PISTAS.

1. Para definir una plantilla de clase variádica, utilizamos el operador de puntos suspensivos ... seguido de un identificador de plantilla:

1. template<typename... Args>

2. class MyClass {

3. // ...

4. };

2. En el constructor de la plantilla de clase, podemos utilizar el operador de puntos suspensivos para recibir un número variable de argumentos de plantilla:

1. template<typename... Args>

2. class MyClass {

3. public:

4. MyClass(Args... args) {

5. // ...

6. }

7. };

8.

3. Podemos utilizar una tupla para almacenar los argumentos de plantilla recibidos en el constructor de la plantilla de clase:

1. #include <tuple>

2.

3. template<typename... Args>

4. class MyClass {

5. public:

6. MyClass(Args... args) {

7. std::tuple<Args...> t(args...);

8. // ...

9. }

10. };

11.

4. Podemos acceder a los elementos de la tupla utilizando el operador std::get y un índice:

1. #include <tuple>

2.

3. template<typename... Args>

4. class MyClass {

5. public:

6. MyClass(Args... args) {

7. std::tuple<Args...> t(args...);

8. const size\_t size = sizeof...(Args);

9. for (size\_t i = 0; i < size; ++i) {

10. std::cout << std::get<i>(t) << std::endl;

11. }

12. }

13. };

14.

5. Podemos utilizar un patrón de implementación para expandir los argumentos de plantilla en tiempo de compilación y acceder a ellos utilizando un índice. Por ejemplo, podemos definir una plantilla de clase IndexSequence que genere una secuencia de índices para un número dado de argumentos:

1. template<size\_t... Is>

2. struct IndexSequence {};

3.

4. template<size\_t N, size\_t... Is>

5. struct GenerateIndexSequence : GenerateIndexSequence<N - 1, N - 1, Is...> {};

6.

7. template<size\_t... Is>

8. struct GenerateIndexSequence<0, Is...> : IndexSequence<Is...> {};

9.

10. template<size\_t N>

11. using MakeIndexSequence = typename GenerateIndexSequence<N>::type;

12.

6. Luego podemos utilizar esta plantilla para generar una secuencia de índices y acceder a los elementos de la tupla:

1. #include <tuple>

2.

3. template<typename... Args>

4. class MyClass {

5. private:

6. template<size\_t... Is>

7. void printArgs(IndexSequence<Is...>) {

8. const std::tuple<Args...> t(args...);

9. const size\_t size = sizeof...(Args);

10. int dummy[] = { 0, ((void)std::cout << std::get<Is>(t) << std::endl, 0)... };

11. (void)dummy; // suppress unused variable warning

12. }

13.

14. public:

15. MyClass(Args... args) : args(args...) {

16. printArgs(MakeIndexSequence<sizeof...(Args)>());

17. }

18.

19. private:

20. std::tuple<Args...> args;

21. };

22.