1. Resumo: Você deverá escrever em C++ uma implementação de Dicionário baseada em Tabela de Dispersão com Encadeamento Externo. O seu código-fonte deverá entregue por meio do endereço

http://dc.ufc.br/~pablo/2019-2/ed/trab3/

até 01/11/2019. A página acima, quando estiver no ar, fornecerá correção automática imediata e permitirá múltiplas submissões, desde que no prazo.

2. Introdução: A sua implementação deverá possuir as funções típicas de dicionário: `inserir`, `procurar` e `remover`, assim como `inicializar` e `terminar`. As funções `inserir` e `remover` deverão redimensionar a tabela de dispersão adequadamente, com base no número de elementos armazenados pela estrutura.

Provavelmente a maior novidade técnica desta implementação é que ela deverá ser genérica quanto ao tipo de função de dispersão utilizada. Por essa razão, você não irá implementar, mas sim apenas receber e usar, uma função de dispersão. Mais precisamente, assim como o tipo `TC` da chave e o tipo `TV` dos valores, será **recebido (pronto, já implementado)** o tipo `TD` da dispersão. A maior utilidade do tipo `TD` é que ele possuirá uma função

unsigned int dispersao (TC c)

que receberá uma chave `c` e retornará uma posição da tabela de dispersão. Observe que a função acima não recebe o tamanho `m` da tabela, embora essa informação seja essencial; a razão é que, por eficiência, o valor de `m` não será informado toda vez que a função de dispersão for chamada, mas sim apenas uma vez (pelo menos enquanto o valor de `m` não mudar), por meio da função

void registrar_tam (**unsigned int** m)

de `TD`. Observe, então, que o `struct` que representará a sua tabela de dispersão receberá um argumento "template" `TD` que possuirá, ele mesmo, funções-membro que podem ser executadas!

Apesar de o tipo `TD` ser uma novidade técnica, o uso dele nesse trabalho é bastante simples, a começar pelo fato de que você apenas usará esse tipo. Antes, porém, de mostrar como ele pode ser usado, é útil mostrar um exemplo de como ele poderia ser implementado, para que você entenda concretamente do que estamos falando. Assim sendo, analise a implementação abaixo do método da divisão, supondo que o tipo `TC` das chaves é o tipo `unsigned int`:

```
struct MD // Implementa o Metodo da Divisao.
  {
   unsigned int m;

   void registrar_tam (unsigned int tam_tabela) { m = tam_tabela; }

   unsigned int dispersao (unsigned int c) { return c % m; }
  };
```

Claramente, a partir da definição acima, nós podemos escrever:

```
#include <iostream>

using std::cout;

template <typename TD>
void exemplo_de_uso_de_TD ()
  {
   TD h;  cout << "Registrando o tamanho da tabela como 5.\n\n";

   h.registrar_tam(5);

   cout << "Posicao da chave 21: " << h.dispersao(21) << '\n'
        << "Posicao da chave 48: " << h.dispersao(48) << "\n\n";

   cout << "Registrando o tamanho da tabela como 11.\n\n";

   h.registrar_tam(11);

   cout << "Posicao da chave 21: " << h.dispersao(21) << '\n'
        << "Posicao da chave 48: " << h.dispersao(48) << "\n\n";
  }

int main ()
  {
   exemplo_de_uso_de_TD <MD> ();
  }
```

Observe que o código da função `exemplo_de_uso_de_TD` acima não faz suposições sobre qual é a função de dispersão implementada pelo tipo `TD`. Naturalmente, nós sabemos que a função `main` está fornecendo `MD` como o tipo `TD` a ser utilizado por `exemplo_de_uso_de_TD`, e também sabemos que `MD` implementa o método da divisão; porém, é evidente que, na chamada a `exemplo_de_uso_de_TD`, nós poderíamos trocar `MD` por qualquer outro tipo adequadamente definido (por exemplo um que implementasse o método da multiplicação), e nada precisaria mudar no código da função `exemplo_de_uso_de_TD`. É exatamente esse o caso da implementação de dicionário deste trabalho: você deverá simplesmente usar as funções `registrar_tam` e `dispersao` do tipo `TD` recebido pelo seu dicionário, sem se preocupar com como esse tipo é definido; nos testes, diferentes funções de dispersão serão definidas e fornecidas para o seu dicionário.

DICA: Para testar a sua implementação "em casa", você pode usar o tipo `MD` acima ao declarar um dicionário. Nesse caso, é importante fornecer `unsigned int` como o tipo `TC` das chaves do dicionário:

```
DicioDisp <unsigned int, ...tipo dos valores..., MD> D;
```

3. Requisitos: Segue abaixo uma especificação precisa do restante 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

```
template <typename TC, typename TV, typename TD>
struct DicioDisp
```

que implemente um dicionário via tabela de dispersão com encadeamento externo. Os parâmetros `TC`, `TV` e `TD` estão explicados na introdução.

(b) **bool** inicializar ()

Esta função-membro de `DicioDisp` deve deixar o dicionário no estado vazio, usando uma tabela de dispersão de tamanho 1, retornando `true` se e somente se houver erro de alocação de memória.

(c) **bool** inserir (TC c, TV v)

Esta função-membro insere a chave `c` e o valor `v` no dicionário, já partindo da hipótese de que `c` é uma chave nova (não é necessário fazer esse teste), e retorna `true` se e somente se houver erro de alocação de memória. A função deve garantir que, ao fim de sua execução, valha $n \leq m$, sendo n o número de chaves armazenadas e m o tamanho da tabela; quando for necessário fazer realocação, deve ser alocado um novo vetor com o dobro do tamanho do atual.

(d) **bool** procurar (TC c, TV &var)

Esta função-membro deve procurar a chave `c` no dicionário, retornando `true` se e somente se ela estiver presente. Se a chave for encontrada, então o valor a ela correspondente deve ser atribuído a `var`; em caso contrário, `var` não deve ser utilizado.

(e) **bool** remover (TC c)

Esta função-membro deve remover do dicionário a chave `c` e seu valor correspondente; se a chave não estiver presente, a função deve retornar sem alterar o dicionário. A função também deve garantir que, ao fim de sua execução, se $n > 0$, então $m < 4n$, alocando um vetor com a metade do tamanho do atual quando necessário. A função deve retornar `true` se e somente se houver erro de alocação de memória (em particular, portanto, quando não houver redimensionamento, a função deve retornar `false`).

(f) **void** terminar ()

Esta função deve desalocar a memória utilizada pela tabela de dispersão (o vetor e os nós das listas encadeadas).