Tipo Carácter

El tipo `char` en Rust es una representación completa y segura de cualquier carácter Unicode.

Representación de Caracteres

Los literales de tipo `char` se escriben entre comillas simples (`'`), distinguiéndose así de las cadenas de texto que utilizan comillas dobles (`"`):

```
let letra: char = 'A';
let minuscula: char = 'ñ';
let numero: char = '7';
let simbolo: char = '©';
let emoji: char = '😎';
let kanji: char = '字';
let arabe: char = 'ع';
```

Métodos de Inspección

Rust proporciona una amplia colección de métodos para analizar las propiedades de un carácter:

Clasificación de Caracteres

```
let c = 'A';

// Propiedades alfabéticas
println!("¿Alfabético?: {}", c.is_alphabetic());      // true
println!("¿Alfanumérico?: {}", c.is_alphanumeric());  // true
println!("¿Mayúscula?: {}", c.is_uppercase());        // true
println!("¿Minúscula?: {}", c.is_lowercase());        // false

// Propiedades numéricas
let numero = '7';
println!("¿Dígito?: {}", numero.is_numeric());        // true
println!("¿Dígito ASCII?: {}", numero.is_ascii_digit()); // true

// Espacios en blanco
let espacio = ' ';
println!("¿Espacio?: {}", espacio.is_whitespace());   // true

// Control y formato
let tab = '\t';
println!("¿Control?: {}", tab.is_control());          // true
```

Secuencias de Escape

Rust soporta varias secuencias de escape para representar caracteres especiales:

Secuencia	Carácter	Descripción
`\n`	Nueva línea	Line feed (LF)
`\r`	Retorno de carro	Carriage return (CR)
`\t`	Tabulador	Tabulador horizontal
`\\`	Barra invertida	Backslash literal
`\'`	Comilla simple	Necesaria en literales char
`\0`	Carácter nulo	Byte cero
`\x7F`	ASCII hex	Carácter ASCII en hexadecimal (2 dígitos)
`\u{1F680}`	Unicode	Carácter Unicode (hasta 6 dígitos hex)

Tabla 1: Secuencias de escape para caracteres

```
// Secuencias comunes
let nueva_linea = '\n';
let tab = '\t';
let comilla = '\'';
let backslash = '\\';

// ASCII hexadecimal
let delete = '\x7F';  // Carácter DEL

// Unicode con código
let cohete = '\u{1F680}';  // 🚀
let corazon = '\u{2764}';  // ❤

println!("Cohete: {}", cohete);
println!("Corazón: {}", corazon);
```

Comparación y Ordenamiento

Los caracteres pueden compararse directamente usando operadores estándar. La comparación se basa en los valores numéricos de sus códigos Unicode:

```
println!("'A' < 'B': {}", 'A' < 'B');         // true (65 < 66)
println!("'a' > 'A': {}", 'a' > 'A');         // true (97 > 65)
println!("'0' < '9': {}", '0' < '9');         // true (48 < 57)
println!("'😎' < '🚀': {}", '😎' < '🚀');   // true (U+1F60E < U+1F680)

let mut letras = vec!['C', 'A', 'B'];
letras.sort();
println!("{:?}", letras);  // ['A', 'B', 'C']
```

Consideraciones de Rendimiento

- Tamaño fijo: Cada `char` ocupa 4 bytes, lo que facilita el acceso directo pero puede desperdiciar memoria si se almacenan grandes cantidades de caracteres ASCII.
- UTF-8 para cadenas: Para texto, Rust usa `String` y `&str` que codifican en UTF-8, siendo más eficientes en memoria.

- Acceso a caracteres: Obtener el n-ésimo carácter de una cadena es O(n) debido a UTF-8, a diferencia de arrays de `char` que son O(1).

```
// Ineficiente para texto largo
let chars: Vec<char> = "texto largo".chars().collect();  // 44 bytes (11 * 4)

// Eficiente
let texto: &str = "texto largo";  // 11 bytes en UTF-8
```

Resumen

El tipo `char` en Rust proporciona:

- Representación segura de valores escalares Unicode (4 bytes por carácter)
- Soporte completo para el rango Unicode de `U+0000` a `U+10FFFF` (excluyendo surrogates)
- Conversión segura entre caracteres y códigos numéricos
- Amplia biblioteca de métodos para clasificación y transformación
- Integración con codificaciones UTF-8 y UTF-16
- Semántica clara que previene errores comunes de manipulación de texto
- Distinción explícita entre caracteres individuales y cadenas de texto

La decisión de usar 4 bytes por `char` refleja la filosofía de Rust: privilegiar la corrección y seguridad sobre la eficiencia de memoria cuando se trata de tipos individuales, mientras que las colecciones como `String` usan codificación variable UTF-8 para optimizar el almacenamiento de texto.