1. Resumo: Você deverá escrever em C++ um simulador de alocação dinâmica baseado em lista encadeada. O seu código-fonte deverá entregue por meio do endereço

http://dc.ufc.br/~pablo/2019-2/ed/trab2/

até 07/10/2019. A página acima, que já está no ar, fornece correção automática imediata e permite múltiplas submissões, desde que no prazo.

2. Introdução: Você deverá escrever um *simulador* de alocador dinâmico. Ele não manipulará a memória propriamente dita: ao invés disso, ele *simulará* a gerência de uma área de memória fictícia, recebendo solicitações fictícias de alocação, as quais serão atendidas, quando possível, com pedaços da área de memória sendo gerenciada.

Assim, por exemplo, se o alocador for inicializado com a "área de memória" que começa no "endereço" 10 e que tem tamanho 50, então os endereços de 10 a 59 da "memória" estarão inicialmente disponíveis. Se o usuário então solicitar uma alocação de tamanho 15, o alocador deverá (conforme regras a seguir) retornar o "endereço" 45, indicando ao usuário que os endereços de 45 a 59 foram alocados. Se, em seguida, o usuário solicitar uma alocação de tamanho 20, então o alocador deverá retornar o "endereço" 25, indicando que os endereços de 25 a 44 foram alocados. Se, em seguida, o usuário solicitar uma alocação de tamanho 30, então o alocador deverá informar falha de alocação, indicando que não há memória disponível. Se, em seguida, o usuário desalocar o trecho iniciado no "endereço" 45, então os endereços de 10 a 24 e de 45 a 59 estarão disponíveis, e os endereços de 25 a 44 estarão alocados. Finalmente, se o usuário solicitar uma alocação de tamanho 5, então o alocador deverá retornar o endereço 20, indicando que os endereços de 20 a 24 foram alocados.

3. Requisitos: Segue abaixo a especificação precisa do trabalho. É importante atender aos requisitos inteiramente, pois o código de correção automática interagirá com o código do trabalho para a realização dos testes e a atribuição da nota. Em caso de dúvida, por favor entre em contato sem demora.

(a) O código C++ a ser entregue consiste na definição de um struct `SimulAloc`, que implementará o simulador. Os "endereços de memória" disponíveis para "alocação" devem ser armazenados por meio de uma lista simplesmente encadeada. O tipo dos nós da lista será `struct Noh { int ini, tam; Noh *prox; };` (definido dentro de `SimulAloc`). A lista do final do exemplo da introdução acima, por exemplo, deve estar no estado

`[10;10;->], [45;15;nullptr]`,

isto é, o primeiro nó representa o intervalo de início 10 e tamanho 10, e o segundo nó representa o intervalo de início 45 e tamanho 15. O primeiro nó da lista será apontado pelo campo `prim` de `SimulAloc`; caso a lista esteja vazia (isto é, toda a memória gerenciada está alocada), então esse ponteiro deverá ser nulo.

(b) `bool inicializar (int ini, int tam)`

Esta função-membro de `SimulAloc` inicializará o simulador, recebendo o início `ini` e o tamanho `tam` da "área de memória" a ser gerenciada. A função deve retornar `true` se e somente se houver erro na alocação do `Noh` inicial da lista encadeada.

(c) `int alocar (int tam)`

Por meio desta função-membro, o usuário fornece o tamanho da "área de memória" que deseja "alocar", recebendo como retorno o "endereço" inicial do trecho que foi alocado (ou -1, se não houver "memória" o suficiente).

Para viabilizar a avaliação automática do simulador, o trecho a ser alocado **será, obrigatoriamente, o trecho final do primeiro nó da lista encadeada que possuir tamanho suficiente** (maior ou igual ao solicitado). Assim, por exemplo, quando a última alocação da introdução acima é solicitada, a lista encadeada está no estado

`[10;15;->], [45;15;nullptr]`,

e, embora um trecho de tamanho 5 em princípio pudesse ser obtido a partir de qualquer um dos dois nós da lista, a implementação solicitada necessariamente utilizará o trecho final do primeiro nó, isto é, os endereços de 20 a 24.

Observe que a estratégia acima pode ser implementada de forma a não exigir a alocação de um nó adicional para a lista encadeada, pois basta modificar (ou remover) o nó de onde será retirado o "trecho de memória alocado". Como nenhum `Noh` deve ser alocado, a função não possui previsão de indicação de erro real para o usuário. Em outras palavras, a única "falha" que pode ocorrer é não haver "memória" para a solicitação de alocação simulada (caso em que a função retorna -1).

Por fim, a lista encadeada não deve possuir nós representando "intervalos de memória" de tamanho zero: caso uma alocação utilize toda a memória representada por um nó, ele deverá ser retirado da lista e desalocado.

(d) `bool desalocar (int ini, int tam)`

Por meio desta função-membro, o usuário devolve um "trecho de memória" previamente "alocado", informando o início e o tamanho do trecho. (Embora `free` e `delete`, em C++, não exijam que o tamanho do trecho a ser desalocado seja informado, nós aqui o faremos para simplificar o trabalho). A função deverá retornar `true` se e somente se houver erro na alocação de um `Noh`.

Importante: essa função deverá unir "trechos de memória" contíguos, de forma que a lista encadeada da implementação nunca possua nós representando trechos consecutivos da "memória". Assim, por exemplo, se a lista encadeada estiver no estado

`[10;10;->], [45;15;nullptr]`

quando for desalocado o trecho de início 20 e tamanho 5, então a lista deve passar ao estado

`[10;15;->], [45;15;nullptr]`.

De forma semelhante, se, em seguida, for desalocado o trecho de início 25 e tamanho 20, então a lista deve passar ao estado

`[10;50;nullptr]`.

(e) `void terminar ()`

Esta função-membro deve apenas desalocar os nós da lista encadeada mantida pelo simulador.