MODUL 11: Hashing

11.1 Deskripsi Singkat

Hashing adalah suatu metode untuk menyimpan data dalam sebuah array agar penyimpanan, pencarian, penambahan, dan penghapusan data dapat dilakukan dengan cepat dengan cara mengakses lokasi/index penyimpanan data secara langsung.

Dalam hashing, untuk penambahan atau pencarian, data atau key tersebut akan dipetakan ke dalam suatu index/lokasi dari suatu data menggunakan hash function, dan terdapat hash table yang merupakan array tempat penyimpanan index/lokasi data yang berdasarkan output hash function.

11.2 Tujuan Praktikum

Mahasiswa mampu mengimplementasikan Hashing dengan menggunakan bahasa C.

11.3 Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

11.4 Kegiatan Praktikum

Hashing

Pada modul ini akan dibuat sebuah program yang memuat sebuah hash table dengan ukuran 10 sehingga fungsi hashing-nya adalah

key mod 10

dan data yang dapat disimpan hanya 10 item data.

Proses pembuatan program ini terdiri dari 4 tahap, yaitu:

1. Persiapan,
2. Membuat fungsi insert,
3. Membuat fungsi remove_element,
4. Finalisasi.

1. Persiapan

Pada tahap ini akan dibuat hash table (dalam program akan diberi nama array) dengan bentuk struktrur seperti di bawah ini:

Array[]

Indeks	flag	*data	
		key	value
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

Pertama-tama akan dibuat struktur item dengan 2 field, key dan value.

```
struct item
{
    int key;
    int value;
};
```

Sehingga terbentuk struktur

Key	value

Selanjutnya dibentuk struktur hash table untuk satu record dengan menambahkan flag.

```
struct hashtable_item
{
    int flag;
    /* flag = 0 : Tidak ada data
```

```
     * flag = 1 : Ada data
     * flag = 2 : Sebelumnya ada datanya */

    struct item *data;
};
```

Sehingga struktur datanya menjadi seperti di bawah ini.

flag	*data	
	key	value

Selanjutnya dibuat hash table dengan nama array (juga bertipe array) bertipe struktur hashtable-item. Setelah terbentuk, dilakukan proses ini pemberian nilai awal, flag = 0 dan data = null.

```
struct hashtable_item *array;
int max = 10;

/* initializing hash table array */
void init_array()
{
    int i;
    for (i = 0; i < max; i++)
    {
        array[i].flag = 0;
        array[i].data = NULL;
    }
}
```

Sampai di sini, sudah terbentuk struktur data dari hash table yang kita inginkan.

Selanjutnya dibuat fungsi hashing dan fungsi menghitung ukuran dari hash table.

```
/* to every key, it will generate a corresponding index */
int hashcode(int key)
{
    return (key % max);
}
int size = 0; /* size dari hashtable */

int size_of_hashtable()
{
    return size;
}
```

2. Membuat fungsi insert

   Ada beberapa hal yang harus diperhatikan saat menambahkan sebuah item. Pertama apakah key dari item yang baru ini sudah ada atau tidak. Jika sudah ada, apa yang harus dilakukan? Apakah tidak dilakukan apa2 atau di-update value-nya? Di sini kita memilih untuk meng-update value-nya. Kemudian yang kedua adalah apakan hash table-nya sudah penuh atau tidak. Jika sudah penuh maka akan ditampilkan pesan bahwa hash table sudah penuh. Jika tidak maka akan dicari posisi indeks yang sesuai. Jika index hasil penghitungan fungsi hashing belum terisi (flag tidak sama dengan 1) maka item baru akan diletakkan pada index tersebut. Tetapi jika tidak maka akan dilakulan collision dengan linear probing.

```
void insert(int key, int value)
{
    int index = hashcode(key);
    int i = index;

    /* creating new item to insert in the hash table array */
    struct item *new_item = (struct item*)
malloc(sizeof(struct item));
    new_item->key = key;
    new_item->value = value;
```

```
    /* probing through the array until we reach an empty space
- LINEAR PROBING*/
    while (array[i].flag == 1)
    {
        if (array[i].data->key == key)
        {
            /* case where already existing key matches the
given key */
            printf("\n Key already exists, hence updating its
value \n");
            array[i].data->value = value;
            return;
        }

        i = (i + 1) % max; //maju satu langkah
        if (i == index) //jika sudah mengecek satu - satu
index sampai balik lagi ke index semula, artinya tabel penuh
        {
            printf("\n Hash table is full, cannot insert any
more item \n");
            return;
        }
    }

    array[i].flag = 1;
    array[i].data = new_item;
    size++;
    printf("\n Key (%d) has been inserted \n", key);

}
```

3. Membuat fungsi delete (remove_element)

   Untuk menghapus sebuah elemen yang diinginkan, maka harus dipastikan dulu keberadaan elemen tersebut. Jika sudah didapatkan keberadaannya maka nilai flag pada elemen tersebut dinolkan dan datanya di-null-kan. Jika elemennya tidak ditemukan maka akan ditampilkan pesan bahwa tidak ada elemen dengan key sesuai yang diinput pengguna.

```
void remove_element(int key)
{
    int index = hashcode(key);
    int  i = index;

    /* probing through array until we reach an empty space
where not even once an element had been present */
    while (array[i].flag != 0)
    {
        if (array[i].flag == 1  &&  array[i].data->key == key
)
        {
            // case when data key matches the given key
            array[i].flag =  2;
            array[i].data = NULL;
            size--;
            printf("\n Key (%d) has been removed \n", key);
            return;
        }
        i = (i + 1) % max;
        if (i == index)
        {
            break;
        }
    }
    printf("\n This key does not exist \n");

}
```

4. Finalisasi

   Pada tahapan ini kita akan membuat fungsi display yang akan menampilkan seluruh isi dari hash table. Kemudian dalam fungsi utama akan dibuatkan menu agar pengguna dapat menambahkan, menghapus, mengetahui jumlah data dan isi dari hash table yang dibuat secara berulang.

```
/* to display all the elements of hash table */
void display()
{
    int i;
    for (i = 0; i < max; i++)
    {
        struct item *current = (struct item*) array[i].data;
        if (current == NULL)
        {
            printf("\n Array[%d] has no elements \n", i);
        }
        else
        {
            printf("\n Array[%d] has elements -: \n %d(key)
and %d(value) ", i, current->key, current->value);
        }
    }
}

int main()
{
     int choice, key, value, n, c;

     array = (struct hashtable_item*) malloc(max *
sizeof(struct hashtable_item));
     init_array();

     do {
```

```
       printf("Implementation of Hash Table in C with Linear
Probing \n\n");
       printf("MENU-: \n1.Inserting item in the Hashtable"
                      "\n2.Removing item from the Hashtable"
                      "\n3.Check the size of Hashtable"
                      "\n4.Display Hashtable"
              "\n\n Please enter your choice-:");

       scanf("%d", &choice);

       switch(choice)
        {
            case 1:
                  printf("Inserting element in Hashtable\n");
                  printf("Enter key-:\t");
                  scanf("%d", &key);
                  printf("Enter value-:\t");
                  scanf("%d", &value);
                  insert(key, value);

                  break;

            case 2:
                  printf("Deleting in Hashtable \n Enter the
key to delete-:");
                  scanf("%d", &key);
                  remove_element(key);

                  break;

            case 3:
                  n = size_of_hashtable();
                  printf("Size of Hashtable is-:%d\n", n);
```

```
                break;
            case 4:
                display();
                break;
            default:
                 printf("Wrong Input\n");

        }
        printf("\n Do you want to continue-:(press 1 for
yes)\t");
        scanf("%d", &c);
    } while(c == 1);

    getchar();
    return 0;
}
```

Simpan potongan program di atas dengan nama **Praktikum11A.c,** kemudian coba jalankan dengan menggunakan key-value berikut:

45 – 100,
72 – 200,
39 – 300,
48 – 400,
56 – 500,
77 – 600,
91 – 700,
63 – 800,
84 – 900,
90 – 1000

Bagaimana outputnya? Jika sudah berjalan dengan baik, cobalah dengan menggunakan data yang lain

11.5 Penugasan

Modifikasi program **Praktikum11A.c** dengan ketentuan sebagai berikut:

1. Jumlah data yang disimpan pada tabel hash bisa fleksibel**.**
2. Data pada baris/indeks yang dihapus (flag=2) tidak dapat diisi lagi.
3. Collision resolutionnya menggunakan metode quadratic probing.

Simpan hasil modifikasi Anda pada file **Praktikum11B_kelas_nim.c.** Unggah file pada Google Classroom sesuai dengan batas waktu yang telah ditetapkan (jangan dizip/rar). Keterlambatan pengumpulan dikenakan sanksi pemotongan nilai sebesar 10 poin per jam.