadeada com cabeca
  celula c, *lst;

  // Inicialize a lista como vazia
  c.seg = NULL;

  // lst recebe o endereco da cabeca
  lst = &c;

  return 0;
}
```

3. Suponha que desejamos inserir uma nova célula com conteúdo `y` entre a célula apontada por `p` e a seguinte (ou seja, `p->seg`). É claro que isso só faz sentido se `p` for diferente de `NULL`. Adicione a função a seguir (não esqueça de escrever o protótipo antes da `main`):

```c
/* A funcao insere uma nova celula em uma lista encadeada
 * entre a celula p e a seguinte (supoe-se que p != NULL).
 * A nova celula tera conteudo y. */
void insira(int y, celula *p) {
  celula *nova;

  nova = malloc(sizeof(celula));
  nova->conteudo = y;
  nova->seg = p->seg;
  p->seg = nova;
}
```

4. Agora que você implementou a função de inserção, você consegue testar de fato a sua lista encadeada. Adicione o seguinte trecho dentro da `main`, logo após a criação e inicialização da lista encadeada.

```c
...
int main() {
  // Criacao e inicializacao
  ...

  // Sequencia de insercoes
  insira(1, lst);
  insira(3, lst);
  insira(5, lst);
  insira(7, lst);

  return 0;
}
```

5. Para verificar se o código até o momento está funcionando como esperado, adicione ao seu código a função a para impressão da lista. Lembre-se que essa função funciona especificamente para listas com cabeça (assim como o código `insira`).

```c
void imprima(celula *lst) {
  celula *p;

  printf("cabeca -> ")
  for (p = lst->seg; p != NULL; p = p->seg)
    printf("%d -> ", p->conteudo);
  printf("NULL\n");
}
```

6. Desta forma você consegue utilizar a função `imprima` para verificar se o seu código está funcionando corretamente. Chame-a dentro da `main`. Confira se a saída obtida é igual a saída esperada.

```c
...
int main() {
  ...

  // Sequencia de insercoes
  insira(1, lst);
  insira(3, lst);
  insira(5, lst);
```

```
  insira(7, lst);

  // Imprima a lista na tela
  imprima(lst);

  return 0;
}
```

A saída esperada é:

```
cabeca -> 7 -> 5 -> 3 -> 1 -> NULL
```

7. Um recurso importante para melhor compreender listas encadeadas é a vizualização. Você pode executar o passo a passo e tentar visualizar o que está acontecendo utilizando papel e lápis. Entretanto, existem também ferramentas próprias para auxiliá-lo. Copie o código desenvolvido e o cole neste site: `https://pythontutor.com/cpp.html#mode=edit`. Com o `Python Tutor` você conseguirá visualizar o passo a passo do seu código C++ (apesar do `Python` no nome), o ajudando na compreensão de listas encadeadas. Execute o passo a passo no código.

8. Crie uma função denominada `insira_em_ordem`, onde como o próprio nome indica insere um elemento em ordem. Essa função pode fazer uso da função `insira` já implementada. Altere as chamadas na `main` de `insira` para `insira_em_ordem`. Certifique-se de que a saída obtida é idêntica a saída esperada apresentada.

```
...
int main() {
  ...

  // Sequencia de insercoes em ordem
  insira_em_ordem(5, lst);
  insira_em_ordem(1, lst);
  insira_em_ordem(3, lst);
  insira_em_ordem(7, lst);

  // Imprima a lista na tela
  imprima(lst);

  return 0;
}
```

A saída na tela esperada é:

```
cabeca -> 1 -> 3 -> 5 -> 7 -> NULL
```

9. Uma operação sempre importante em listas encadeadas é a busca. Implemente a função `busque`, que recebe um inteiro `x` e uma lista encadeada `p` e retorna um ponteiro para célula que contém `x` como o seu conteúdo. Se tal célula não existir, retorne `NULL`. Teste o seu código.

10. Implemente a função `remova_seguinte`, que possui um único parâmetro `celula *p`, e remove a célula seguinte à `p`. Este código está disponível nos slides da aula, porém tente desenvolver primeiro antes de consultar. Teste o seu código.

11. Agora, desenvolva a função `void remova(int x, celula *lst)`. Esta função deve buscar e remover da lista encadeada `lst` a célula que possui o conteúdo `x` (se tal célula existir). É possível implementá-la fazendo uso da função `remova_seguinte`.

12. Por fim, crie a função `remova_tudo`, responsável por remover todas as células da lista encadeada.

13. Provavelmente, a sua implementação de `remova_tudo` é iterativa. Crie uma nova função `remova_tudoR` com uma implementação recursiva.