MC202 — ESTRUTURAS DE DADOS

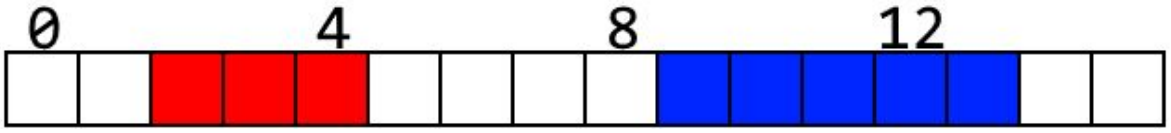
Lab 03 — Lista Ligada

Problema

Por trás de funções relacionadas à alocação dinâmica de memória na heap — como `malloc` e `free` — existe uma biblioteca que interage com o sistema operacional para obter e reservar segmentos da memória para o usuário. Neste trabalho iremos considerar um modelo simplificado de como uma dessas bibliotecas seleciona quais segmentos da heap serão disponibilizados para um usuário de acordo com as operações realizadas ao longo da execução de um programa.

Suponha que a heap já exista como uma sequência de *blocos* (unidade atômica de memória) com endereços sequenciais começando do 0. Um *bloco* é dito *livre* caso não esteja reservado para o usuário. Uma sequência de blocos é chamada de *segmento*. Para gerenciar o fornecimento de memória para o usuário, uma abordagem típica é utilizar alguma estrutura de dados para representar os segmentos livres da heap e inspecioná-la para selecionar onde ocorrerão novas alocações.

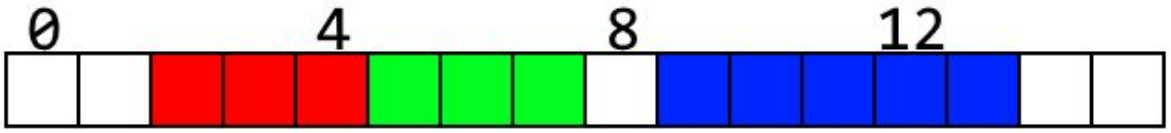
Neste laboratório iremos simular o componente que gerencia o espaço livre da heap utilizando uma lista ligada com o endereço inicial e tamanho (número de blocos) de cada segmento livre. Esta lista deve estar **ordenada** por endereço e cada segmento livre contíguo deve ser representado por um **único nó**. Por exemplo, suponha que, em algum ponto da execução do programa, o estado da heap seja:



onde os blocos brancos estão livres e os coloridos estão reservados para o usuário, cada cor sendo resultado de uma *alocação* distinta[1]. Representando um nó da lista de segmentos livres com o par ordenado (endereço, tamanho), a lista neste ponto pode ser visualizada como $(0,\ 2) \rightarrow (5,\ 4) \rightarrow (14,\ 2)$.

Seguindo o mesmo exemplo, um usuário então requisita a *alocação* de um segmento de 3 blocos da memória. A *alocação* deve ser feita no **primeiro** segmento livre que comporte o tamanho requisitado. Essa política é conhecida como *first-fit*. Você deve atualizar a lista para refletir o novo estado da heap:

[1] A distinção entre diferentes segmentos coloridos é apenas para facilitar o exemplo, em nosso modelo não haverá uma diferenciação explícita entre os blocos alocados em diferentes operações.



Com a ocupação parcial do segundo segmento livre pelo novo segmento verde, a lista deve conter $(0,\ 2) \rightarrow (8,\ 1) \rightarrow (14,\ 2)$. O usuário segue então para realizar uma *desalocação* do segmento azul:

Como um segmento livre contíguo deve sempre ser representado por um **único nó**, faz-se necessária a junção dos dois segmentos adjacentes ao espaço liberado de forma que o novo estado da lista passa a ser $(0,\ 2) \rightarrow (8,\ 8)$. O usuário decide então fazer uma *realocação* do segmento vermelho para que este tenha tamanho 2:

A lista passa a conter $(0,\ 2) \rightarrow (4,\ 1) \rightarrow (8, 8)$. Sempre que possível, a *realocação* é feita sem alterar o endereço inicial do espaço previamente alocado. No caso de uma realocação para um tamanho menor, este é sempre o caso. Por último, o usuário realiza outra *realocação* do segmento vermelho, desta vez para tamanho 5, obtendo:

Como não havia espaço suficiente para o novo tamanho no endereço anterior do segmento vermelho, este foi movido para outra região que o comportasse. O novo estado da lista passa a ser $(0,\ 5) \rightarrow (13,\ 3)$. Note que a seleção de onde alocar o novo bloco ocorre com o estado da lista de segmentos livres **antes** do início da realocação, assim o segmento fica no endereço 8 ao invés de ficar na endereço 0.

Partindo de uma heap completamente livre com tamanho pré-determinado, o seu trabalho consiste em representá-la com a estrutura de dados e as três operações de alocação, desalocação e realocação da memória descritas acima. 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.

Tipicamente, o endereço e tamanho dos segmentos alocados para o usuário são armazenados no próprio espaço do usuário, sendo visíveis para o seu gerenciador de memória apenas quando ocorre uma operação. Desta forma, sua estrutura de dados **não**

deve armazenar explicitamente a organização dos blocos alocados, operando apenas com base nos segmentos livres e nos parâmetros de cada operação.

Entrada

A primeira linha da entrada contém um inteiro *N* indicando quantas operações serão realizadas e um inteiro *S* indicando o tamanho total de espaço disponível na Heap. Cada uma das demais *N* linhas contém uma operação, onde o primeiro campo indica qual a operação com um caracter — **`A`** para alocação, **`D`** para desalocação, **`R`** para realocação e **`P`** para impressão — e os demais campos dependem da operação em questão. Uma alocação é seguida por um inteiro positivo indicando o *tamanho* do segmento a ser alocado. Tanto a desalocação quanto a realocação são seguidas por dois inteiros positivos indicando respectivamente o *endereço* e *tamanho* do segmento a ser operado. No caso da realocação o *endereço* e *tamanho* são seguidos de um inteiro positivo adicional indicando o novo *tamanho* da realocação. A operação de impressão não possui nenhum parâmetro adicional.

Saída

Para cada chamada do comando **`P`**, o seu programa deve imprimir uma linha "`Segmentos livres da heap:`" e mais 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. Note que é possível que haja 0 linhas caso a memória esteja completamente ocupada.

Exemplo

No exemplo abaixo, os retângulos internos correspondem à saída que seu programa deve gerar:

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

```
Segmentos livres da heap:
(14, 2)
```

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

```
Segmentos livres da heap:
(0, 2)
(5, 4)
(14, 2)
```

```
A 3
P
```

```
Segmentos livres da heap:
(0, 2)
(8, 1)
(14, 2)
```

```
D 9 5
P
```

```
Segmentos livres da heap:
(0, 2)
(8, 8)
```

```
R 2 3 2
P
```

```
Segmentos livres da heap:
(0, 2)
(4, 1)
(8, 8)
```

```
R 2 2 5
P
```

```
Segmentos livres da heap:
(0, 5)
(13, 3)
```

Observação

- Para as turmas A, B e C, esse laboratório tem peso 2.
- Será necessário implementar os arquivos:
  - lab03.c - Função principal e leitura da entrada
  - lista.h - Estrutura e protótipo das funções da lista ligada
  - lista.c - Implementação das funções da lista ligada
  - memoria.h - Protótipo das funções de gerenciamento da memória
  - memoria.c - Implementação das funções de gerenciamento da memória
- Observações sobre SuSy:
  - Versão do GCC: 4.4.7 20120313 (Red Hat 4.4.7-17)
  - *Flags* de compilação: `-g -ansi -pedantic-errors -Wall -Werror -lm`
  - Utilize comentários do tipo `/* comentário */`;
    comentários do tipo `//` serão tratados como erros pelo SuSy
  - Tempo máximo de execução: 2 segundos
- Além das observações acima, esse laboratório será avaliado pelos critérios:
  - Indentação de código
  - Organização
  - Corretude

Sugestões

- Se atente aos casos particulares de cada operação em que ocorre a criação, junção ou apenas a alteração de segmentos livres.