Contents

Penjelasan trimString

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

char* trimString(char* s){
    char* ptr;
    if(!s) return NULL;
    if(!*s) return s;
    for(ptr= s+strlen(s)-1; (ptr>=s) && isspace(*ptr); --ptr);
    *(ptr+1)='\0';
    return s;
}
```

Kode C ini berfungsi untuk melakukan **Right Trim**, yaitu menghapus karakter *whitespace* (spasi, tab, newline) yang ada di bagian **belakang (akhir)** string. Code ini tidak menghapus spasi di awal string.

Berikut adalah penjelasan mendalam baris demi baris:

Ringkasan Logika

Fungsi ini bekerja dengan cara memindai string dari karakter paling belakang, bergerak mundur selama menemukan spasi, lalu memotong string tersebut tepat setelah karakter non-spasi terakhir ditemukan.

Penjelasan Line-by-Line

1. char *trimString(char *s) {

- Mendefinisikan fungsi bernama trimString.
- Menerima parameter char *s: sebuah pointer ke string (array of characters) yang ingin diubah.
- **Penting:** Karena fungsi ini akan mengubah isi string secara langsung (*in-place modification*), string yang dikirim tidak boleh berupa *string literal* (seperti "text"), melainkan harus berupa array yang dialokasikan di memori (seperti char str[] = "text").

2. char *ptr;

- Mendeklarasikan variabel pointer ptr.

- Variabel ini akan digunakan sebagai "kursor" untuk menelusuri string dari belakang ke depan.

3. if (!s) return NULL; // NULL string

- Mengecek apakah pointer s bernilai NULL (tidak menunjuk ke memori manapun).
- Jika ya, fungsi langsung berhenti dan mengembalikan NULL untuk mencegah error (segmentation fault).

4. if (!*s) return s; // empty string

- Mengecek apakah string kosong (""). *s mengambil karakter pertama. Jika karakter pertamanya adalah \0 (null terminator), berarti string kosong.
- Jika kosong, tidak ada yang perlu di-trim, jadi langsung kembalikan string aslinya.

5. for (ptr = s + strlen(s) - 1; (ptr >= s) && isspace(*ptr); --ptr);

Ini adalah inti dari logika pemotongan. Mari kita bedah loop for ini:

- **Inisialisasi (ptr = s + strlen(s) - 1):**
  - Menempatkan pointer ptr tepat di karakter terakhir string (sebelum \0).
  - Contoh: Jika string adalah "A B ", ptr menunjuk ke spasi terakhir.
- **Kondisi ((ptr >= s) && isspace(*ptr)):**
  - Loop akan terus berjalan selama dua syarat terpenuhi:
    1. ptr >= s: Pointer belum melewati batas awal string (mencegah akses memori ilegal di depan string).
    2. isspace(*ptr): Karakter yang sedang ditunjuk adalah whitespace (spasi, \t, \n, dll). Fungsi isspace memerlukan library <ctype.h>.
- **Update (--ptr):**
  - Setiap kali loop berjalan (artinya ketemu spasi), geser pointer ptr mundur satu langkah ke kiri.
- **Body (;):**
  - Perhatikan titik koma di akhir baris. Loop ini **tidak memiliki body**. Semua pekerjaan dilakukan di dalam definisi for itu sendiri. Loop hanya berhenti ketika ptr menunjuk ke karakter **bukan** spasi, atau sudah mentok di awal string.

6. ptr[1] = '\0';

- Setelah loop berhenti, ptr saat ini menunjuk pada karakter **non-spasi** terakhir (atau di posisi sebelum awal string jika semua isinya spasi).
- Kita ingin mengakhiri string tepat **setelah** karakter tersebut.
- ptr[1] mengakses alamat satu langkah di depan ptr. Kita mengisinya dengan \0 (Null Terminator). Ini secara efektif "memotong" string dan membuang semua karakter (spasi) setelahnya.

7. return s;

- Mengembalikan pointer ke awal string yang sudah dimodifikasi.

Visualisasi Eksekusi

Misalkan input string s adalah "Tes " (Tes + 3 spasi).

1. **Awal:** ptr menunjuk ke spasi terakhir (indeks 5).
2. **Loop 1:** isspace benar. ptr mundur ke indeks 4.
3. **Loop 2:** isspace benar. ptr mundur ke indeks 3.
4. **Loop 3:** isspace benar. ptr mundur ke indeks 2 (huruf 's').
5. **Loop 4:** Cek isspace('s') -> **Salah**. Loop berhenti.
6. **Potong:** ptr ada di 's'. ptr[1] (indeks 3) diisi \0.
7. **Hasil:** String menjadi "Tes\0".

Catatan Teknis

Untuk menjalankan kode ini, Anda perlu meng-include header berikut di bagian atas file C Anda:

#include <string.h> // untuk strlen

#include <ctype.h> // untuk isspace

Penjelasan (to split) string

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

int main() {
    char str[] = "C programming is fun";
    char* delim = " ";

    char* token = strtok(str, delim);

    while (token) {
        printf("%s\n", token);
        token = strtok(NULL, delim);
    }
    return 0;
}
```

Kode ini mendemonstrasikan cara melakukan **Tokenizing** (pemecahan string) dalam bahasa C menggunakan fungsi library standar strtok (String Tokenizer).

Sebelum masuk ke penjelasan per baris, ada sedikit typo pada kode yang Anda berikan di bagian char* delim dan je = " ";. Saya asumsikan maksud Anda adalah:

char* delim = " "; (mendefinisikan spasi sebagai pemisah).

Berikut adalah penjelasan **line by line**:

1. char str[] = "C programming is fun";

- Mendeklarasikan array of characters (string) yang berisi kalimat yang akan dipecah.
- **Penting:** Anda menggunakan char str[] (array), bukan char *str (string literal). Ini **krusial** karena strtok bekerja dengan cara **memodifikasi string asli** (mengubah delimiter menjadi \0). Jika Anda menggunakan char *str, program akan *crash* (Segmentation Fault) karena mencoba menulis ke *read-only memory*.

2. char* delim = " ";

- Mendefinisikan karakter pemisah (delimiter). Di sini pemisahnya adalah spasi.
- Setiap kali strtok menemukan karakter ini, ia akan memotong string di situ.

3. char* token = strtok(str, delim);

- Ini adalah **panggilan pertama** ke fungsi strtok.
- **Parameter 1 (str):** Kita memberikan string target yang ingin dipecah.
- **Cara kerja:**
    1. strtok memindai str untuk mencari karakter bukan spasi pertama (awal token).
    2. Lalu ia mencari karakter spasi (delimiter) berikutnya.
    3. Saat spasi ditemukan, strtok mengganti spasi tersebut dengan karakter **Null Terminator (\0)**.
    4. strtok menyimpan posisi "sisa string" di dalam variabel internal (static pointer) miliknya sendiri.
    5. Mengembalikan pointer ke awal token pertama (kata "C").

4. while (token) {

- Memulai loop while.
- Loop akan terus berjalan selama token **tidak bernilai NULL**.
- strtok akan mengembalikan NULL jika sudah tidak ada lagi token yang bisa ditemukan di dalam string.

5. printf("%s\n", token);

- Mencetak token yang baru saja ditemukan ke layar, diikuti baris baru (\n).
- Pada iterasi pertama, ini mencetak "C".

6. token = strtok(NULL, delim);

- Ini adalah bagian paling "ajaib" dari strtok.
- **Parameter 1 (NULL):** Perhatikan kita mengirim NULL, bukan str.
- **Maksudnya:** "Lanjutkan pencarian dari posisi terakhir yang kamu ingat."
- Karena strtok menyimpan *state* (posisi terakhir) secara internal (static), memanggilnya dengan NULL membuatnya melanjutkan pemindaian tepat setelah token sebelumnya berakhir.
- Ia akan mencari delimiter berikutnya, menggantinya dengan \0, dan mengembalikan pointer ke kata berikutnya ("programming", lalu "is", lalu "fun").

7. return 0;

- Menutup fungsi main dan mengakhiri program.

Visualisasi Perubahan Memori

Agar lebih paham bagaimana strtok "merusak" string asli Anda, lihat ilustrasi ini:

Awal:

['C', ' ', 'p', 'r', 'o', ...]

Setelah strtok pertama ("C"):

['C', '\0', 'p', 'r', 'o', ...]

(Spasi diganti \0. Pointer internal strtok sekarang menunjuk ke huruf 'p')

Setelah strtok kedua (NULL, " "):

['C', '\0', 'p', 'r', 'o', ..., 'g', '\0', 'i', 's', ...]

(Spasi setelah 'programming' diganti \0. Pointer internal menunjuk ke 'i')

Output Program

C

programming

is

fun

Catatan untuk Mahasiswa CS (Advanced)

Karena strtok menggunakan **static internal variable** untuk mengingat posisi, fungsi ini **tidak thread-safe**. Jangan gunakan strtok jika Anda membuat program *multithreading*. Sebagai gantinya, gunakan **strtok_r** (reentrant version) yang lebih aman karena kita menyimpan pointernya sendiri secara manual.

String to integer

Dalam bahasa C, karena Anda adalah mahasiswa CS tahun ke-3, saya sarankan untuk tidak hanya sekadar "bisa", tapi juga memahami **cara yang aman dan robust**.

Anda memerlukan library **<stdlib.h>** untuk fungsi-fungsi ini.

Berikut adalah 3 cara utamanya, diurutkan dari yang "paling gampang" hingga "paling aman":

1. atoi (ASCII to Integer) - Cara Cepat (Tapi Bahaya)

Fungsi ini paling simpel, tapi **tidak disarankan** untuk aplikasi serius atau *production code* karena tidak memiliki *error handling*.

```c
#include <stdio.h>
#include <stdlib.h> // Wajib

int main() {
    char str[] = "12345";
    int num = atoi(str);
    printf("Hasil: %d\n", num);
    return 0;
}
```

Kelemahan:

- Jika string bukan angka (misal "hello"), dia mengembalikan 0. Anda tidak bisa membedakan antara input angka 0 atau input error.
- Jika angka terlalu besar (overflow), perilakunya *undefined*.

---

2. strtol (String to Long) - Cara "Professional"

Ini adalah cara yang harus digunakan oleh mahasiswa CS untuk tugas kuliah atau sistem nyata. Fungsi ini memberikan kontrol penuh atas *error handling* dan basis angka (desimal, hex, biner).

Syntax: long int strtol(const char *str, char **endptr, int base);

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str[] = "12345abc"; // Contoh input "kotor"
    char *endptr; // Pointer untuk menangkap di mana angka berhenti

    // Basis 10 untuk desimal
    long num = strtol(str, &endptr, 10);

    if (str == endptr) {
        printf("Error: Tidak ada angka yang ditemukan.\n");
    } else if (*endptr != '\0') {
```

```
        printf("Info: Angka valid '%ld', tapi ada sisa karakter '%s'\n", num,
endptr);
    } else {
        printf("Sukses: Full angka '%ld'\n", num);
    }

    return 0;
}
```

Kenapa strtol lebih baik?

1. **Deteksi Error:** Dengan endptr, Anda tahu jika user memasukkan "123mobil". strtol akan mengambil 123, dan endptr akan menunjuk ke "mobil". Anda bisa menolak input ini jika ingin validasi ketat.
2. **Basis Angka:** Anda bisa ubah parameter ke-3 menjadi 16 untuk Hexadecimal, 2 untuk Binary, atau 0 (otomatis deteksi 0x...).

---

3. sscanf - Cara Parsing

Berguna jika angka tersebut berada di dalam kalimat yang terformat.

```
#include <stdio.h>

int main() {
    char str[] = "Umur: 21 tahun";
    int umur;

    // Mengambil angka setelah kata "Umur: "
    if (sscanf(str, "Umur: %d", &umur) == 1) {
        printf("Umur terdeteksi: %d\n", umur);
    }
    return 0;
}
```

Ringkasan: Mana yang harus saya pakai?

Fitur	**atoi**	**strtol**	**sscanf**
Kemudahan	Sangat Mudah	Sedang	Sedang
Keamanan	Rendah	**Tinggi**	Sedang
Deteksi Error	Tidak Ada	**Lengkap**	Terbatas

Fitur	**atoi**	**strtol**	**sscanf**
Kapan dipakai	Competitive Programming (cepat)	**Tugas Kuliah / Software Development**	Parsing format string

Untuk level Anda saat ini, biasakan menggunakan **strtol**.