Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: Algoritmos e Estrutura de Dados I – CIC102 / 20092
Professor: David Menotti (menottid@gmail.com)

Trabalho Prático 2 – Mundo dos Blocos
Alocação Dinâmica / Listas Encadeadas

Valor: 0,5 pontos (5% da nota total) **Data de entrega: 06/10/2009**
Documentação não-Latex: -0,1 pontos
Impressão não-frente-verso: -0,05 pontos

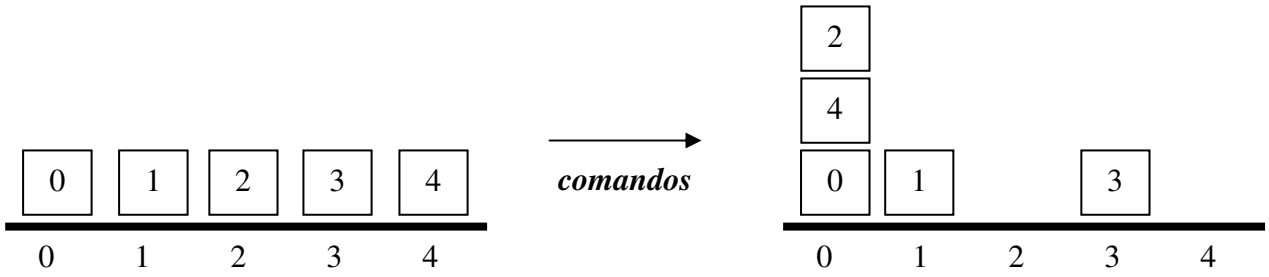
Objetivos

Consiste em praticar os conceitos de alocação dinâmica de memória, apontadores e listas encadeadas. Esse trabalho é baseado em um dos problemas da Maratona de Programação da ACM [1].

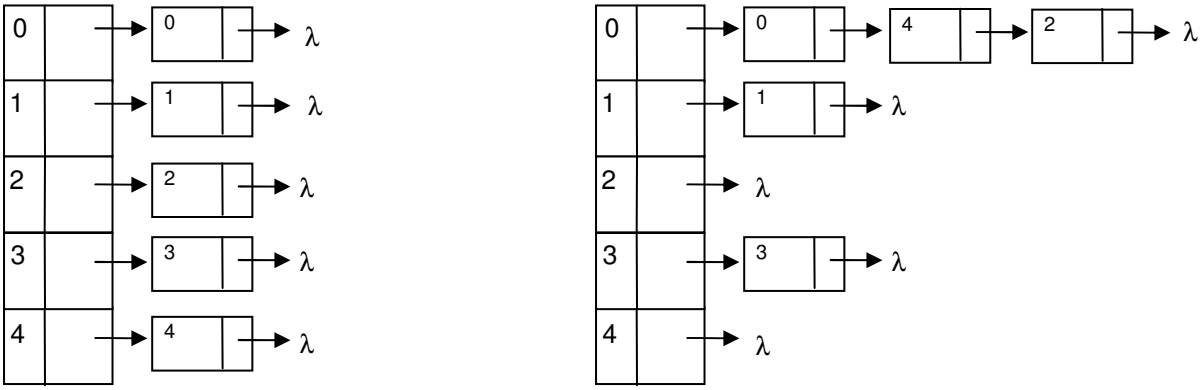
Descrição

Várias áreas da ciência da computação usam domínios simplificados para abstrair diversos tipos de problemas. Por exemplo, algumas das primeiras pesquisas de inteligência artificial nas áreas de planejamento e robótica eram feitas utilizando o "mundo dos blocos", no qual um braço robótico realizava tarefas simuladas envolvendo a manipulação de blocos [2].

Nesse trabalho você vai implementar um **mundo de blocos** bem simples [3], que vai funcionar de acordo com certas regras e obedecer comandos de movimentação de blocos dados pelo usuário, simulando o que seria a manipulação através um braço robótico. O seu mundo de blocos começa com cada bloco na sua posição inicial, e depois de uma série de comandos deve terminar em uma configuração final, como mostrado da figura abaixo para 5 blocos (n=5).

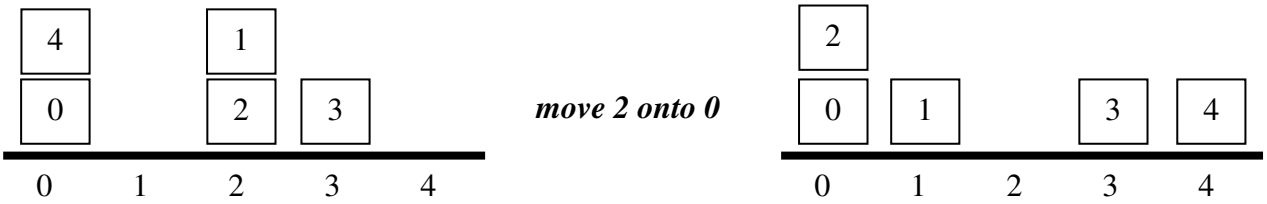


Para implementar o seu mundo de blocos, você vai criar um tipo abstrato de dados chamado **`TBlocos`** que é composto por um vetor de listas encadeadas (onde cada lista corresponde a um monte de blocos e cada célula vai corresponder a um bloco) e pelas funções para a manipulação dessa estrutura. Por exemplo, as duas configurações mostradas acima estariam implementadas da seguinte forma utilizando o seu TAD:

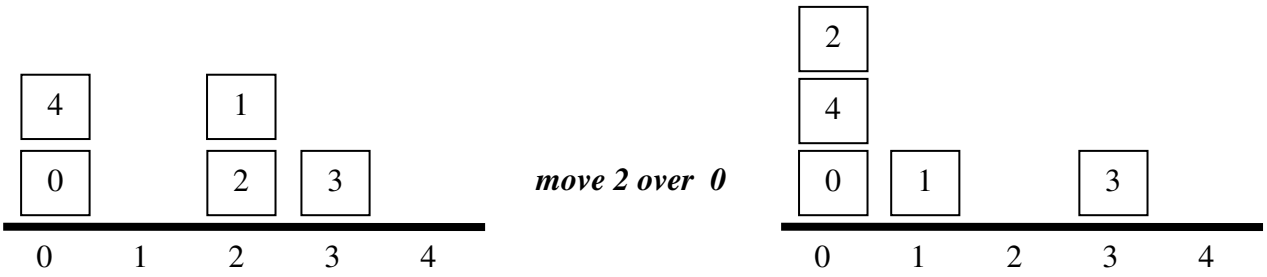


Os comandos possíveis para a manipulação do seu mundo de blocos são:

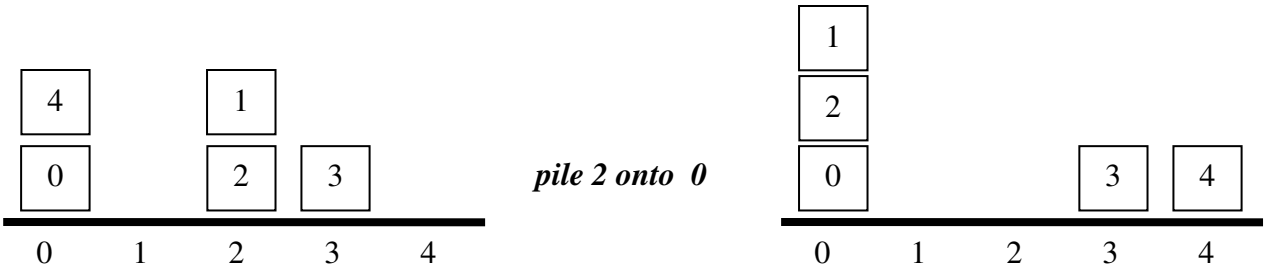
- **move *a* onto *b***: Move o bloco *a* para cima do bloco *b* retornando eventuais blocos que já estiverem sobre *a* ou *b* para as suas posições originais.



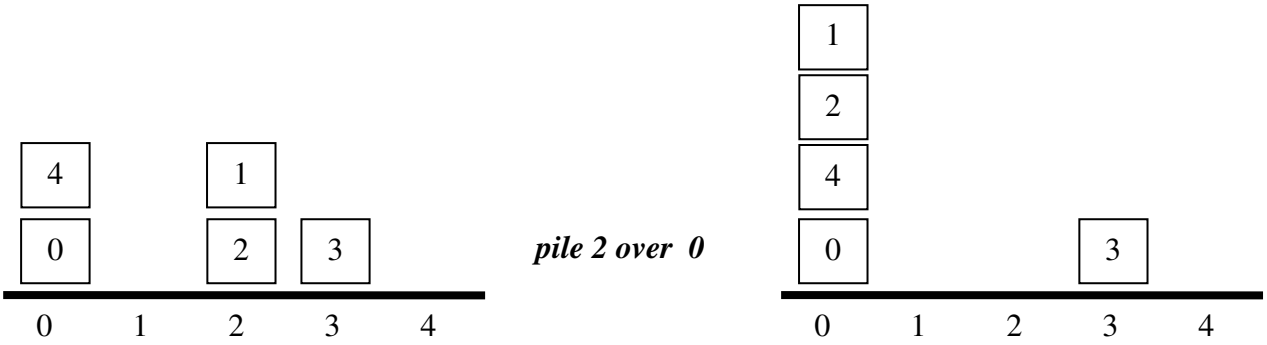
- **move *a* over *b***: Coloca o bloco *a* no topo do monte onde está o bloco *b* retornando eventuais blocos que já estiverem sobre *a* às suas posições originais.



- **pile *a* onto *b***: Coloca o bloco *a* juntamente com todos os blocos que estiverem sobre ele em cima do bloco b, retornando eventuais blocos que já estiverem sobre *b* as suas posições originais.



- **pile *a* over *b***: Coloca o bloco *a* juntamente com todos os blocos que estiverem sobre ele sobre o monte que contem o bloco b.



- **quit**: termina a execução.

Comandos onde $a = b$ ou onde a e b estejam no mesmo monte devem ser ignorados.

Além de funções para a implementação desses comandos, crie funções auxiliares no seu TAD para simplificar a implementação e evitar a repetição de código. Por exemplo, uma função para retornar um ou mais blocos para as suas posições originais vai ser utilizada por quase todos os comandos.

O seu programa principal deverá ser basicamente um *parser* que lê comandos a partir de um arquivo de entrada, executa os comandos no mundo dos blocos e imprime a saída (configuração final do mundo de blocos) para um arquivo. O programa pode ainda imprimir a representação do mundo dos blocos após cada comando. Os arquivos de entrada e saída devem obedecer os seguintes formatos:

- **Arquivo de entrada:** é composto pelo número *n* de blocos na primeira linha, uma sequência de comandos (um por linha) e o comando *quit* na última linha. Você pode considerar que não haverá erros de sintaxe no arquivo, ou seja, os comandos vão estar escritos corretamente.
- **Arquivo de saída:** o arquivo de saída vai representar a configuração final do seu mundo de blocos. Em cada linha deve estar o número da posição original, seguida de um dois pontos (:) e a lista de blocos que estão naquela posição com um espaço entre eles. Esse formato deve ser rigorosamente seguido.

Para o primeiro exemplo mostrado, os arquivos de entrada e saída são listados abaixo. Note que vários arquivos de entrada podem levar a mesma configuração final.

Entrada:	Saída:

`5`
`move 3 onto 0`
`move 2 over 4`
`pile 4 onto 0`
`quit`
	`0: 0 4 2`
	`1: 1`
	`2:`
	`3: 3`
	`4:`

Você pode testar arquivos de entrada e saída utilizando um dos vários *problem solvers* disponíveis para problemas da maratona de programação. Por exemplo, acesse http://www.uvatoolkit.com/problemssolve.php clique no problema 101 *The Blocks Problem* e teste arquivos de entrada e saída.

Para executar seu programa, use parâmetros de linha de comando para fazer sua chamada (faz parte do trabalho descobrir como isso é feito em C). Esse tipo de execução é bastante comum em sistemas Unix / Linux e no antigo DOS. Por exemplo, se o seu programa chama-se **TP2** e você quiser executar o programa com a entrada `arq1.txt` gerando a saída para o arquivo `saida.txt`

```
> TP2 arq1.txt saida.txt
```

O que deve ser entregue

- Código fonte do programa em C ou C++ (bem identada e comentada).
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
  1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  3. Análise de Complexidade: estudo da complexidade de tempo e espaço das funções implementados e do programa como um todo (notação O).
  4. Listagem de testes executados: os testes executados devem ser simplesmente apresentados.
  5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso. Uma referência bibliográfica deve ser citada no texto quando da sua utilização
  7. Em Latex: Caso o trabalho não seja elaborado/escrito em latex, perde-se 0,1 pontos.
  8. Impressão: Caso o trabalho não seja impresso usando modo frente-verso, perde-se 0,05 pontos.

9. Formato: mandatoriamente em PDF (http://www.pdf995.com/).

Obs: Veja modelo de como fazer o trabalho em latex:
http://www.decom.ufop.br/prof/menotti/aedI092/tps/modelo.zip

Como deve ser feita a entrega:

A entrega DEVE ser feita via Moodle (www.decom.ufop.br/moodle) na forma de um **único** arquivo zipado, contendo o código, os arquivos e a documentação. Também deve ser entregue a documentação impressa na próxima aula teórica após a data de entrega do trabalho.

Comentários Gerais:

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- Clareza, identação e comentários no programa também vão valer pontos;
- O trabalho é individual (grupo de UM aluno);
- Trabalhos copiados (e FONTE) terão nota zero;
- Trabalhos entregue em atraso serão aceitos, todavia a nota atribuída ao trabalho será zero;
- Evite discussões inócuas com o professor em tentar postergar a data de entrega do referido trabalho.

Referências

[1] *Maratona de Programação da ACM* - http://maratona.ime.usp.br/

[2] *Artificial Intelligence – A Modern Approach*. Stuart Russell and Peter Norvig, 2nd Edition, Prentice Hall, 2003

[3] *The Blocks Problem* - http://online-judge.uva.es/p/v1/101.html