

## BAB XII

# ALOKASI MEMORI & LINK LIST

### Tujuan :

1. Menjelaskan cara mencari ukuran sebuah obyek
2. Menjelaskan cara mengalokasikan memori
3. Menjelaskan cara membebaskan memori
4. Menjelaskan cara merealokasikan memori
5. Menjelaskan aplikasi link list

Problem yang terjadi ketika bekerja dengan tipe data array adalah bahwa array cenderung menggunakan ukuran yang tetap, yang telah dipesan di awal deklarasi program. Ukuran ini tidak dapat diperbesar atau diperkecil ketika ternyata kebutuhannya menghendaki demikian.

Pada beberapa kasus, ukuran dari sebuah obyek tidak bisa dipastikan sampai dengan waktu dieksekusi (*run time*). Sebagai contoh, panjang dari string yang dimasukkan oleh user tidak bisa diketahui sebelum eksekusi, sehingga ukuran array bisa jadi bergantung pada parameter yang tidak bisa diketahui sebelum eksekusi program.

Alokasi memori (*memory allocation*) menyediakan fasilitas untuk membuat ukuran buffer dan array secara dinamik. Dinamik artinya bahwa ruang dalam memori akan dialokasikan ketika program dieksekusi. Fasilitas ini memungkinkan user untuk membuat tipe data dan struktur dengan ukuran dan panjang berapapun yang disesuaikan dengan kebutuhan di dalam program.

### 12.1 Mengetahui kebutuhan alokasi memori dengan fungsi `sizeof()`

Ukuran dari obyek yang akan dialokasikan lokasinya sangat bervariasi bergantung dengan tipe obyeknya. Ukuran ini berkenaan dengan jumlah byte memori yang akan disediakan, misalnya data bertipe `char` akan disimpan dalam 1 byte sedangkan `float` 4 byte. Pada sebagian besar mesin 32 bit `int` disimpan dalam 4 byte.

Cara paling mudah untuk menentukan ukuran obyek yang akan disimpan adalah dengan menggunakan fungsi `sizeof()`. Fungsi ini dapat digunakan untuk mendapatkan ukuran dari berbagai tipe data, variabel ataupun struktur. *Return value* dari fungsi ini adalah ukuran dari obyek yang bersangkutan dalam byte. Fungsi `sizeof()` dapat dipanggil dengan menggunakan sebuah obyek atau sebuah tipe data sebagai parameternya.

---

```
/* File program : size.c
Menggunakan fungsi sizeof() untuk menentukan ukuran obyek */

#include <stdio.h>

typedef struct employee_st {
    char name[40];
    int id;
} Employee;

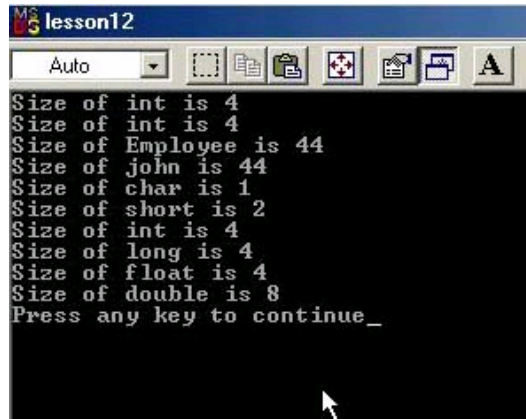
main()
{
    int myInt;
    Employee john;
```

```

printf("Size of int is %d\n",sizeof(myInt));
/* The argument of sizeof is an object */
printf("Size of int is %d\n",sizeof(int));
/* The argument of sizeof is a data type */
printf("Size of Employee is %d\n",sizeof(Employee));
/* The argument of sizeof is an object */
printf("Size of john is %d\n",sizeof(john));
/* The argument of sizeof is a data type */
printf("Size of char is %d\n",sizeof(char));
printf("Size of short is %d\n",sizeof(short));
printf("Size of int is %d\n",sizeof(int));
printf("Size of long is %d\n",sizeof(long));
printf("Size of float is %d\n",sizeof(float));
printf("Size of double is %d\n",sizeof(double));
return 0;
}

```

Bila dijalankan akan didapatkan keluaran sebagai berikut :



```

lesson12
Auto
Size of int is 4
Size of int is 4
Size of Employee is 44
Size of john is 44
Size of char is 1
Size of short is 2
Size of int is 4
Size of long is 4
Size of float is 4
Size of double is 8
Press any key to continue_

```

Gambar 12.1 Keluaran program size.c

## 12.2 Mengalokasikan memori dengan fungsi `malloc()`

Fungsi standar yang digunakan untuk mengalokasikan memori adalah `malloc()`. Bentuk prototypenya adalah

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

Banyaknya byte yang akan dipesan dinyatakan sebagai parameter fungsi. Return value dari fungsi ini adalah sebuah pointer yang tak bertipe (*pointer to void*) yang menunjuk ke buffer yang dialokasikan. Pointer tersebut haruslah dikonversi kepada tipe yang sesuai (dengan menggunakan *type cast*) agar bisa mengakses data yang disimpan dalam buffer. Jika proses alokasi gagal dilakukan, fungsi ini akan memberikan return value berupa sebuah pointer NULL. Sebelum dilakukan proses lebih lanjut, perlu terlebih dahulu dipastikan keberhasilan proses pemesanan memori, sebagaimana contoh berikut :

```

char *cpt;
...
if ((cpt = (char *) malloc(25)) == NULL)
{

```

```

        printf("Error on malloc\n");
        exit(0);
    }

/* File program : alokasi1.c
Menggunakan fungsi malloc() untuk mengalokasikan memori */

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

main()
{
    char s1[] = "This is a sentence";
    char *pblok;

    pblok = (char *) malloc(strlen(s1) + 1);
    /* Remember that strings are terminated by the null
       terminator, '\0', and the strlen returns the length
       of a string not including the terminator */
    if (pblok == NULL)
        printf("Error on malloc\n");
    else {
        strcpy(pblok, s1);
        printf("s1: %s\n", s1);
        printf("pblok: %s\n", pblok);
        return 0;
    }
}

```

### 12.3 Membebaskan kembali memori dengan fungsi `free()`

Jika bekerja dengan menggunakan memori yang dialokasikan secara dinamis, maka seorang programmer haruslah membebaskan kembali memori yang telah selesai digunakan untuk dikembalikan kepada sistem. Setelah suatu ruang memori dibebaskan, ruang tersebut bisa dipakai lagi untuk alokasi variabel dinamis lainnya. Untuk itu digunakan fungsi `free()` dengan prototype sebagai berikut :

```
void free(void *pblok);
```

dengan `pblok` adalah pointer yang menunjuk ke memori yang akan dibebaskan. Dengan demikian memori yang telah dibebaskan tersebut akan dapat digunakan oleh bagian lain dari program. Dalam hal ini, pointer `pblok` tidak perlu di-cast kembali ke `void` terlebih dahulu. Compiler otomatis telah menangani proses ini.

Sebuah contoh sederhana mengenai pengalokasian, pengaksesan terhadap variabel dinamis, dan pembebasan kembali ruang memori yang telah dipesan ditunjukkan pada contoh program `alokasi2.c` di bawah ini.

```

/* File program : alokasi2.c
Menggunakan fungsi free() untuk membebaskan kembali memori */

#include <stdio.h>

```

```
#include <stdlib.h>

main()
{
    char *pblok;

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

    if (pblok == NULL)
        puts("Error on malloc");
    else {
        puts("OK, alokasi memori sudah dilakukan");
        puts("-----");
        free(pblok);
        puts("Blok memori telah dibebaskan kembali");
    }
}
```

### **Contoh eksekusi :**

```
OK, alokasi memori sudah dilakukan
-----
Blok memori telah dibebaskan kembali
```

---

**Catatan :** Prototype fungsi `malloc()` dan `free()` terdapat pada file `stdlib.h`

Pada pernyataan

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

`(char *)` merupakan upaya untuk mengkonversikan pointer hasil `malloc()` agar menunjuk ke tipe `char` (sebab `pblok` dideklarasikan sebagai pointer yang menunjuk obyek bertipe `char` / *pointer to char*). Sedangkan

```
malloc(500 * sizeof(char))
```

digunakan untuk memesan ruang dalam memori sebanyak : 500 x ukuran `char` atau : 500 x 1 byte.

### **12.4 Mengalokasikan ulang memori dengan fungsi `realloc()`**

Bisa jadi terjadi ketika hendak mengalokasikan memori, user tidak yakin berapa besar lokasi yang dibutuhkannya. Misalnya user tersebut memesan 500 lokasi, ternyata setelah proses pemasukan data kebutuhannya melebihi 500 lokasi menjadi 600. Maka user tersebut dapat mengalokasikan ulang memori yang dipesannya dengan menggunakan fungsi `realloc()`. Fungsi ini akan mengalokasikan kembali pointer yang sebelumnya telah diatur untuk menunjuk sejumlah lokasi, memberinya ukuran yang baru (bisa jadi lebih kecil atau lebih besar). Sebagai contoh, adalah `pblok` adalah pointer yang menunjuk kepada 500 lokasi `char`, maka user bisa mengalokasikan ulang agar pointer `pblok` menunjuk kepada 600 lokasi `char` sebagai berikut :

```
...
pblok = (char *) malloc(500 * sizeof(char));
...
pblok = realloc(pblok, 600 * sizeof(char));
```

Fungsi `realloc()` akan merelokasi blok memori lama menjadi sebesar kapasitas baru yang dipesan. Jika `realloc()` berhasil melakukan relokasi baru, fungsi ini memberikan return value berupa pointer yang sama, meng-copy data lama ke lokasi baru dan mengarahkan pointer ke sejumlah lokasi baru tersebut (dalam hal ini setelah berhasil meng-copy data lama, fungsi ini juga otomatis membebaskan blok memori yang lama). Namun jika fungsi ini tidak berhasil menemukan

lokasi baru yang mampu menampung kapasitas baru yang dipesan oleh user, maka fungsi ini memberikan return value berupa NULL. Itulah sebabnya, cara terbaik adalah terlebih dahulu mengecek hasil `realloc()` untuk memastikan hasilnya bukan NULL pointer, sebelum kemudian menumpuki pointer lama dengan return value dari `realloc()`. Untuk itu bisa dibuat potongan program sebagai berikut :

```
char *newp;

newp = realloc(pblok, 600 * sizeof(char));
if(newp != NULL)
    pblok = newp;
else {
    printf("out of memori\n");
    /* exit or return */
    /* but pblok still points at 500 chars */
}
```

Pada potongan program tersebut, jika `realloc()` memberikan return value selain NULL pointer, berarti proses relokasi sukses, maka pointer `pblok` diset sama dengan `newp`. Namun, jika proses `realloc()` gagal (return valuenya NULL), maka pointer lama (yaitu `pblok`) masih menunjuk kepada 500 data asal.

## 12.5 Beberapa kesalahan umum

Ada beberapa kesalahan yang seringkali terjadi ketika bekerja dengan alokasi memori secara dinamis. Beberapa di antaranya adalah :

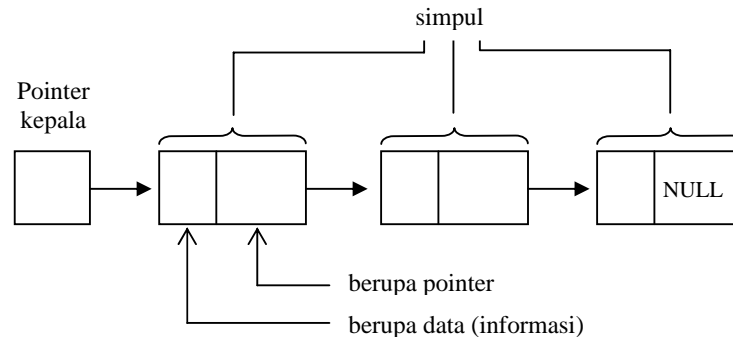
- ❑ Mengabaikan pengecekan hasil operasi `malloc()`  
Ketika gagal mengalokasikan memori, `malloc()` akan memberikan return value berupa NULL pointer. Jika hal ini terjadi, maka ketika pointer tersebut selanjutnya diakses dalam program akan mengakibatkan program mengalami ‘crash’. Sebaliknya, jika hasil operasi `malloc()` dicek terlebih dahulu, akan memungkinkan untuk bisa keluar dari program atau mengabaikan bagian-bagian dari program yang menggunakan buffer yang dialokasikan tadi.
- ❑ Mengabaikan pembebasan memori setelah digunakan  
Ada batasan besarnya kapasitas memori yang tersedia untuk dipakai oleh sebuah program. Jika proses menjalankan program yang panjang mengabaikan pembebasan memori, bahkan terus menerus mengalokasikan yang tersedia, maka program tersebut akan kehabisan memori (*run out*). Kondisi ini disebut dengan *memory leak*, yang merupakan sebuah *bug* yang sangat serius. Hal ini bisa berimplikasi pada unjuk kerja keseluruhan sistem.
- ❑ Pointer yang mengambang (*Dangling Pointer*)  
Setelah pembebasan sekumpulan lokasi memori, pointer yang menunjuk kepada lokasi yang telah dibebaskan tersebut masih berisi alamat memori dari titik awal lokasi tersebut. Secara alami, setelah proses pembebasan, maka pointer penunjuk lokasi tersebut tidak bisa digunakan lagi. Jika pointer ini kemudian digunakan lagi dalam program, maka apapun nilai yang ada pada lokasi tersebut akan bisa digunakan (yang bisa jadi merupakan nilai dari variabel lain). Cara terbaik untuk menghindari hal ini adalah dengan secara langsung mereset pointer menjadi NULL setelah lokasi memori yang ditunjuknya dibebaskan.  

```
free(pt);
pt = NULL;
```

Dengan demikian, pointer yang menunjuk kepada lokasi yang telah dibebaskan juga telah dinon-aktifkan.

## 12.6 Aplikasi pada Link List

Contoh penerapan alokasi dinamis yaitu untuk membuat struktur data yang disebut daftar berantai (*link list*). Gambaran sebuah link list ditunjukkan pada gambar 12.2 di bawah ini. Pada gambar tersebut terdapat 3 simpul (node). Namun dalam contoh nantinya akan terlihat bahwa jumlah simpul dalam daftar berantai dapat diperbanyak atau dikurangi.



Gambar 12.2 Bentuk sebuah daftar berantai (*link list*)

### 12.6.1 Membentuk Daftar Berantai

Proses pembentukan daftar berantai akan dijelaskan melalui contoh di bawah ini. Misalnya didefinisikan :

```
#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_siswa` berisi :

- ❑ Informasi berupa nomor dan nama, serta
- ❑ Pointer bernama `lanjutan` yang menunjuk ke obyek bertipe `simpul_siswa`

Dengan adanya pendeklarasian variabel pointer bersifat global/eksternal bernama `ptr_kepala` (yang menunjuk ke obyek bertipe `simpul_siswa`) melalui pernyataan:

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

maka `ptr_kepala` akan berisi `NULL` (artinya daftar berantai belum memiliki simpul)

`ptr _kepala`



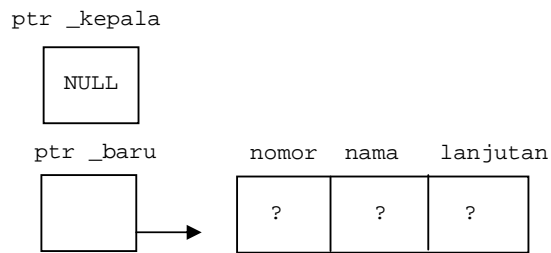
Gambar 12.3 `ptr_kepala` yang berisi `NULL`

Catatan : `NULL` adalah konstanta yang didefinisikan pada 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 berhasil dilakukan) maka terbentuk diagram sebagai berikut :



Gambar 12.4 Pembentukan simpul pertama

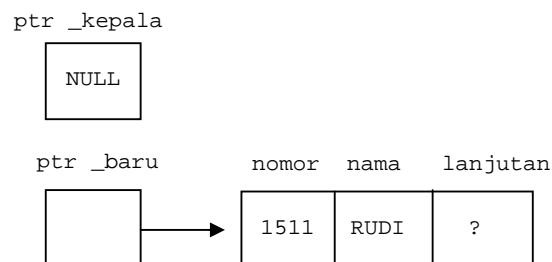
Pada gambar di atas tanda ? menyatakan isinya belum dispesifikasikan. Pernyataan

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

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

akan menyebabkan :

- ❑ field nomor berisi "11511"
- ❑ field nama berisi "RUDI"

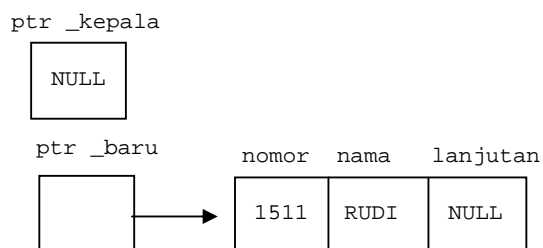


Gambar 12.5 Simpul pertama yang telah diisi field-fieldnya

Selanjutnya pernyataan keempat

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

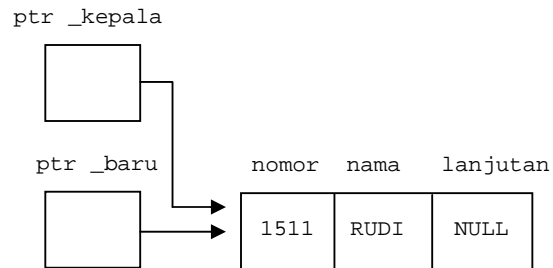
akan menyebabkan field lanjutan yang ditunjuk oleh ptr\_baru akan diisi dengan ptr\_kepala (yaitu NULL). Hasilnya adalah seperti pada Gambar 12.6



Gambar 12.6

Pernyataan kelima yaitu  
`ptr_kepala = ptr_baru;`

dipakai untuk mengisikan nilai pada `ptr_baru` ke `ptr_kepala`. Sebagai akibatnya, `ptr_kepala` akan menunjuk ke simpul yang ditunjuk `ptr_baru`. Hasilnya ditunjukkan pada Gambar 12.7



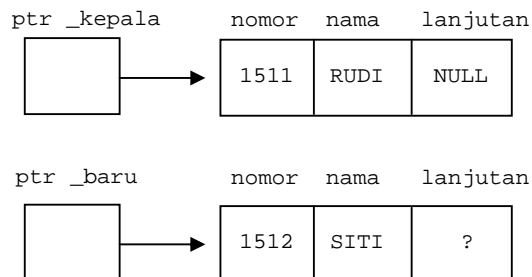
Gambar 12.7 Mengarahkan `ptr_kepala` agar menunjuk ke simpul baru

Untuk membentuk simpul kedua, pernyataan yang diperlukan berupa

```

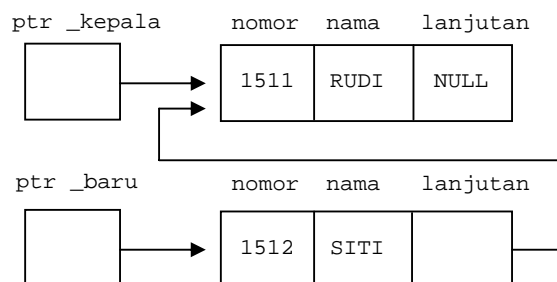
1) ptr_baru = (struct simpul_siswa *)
    malloc(sizeof(struct simpul_siswa));
2) ptr_baru->nomor = "11512";
3) ptr_baru->nama = "SITI";
4) ptr_baru->lanjutan = ptr_kepala;
5) ptr_kepala = ptr_baru;
  
```

Sesudah pernyataan 1 sampai dengan 3 akan terbentuk diagram sebagai berikut :



Gambar 12.8 Tahap pertama pembentukan simpul kedua

Adapun pernyataan keempat akan menyebabkan pointer lanjutan yang ditunjuk oleh `ptr_baru` akan menunjuk ke simpul yang ditunjuk oleh `ptr_kepala` (diperlihatkan pada Gambar 12.9)



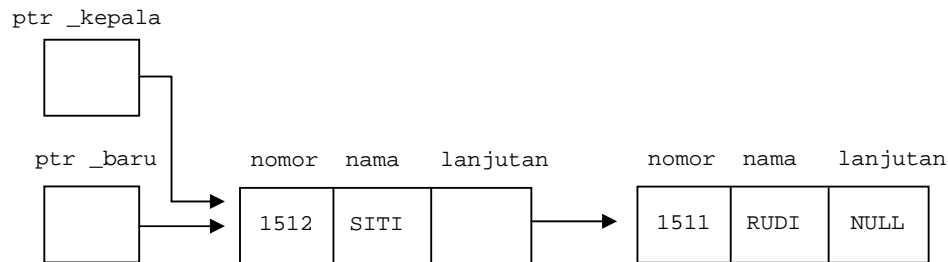
Gambar 12.9



Adapun pernyataan :

```
ptr_kepala = ptr_baru;
```

mengakibatkan ptr\_kepala akan menunjuk ke simpul yang ditunjuk ptr\_baru (sebab nilai ptr\_kepala sama dengan ptr\_baru). Hasil akhir ditunjukkan pada Gambar 12.10



Gambar 12.10 Hasil akhir

### 12.6.2 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.

```

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) //alokasi berhasil
        {
            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("Memasukkan data lagi (Y?T) ? ");
            do
                jawaban = toupper(getchar());
            while(!(jawaban == 'Y' || jawaban == 'T'));
            printf("%c\n", jawaban);
        }
        else {
            puts("Memori tak cukup!");
            break; //keluar dr do-while
        }
    } while(jawaban == 'Y');
}
  
```

Catatan: Fungsi masukan\_string() yang digunakan pada fungsi pemasukan\_data() merupakan fungsi untuk memasukkan data string. Definisi fungsi tersebut dapat dilihat pada program list1.c

Pada definisi fungsi `pemasukan_data()` terlihat bahwa setiap kali simpul akan diciptakan, memori dipesan terlebih dahulu. Jika alokasi berhasil maka proses penambahan simpul dalam daftar berantai dapat dilaksanakan. Jika pada saat pemesanan ternyata dalam memori tidak ada ruang lagi, maka pesan : “memori tak cukup!” akan ditampilkan di layar.

### 12.6.3 Menampilkan Isi Daftar Berantai

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

- ❑ isi simpul yang ditunjuk ditampilkan
- ❑ dan kemudian `ptr_smtr` diatur sesuai dengan nilai dari pointer lanjutan yang ditunjuknya.

Implementasinya adalah sebagai berikut :

```
void tampilkan_data(void)
{
    struct simpul_siswa *ptr_smtr;

    puts("\nIsi link list : \n");

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

Contoh program selengkapnya untuk membuat daftar berantai dan menampilkan isinya dapat dilihat pada program `list1.c` di bawah ini.

---

```
/* File program : list1.c
Membuat link list dan cara menampilkan isinya */

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.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 link list

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

```

main()
{
    pemasukan_data();
    tampilkan_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)
        {
            fflush(stdin);
            masukan_string("Nomor siswa : ", ptr_baru->nomor,
                PANJANG_NOMOR);
            fflush(stdin);
            masukan_string("Nama siswa : ", ptr_baru->nama, PANJANG_NAMA);

            ptr_baru->lanjutan = ptr_kepala;
            ptr_kepala = ptr_baru;
            printf("Memasukkan data lagi (Y/T) ? ");
            do
                jawaban = toupper(getchar());
            while(!(jawaban == 'Y' || jawaban == 'T'));
            printf("\n");
        }
        else {
            puts("Memori tak cukup!");
            break;          //keluar dr do-while
        }
    } while(jawaban == 'Y');
}

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

    do {
        printf(keterangan); //tampilkan keterangan
        gets(st);           //baca string dr keyboard
        if(strlen(st) > panjang_maks)
            printf("Terlalu panjang, maksimum %d karakter\n", panjang_maks);
    } while(strlen(st) > panjang_maks);
    strcpy(masukan, st);    //salin string st ke masukan
}

void tampilkan_data(void)
{
    struct simpul_siswa *ptr_smtr;

    puts("\nIsi link list : \n");
    ptr_smtr = ptr_kepala;
    while(ptr_smtr) {

```

```

        printf("%s %s\n", ptr_smtr->nomor, ptr_smtr->nama);
        ptr_smtr = ptr_smtr->lanjutan;
    }
}

```

### **Contoh eksekusi :**

```

Nomor siswa : 1234
Nama siswa  : Afif
Memasukkan data lagi (Y/T)? Y

```

```

Nomor siswa : 1235
Nama siswa  : Fariz
Memasukkan data lagi (Y/T)? Y

```

```

Nomor siswa : 1236
Nama siswa  : Wafiq
Memasukkan data lagi (Y/T)? T

```

Isi daftar berantai :

```

1234  Afif
1235  Fariz
1236  Wafiq

```

Pada daftar berantai yang telah dibahas, urutan data pada saat dimasukkan berlawanan dengan urutan saat ditampilkan (perhatikan eksekusi program `list1.c` di atas). Data terakhir yang dimasukkan akan menjadi data pertama saat ditampilkan. Struktur data semacam ini dikenal dengan sebutan LIFO (*Last in first out*) atau yang masuk terakhir akan keluar pertama.

### **12.6.4 Mencari Data dalam Simpul**

Pada pembahasan berikut akan diterangkan bagaimana caranya mencari suatu data dalam daftar berantai. Hal ini sangat diperlukan terutama untuk keperluan menghapus suatu simpul atau mengubah isi suatu simpul. Untuk keperluan ini, diperlukan dua tambahan variabel pointer eksternal, berupa

- `ptr_pos_data` : untuk menunjuk simpul yang berisi data yang dicari
- `ptr_pendahulu` : untuk menunjuk simpul yang merupakan pendahulu dari simpul yang dicari.

Deklarasinya :

```

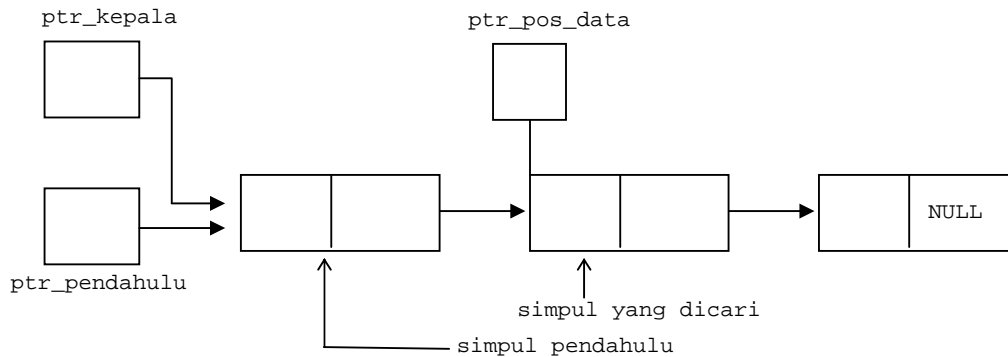
struct simpul_siswa *ptr_pos_data;
struct simpul_siswa *ptr_pendahulu;

```

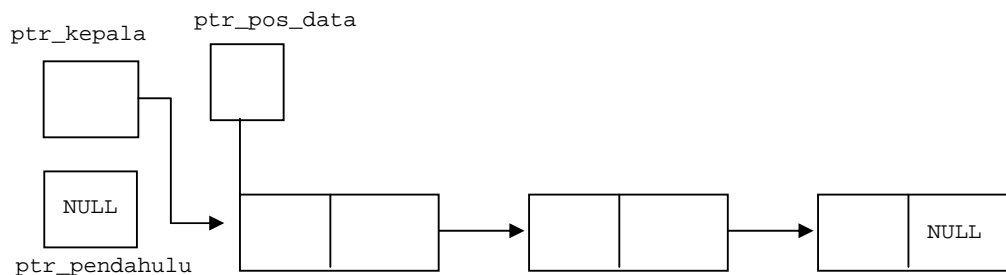
Aturan yang digunakan untuk mencari suatu data dalam daftar berantai adalah sebagai berikut :

1. Jika data yang dicari tidak ditemukan, `ptr_pos_data` berisi NULL
2. Jika data yang dicari ditemukan, maka
  - `ptr_pos_data` menunjuk ke simpul yang dicari
  - apabila simpul yang ditunjuk `ptr_pos_data` merupakan simpul yang ditunjuk oleh `ptr_kepala` (berarti simpul yang tak mempunyai simpul pendahulu), maka `ptr_pendahulu` berisi NULL.

- Sedangkan bila simpul yang ditunjuk oleh `ptr_pos_data` merupakan simpul yang tidak ditunjuk oleh `ptr_kepala`, maka `ptr_pendahulu` berisi alamat dari simpul pendahulu dari simpul yang ditunjuk oleh `ptr_pos_data`.



Gambar 12.11 Simpul yang dicari memiliki simpul pendahulu



Gambar 12.12 Simpul yang dicari tidak memiliki simpul pendahulu

Implementasinya adalah sebagai 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;      //ketemu, selesai
    }
}

```

### 12.6.5 Menghapus Suatu Simpul

Simpul yang tidak digunakan lagi perlu dihapus, dengan maksud agar ruang yang digunakan nantinya 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 akan dihapus memiliki simpul pendahulu

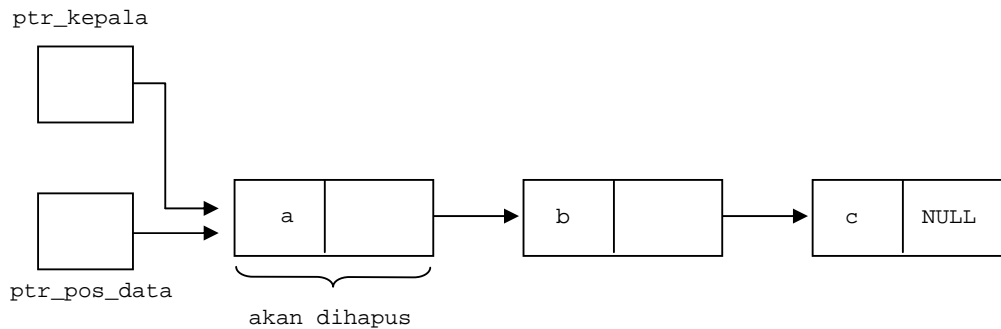
Pada kondisi pertama (Lihat Gambar 12.13), penghapusan dilakukan dengan mula-mula mengatur `ptr_kepala` agar menunjuk simpul yang ditunjuk oleh pointer lanjutan.

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

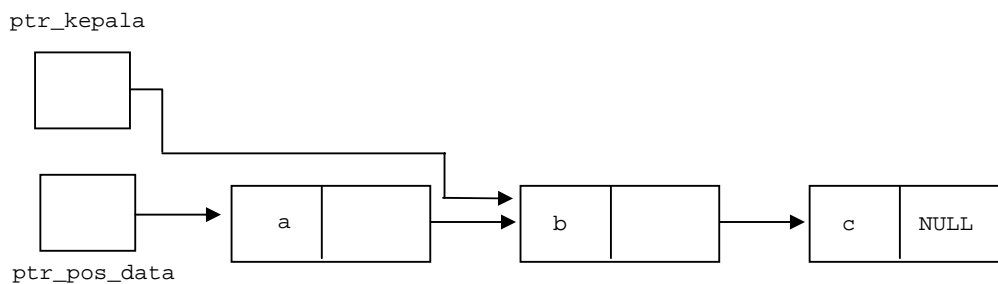
Selanjutnya, simpul yang ditunjuk oleh `ptr_pos_data` tinggal dibebaskan melalui pernyataan :

```
free(ptr_pos_data);
```

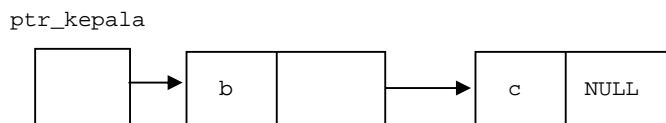
a) Sebelum dihapus (keadaan awal):



b) Sesudah pernyataan : `ptr_kepala = ptr_kepala->lanjutan;`



c) Sesudah pernyataan : `free(ptr_pos_data);`



Gambar 12.13 Proses penghapusan simpul yang ditunjuk oleh `ptr_kepala`

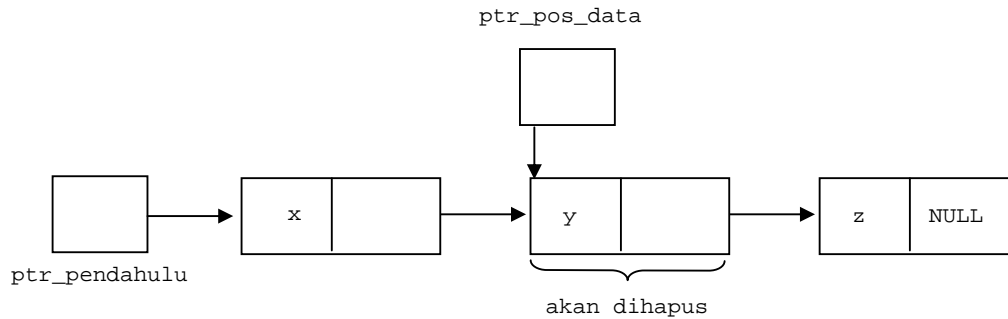
Pada kondisi kedua (lihat gambar 12.14), mula-mula pointer lanjutan dari simpul yang ditunjuk oleh `ptr_pendahulu` digeser agar menunjuk simpul yang ditunjuk oleh pointer lanjutan dari simpul yang ditunjuk oleh `ptr_pos_data`.

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

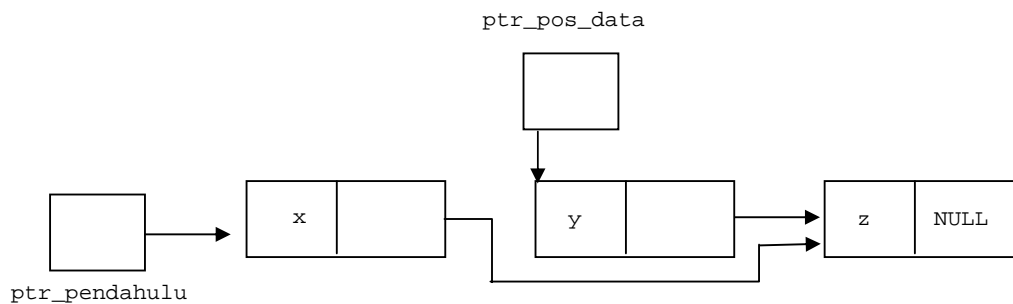
Lalu diikuti dengan penghapusan simpul yang ditunjuk oleh `ptr_pos_data` melalui pernyataan :

```
free(ptr_pos_data);
```

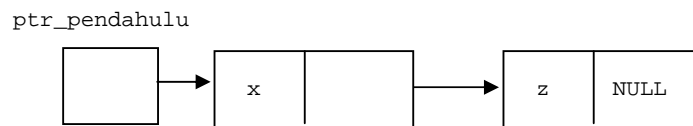
a) Sebelum dihapus (keadaan awal):



b) Sesudah pernyataan : `ptr_pendahulu->lanjutan = ptr_pos_data->lanjutan;`



c) Sesudah pernyataan : `free(ptr_pos_data);`



Gambar 12.14 Proses penghapusan simpul yang tidak ditunjuk oleh `ptr_kepala`

Rangkaian lengkap tentang cara menghapus suatu simpul bisa dilihat pada fungsi `hapus_data()` pada program berikut :

---

```

/* File program : list2.c
Membuat link list untuk memasukkan, menampilkan dan menghapus data */

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

#define PANJANG_NOMOR 5
#define PANJANG_NAMA 20
#define ENTER 13

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 link list
struct simpul_siswa *ptr_pos_data;      //posisi data
struct simpul_siswa *ptr_pendahulu; //posisi pendahulu dr ptr_pos_data

//deklarasi fungsi
void pemasukan_data(void);
void masukan_string(char *keterangan, char *masukan,
                    unsigned panjang_maks);
void tampilkan_data(void);
void cari_data(char *nama);
void hapus_data(void);
void menu_pilihan(void);

//PROGRAM UTAMA
main()
{
    do {
        menu_pilihan();
        puts("Tekanlah ENTER untuk melanjutkan");
        while( getchar() == ENTER);
    } while( getchar() != ENTER);           //baca ENTER
}

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("Memasukkan data lagi (Y/T) ? ");
            do
                jawaban = toupper(getchar());
            while(!(jawaban == 'Y' || jawaban == 'T'));
            printf("\n", jawaban);
        }
        else {
            puts("Memori tak cukup");
            break; //keluar dr do-while
        }
    } while(jawaban == 'Y');
}

```



```

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

    do {
        printf(keterangan); //tampilkan keterangan
        fflush(stdin);
        gets(st); //baca string dr keyboard
        if(strlen(st) > panjang_maks)
            printf("Terlalu panjang, maksimum %d karakter\n",
                panjang_maks);
    } while(strlen(st) > panjang_maks);

    strcpy(masukan, st); //salin string st ke masukan
}

void tampilkan_data(void)
{
    struct simpul_siswa *ptr_smtr;

    puts("\nIsi link list : \n");

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

void cari_data(char *nama)
//untuk mencari data pd daftar berantai
//Jika data ditemukan, maka
//1. ptr_pos_data menunjuk simpul yang berisi data yang dicari
//2. ptr_pendahulu menunjuk simpul pendahulu dari simpul posisi data yang
//   dicari. Jika berisi NULL, menyatakan bahwa ptr_pendahulu menunjuk
//   simpul yang ditunjuk oleh ptr_kepala
//Jika data tidak ditemukan, ptr_pos_data berisi NULL
{
    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; //ketemu, selesai
    }
}

```

```

void hapus_data(void)
// untuk menghapus simpul yg ada pd daftar berantai
{
    char nama[21];

    masukan_string("Masukkan nama yang akan dihapus : ", nama, 20);
    cari_data(nama);
    if(ptr_pos_data == NULL)
        puts("Nama tdk ditemukan pada link list.");
    else {
        //proses penghapusan
        if(ptr_pendahulu == NULL)
            //simpul ujung yg akan dihapus
            ptr_kepala = ptr_kepala->lanjutan;
        else
            ptr_pendahulu->lanjutan = ptr_pos_data->lanjutan;

        free(ptr_pos_data);
        puts("OK, penghapusan sudah dilakukan");
    }
}

void menu_pilihan(void)
{
    char pilihan;

    puts("MENU PILIHAN :");
    puts("1. Memasukkan data ");
    puts("2. Menghapus data ");
    puts("3. Menampilkan data ");
    puts("4. Selesai ");
    printf("\nPilihan Anda (1..4) : ");

    do
        pilihan = getchar();
    while (pilihan < '1' || pilihan > '4');
    printf("\n", pilihan);

    switch(pilihan)
    {
        case '1' : pemasukan_data();
            break;
        case '2' : hapus_data();
            break;
        case '3' : tampilkan_data();
            break;
        case '4' : exit(0); //selesai
    }
}

```

### **Contoh eksekusi :**

```

MENU PILIHAN :
1. Memasukkan data
2. Menghapus data
3. Menampilkan data
4. Selesai

```

Pilihan Anda (1..4) : 1

Nomor siswa : 1237

Nama siswa : Wildan

Memasukkan data lagi (Y/T)? Y

Nomor siswa : 1238

Nama siswa : Hanifah

Memasukkan data lagi (Y/T)? Y

Nomor siswa : 1239

Nama siswa : Ainur Rosyidah

Memasukkan data lagi (Y/T)? T

Tekanlah ENTER untuk melanjutkan

MENU PILIHAN :

1. Memasukkan data
2. Menghapus data
3. Menampilkan data
4. Selesai

Pilihan Anda (1..4) : 3

Isi daftar berantai :

1237 Wildan

1238 Hanifah

1239 Ainur Rosyidah

Tekanlah ENTER untuk melanjutkan

MENU PILIHAN :

1. Memasukkan data
2. Menghapus data
3. Menampilkan data
4. Selesai

Pilihan Anda (1..4) : 2

Masukkan nama yang akan dihapus : Wildan

OK, penghapusan sudah dilakukan

Tekanlah ENTER untuk melanjutkan

MENU PILIHAN :

1. Memasukkan data
2. Menghapus data
3. Menampilkan data
4. Selesai

Pilihan Anda (1..4) : 3

Isi daftar berantai :

1238 Hanifah

1239 Ainur Rosyidah

Tekanlah ENTER untuk melanjutkan

---

**Kesimpulan :**

- Struktur adalah pengelompokan variabel-variabel yang bernaung dalam satu nama yang sama.
- Variabel-variabel yang membentuk suatu struktur, selanjutnya disebut sebagai elemen dari struktur atau *field*.
- 

**Latihan :**

**Buatlah potongan program untuk soal-soal di bawah ini**

1. Definisikan sebuah struktur (misalkan namanya = **record**) yang memiliki 3 buah *field* berupa sebuah integer (misalkan namanya = **loop**), sebuah array karakter dengan 5 elemen (misalkan namanya = **word**) dan sebuah *float* (misalkan namanya = **sum**).
2. Deklarasikan sebuah variabel struktur (misalkan namanya = **sample**) yang didefinisikan memiliki tipe struktur **record**.