MC202E - ESTRUTURAS DE DADOS

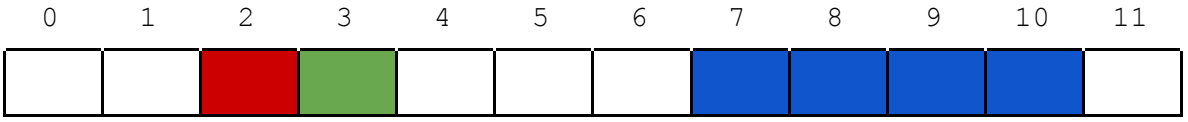
Lab05: Gerenciamento de Memória

O sistema operacional, ao carregar um programa para execução, disponibiliza um espaço de endereçamento, i.e., um espaço de memória que estará disponível para o programa do usuário. Um espaço de endereçamento de memória em particular, conhecido como área de alocação dinâmica, ou simplesmente *heap*, é o espaço reservado para blocos de memória alocados dinamicamente durante a execução do programa. Na linguagem C, para alocar memória no *heap*, utilizamos, por exemplo, as funções `malloc()` e `realloc()`, e para desalocar blocos, usamos a função `free()`.

Tarefa

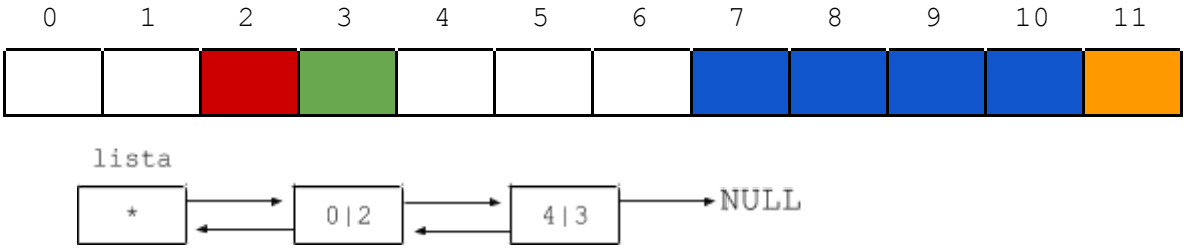
Nesta tarefa iremos implementar um modelo simplificado que simula o gerenciamento de memória do *heap*, i.e., gerencia segmentos do *heap* a serem disponibilizados de acordo com as operações realizadas ao longo da execução de um programa. Para tal, a estratégia que deverá ser adotada é a *best-fit*, que utiliza o segmento que deixar a menor sobra.

Suponha que o *heap* é uma sequência contígua de blocos com endereços sequenciais começando a partir do índice 0. Para ilustrar, considere o exemplo a seguir. Nessa ilustração, o *heap* possui 12 blocos, onde os segmentos [2, 3] e [7, 10] são regiões de memória em uso pelo programa durante sua execução, que correspondem a três alocações. A distinção entre diferentes segmentos coloridos é apenas para facilitar o entendimento. Em nosso modelo não haverá uma diferenciação explícita entre os blocos alocados em diferentes operações.

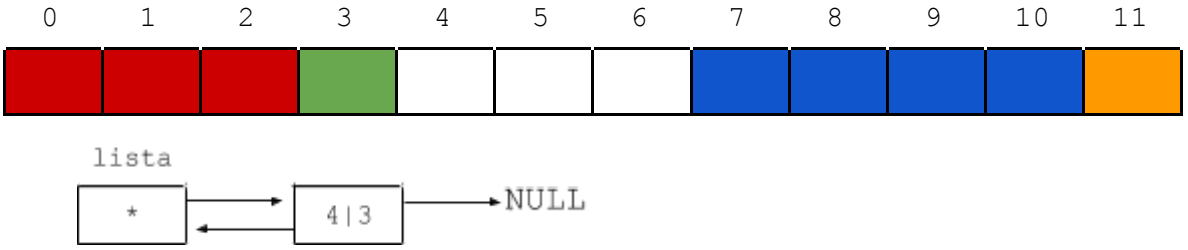


Para gerenciar a memória do *heap*, uma abordagem típica é utilizar alguma estrutura de dados para representar os segmentos livres e inspecioná-lo para selecionar onde ocorrerão novas alocações. A estrutura de dados que iremos utilizar é uma **lista duplamente ligada**, em que cada nó contém o endereço inicial e tamanho (número de blocos) de cada segmento livre. Esta lista deve estar ordenada por endereço e cada segmento livre contíguo deve ser representado por um único nó. Por exemplo, para a ilustração anterior, a lista pode ser visualizada conforme apresentado a seguir.

Seguindo o exemplo anterior, se for requisitado a alocação de um segmento de 1 bloco, então a alocação deve ser feita no **primeiro segmento livre que deixar a menor sobra**, i.e., o segmento iniciado no endereço 11, que corresponde, neste caso, ao último nó da lista.

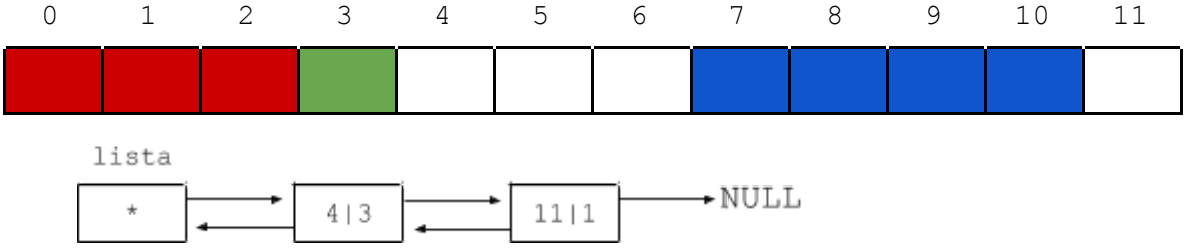


Uma operação de realocação consiste em alterar o tamanho do segmento previamente alocado. **Sempre que possível, a realocação é feita sem alterar o endereço inicial do segmento** – e.g., uma realocação para um tamanho menor. No caso de uma realocação para um tamanho maior e que **não é possível manter o endereço inicial, este deverá ser interpretado como uma desalocação seguida de uma alocação**. Por exemplo, se for requisitado a realocação do segmento iniciado no endereço 2 para o tamanho de 3 blocos, a lista ficará da seguinte maneira.



Embora existam casos em que uma operação pode falhar (e.g., memória insuficiente para uma alocação), você pode assumir que todas as operações serão bem sucedidas, ou seja, seu código não precisa tratar os casos de erro.

Para completar as ilustrações, se for requisitado a desalocação do segmento iniciado no endereço 11, a lista ficará da seguinte maneira.



Entrada

A primeira linha da entrada contém um inteiro positivo ***m*** indicando quantas operações serão realizadas e um inteiro positivo ***n*** indicando o tamanho total de espaço disponível no *heap*. Cada uma das demais ***m*** linhas contém uma operação, onde o primeiro campo

indica qual a operação com um caractere, sendo `'A'` para alocação, `'D'` para desalocação, `'R'` para realocação e `'P'` para impressão — e os demais campos dependem da operação em questão, sendo eles:

`'A'` – um inteiro indicando o tamanho do segmento a ser alocado;

`'D'` – dois inteiros indicando respectivamente o endereço inicial e o tamanho do segmento;

`'R'` – três inteiros indicando respectivamente o endereço inicial, o tamanho atual do segmento e seu o novo tamanho após a realocação; e

`'P'` – não possui nenhum parâmetro adicional.

Saída

Para cada chamada do operação de impressão, o seu programa deverá imprimir uma linha contendo a frase `"heap:"`, seguida de uma linha para cada nó da lista de segmentos livres naquele ponto da execução. O formato de cada linha é `"`***e t***`"`, onde ***e*** é o endereço do segmento e ***t*** o seu tamanho. É possível que haja apenas a primeira linha caso o heap esteja completamente ocupado.

Exemplo

A grafia da saída abaixo deve ser seguida rigorosamente por seu programa, inclusive a impressão de uma linha em branco no final da saída.

#1 Entrada & Saída

```
23 16
A 2
A 3
A 4
A 5
P
```

```
heap:
14, 2
```

```
D 0 2
D 5 4
P
```

```
heap:
0, 2
5, 4
14, 2
```

```
A 3
P
```

```
heap:
0 2
8 1
14 2
```

```
D 9 5
P
```

```
heap:
0 2
8 8
```

```
R 2 3 2
P
```

```
heap:
0 2
4 1
8 8
```

```
R 2 2 5
P
```

```
heap:
8 8
```

```
A 4
A 4
P
```

```
heap:
```

```
D 0 5
D 8 4
A 3
P
```

```
heap:
0 5
11 1
```

Critérios específicos

Os seguintes critérios específicos sobre o envio, implementação, compilação e execução devem ser satisfeitos.

i. Submeter no *SuSy* os arquivos:
Obrigatórios
⇒ **`lab05.c`**: Deverá conter a função principal para a solução do problema.

⇒ `lista.*`: Arquivos de cabeçalho e fonte contendo respectivamente a interface e implementação da estrutura de lista duplamente ligada.

Opcionais

⇒ `*.*`: Enviar até 2 arquivos cabeçalho e 2 arquivos fonte, desde de que contribuam para a modularização da solução.

ii. **É obrigatório** implementar uma solução que utiliza **lista duplamente ligada**. A lista **não** poderá armazenar explicitamente os segmentos alocados, operando apenas com base nos segmentos livres e nos parâmetros de cada operação.

iii. Flags de compilação:

```
-std=c99 -Wall -Werror -g -lm
```

iv. Tempo máximo de execução: **1 segundo**.

Observações gerais

No decorrer do semestre haverá 3 tipos de tarefas no *SuSy* (descritas logo abaixo). As tarefas possuirão os mesmos casos de testes abertos e fechados, no entanto o número de submissões permitidas e prazos são diferentes. As seguintes tarefas estão disponíveis no *SuSy*:

- **Lab05-AmbienteDeTeste**: Esta tarefa serve para testar seu programa no *SuSy* antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém os arquivos submetidos aqui **não serão corrigidos**.
- **Lab05-Entrega**: Esta tarefa tem limite de uma **única** submissão e serve para entregar a **versão final** dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa e submeta aqui apenas quando não for mais fazer alterações no seu programa.
- **Lab05-ForaDoPrazo**: Esta tarefa tem limite de uma **única** submissão e serve para entregar a versão final fora prazo estabelecido para o laboratório. Esta tarefa irá substituir a nota obtida na tarefa **Lab05-Entrega** apenas se o aluno tiver realizado as correções sugeridas no *feedback* ou caso não tenha enviado anteriormente na tarefa **Lab05-Entrega**.

Avaliação

Este laboratório será avaliado conforme o número de **casos de teste fechados** em que o seu programa apresentou saída correta, menos possíveis descontos referentes aos critérios de correção e de qualidade de código, os quais estão disponíveis na **planilha de notas**. Entretanto, outros critérios podem ser incorporados na avaliação desta tarefa se for julgado pertinente; e **a nota pode ser zerada caso não atender os critérios específicos.**