Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Estrutura de Dados I
Prof. Oberlan Romão

Pilhas e Filas

Atenção: Ao terminar, não se esqueça de enviar as soluções no AVA.

Pilha

Como vimos na aula teórica, uma pilha é uma estrutura do tipo *LIFO* (do inglês, *last in, first out* - "o primeiro que sai é o último que entrou"). Geralmente, uma pilha é muito útil em situações em que os dados têm que ser armazenados e então recuperados na ordem inversa. Uma aplicação de pilha é a verificação do "casamento" de símbolos (delimitadores) de um programa. Por exemplo, nas sequências "`(())`" e "`()()`" os parênteses aparecem de maneira balanceada, enquanto nas sequências "`(()))`" e "`)(`" esse balanceamento não acontece. Nesse link, você encontra mais informações sobre parênteses balanceados.

Considerando que os delimitadores aceitos em um programa são parênteses "(" e ")", colchetes "[" e "]" e chaves "{" "}", as declarações abaixo usam apropriadamente os delimitadores:

```
a = a + (c + d) * (e - f);
g[10] = h[i[9]] + (j + k)*1;
while (m < (n[8] + o) ) { p = (7 + 2) * 3; }
```

Já as declarações abaixo são exemplos do uso incorreto dos delimitadores (não ocorre o "casamento"):

```
a = a + (c + d) * (e - f));
g[10] = h[i[9]] + (j + k*1;
while (m < (n[8) + o] ) { p = (7 + 2) * 3; }
```

Um delimitador em particular pode ser separado a partir de seu par por outros delimitadores, isto é, os delimitadores podem ser aninhados. Em consequência, um delimitador em particular está casado somente depois que todos os delimitadores que o seguem e que o precedem tenham sido casados.

O algoritmo de casamento de delimitador lê um caractere e o empilha em uma pilha se for um delimitador de abertura. Se um delimitador de fechamento é encontrado, ele é comparado a um delimitador que está no topo da pilha. Se eles se casam, o processamento continua. Caso contrário, o processamento para, assinalando um erro. O processamento termina com sucesso depois que todos os caracteres forem analisados e a pilha estiver vazia.

Baixe o arquivo `Codigos.zip` e extraia em alguma pasta. Analise o código presente na pasta `ParentesesBalance`. Esse código verifica se uma sequencia de parênteses está ou não bem balanceada (seguindo o algoritmo anterior). Note que a implementação da pilha foi feita usando lista encadeada.

Exercício 1: Implemente a função `verificaDelimitadores` (arquivo `main.c`) que está na pasta `Exercicio1`. Essa função deve receber e validar (ou não) uma sequência de delimitadores. Note que não existem os arquivos `Pilha.h` e `Pilha.c`, você pode usar/modificar algum dos que foram passado na aula teórica (usando lista ou vetor) ou implementar o seu próprio código.

Notação pós-fixa

Na notação tradicional (notação infixa) de uma expressão matemática temos que o operador aparece entre os seus dois operandos. Já na notação pós-fixa, também conhecida como notação polonesa inversa, o operador é colocado após os seus dois operandos. Por exemplo, a expressão infixa $(1 - 2) * (4 + 5)$ é equivalente a expressão pós-fixa $1\ 2 - 4\ 5 +\ *$.

Note que os operandos aparecem na mesma ordem na expressão infixa e na correspondente expressão pós-fixa. Note também que a notação pós-fixa dispensa parênteses e regras de precedência entre operadores (como a precedência de $*$ sobre $-$ por exemplo), que são indispensáveis na notação infixa. Na notação pós-fixa, a ordem dos operadores na expressão diz a ordem em que eles vão ser executados (da esquerda para a direita). Isso facilita o calculo do valor de uma expressão.

Exercício 2: Transforme as expressões infixa abaixo em pós-fixa:

- $10 - 20$
- $13 - 2 * 5$
- $(1 + 3) * 2$
- $(1 - 2 + 3) * (4 + 5)$
- $1 + 2/2$
- $(1 + 2)/2$

Exercício 3: Descreva os passos (pseudo-código) de um algoritmo que recebe uma expressão em notação pós-fixa e calcula o resultado da expressão. Pense apenas nos operadores básicos $+$, $-$, $*$ e $/$.

Desafio: Implemente esses algoritmos :)

Deque

Deque (do inglês, *double-ended queue*, fila com duas extremidades ou fila dupla) é uma generalização de fila na qual podemos inserir novos elementos em ambas as extremidades, no início e no fim. Consequentemente, também podemos retirar elementos de ambos os extremos. Dessa forma, um TAD que representa uma fila dupla deve possuir as seguinte funções:

- Criar uma estrutura de fila dupla
- Liberar a fila
- Inserir um elemento no início (`push_front`)

- Inserir um elemento no fim (`push_back`)
- Retirar o primeiro elemento (`pop_front`)
- Retirar o último elemento (`pop_back`)
- Acessar o primeiro elemento (`front`)
- Acessar o último elemento (`back`)
- Verificar se a fila está vazia

É importante notar que, embora a *deque* possa assumir muitas das características de pilhas e filas, não requer a ordenação LIFO nem a FIFO que são impostas para essas estruturas de dados. É sua responsabilidade fazer uso consistente das operações de inserção e remoção.

Exercício 4: Complete o código da pasta `Deque`. Como você pode notar, a implementação deve ser feita usando listas (e não vetor). Qual tipo de lista é mais recomendável utilizar nessa estrutura? Por quê?

Fila de prioridade

Uma fila de prioridade é um tipo abstrato de dado para armazenar uma coleção de elementos priorizados em que a inserção de elementos é feita de forma arbitrária, mas a remoção (ou acesso) é feita em ordem de prioridade, ou seja, o elemento com maior prioridade é removido (ou acessado) primeiro. O termo "maior prioridade" pode ter significados diferentes, dependendo da aplicação. Esta estrutura é utilizada em aplicações onde se devem processar elementos por ordem de prioridade. Como exemplo, pode-se representar tarefas que um computador (ou sistema operacional) deve processar por ordem de importância. Pode-se também representar a fila espera em um hospital: a prioridade é para os pacientes "mais críticos" em invés de ser na ordem de chegada.

Independente do tipo de implementação utilizada para representar a fila de prioridade, as funções fundamentais de uma fila de prioridade são as seguintes:

- Criação e destruição da fila;
- Inserção de um elemento na fila com prioridade;
- Remoção de um elemento da fila com maior prioridade;
- Acesso ao elemento do início da fila (que possui maior prioridade).

Pergunta: Como você implementaria uma fila de prioridade? Qual a complexidade de cada uma das funções anteriores?

Não é difícil imaginar maneiras de implementar uma fila de prioridades. Infelizmente, nas implementações mais óbvias, algumas das operações ficam rápidas, mas as outras ficam lentas. O desafio está em elaborar uma implementação em que todas as operações sejam rápidas.

Exercício 5: Complete o código da pasta `FilaPrioridade`. Para simplificar, considere que cada elemento da fila será um `int` que indica a prioridade (quanto maior, maior a prioridade).

Pergunta: Como você usaria uma fila de prioridade para ordenar valores?