No, you need form 28B, not 28C.

El ejercicio propuesto se centra en la creación y gestión de formularios burocráticos en C++98, utilizando conceptos avanzados como clases abstractas, herencia y polimorfismo. A continuación, se detalla en qué consiste el ejercicio y cómo abordarlo:

Descripción del Ejercicio:

El objetivo es ampliar una jerarquía de clases existente para manejar diferentes tipos de formularios burocráticos que realizan acciones específicas. La clase base, inicialmente denominada Form, debe ser renombrada a AForm y convertida en una clase abstracta. Esta clase contendrá atributos privados relacionados con los formularios, como el nombre, el estado de firma y los grados requeridos para firmar y ejecutar el formulario.

Se deben implementar las siguientes clases derivadas concretas, cada una representando un tipo específico de formulario:

1. **ShrubberyCreationForm**:
    a. **Grados requeridos**: Firma 145, Ejecución 137.
    b. **Acción**: Crea un archivo llamado <objetivo>_shrubbery en el directorio de trabajo y escribe en él una representación en ASCII de árboles.
2. **RobotomyRequestForm**:
    a. **Grados requeridos**: Firma 72, Ejecución 45.
    b. **Acción**: Emite sonidos de perforación y luego informa que el objetivo ha sido robotizado con éxito el 50% de las veces; en caso contrario, indica que la robotización ha fallado.
3. **PresidentialPardonForm**:
    a. **Grados requeridos**: Firma 25, Ejecución 5.
    b. **Acción**: Informa que el objetivo ha sido indultado por Zaphod Beeblebrox.

Cada una de estas clases debe tener un constructor que acepte un parámetro: el objetivo del formulario (por ejemplo, "hogar" si se desea plantar arbustos en casa).

Además, se debe añadir la función miembro execute(const Bureaucrat& executor) const a la clase base AForm y proporcionar una implementación en cada clase derivada que realice la acción específica del formulario. Es necesario verificar que el formulario esté firmado y que el grado del burócrata que intenta ejecutarlo sea suficientemente alto; de lo contrario, se debe lanzar una excepción apropiada.

Por último, se debe agregar la función miembro executeForm(const AForm& form) a la clase Bureaucrat. Esta función intentará ejecutar el formulario y, si tiene éxito, imprimirá un mensaje indicando que el burócrata ha ejecutado el formulario; si falla, imprimirá un mensaje de error explícito.

Cómo Abordar el Ejercicio en C++98:

1. **Renombrar y Modificar la Clase Base:**

   a. Renombra la clase Form a AForm y declárala como una clase abstracta. En C++98, una clase se convierte en abstracta declarando al menos una función miembro virtual pura. Por ejemplo, puedes declarar una función virtual pura llamada execute en AForm.
2. **Definir los Atributos Privados:**
   a. Mantén los atributos relacionados con el formulario (nombre, estado de firma, grados requeridos para firmar y ejecutar) como privados en la clase AForm. Proporciona funciones miembro para acceder y modificar estos atributos según sea necesario.
3. **Implementar las Clases Derivadas:**
   a. Crea las clases ShrubberyCreationForm, RobotomyRequestForm y PresidentialPardonForm, cada una derivada de AForm. Implementa el constructor de cada clase para aceptar el objetivo del formulario y establece los grados requeridos para firmar y ejecutar según lo especificado.
4. **Implementar la Función execute:**
   a. En cada clase derivada, implementa la función execute que realice la acción específica del formulario. Antes de ejecutar la acción, verifica si el formulario está firmado y si el burócrata tiene el grado necesario para ejecutarlo. Si alguna de estas condiciones no se cumple, lanza una excepción adecuada.
5. **Agregar la Función executeForm a la Clase Bureaucrat:**
   a. En la clase Bureaucrat, implementa la función executeForm que intenta ejecutar un formulario dado. Si la ejecución es exitosa, imprime un mensaje indicando que el burócrata ha ejecutado el formulario; de lo contrario, imprime un mensaje de error detallado.
6. **Crear un Makefile:**
   a. Especifica las reglas para compilar el programa, incluyendo los archivos fuente (main.cpp, Bureaucrat.cpp, AForm.cpp, etc.) y las dependencias necesarias.
7. **Escribir Pruebas:**
   a. Desarrolla pruebas en main.cpp para asegurarte de que todas las funcionalidades funcionan según lo esperado. Por ejemplo, crea instancias de cada tipo de formulario, asígnales objetivos, firma y ejecuta los formularios con burócratas de diferentes grados, y verifica que las acciones se realicen correctamente o que se lancen las excepciones adecuadas cuando corresponda.

Recursos Recomendados:

Para profundizar en los conceptos clave necesarios para abordar este ejercicio, se recomiendan los siguientes videos en español:

1. **"Programación Orientada a Objetos en C++: Clases y Objetos"**
   a. Este video ofrece una introducción a la programación orientada a objetos en C++, cubriendo conceptos fundamentales como clases y objetos.
   b. Enlace: https://www.youtube.com/watch?v=video1
2. **"Herencia en C++: Conceptos y Ejemplos"**
   a. Explora la herencia en C++, incluyendo la herencia simple y múltiple, con ejemplos prácticos para una mejor comprensión.
   b. Enlace: [https://www.youtube.com/watch?v=video2](https://
3. **"Curso C++. Clases abstractas."**
   a. Este video ofrece una explicación detallada sobre las clases abstractas en C++, incluyendo su definición y uso práctico.
   b. Enlace: https://www.youtube.com/watch?v=ZvnEdRz-Fc0

4. **"Herencia en C++ - Yolanda Martinez Treviño"**
    a. Este video explica el concepto de herencia en C++ mediante un ejemplo práctico, facilitando la comprensión de cómo las clases derivadas pueden heredar propiedades y métodos de las clases base.
    b. Enlace: https://www.youtube.com/watch?v=VxqwwlZQXgw
5. **"3d. Clases Abstractas en C++"**
    a. Este video profundiza en el concepto de clases abstractas en C++, proporcionando ejemplos claros y concisos para ilustrar su implementación y uso.
    b. Enlace: https://www.youtube.com/watch?v=dwWslFxGCgY
6. **"031.- Curso de C++. Polimorfismo y Funciones Virtuales."**
    a. Este video introduce el polimorfismo y las funciones virtuales en C++, conceptos fundamentales para la programación orientada a objetos y relevantes para la implementación de este ejercicio.
    b. Enlace: https://www.youtube.com/watch?v=HY-cTsxajTk
7. **"Ejemplo de clase abstracta y función virtual pura en C++"**
    a. Este video muestra un ejemplo práctico de cómo implementar una clase abstracta y una función virtual pura en C++, lo que es esencial para comprender la estructura requerida en este ejercicio.
    b. Enlace: https://www.youtube.com/watch?v=I_h4ytCc8g4

Estos recursos proporcionan explicaciones detalladas y ejemplos prácticos que te ayudarán a comprender y aplicar los conceptos necesarios para completar con éxito el ejercicio propuesto.

El código

Este código implementa un sistema de formularios y burócratas en C++ siguiendo el patrón de diseño de herencia y excepciones. Voy a explicarlo paso a paso.

📌 Resumen general

El código define una jerarquía de clases para formularios administrativos que pueden ser firmados y ejecutados por burócratas con diferentes rangos.

- **AForm** (clase base abstracta): Representa un formulario con restricciones de firma y ejecución según el rango del burócrata.
- **Bureaucrat**: Representa a un burócrata con un nombre y un rango, que puede firmar y ejecutar formularios.
- **Tres tipos de formularios específicos** (que heredan de AForm):
    - **PresidentialPardonForm** → Indulta a alguien.
    - **RobotomyRequestForm** → Realiza una "robotomización" (50% de éxito).
    - **ShrubberyCreationForm** → Crea un archivo con un dibujo de árboles.

📌 Explicación detallada por partes

1️⃣ Clase AForm (clase base abstracta)

Propósito:

Es la clase madre de todos los formularios y define las reglas para firmar y ejecutar formularios.

Atributos:

```
const std::string _name;    // Nombre del formulario
const std::string _target;  // Objetivo del formulario
const int _gradeSign;       // Rango necesario para firmarlo
const int _gradeExec;       // Rango necesario para ejecutarlo
bool _signed;               // Indica si el formulario está firmado
```

- gradeSign: Nivel de burócrata requerido para firmarlo.
- gradeExec: Nivel de burócrata requerido para ejecutarlo.
- _signed: Indica si el formulario ha sido firmado.

Métodos importantes:

```
void beSigned(Bureaucrat &bureaucrat);
void execute(Bureaucrat const &executor) const;
```

- beSigned(): Un burócrata intenta firmar el formulario.
- execute(): Ejecuta el formulario llamando a un método abstracto executeSuperClassForm(), que cada subclase debe implementar.

Excepciones:

```
class gradeTooHighException : public std::exception { ... };
class gradeTooLowException : public std::exception { ... };
```

Si el rango del burócrata es demasiado alto o bajo, se lanza una excepción.

2 Clase Bureaucrat

Propósito:

Representa a un burócrata con un nombre y un rango, que puede firmar y ejecutar formularios.

Atributos:

```
const std::string _name;  // Nombre del burócrata
int _grade;           // Rango del burócrata (1 = mejor, 150 = peor)
```

Métodos importantes:

```
void signForm(std::string formName, bool wasSigned);
void executeForm(AForm const &form);
```

- signForm(): Intenta firmar un formulario.
- executeForm(): Ejecuta un formulario (si está firmado y el burócrata tiene el rango suficiente).

También permite **incrementar o disminuir el rango** del burócrata:

```
void incrementGrade();
void decrementGrade();
```

Si el rango supera los límites (1-150), lanza excepciones.

3 Subclases de AForm

Son formularios específicos con sus propias implementaciones de executeSuperClassForm().

PresidentialPardonForm

```
void executeSuperClassForm(Bureaucrat const &executor) const {
  if (executor.getGrade() > this->getGradeExec())
    throw Bureaucrat::GradeTooLowException();
  else if (!this->getSigned())
    cerr << executor.getName() << " cannot execute " << this->getName() << " because the form is not
signed" << endl;
```

```
  else
    cout << this->getTarget() << " has been pardoned by Zafod Beeblebrox" << endl;
}
```

- **Si el burócrata tiene el rango suficiente y el formulario está firmado**, se muestra un mensaje diciendo que el objetivo ha sido indultado.

☑ *RobotomyRequestForm*

```
void executeSuperClassForm(Bureaucrat const &executor) const {
  if (executor.getGrade() > this->getGradeExec())
    throw Bureaucrat::GradeTooLowException();
  else if (!this->getSigned())
    cerr << executor.getName() << " cannot execute " << this->getName() << " because the form is not
signed" << endl;
  else {
    std::srand(std::time(NULL));
    cout << GREEN << "\n* * * * * * * drilling noises * * * * * * *\n" << RESET << endl;
    if (std::rand() % 2 == 0)
      cout << this->getTarget() << " has been robotomized successfully" << endl;
    else
      cout << this->getTarget() << " has not been robotomized successfully" << endl;
  }
}
```

- **Simula una operación de robotomización con un 50% de éxito** usando std::rand().

☑ *ShrubberyCreationForm*

```
void executeSuperClassForm(Bureaucrat const &executor) const {
  if (executor.getGrade() > this->getGradeExec())
    throw Bureaucrat::GradeTooHighException();
  else if (!this->getSigned())
    cerr << executor.getName() << " cannot execute " << this->getName() << " because the form is not
signed" << endl;
  else {
    std::ofstream out;
    out.open((this->getTarget() + "_shrubbery").c_str(), std::ofstream::out | std::ofstream::trunc);
```

```
    out << "    #    " << "\n"
      << "   ###   " << "\n"
      << "  #####  " << "\n"
      << " ####### " << "\n"
      << "    |    " << endl;
    out.close();
  }
}
```

- **Crea un archivo con un dibujo de un árbol ASCII**.

📌 Ejemplo de Uso

```
int main() {
  try {
    Bureaucrat bob("Bob", 5);
    PresidentialPardonForm form("Alice");

    bob.signForm(form.getName(), false);  // Intenta firmar el formulario
    form.beSigned(bob);              // Bob firma el formulario
    bob.executeForm(form);           // Bob ejecuta el formulario
  }
  catch (std::exception &e) {
    std::cerr << e.what() << std::endl;
  }
}
```

🔹 **Bob (rango 5) firma y ejecuta un PresidentialPardonForm para "Alice".**

📌 Conclusión

1. **Los formularios (AForm) necesitan ser firmados antes de ser ejecutados.**
2. **Cada formulario tiene requerimientos de rango diferentes.**
3. **Los burócratas (Bureaucrat) tienen un rango y pueden firmar o ejecutar formularios si cumplen los requisitos.**

4. **Existen 3 formularios específicos con diferentes efectos (indulto, robotización, dibujo de árbol).**
5. **El código usa excepciones para manejar errores de rango.**

En resumen, el código simula un sistema burocrático donde los formularios deben ser firmados y ejecutados por burócratas con un nivel adecuado.