Notas de aula – AED 1 – pilhas estáticas
Prof. Jefferson T. Oliva

Na aula anterior, vimos conceitos e aplicações de listas. O foco da aula foi em listas estáticas.

Lista é uma estrutura de dados que implementa uma coleção de elementos (como structs) organizados em sequência. Essa estrutura agrupa informações cujos elementos são organizados de alguma forma.

Os itens da lista estática são armazenados em posições contíguas da memória (vetores/alocação estática): lista->item[i].

Quais são as vantagens e desvantagens das listas estáticas?
- Vantagens: simples implementação e economia de memória (é definido um limite)
- Desvantagens: custo para inserir e retirar itens (podendo deslocar n ou n-1 elementos) e não há previsão de crescimento da lista (pode exigir realocação de memória)

Na aula também foi implementada um TAD (código disponível no Ownclould) para listas estáticas:

Lista criar();*

*int vazia(Lista *l);*

*int cheia(Lista *l);*

*void liberar(Lista *l);*

*void imprimir(Lista *l);*

*int buscar(int key, Lista *l);*

*int inserir(int key, Lista *l);*

*int remover(int key, Lista *l);*

*int copiar(Lista *l1, Lista *l2);*

*int concatenar(Lista *l1, Lista *l2);*

*int intecalar(Lista *l1, Lista *l2, Lista *l3);*

O tema da aula de hoje é um tipo especial de lista: pilha

Pilha

É uma lista linear em que os elementos são inseridos em uma de suas extremidades.

Conhecida como LIFO (last-in, first-out – último a entrar, primeiro a sair).

Em outras palavras, a inserção e a remoção é sempre no topo da estrutura.

Todas as operações em uma pilha podem ser imaginadas como as que ocorre em uma pilha de pratos ou no deck (baralho) no jogo Yu Gi Oh. No em um jogo de baralho, ao sacar uma carta, sempre será a do topo.

Principais operações: criar, verificar se a pilha está cheia, verificar se a pilha está vazia, empilhar, desempilhar, verificar o item que está no topo.

Aplicações
- Avaliação de expressões numéricas
- Processamento de linguagens (compilação): por exemplo, análise sintática
- Conversão de número decimal para binário (isso vale para outras bases numéricas)
- Mecanismo de fazer/desfazer em editores de texto
- Mecanismo de avançar/retornar em páginas web
- Execução de programas

Representação de pilhas: alocação estática e alocação dinâmica.

Na apresentação da aula de hoje, entre os slides 12 e 21 é apresentado um exemplo de aplicação de pilha para a execução de programas.

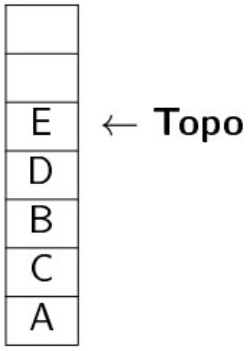
Para simplificação, cada chamada de função foi representada por um retângulo na pilha.

Na realidade, cada chamada é composta por um frame que contém os parâmetros da função, suas variáveis locais e endereço de retorno.

Pilhas Estáticas

Assim como em listas, a implementação de pilhas estáticas é também feita por uso de vetores.

Um cursor (número inteiro) é utilizado para controlar a posição do topo. Enquanto em uma lista é utilizada uma variável para controlar o seu tamanho, na pilha é utilizada uma variável para armazenar a posição no topo.



Durante a criação da pilha, a variável que indica o topo é inicializada com -1, indicando que a pilha está vazia.

Na operação empilhar, primeiramente é verificada se a pilha está cheia (vocês já deparam com o problema *Pilha overflow*?). Caso não esteja cheia, a variável que controla o topo é incrementada e um item é adicionado no topo.

Na operação desempilhar, primeiramente é verificada se a pilha está vazia (Pilha underflow). Caso não esteja vazia, a função deverá retornar o item no topo e decrementar a variável que controla o topo.

TAD Pilhas Estáticas

Operações básicas para uma pilha
- Criar uma pilha
- Verificar se a pilha está vazia
- Verificar se a pilha está cheia
- Empilhar
- Desempilhar
- Imprimir pilha
- Liberar a pilha

Implementações estão disponibilizadas no Ownclould

Expressões Matemáticas

Pilhas são muito usadas no processamento de linguagens (compiladores) e durante a execução de programas (operações de voltar em editores de texto).

Uma das aplicações interessantes é a conversão e a avaliação de expressões algébricas/numéricas:
- Consistência de parênteses: verificar a existência de fechamento de parênteses para cada abertura
- Notação infixa: o operador está entre operandos (A + B)
- Notação pré-fixa (polonesa): o operador precede os operandos (+AB)
- Notação pós-fixa (polonesa inversa): o operador procede os operandos (AB+)

Consistência de parênteses
- é verificado se uma expressão algébrica está correta quanto a parentização, ou seja, se para cada "(" há o ")"
- a função recebe uma string que representa uma expressão algébrica. Quando "(" é lido, o mesmo é empilhado. Quando ")" é lido, "(" é desempilhado. Assim, para uma expressão incorreta pode ocorrer dois cenários: stack underflow ou a pilha ainda tem itens. Em outras palavras, para um expressão estar correta, no final da execução da função, a pilha deve estar vazia e não deve haver ocorrência de stack underflow.

Notação infixa: é a notação convencional (A + B * C, A /C, etc). Portanto, pode ser necessário o uso de parênteses.

Na notação pré-fixa, os operadores vêm antes dos operandos. Essa notação determina os operadores e a respectiva ordem para o cálculo de uma expressão. Também, nessa notação não é necessário o uso de parênteses.

Exemplos infixo × pré-fixo
→ A + B − C : + − ABC
→ (A + B) ∗ C : + ∗ ABC
→ A + B ∗ C : +A ∗ BC
→ A ∗ B − C/D : − ∗ AB/CD
→ (A+ B) *D + E/(F + A * D) + C: +*+A B D +/ E+F * A D C

Na notação pós-fixa (polonesa reversa), os operadores são posicionados após os operandos e é utilizado atualmente em vários equipamentos eletrônicos (calculadoras e computadores) devido a sua simplicidade. Nessa notação, os operadores aparecem na ordem em que devem ser calculados.

Exemplos infixo × pós-fixo
→ A + B − C : ABC + -
→ (A + B) ∗ C : AB + C ∗
→ A + B ∗ C : ABC ∗ +
→ A ∗ B − C/D : AB ∗ CD/−
→ (A+ B) *D + E/(F + A * D) + C: AB+ D * EF A D*+ /+ C+

Implementação do processador de expressões na notação pós-fixa
1 – Cada operando é colocado na pilha
2 – Processamento de cada operador
2.1 – Dois operandos são desempilhados
2.1 – A operação é executada
2.1 – O resultado é empilhado
3 – O resultado da expressão é retornado

Exemplo de processamento para a expressão 7 − (6 + 2) / 4 + 3
- Na forma pós-fixa: 762+4/-3+

Conversão de infixa para pós-fixa
- Deve ser utilizada uma pilha
- A expressão infixa deve ser percorrida da esquerda para a direita
→ Se um operando é encontrado, o mesmo é colocado na saída
→ Se um operador é encontrado, o mesmo é colocado na pilha, desempilhando e colocando na saída os operadores na pilha até encontrarmos um operador com precedência menor: Precedência: + e -, seguida por * e /, seguida por ^

Exemplo: A - B * C + D

Entrada	Pilha	Saída
A		A
-	-	A
B	-	A B
*	- *	A B
C	- *	A B C
+	+	A B C * -
D		A B C * - D +

infixa	**pós-fixa**
a-b	ab-
a-b*c	abc*-
(a-b)*c	ab-c*
a+b*c^d-e	abcd^*+e-
a*(b+c)*(d-g)*h	abc+*dg-*h*
a*b-c*d^e/f+g*h	ab*cde^*f/-gh*+

Conversão de infixa para pós-fixa
- Caso seja encontrada uma abertura de parênteses, a mesma deve ser colocada na pilha
- Se o fechamento de parênteses for encontrado, os operadores são desempilhados e copiados na saída, até a abertura de parênteses correspondente
- Ao final, os operadores restantes são desempilhados e colocados na saída

Parênteses:
- Uma nova pilha pode ser criada para cada abertura de parênteses: Cada nova pilha deve ser operada até o fechamento do parênteses correspondente.
- Isso também pode ser feito usando apenas uma única pilha, simulando a abertura e o fechamento de outras pilhas no seu interior

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. Third edition, The MIT Press, 2009.

Pereira, S. L. Estrutura de Dados e em C: uma abordagem didática. Saraiva, 2016.

Szwarcfiter, J.; Markenzon, L. Estruturas de Dados e Seus Algoritmos. LTC, 2010.

Tenenbaum, A.; Langsam, Y. Estruturas de Dados usando C. Pearson, 1995.

Ziviani, M. Projetos de Algoritmos: com implementações em Pascal e C. Thomson, 2004.