Lectura de caracteres en C

En el lenguaje C, la lectura de caracteres y cadenas de texto se realiza comúnmente utilizando funciones de la biblioteca estándar de entrada/salida (**`stdio.h`**), principalmente **`scanf()`**, **`getchar()`**, **`fgets()`**, y **`gets()`** (aunque **`gets()`** no es recomendable por razones de seguridad).

1. Lectura de un Carácter

Para leer un **único carácter**, la forma más directa es usando **`getchar()`** o **`scanf()`** con el especificador de formato **`%c`**.

1.1. Ejemplo: Usando `scanf("%c")`

Este método **es útil si también estás leyendo otros tipos de datos** (como números) y quiere controlar la entrada con el formato.

```
#include <stdio.h>

void main()
{
    char caracter;

    printf("Introduce un solo caracter: ");

    // Lee un único caracter y lo almacena en 'caracter'
    scanf("%c", &caracter);

    printf("El caracter ingresado es: %c\n", caracter);
}
```

1.2. Ejemplo: Usando `getchar()`

La función **`getchar()`** lee el siguiente carácter disponible de la entrada estándar y es a menudo más eficiente para la lectura de caracteres individuales.

```
#include <stdio.h>

void main()
{
    char caracter;

    printf("Introduce un solo caracter (usando getchar): ");

    // Lee un único caracter de la entrada estándar
    caracter = getchar();

    printf("El caracter ingresado es: %c\n", caracter);
}
```

2. Lectura de Palabras y Texto

En C, una **cadena de texto** (*string*) es simplemente un **arreglo de caracteres** que termina con el carácter nulo (`\0`). La forma en que lee la entrada determina si obtiene una sola palabra o una línea completa.

2.1. Ejemplo: Lectura de una Sola Palabra (usando `scanf("%s")`)

El especificador de formato `%s` en **`scanf()`** lee una secuencia de caracteres hasta que encuentra un **espacio en blanco** (espacio, tabulador, o salto de línea). Esto es ideal para leer una sola palabra.

Importante: Al usar `%s` con **`scanf()`**, **no** se usa el operador de dirección (`&`) para la variable, ya que el nombre del arreglo ya es un puntero a su primer elemento.

```
#include <stdio.h>

void main()
{
    // Declara un arreglo de 50 caracteres para almacenar la palabra
    char palabra[50];

    printf("Introduce una palabra: ");

    // Lee caracteres hasta un espacio y los almacena en 'palabra'
    scanf("%s", palabra);

    printf("La palabra ingresada es: %s\n", palabra);
}
```

3. Ejemplo: Lectura de Texto con Espacios (Frase o Línea Completa)

Para leer una línea de texto completa que **incluya espacios**, debes usar la función **`fgets()`**.

3.1. Usando `fgets()`

`fgets()` es la forma **más segura y recomendada** para leer líneas de texto, ya que le permite especificar el tamaño máximo del *buffer* (el arreglo) para evitar desbordamientos.

```
#include <stdio.h>
#include <string.h> // Necesaria para strlen

void main()
{
```

```
    // Declara un arreglo de 100 caracteres para almacenar la frase
    char frase[100];

    printf("Introduce una frase con varios espacios: ");

    // Lee hasta 99 caracteres (el tamaño - 1 para el '\0') desde stdin (entrada estándar)
    // El '\n' (salto de línea) también se almacena si cabe en el buffer.
    fgets(frase, sizeof(frase), stdin);

    // Opcional: Eliminar el salto de línea que fgets puede incluir
    size_t len = strlen(frase);

    if (len > 0 && frase[len - 1] == '\n') {
        frase[len - 1] = '\0';
    }

    printf("La frase ingresada es: %s\n", frase);
}
```

3.2. Alternativa: Usando `scanf()` con formato de conjunto de exploración (Scan Set)

Puedes forzar a **`scanf()`** a leer hasta que encuentre un salto de línea usando el formato %[^\n], pero es más complejo de manejar y a menudo requiere una limpieza del *buffer* de entrada después.

```
#include <stdio.h>

void main()
{
    char frase_scanf[100];
    printf("Introduce una frase con varios espacios (con scanf): ");

    // Limpia el buffer de entrada para asegurar que el scanf no lea un '\n' anterior
    int c;

    while ((c = getchar()) != '\n' && c != EOF) { }

    // Lee todos los caracteres hasta (^) que encuentre un salto de línea (\n)
    scanf("%[^\n]", frase_scanf);
    printf("La frase ingresada es: %s\n", frase_scanf);
}
```

4. Resumen

Propósito	Función Recomendada	Comentarios
Un Carácter	`getchar()`	Más directo y eficiente.
Una Sola Palabra	`scanf("%s", arr)`	Detiene la lectura en el primer espacio en blanco.
Línea Completa (con Espacios)	`fgets(arr, tamaño, stdin)`	**Más seguro** ya que limita la cantidad de caracteres leídos.

5. Procesamiento Carácter por Carácter con Ciclos

Usaremos las funciones de la biblioteca **`ctype.h`** (como **`islower()`**, **`isupper()`**, **`isdigit()`**, etc.) para verificar el tipo de cada carácter.

La forma más común y segura de procesar la entrada carácter por carácter hasta un límite o hasta que se presione **Enter** es usar la función **`getchar()`** dentro de un ciclo **`while`**.

5.1. Validar SOLO Dígitos Numéricos

Este ejemplo usa un ciclo **`do-while`** para pedir al usuario que ingrese una cadena de caracteres y verifica que cada uno sea un dígito (**`0-9`**).

```
#include <stdio.h>
#include <ctype.h> // Para la función isdigit()

#define MAX_SIZE 20

void main() {
    char input[MAX_SIZE];
    int valido;
    int c; // Usado para consumir el '\n'

    do {
        printf("Ingrese un numero de maximo %d digitos (solo digitos 0-9): ", MAX_SIZE - 1);

        // Reiniciamos el estado de validacion y el indice del array
        valido = 1;
        int i = 0;

        // Leemos caracter por caracter
        while ((c = getchar()) != '\n' && c != EOF && i < MAX_SIZE - 1) {
            char caracter = (char)c;

            // Verificamos si NO es un digito
            if (!isdigit(caracter)) {
                valido = 0; // Marcamos como no valido
                // No rompemos el ciclo para consumir el resto de la linea de entrada
            }

            // Almacenamos el caracter (sea valido o no)
            input[i++] = caracter;
        }

        // Finalizamos la cadena con el caracter nulo
        input[i] = '\0';

        // Si la entrada no fue una nueva linea y el buffer aun tiene caracteres, los limpiamos
        if (c != '\n' && c != EOF) {
            while ((c = getchar()) != '\n' && c != EOF); // Limpiar buffer
        }

        // Si es valido, O si la cadena estaba vacia
        if (valido && i > 0) {
            printf("Entrada valida: %s\n", input);
            break; // Salimos del do-while
        } else {
            // Si la cadena tiene el salto de linea o esta vacia la entrada
```

```
            if (i == 0 |(i == 1 && input[0] == '\n')) {
                 printf("Error: La entrada no puede estar vacia.\n");
            } else if (!valido) {
                printf("Error: '%s' contiene caracteres no validos.
                        Solo se permiten digitos.\n", input);
            }
        }
    } while (1); // Ciclo infinito hasta que se ingrese una entrada valida
}
```

5.2. Validar SOLO Letras Minúsculas

Este ejemplo pide una contraseña y asegura que todos los caracteres alfabéticos ingresados sean minúsculas.

```
#include <stdio.h>
#include <ctype.h> // Para islower()

#define PASS_LEN 15

void main() {
    char password[PASS_LEN];
    int es_minuscula_valida;
    int c;

    do {
        printf("Ingrese una clave (solo minusculas, maximo %d caracteres): ", PASS_LEN - 1);

        es_minuscula_valida = 1;
        int i = 0;

        // Leemos caracter por caracter
        while ((c = getchar()) != '\n' && c != EOF && i < PASS_LEN - 1) {
            char caracter = (char)c;

            // 1. Verificamos si es una letra (isalpha) Y si NO es minuscula (!islower)
            // Esto asegura que si ingresa un numero o un simbolo, se acepta, pero si ingresa
            // una mayuscula, se rechaza la entrada.
            if (isalpha(caracter) && !islower(caracter)) {
                es_minuscula_valida = 0; // Marcamos como no valida
            }

            password[i++] = caracter;
        }

        password[i] = '\0';

        // Limpieza de buffer
        if (c != '\n' && c != EOF) {
            while ((c = getchar()) != '\n' && c != EOF);
        }

        if (i > 0 && es_minuscula_valida) {
            printf("Clave valida: %s\n", password);
            break;
        } else {
             if (i == 0) {
                printf("Error: La clave no puede estar vacia.\n");
            } else if (!es_minuscula_valida) {
                printf("Error: La clave '%s' contiene letras mayusculas.
                     Solo se permiten minusculas.\n", password);
            }
```

```
        }
    } while (1);
}
```

5.3. Funciones Clave de `ctype.h`

Estas funciones te permiten clasificar los caracteres en C:

Función	Descripción	Ejemplo
`isdigit(c)`	¿Es un dígito ('0'-'9')?	Valida números de teléfono.
`isalpha(c)`	¿Es una letra del alfabeto?	Valida campos de solo texto.
`islower(c)`	¿Es una letra minúscula?	Valida el formato de correo electrónico.
`isupper(c)`	¿Es una letra mayúscula?	Valida códigos de productos.
`isalnum(c)`	¿Es alfanumérico (letra o dígito)?	Valida nombres de usuario.

Este enfoque de lectura carácter por carácter te da un **control granular** sobre la entrada, lo que es esencial para la validación y la seguridad de los datos.

Vamos a enfocarnos en la validación de un palíndromo de **frase** ('*dabale arroz a la zorra el abad*'), ya que es el más completo: requiere ignorar espacios y signos de puntuación, y no distinguir entre mayúsculas y minúsculas.

5.4. Ejemplo de Palíndromo de frase en C

Para verificar si una frase es un **palíndromo**, debemos seguir estos pasos clave:

1. **Leer la frase completa** (incluyendo espacios).
2. **Limpiar la frase**: Crear una nueva cadena que solo contenga caracteres alfabéticos, convertidos a una misma caja (minúsculas).
3. **Comparar la frase limpiada** con su versión invertida.

Utilizaremos **`fgets()`** para la entrada y funciones de **`ctype.h`** (como **`tolower()`** y **`isalpha()`**) y **`string.h`** (como **`strlen()`**).

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// Definimos el tamaño maximo de la entrada y de la cadena limpia
#define MAX_ENTRADA 256
#define MAX_LIMPIO  256

// Funcion para verificar si la cadena limpia es un palindromo
int es_palindromo(const char *cadena, int longitud) {
    int inicio = 0;
    int fin = longitud - 1;
```

```
    // Comparamos los caracteres desde los extremos hacia el centro
    while (fin > inicio) {
        if (cadena[inicio++] != cadena[fin--]) {
            return 0; // No es palindromo
        }
    }
    return 1; // Si el ciclo termina, es palindromo
}

void main() {
    char frase_entrada[MAX_ENTRADA];
    char frase_limpia[MAX_LIMPIO];
    int j = 0; // Indice para la frase_limpia

    printf("Ingrese una frase para verificar si es un palindromo:\n");

    // 1. Lectura de la frase
    if (fgets(frase_entrada, MAX_ENTRADA, stdin) == NULL) {
        printf("Error al leer la entrada.\n");
    }

    // 2. Limpieza de la frase (solo letras y a minusculas)
    for (int i = 0; frase_entrada[i] != '\0' && j < MAX_LIMPIO - 1; i++) {
        char c = frase_entrada[i];

        // Ignoramos el salto de linea que agrega fgets
        if (c == '\n') {
            continue;
        }

        // Verificamos si es una letra del alfabeto
        if (isalpha(c)) {
            // Convertimos a minuscula y almacenamos
            frase_limpia[j++] = tolower(c);
        }

        // Si no es un caracter alfabetico, simplemente lo ignoramos (espacios, comas, etc.)
    }

    // Finalizamos la cadena limpia
    frase_limpia[j] = '\0';

    int longitud_limpia = j;

    printf("\nFrase original (entrada): %s", frase_entrada);
    printf("Frase procesada (limpia): %s\n", frase_limpia);

    // 3. Verificacion del palindromo
    if (longitud_limpia == 0) {
        printf("Resultado: La entrada no contenia caracteres alfabeticos.\n");
    } else if (es_palindromo(frase_limpia, longitud_limpia)) {
        printf("\n ¡FELICIDADES! Es un palindromo (capicua).\n");
    } else {
        printf("\n No es un palindromo.\n");
    }
}
```

5.5. Funcionamiento Clave

- **Limpieza de datos**: El ciclo **`for`** recorre la cadena de entrada. La línea **`if (isalpha(c))`** asegura que solo las letras pasen el filtro.

La línea `frase_limpia[j++] = tolower(c)` convierte cualquier mayúscula a minúscula antes de ser almacenada.

- **Verificación**: La función `es_palindromo` utiliza dos punteros (o índices): `inicio` y `fin`. Se mueven uno hacia el otro (`inicio++`, `fin--`). Si en algún momento los caracteres en esas posiciones son diferentes, la función retorna `0` (`false`) inmediatamente. Si el ciclo termina, significa que todos los caracteres coincidieron, y retorna `1` (`true`).

Este código resuelve los desafíos del palíndromo de frase al ignorar automáticamente los espacios y la puntuación y al no distinguir entre mayúsculas y minúsculas.

6. Referencias

6.1. La Referencia Canónica (El Clásico)

Si hay un libro esencial para cualquier programador de C, es este:

Recurso	Enfoque	Notas
El Lenguaje de Programación C (The C Programming Language) - **Brian W. Kernighan y Dennis M. Ritchie** (K&R)	La Biblia del Lenguaje C.	Es la referencia definitiva, escrita por los creadores del lenguaje. Es conciso, profundo y esencial para comprender el **estándar ANSI C** y las bases del manejo de punteros y cadenas, que es el núcleo de tu pregunta.

6.2. Libros para Aprendizaje Estructurado

Estos son excelentes para principiantes e intermedios que buscan una progresión clara en el aprendizaje:

- **C Programming: A Modern Approach** - **K.N. King**: Considerado por muchos como una excelente alternativa moderna y más detallada que K&R, ideal para entender el C actual y buenas prácticas de programación.
- **Aprender a programar en C: de 0 a 99 en un solo libro** - **A. M. Vozmediano**: (Disponible en formato digital gratuito, según la búsqueda). Un recurso que a menudo se usa para una introducción práctica desde la lógica hasta C, que puede ser útil si buscas más ejercicios resueltos.
- **Programación en ANSI C** - **E. Balagurusamy**: Un libro muy popular en la comunidad académica por su enfoque en la versión estandarizada del lenguaje y su claridad.

6.3. Recursos en Línea y Tutoriales (Gratuitos)

Para una referencia rápida, comprensión de funciones específicas (‘ctype.h‘, ‘string.h‘) y tutoriales actualizados.

6.3.1. Referencias Esenciales

- **cplusplus.com (Sección C)**: Aunque el sitio se centra en C++, su sección de referencia para C (como la biblioteca **`stdio.h`**, **`string.h`** y **`ctype.h`**) es muy completa, con ejemplos claros y descripciones de todas las funciones que hemos usado (**`fgets`**, **`getchar`**, **`islower`**, etc.).
- **Manual de Linux (`man pages`)**: Si utilizas Linux o macOS, la documentación oficial del sistema operativo (man pages) es la referencia más precisa para las funciones estándar de C. Simplemente escribe, por ejemplo, **`man islower`** en tu terminal.

6.3.2. Tutoriales y Manuales

- **The C Beginner’s Handbook (Manual para principiantes C)** de **freeCodeCamp** (Traducido al español): Un excelente recurso en línea que sigue el principio 80/20, enseñando los fundamentos y el uso de librerías esenciales como **`string.h`** de manera práctica y rápida.
- **Tutoriales de C en sitios de programación**: Sitios como **Programiz**, **TutorialsPoint** o **GeeksforGeeks** suelen tener guías detalladas sobre el manejo de **`Strings`** en C, punteros y asignación de memoria.