Praktikum 4

Alokasi Memori

A. TUJUAN PEMBELAJARAN

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Memahami perbedaan penggunaan tipe data array dengan pointer menggunakan alokasi memori
2. Memahami setiap menggunakan tipe data pointer maka hendaknya melakukan alokasi memori

B. DASAR TEORI

B.1 Mengapa Menggunakan Alokasi Memory

Ketika kita mempelajari tipe data array, nampak kelemahan tipe data ini adalah sifatnya yang statis. Artinya ketika kita mendeklarasikan sebuah variabel dengan tipe data array maka data yang kita deklarasikan disimpan pada memori harus dalam kondisi terurut. Selain itu selama program berjalan ukuran dari array bersifat tetap atau kita tidak dapat merubahnya.

Adakalanya dalam pemrograman ukuran sebuah obyek belum dapat kita tentukan sampai program kita jalankan. Alokasi memori menyediakan fasilitas untuk membuat ukuran buffer dan array secara dinamik. Dinamik artinya bahwa ruang dalam memori akan dialokasikan ketika program dieksekusi (*run time*). Fasilitas ini memungkinkan user untuk membuat tipe data dan struktur dengan ukuran dan panjang berapapun yang disesuaikan dengan kebutuhan di dalam program.

1. Perintah *sizeof()*

Sebelum kita menggunakan alokasi memori, kita harus mengenal perintah sizeof. Perintah ini digunakan untuk:

- Untuk mendapatkan ukuran dari berbagai tipe data, variabel ataupun struktur.
- Return value : ukuran dari obyek yang bersangkutan dalam byte.
- Parameter dari sizeof() : sebuah obyek atau sebuah tipe data

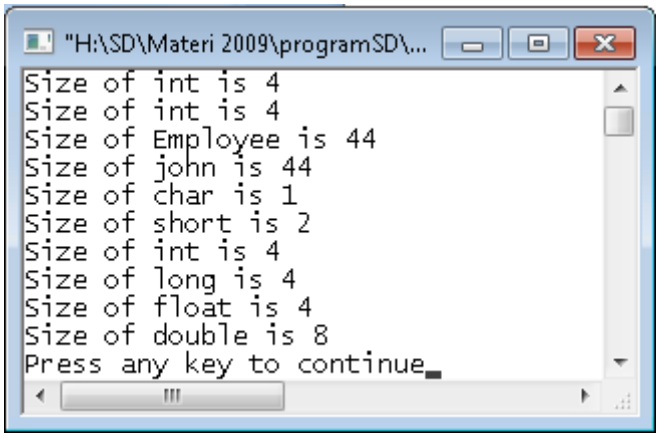
Pada Gambar 4.1 ditunjukkan contoh penggunaan perintah sizeof() dan hasil outputnya pada Gambar 4.2.

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

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

main()
{
    int myInt;
    Employee john;
    printf("Size of int is %d\n",sizeof(myInt));
    printf("Size of int is %d\n",sizeof(int));
    printf("Size of Employee is %d\n",sizeof(Employee));
    printf("Size of john is %d\n",sizeof(john));
    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;
}
```

Gambar 4.1 Listing Program Contoh Penggunaan sizeof



Gambar 4.2 Output dari Listing Program pada Gambar 4.1

B.2 Perintah *malloc()*

Fungsi standar dalam C yang digunakan untuk mengalokasikan memori adalah `malloc()`. Prototype dari fungsi ini adalah sebagai berikut:

```
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, seperti ditunjukkan pada Gambar 4.3.

```
int *x;
x = (int *) malloc(3 * sizeof(int));
if (x== NULL) {
    printf("Error on malloc\n");
    exit(0);
} else {
    lakukan operasi memori dinamis…
}
```

Gambar 4.3 Listing Program Pemesanan Memori

Dengan perintah seperti di bawah ini terjadi pengalokasian memori seperti yang digambarkan pada Gambar 4.4.

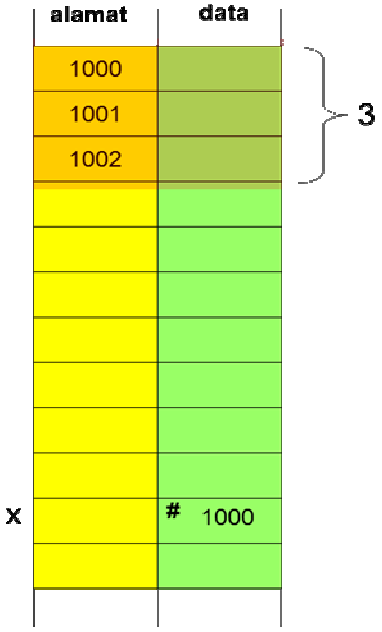
```
int *x;
x = (int *) malloc(3 * sizeof(int));
```

B.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. Pada Gambar 4.5 ditunjukkan listing program untuk membebaskan memori tersebut.



Gambar 4.4 Alokasi 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");
      }
 }
```

Gambar 4.5 Listing Program Pembebasan Memori

B.3 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 yang ditunjukkan pada Gambar 4.6.

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

Gambar 4.6 Listing Program Alokasi Ulang Memori

C. TUGAS PENDAHULUAN

Untuk semua persoalan di bawah ini, desainlah algoritma atau flowchartnya

1. Menampilkan bilangan fibonanci pertama sampai ke-n
2. Menampilkan bilangan prima pertama sampai ke-n

D. PERCOBAAN

1. Tampilkan bilangan Fibonaci pertama sampai ke-n menggunakan Pointer dengan Malloc, dimana n dimasukkan oleh user. Gunakan pengecekan pengalokasian dan fungsi free di akhir program.
2. Tampilkan bilangan Prima pertama sampai ke-n menggunakan Pointer dengan Malloc, dimana n dimasukkan oleh user. Gunakan fungsi realloc() untuk mengalokasikan ukuran memori yang baru (m), dimana m dimasukkan oleh user.

E. LATIHAN

1. Dengan pointer menggunakan malloc tampilkan deret angka dengan rumus di bawah ini, dimana inputnya berupa bilangan untuk batas (n).

   $C_n = 2\ C_{n-1} + 1$ jika $C_0 = 1$

   $S_n = S_{n-1} + n - 1$ jika $S_1 = 0$

F. LAPORAN RESMI

1. Untuk setiap listing program dari percobaan-percobaan di atas, ambil *capture* outputnya.
2. Tuliskan kesimpulan dari percobaan yang telah anda lakukan.