UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br/

Professore: Fernando V. Paulovich (paulovic at icmc.usp.br)
Aluno PAE: Felipe S. L. G. Duarte (fgduarte at icmc.usp.br)

Trabalho 02: ArrayList

1 Prazos, Especificações e Observações importantes:

- O trabalho descrito a seguir é individual e não será tolerado qualquer tipo de plágio ou cópia em partes ou totalidade do código. Caso seja detectada alguma irregularidade, os envolvidos serão chamados para conversar com o professor responsável pela disciplina e os trabalhos serão zerados.

- A entrega deverá ser feita única e exclusivamente por meio do Sistema de Submissão de Programas (SSP) no endereço eletrônico `http://ssp.icmc.usp.br` até o dia **15 de Setembro de 2013 as 23 horas e 59 minutos**. Sejam responsáveis com o prazo final para entrega, o SSP está programado para não aceitar submissões após este horário e não será aceito entrega fora do sistema.

- A interpretação desta descrição faz parte do trabalho. Leia a descrição do trabalho com atenção e várias vezes, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para não ficar dúvidas e você consiga entregar o trabalho a tempo.

- O trabalho deverá ser submetido em formato zip/Makefile contendo o arquivo principal, o Makefile e possíveis TAD's implementadas (pares `.h .c`). Atente para o fato que todos os arquivos enviados deveram conter um cabeçalho contendo nome e número USP.

- A implementação é livre, crie quantas TAD's julgarem necessárias. Importante notar que a modularização do código bem como a forma como as TAD's foram criadas serão levados em consideração na atribuição final da nota.

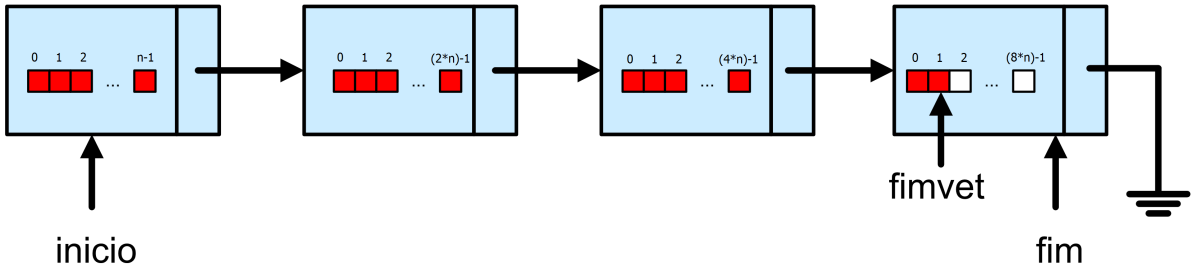
- Compile o programa em um sistema operacional linux antes de submeter ao SSP. **Trabalho em que o comando `make` e `make run` não funcione, não será corrigido e consequentemente receberá nota zero**. Caso não tenha o sistema operacional linux instalado em sua maquina, aconselha-se a utilização de uma máquina virtual. Como instalar o sistema operacional linux em uma maquina virtual está descrito no seguinte tutorial: `http://maistutoriais.com/2012/01/instalar-maquina-virtual-usando-o-virtual-box/`.

- Referencie, com um comentário no próprio código, qualquer algoritmo ou trecho de código retirado da internet. Código copiado sem a devida referencia é considerado plágio que por sua vez é crime.

- Nenhuma saída do trabalho será em arquivo, ou seja, o resultado dos algoritmos deverá ser exibidos utilizando o stdout `printf`.

2 Descrição do Problema:

ArrayList é uma estrutura de dados muito utilizada na linguagem de programação JAVA. Em uma implementação similar utiliza-se uma abordagem híbrida que mistura o paradigma de lista encadeada e lista estática. Nessa abordagem, cada nó, ao invés de armazenar apenas um único item, armazena um vetor de itens (uma lista estática). No momento de se adicionar uma novo item a lista, caso a lista estática do nó esteja cheia, um novo nó (da lista ligada) será criado contendo um vetor de tamanho 2 vezes maior que o vetor do nó anterior. A figura abaixo ilustra bem essa abordagem.



Seu trabalho é programar a TAD ArrayList. **O primeiro nó da lista encadeada deverá conter um vetor de tamanho 10, cada novo nó seguinte criado na lista encadeada deverá conter um vetor com tamanho duas vezes o tamanho do vetor contido no nó anterior.** Por exemplo, o primeiro nó conterá um vetor de tamanho 10, o segundo nó um vetor de tamanho 20, o terceiro um vetor de tamanho 40 e assim por diante.

Um nó só deverá estar na memória caso exista um ITEM em sua lista estática, caso contrario a memória alocada por ele deverá ser liberada. Assim, caso seja removido o ultimo elemento da lista estática de qualquer um dos nós, sua memória deve ser imediatamente liberada. Do mesmo modo, não se deve alocar memória para um nó caso não irá inserir um elemento em sua lista interna.

Toda e qualquer inserção no ArrayList deve ser ordenada. Esta ordenação é necessária para garantir uma busca binária nos elementos da lista estática dentro de cada nó do ArrayList. Assim, outra particularidade do sistema, está na eficiência em buscar elementos. **Toda e qualquer busca por elementos dentro da sua ArrayList deve ser feita sequencial na lista encadeada e binária nos elementos da lista estática.** Assim, ordem de busca por um elemento dentro do ArrayList é $O(n_1) + O(\log n_2)$ em que n_1 é o número de nós dentro do ArrayList e n_2 é o número de elementos da lista estática que será pesquisada.

Serão armazenados no ArrayList ITEM. **Utilize a TAD `item.h e item.c`, já implementada pelo professor em sala de aula, para resolução do trabalho aqui descrito. Além disso, implemente a TAD `arrayList.c e arrayList.h` que deverá conter todas as funções específicas da ArrayList. Caso sinta necessidade, implemente uma TAD `utils.c e utils.h` que deverá conter todas as outras funções que julgar necessárias.**

Para facilitar, segue em anexo no documento uma possível estrutura da TAD `arrayList.c e arrayList.h` bem como a struct necessária para implementação da mesma. Caso deseje, crie novas funções, remova ou insira novos campos na struct, a estrutura em anexo é somente para guiar o desenvolvimento.

Para correção no SSP e correto funcionamento das entradas e saídas do sistema, iremos utilizar de comandos textuais para guiar na execução e chamada de funções dentro do

código. Os comandos do sistema são:

3 Comandos do Sistema:

- `'sair'` - Este comando deverá sair do sistema liberando toda memória possivelmente alocada;
- `'tamanho'` - Este comando deverá imprimir o tamanho total do ArrayList utilizando a seguinte função:

```
printf("%d\n", tamanho_arraylist);
```

- `'add <chave> <valor>'` - Este comando deverá inserir um novo ITEM no ArrayList. Atente para o fato que **a inserção deve ser obrigatoriamente ordenada pelas chaves dos elementos**. Assim após o comando `add` serão fornecidos os dados correspondentes a chave e valor do novo ITEM a ser inserido. Mais uma vez, a ordenação deverá ser feita pelas chaves e **não** pelos valores.
- `'sub <pos1> <pos2>'` - Este comando deverá criar um sub ArrayList contendo alguns elementos do ArrayList original. Os elementos que devem ser inseridos no novo ArrayList são os elementos do ArrayList original no intervalo `[pos1, pos2[`, ou seja, o elemento determinado pela posição `pos1` deverá estar presente no sub ArrayList e o elemento na `pos2` não deve estar presente no sub ArrayList. Qualquer outro valor neste intervalo, obviamente, deverá estar presente no novo ArrayList. Após a criação deste novo ArrayList, Imprima-o utilizando a mesma função de impressão do comando `'print'`.
- `'set <pos> <chave> <valor>'` - Este comando deverá alterar os valores de `chave` e `valor` na posição `pos` do ArrayList. Atente para o fato que após esta modificação a lista poderá ficar desordenada. Assim, após a mudança, reordene o ArrayList para garantir que ele sempre estará ordenado;
- `'print'` - Este comando deverá imprimir na tela todo o conteúdo do ArrayList, para isso construa uma função de acordo com o pseudocódigo a seguir:

```
Para cada Registro da lista encadeada faça:
|
|   Para cada ITEM no vetor faça:
|   |
|   |   printf("%d: ", Indice Global do ArrayList);
|   |   printf( "%d/%d\n", chave do ITEM, valor do ITEM);
|   |
|   fim de para
|
|   printf("\n\n");
|
fim de para
```

- ‘contem <chave>’ - Este comando deverá imprimir ‘1’ quando o ArrayList contiver algum ITEM que a chave é idêntica a fornecida no comando ou ‘0’ caso contrario. Para isso utilize o seguinte comando:

```
printf("%d\n", contem);
```

- ‘indice <chave>’ - Este comando deverá efetuar uma busca no ArrayList e imprimir o índice do ArrayList que contem um ITEM com a mesma chave fornecida no comando. Caso haja 2 ou mais ITEM's com a mesma chave, imprima o índice da primeira ocorrência; ara isso utilize o seguinte comando:

```
printf("%d\n", indice);
```

- ‘vazia’ - Este comando deverá imprimir ‘1’ quando o ArrayList estiver vazio e ‘0’ caso contrario. Para isso utilize o seguinte comando:

```
printf("%d\n", vazia);
```

- ‘remover <pos>’ - Este comando deverá remover o elemento que ocupar a posição <pos> do ArrayList. Ao contrário do trabalho 1 que poderia apenas invalidar o registro, a remoção aqui deve realmente ser feita liberando a memoria alocada para o ITEM removido;

- ‘get <pos>’ - Este comando deverá imprimir o elemento que ocupar a posição <pos> do ArrayList. Caso não exista nenhum elemento na posição requerida, nada deverá ser impresso. Utilize o seguinte comando para impressão do ITEM:

```
printf( "%d/%d\n", item->chave, item->valor);
```

4 ArrayList.h

```
#ifndef ARRAY_LIST_H
#define ARRAY_LIST_H

#include "item.h"

typedef struct ArrayList ArrayList;

//Cria a ArrayList e aloca toda memória necessaria
ArrayList *new_arrayList();

//adiciona elemento ao arraylist
int add_arrayList(ArrayList *arrayList, ITEM *element);

//verifica no arraylist se existe um elemento com a chave informada
int contains_arrayList(ArrayList *arrayList, int chave);

//recupera um ITEM na posicao informada
ITEM *get_arrayList(ArrayList *arrayList, int pos);

//retorna qual a posicao do primeiro elemento com a chave informada
int indexOf_arrayList(ArrayList *arrayList, int chave);

//verifica se o arraylist esta vazio
int isEmpty_arrayList(ArrayList *arrayList);

//remove um elemento do arraylist
int remove_arrayList(ArrayList *arrayList, int pos);

//modifica um elemento do arraylist
int set_arrayList(ArrayList *arrayList, int pos, ITEM *element);

//retorna o tamanho total do arraylist
int size_arrayList(ArrayList *arrayList);

//recupera um novo subarray no intervalo [beginIndex, endIndex[
ArrayList *subArray_arrayList(ArrayList *arrayList, int beginIndex,
                              int endIndex);

//desaloca memoria alocada pelo arraylist
int destruct_arrayList(ArrayList **arrayList);

//imprime toda a lista
void print_arrayList(ArrayList *arrayList);

#endif
```

5 ArrayList.c

```
#include "ArrayList.h"

typedef struct NO {
  ITEM **list;//lista interna de void
  int fimvet;//tamanho ocupado na lista
  int tamanho;//tamanho da lista interna
  struct NO *proximo;
} NO;

struct ArrayList{
    NO *inicio;
};

ArrayList *new_arrayList(){
    printf("To be implemented...");
    return NULL;
}

int add_arrayList(ArrayList *arrayList, ITEM *element){
    printf("To be implemented...");
    return 1;
}

int clear_arrayList(ArrayList *arrayList){
    printf("To be implemented...");
    return 1;
}

ArrayList *clone_arrayList(ArrayList *arrayList){
    printf("To be implemented...");
    return NULL;
}


int contains_arrayList(ArrayList *arrayList, int chave){
    printf("To be implemented...");
    return 1;
}


ITEM *get_arrayList(ArrayList *arrayList, int pos){
    printf("To be implemented...");
    return NULL;
}

int indexOf_arrayList(ArrayList *arrayList, int chave){
    printf("To be implemented...");
    return 1;
```

```
}


int isEmpty_arrayList(ArrayList *arrayList){
    printf("To be implemented...");
    return 1;
}


int remove_arrayList(ArrayList *arrayList, int pos){
    printf("To be implemented...");
    return 0;
}


int set_arrayList(ArrayList *arrayList, int pos, ITEM *element){
    printf("To be implemented...");
    return 0;
}


int size_arrayList(ArrayList *arrayList){
    printf("To be implemented...");
    return 1;
}

ArrayList *subArray_arrayList(ArrayList *arrayList, int beginIndex,
                              int endIndex){
    printf("To be implemented...");
    return 1;
}

int destruct_arrayList(ArrayList **arrayList){
    printf("To be implemented...");
    return 1;
}

void print_arrayList(ArrayList *arrayList){
    ArrayList *paux;
    int i = 0;
    for(paux = arrayList; paux != NULL; paux = paux->next){
        for(i=0; i<paux->length; ++i){
            printf("%d: ", paux->globalBeginIndex + i);
            print_item(paux->list[i]);
        }
        printf("\n\n");
    }
}
```