Conversion of scalar types

El ejercicio 00, titulado "Conversión de tipos escalares", tiene como objetivo desarrollar una clase en C++98 llamada ScalarConverter que contiene un único método estático convert. Este método recibe como parámetro una cadena de texto que representa un literal en C++ y muestra su valor convertido en los siguientes tipos escalares:

- char
- int
- float
- double

La clase ScalarConverter no debe ser instanciable, ya que no necesita almacenar ningún estado interno. Por lo tanto, todos sus métodos y atributos deben ser estáticos.

Descripción detallada del ejercicio:

1. **Detección del tipo de literal:**
    a. **char**: Literales de un solo carácter, como 'c' o 'a'. No se deben utilizar caracteres no imprimibles como entrada. Si la conversión a char no es imprimible, se debe mostrar un mensaje informativo.
    b. **int**: Números enteros en notación decimal, como 0, -42 o 42.
    c. **float**: Números de punto flotante con sufijo f, como 0.0f, -4.2f o 4.2f. También se deben manejar los pseudo-literales -inff, +inff y nanf.
    d. **double**: Números de punto flotante sin sufijo, como 0.0, -4.2 o 4.2. También se deben manejar los pseudo-literales -inf, +inf y nan.
2. **Conversión entre tipos:**
    a. Una vez detectado el tipo del literal, se debe convertir la cadena de texto a su tipo correspondiente (char, int, float o double).
    b. Posteriormente, se debe realizar una conversión explícita a los otros tres tipos de datos.
3. **Manejo de conversiones imposibles o desbordamientos:**
    a. Si una conversión no tiene sentido o produce un desbordamiento, se debe mostrar un mensaje informando al usuario que la conversión es imposible.
    b. Es necesario incluir los encabezados adecuados para manejar los límites numéricos y valores especiales.
4. **Formato de salida:**
    a. El programa debe mostrar los resultados de las conversiones en el siguiente formato:

```
./convert 0
char: Non displayable
int: 0
```

float: 0.0f
double: 0.0
./convert nan
char: impossible
int: impossible
float: nanf
double: nan
./convert 42.0f
char: '*'
int: 42
float: 42.0f
double: 42.0

Cómo abordar el ejercicio en C++98:

1. **Definición de la clase ScalarConverter:**
   a. Declarar la clase con un constructor privado o eliminarlo para evitar la instanciación.
   b. Definir el método estático convert que tomará una cadena de texto como parámetro.
2. **Implementación del método convert:**
   a. **Detección del tipo de literal:**
      Analizar la cadena de entrada para determinar su tipo. Por ejemplo, verificar si está entre comillas simples para identificar un char, si contiene solo dígitos para un int, si tiene un punto decimal y un sufijo 'f' para un float, etc.
   b. **Conversión al tipo detectado:**
      Utilizar funciones de conversión estándar de C++ como strtol para int, strtof para float y strtod para double.
   c. **Conversión a los otros tipos:**
      Realizar conversiones explícitas entre los tipos según sea necesario. Por ejemplo, de int a char, de float a double, etc.
   d. **Manejo de errores y casos especiales:**
      Verificar si las conversiones son válidas y si los valores resultantes son representables en el tipo de destino. Manejar casos como nan, inf y valores fuera de los rangos permitidos.
3. **Programa de prueba:**
   Escribir un programa que tome argumentos de línea de comandos y llame al método convert de ScalarConverter con cada argumento para demostrar su funcionalidad.

Para complementar la información proporcionada anteriormente sobre el ejercicio de conversión de tipos escalares en C++98, es fundamental profundizar en algunos aspectos clave que facilitarán su implementación exitosa.

1. Literales de Caracteres en C++:

En C++, un literal de carácter se representa encerrando un único carácter entre comillas simples, por ejemplo, 'a' o '1'. Estos literales son de tipo char y almacenan un valor numérico correspondiente al código ASCII del carácter. Es importante destacar que un literal de carácter contiene exactamente un carácter; si se incluyen más caracteres, el comportamiento es indefinido y depende de la implementación del compilador. cite turn0search0

2. Conversión de Cadenas a Números en C++98:

Para convertir una cadena de texto que representa un número a un tipo numérico en C++98, se pueden utilizar las funciones de la biblioteca estándar incluidas en <cstdlib>. Por ejemplo, std::strtol se utiliza para convertir una cadena a long int, y std::strtod para convertir a double. Estas funciones permiten especificar un puntero para indicar dónde termina la conversión en la cadena, lo cual es útil para validar la entrada. cite turn0search8

3. Manejo de Pseudo-Literales:

Los pseudo-literales como nan, -inf y +inf representan valores especiales en punto flotante. En C++, estos valores se pueden obtener utilizando constantes definidas en la biblioteca <cmath>, como NAN, INFINITY y -INFINITY. Al implementar la conversión, es esencial reconocer estas cadenas y asignarles los valores correspondientes. Además, al convertir estos valores a tipos enteros o char, se debe manejar adecuadamente la imposibilidad de representación, mostrando mensajes informa-tivos al usuario.

4. Conversión entre Tipos Numéricos:

La conversión entre tipos numéricos en C++ puede ser implícita o explícita. Las conversiones implícitas ocurren cuando se asigna un valor de un tipo a una variable de otro tipo compatible, y el compilador realiza la conversión automáticamente, aunque esto puede conllevar pérdida de precisión o desbordamientos. Por otro lado, las conversiones explícitas, o *casts*, se realizan utilizando una sintaxis específica, como static_cast. En C++98, la forma más común de realizar un *cast* es utilizando la sintaxis tradicional de C, por ejemplo, (int)valor_double. Sin embargo, es recomendable utilizar los *casts* específicos de C++ por motivos de claridad y seguridad. cite turn0search17

5. Manejo de Errores y Validación de Entrada:

Al implementar el método convert, es crucial validar la entrada para asegurarse de que la cadena representa un literal válido. Esto incluye verificar que las conversiones no resulten en desbor-damientos y que los valores sean representables en el tipo de destino. Por ejemplo, al convertir una cadena a int, se debe verificar que el valor esté dentro del rango permitido para los enteros en C++. Si una conversión no es posible o resulta en un valor fuera de rango, se debe informar al usuario mediante mensajes claros.

Recursos Adicionales:

Para profundizar en los temas mencionados, se recomiendan los siguientes recursos en español:

1. **Curso Maestro de C++: Conversión de tipos en C++ #9**
   a. *Descripción:* Este video ofrece una explicación detallada sobre cómo se manejan las conversiones de tipos en C++, incluyendo ejemplos prácticos y consejos para evitar errores comunes.
   b. *Enlace:* https://www.youtube.com/watch?v=eSXvkZETw58
2. **CADENAS de Caracteres en Programación - (C++) - E#26**
   a. *Descripción:* Este video aborda el manejo de cadenas de caracteres en C++, inclu-yendo la conversión de cadenas a otros tipos de datos y las funciones estándar disponibles para este propósito.
   b. *Enlace:* https://www.youtube.com/watch?v=q03fCuLh8t4
3. **Conversión de tipos en C++ | by Jesús Torres - Medium**
   a. *Descripción:* Artículo que explica las diferentes formas de conversión de tipos en C++, tanto implícitas como explícitas, y cómo utilizarlas de manera segura.
   b. *Enlace:* https://medium.com/jmtorres/conversi%C3%B3n-de-tipos-en-c-ce37d8ba7e46
4. **Conversión de tipos en C++ - Jesús Torres**
   a. *Descripción:* Este artículo profundiza en las conversiones de tipos en C++, incluyendo el uso de operadores de *cast* específicos del lenguaje y las mejores prácticas para su uso.
   b. *Enlace:* https://jesustorres.hashnode.dev/conversion-de-tipos-en-cpp
5. **Conversión de tipos en C++ - Luis Llamas**
   a. *Descripción:* Artículo que aborda las conversiones de tipos en C++, explicando las diferencias entre conversiones implícitas y explícitas, y cómo utilizarlas correctamente para evitar errores.
   b. *Enlace:* https://www.luisllamas.es/cpp-conversion-tipos-cast/

Estos recursos proporcionan información adicional y ejemplos prácticos que te ayudarán a comprender y abordar con éxito el ejercicio de conversión de tipos escalares en C++98.

El código

Explicación detallada del código

El código proporciona una implementación en C++ para convertir un valor ingresado como cadena de texto (std::string) a diferentes tipos de datos primitivos: char, int, float y double. Se trata de una clase estática ScalarConverter, que puede identificar el tipo del valor de entrada y realizar la conversión apropiada.

El programa toma un argumento en la línea de comandos y lo pasa a la función ScalarConverter::convert(), que maneja la conversión y muestra los resultados.

1. Archivos y su función

El código está compuesto por tres archivos:

- **ScalarConverter.hpp**: Archivo de encabezado con la definición de la clase ScalarConverter.
- **ScalarConverter.cpp**: Implementación de la clase ScalarConverter, con la lógica para la conversión.
- **main.cpp**: Punto de entrada del programa, maneja los argumentos de línea de comandos y llama a la conversión.

2. Análisis de ScalarConverter.hpp

Este archivo define la clase ScalarConverter, que solo tiene una función pública:

static void convert(const std::string &input);

📌 **Propiedades importantes:**

- Es **estática**, lo que significa que no necesita una instancia de ScalarConverter para ser utilizada.
- Recibe un std::string y lo convierte a los tipos char, int, float y double.

Además, la clase tiene un **constructor privado, un destructor y operadores de copia privados** para impedir la creación de instancias:

```
private:
    ScalarConverter();
    ~ScalarConverter();
    ScalarConverter(const ScalarConverter &);
    ScalarConverter &operator=(const ScalarConverter &);
```

🔹 **Propósito**: La clase no tiene atributos y todas sus funciones son estáticas, por lo que no necesita ser instanciada.

3. Análisis de ScalarConverter.cpp

3.1 Conversión de cadenas a números (ft_atoi, ft_atof, ft_atod)

El código define tres funciones auxiliares para convertir una std::string en int, float y double, verificando errores:

```
static int ft_atoi(const std::string &str) {
   char *end;
   errno = 0;
   long i = std::strtol(str.c_str(), &end, 10);
   if (*end != '\0' || errno == ERANGE || i < std::numeric_limits<int>::min()
      || i > std::numeric_limits<int>::max()) {
      cerr << "Error: out of range !!!" << endl;
      exit(1);
   }
   return static_cast<int>(i);
}
```

📌 **Explicación**:

- std::strtol() convierte la cadena a un long int, almacenando en end cualquier carácter no numérico.
- Si hay caracteres extraños (*end != '\0'), si el número es muy grande (ERANGE), o si se sale del rango de un int, se muestra un error y se termina la ejecución.

Las funciones ft_atof y ft_atod hacen lo mismo, pero para float y double usando std::strtof() y std::strtod().

3.2 Detección del tipo de entrada

Para determinar a qué tipo de dato pertenece la entrada, el código usa varias funciones isChar(), isInt(), isFloat(), isDouble() e isPseudo():

Detección de un char

```
static bool isChar(const std::string &input) {
   if (input.length() != 1) return false;
   if (input.at(0) < std::numeric_limits<char>::min()
      || input.at(0) > std::numeric_limits<char>::max()
```

```
    || isdigit(input.at(0)))
    return false;
  return true;
}
```

📌 **Reglas**:

- Debe ser un solo carácter.
- No puede ser un número (isdigit()).
- Debe estar en el rango de char.

Detección de un int

```
static bool isInt(const std::string &input) {
  for (size_t i = (input.at(0) == '-' ? 1 : 0); i < input.length(); i++) {
    if (!isdigit(input.at(i))) return false;
  }
  return true;
}
```

📌 **Reglas**:

- Puede empezar con -.
- Todos los caracteres deben ser dígitos.

Detección de un float

```
static bool isFloat(const std::string &input) {
  bool point = false;
  if (input == "-inff" || input == "+inff" || input == "nanf") return true;
  if (input.at(input.length() - 1) != 'f') return false;
  for (size_t i = 0; i < input.length() - 1; i++) {
    if (input.at(i) == '.' && point) return false;
    else if (input.at(i) == '.') {
      point = true;
      continue;
    }
    if (!isdigit(input.at(i)) && input.at(i) != '-' && input.at(i) != '+') return false;
  }
  return true;
```

```
}
```

📌 **Reglas**:

- Debe terminar en 'f'.
- Puede tener un solo '.' (punto decimal).
- Puede comenzar con - o +.
- No debe contener caracteres no numéricos.

Detección de un double

```
static bool isDouble(const std::string &input) {
  bool point = false;
  if (input == "-inf" || input == "+inf" || input == "nan") return true;
  for (size_t i = 0; i < input.length(); i++) {
    if (input.at(i) == '.' && point) return false;
    else if (input.at(i) == '.') {
      point = true;
      continue;
    }
    if (!isdigit(input.at(i)) && input.at(i) != '-' && input.at(i) != '+') return false;
  }
  return true;
}
```

📌 **Reglas**:

- Igual que float, pero no requiere 'f' al final.

3.3 Conversión y salida de resultados

La función convert() usa getType() para identificar el tipo y luego llama a la función correspondiente:

```
void ScalarConverter::convert(const std::string &input) {
  switch (getType(input)) {
    case CHAR:
      printChar(input.at(0));
      break;
    case INT:
      printInt(ft_atoi(input));
```

```
        break;
      case FLOAT:
        if (isPseudo(input))
          pseudo(FLOAT, input);
        else
          printFloat(ft_atof(input));
        break;
      case DOUBLE:
        if (isPseudo(input))
          pseudo(DOUBLE, input);
        else
          printDouble(ft_atod(input));
        break;
      default:
        cout << "Error: invalid input" << endl;
        break;
    }
}
```

📌 **Flujo de conversión**:

1. Se determina el tipo con getType().
2. Se convierte el valor y se imprime usando printChar(), printInt(), printFloat() o printDouble().
3. Si es un "pseudo-valor" como nan, se maneja con pseudo().

4. main.cpp

Este archivo recibe el valor desde la línea de comandos y llama a convert():

```
int main(int argc, char **argv) {
  if (argc != 2 || !argv[1][0]) {
    cerr << "Error: Invalid arguments" << endl;
    cerr << "Usage: ./convert [value]" << endl;
    return 1;
  }
  ScalarConverter::convert(argv[1]);
  return 0;
}
```

📌 **Verificaciones**:

- Debe haber **un solo argumento**.
- Si es incorrecto, muestra un mensaje de error.

5. Conclusión

Este código es un **convertidor de tipos** que:

- Detecta si el valor ingresado es char, int, float o double.
- Realiza la conversión.
- Imprime los valores convertidos.

Es una implementación **educativa** para trabajar con conversiones numéricas en C++.