BCC202 – Estruturas de Dados I (2018-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG

Trabalho Prático 02

Data de entrega: 10/11/2018 até 22:00.
Atenção: *O que vale é o horário do RunCodes, e não do* seu*, ou do* meu*, relógio.*

Procedimento para a entrega:.

1. Para cada questão, implemente sua solução.
2. Para testar suas soluções, implemente um único método *main()*, que poderá conter, por exemplo, um *Menu* de interações e possibilidades do usuário especificar os dados de entrada.
3. Especifique o *Makefile* com as instruções necessárias para compilação e execução do seu código, sendo que o *Makefile* deve conter também o redirecionamento da entrada e da saída (e.g., **`./prog.exe < input.txt > output.txt`**.)
4. Compacte em um único diretório o seu código fonte juntamente com o *Makefile*, o arquivo de entrada e o arquivo de saída usados para testes (e.g., *input.txt, output.txt*).
5. Faça a entrega do arquivo compactado, obrigatoriamente em formato *.zip*, no *RunCodes*, na tarefa correspondente.

- **Não utilize caracteres acentuados ou especiais para nomes de pastas, arquivos e na especificação de comentários no código**.
- **Especifique funções para modularizar seu código.**

- Bom trabalho!

Questão 01 (3 pontos)

A representação de informação por um ponteiro nos permite simular listas, pilhas e filas heterogêneas, isto é, as informações armazenadas podem diferir de nó para nó. Como exemplo, vamos considerar uma aplicação que necessite manipular filas de objetos geométricos planos para cálculo de áreas. Para simplificar, vamos considerar que os objetos podem ser apenas retângulos, triângulos ou círculos. Nesse contexto, deve ser definido um tipo para cada objeto representado (*i.e.,* retângulo, triângulo e círculo). O nó da fila, por sua vez, deve então ser composto por três campos:

- um identificador de qual objeto está armazenado no nó;
- um ponteiro para a estrutura que contém a informação;
- um ponteiro para o próximo nó da fila.

É importante salientar que, a rigor, a fila é homogênea, no sentido de que todos os nós contêm os mesmos campos. O ponteiro para a informação deve ser do tipo genérico, pois não sabemos a princípio para qual estrutura ele irá apontar: pode apontar para um retângulo, um triângulo ou um círculo. Um ponteiro genérico em C/C++ é representado pelo tipo **void ***. Uma variável do tipo ponteiro genérico pode representar qualquer endereço de memória, independente da informação de fato armazenada no espaço. No entanto, de posse de um ponteiro genérico, não podemos acessar a memória por ele apontada, já que não sabemos o tipo de informação armazenada. Por esta razão, o nó de uma fila genérica deve guardar explicitamente um identificador do tipo de objeto de fato armazenado. Consultando esse identificador, podemos converter, utilizando uma variável auxiliar e ***casting***, o ponteiro genérico no ponteiro específico para o objeto em questão e, então, acessar campos do objeto. Como identificador de tipo, podemos usar uma enumeração (pesquise pelo conceito de ***enum*** em C/C++). Assim, na criação de nós, armazenamos o identificador de tipo correspondente ao objeto a ser representado.

Considerando os conceitos acima expostos, implemente uma fila dos objetos geométricos mencionados utilizando ***enum***. Sua fila deve contemplar as seguintes operações.

- Criar fila vazia.
- Inserir um elemento ao final da fila.
- Remover um elemento do início da fila.
- Verificar se a fila está vazia.
- Liberar a estrutura de fila.
- Calcular a área de todos os objetos geométricos armazenados na fila.

Implemente também as operações para criar, ler e atualizar cada uma dos objetos geométricos planos. É possível considerar o uso de uma lista encadeada na implementação da estrutura fila, mais precisamente, reúso do TAD **ListaNo**. Outra possibilidade, seria implementar a fila a partir do "zero", utilizando ponteiros e encadeamento de nós. Não serão aceitas soluções de filas usando vetores.

Questão 02 (4 pontos)

Os estudantes de computação são convidados a desenvolver um sistema para o registro de pratos gastronómicos e os seus ingredientes. O computador onde será implantado este programa não tem tamanho de memória suficiente e, portanto, o sistema deve aproveitar deste recurso da melhor forma. No início, o número de pratos é conhecido, mas este número pode aumentar ou diminuir dependendo das vendas. O número de ingredientes em cada prato também pode variar. Portanto, usar uma representação em forma de matriz de tamanho fixo, para armazenar pratos e seus ingredientes, não é viável pois teríamos espaços de memória não usados.

Uma solução proposta foi utilizar listas encadeadas para simular uma matriz dinâmica de modo que uma lista encadeada vertical representa os pratos; cada prato possui dois campos: código, do tipo *int*, e nome, do tipo *string*. Por sua vez, cada elemento do tipo prato está associado com uma lista encadeada horizontal, que contém os ingredientes de tal prato; novamente, cada ingrediente possui dois campos: código, do tipo *int*, e nome, do tipo *string*, tal como representado a seguir.

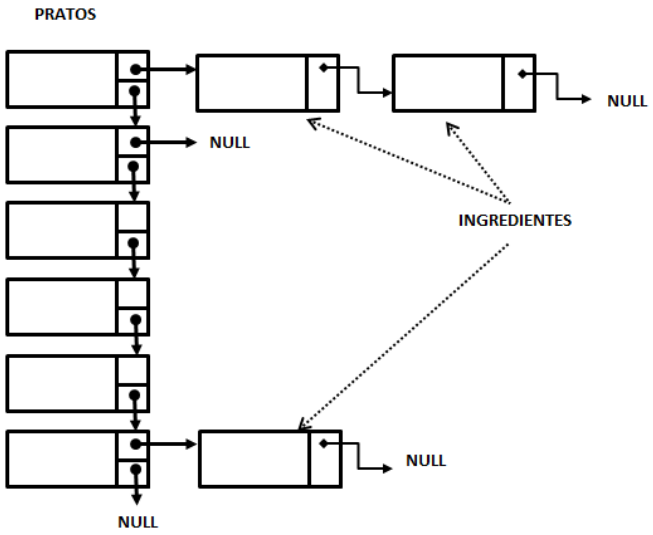

Figura 1: Encadeamentos: pratos e seus ingredientes

Implemente e execute as seguintes funções:

- Adicionar um prato com todos os seus ingredientes.
- Apagar um prato com todos os seus ingredientes
- Dado o código de um prato, imprimir a sua lista de ingredientes.
- Dado o código de um prato, adicionar um ingrediente.

- Dado o código de um prato, eliminar qualquer ingrediente mediante seu código ou nome (primeiro terá que ser feita a validação para saber se o ingrediente existe).
- Dado o código de um prato, imprimir a sua lista de ingredientes.
- Imprimir a lista completa de pratos e ingredientes

Para este exercício, pode ser utilizado qualquer tipo das listas estudadas nas aulas teóricas. As operações previamente mencionadas devem estar disponíveis num *menu* de interações.

Questão 02 (3 pontos)

Considere que uma empresa quer ter acesso às datas dos aniversários de seus funcionários. Cada funcionário, no vetor, será identificado pelo nome (*char[81]* ou *string* nome) e dia/mês (*int*) de aniversário.

- Implemente uma função que receba um vetor de ponteiros para o tipo Funcionário e ordene o vetor em ordem crescente de aniversário usando o método *QuickSort* iterativo. Ou seja, use como critério de ordenação o mês; para valores de mês iguais, use o dia. Se dois ou mais funcionários fizerem aniversário na mesma data, use a ordem alfabética do nome como critério de desempate. Sua função deve receber o número de funcionários e o vetor de ponteiros como parâmetros.
- Implemente uma função que receba um vetor de ponteiros para os funcionários, ordenado conforme o critério citado, e exiba na tela os nomes dos aniversariantes do mês especificado. Sua função deve receber como parâmetros o número de funcionários, o vetor de ponteiros (já ordenado) e o mês que deseja consultar. Utilize o algoritmo de busca binária para procurar um funcionário que faça aniversário no mês especificado. A partir desta posição, percorra os elementos vizinhos no vetor para achar o início e o fim dos funcionários que fazem aniversário no mesmo mês, e exiba seus nomes na tela, em ordem do dia de aniversário.
- Para testar sua solução, implemente uma função que deve definir um vetor de ponteiros para funcionários com dados numa ordem qualquer, chama a função de ordenação, e, então, chama a função para exibir os aniversariantes de uma data.
- Discuta, num arquivo *.pdf*, o funcionamento do *QuickSort* iterativo.

Referência Bibliográfica

- Exercícios parcialmente extraídos do livro: *Introdução a Estrutura de Dados: Com técnicas de programação em C - Waldemar Celes, Renato Cerqueira, José Lucas Rangel. Editora Campus. Elsevier 2016 (Segunda Edição).*
- Material complementar sobre listas/pilhas/filas "heterogêneas":
http://www.decom.ufop.br/anascimento/ensino/bcc202/aulas-teoricas/