1151060 - Temas Selectos de Lenguajes de Programación
Tarea 4 - 2022-O.

El código disponible en https://racc.mx/uam/home/2022-o/tslp/tarea4.cpp es una implementación simplificada del tipo `tabla` solicitado en la tarea 3. Sea `T` un tipo, implementa una plantilla de interfaz llamada `tabla_interfaz<T>` que defina las siguientes funciones polimórficas en tiempo de ejecución:

- `virtual ~tabla_interfaz( ) = default;`
- `virtual bool con_capacidad( ) const = 0;`
- `virtual bool existe(uint8_t clave) const = 0;`
- `virtual const T* consulta(uint8_t clave) const = 0;`
- `virtual void agrega(uint8_t clave, const T& valor) = 0;`
- `virtual bool operator==(const tabla_interfaz<T>& t) const = 0;`
- `virtual std::unique_ptr<tabla_interfaz<T>> clona_mayor( ) const = 0;`

De manera similar a la tarea 3, se debe definir una especialización de la plantilla anterior para `T = void` que no contenga la función `consulta` y en donde la función `agrega` no tome el parámetro `const T&` del valor. El destructor virtual sirve para habilitar el polimorfismo en tiempo de ejecución de tipos derivados que implementen la interfaz. La semántica de casi todas las funciones es la misma que la descrita en la tarea 3 salvo por lo siguiente. El `operator==` debe determinar si las parejas (o las claves si `T = void`) de la tabla actual son iguales a las de `t`, a pesar de que no se conoce el tipo concreto de `t` pero se sabe que también implementa la interfaz. La función `clona_mayor` debe usar memoria dinámica para crear y regresar una tabla que contenga una copia de las parejas (o de las claves si `T = void`) de la tabla actual, donde además la tabla devuelta tenga mayor capacidad que la tabla actual (salvo si la tabla actual tiene capacidad de 256 parejas o claves, en cuyo caso la tabla devuelta debe tener la misma capacidad).

Además de definir la plantilla de la interfaz y su especialización, se debe implementar una plantilla de tipo llamada `tabla_polimorfica<T, N>` que derive e implemente `tabla_interfaz<T>`, siendo `T` un tipo y siendo `N` un entero no negativo menor o igual que 256 que denota la capacidad de la tabla. Como ocurre con `tabla<T, N>`, debe cumplirse que `tabla_polimorfica<T, N>` sólo use memoria dinámica para almacenar los valores de las parejas (esto si `T != void`). También debe cumplirse que `sizeof(tabla_polimorfica<T, N>)` $\leq 16 +$ `sizeof(tabla<T, N>)`. Al implementar la función `clona_mayor`, si la tabla actual es de tipo `tabla_polimorfica<T, N>` entonces la tabla devuelta debe ser de tipo `tabla_polimorfica<T, std::min(std::max(N + 1, 2 * N), size_t(256))>`.

Finalmente, se debe implementar la siguiente plantilla de función y su sobrecarga:

```
template<typename T>
void agrega_seguro(std::unique_ptr<tabla_interfaz<T>>& p, uint8_t c, const T& v);

void agrega_seguro(std::unique_ptr<tabla_interfaz<void>>& p, uint8_t c);
```

La semántica de esta plantilla de función debe ser la siguiente. Si la tabla apuntada por `p` puede llamar a `agrega` sin incurrir en problemas de capacidad, entonces debe hacerlo. En caso contrario, el apuntador `p` debe pasar a apuntar a la tabla devuelta por `clona_mayor` y posteriormente debe llamar a `agrega` sobre ella.

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 invoque mil veces cualquier función solicitada o 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/tarea4.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( ) {`
`  std::unique_ptr<tabla_interfaz<double>> p`
`  = std::make_unique<`
`  tabla_polimorfica<double, 0>`
`  >( );`
`  std::cout << p->existe(1) << "\n";`
`  agrega_seguro(p, 0, 3.14);`
`  agrega_seguro(p, 1, 2.71);`
`  agrega_seguro(p, 2, 8.65);`
`  std::cout << p->existe(1) << " "`
`  << *p->consulta(1) << "\n";`
`}`
	`0`
	`1 2.71`