1151060 - Temas Selectos de Lenguajes de Programación
Tarea 3 - 2022-O.

El código disponible en https://racc.mx/uam/home/2022-o/tslp/tarea3.cpp implementa tres representaciones distintas de un tipo que modela un diccionario de parejas *(clave, valor)* donde la clave es un entero de 8 bits sin signo y el valor es de tipo `T` y se almacena con memoria dinámica usando `new`. El tipo `T` es un parámetro de plantilla. La semántica de las funciones miembro de tales representaciones es la misma que la descrita en la tarea 2. Las representaciones 1 y 2 también están parametrizadas por un entero no negativo `N` que indica que la representación podrá almacenar parejas de hasta `N` claves distintas, con un máximo de 256. Además, la funcion miembro `bool con_capacidad( ) const;` está disponible en todas las representaciones y regresa verdadero si y sólo la representación tiene capacidad suficiente para lograr que la siguiente llamada a `agrega` tenga éxito si se inserta una pareja con una clave ausente en la tabla.

La representación 1 mantiene un arreglo ordenado de claves y un arreglo de apuntadores a valores, de modo que el apuntador al valor de una pareja con clave `claves[i]` está en `valores[i]`. La representación 2 mantiene un arreglo de 256 bits y un arreglo de apuntadores a valores, de modo que si `valores[k]` es el apuntador al valor de la pareja con clave `i`, entonces hay `k` bits prendidos antes del bit `i` en el arreglo de bits. La representación 3 mantiene un arreglo de apuntadores a valores, de modo que `valores[i]` es distinto de nulo si y sólo si es el apuntador al valor de la pareja con clave `i`.

Implementa especializaciones `representacion1<void, N>`, `representacion2<void, N>` y `representacion3<void>` que sirvan para almacenar claves que no vienen acompañadas de valores. En las especializaciones, la función miembro `agrega` sólo debe tomar la clave y la función miembro `consulta` no debe existir. El parámetro de plantilla `N` de las representaciones 1 y 2 ahora debe indicar que representación podrá almacenar al menos `N` claves distintas. La implementación de la funcion miembro `bool con_capacidad( ) const;` debe responder verdadero si y sólo la representación tiene capacidad suficiente para lograr que la siguiente llamada a `agrega` tenga éxito si se inserta una clave ausente en la tabla. Además, debe cumplirse que `sizeof(representacion1<void, N>)` $\leq$ máx$(2 + \mathtt{N} + \mathtt{N} \,\%\, 2, 4)$ bytes, que `sizeof(representacion2<void, N>)` ≤ 32 bytes y que `sizeof(representacion3<void>)` ≤ 32 bytes.

Finalmente, define una plantilla de tipo `tabla<T, N>` donde `T` es un tipo y `N` es un entero no negativo con un valor máximo de 256, la cual provea un tipo con la misma semántica y el mismo `sizeof` que el tipo que tenga el menor `sizeof` de entre `representacion1<T, N>`, `representacion2<T, N>` y `representacion3<T>`. En caso de empate en el `sizeof` de varias representaciones, puedes elegir cualquiera de ellas.

Tu código no declarar `main` y no debe usar variables estáticas o globales. Se permite declarar tipos y funciones auxiliares, así como constantes globales en tiempo de compilación. Cada función puede declarar variables auxiliares que no superen los 500 kilobytes en total por función (es decir, las variables de una función pueden superar el límite de memoria del tipo `tabla`). Un programa que use cualquiera de los tipos solicitados para invocar cada función miembro mil veces debe terminar su ejecución en menos de un segundo. Sólo se pueden usar los archivos de biblioteca ya presentes en el código original.

Puedes consultar una página de prueba en https://racc.mx/uam/home/2022-o/tslp/tarea3.html. Deberás enviar el código fuente de tu programa desde tu cuenta institucional al formulario https://forms.gle/jWVv9LZrjzN39vcz5. Tu código será evaluado con varios casos de prueba y se espera que cumpla la semántica descrita.

Ejemplo de uso	Ejemplo de salida

```
int main( ) {
   representacion1<void, 1> r1;
   std::cout << r1.con_capacidad( ) << " "
             << r1.existe(8) << "\n";
   r1.agrega(8);
   std::cout << r1.con_capacidad( ) << " "
             << r1.existe(8) << "\n\n";

   tabla<double, 100> t;
   std::cout << sizeof(t) << "\n";
   std::cout << t.existe(8) << "\n";
   t.agrega(8, 3.14);
   std::cout << t.existe(8) << " "
             << *t.consulta(8) << "\n";
}
```

```
1 0
0 1

832
0
1 3.14
```