**Jobsheet 1**

**Praktikum Struktur Data**

****

**Dosen pengampu : Randi Proska Sandra, S.Pd, M.Sc**

**Kode Kelas : 202323430158**

**Disusun Oleh :**

**Ketherin Fathur Rahma**

**23343072**

**PROGRAM STUDI INFORMATIKA (NK)**

**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2024**

1. **Pointer :**

* Pointer adalah variabel yang menyimpan alamat memori suatu nilai atau objek. Sebagai contoh, ketika kita mendeklarasikan variabel bertipe int seperti berikut:

int angka = 10;

Maka, kita dapat membuat pointer untuk menyimpan alamat memori variabel tersebut:

int \*ptrAngka = &angka;

Dalam contoh di atas, ptrAngka adalah pointer yang menyimpan alamat memori dari variabel angka.

* Dereferensi pointer adalah tindakan mengakses nilai yang disimpan di alamat memori yang ditunjukkan oleh pointer. Dengan menggunakan operator dereference (\*), kita dapat mengambil nilai dari variabel yang diakses melalui pointer. Contoh:

printf("Nilai dari angka: %d", \*ptrAngka);

Dalam kasus ini, \*ptrAngka akan memberikan nilai 10, yaitu nilai yang disimpan dalam variabel angka.

* Manfaat Penggunaan Pointer :
* **Efisiensi Memori :** Dengan menggunakan pointer, program dapat mengakses dan memanipulasi data secara langsung pada lokasi memori tertentu, menghemat ruang memori dan meningkatkan efisiensi.
* **Pengembangan Struktur Data :** Pointer memungkinkan pembuatan struktur data yang kompleks, seperti linked list, stack, dan tree, yang sering digunakan dalam pengembangan perangkat lunak.
* **Pengoptimalan Kinerja:** Dengan menggunakan pointer, kita dapat menghindari penggunaan variabel sementara atau pengoperasian langsung pada data, yang dapat meningkatkan kinerja program.

1. **Struct :**

* Struct adalah konstruksi dalam pemrograman yang memungkinkan pengguna untuk menggabungkan beberapa tipe data yang berbeda menjadi satu kesatuan logis. Dengan kata lain, struct memungkinkan kita untuk membuat tipe data khusus yang dapat mencakup berbagai macam informasi terkait. Contoh sederhana struktur data dalam bahasa C adalah sebagai berikut:

struct Mahasiswa {

char nama[50];

int umur;

float IPK;

};

Dalam contoh di atas, Mahasiswa adalah tipe data baru yang memiliki tiga anggota: nama (tipe char array), umur (tipe int), dan IPK (tipe float).

* Setelah mendeklarasikan struct, kita dapat membuat variabel menggunakan tipe data yang telah dibuat tersebut:

struct Mahasiswa mhs1;

Kemudian, kita dapat mengakses dan mengisi nilai anggota struct seperti berikut:

strcpy(mhs1.nama, "John Doe");

mhs1.umur = 20;

mhs1.IPK = 3.75;

* Manfaat Penggunaan Struct :
* **Organisasi Data :** Struct membantu pengorganisasian data dengan mengelompokkan informasi terkait ke dalam satu entitas.
* **Penghematan Memori :** Dengan mengelompokkan data terkait dalam satu struktur, program dapat menghemat ruang memori dan mengoptimalkan penggunaannya.
* **Pembuatan Tipe Data Baru :** Struct memungkinkan pembuatan tipe data baru yang sesuai dengan kebutuhan aplikasi, meningkatkan ekspresivitas dan struktur program.

1. **Array :**

* Array adalah kumpulan elemen data dengan tipe yang sama, yang disimpan dalam urutan terindeks. Setiap elemen dalam array dapat diakses menggunakan indeksnya. Misalnya, dalam bahasa C, kita dapat mendeklarasikan array bilangan bulat sebagai berikut:

int angka[5] = {1, 2, 3, 4, 5};

Dalam contoh ini, angka adalah array yang terdiri dari lima elemen bertipe int.

* Array dapat dideklarasikan dengan menentukan tipe elemennya dan jumlah elemen yang akan disimpan. Proses akses elemen menggunakan indeks, yang dimulai dari 0. Contoh:

int nilai = angka[2]; // Mengakses elemen ketiga array, nilai sekarang 3

* Manfaat Penggunaan Array :
* **Manajemen Data Terstruktur:** Array membantu dalam menyimpan dan mengorganisir data secara terstruktur, membuatnya lebih mudah untuk dikelola.
* **Pemrograman Efisien:** Penggunaan array memungkinkan kita untuk mengakses dan memanipulasi sejumlah besar data dengan cara yang efisien.
* **Iterasi dan Pengulangan:** Array mempermudah proses pengulangan atau iterasi melalui elemen-elemen data dengan menggunakan loop, mengurangi kerja manual dalam pengolahan data.
* Jenis-jenis Array :
* **Satu Dimensi:** Array dengan satu dimensi, seperti pada contoh di atas, digunakan untuk menyimpan data dalam satu baris atau satu kolom.
* **Dua Dimensi:** Array dua dimensi digunakan untuk menyimpan data dalam bentuk matriks, dengan baris dan kolom.
* **Multidimensi:** Array multidimensi digunakan untuk struktur data yang lebih kompleks, seperti matriks tiga dimensi atau lebih.

1. **Linked List :**

* Linked list adalah struktur data linier yang terdiri dari sejumlah simpul (node) yang saling terhubung. Setiap simpul menyimpan data dan alamat memori dari simpul berikutnya dalam urutan tertentu. Ini berbeda dengan array yang menyimpan elemen secara berurutan dalam lokasi memori yang terus-menerus.
* Dalam bahasa pemrograman seperti C atau C++, linked list dapat didefinisikan dengan membuat sebuah struktur yang memiliki dua bagian: data dan pointer ke simpul berikutnya (dan sebelumnya untuk doubly linked list). Contoh struktur untuk singly linked list adalah sebagai berikut:

struct Node {

int data;

struct Node\* next;

};

Dalam penggunaannya, linked list dapat dimanipulasi dengan menambah, menghapus, atau mencari simpul sesuai kebutuhan aplikasi.

* Jenis Utama Linked List :
* **Singly Linked List :** Setiap simpul hanya terhubung ke simpul berikutnya. Simpul terakhir menunjuk ke NULL, menandakan akhir dari linked list.
* **Doubly Linked List:** Setiap simpul terhubung ke simpul sebelumnya dan berikutnya. Ini memungkinkan navigasi maju dan mundur dalam linked list.
* Manfaat Penggunaan Linked List :
* **Penyisipan dan Penghapusan Efisien :** Linked list memungkinkan penyisipan dan penghapusan elemen dengan lebih efisien daripada array karena tidak perlu memindahkan elemen lain.
* **Alokasi Dinamis :** Linked list memungkinkan alokasi dan dealokasi memori secara dinamis, mengoptimalkan penggunaan sumber daya.
* **Struktur Data Fleksibel :** Dengan linked list, pengembang dapat dengan mudah menyesuaikan ukuran dan struktur data sesuai kebutuhan tanpa harus menentukan ukuran sebelumnya seperti pada array.

1. **Double Link List :**

* Double linked list adalah bentuk yang lebih kompleks dari linked list yang memungkinkan setiap simpul (node) terhubung ke simpul sebelumnya dan berikutnya. Dalam artikel ini, kita akan menjelaskan konsep dasar double linked list, kelebihan yang dimilikinya, serta bagaimana struktur data ini digunakan dalam pengembangan perangkat lunak. Double linked list mirip dengan singly linked list, namun setiap simpul dalam double linked list memiliki dua pointer: satu untuk menunjuk ke simpul berikutnya (next) dan satu untuk menunjuk ke simpul sebelumnya (prev). Struktur node dalam double linked list dapat didefinisikan sebagai berikut:

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

* Penggunaan double linked list mirip dengan singly linked list. Setiap simpul dapat diakses dan dimanipulasi dengan menggunakan pointer next dan prev. Contoh sederhana penggunaan double linked list adalah sebagai berikut:

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

// Membuat simpul baru

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = 42;

newNode->next = NULL;

newNode->prev = NULL;

* Manfaat Penggunaan Double Linked List :
* **Pengolahan Data Secara Dinamis:** Double linked list memungkinkan alokasi dan dealokasi memori secara dinamis, memberikan fleksibilitas dalam pengelolaan data.
* **Pencarian dan Penghapusan Efisien:** Kemampuan navigasi maju dan mundur membuat double linked list efisien dalam operasi pencarian dan penghapusan elemen.

1. **Circular Link List :**

* Circular linked list adalah bentuk khusus dari linked list di mana ujung terakhir dari linked list terhubung kembali ke ujung pertamanya, membentuk suatu lingkaran tertutup. Dalam artikel ini, kita akan membahas konsep dasar circular linked list, cara penggunaannya, dan keunggulan yang dimilikinya dalam pengembangan perangkat lunak. Circular linked list mirip dengan singly linked list, namun simpul terakhirnya tidak menunjuk ke NULL, melainkan kembali ke simpul pertama. Struktur node dalam circular linked list dapat didefinisikan sebagai berikut:

struct Node {

int data;

struct Node\* next;

};

* Penggunaan circular linked list mirip dengan singly linked list, dengan penambahan bahwa simpul terakhir tidak menunjuk ke NULL, melainkan kembali ke simpul pertama. Contoh sederhana penggunaan circular linked list adalah sebagai berikut:

struct Node {

int data;

struct Node\* next;

};

// Membuat simpul baru

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = 42;

newNode->next = NULL; // Perlu diatur saat menghubungkan dengan simpul lainnya

* Manfaat Penggunaan Circular Linked List :
* **Pengolahan Data Berkelanjutan:** Circular linked list memungkinkan pengolahan data berkelanjutan tanpa harus kembali ke simpul pertama setelah mencapai simpul terakhir.
* **Penghematan Penyimpanan Memori:** Struktur circular linked list dapat membantu menghemat penyimpanan memori karena tidak memerlukan alokasi tambahan untuk menyimpan simpul terakhir.