Atividade II

Linguagem de Programação I Instituto Metrópole Digital

2020.1

Introdução

Nesta semana vimos como modularizar nossos programas de uma forma que sua manutenção se torne mais fácil. Também vimos como podemos alocar memória dinamicamente (`new` e o `delete`), e aprendemos um pouco como utilizar um `std::stringstream` pra manipularmos *strings* de forma fácil.

Nesta atividade utilizaremos todos estes conhecimentos para continuarmos melhorando nosso programa de diário.

Descrição da Atividade

A atividade consiste na implementação de 3 funcionalidades pendentes, cada uma com uma pontuação diferente, totalizando 5 pontos:

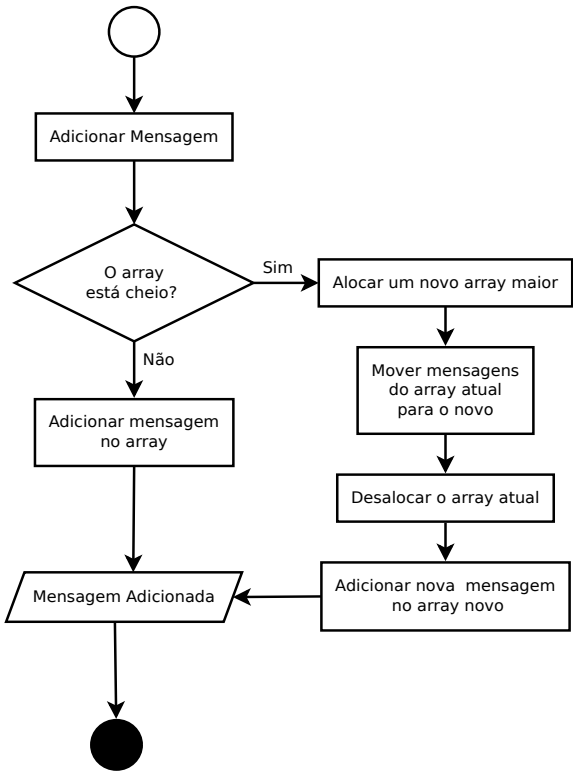
- **Carregar as mensagens previamente salvas ao instanciar um `Diary` (1,5 pontos):**
  Ao criar uma variável do tipo `Diary` e passar um nome de arquivo pra ele, o programa deve carregar as mensagens previamente salvas no arquivo, se existirem;

- **Salvar as mensagens no disco utilizando o formato Markdown (0,5 ponto):**
  Ao adicionar uma nova mensagem no diário, seu programa deve atualizar o arquivo para armazenar a mensagem nova;

- **Permitir o armazenamento de qualquer quantidade de mensagens no array de mensagens (3,0 pontos):**
  Se, ao tentar adicionar uma nova mensagem o *array* estiver cheio, vocês devem criar um novo *array* maior, mover todas as mensagens pra ele, adicionar a mensagem nova e deletar o *array* antigo. Tudo isso sem deixar vazar memória.

Aceitando Muuuuuitas Mensagens

Uma limitação atual do nosso programa é que ele só comporta até 10 mensagens. Com o uso de memória dinâmica, podemos remover esta restrição e fazer com que aceitemos qualquer quantidade de mensagens (o limite é a memória RAM disponível no nosso computador). Uma das estratégias para resolver isto é alocando um *array* maior, sempre que necessário, mover os dados atuais pra ele e depois desalocar o *array* antigo. O diagrama a seguir mostra como isto é feito:



Dessa forma, conseguimos fazer nosso programa aceitar quantas mensagens o nosso computador aguentar. É comum **dobrarmos o tamanho do *array* ao invés de aumentarmos de um em um**, pois o processo de cópia vai ficando cada vez mais lento com o aumento da quantidade de itens. Ou seja, se o nosso *array* estiver cheio e quisermos adicionar uma mensagem nova, seu novo tamanho, preferencialmente, deve ser 20 ao invés de 11 (e depois, 40, 80, 160, 320…).

Um erro bem comum ao fazermos a operação acima é tentarmos atribuir o ponteiro diretamente:

```
1  size_t capacity = 10;
2  Message* messages = new Message[capacity];
3
4  // vamos assumir que a partir daqui messages está lotado
5  // e precisamos criar um novo array...
6
7  capacity *= 2;
8  Message* new_array = new Message[capacity];
9  new_array = messages;
```

Na linha 9, podemos pensar `new_array` está recebendo o conteúdo que está presente em `messages`, mas não é isto que ocorre. Estamos atribuindo o endereço de memória de `messages` em `new_array` e assim perdemos o endereço de memória alocado na linha 8. A cópia deve ser feita percorrendo todos os elementos de `messages` e colocando em *new_array*:

```
1  size_t capacity = 10;
2  Message* messages = new Message[capacity];
3
4  // vamos assumir que a partir daqui messages está lotado
5  // e precisamos criar um novo array...
6
7  capacity *= 2;
8  Message* new_array = new Message[capacity];
9  // percorro todos os elementos de messages e coloco em new_array
10 // desaloco a memória apontada por messages
11 messages = new_array;
```

Depois de desalocar a memória antiga, podemos apontar nossa variável `messages` para a localização do novo array (Linha 11).

Carregando as Mensagens ao Instanciar um `Diary`

O formato do nosso arquivo possui características bem úteis que podem servir pra transformar o texto em `Message`s:

- Se a linha começar com uma cerquilha (#) sabemos que é uma data;
- Se começar com um traço (-) sabemos que é uma mensagem e que depois do traço tem a hora da mensagem e por último a mensagem em si.

Vocês podem utilizar estas informações e com a ajuda de `std::stringstream`s, transformar facilmente o texto do arquivo em `Message`s que deverão ser adicionados ao *array* no início do programa.

Relembrando o formato do arquivo

O formato que utilizaremos no texto será o **markdown** pela sua simplicidade e facilidade de exportação para outros formatos. O arquivo que armazena as mensagens deve possuir o seguinte formato:

- Para cada dia, deverá ser criado um **título**. Para criar um título em *markdown*, utilizamos uma cerquilha no início da linha (#). Exemplo:

```
# 17/06/2020
<mensagens do dia 17/06/2020>

# 18/06/2020
<mensagens do dia 18/06/2020>

# 19/06/2020
<mensagens do dia 19/06/2020>
...
```

- Cada mensagem deve ser precedida do horário em que foi cadastrada no formato **<horas>:<minutos>:<segundos>**, e devem ser colocadas em uma lista não ordenada. Para fazer isto com *markdown*, é necessário colocar um traço no início da linha (-). Exemplo:

```
# 17/06/2020

- 08:00:00 <mensagem 1>
- 15:33:25 <mensagem 2>
- 19:22:03 <mensagem 3>

# 18/06/2020

- 08:00:00 <mensagem 4>
- 15:33:25 <mensagem 5>
- 19:22:03 <mensagem 6>

# 19/06/2020

- 08:00:00 <mensagem 7>
- 15:33:25 <mensagem 8>
- 19:22:03 <mensagem 9>
```

Linhas em branco podem ser utilizadas para organizar o arquivo como desejarem. Só não são permitidas entre as mensagens de um dia, pois isto criaria várias listas ao invés de uma só.

Entrega

- O exercício deve ser entregue em um arquivo zip.
- Não será tolerado qualquer tipo de compartilhamento de trabalho entre os alunos. Tal ação resulta em anulação da pontuação para **TODOS** os envolvidos. Ou seja, colou ou deu cola? Zero na nota.
- **NÃO** serão aceitos envios por e-mail. O arquivo deve ser enviado **apenas** pelo *SIGAA* através da opção *Tarefas* até a data divulgada no sistema.