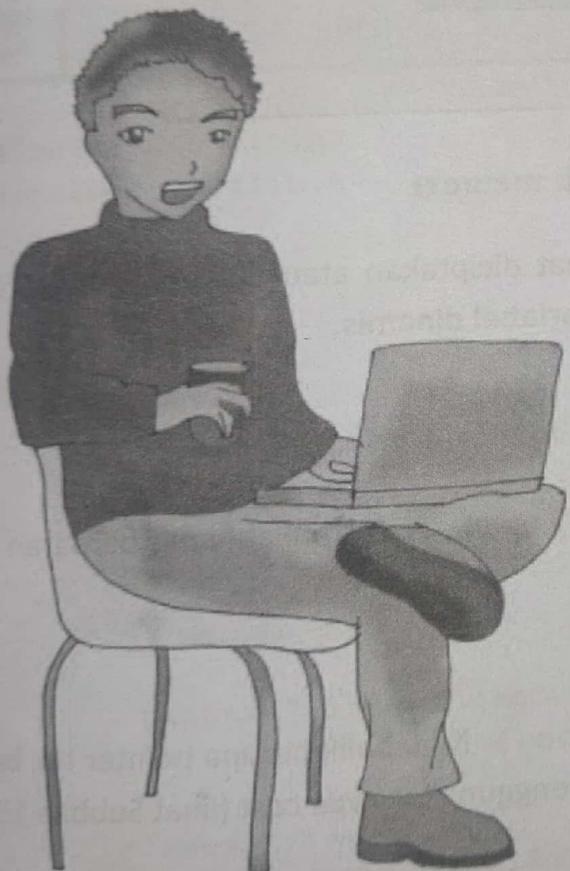


BAB

15

Variabel Dinamis dan Struktur Data

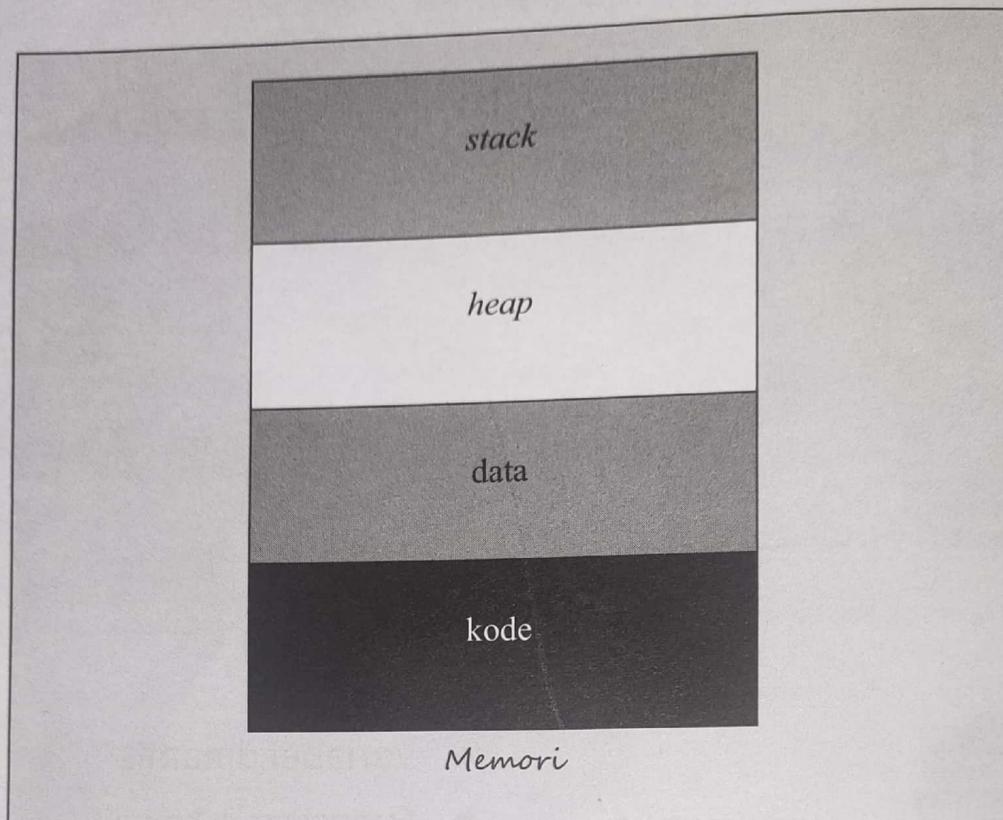
Materi:



- Variabel dinamis
- Fungsi malloc()
- Fungsi free()
- Aplikasi senarai berantai
- Membentuk senarai berantai
- Menambah simpul
- Menampilkan isi senarai berantai
- Mencari data dalam simpul
- Menghapus suatu simpul

15.1 Variabel Dinamis

Data dapat ditempatkan secara dinamis dalam tempat yang disebut **heap**. Lokasi heap ditunjukkan di Gambar 15.1.



Gambar 15.1 Tata letak memori

Dengan memanfaatkan ruang di heap, suatu variabel dapat diciptakan atau dihapus ketika program dijalankan. Oleh karena itu, variabel demikian dinamakan **variabel dinamis**.

15.2 Fungsi malloc()

Kunci penciptaan variabel dinamis di C adalah fungsi `malloc()`. Fungsi inilah yang berperan untuk memesan memori di heap. Prototipnya seperti berikut:

```
void *malloc(int jumlah_byte);
```

Jumlah byte yang dipesan ditentukan dalam argumen `malloc()`. Nilai balik berupa pointer tak bertipe. Oleh karena itu, nilai balik ini perlu dikonversikan dengan menggunakan *type cast* (lihat Subbab 13.4).

Kemungkinan nilai balik `malloc()` ada dua, yaitu:

- Nilai balik berupa NULL. Hal ini menyatakan bahwa ruang bebas di heap sudah habis.
- Nilai balik tidak NULL. Hal ini menyatakan pemesanan atau alokasi memori berhasil dilaksanakan dan nilai balik tersebut adalah pointer yang menunjuk ke lokasi di heap. Nah, lokasi itulah yang merupakan variabel dinamis.

15.3 Fungsi free()

Untuk kepentingan membebaskan memori di heap yang dipesan melalui malloc(), Anda bisa memanfaatkan fungsi bernama free(). Bentuk prototipenya sebagai berikut:

```
void free(void *ptr);
```

Dalam hal ini, ptr adalah pointer yang menunjuk ke memori yang akan dibebaskan.

Catatan



- Pembebasan memori di heap untuk data yang tidak diperlukan lagi adalah dalam rangka untuk menjaga agar memori untuk variabel dinamis tidak habis. Setelah dibebaskan, memori dapat dimanfaatkan oleh variabel lain yang perlu diciptakan sewaktu-waktu. Oleh karena itu, sebaiknya biasakan untuk membebaskan memori yang digunakan oleh variabel dinamis di heap sekiranya variabel tidak diperlukan lagi.
- Prototipe malloc() dan free() ada di stdlib.h. Jangan lupa menyertakan #include <stdlib.h> kalau melibatkan fungsi-fungsi tersebut.

Contoh yang menunjukkan penggunaan fungsi malloc() dan free() ditunjukkan di program berikut:



Program 15.1: dinamis.c

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

int main()
{
    char *ptr;

    ptr = (char *) malloc(500 * sizeof(char));

    if (!ptr)
        printf("Memori tak cukup\n");
    else
    {
        printf("Alokasi memori sudah dilakukan\n");
        printf("-----\n");

        free(ptr);
        printf("Lokasi memori sudah dibebaskan kembali\n");
    }

    return 0;
}
```

Akhir Program

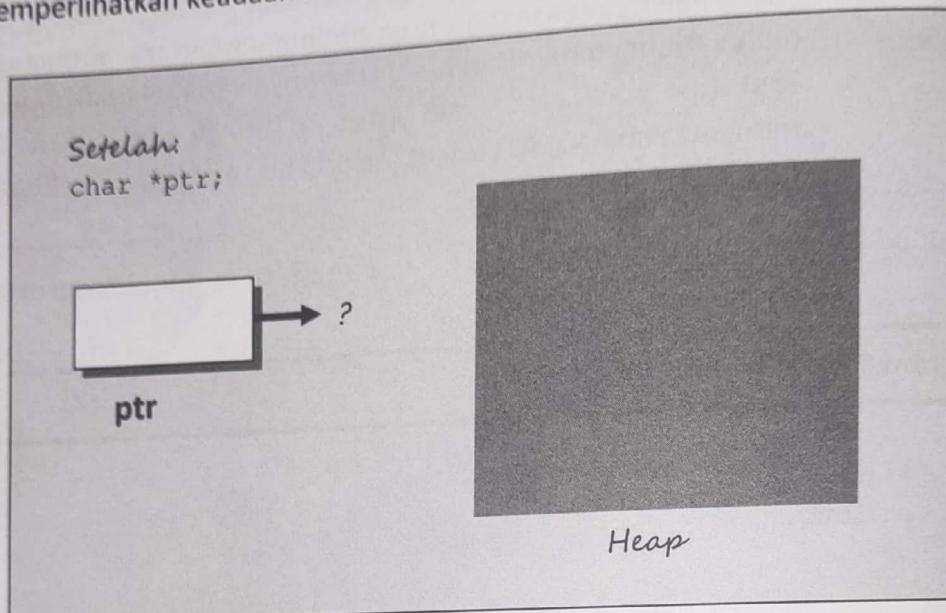
Hasil eksekusi:

Alokasi memori sudah dilakukan
Lokasi memori sudah dibebaskan kembali

Perhatikan, mula-mula ptr dideklarasikan sebagai pointer yang menunjuk ke tipe char melalui:

```
char *ptr;
```

Gambar 15.2 memperlihatkan keadaan setelah pernyataan di atas dieksekusi.



Gambar 12.2 Deklarasi variabel pointer

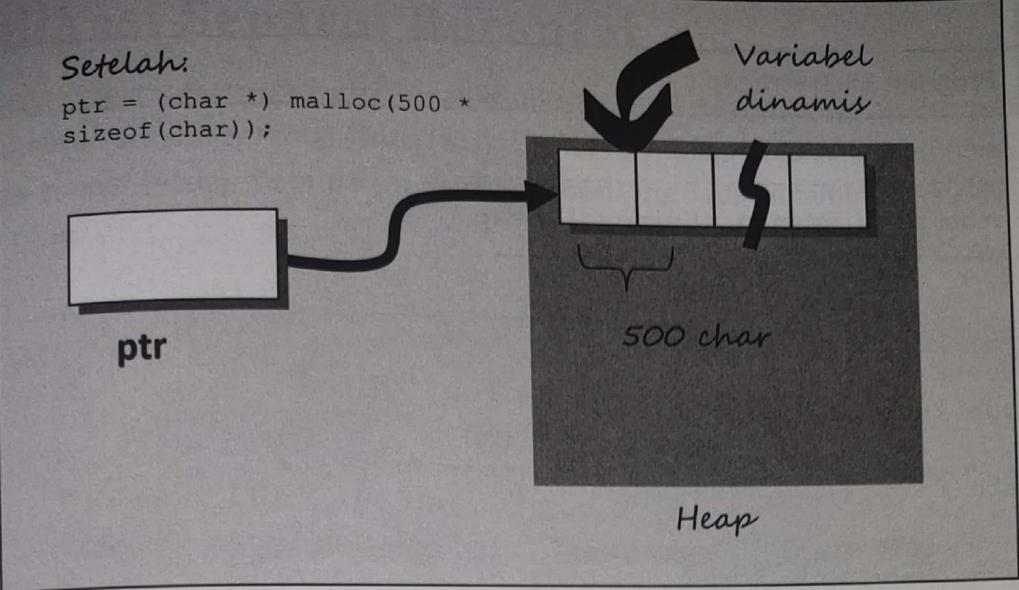
Selanjutnya, dilakukan pemesanan sebanyak 500 karakter melalui pernyataan

```
ptr = (char *) malloc(500 * sizeof(char));
```

Dalam hal ini, `500 * sizeof(char)` menentukan jumlah byte di heap yang dipesan. Kalau alokasi memori berhasil dilaksanakan, ptr akan menunjuk ke lokasi memori tersebut. Gambar 15.3 memperlihatkan keadaan jika memori di heap berhasil dialokasikan.

Setelah:

```
ptr = (char *) malloc(500 *  
sizeof(char));
```



Gambar 15.3 Penciptaan variabel dinamis

Akan tetapi, adakalanya memori tidak berhasil dialokasikan. Oleh karena itu, diperlukan pemeriksaan untuk menentukan alokasi memori berhasil dilakukan atau tidak melalui:

```
if (!ptr)  
    printf("Memori tak cukup\n");  
else  
{  
    ...  
}
```

Dalam hal ini, if (!ptr) menyatakan kalau pointer ptr berisi NULL atau berarti bahwa memori yang dipesan tidak berhasil dilakukan. Pada keadaan demikian, pernyataan

```
printf("Memori tak cukup\n");
```

akan dieksekusi. Adapun bagian else akan dijalankan sekiranya memori berhasil dialokasikan.

Pada contoh di depan, pernyataan

```
free(ptr);
```

digunakan untuk membebaskan memori di heap yang ditunjuk oleh ptr.

Program berikut memperlihatkan contoh lengkap penggunaan variabel dinamis untuk menyimpan data string.



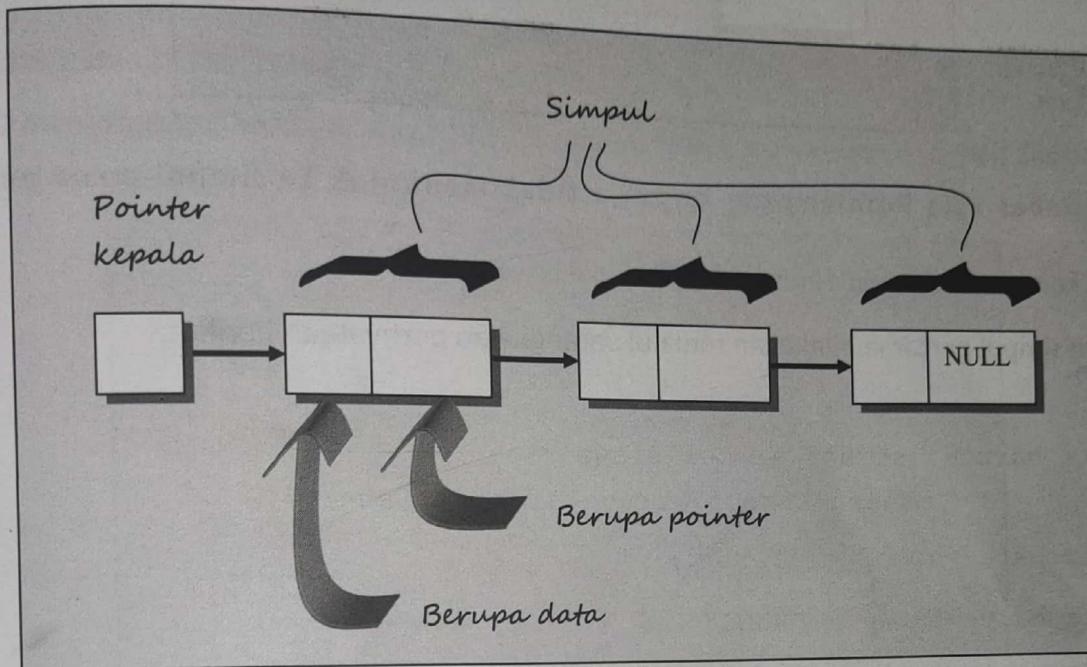
Program 15.2: strheap.c

```
// -----  
// Contoh penyimpanan string di heap  
// -----  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main()  
{  
    char *ptr;  
  
    ptr = (char *) malloc(500 * sizeof(char));  
  
    if (!ptr)  
        printf("Memori tak cukup\n");  
    else  
    {  
        // Letakkan string ke heap  
        strcpy(ptr, "Saat yang tepat untuk memulai\n");  
        strcat(ptr, "sesuatu yang baru.\n");  
        strcat(ptr, "Jangan ragu-ragu!");  
  
        // Tampilkan string  
        printf("%s", ptr);  
  
        free(ptr); // Bebaskan memori  
    }  
  
    return 0;  
}
```

Akhir Program

15.4 Aplikasi Senarai Berantai

Contoh penerapan alokasi memori secara dinamis adalah untuk keperluan pembuatan suatu struktur data yang dinamakan senarai berantai (*linked list*). Gambar 15.4 menunjukkan contoh senarai berantai yang berisi tiga simpul (*node*). Data dalam senarai seperti itu dapat ditambah atau dikurangi ketika program dijalankan.



Gambar 15.4 Bentuk suatu senarai berantai

15.4.1 Membentuk Senarai Berantai

Proses pembentukan senarai berantai dapat dilihat pada penjelasan berikut. Pertama-tama, dilakukan pendefinisian seperti berikut:

```
#define PANJANG_NOMOR 5
#define PANJANG_NAMA 20

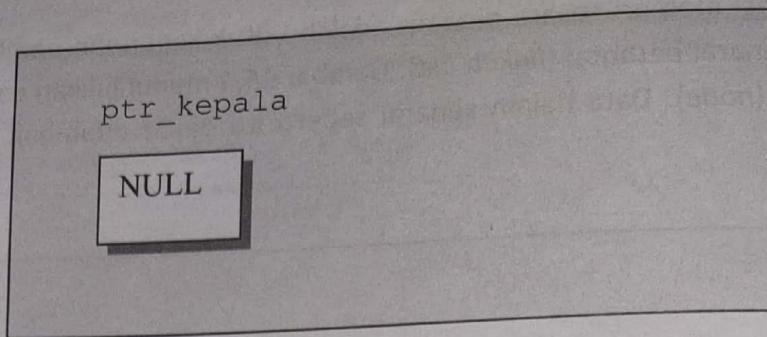
struct simpul_siswa
{
    char nomor[PANJANG_NOMOR + 1];
    char nama[PANJANG_NAMA + 1];
    struct simpul_siswa *lanjutan;
};
```

Dalam hal ini, tipe_simpul berisi:

- Informasi berupa nomor dan nama.
- Pointer bernama lanjutan, yang menunjuk ke objek bertipe simpul_data.

Selanjutnya, dilakukan deklarasi pointer bernama ptr_kepala:

```
struct simpul_siswa *ptr_kepala = NULL;
```



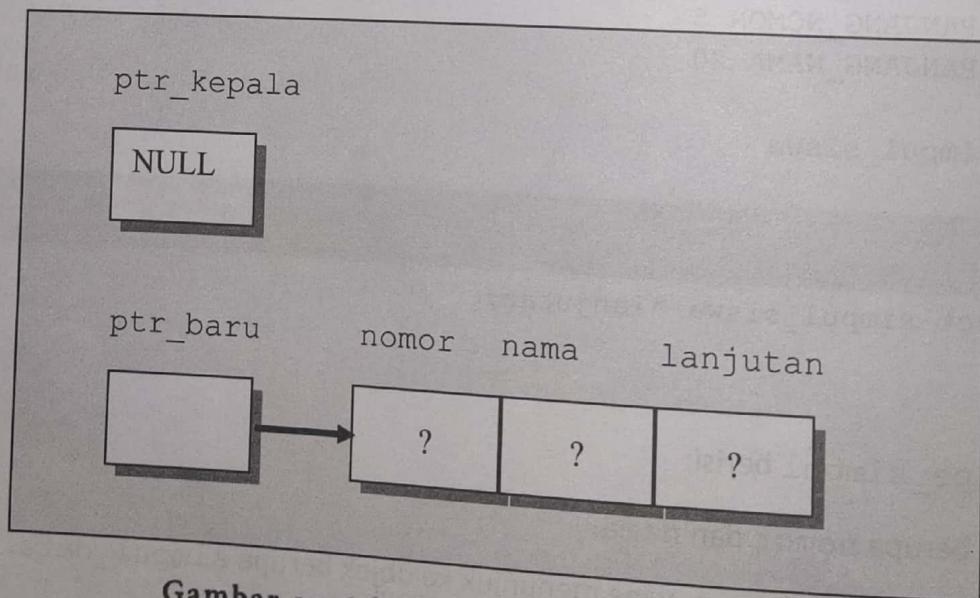
Gambar 15.5 Pointer `ptr_kepala` tidak menunjuk ke simpul mana pun

NULL adalah konstanta yang didefinisikan di file `stdio.h`.

Pembentukan simpul pertama dilakukan melalui serangkaian pernyataan berikut:

- ① `ptr_baru = (struct simpul_siswa *) malloc(sizeof(struct simpul_siswa));`
- ② `ptr_baru->nomor = "11511";`
- ③ `ptr_baru->nama = "RUDI";`
- ④ `ptr_baru->lanjutan = ptr_kepala;`
- ⑤ `ptr_kepala = ptr_baru;`

Sesudah pernyataan pertama (dengan anggapan alokasi memori dalam heap berhasil dilaksanakan) maka akan terbentuk diagram seperti terlihat di Gambar 15.6.



Gambar 15.6 Satu simpul baru diciptakan

Pada gambar di atas, tanda ? menyatakan isinya belum ditentukan.

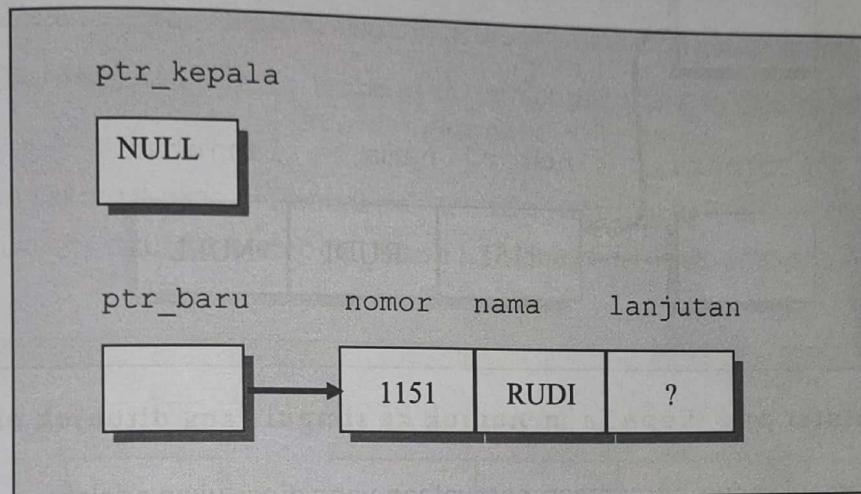
pernyataan

```
ptr_baru->nomor = "11511";
ptr_baru->nama = "RUDI";
```

menyebabkan:

- Anggota nomor berisi "11511".
- Anggota nama berisi "RUDI".

Gambar 15.7 menunjukkan keadaan secara visual setelah kedua pernyataan di depan dijalankan.

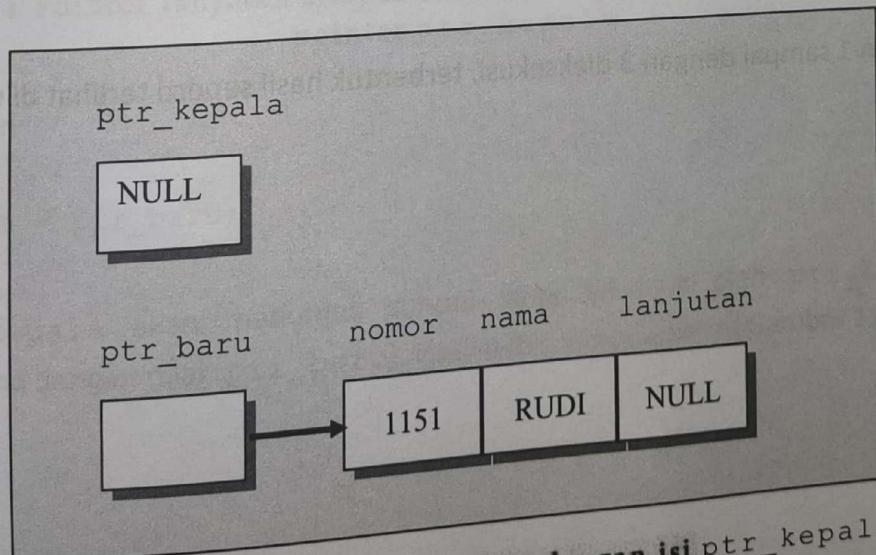


Gambar 15.7 Simpul baru diisi dengan data

Selanjutnya, pernyataan

```
ptr_baru->lanjutan = ptr_kepala;
```

membuat anggota `lanjutan` yang ditunjuk oleh `ptr_baru` diisi dengan nilai `ptr_kepala` (yaitu `NULL`). Hasilnya terlihat seperti pada Gambar 15.8.

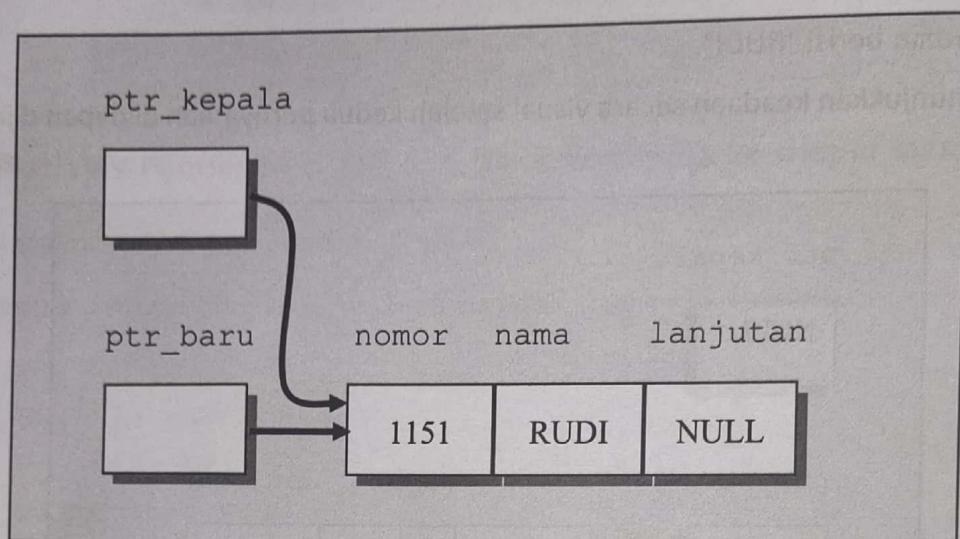


Gambar 15.8 Pointer lanjutan milik simpul diisi dengan isi `ptr_kepala`, yang bernilai `NULL`

Pernyataan

```
ptr_kepala = ptr_baru;
```

dipakai untuk mengisikan nilai pada `ptr_baru` ke `ptr_kepala`. Sebagai akibatnya, `ptr_kepala` akan menunjuk ke simpul yang ditunjuk oleh `ptr_baru`. Hasilnya diperlihatkan di Gambar 15.9.

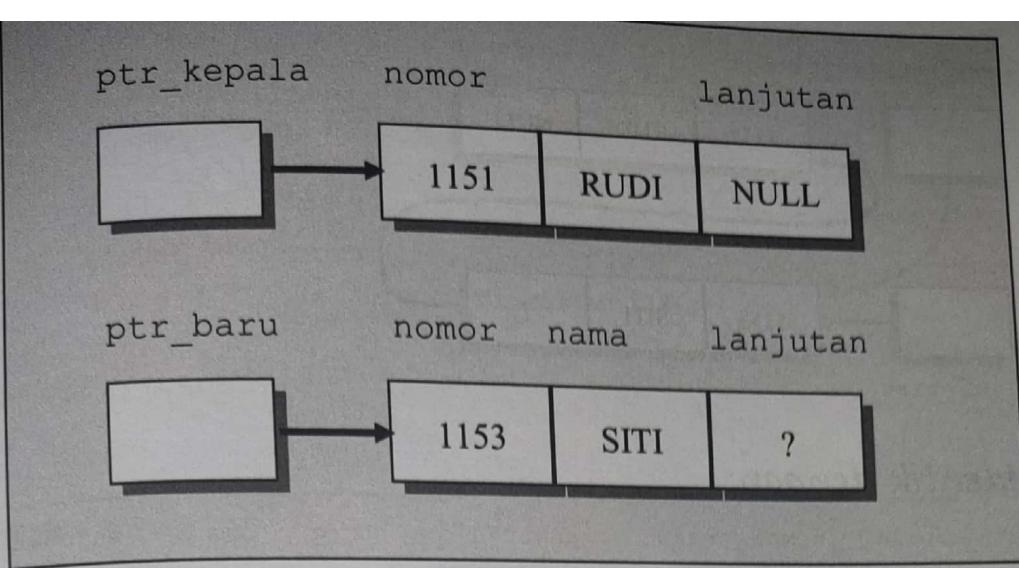


Gambar 15.9 Pointer `ptr_kepala` menunjuk ke simpul yang ditunjuk oleh `ptr_baru`

Untuk membentuk simpul kedua, pernyataan-pernyataan yang diperlukan adalah:

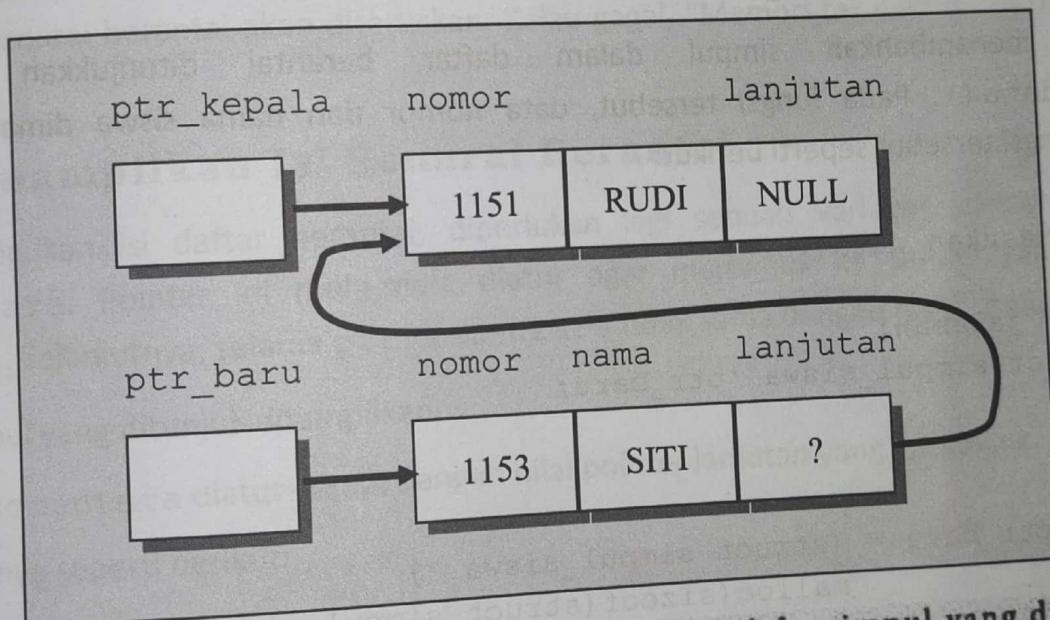
- 1 `ptr_baru = (struct simpul_siswa *) malloc(sizeof(struct simpul_siswa));`
- 2 `ptr_baru->nomor = "11512";`
- 3 `ptr_baru->nama = "SITI";`
- 4 `ptr_baru->lanjutan = ptr_kepala;`
- 5 `ptr_kepala = ptr_baru;`

Sesudah pernyataan 1 sampai dengan 3 dieksekusi, terbentuk hasil seperti terlihat di Gambar 15.10.



Gambar 15.10 Pointer `ptr_baru` menunjuk ke simpul yang baru ditambahkan dan diisi dengan data

Adapun pernyataan keempat akan membuat pointer lanjutan yang ditunjuk oleh `ptr_baru` akan menunjuk ke simpul yang ditunjuk oleh `ptr_kepala`, sebagaimana diperlihatkan di Gambar 15.11.

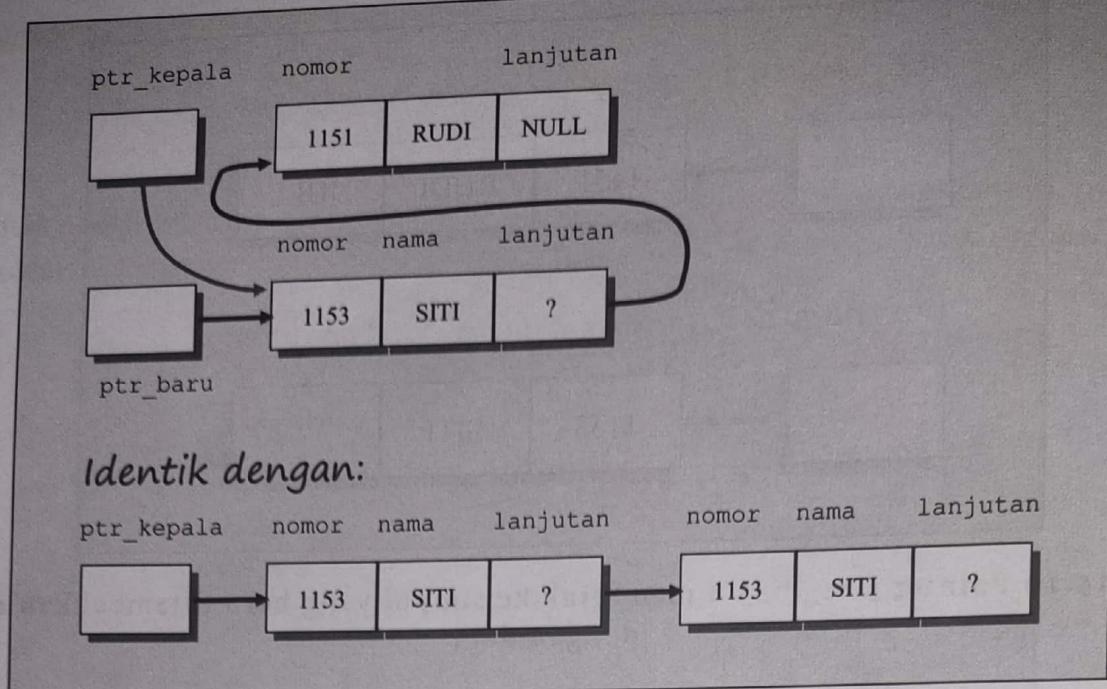


Gambar 15.11 Pointer lanjutan simpul baru menunjuk ke simpul yang ditunjuk oleh pointer `ptr_kepala`

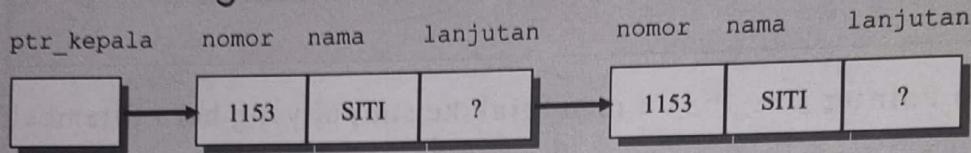
Adapun pernyataan

```
ptr_kepala = ptr_baru;
```

membuat `ptr_kepala` akan menunjuk simpul yang ditunjuk oleh `ptr_baru` (sebab nilai `ptr_kepala` sama dengan nilai `ptr_baru`). Hasil akhir ditunjukkan di Gambar 15.12.



Identik dengan:



Gambar 15.12 Senarai berantai mengandung dua simpul

15.4.2 Menambah Simpul

Cara untuk menambahkan simpul dalam daftar berantai ditunjukkan pada fungsi pemasukan_data(). Pada fungsi tersebut, data nomor dan nama siswa dimasukkan melalui keyboard. Isi fungsi tersebut seperti berikut:

```
void pemasukan_data(void)
{
    char jawaban;
    struct simpul_siswa *ptr_baru;

    do
    {
        ptr_baru = (struct simpul_siswa *)
                    malloc(sizeof(struct simpul_siswa));

        if (ptr_baru)
        {
            masukan_string("Nomor siswa : ",
                           ptr_baru->nomor,
                           PANJANG_NOMOR);
            masukan_string("Nama siswa : ",
                           ptr_baru->nama,
                           PANJANG_NAMA);

            ptr_baru->lanjutan = ptr_kepala;
            ptr_kepala = ptr_baru;

            printf("Mau memasukkan data lagi (Y/T)? ");
            do
            {

```

```

        jawaban = toupper(getchar());
        fflush(stdin); // Hapus sisa data di keyboard
    } while (!(jawaban == 'Y' || jawaban == 'T'));
}
else
{
    printf("Memori tak cukup!");
    break; // Keluar dari do-while
}
} while (jawaban == 'Y');
}

```

Catatan



Fungsi `masukan_string()` yang digunakan pada fungsi `pemasukan_data()` merupakan fungsi untuk memasukkan data string. Definisinya dapat dilihat di program `senarai.c`.

Pada definisi fungsi `pemasukan_data()` terlihat bahwa setiap kali suatu simpul akan diciptakan, memori di heap dipesan terlebih dahulu. Jika alokasi memori berhasil dilakukan, proses penambahan simpul dalam senarai berantai akan diteruskan. Kalau gagal, "Memori tak cukup!" akan ditampilkan di layar.

15.4.3 Menampilkan Isi Senarai Berantai

Untuk menampilkan isi daftar berantai, diperlukan lagi sebuah variabel pointer lokal bernama `ptr_sementara`. Pointer ini mula-mula diatur agar menunjuk ke simpul yang ditunjuk oleh `ptr_kepala`. Selanjutnya, selama `ptr_sementara` tidak sama dengan `NULL`, maka:

- Isi simpul yang ditunjuk ditampilkan.
- `ptr_sementara` diatur sesuai dengan nilai pointer lanjutan yang dia tunjuk.

Implementasinya seperti berikut:

```

void tampilan_data(void)
{
    struct simpul_siswa *ptr_sementara;
    printf("\nIsi senarai berantai\n");
    ptr_sementara = ptr_kepala;
    while (ptr_sementara)
    {
        printf("%s %s\n", ptr_sementara->nomor,
               ptr_sementara->nama);
        ptr_sementara = ptr_sementara->lanjutan;
    }
}

```

Contoh program untuk membuat senarai berantai dan menampilkan isinya dapat dilihat di bawah ini:



Program 15.3: senarai.c

```
// -----
// Contoh pembuatan senarai berantai
// dan cara menampilkan isinya
// -----
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define PANJANG_NOMOR 5
#define PANJANG_NAMA 20

struct simpul_siswa
{
    char nomor[PANJANG_NOMOR + 1];
    char nama[PANJANG_NAMA + 1];
    struct simpul_siswa *lanjutan;
};

struct simpul_siswa *ptr_kepala = NULL; // Ujung senarai

void pemasukan_data(void);
void masukan_string(char *keterangan, char *masukan,
                     int panjang_maks);
void tampilkan_data(void);

int main()
{
    pemasukan_data();
    tampilkan_data();

    return 0;
}
// ----- Akhir program utama
// -----
// Definisi fungsi untuk menangani pemasukan data
// -----



void pemasukan_data(void)
{
    char jawaban;
    struct simpul_siswa *ptr_baru;
    do
    {
        ptr_baru = (struct simpul_siswa *)
            malloc(sizeof(struct simpul_siswa));
        // -----
```

```

if (ptr_baru)
{
    masukan_string("Nomor siswa : ",
                   ptr_baru->nomor,
                   PANJANG_NOMOR);
    masukan_string("Nama siswa : ",
                   ptr_baru->nama,
                   PANJANG_NAMA);

    ptr_baru->lanjutan = ptr_kepala;
    ptr_kepala = ptr_baru;

    printf("Mau memasukkan data lagi (Y/T) ? ");
    do
    {
        jawaban = toupper(getchar());
        fflush(stdin); // Hapus sisa data di keyboard
    } while (!(jawaban == 'Y' || jawaban == 'T'));
}
else
{
    printf("Memori tak cukup!");
    break; // Keluar dari do-while
}
} while (jawaban == 'Y');

}

// -----
// Definisi fungsi untuk membaca string
// -----
void masukan_string(char *keterangan, char *masukan,
                     int panjang_maks)
{
    char st[256];

    do
    {
        printf(keterangan); // Tampilkan keterangan
        gets(st);           // Baca string
        if (strlen(st) > panjang_maks)
            printf("Terlalu panjang. Maksimal %d karakter\n",
                   panjang_maks);
    } while (strlen(st) > panjang_maks);

    // Salin string st ke mauskan
    strcpy(masukan, st);
}

```

15.4.4 Mencari Data dalam Simpul

pembahasan berikut menerangkan cara mencari suatu data dalam senarai berantai. Hal ini sangat diperlukan terutama untuk keperluan menghapus suatu simpul atau mengubah isi suatu simpul. Untuk keperluan ini, diperlukan dua tambahan variabel pointer berupa:

- `ptr_pos_data` untuk mencari simpul yang berisi data yang dicari.
- `ptr_pendahulu` untuk menunjuk ke simpul yang merupakan pendahulu simpul yang dicari.

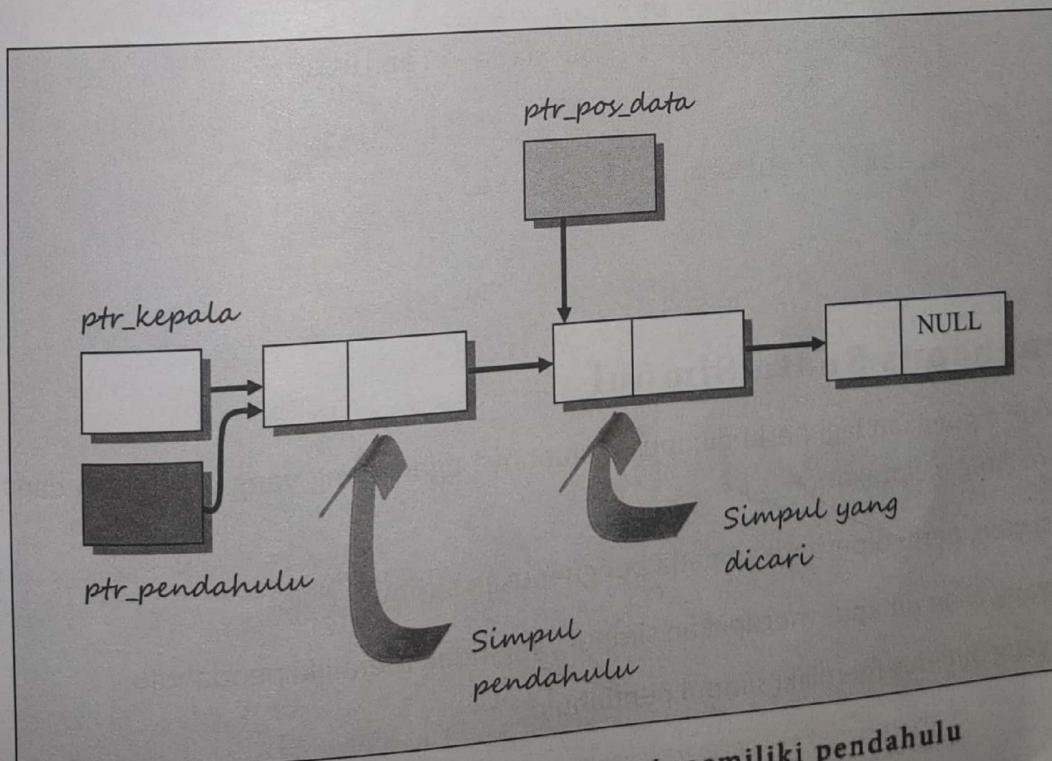
Deklarasinya seperti berikut:

```
struct simpul_siswa *ptr_pos_data;  
struct simpul_siswa *ptr_pendahulu;
```

Aturan yang digunakan untuk mencari suatu data dalam senarai berantai adalah:

1. Jika data yang dicari tidak ditemukan, `ptr_pos_data` berisi `NULL`.
2. Jika data yang dicari ditemukan, maka:
 - a. Pointer `ptr_pos_data` menunjuk ke simpul yang dicari.
 - b. Apabila simpul yang ditunjuk `ptr_pos_data` merupakan simpul yang ditunjuk oleh `ptr_kepala` (berarti simpul tidak mempunyai pendahulu), `ptr_pendahulu` berisi `NULL`.
 - c. Apabila simpul yang ditunjuk `ptr_pos_data` merupakan simpul yang tidak ditunjuk oleh `ptr_kepala`, `ptr_pendahulu` berisi alamat simpul pendahulunya yang ditunjuk oleh `ptr_pos_data`.

Lihat penjelasan secara visual di Gambar 15.13 dan 15.14.



Gambar 15.13 Simpul yang dicari memiliki pendahulu

```
// Definisi fungsi untuk menampilkan
// isi senarai berantai
// -----
void tampilkan_data(void)
{
    struct simpul_siswa *ptr_sementara;

    printf("\nIsi senarai berantai\n");
    ptr_sementara = ptr_kepala;
    while (ptr_sementara)
    {
        printf("%s %s\n", ptr_sementara->nomor,
               ptr_sementara->nama);
        ptr_sementara = ptr_sementara->lanjutan;
    }
}
```

Akhir Program

Contoh hasil eksekusi:

```
Nomor siswa : 11896
Nama siswa : Andi Damara
Mau memasukkan data lagi (Y/T) ? Y
Nomor siswa : 11897
Nama siswa : Dian Anjani
Mau memasukkan data lagi (Y/T) ? y
Nomor siswa : 11898
Nama siswa : Fahmi Idris
Mau memasukkan data lagi (Y/T) ? y
Nomor siswa : 118999
Terlalu panjang. Maksimal 5 karakter
Nomor siswa : 11899
Nama siswa : Sita Aini
Mau memasukkan data lagi (Y/T) ? t

Isi senarai berantai
11899 Sita Aini
11898 Fahmi Idris
11897 Dian Anjani
11896 Andi Damara
```

Pada senarai berantai yang telah dibahas, urutan data pada saat dimasukkan akan berlawanan dengan urutan saat ditampilkan (perhatikan eksekusi program `senarai.c`). Data terakhir yang dimasukkan akan menjadi data pertama saat ditampilkan. Karena sifatnya demikian, struktur data senarai berantai dikenal dengan sebutan *last in last out* (LIFO) atau yang masuk terakhir akan keluar pertama kali.

15.4.4 Mencari Data dalam Simpul

Pembahasan berikut menerangkan cara mencari suatu data dalam senarai berantai. Hal ini sangat diperlukan terutama untuk keperluan menghapus suatu simpul atau mengubah isi suatu simpul. Untuk keperluan ini, diperlukan dua tambahan variabel pointer berupa:

- `ptr_pos_data` untuk mencari simpul yang berisi data yang dicari.
- `ptr_pendahulu` untuk menunjuk ke simpul yang merupakan pendahulu simpul yang dicari.

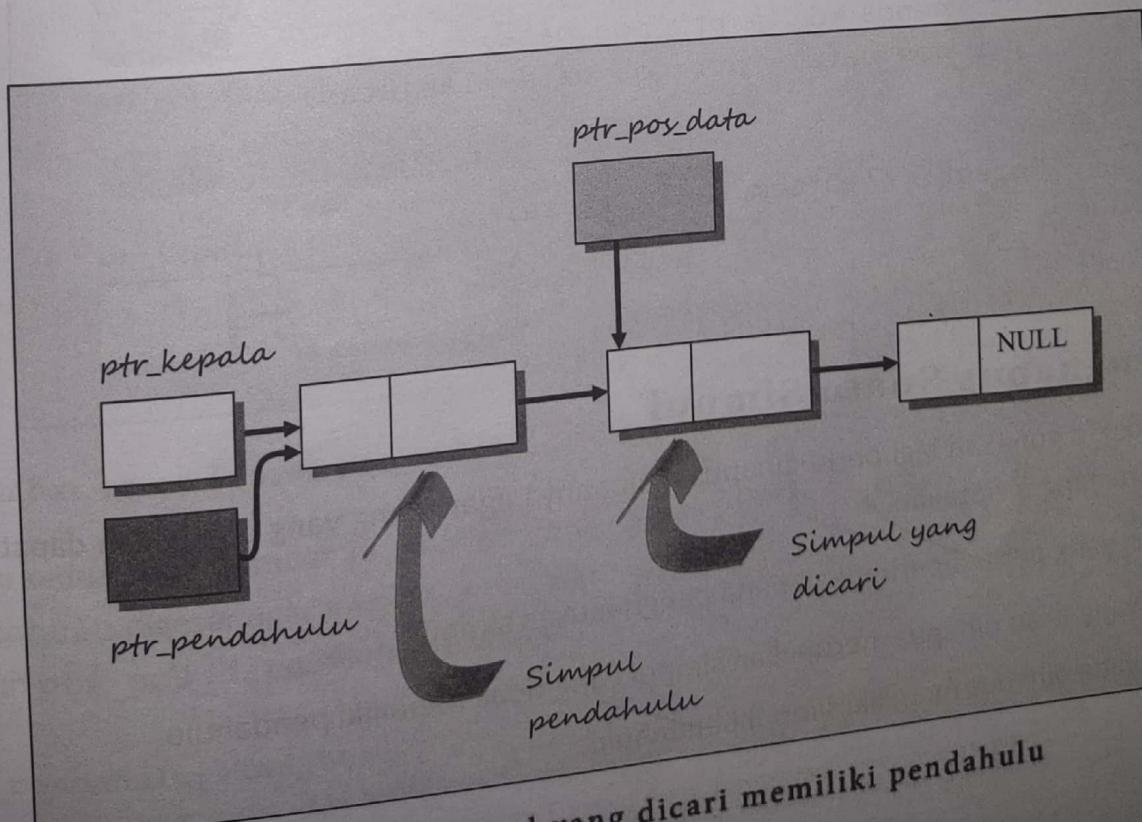
Deklarasinya seperti berikut:

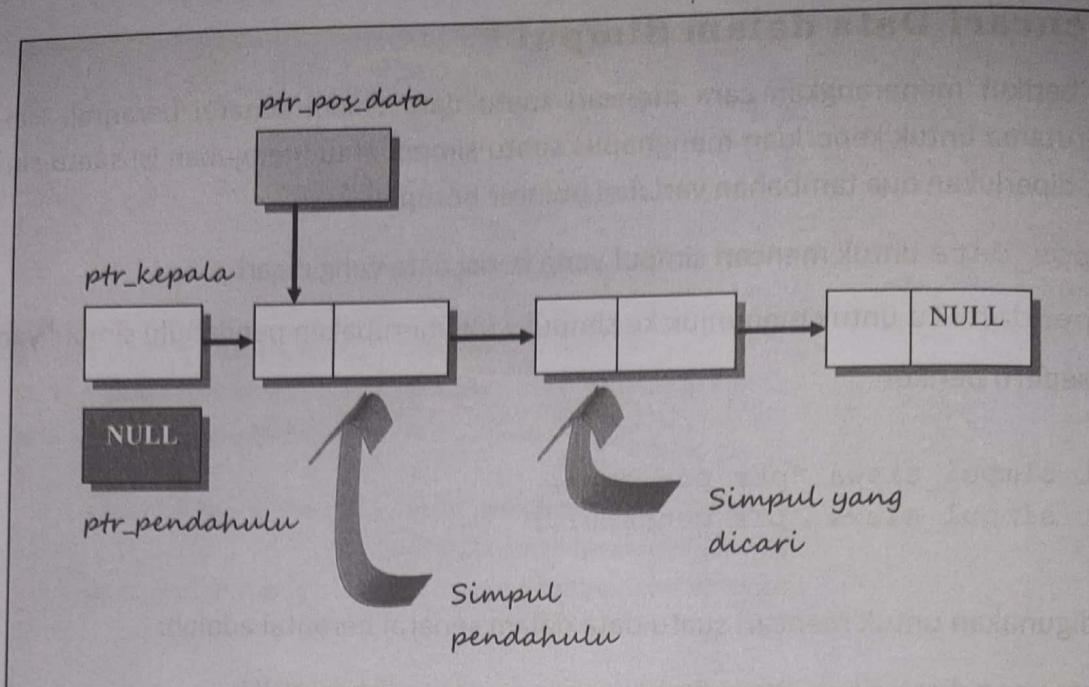
```
struct simpul_siswa *ptr_pos_data;  
struct simpul_siswa *ptr_pendahulu;
```

Aturan yang digunakan untuk mencari suatu data dalam senarai berantai adalah:

1. Jika data yang dicari tidak ditemukan, `ptr_pos_data` berisi `NULL`.
2. Jika data yang dicari ditemukan, maka:
 - a. Pointer `ptr_pos_data` menunjuk ke simpul yang dicari.
 - b. Apabila simpul yang ditunjuk `ptr_pos_data` merupakan simpul yang ditunjuk oleh `ptr_kepala` (berarti simpul tidak mempunyai pendahulu), `ptr_pendahulu` berisi `NULL`.
 - c. Apabila simpul yang ditunjuk `ptr_pos_data` merupakan simpul yang tidak ditunjuk oleh `ptr_kepala`, `ptr_pendahulu` berisi alamat simpul pendahulunya yang ditunjuk oleh `ptr_pos_data`.

Lihat penjelasan secara visual di Gambar 15.13 dan 15.14.





Gambar 15.14 Simpul yang dicari tidak memiliki pendahulu

Implementasinya seperti berikut:

```

void cari_data(char *nama)
{
    ptr_pendahulu = NULL;
    ptr_pos_data = ptr_kepala;

    while (ptr_pos_data)
    {
        if (strcmp(nama, ptr_pos_data->nama) != 0)
        {
            ptr_pendahulu = ptr_pos_data;
            ptr_pos_data = ptr_pos_data->lanjutan;
        }
        else
            break; // Ditemukan - Selesai
    }
}

```

15.4.5 Menghapus Suatu Simpul

Simpul yang tidak digunakan lagi perlu dihapus. Tujuannya agar ruang yang digunakan dapat ditempati oleh simpul baru yang diciptakan.

Ada dua kondisi yang perlu diperhatikan pada penghapusan simpul:

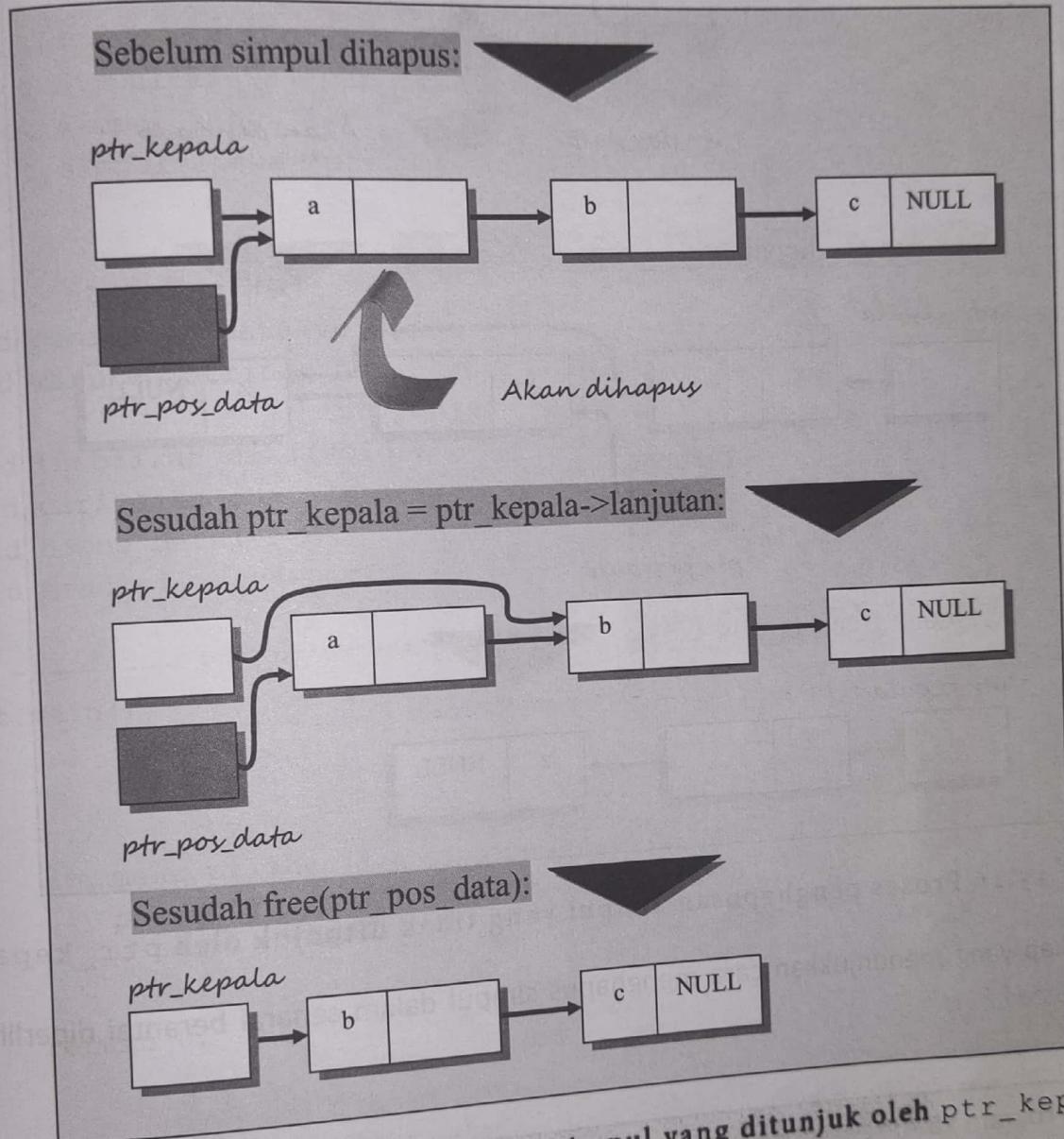
1. Simpul yang akan dihapus merupakan simpul yang tidak memiliki pendahulu.
2. Simpul yang dihapus memiliki simpul pendahulu.

pada kondisi pertama (lihat Gambar 15.15), penghapusan dilakukan dengan mula-mula mengatur `ptr_kepala` agar menunjuk simpul yang ditunjuk oleh pointer lanjutan. Pernyataan yang diperlukan:

```
ptr_kepala = ptr_kepala->lanjutan;
```

Selanjutnya, simpul yang ditunjuk oleh `ptr_pos_data` tinggal dibebaskan melalui pernyataan:

```
free(ptr_pos_data);
```



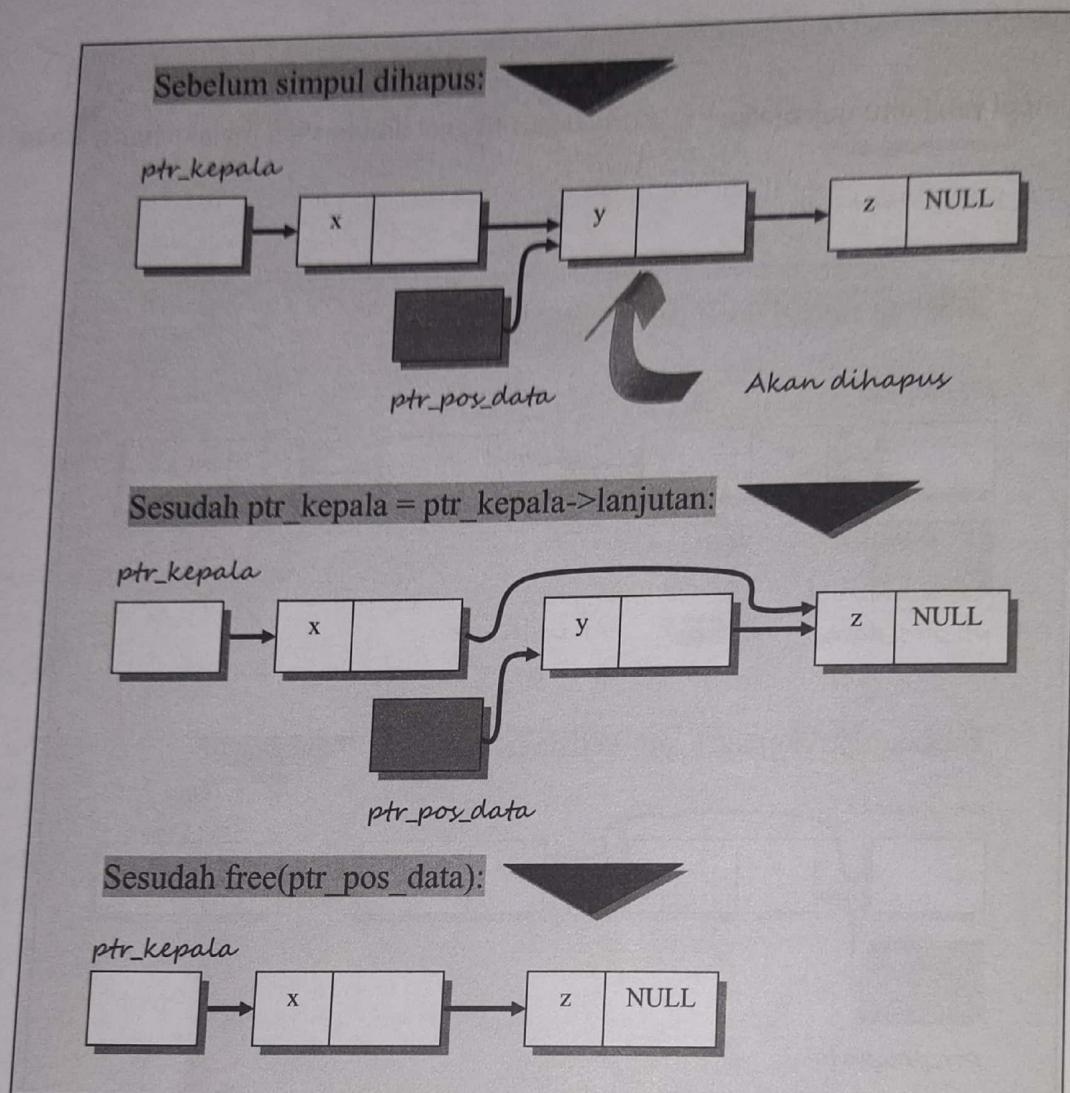
Gambar 15.15 Proses penghapusan simpul yang ditunjuk oleh `ptr_kepala`

Pada kondisi kedua (lihat Gambar 15.16), mula-mula pointer lanjutan pada simpul yang ditunjuk oleh `ptr_pendahulu` digeser agar menunjuk simpul yang ditunjuk oleh pointer lanjutan milik simpul yang ditunjuk oleh `ptr_pos_data`. Pernyataannya:

```
ptr_pendahulu->lanjutan =
ptr_pos_data->lanjutan;
```

Lalu, penghapusan simpul yang ditunjuk oleh `ptr_pos_data` dilaksanakan melalui:

```
free(ptr_pos_data);
```



Gambar 15.16 Proses penghapusan simpul yang tidak ditunjuk oleh `ptr_kepala`

Program lengkap yang menunjukkan cara menghapus simpul dalam senarai berantai diperlihatkan di program `senarai2.c`.



Program 15.4: `senarai2.c`

```
// -----
// Program senarai berantai
// - memasukkan data
// - menampilkan data
// - menghapus data
// -----
```

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

```

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

#define PANJANG_NOMOR 5
#define PANJANG_NAMA 20

struct simpul_siswa
{
    char nomor[PANJANG_NOMOR + 1];
    char nama[PANJANG_NAMA + 1];
    struct simpul_siswa *lanjutan;
};

// Deklarasi variabel global
struct simpul_siswa *ptr_kepala = NULL; // Ujung senarai
struct simpul_siswa *ptr_pos_data; // Posisi data
struct simpul_siswa *ptr_pendahulu; // Posisi pendahulu
                                         // ptr_pos_data

// Prototipe fungsi
void pemasukan_data(void);
void masukan_string(char *keterangan, char *masukan,
                     int panjang_maks);
void tampilan_data(void);
void cari_data(char *nama);
void hapus_data(void);
void menu_pilihan(void);

// ----- PROGRAM UTAMA
int main()
{
    for ( ; ; )
    {
        menu_pilihan();
        puts(""); // Baris kosong
    }

    return 0;
}

// ----- Akhir program utama
// -----
// Definisi fungsi untuk menangani pemasukan data
// -----
```

```

void pemasukan_data(void)
{
    char jawaban;
    struct simpul_siswa *ptr_baru;
```

```

    {
        ptr_baru = (struct simpul_siswa *)
            malloc(sizeof(struct simpul_siswa));
        if (ptr_baru)
        {
            masukan_string("Nomor siswa : ",
                ptr_baru->nomor,
                PANJANG_NOMOR);
            masukan_string("Nama siswa : ",
                ptr_baru->nama,
                PANJANG_NAMA);

            ptr_baru->lanjutan = ptr_kepala;
            ptr_kepala = ptr_baru;
            printf("Mau memasukkan data lagi (Y/T)? ");
            do
            {
                jawaban = toupper(getchar());
                fflush(stdin); // Hapus sisa data di keyboard
            } while (!(jawaban == 'Y' || jawaban == 'T'));

            }
        else
        {
            printf("Memori tak cukup!");
            break; // Keluar dari do-while
        }
    } while (jawaban == 'Y');
}

// -----
// Definisi fungsi untuk membaca string
// -----


void masukan_string(char *keterangan, char *masukan,
                     int panjang_maks)
{
    char st[256];

    do
    {
        printf(keterangan); // Tampilkan keterangan
        gets(st);           // Baca string
        if (strlen(st) > panjang_maks)
            printf("Terlalu panjang. Maksimal %d karakter\n",
                   panjang_maks);
    } while (strlen(st) > panjang_maks);
    // Salin string st ke mauskan
    strcpy(masukan, st);
}

```

```

// -----
// Definisi fungsi untuk menampilkan
// isi senarai berantai
// -----
void tampilkan_data(void)
{
    struct simpul_siswa *ptr_sementara;

    printf("\nIsi senarai berantai\n");
    ptr_sementara = ptr_kepala;
    while (ptr_sementara)
    {
        printf("%s %s\n", ptr_sementara->nomor,
               ptr_sementara->nama);
        ptr_sementara = ptr_sementara->lanjutan;
    }
}

// -----
// Definisi fungsi cari_data
// -----
// Digunakan untuk mencari nama pada senarai berantai
// Jika nama ditemukan, maka
// 1. ptr_pos_data menunjuk simpul yang berisi
//    nama yang dicari
// 2. ptr_pendahulu menunjuk simpul pendahulu
//    dari simpul posisi nama yang dicari.
//    Jika berisi NULL, menyatakan bahwa
//    ptr_pendahulu menunjuk simpul yang ditunjuk
//    oleh ptr_kepala
// Jika nama tak ditemukan,
//     ptr_pos_data berisi NULL
// -----

```

```

void cari_data(char *nama)
{
    ptr_pendahulu = NULL;
    ptr_pos_data = ptr_kepala;

    while (ptr_pos_data)
    {
        if (strcmp(nama, ptr_pos_data->nama) != 0)
        {
            ptr_pendahulu = ptr_pos_data;
            ptr_pos_data = ptr_pos_data->lanjutan;
        }
        else
            break; // Ditemukan - Selesai
    }
}

```

```

}

// -----
// Definisi fungsi hapus_data()
//
// Digunakan untuk menghapus simpul
// pada senarai berantai
// ----

void hapus_data(void)
{
    char nama[PANJANG_NAMA + 1];

    masukan_string("Masukkan nama yang akan dihapus: ",
                   nama, PANJANG_NAMA);

    cari_data(nama);
    if (ptr_pos_data == NULL)
        puts("Nama tak ditemukan di senarai berantai");
    else
    {
        // Proses penghapusan
        if (ptr_pendahulu == NULL)
            // Simpul ujung akan dihapus
            ptr_kepala = ptr_kepala->lanjutan;
        else
            // Bukan simpul ujung yang akan dihapus
            ptr_pendahulu->lanjutan =
                ptr_pos_data->lanjutan;

        // Bebaskan memori
        free(ptr_pos_data);

        puts("OK. Penghapusan sudah dilakukan");
    }
}

// -----
// Definisi fungsi untuk menangai menu pilihan
// -----

void menu_pilihan(void)
{
    char pilihan;

    puts("***** MENU PILIHAN *****");
    puts("*");
    puts("* [1] Memasukkan data");
    puts("* [2] Menghapus data");
    puts("* [3] Menampilkan data");
    puts("* [4] Selesai");
    puts("*");
    puts("*****");
}

```

```

printf("\nPilihan Anda (1..4): ");
do
{
    pilihan = getchar();
    fflush(stdin); // Hapus sisa data di keyboard
} while (pilihan < '1' || pilihan > '4');
switch(pilihan)
{
    case '1':
        pemasukan_data();
        break;
    case '2':
        hapus_data();
        break;
    case '3':
        tampilan_data();
        break;
    case '4':
        puts("Selesai");
        exit(0); // Selesai
}
}

```

Akhir Program

Contoh hasil eksekusi:

```

***** MENU PILIHAN *****
*
* [1] Memasukkan data
* [2] Menghapus data
* [3] Menampilkan data
* [4] Selesai
*
*****

```

```

Pilihan Anda (1..4): 1↙
Nomor siswa : 12336↙
Nama siswa : DIDIN WAHIDIN↙
Mau memasukkan data lagi (Y/T)? y↙
Nomor siswa : 12337↙
Nama siswa : ESTI PANDUWINATA↙
Mau memasukkan data lagi (Y/T)? y↙
Nomor siswa : 12338↙
Nama siswa : FAHMI JOHAN↙
Mau memasukkan data lagi (Y/T)? t↙

```

***** MENU PILIHAN *****
* [1] Memasukkan data *
* [2] Menghapus data *
* [3] Menampilkan data *
* [4] Selesai *

Pilihan Anda (1..4): 3^q

Isi senarai berantai
12338 FAHMI JOHAN
12337 ESTI PANDUWINATA
12336 DIDIN WAHIDIN

***** MENU PILIHAN *****
* [1] Memasukkan data *
* [2] Menghapus data *
* [3] Menampilkan data *
* [4] Selesai *

Pilihan Anda (1..4): 2^q

Masukkan nama yang akan dihapus: **ESTI**^q
Nama tak ditemukan di senarai berantai

***** MENU PILIHAN *****
* [1] Memasukkan data *
* [2] Menghapus data *
* [3] Menampilkan data *
* [4] Selesai *

Pilihan Anda (1..4): 2^q

Masukkan nama yang akan dihapus: **ESTI PANDUWINATA**^q
OK. Penghapusan sudah dilakukan

***** MENU PILIHAN *****
* [1] Memasukkan data *
* [2] Menghapus data *
* [3] Menampilkan data *
* [4] Selesai *

Pilihan Anda (1..4) : 3^v

Isi senarai berantai
12338 FAHMI JOHAN
12336 DIDIN WAHIDIN

***** MENU PILIHAN *****

- * [1] Memasukkan data *
 - * [2] Menghapus data *
 - * [3] Menampilkan data *
 - * [4] Selesai *
- *****

Pilihan Anda (1..4) : 4^v

Selesai