



# STRUKTUR DATA

Modul Praktikum

# MODUL PRAKTIKUM STRUKTUR DATA

Disusun oleh :  
Firdaus, MBA  
Nori Wilintika, SST, MTI  
Ibnu Santos SST, MT

Hak Cipta 2021 pada penulis.

Edisi Pertama,  
Cetakan Pertama: 2021

Penerbit:  
Politeknik Statistika STIS  
Jl. Otto Iskandardinata No. 64C Jakarta Timur 13330  
Telpon. (021) 8508812, 8191437,  
Facs (021) 8197577

Hak cipta dilindungi Undang-Undang. Dilarang memperbanyak sebagian atau seluruh isi bahan ajar ini dalam bentuk apa pun, baik secara elektronik maupun mekanik, termasuk mempotokopi, merekap, atau menggunakan sistem penyimpanan lainnya, tanpa izin tertulis dari Penerbit.

UNDANG-UNDANG NOMOR 19 TAHUN 2002 TENTANG HAK CIPTA
<ol style="list-style-type: none"><li>1. <i>Barang siapa dengan sengaja dan tanpa hak mengumukan atau memperbanyak suatu ciptaan atau member izin untuk itu, dipidana dengan penjara paling lama 7 (tujuh) tahun dan atau denda paling banyak Rp. 5.000.000.000,00 (lima miliar rupiah)</i></li><li>2. <i>Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran Hak Cipta atau Hak Terkait sebagaimana dimaksud pada ayat (1), dipidana dengan pidana penjara paling lama 5 (lima) tahun dan atau denda paling banyak Rp. 5.000.000.000,00 (lima miliar rupiah)</i></li></ol>

## KATA PENGANTAR

---

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa karena dengan rahmat, karunia, serta taufik dan hidayah-Nya kami dapat menyelesaikan modul Praktikum Struktur Data ini dengan tepat waktu. Kami mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu penyusunan modul ini.

Kami sangat berharap modul ini dapat berguna dalam mengantarkan mahasiswa Program D-IV Komputasi Statistik Politeknik Statistika STIS dalam memahami dan menerapkan mata kuliah Struktur Data. Kami juga menyadari sepenuhnya bahwa modul ini terdapat kekurangan dan jauh dari kata sempurna. Oleh sebab itu, kami berharap adanya kritik, saran dan usulan demi perbaikan modul yang telah kami buat di masa yang akan datang, mengingat tidak ada sesuatu yang sempurna tanpa saran yang membangun.

Akhir kata, semoga modul ini dapat bermanfaat dan dipahami bagi siapapun yang membacanya. Tak lupa kami mohon maaf apabila terdapat kesalahan kata-kata yang kurang berkenan dan kami memohon kritik dan saran yang membangun dari pembaca demi perbaikan modul ini di waktu yang akan datang. Terima kasih.

Jakarta, November 2021

Tim Penulis

## Daftar Isi

---

Daftar Isi .....	1
MODUL 1: PENGENALAN DAN PERSIAPAN PEMROGRAMAN BAHASA C .....	5
1.1    Deskripsi Singkat .....	5
1.2    Tujuan Praktikum .....	6
1.3    Material Praktikum .....	6
1.4    Kegiatan Praktikum .....	6
A.    Instalasi Code::Blocks .....	6
B.    Membuat kode program C menggunakan Code::Blocks .....	10
1.5    Penugasan .....	13
MODUL 2: TIPE DATA DAN ARRAY PADA C .....	15
2.1    Deskripsi Singkat .....	15
2.2    Tujuan Praktikum .....	15
2.3    Material Praktikum .....	15
2.4    Kegiatan Praktikum .....	15
A.    Tipe Data .....	15
B.    Typedef .....	20
C.    Array .....	21
2.5    Penugasan .....	25
MODUL 3: STRUCTURE DAN POINTER .....	27
3.1    Deskripsi Singkat .....	27
3.2    Tujuan Praktikum .....	27
3.3    Material Praktikum .....	27
3.4    Kegiatan Praktikum .....	27
A.    Structure .....	27
B.    Pointer .....	34
3.5    Penugasan .....	40
MODUL 4: SINGLE LINKED LIST .....	41
4.1    Deskripsi Singkat .....	41
4.2    Tujuan Praktikum .....	41
4.3    Material Praktikum .....	41
4.4    Kegiatan Praktikum .....	41
A.    Inisialisasi .....	42

B.	Pembuatan Sebuah Simpul .....	43
C.	Menampilkan Nilai dari Linked List .....	44
D.	Penambahan Simpul ke Dalam Linked List.....	45
E.	Penghapusan Simpul dari Linked List .....	47
4.5	Penugasan .....	50
MODUL 5: DOUBLE LINKED LIST.....		51
5.1	Deskripsi Singkat .....	51
5.2	Tujuan Praktikum .....	51
5.3	Material Praktikum .....	51
5.4	Kegiatan Praktikum .....	51
A.	Inisialisasi .....	51
B.	Pembuatan Sebuah Simpul .....	52
C.	Menampilkan Nilai dari Linked List .....	54
D.	Penambahan Simpul ke Dalam Linked List.....	54
E.	Penghapusan Simpul dari Linked List .....	57
5.5	Penugasan .....	59
MODUL 6: STACK (TUMPUKAN) .....		61
6.1	Deskripsi Singkat .....	61
6.2	Tujuan Praktikum .....	61
6.3	Material Praktikum .....	61
6.4	Kegiatan Praktikum .....	61
A.	Persiapan.....	61
B.	Pembuatan Fungsi push.....	64
C.	Pembuatan Fungsi pop .....	64
D.	Finalisasi .....	65
6.5	Penugasan .....	65
MODUL 7: Antrian .....		67
7.1	Deskripsi Singkat .....	67
7.2	Tujuan Praktikum .....	67
7.3	Material Praktikum .....	67
7.4	Kegiatan Praktikum .....	67
A.	Persiapan.....	67
B.	Pembuatan Fungsi dequeue .....	69
C.	Pembuatan Fungsi enqueue .....	69
D.	Finalisasi .....	71

7.5	Penugasan .....	72
MODUL 8: TREE BAGIAN 1 .....		73
8.1	Deskripsi Singkat .....	73
8.2	Tujuan Praktikum .....	73
8.3	Material Praktikum .....	73
8.4	Kegiatan Praktikum .....	73
A.	Binary Tree .....	73
B.	Binary Search Tree .....	76
8.5	Penugasan .....	80
MODUL 9: TREE BAGIAN 2 .....		81
9.1	Deskripsi Singkat .....	81
9.2	Tujuan Praktikum .....	81
9.3	Material Praktikum .....	81
9.4	Kegiatan Praktikum .....	81
A.	Menambah Simpul Baru.....	81
B.	Menghapus Sebuah Simpul.....	86
9.5	Penugasan .....	88
MODUL 10: Hashing .....		89
10.1	Deskripsi Singkat .....	89
10.2	Tujuan Praktikum .....	89
10.3	Material Praktikum .....	89
10.4	Kegiatan Praktikum .....	89
	Hashing.....	89
10.5	Penugasan .....	97
MODUL 11: Pencarian .....		99
11.1	Deskripsi Singkat .....	99
11.2	Tujuan Praktikum .....	99
11.3	Material Praktikum .....	99
11.4	Kegiatan Praktikum .....	99
A.	Pencarian Sekuensial.....	99
11.5	Penugasan .....	109
MODUL 12: Pengurutan.....		110
12.1	Deskripsi Singkat .....	110
12.2	Tujuan Praktikum .....	110
12.3	Material Praktikum .....	110

12.4	Kegiatan Praktikum .....	110
A.	Insertion Sort.....	110
12.5	Penugasan .....	117
MODUL 13: GRAPH .....		119
13.1	Deskripsi Singkat .....	119
13.2	Tujuan Praktikum .....	119
13.3	Material Praktikum .....	119
13.4	Kegiatan Praktikum .....	119
A.	Graph Berarah tak berbobot.....	119
13.5	Penugasan .....	126
MODUL 14: Tugas Akhir .....		128

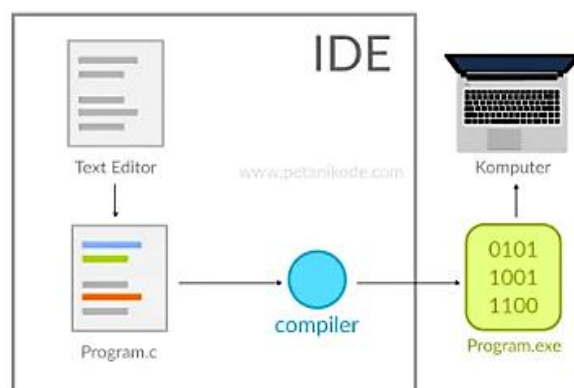


## MODUL 1: PENGENALAN DAN PERSIAPAN PEMROGRAMAN BAHASA C

### 1.1 Deskripsi Singkat

Dalam praktikum mata kuliah Struktur Data, kita akan menggunakan bahasa C. Bahasa C merupakan bahasa pemrograman yang berada di antara bahasa tingkat rendah (*low level programming language*) dan bahasa tingkat tinggi (*high level programming language*). Selain memiliki keunggulan dari bahasa pemrograman tingkat tinggi, yaitu menggunakan perintah yang mudah dipahami manusia, bahasa C juga mempunyai kecepatan eksekusi mendekati kecepatan eksekusi bahasa tingkat rendah dan juga kemampuan memanipulasi alamat data secara langsung. Keunggulan inilah yang kemudian menjadikan bahasa C banyak dipilih oleh programmer komputer untuk menyusun aplikasi komputer.

Untuk dapat mengembangkan program menggunakan suatu bahasa pemrograman, dalam kasus kita yaitu bahasa C, diperlukan adanya lingkungan pengembangan (*development environment*) yang mendukung bahasa tersebut. Pada dasarnya kita hanya membutuhkan dua alat saja, yaitu: *text editor* dan *compiler*. *Text editor* adalah program yang digunakan untuk menulis kode program C. *Compiler* adalah program yang digunakan untuk menerjemahkan bahasa C ke dalam bahasa mesin sehingga dapat dimengerti oleh komputer. Ada banyak pilihan yang teks editor yang bisa digunakan untuk menuliskan kode dalam bahasa C, seperti Visual Studio Code, Sublime, Brackets, Atom, Geany, dan lain-lain. Begitu pula dengan Compiler bahasa C, juga ada banyak jenisnya, seperti GCC, Tiny C, Clang, Small-C, Borland Turbo C, Power C, Quick C, dan sebagainya. Namun terdapat *software* yang di dalamnya memiliki *text editor* dan *compiler* sekaligus, sehingga kita tidak perlu untuk menginstall teks editor dan compiler secara terpisah. *Software* inilah yang disebut IDE (*Integrated Development Environment*). Alur *compile* dengan menggunakan IDE ditunjukkan pada Gambar berikut.



Alur Compile dengan IDE



Ada berbagai IDE yang bisa digunakan dalam pemrograman bahasa C, seperti Dev-C++, Turbo C++, Code::Blocks, Qt Creator, dan lain-lain. Dalam praktikum ini akan dicontohkan instalasi dan penggunaan Code::Blocks.

## 1.2 Tujuan Praktikum

Setelah praktikum pada modul 1 ini diharapkan mahasiswa mempunyai kompetensi sebagai berikut:

- 1) Dapat mengenal bahasa pemrograman C dan Code::Blocks
- 2) Dapat melakukan melakukan instalasi Code::Blocks dan mengelola kode program C menggunakan Code Blocks.

## 1.3 Material Praktikum

Pada kegiatan modul 1 diperlukan beberapa material berupa file, yaitu:

- 1) File installer Code::Blocks
- 2) Referensi Bahasa Pemrograman C,  
seperti : <https://www.tutorialspoint.com/cprogramming/index.htm>

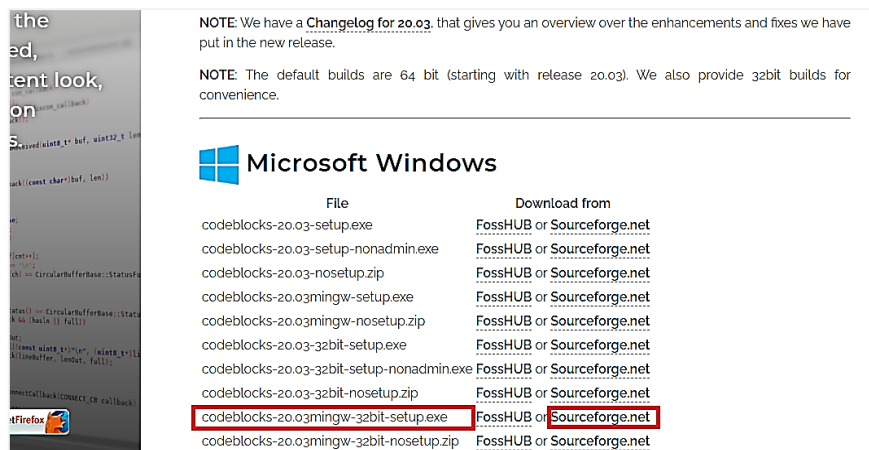
## 1.4 Kegiatan Praktikum

### A. Instalasi Code::Blocks

CodeBlocks merupakan aplikasi Open Source yang bisa didapat dengan gratis. IDE ini juga tersedia untuk sistem operasi Linux, Mac, dan Windows. Untuk mulai mengunduh aplikasi Code::Blocks, silahkan buka alamat dibawah ini:

<https://www.codeblocks.org/downloads/binaries/>

Pada halaman tersebut silahkan pilih versi sistem operasi yang anda gunakan dan pilih sumber server untuk mengunduh file. Jika tidak yakin versi mana yang harus digunakan, pilih yang **“mingw-setup”**, atau tepatnya **“codeblocks-20.03mingw-32bit-setup.exe”**. Langkah ini sangat penting karena jika salah download, compiler C tidak akan terinstall.



the  
ed,  
tent look,  
on  
S  
SerialDate, buf, size32\_t len  
lock(const char\* buf, len)  
0;  
[ctrl];  
ch) = CircularBufferBase::StatusFa  
Status() = CircularBufferBase::Stat  
K 4u (size || full)  
if (const uint8\_t\* u8, (uint8\_t\*) l  
CircularBuffer::Index, full)  
SerialCallback.CONNECT\_OR\_CALLBACK  
Firefox

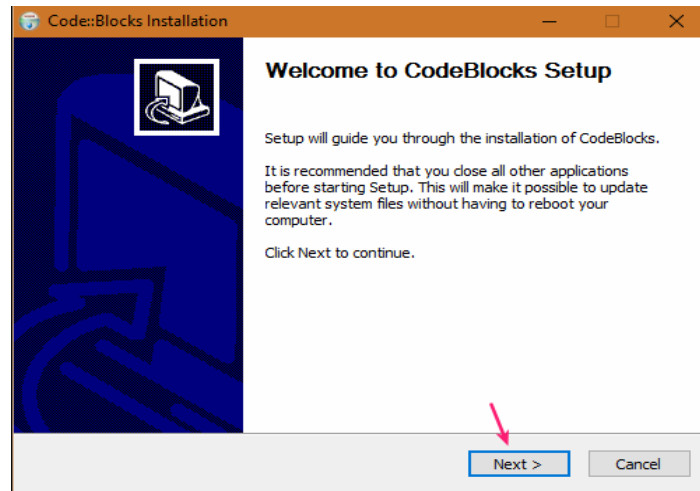
NOTE: We have a [Changelog for 20.03](#), that gives you an overview over the enhancements and fixes we have put in the new release.

NOTE: The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.

### Microsoft Windows

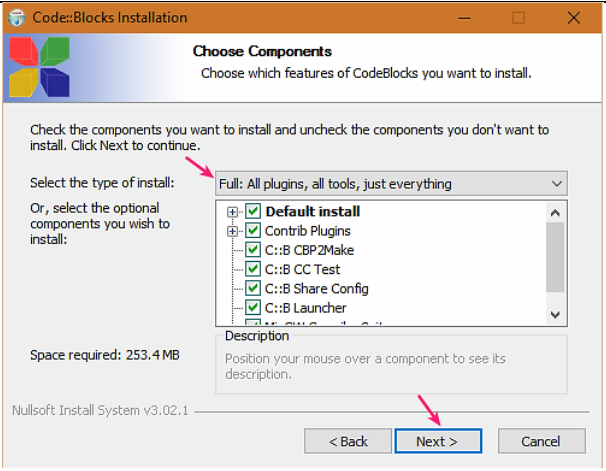
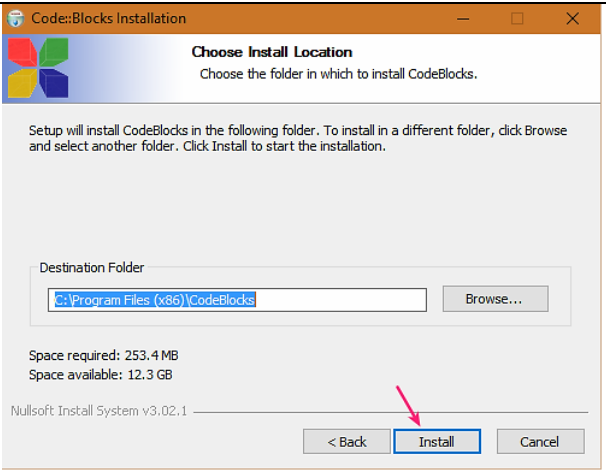
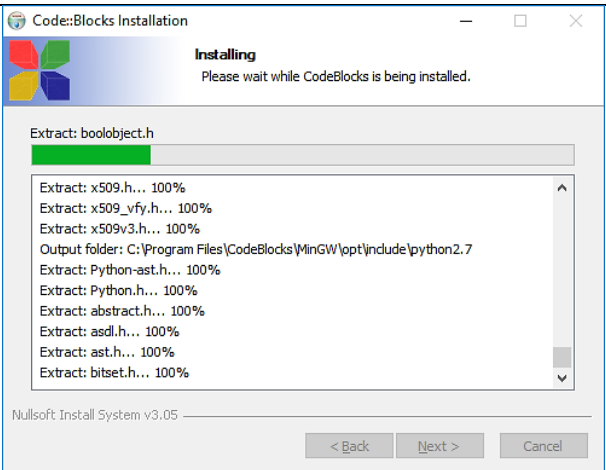
File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
<b>codeblocks-20.03mingw-32bit-setup.exe</b>	<b>FossHUB or Sourceforge.net</b>
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

Jika tidak ada kendala, proses download akan berjalan. Setelah itu, kita akan mendapatkan file **codeblocks-20.03mingw-32bit-setup.exe** berukuran sekitar **145MB**. Bukalah file tersebut dengan level administrator. Klik kanan, kemudian pilih Run as administrator. Maka akan tampil jendela awal proses instalasi.

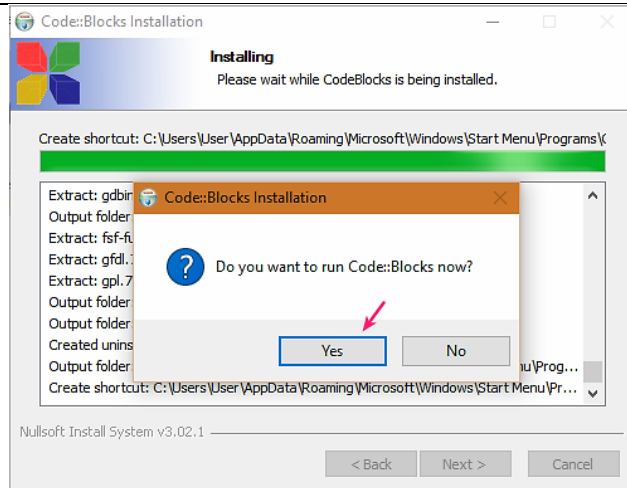


Selanjutnya, klik tombol **“Next”** dan lanjutkan instalasi dengan mengikuti Langkah-langkah berikut ini.

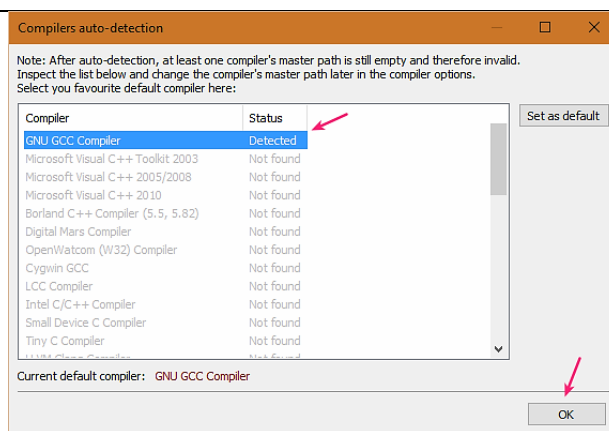
Langkah	Tampilan Windows
1. Pada halaman <b>“License Agreement”</b> , klik saja tombol <b>“I Agree”</b>	

<p>2. Halaman berikutnya adalah <b>“Choose Components”</b>, biarkan pilihan default (seluruh pilihan di centang), lalu klik tombol <b>“Next”</b>.</p>	
<p>3. Selanjutnya kita akan diminta untuk menentukan lokasi folder instalasi.</p>	
<p>4. Klik Tombol <b>“Install”</b> dan proses instalasi akan berlangsung beberapa saat.</p>	

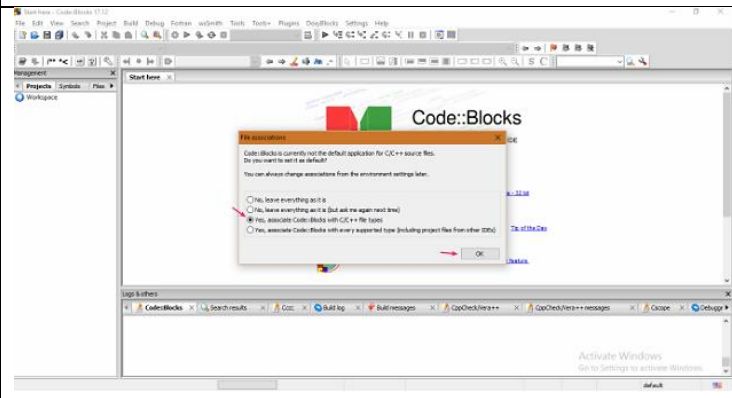
5. Setelah proses instalasi selesai, akan tampil jendela konfirmasi **“Do you want to run Code::Blocks now?”** Klik **Yes** agar setelah proses instalasi, IDE Code::Blocks langsung tampil.



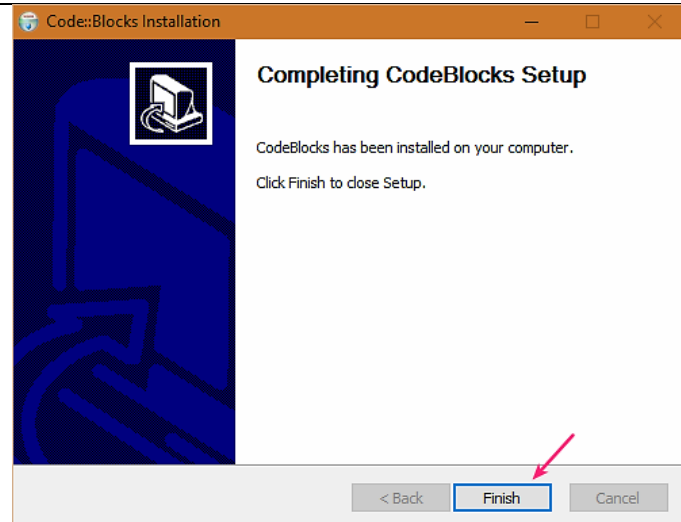
6. Saat Codeblocks baru pertama kali dibuka, kita diminta untuk menentukan compiler yang akan digunakan. Pilih saja **GNU GCC**.



7. Kemudian pilih **“Yes, associate Codeblocks With C/C++ file type”** agar source code program C dan C++ dibuka otomatis dengan Code::Blocks.

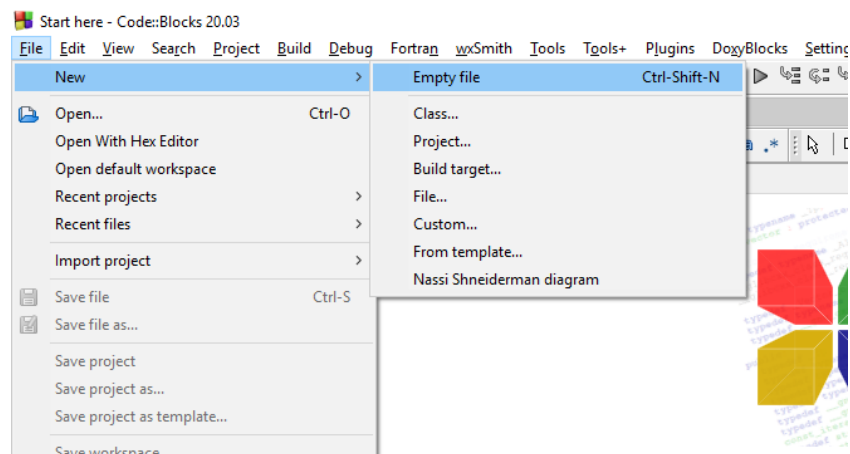


8. Terakhir, klik **Finish**. Instalasi Code::Blocks selesai. Sekarang kita bisa menggunakannya untuk membuat program C.



## B. Membuat kode program C menggunakan Code::Blocks

1. Untuk membuat file baru, klik menu **File -> New -> Empty File**, atau bisa juga dengan menekan kombinasi tombol **CTRL+ SHIFT+ N**.



Di bagian tengah Code::Blocks akan tampil sebuah file teks kosong. Disinilah nantinya kita menulis kode program bahasa C.

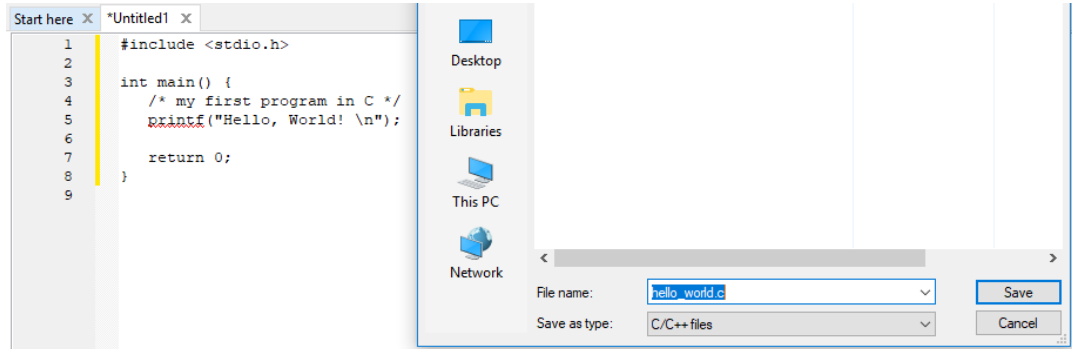
2. Coba ketik kode program berikut ke dalam Code::Blocks:

```
#include <stdio.h>

int main() {
    /* my first program in C */
    printf("Hello, World! \n");

    return 0;
}
```

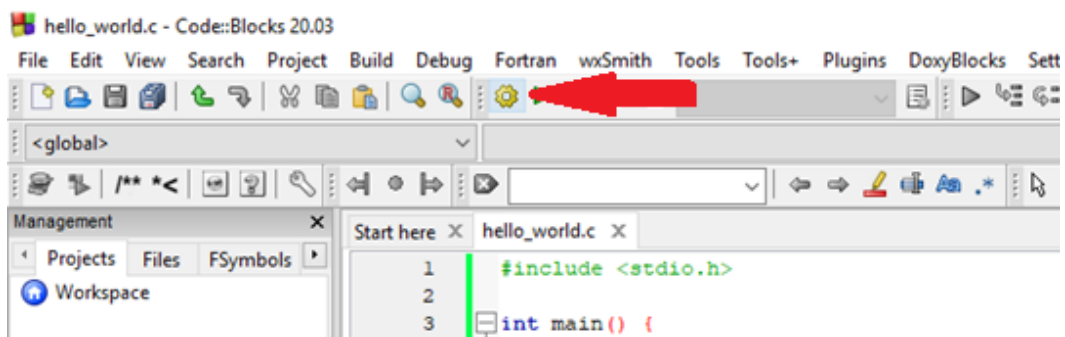
3. Simpan file kode program yang sudah kita buat dengan cara klik menu **File-> Save file** atau dengan cara menekan kombinasi tombol **CTRL+ S**. Sesaat kemudian akan tampil jendela **Save File**. Simpan file ke dalam folder yang sudah disiapkan sebelumnya. Isi nama file, misalnya dengan **hello\_world.c**.



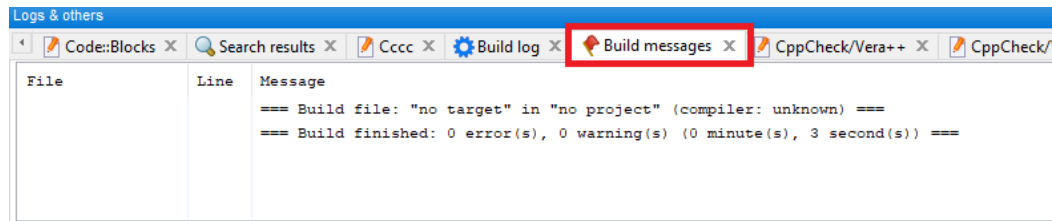
Ketika file sudah disimpan, tampilan akan kembali ke editor Code::Blocks dan kode program yang sudah kita tulis sebelumnya akan berwarna. Inilah fitur *syntax highlighting* dari Code::Blocks. IDE Code::Blocks memberi warna berbeda untuk setiap kode program C. Dengan demikian jika kita salah ketik atau salah tulis kode program, warna teks yang dihasilkan juga akan berbeda.

4. Sekarang saatnya kita jalankan kode tersebut. Untuk bahasa pemrograman dengan teknik compiler seperti C, ada 2 proses yang harus dilakukan: *compile* dan *run*. Proses compile dipakai untuk mengolah file bahasa C menjadi object file dan juga file .exe. Sedangkan proses run akan menjalankan file .exe yang dihasilkan.

Untuk meng-compile file kode program C di Code::Block, pilih menu **Build -> Compile current File** atau bisa juga dengan menekan tombol **CTRL + SHIFT + F9**. Di dalam Code::Block juga terdapat proses build yang pada dasarnya nyaris sama dengan proses compile. Ini pun juga bisa dijalankan dengan cara mengklik icon gear berikut:

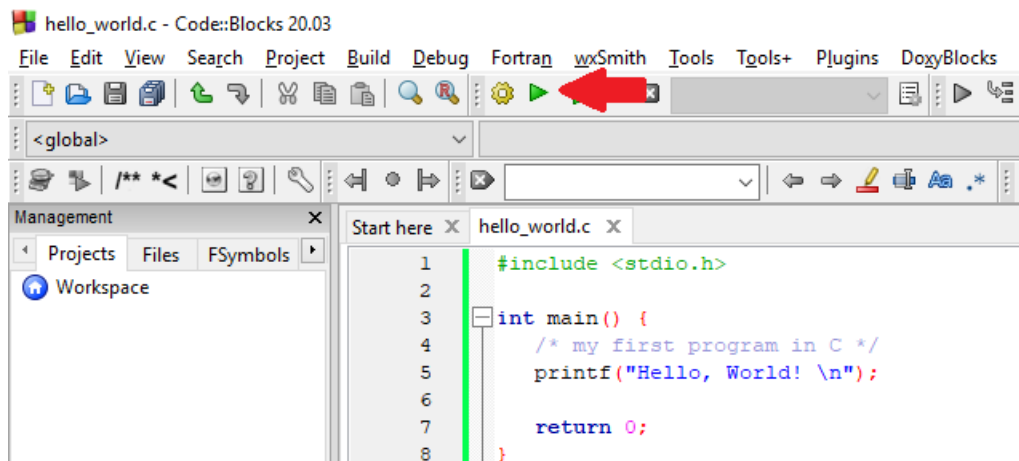


Proses compile akan berlangsung beberapa saat (mungkin hanya 1 detik untuk kode sederhana seperti ini). Log atau catatan mengenai proses compile juga bisa dilihat pada tab **"Build messages"** di bagian bawah:



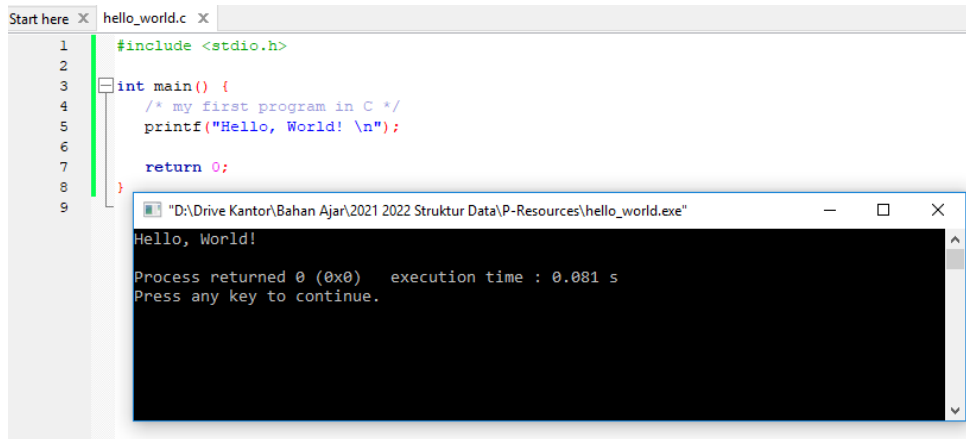
Jika tampil teks seperti pada gambar di atas, artinya proses compile berhasil dan kode program kita tidak memiliki error.

5. Untuk lebih memastikan, silahkan buka kembali folder dimana file **hello\_world.c** disimpan. Sebelumnya hanya terdapat 1 file, yakni **hello\_world.c**. sekarang akan ada 2 file tambahan: **hello\_world.o** dan **hello\_world.exe**. File **hello\_world.o** merupakan object file hasil proses compile. Object file ini digunakan secara internal oleh compiler bahasa C dan tidak perlu kita utak-atik. Sedangkan file **hello\_world.exe** adalah file akhir hasil compiler + linker, dimana object file sudah diproses lebih lanjut untuk menjadi program akhir.
6. Untuk melihat hasil kode program, kembali lagi ke Code::Block. Sekarang kita jalankan proses Run. Caranya, buka menu **Build -> Run**, atau tekan tombol **Ctrl + F10**, atau bisa juga klik icon panah warna hijau di sebelah icon build:



7. Dan berikut hasil dari run file **hello\_world.c**:





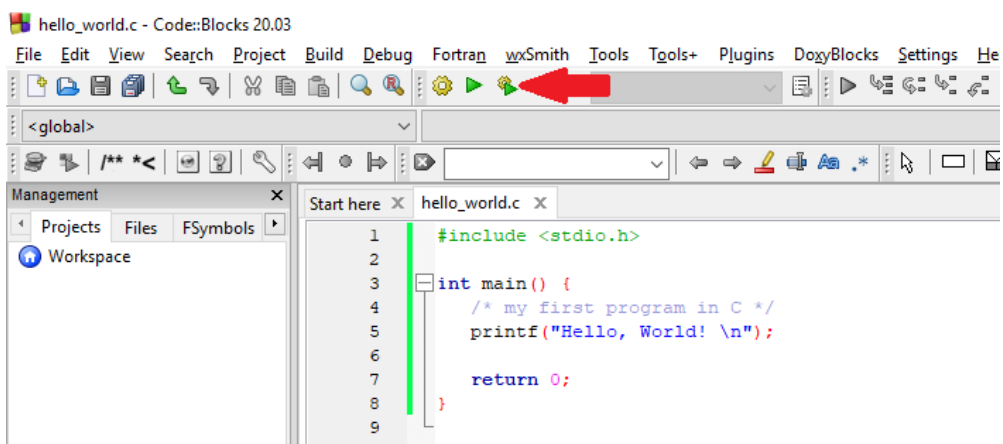
```
1 #include <stdio.h>
2
3 int main() {
4     /* my first program in C */
5     printf("Hello, World! \n");
6
7     return 0;
8 }
9
```

Hello, World!

Process returned 0 (0x0) execution time : 0.081 s  
Press any key to continue.

Jika dalam jendela cmd tampil teks “Hello World!” di baris pertama, Selamat!, artinya kita sudah berhasil men-compile dan menjalankan kode program yang ditulis dalam bahasa C. Untuk menutup jendela ini, tekan sembarang tombol *keyboard* atau bisa juga klik icon silang di sudut kanan atas (seperti menutup aplikasi pada umumnya). Selama jendela hasil ini tidak ditutup, maka kita tidak akan bisa melakukan proses compile atau run file lain dari Code::Block.

8. Code::Block juga menyediakan tombol untuk proses **build dan run** sekaligus, yakni dengan mengklik icon “**Build and Run**” atau menekan tombol F9:



Dengan menekan tombol “**Build and Run**” atau tombol F9, kode C yang ada langsung di compile dan dijalankan sekaligus. Ini sangat praktis karena hasilnya bisa langsung terlihat.

## 1.5 Penugasan

Menggunakan Bahasa Pemrograman C:

1. Buatlah sebuah program yang dapat yang dapat mengolah data mahasiswa dengan ketentuan:
  - Pengguna menginputkan NIM, Nama, Nilai Kuis, Nilai UTS, dan Nilai UAS.

- Hasil atau output yang diinginkan: NIM, Nama, Nilai Kuis, Nilai UTS, Nilai UAS, dan Nilai Akhir.
  - Nilai akhir = 20% dari nilai kuis + 30% dari nilai UTS + 50% dari nilai UAS.
2. Buatlah program untuk menghitung luas dan keliling persegi panjang, dengan ketentuan sebagai berikut:
- Data panjang dan lebar persegi panjang diinput oleh pengguna.
  - Hitung luas dan keliling ditulis pada fungsi-fungsi tersendiri yang kemudian dipanggil pada fungsi `main()`.
  - Luas dan keliling bangun persegi panjang ditampilkan sebagai keluaran.

## MODUL 2: TIPE DATA DAN ARRAY PADA C

---

### 2.1 Deskripsi Singkat

Secara umum, struktur data dikelompokkan menjadi dua kategori, yaitu struktur data primitif (sederhana) dan struktur data non primitif (majemuk). Struktur data primitif adalah tipe data dasar (*primitive data types*) yang didukung oleh bahasa pemrograman (*built-in*), misalnya Integer, floating point, characters, pointer, dan-lain-lain. Struktur data non primitif misalnya arrays, structure, stack, queues, linked lists, dan lain-lain. Struktur data non-primitif diturunkan dari struktur data primitif. Oleh karena itu, pemahaman mengenai tipe data sangat penting untuk memahami dan mengimplementasikan struktur data. Dalam praktikum kita ini akan memahami tipe data dasar dan Array dalam Bahasa Pemrograman C serta mempraktekkan penggunaannya.

### 2.2 Tujuan Praktikum

Setelah praktikum pada modul 2 ini diharapkan mahasiswa mampu:

- 1) Memahami tipe data dasar dalam Bahasa Pemrograman C
- 2) Mendeklarasikan deklarasi data, serta mengoperasikan dan menampilkannya dalam format yang sesuai
- 3) Menggunakan array, khususnya array multidimensi, dalam penyimpanan data

### 2.3 Material Praktikum

Kegiatan pada modul 2 ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

### 2.4 Kegiatan Praktikum

#### A. Tipe Data

Dalam bahasa C terdapat beberapa jenis tipe data yang bisa digunakan untuk mendeklarasikan sebuah variabel, konstanta, atau fungsi pada program. Jenis variabel menentukan berapa banyak ruang yang ditempati dalam penyimpanan dan bagaimana pola bit yang disimpan ditafsirkan. Variabel biasa digunakan di dalam program dengan tujuan untuk menampung data. Nilai yang terdapat pada variabel sewaktu-waktu dapat diubah. Jumlah variabel yang dibuat dapat tidak terbatas, namun masing-masing variabel tersebut harus bersifat unik dan tidak boleh ada nama variabel yang sama. Agar dapat digunakan, sebelumnya variabel dideklarasikan dahulu beserta tipe data yang akan disimpan dalam variabel tersebut. variabel dideklarasikan dengan format:

```
tipe_data nama_variabel
```

Adapun aturan-aturan penamaan variabel dalam bahasa C adalah sebagai berikut :

1. Harus diawali dengan huruf (tidak boleh diawali dengan angka), selanjutnya dapat diikuti dengan angka atau underscore (\_).
2. Tidak boleh mengandung spasi, simbol, atau karakter khusus lain selain huruf, angka, dan underscore.
3. Bersifat case sensitive.
4. Tidak boleh memakai kata kunci (keywords) yang telah digunakan oleh Bahasa C.

Selain variabel terdapat konstanta yang juga dapat menampung data. Hanya saja dalam konstanta nilai yang ada tidak dapat diubah atau bernilai pasti. Konstanta dideklarasikan dengan menggunakan preprocessor **#define**:

```
#define nama_konstanta
```

atau dengan dengan singkatan **const**:

```
const tipe_data dan nama_konstanta.
```

Dalam pembuatan fungsi, tipe data digunakan untuk mendeklarasikan tipe data kembalian dari fungsi. Berikut format dasar cara penulisan fungsi dalam bahasa C:

```
tipeDataKembalian namaFunction() {  
    // Isi function disini...  
    // Isi function disini...  
    return nilai;  
}
```

Jika suatu fungsi tidak mengembalikan nilai, **tipeDataKembalian** ditulis sebagai **void**. Sebuah fungsi yang tidak mengembalikan nilai kadang disebut juga sebagai **procedure**.

C menyediakan lima macam tipe data dasar, yaitu:

- tipe data integer (nilai numerik bulat, yang dideklarasikan dengan **int**)
- floating-point (nilai numerik pecahan ketetapan tunggal, yang dideklarasikan dengan **float**)
- double-precision (nilai numerik pecahan ketetapan ganda, yang dideklarasikan dengan **double**)
- karakter (dideklarasikan dengan **char**)
- **void** (tidak bertipe, tidak menyimpan apapun, biasanya untuk tipe fungsi yang tidak return value apapun)

1. a. Berikut cara mendeklarasikan variable menggunakan **int**. Ketik ulang kode program berikut ke dalam Code::Blocks atau IDE C lainnya yang Anda gunakan.

```
#include <stdio.h>

int main() {
    int tanggal = 17;
    printf("Tanggal %i", tanggal);
    return 0;
}
```

Simpan potongan program tersebut dengan nama **praktikum2\_1a.c**. Jalankan program tersebut. Output yang didapatkan adalah sebagai berikut:

```
Tanggal 17
```

- 
- b. Kita juga bisa mendeklarasikan dua variable sekaligus jika tipe datanya sama. Modifikasi program **praktikum2\_1a.c** menjadi seperti berikut ini.

```
#include <stdio.h>

int main() {
    int tanggal = 17;
    int bulan = 8;
    printf("Tanggal %i, Bulan %d", tanggal, bulan);
    return 0;
}
```

Simpan ulang dan jalankan program. Hasilnya adalah sebagai berikut:

```
Tanggal 17, Bulan 8
```

- 
- 
- c. Bisa kita perhatikan, pada variabel tanggal kita menggunakan format specifier **%i** sedangkan pada variabel bulan kita menggunakan format specifier **%d**. Pada **printf()** tidak ada perbedaan antara keduanya. Namun ketika menggunakan **scanf()** dan menginput nilai dengan 0 didepan angka (011, 012, 013), maka akan terlihat perbedaannya. Pada **%d** diasumsikan sebagai basis 10 (desimal) sedangkan **%i** diasumsikan sebagai basis 8 (oktal). Silakan dicoba, dengan mengetikkan ulang potongan program di bawah ini, lalu simpan dengan nama **praktikum2\_1c.c**.

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

```

int main()
{
    int a, b;
    printf("Angka pertama: ");
    scanf("%d", &a);
    printf("Angka kedua: ");
    scanf("%i", &b);
    printf("\nNilai desimal dari angka pertama adalah:
%i", a);
    printf("\nNilai oktal dari angka kedua adalah:
%i", b);

    return 0;
}

```

Jika kita berikan **011** sebagai input, maka output yang didapatkan adalah sebagai berikut:

```

Angka pertama: 011
Angka kedua: 011

Nilai desimal dari angka pertama adalah: 11
Nilai oktal dari angka kedua adalah: 9

```

2. Jika kita menggunakan **float** atau **double** untuk bilangan desimal, double akan dua kali lebih teliti daripada float. Float memiliki ketepatan 7 digit desimal, sedangkan double memiliki ketepatan hingga 15 digit desimal. Contoh kode penggunaan kedua tipe data ini adalah sebagai berikut:

```

#include <stdio.h>

void pifloat()
{
    float pi = 22.0/7.0;
    printf("=== FLOAT ===\n");
    printf("%.15lf\n", pi);
    printf("%lf\n", pi);
}

```

```

}

void pidouble()
{
    double pi = 22.0/7.0;
    printf("=== DOUBLE ===\n");
    printf("%.15lf\n", pi);
    printf("%.1f\n", pi);
}

int main()
{
    pifloat();
    printf("\n");
    pidouble();

    return 0;
}

```

Ketik ulang program tersebut, lalu simpan dengan nama **praktikum2\_2.c**. Bila dijalankan, outputnya adalah:

```

=== FLOAT ===
3.142857074737549
3.142857

=== DOUBLE ===
3.142857142857143
3.142857

```

Bisa kita lihat, ada perbedaan antara penggunaan float dan double pada desimal ketujuh.

3. Keyword **char** digunakan untuk mendeklarasikan tipe data yang memuat nilai berupa karakter, contohnya adalah sebagai berikut:

```

#include <stdio.h>

int main() {

```



```

char huruf = 'Y';

printf("%c", huruf);

return 0;
}

```

Ketik ulang program di atas, lalu simpan dengan nama **praktikum2\_3a.c**. Output:

```
Y
```

Dengan menggunakan kode diatas hanya akan mengeluarkan satu huruf saja. Untuk menyimpan kalimat atau **string**, maka harus kita ubah terlebih dahulu kodenya. Simpan potongan program di bawah ini dengan nama **praktikum2\_3b.c**.

```

#include <stdio.h>

int main() {
    char kalimat[10] = "AnbiDev";
    printf("%s", kalimat);

return 0;
}

```

Output yang dihasilkan:

```
AnbiDev
```

Terlihat perbedaan pada kode pada **praktikum2\_3a.c** dan **praktikum2\_3b.c**. Pada **praktikum2\_3a.c** kita menggunakan tanda petik satu untuk nilainya dan menggunakan format specifier **%c**. Pada **praktikum2\_3b.c**, kita menggunakan tanda petik dua dan menggunakan format specifier **%s** sebagai ganti **%c**.

## B. Typedef

Pada bahasa C, kita dapat memberikan sebuah nama baru pada tipe data. Keyword **typedef** biasanya digunakan untuk membuat nama alias dari tipe data.

Bentuk umum:

```
typedef <bentuk asal> <nama type>;
```

Contoh:

```
typedef int bulat;

typedef struct {
    double r, theta;
}Kompleks;
```

4. Sebagai contoh penggunaan typedef, simpan program di bawah ini dengan nama **praktikum2\_4.c**, lalu coba jalankan.

```
#include <stdio.h>

typedef unsigned char BYTE;

int main()
{
    BYTE b1, b2;
    b1 = 'c';
    printf("%c ", b1);
    return 0;
}
```

Pada program tersebut, dapat dilihat bahwa **unsigned char** selanjutnya dapat digunakan/dipanggil dengan nama **BYTE**.

### C. Array

Array merupakan suatu tipe data yang terstruktur dan dapat digunakan untuk menyimpan data yang memiliki tipe data yang sama. Array biasa juga disebut larik. Penggunaan array dapat mengurangi kerumitan dalam proses penyimpanan data dalam jumlah yang besar. Jika kita hendak menyimpan variabel nama yang berjumlah sepuluh nama, kita tidak harus membuat 10 variabel untuk nama tersebut. Dengan menggunakan array kita tidak perlu membuat banyak variabel untuk data yang memiliki tipe yang sama. Selain itu, dalam alokasi memori penyimpanan, tipe data array melakukan pemesanan tempat terlebih dahulu sesuai dengan kebutuhan yang ada.

Terdapat 2 jenis array, yaitu array 1 dimensi dan array 2 dimensi. array dengan 1 dimensi merupakan array yang dapat digambarkan sebagai sebuah baris. Dalam array 1 dimensi, elemen yang ada di dalamnya dapat diakses hanya dengan menggunakan 1 indeks saja. Sedangkan array 2 dimensi merupakan array yang dapat digambarkan seperti sebuah matrik. Selain itu elemen yang ada dalam array 2 dimensi dapat diakses dengan menggunakan 2 indeks, yaitu indeks kolom dan juga indeks baris. Berikut format pendeklarasian array:

```
tipe_data nama_array[jumlah_element];
```

Contoh:

```
int Number[10];
```

Elemen pada array dapat diinisialisasi sebagai berikut:

```
int Number[10]={1,3,2,4,5,7,6,9,8,0};
```

5. Salinlah program di bawah ini untuk mempraktekkan pendeklarasian dan pengaksesan array 1 dimensi. Simpan dengan nama **praktikum2\_5.c**, lalu jalankan.

```
#include<stdio.h>
int main()
{
    char vokal[5];
    int i;
    vokal[0]='A';
    vokal[1]='I';
    vokal[2]='U';
    vokal[3]='E';
    vokal[4]='O';

    for(i=0;i<5;i++)
    {
        printf("%c\n", vokal[i]);
    }
    return 0;
}
```

Setelah program dijalankan, diketahui bahwa pendeklarasian array dapat dilakukan dengan langsung mengisi nilai dari tiap indeksinya.

6. Contoh dibawah ini adalah penggunaan array untuk menyimpan sejumlah angka dan menghitung rata-ratanya. Ketik program pada IDE C Anda, lalu simpan dengan nama **praktikum2\_6.c**.

```
#include <stdio.h>
int main(void)
{
    int numbers[10];
    int count = 10;
```

```

long sum = 0L;
float average = 0.0f;
printf("\n Masukkanlah 10 Angka:\n");
for(int i = 0; i < count; i ++)
{
printf("%2d> ", i+1);
scanf("%d", &numbers[i]); /* Read a number */
sum += numbers[i]; /* Jumlahkan setiap elemen */
}
average = (float)sum/count; /* Hitung rata-rata */
printf("\n Rata-rata dari sepuluh Angka yang
dimasukkan: %f\n", average);
return 0;
}

```

Jika program dijalankan, hasilnya adalah:

```

Masukkanlah 10 Angka:
1> 450
2> 765
3> 562
4> 700
5> 598
6> 635
7> 501
8> 720
9> 689
10> 527
Rata-rata dari sepuluh Angka yang dimasukkan:
614.700000

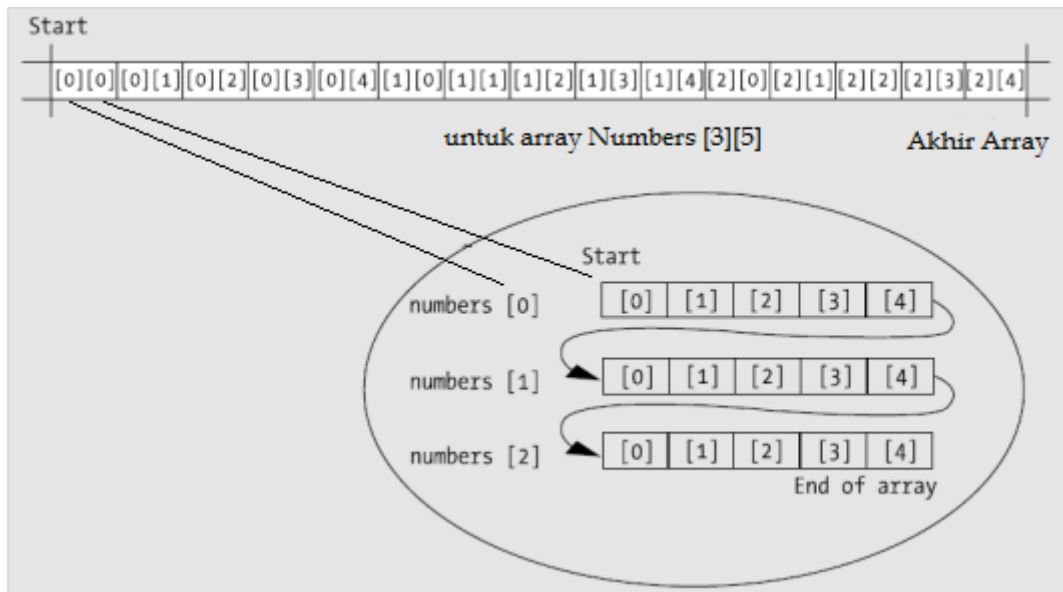
```

Pada contoh sebelumnya, array yang digunakan adalah array 1 dimensi, yang dicirikan dengan menggunakan 1 tanda “[ ]”. Array multi dimensi akan menggunakan lebih dari satu “[ ]” untuk menyatakan elemen yang ada pada variabel tersebut. Contoh:

```

float Numbers[3][5];

```



Contoh inisiasi array multidimensi seperti yang ditampilkan dibawah ini:

```
int numbers[3][4] = {
    { 10, 20, 30, 40 },
    { 15, 25, 35, 45 },
    { 47, 48, 49, 50 }
};

int numbers[2][3][4] = {
    {
        { 10, 20, 30, 40 },
        { 15, 25, 35, 45 },
        { 47, 48, 49, 50 }
    },
    {
        { 10, 20, 30, 40 },
        { 15, 25, 35, 45 },
        { 47, 48, 49, 50 }
    }
};
```

7. Contoh di bawah ini menunjukkan penggunaan array 2 dimensi dalam hal penjumlahan dua matriks.

```
#include <stdio.h>

#define ROW 2
#define COL 3

int main(void)
{
    int Matriks_A[ROW][COL]={ {4,2,3}, {5,7,6} };
    int Matriks_B[ROW][COL]={ {1,8,9}, {3,5,4} };
}
```

```

int Matriks_C[ROW][COL];

printf("Matriks_C adalah : \n");
for(int i = 0; i < ROW; i++)
{
    for(int j = 0; j < COL; j++)
    {
        Matriks_C[i][j] = Matriks_A[i][j]+ Matriks_B[i][j];
    }
    printf("%d %d %d\n", Matriks_C[i][0], Matriks_C[i][1],
    Matriks_C[i][2]);
}
return 0;
}

```

Simpan program tersebut dengan nama **praktikum2\_7.c**. Cobalah untuk memodifikasi program tersebut untuk penjumlahan matriks berukuran lainnya.

## 2.5 Penugasan

1. **sizeof()** adalah sebuah operator untuk mengetahui jumlah memori (byte) yang diperlukan oleh suatu tipe data pada bahasa C. Gunakan **sizeof()** untuk mengetahui ukuran memori pada berbagai tipe data pada bahasa pemrograman C, seperti **char**, **int**, **float**, **double**. Catat hasil yang Anda dapatkan. Bandingkan dengan hasil yang didapatkan oleh teman-teman Anda. Diskusikan apakah yang menyebabkan hasil yang didapatkan berbeda-beda.
2. Ketik program berikut pada IDE Anda, lalu simpan.

```

/* Aturan Scope pada Bahasa C */
#include<stdio.h>

int main()
{
    {
        int x = 10, y = 20;
        {
            printf("x = %d, y = %d\n", x, y);
            {

```

```
        int y = 40;
        x++;
        y++;
        printf("x = %d, y = %d\n", x, y);
    }
    printf("x = %d, y = %d\n", x, y);
}
return 0;
}
```

Amati output yang dihasilkan. Jelaskan mengapa bisa terjadi perubahan demikian pada nilai  $x$  maupun  $y$ .

3. Buatlah program untuk menginput nilai elemen-elemen Matriks A berukuran 3x4 dan mencetak Matriks A tersebut. Contoh output sebagai berikut.

1	3	4	5
2	4	6	8
3	5	7	9



## MODUL 3: STRUCTURE DAN POINTER

---

### 3.1 Deskripsi Singkat

Pada praktikum sebelumnya kita telah mehami dan mempraktikkan penggunaan tipe data dasar dan array dalam Bahasa Pemrograman C. Pada array, kita dapat menampung kumpulan data yang sejenis. Sebagai contoh, jika sebuah array kita deklarasikan bertipe integer, maka semua data yang ada di dalam array tersebut semuanya harus bertipe integer. Untuk menampung sejumlah data yang memiliki tipe data yang berbeda, kita dapat menggunakan structure. Pada praktikum kali ini, kita akan mencoba mehami dan mempraktikkan penggunaan structure dalam Bahasa Pemrograman C. Selain kita juga akan mencoba memahami dan mempraktikkan penggunaan variabel pointer.

### 3.2 Tujuan Praktikum

- 1) Mahasiswa mampu mendeklarasikan dan menggunakan struct dalam bahasa C.
- 2) Mahasiswa mampu mendeklarasikan dan menggunakan pointer dalam bahasa C.

### 3.3 Material Praktikum

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

### 3.4 Kegiatan Praktikum

#### A. Structure

Structure (struktur) adalah kumpulan elemen-elemen data yang digabungkan menjadi satu kesatuan. Di dalam beberapa bahasa pemrograman, structure dikenal dengan nama **record**. Masing-masing elemen data tersebut dikenal juga dengan sebutan **field**. Elemen data tersebut dapat memiliki tipe data yang sama ataupun berbeda.

Mengapa kita membutuhkan structure? Misalnya kita ingin menyimpan data mahasiswa. Kita bisa saja melakukannya seperti ini:

```
char name[] = "Dian";  
char address[] = "Mataram";  
int age = 22;
```

Lalu bagaimana kalau ada lebih dari satu mahasiswa? Mungkin bisa saja kita buat seperti ini:

```
char name[] = "Dian";  
char address[] = "Mataram";  
int age = 22;
```

```
char name2[] = "Bambang";
char address2[] = "Surabaya";
int age2 = 23;

char name3[] = "Bimo";
char address3[] = "Jakarta";
int age3 = 23;
```

Untuk tujuan kemudahan dalam operasinya, elemen-elemen tersebut akan lebih baik bila digabungkan menjadi satu, yaitu dengan menggunakan structure. Dengan kata lain, structure merupakan bentuk struktur data yang dapat menyimpan variabel dengan satu nama.

1. Pendeklarasian structure selalu diawali kata baku **struct**, diikuti nama structure dan deklarasi field-field yang membangun structure di antara pasangan tanda kurung kurawal " {}". Berikut adalah bentuk umum dan contohnya:



Agar structure dapat digunakan, kita harus membuat variabel untuknya. Variabel structure dapat dideklarasikan bersamaan dengan deklarasi structure atau sebagai deklarasi terpisah seperti mendeklarasikan variabel dengan tipe data dasar. Cobalah untuk mendeklarasikan structure seperti contoh di bawah ini, lalu simpan dengan nama **praktikum3\_1a.c**. Pastikan tidak ada error saat program dijalankan.

```
struct data_tanggal
{
    int tahun;
    int bulan;
    int tanggal;
} today;
```

Potongan program yang baru saja dijalankan adalah contoh deklarasi structure (**data\_tanggal**) bersamaan dengan deklarasi variabel structure (**today**). Variabel structure yang dideklarasikan secara terpisah dapat dilakukan dengan cara berikut.

```

struct data_tanggal
{
    int tahun;
    int bulan;
    int tanggal;
};
struct data_tanggal today;

```

Ketik ulang potongan program di atas lalu simpan dengan nama **praktikum3\_1b.c**. Pastikan tidak ada error yang muncul saat program dijalankan.

Kita juga dapat mendeklarasikan structure di luar fungsi **main()**, dan variabel structurenya dideklarasikan di dalam fungsi **main()**, seperti yang ditunjukkan pada potongan program di bawah ini. Ketik ulang potongan program tersebut, lalu simpan dengan nama **praktikum3\_1c.c**. Pastikan tidak ada error yang muncul saat program dijalankan.

```

struct data_tanggal
{
    int tahun;
    int bulan;
    int tanggal;
};

int main()
{
    struct data_tanggal today;
    return 0;
}

```

2. Kita juga dapat menggunakan **typedef** pada structure. Kata kunci **typedef** adalah kata kunci untuk mendefinisikan tipe data baru. Kita bisa menggunakan kata kunci ini di depan **struct** untuk menyatakannya sebagai tipe data baru. Sebagai contoh, pertama-tama ketik ulang potongan program di bawah ini lalu simpan dengan nama **praktikum3\_2.c**.

```

// membuat struct
struct Distance{
    int feet;
}

```

```

    float inch;
};

void main() {
    // menggunakan struct
    struct Distance d1, d2;
}

```

Tanpa **typedef**, kita akan menggunakan struct dengan cara seperti itu. Jika menggunakan **typedef**, maka penggunaan struct akan menjadi seperti ini:

```

// membuat struct dengan typedef
typedef struct Distance{
    int feet;
    float inch;
} distances;

void main() {
    // menggunakan struct
    distances dist1, dist2, sum;
}

```

Perhatikan bedanya, lalu modifikasi potongan program yang telah Anda simpan pada file **praktikum3\_2.c**. Simpan modifikasi yang Anda buat, lalu jalankan **praktikum3\_2.c**. Pastikan tidak ada error yang muncul.

3. Setelah berhasil mendeklarasikan structure, kita dapat menginisialisasi nilai-nilai elemen yang terdapat di dalamnya. Anggota struktur dapat diinisialisasi dengan menggunakan kurung kurawal '{ }'. Kita akan mempraktikkan beberapa cara dalam menginisialisasi elemen structure, menggunakan file-file praktikum pada poin 1.

Cara yang pertama untuk variabel structure yang dideklarasikan bersamaan dengan deklarasi structure. Pada **praktikum3\_1a.c**, baris terakhir program dapat dimodifikasi menjadi seperti berikut ini:

```

} today = {1998, 7, 24};

```

Simpan **praktikum3\_1a.c** yang telah ditambahkan potongan program di atas, lalu coba jalankan.

Cara yang kedua, bila variabel structure dideklarasikan secara terpisah dari deklarasi structure, maka inisialisasi nilai-nilai elemennya dapat dilakukan seperti ini:

```
struct data_tanggal today = {1998,7,28};
```

Modifikasi potongan program **praktikum3\_1b.c** dan **praktikum3\_1c.c** mengikuti cara inisialisasi yang kedua ini. Pastikan tidak ada error saat program dijalankan kembali.

4. Walaupun elemen-elemen di dalam structure berada dalam satu kesatuan, masing-masing elemen tersebut tetap dapat diakses secara individual. Untuk mengakses elemen-elemen pada structure, gunakan operator titik (.). Berikut ini adalah contoh dalam mengakses elemen-elemen pada suatu struct. Ketik ulang potongan program di bawah ini, modifikasi bagian-bagian yang ditandai dengan komentar.

```
#include<stdio.h>

struct data_tanggal
{
    int tahun;
    int bulan;
    int tanggal;
}ultah;

int main()
{
    //menginisialisasi elemen-elemen struct ultah
    ultah.tanggal = 28; //ganti dengan tanggal lahir
    Anda
    ultah.bulan = 7;    //ganti dengan bulan lahir Anda
    ultah.tahun = 1998; //ganti dengan tahun lahir Anda

    //mengakses elemen-elemen struct ultah
    printf ("tanggal = %d, bulan = %d, tahun = %d"
           ,ultah.tanggal, ultah.bulan, ultah.tahun);
    return 0;
}
```

Simpan program dengan nama **praktikum3\_4.c**, lalu jalankan. Bila tidak ada error dan program berhasil dijalankan, artinya kita sudah bisa menginisialisasi sekaligus mengakses elemen-elemen yang terdapat pada structure. Anda juga dapat menambahkan elemen-elemen lain pada structure yang terdapat pada **praktikum3\_4.c**.

Berikut ini adalah contoh-contoh program menggunakan structure. Silakan Anda ketik ulang pada IDE Anda, lalu cobalah untuk memodifikasi elemen-elemen yang terdapat pada structure pada contoh-contoh program tersebut.

5. Pertama, contoh program yang menerapkan konsep struktur untuk buku yang terdiri dari elemen: judul, pengarang dan id, simpan dengan nama **praktikum3\_5.c**.

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

struct Buku {
    char  judul[50];
    char  pengarang[50];
    int   id;
};

int main( ) {
    struct Buku Buku1;
    struct Buku Buku2;

    /* Spesifikasi Buku 1 */
    strcpy( Buku1.judul, "C Programming");
    strcpy( Buku1.pengarang, "Nuha Ali");
    Buku1.id = 6495407;

    /* Spesifikasi Buku 2 */
    strcpy( Buku2.judul, "Telecom Billing");
    strcpy( Buku2.pengarang, "Zara Ali");
    Buku2.id = 6495700;

    /* Cetak informasi Buku 1 */
    printf( "Judul Buku 1   : %s\n", Buku1.judul);
    printf( "Pengarang Buku 1 : %s\n",
Buku1.pengarang);
    printf( "Id Buku 1   : %d\n\n", Buku1.id);
```

```

        /* Cetak informasi Buku 2 */
        printf( "Judul Buku 2   : %s\n", Buku2.judul);
        printf( "Pengarang Buku 2   : %s\n",
Buku2.pengarang);
        printf( "Id Buku 2 : %d\n", Buku2.id);

        return 0;
    }

```

Jalankan program tersebut. Apa output yang didapatkan?

6. Contoh berikutnya adalah Struct Bersarang. Struct dapat dibuat bersarang (*nested*), yang artinya ada struct di dalam struct.

```

struct complex
{
    int imag;
    float real;
};

struct number
{
    struct complex comp;
    int integers;
} num1, num2;

```

Cara menggunakannya adalah seperti ini:

```

num1.integers = 12;
num1.comp.real = 44.12;
num2.comp.imag = 11;

```

Satukan kedua potongan program di atas dan simpan dengan nama **praktikum3\_6.c**. Modifikasi **praktikum3\_6.c** dengan menambahkan potongan program untuk menampilkan nilai setiap elemen ke layar. Lalu coba jalankan.

7. Structure juga dapat kita buat sebagai parameter untuk fungsi. Contoh:

```

#include <stdio.h>

struct student
{

```



```

    char name[50];
    int age;
};

void main() {
    struct student s1;
    printf("Enter name: ");
    gets(s1.name);
    printf("Enter age: ");
    scanf("%d", &s1.age);
    display(s1);    // passing structure as an argument
}

// membuat fungsi dengan struct sebagai parameter
void display(struct student s) {
    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);
}

```

Ketik ulang potongan program tersebut, lalu simpan dengan nama **praktikum3\_7.c**. Jalankan program tersebut. Apa output yang didapatkan?

## B. Pointer

Setiap variabel yang kita buat pada program akan memiliki alamat memori. Alamat memori berfungsi untuk menentukan lokasi penyimpanan data pada memori (RAM). Kadang alamat memori ini disebut *reference* atau referensi. Untuk melihat alamat memori yang digunakan pada variabel, kita bisa pakai simbol **&** (referensi). Tak perlu jauh-jauh, kita sudah tahu tentang fungsi **scanf()**, yaitu fungsi yang meminta input dari pengguna. Contoh:

```
scanf("%d", &a);
```

Pada contoh tersebut, terdapat '**&a**' yang artinya program akan menyimpan nilai input ke dalam alamat memori **a**. Alamat memori memiliki format heksadesimal (hex), yaitu sistem bilangan yang memiliki 16 simbol dalam penomoran. Contoh untuk mencetak alamat dari memori **a**:

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

```
int main() {
    int a;
    printf("%p\n", &a);
}
```

Hasil keluaran (Hasil bisa berbeda setiap saat dan berbeda antara perangkat satu dengan lainnya):

```
0x7ffef6dec9ec
```

Lalu apa hubungannya alamat memori dengan pointer? Pointer adalah sebuah **variabel yang berisi alamat memori** dari variabel yang lain. Setelah mengetahui cara mengambil alamat sebuah variabel, sekarang kita harus menggunakan pointer untuk menyimpannya. Pointer juga bisa mengakses data yang ada di suatu alamat memori.

8. Meskipun berbeda dari variabel umumnya, namun cara mendeklarasikan pointer hampir sama dengan yang lain. Deklarasi pointer:

```
tipe_data *nama_variabel;
```

Karena pointer menampung alamat memori, maka dalam melakukan inisialisasi juga harus memberikan alamat memori. Untuk memahami deklarasi dan inisialisasi pointer, ketik ulang potongan program di bawah ini.

```
#include <stdio.h>
int main() {
    int a = 5;
    int *p;
    p = &a;
    printf("%p\n", p);
}
```

Simpan program dengan nama **praktikum3\_8.c**. Setelah program dijalankan, apakah output yang didapatkan? Pada contoh tersebut, saat memberikan nilai pada variabel pointer **p**, tak perlu menambahkan \*. Kemudian, untuk mencetaknya tak perlu menggunakan &, karena **p** merupakan variabel pointer.

Operator dereferensi (\*) merupakan kebalikan dari operator referensi (&) karena operator ini akan menunjuk nilai yang berada di alamat memori. Modifikasi baris sebelum kurung penutup pada **praktikum3\_8.c**, menjadi seperti berikut:

```
printf("%d\n", *p);
```

Simpan ulang program, kemudian jalankan. Maka output yang didapatkan akan berbeda dengan sebelumnya. **p** merupakan pointer yang menyimpan alamat dari **a**, sehingga **\*p** berisi nilai dari **a**.

9. Agar lebih mengerti mengenai penggunaan pointer, cobalah untuk menjalankan dan memodifikasi contoh yang menggunakan pointer berikut ini. Buatlah file baru dengan nama **praktikum3\_9.c**, kemudian isi dengan kode berikut:

```
#include <stdio.h>

void main(){
    // membuat variabel
    int umur = 19;
    float tinggi = 175.6;

    // membuat pointer
    int *pointer_umur = &umur;
    int *pointer2 = &umur;
    float *p_tinggi = &tinggi;

    // mencetak alamat memori variabel
    printf("alamat memori variabel 'umur' = %d\n",
&umur);
    printf("alamat memori variabel 'tinggi' = %d\n",
&tinggi);
    // mencetak referensi alamat memori pointer
    printf("referensi alamat memori *pointer_umur =
%d\n", pointer_umur);
    printf("referensi alamat memori *pointer2 = %d\n",
pointer2);
    printf("referensi alamat memori *p_tinggi = %d\n",
p_tinggi);
}
```

Setelah itu, jalankan program. Perhatikan hasilnya. Alamat memori yang digunakan sebagai referensi pada pointer akan sama dengan alamat memori dari variabel yang kita gunakan sebagai referensi.

10. Pointer juga memiliki alamat memorinya sendiri. Sama seperti variabel biasa, jika ingin melihat alamat memori dari pointer, maka kita harus menggunakan simbol **&**. Sebagai contoh, kita masih menggunakan program yang tersimpan pada **praktikum3\_9.c** Untuk melihat alamat pointer-pointer pada program tersebut, tambahkan potongan program di bawah ini:

```
// mencetak alamat memori pointer
```

```

    printf("alamat memori *pointer_umur = %d\n",
&pointer_umur);
    printf("alamat memori *pointer2 = %d\n",
&pointer2);
    printf("alamat memori *p_tinggi = %d\n",
&p_tinggi);

```

Jalankan kembali **praktikum3\_9.c**. Maka akan kita dapatkan alamat memori dari pointer-pointer yang ada pada program tersebut.

11. Kita sudah tahu bahwa pointer menyimpan alamat memori atau "menunjuk " sebuah variabel. Pointer juga dapat menyimpan alamat memori atau menunjuk pointer lain (*pointer to pointer*). Jadi, ketika kita mendefinisikan pointer ke pointer. Pointer pertama digunakan untuk menyimpan alamat variabel. Dan pointer kedua digunakan untuk menyimpan alamat pointer pertama. Itu sebabnya mereka juga dikenal sebagai pointer ganda. Bagaimana cara mendeklarasikan pointer ke pointer di C? Mari kita memahami dengan bantuan program di bawah ini. Simpan program dengan nama **praktikum3\_10.c**.

```

#include <stdio.h>
int main()
{
    int var = 789;
    int *ptr2; // pointer for var
    int **ptr1; // double pointer for ptr2
    ptr2 = &var; // storing address of var in ptr2
    ptr1 = &ptr2; // Storing address of ptr2 in ptr1

    // Displaying value of var using single and double
    pointers
    printf("Value of var = %d\n", var );
    printf("Value of var using single pointer = %d\n",
*ptr2 );
    printf("Value of var using double pointer = %d\n",
**ptr1);

    return 0;
}

```

Coba jalankan **praktikum3\_10.c**. Dari program tersebut dapat kita pahami bahwa mendeklarasikan Pointer ke Pointer mirip dengan mendeklarasikan pointer di C.

Perbedaannya adalah kita harus menempatkan tambahan '\*' sebelum nama pointer.

12. Berikutnya kita akan mencoba menggunakan pointer untuk melakukan passing argumen berdasarkan referensinya (*pass by reference*). Pertama-tama ketik ulang program di bawah ini, lalu simpan dengan nama **praktikum3\_12.c**.

```
#include <stdio.h>

void add_score(int score){
    score = score + 5;
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(score);
    printf("score setelah diubah: %d\n", score);
}
```

Pada program ini, kita membuat fungsi dengan nama **add\_score()** untuk menambahkan nilai score sebanyak 5. Tapi ketika dijalankan, nilai variabel score tidak berubah, ia tetap bernilai 0. Hal ini karena variabel **score** dibuat di dalam fungsi **main()**, lalu ketika fungsi **add\_score()** mencoba mengubah nilainya, maka perubahan hanya terjadi secara lokal di dalam fungsi **add\_score()** saja. Hal ini dapat dibuktikan dengan menambahkan potongan program berikut ke dalam fungsi **add\_score()**:

```
printf("Score diubah ke %d\n", score);
```

Jalankan ulang **praktikum3\_11.c** Dapat dilihat bahwa nilai **score** pada fungsi **add\_score()** sudah berubah menjadi 5, namun variabel **score** pada fungsi **main()** akan tetap bernilai 0. Permasalah ini salah satunya dapat diselesaikan dengan menggunakan pointer untuk melakukan *pass-by-reference*.

Sekarang, coba ubah kode programnya menjadi seperti ini:

```
#include <stdio.h>

void add_score(int *score){
    *score = *score + 5;
```

```

        printf("score diubah ke: %d\n", *score);
    }

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(&score);
    printf("score setelah diubah: %d\n", score);
}

```

Karena argumen fungsi **add\_score()** kita ubah menjadi pointer, maka kita harus memberikan alamat memori saat memanggilnya. Sekarang, setiap fungsi **add\_score()** dipanggil atau dieksekusi, maka nilai variabel score akan bertambah 5.

13. Pointer juga sering digunakan untuk mengakses elemen array, seperti yang ditunjukkan pada program di bawah ini. Ketik ulang program tersebut, lalu simpan dengan nama **praktikum3\_13.c**.

```

#include <stdio.h>

void main(){
    printf("## Program Antrian CS ##\n");

    char no_antrian[5] = {'A', 'B', 'C', 'D', 'E'};

    // menggunakan pointer
    char *ptr_current = &no_antrian;

    for(int i = 0; i < 5; i++){
        printf("🗨 Pelanggan dengan no antrian %c\n", *ptr_current);
        printf("Saat ini CS sedang melayani: %c\n", *ptr_current);
        printf("----- Tekan Enter untuk Next -----");
        getchar();
        ptr_current++;
    }
}

```

```
    }

    printf("✔ Selesai");
}
```

Pada program ini, kita menggunakan **ptr\_current** untuk mengakses elemen array. Saat pertama kali dibuat, pointer **ptr\_current** akan mereferensi pada elemen pertama array. Lalu pada perulangan, dilakukan increment **ptr\_current++**, maka pointer ini akan mereferensi ke elemen array selanjutnya. Untuk memahami hal ini, jalankan program **praktikum3\_12.c**, dan pelajari hasil yang dikeluarkan oleh program.

### 3.5 Penugasan

1. Buatlah contoh program sederhana yang memuat elemen array di dalam structure DAN sekaligus memuat structure di dalam array.
2. Buatlah contoh program sederhana yang memuat sebuah structure dengan elemen pointer di dalamnya.
3. Buatlah contoh program sederhana yang memuat sebuah pointer yang menunjuk sebuah structure.

## MODUL 4: SINGLE LINKED LIST

---

### 4.1 Deskripsi Singkat

Struktur data, dilihat dari alokasi memori yang digunakan, dibagi menjadi 2 yaitu dinamis dan statis. Pada struktur statis, elemen data bersifat tetap/fixed, artinya ukuran dan urutannya sudah pasti. Contoh struktur statis adalah array. Alokasi memori pada array akan terbuang jika array tidak diisi penuh, dan bermasalah ketika harus menambah ukuran yang ditetapkan di awal. Selain itu, ruang memori yang dipakai oleh array tidak dapat dihapus bila array tersebut sudah tidak digunakan lagi pada saat program dijalankan. Untuk memecahkan masalah di atas, kita dapat menggunakan linked list. Linked List merupakan jenis struktur data yang dinamis. Struktur dinamis memungkinkan jumlah elemen data dapat berubah-ubah (tambah atau kurang) pada saat program dijalankan.

Linked List adalah struktur berupa rangkaian objek saling berkait (linked). Masing-masing objek sering disebut dengan **simpul** atau **node** atau **verteks**. Objek itu sendiri adalah merupakan gabungan beberapa elemen data (variabel) yang dijadikan satu kelompok dalam bentuk **structure**. Untuk menghubungkan objek satu dengan objek lainnya, diperlukan paling tidak sebuah variabel **pointer**. Penggunaan pointer berakibat objek-objek bersebelahan secara logic walaupun tidak bersebelahan secara fisik di memori. Terdapat beberapa jenis linked list, yaitu: **Linear Single Linked List**, **Linear Double Linked List**, **Circular Single Linked List**, dan **Circular Double Linked List**. Pada praktikum ini, kita akan mencoba mempraktikkan pembuatan dan operasi-operasi pada Linear Single Linked List.

Linear Single Linked List merupakan Linked List yang paling sederhana. Setiap simpul dibagi menjadi dua bagian yaitu bagian isi dan bagian pointer. Bagian isi merupakan bagian yang berisi data yang disimpan oleh simpul, sedangkan bagian pointer merupakan bagian yang berisi alamat dari simpul berikutnya.

### 4.2 Tujuan Praktikum

- 1) Mahasiswa mampu membuat Linear Single Linked List menggunakan bahasa C
- 2) Mahasiswa dapat melakukan operasi penyisipan maupun penghapusan simpul pada Linear Single Linked List dengan menggunakan bahasa C

### 4.3 Material Praktikum

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

### 4.4 Kegiatan Praktikum

Pada penggunaan Linked List, ada 4 macam proses dasar, yaitu:

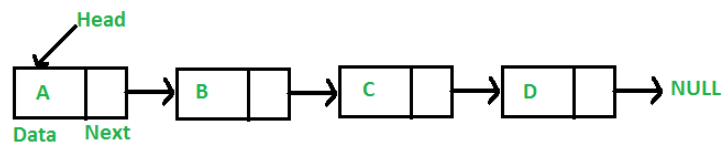


1. Inisialisasi
2. Membuat simpul awal
3. Membuat simpul baru dan menambahkannya ke dalam Linked List
4. Menghapus simpul dari Linked List

## A. Inisialisasi

### 1. Persiapan

Untuk pembuatan linked list, kita memerlukan suatu structure yang nantinya akan digunakan untuk pembuatan simpul. Dalam potongan program di bawah ini, kita deklarasikan structure dengan nama **node**, yang terdiri dari dua elemen. Elemen pertama yaitu **value**, yang berfungsi untuk menyimpan data dengan tipe integer. Elemen kedua adalah **next**, yang bertipe pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** lainnya. Selain pointer **next** yang merupakan bagian dari structure **node**, kita memerlukan pointer lainnya untuk menunjuk simpul pada linked list yang akan kita buat. Perhatikan ilustrasi Linear Single Linked List berikut.



Dari ilustrasi di atas, terdapat pointer **Head** yang menunjuk simpul paling kiri atau simpul awal. Selain pointer **Head**, kita akan memerlukan pointer-pointer lain untuk membantu dalam mengoperasikan Linked List. Oleh karena itu, kita juga mendeklarasikan sebuah pointer dengan tipe data **struct node**, dalam hal ini dimisalkan dengan nama **ptrnode**.

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

struct node{
    int value;
    struct node *next;
};

typedef struct node *ptrnode;
```

Salin potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum4.c**. Tambahkan fungsi **main** pada potongan program tersebut, lalu jalankan.

## 2. Pembuatan simpul awal

Untuk menginisialisasi linked list, kita siapkan simpul paling kiri atau simpul awal atau head. Tambahkan potongan program berikut pada fungsi **main()**. Lalu jalankan ulang

```
ptrnode head = NULL;
head = (ptrnode)malloc(sizeof(struct node));
head->value = 10;
head->next = NULL;
```

### B. Pembuatan Sebuah Simpul

Linked list adalah kumpulan dari simpul-simpul. Simpul-simpul tersebut dibuat satu per satu, tidak sekaligus. Selain simpul awal, maka setiap simpul yang baru dibuat, harus dihubungkan (linked) dengan salah satu simpul yang sudah ada. Terdapat beberapa cara untuk membuat simpul baru. Cara pertama, kita mendeklarasikan **node** terlebih dahulu. Baru kemudian kita mendeklarasikan **pointer**, dimana pointer diisi dengan alamat **node** yang telah sebelumnya dideklarasikan. Untuk mencoba cara ini, tambahkan potongan program berikut pada fungsi **main()**.

```
struct node node_dua;
ptrnode dua = &node_dua;
dua->value = 20;
dua->next = NULL;
```

Agar simpul kedua yang baru saja kita buat tersambung (linked) dengan simpul head yang sebelumnya telah kita buat, maka tambahkan potongan program berikut, masih di dalam fungsi **main()**.

```
head->next = dua;
```

Cara pembuatan simpul yang kedua, sama seperti pembuatan simpul **head** pada tahapan inisialisasi. Kita deklarasikan pointer terlebih dahulu, baru kemudian kita deklarasikan **struct node**, sekaligus mengalokasikan memori untuk **struct node** tersebut dan menyimpannya ke dalam pointer yang telah dideklarasikan. Tambahkan potongan program berikut pada fungsi **main()**. Pada potongan program ini, kita akan membuat dua simpul baru, dan menghubungkannya dengan kedua simpul yang telah dibuat sebelumnya.

```
ptrnode tiga = NULL;
ptrnode empat = NULL;
```

```

tiga = (ptrnode)malloc(sizeof(struct node));
empat = (ptrnode)malloc(sizeof(struct node));

dua->next = tiga;
tiga->value = 30;
tiga->next = empat;

empat->value = 40;
empat->next = NULL;

```

Cara ketiga, agar tidak perlu menuliskan potongan program di atas secara berulang-ulang setiap kali dibutuhkan untuk membuat node baru, maka potongan program untuk membuat simpul baru tersebut kita simpan di dalam sebuah fungsi. Tambahkan potongan program berikut di luar fungsi **main()**, tepatnya setelah deklarasi **struct node** dan **pointer \*ptrnode**.

```

ptrnode createNode(int nilai){
ptrnode p;
p = (ptrnode)malloc(sizeof(struct node));
p->value = nilai;
p->next = NULL ;
return(p);
}

```

Untuk membuat simpul baru, kita tinggal memanggil fungsi **createNode** tersebut di dalam fungsi **main()**. Sebagai contoh, tambahkan potongan program berikut pada fungsi **main()**.

```

ptrnode lima = createNode(50);
empat->next = lima;

```

### C. Menampilkan Nilai dari Linked List

Sebelum kita lanjutkan ke materi praktikum berikutnya, cobalah untuk menampilkan nilai dari simpul-simpul yang telah kita buat pada Kegiatan Praktikum A dan B. Algoritma untuk menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Linked List adalah sebagai berikut:

1. Iterasi setiap node dalam sebuah linked list. Gunakan pointer bantuan untuk memeriksa apakah terdapat **node** ataukah **NULL**.

2. Lakukan **printf()** elemen data dari setiap node mulai dari head, node berikutnya, dan seterusnya sampai node tersebut NULL.

Apakah Anda berhasil menampilkan seluruh nilai pada simpul pertama hingga simpul kelima?

#### D. Penambahan Simpul ke Dalam Linked List

1. Insert kiri, maksudnya menambahkan sebuah simpul baru diujung simpul paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **insert\_head()** berikut di luar fungsi **main()**.

```
ptrnode insert_head(ptrnode head, int nilai){
    ptrnode new_node = createNode(nilai);
    new_node->next = head;
    head = new_node;

    return(head);
}
```

Kemudian coba panggil fungsi **insert\_head()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Tambahkan node baru dengan value = 99 sebagai head */
head = insert_head(head, 99);
```

2. Insert kanan, maksudnya menambahkan sebuah simpul baru diujung simpul paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **insert\_tail()** berikut di luar fungsi **main()**.

```
ptrnode insert_tail(ptrnode head, int nilai){
    /* iterasi mencari node terakhir*/
    ptrnode tail = head;
    while(tail->next != NULL)
        tail = tail->next;

    /* buat node baru */
    ptrnode new_node = createNode(nilai);
    tail->next = new_node;
```

```

    return(head);
}

```

Kemudian coba panggil fungsi **insert\_tail()** dari fungsi **main()** dengan cara seperti di bawah ini.

```

/* Tambahkan node baru dengan value = 11 sebagai tail */
head = insert_tail(head, 11);

```

3. Menambahkan simpul baru setelah simpul tertentu. Tambahkan potongan program berikut di luar fungsi **main()**.

```

ptrnode insert_after(ptrnode head, int nilai, int
prev_nilai){
    /* cari node sebelumnya, starting from the first
node*/
    ptrnode cursor = head;
    while(cursor->value != prev_nilai)
        cursor = cursor->next;

    ptrnode new_node = createNode(nilai);
    new_node->next = cursor->next;
    cursor->next = new_node;

    return(head);
}

```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```

/* Tambahkan node baru dengan value = 60 setelah node
dengan value 50 */
head = insert_after(head, 60, 50)

```

4. Menambahkan simpul baru sebelum simpul tertentu. Tambahkan potongan program berikut di luar fungsi **main()**.

```

ptrnode insert_before(ptrnode head, int nilai, int
next_nilai){
    if (head->value == next_nilai)
        head = insert_head(head, nilai);
}

```

```

else
{
    ptrnode cursor, prevcursor;
    cursor = head;
    do
    { prevcursor = cursor;
      cursor = cursor->next;
    }
    while (cursor->value != next_nilai);

    ptrnode new_node = createNode(nilai);
    new_node->next = cursor;
    prevcursor->next = new_node;
}
return(head);
}

```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```

/* Tambahkan node baru dengan value = 35 sebelum node
dengan value 99 */
head = insert_before(head, 35, 40);

```

Untuk melihat kembali isi keseluruhan simpul setelah beberapa penambahan simpul yang telah kita lakukan, pada fungsi **main()** panggil kembali fungsi untuk menampilkan nilai dari Linked List yang telah Anda buat sebelumnya pada Kegiatan Praktikum C.

## E. Penghapusan Simpul dari Linked List

1. Delete kiri atau delete awal, maksudnya adalah menghapus simpul yang ada di ujung paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **remove\_first()** berikut di luar fungsi **main()**.

```

ptrnode remove_first(ptrnode head){
    if(head == NULL)
        return;

    ptrnode first = head;
    head = head->next;
}

```

```

    first->next = NULL;

    free(first);

    return(head);
}

```

Kemudian coba panggil fungsi **remove\_first()** dari fungsi **main()** dengan cara seperti di bawah ini.

```

/* Hapus node head */
head = remove_first(head);

```

2. Delete kanan atau delete akhir, maksudnya adalah menghapus simpul yang ada di ujung paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **remove\_last()** berikut di luar fungsi **main()**.

```

ptrnode remove_last(ptrnode head) {
    if(head == NULL)
        return;

    ptrnode tail = head;
    ptrnode prevtail = NULL;
    while(tail->next != NULL)
    {
        prevtail = tail;
        tail = tail->next;
    }

    prevtail->next = NULL;
    free(tail);
    return(head);
}

```

Kemudian coba panggil fungsi **remove\_last()** dari fungsi **main()** dengan cara seperti di bawah ini.

```

/* Hapus node tail */
head = remove_last(head);

```

3. Delete tengah, maksudnya yaitu menghapus sebuah simpul yang berada di antara dua buah simpul lain. Bukan menghapus simpul yang paling kiri dan juga bukan menghapus simpul yang paling kanan. Tambahkan potongan program berikut di luar fungsi **main()**.

```
ptrnode remove_middle(ptrnode head, int nilai){
    ptrnode cursor = head;
    while(cursor != NULL)
    {
        if(cursor->next->value == nilai)
            break; //keluar dari iterasi
        cursor = cursor->next;
    }

    if(cursor != NULL)
    {
        ptrnode tmp = cursor->next;
        cursor->next = tmp->next;
        tmp->next = NULL;
        free(tmp);
    }

    return(head);
}
```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```
/* Hapus node di tengah */
head = remove_middle(head, 30);
```

4. Menggunakan salah satu cara penghapusan simpul di atas, kita dapat menghapus keseluruhan simpul yang ada pada Linked List. Sebagai contoh, program di bawah ini menjalankan algoritma yang sama dengan fungsi **remove\_first()**, hanya ditambahkan iterasi agar proses penghapusan dilakukan secara berulang-ulang untuk keseluruhan simpul yang ada pada Linked List. Tambahkan fungsi **dispose()** berikut di luar fungsi **main()**.

```
ptrnode dispose(ptrnode head)
{
```



```

    if(head == NULL)
        return;

    while(head != NULL){
        ptrnode tmp = head;
        head = head->next;

        tmp->next = NULL;
        free(tmp);
    }

    printf("semua node telah dihapus\n");
    return(head);
}

```

Kemudian coba panggil fungsi **dispose()** dari fungsi **main()** dengan cara seperti di bawah ini.

```

/* Hapus/free linked list */
head = dispose(&head);

```

Setelah fungsi-fungsi hapus di atas dijalankan, panggil kembali fungsi untuk menampilkan nilai linked list pada fungsi **main()**, untuk melihat apa isi yang tersisa pada Linked List.

#### 4.5 Penugasan

1. Buatlah fungsi untuk menghitung jumlah node dalam sebuah linked list! (looping sama seperti pada saat menampilkan nilai dari linked list).
2. Buatlah fungsi untuk membalik nilai dari head ke tail!  
Contoh: 5->4->3->2->1 menjadi 1->2->3->4->5  
Hint:
  - hanya nilai saja, memory address (pointer node) tetap sama
  - buat temporary pointer node sebagai bantuan: prev, current, next dan loop dari head ke tail
3. Buat program untuk menyimpan data students berisi **int nim**, **char nama[50]** secara dinamis!

## MODUL 5: DOUBLE LINKED LIST

### 5.1 Deskripsi Singkat

Pada praktikum ini, kita akan mencoba mempraktikkan pembuatan dan operasi-operasi pada Linear Double Linked List. Linear Double Linked List adalah Linked List lurus dengan pointer ganda, yaitu ada dua buah pointer. Jadi, dalam structure simpul, terdapat dua elemen yang bertipe pointer. Pointer pertama menunjuk atau berisi alamat simpul setelahnya (next node), dan pointer yang kedua menunjuk atau berisi alamat simpul sebelumnya (previous node).

### 5.2 Tujuan Praktikum

- 1) Mahasiswa mampu membuat Linear Double Linked List menggunakan bahasa C
- 2) Mahasiswa dapat melakukan operasi penyisipan maupun penghapusan simpul pada Linear Double Linked List dengan menggunakan bahasa C

### 5.3 Material Praktikum

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

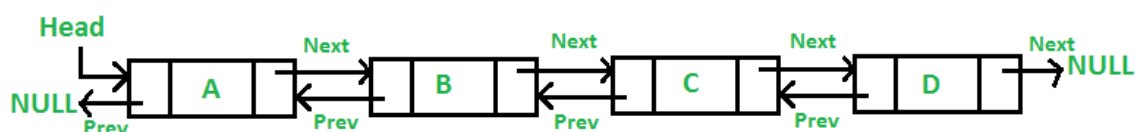
### 5.4 Kegiatan Praktikum

Jenis dan macam proses pada Linear Double Linked List sama saja dengan Linear Single Linked List yang telah kita coba pada praktikum sebelumnya, sehingga pengertian dari tiap-tiap proses dapat dibaca pada bahan praktikum sebelumnya.

#### A. Inisialisasi

##### 1. Persiapan

Kita deklarasikan structure dengan nama **node**, yang terdiri dari **tiga** elemen. Elemen pertama yaitu value, yang berfungsi untuk menyimpan data dengan tipe integer. Elemen kedua adalah **prev**, yaitu pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** sebelumnya. Elemen ketiga adalah **next**, yaitu pointer yang dipakai untuk menunjuk (mencatat alamat) structure **node** setelahnya.



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

struct node{
    int value;
    struct node *next;
    struct node *prev;
};

typedef struct node *ptrnode;
```

Salin potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum5.c**. Tambahkan fungsi **main** pada potongan program tersebut, lalu jalankan.

## 2. Pembuatan simpul awal

Untuk menginisialisasi linked list, kita siapkan simpul paling kiri atau simpul awal atau head. Tambahkan potongan program berikut pada fungsi **main()**. Lalu jalankan ulang

```
ptrnode head = NULL;
ptrnode tail = NULL;
head = (ptrnode)malloc(sizeof(struct node));
tail = head;
head->value = 10;
head->next = NULL;
head->prev = NULL;
```

## B. Pembuatan Sebuah Simpul

Cara pertama, deklarasikan **node** terlebih dahulu, baru kemudian deklarasikan **pointer**, dimana pointer diisi dengan alamat **node** yang telah sebelumnya dideklarasikan. Untuk mencoba cara ini, tambahkan potongan program berikut pada fungsi **main()**.

```
struct node node_dua;
ptrnode dua = &node_dua;
dua->value = 20;
dua->next = NULL;
```

```
dua->prev = NULL;
```

Agar simpul kedua yang baru saja kita buat tersambung (linked) dengan simpul head yang sebelumnya telah kita buat, maka tambahkan potongan program berikut, masih di dalam fungsi **main()**.

```
head->next = dua;  
dua->prev = head;
```

Cara pembuatan simpul yang kedua, deklarasikan pointer terlebih dahulu, baru kemudian deklarasikan **struct node**, sekaligus mengalokasikan memori untuk **struct node** tersebut dan menyimpannya ke dalam pointer yang telah dideklarasikan.

```
ptrnode tiga = NULL;  
ptrnode empat = NULL;  
  
tiga = (ptrnode)malloc(sizeof(struct node));  
empat = (ptrnode)malloc(sizeof(struct node));  
  
dua->next = tiga;  
  
tiga->value = 30;  
tiga->next = empat;  
tiga->prev = dua;  
  
empat->value = 40;  
empat->next = NULL;  
empat->prev = tiga;
```

Agar tidak perlu menuliskan potongan program di atas secara berulang-ulang setiap kali dibutuhkan untuk membuat node baru, maka potongan program untuk membuat simpul baru tersebut kita simpan di dalam sebuah fungsi, dalam hal ini kita beri nama **createNode()**.

```
ptrnode createNode(int nilai){  
    ptrnode p;  
    p = (ptrnode)malloc(sizeof(struct node));  
    p->value = nilai;  
    p->next = NULL;  
    p->prev = NULL;
```

```
return(p);  
}
```

Untuk membuat simpul baru, kita tinggal memanggil fungsi **createNode** tersebut di dalam fungsi **main()**. Sebagai contoh, tambahkan potongan program berikut pada fungsi **main()**.

```
ptrnode lima = createNode(50);  
empat->next = lima;  
lima->prev = empat;
```

### C. Menampilkan Nilai dari Linked List

- Apakah terdapat perbedaan algoritma untuk menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Single Linked List dan Double Linked List?
- Bagaimana cara menampilkan nilai dari seluruh simpul yang terdapat pada sebuah Double Linked List dimulai dari node terakhir atau tail (*backward traversal*)?

### D. Penambahan Simpul ke Dalam Linked List

1. Insert kiri, maksudnya menambahkan sebuah simpul baru diujung simpul paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **insert\_head()** berikut di luar fungsi **main()**.

```
ptrnode insert_head(ptrnode head, int nilai){  
    ptrnode new_node = createNode(nilai);  
    new_node->next = head;  
    head->prev = new_node;  
    head = new_node;  
  
    return(head);  
}
```

Kemudian coba panggil fungsi **insert\_head()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Tambahkan node baru dengan value = 99 sebagai head */  
  
head = insert_head(head, 99);
```

2. Insert kanan, maksudnya menambahkan sebuah simpul baru diujung simpul paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **insert\_tail()** berikut di luar fungsi **main()**.

```
ptrnode insert_tail(ptrnode head, int nilai){
    /* iterasi mencari node terakhir*/
    ptrnode tail = head;
    while(tail->next != NULL)
        tail = tail->next;

    /* buat node baru */
    ptrnode new_node = createNode(nilai);
    tail->next = new_node;

    return(head);
}
```

Kemudian coba panggil fungsi **insert\_tail()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Tambahkan node baru dengan value = 11 sebagai tail
*/
head = insert_tail(head, 11);
```

#### **PERHATIAN!!**

Informasi simpul akhir atau tail akan sering dibutuhkan dalam penggunaan Double Linked List. Sebagaimana **pointer head**, Anda juga dapat mendeklarasikan **pointer tail** saat pembuatan simpul awal. Untuk memperbaharui posisi **pointer tail** tersebut, dapat dituliskan potongan program tambahan pada fungsi **insert kanan** dan **delete kanan**. Dengan demikian, **pointer tail** dapat langsung digunakan setiap kali dibutuhkan dan program tidak perlu melakukan iterasi untuk mencari simpul akhir atau tail.

3. Menambahkan simpul baru setelah simpul tertentu. Tambahkan potongan program berikut di luar fungsi **main()**.

```
ptrnode insert_after(ptrnode head, int nilai, int
nilai_dicari){
    /* cari node sebelumnya, starting from the first
node*/
    ptrnode cursor = head;
    while(cursor->value != nilai_dicari)
        cursor = cursor->next;
```

```

ptrnode new_node = createNode(nilai);
new_node->next = cursor->next;
cursor->next->prev = new_node;
new_node->prev = cursor;
cursor->next = new_node;

return(head);
}

```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```

/* Tambahkan node baru dengan value = 60 setelah node
dengan value 50 */
head = insert_after(head, 60, 50);

```

4. Menambahkan simpul baru sebelum simpul tertentu. Tambahkan potongan program berikut di luar fungsi **main()**.

```

ptrnode insert_before(ptrnode head, int nilai, int
next_nilai){
    if (head->value == next_nilai)
        head = insert_head(head, nilai);
    else
    {

//pencarian nilai sama seperti insert after, tidak
perlu 2 cursor
        ptrnode cursor = head;
        while(cursor->value != nilai_dicari)
            cursor = cursor->next;

        ptrnode new_node = createNode(nilai);
        new_node->prev = cursor->prev;
        cursor->prev->next = new_node;
        new_node->next = cursor;
        cursor->prev = new_node;
    }

return(head);
}

```

```
}
```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```
/* Tambahkan node baru dengan value = 35 sebelum node  
dengan value 40 */  
head = insert_before(head, 35, 40);
```

Untuk melihat kembali isi keseluruhan simpul setelah beberapa penambahan simpul yang telah kita lakukan, pada fungsi **main()** panggil kembali fungsi untuk menampilkan nilai dari Linked List yang telah Anda buat sebelumnya pada Kegiatan Praktikum C.

## E. Penghapusan Simpul dari Linked List

1. Delete kiri atau delete awal, maksudnya adalah menghapus simpul yang ada di ujung paling kiri atau paling awal atau simpul head dari Linked List yang sudah ada. Tambahkan fungsi **remove\_first()** berikut di luar fungsi **main()**.

```
ptrnode remove_first(ptrnode head){  
    if(head == NULL)  
        return;  
  
    ptrnode first = head;  
    head = head->next;  
    head->prev = NULL;  
    first->next = NULL;  
  
    free(first);  
  
    return(head);  
}
```

Kemudian coba panggil fungsi **remove\_first()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Hapus node head */  
head = remove_first(head);
```



2. Delete kanan atau delete akhir, maksudnya adalah menghapus simpul yang ada di ujung paling kanan atau paling akhir atau simpul tail dari Linked List yang sudah ada. Tambahkan fungsi **remove\_last()** berikut di luar fungsi **main()** .

```
ptrnode remove_last(ptrnode head){
    if(head == NULL)
        return;

    //cursor bantuan satu lagi (prev_tail) tidak
    dibutuhkan
    ptrnode tail = head;
    while(tail->next != NULL)
    {
        tail = tail->next;
    }

    tail->prev = NULL;
    tail->prev->next = NULL;
    free(tail);
    return(head);
}
```

Kemudian coba panggil fungsi **remove\_last()** dari fungsi **main()** dengan cara seperti di bawah ini.

```
/* Hapus node tail */
head = remove_last(head);
```

3. Delete tengah, maksudnya yaitu menghapus sebuah simpul yang berada di antara dua buah simpul lain. Bukan menghapus simpul yang paling kiri dan juga bukan menghapus simpul yang paling kanan. Tambahkan potongan program berikut di luar fungsi **main()** .

```
ptrnode remove_middle(ptrnode head, int nilai){
    ptrnode cursor = head;
    while(cursor != NULL)
    {
        if(cursor->next->value == nilai)
            break; //keluar dari iterasi
        cursor = cursor->next;
    }
}
```

```

    }

    if(cursor != NULL)
    {
        ptrnode tmp = cursor->next;
        cursor->next = tmp->next;
        tmp->next->prev = cursor;
        tmp->next = NULL;
        tmp->prev = NULL;
        free(tmp);
    }

    return(head);
}

```

Kemudian coba panggil fungsi tersebut dari fungsi **main()** dengan cara seperti di bawah ini, lalu coba jalankan.

```

/* Hapus node di tengah */
head = remove_middle(head, 30);

```

Menggunakan salah satu cara penghapusan simpul di atas, kita dapat menghapus keseluruhan simpul yang ada pada Linked List.

Setelah fungsi-fungsi hapus di atas dijalankan, panggil kembali fungsi untuk menampilkan nilai linked list pada fungsi **main()**, untuk melihat apa isi yang tersisa pada Linked List.

## 5.5 Penugasan

1. Cobalah untuk memodifikasi potongan program pada pembuatan simpul awal, insert kanan, dan delete kanan sehingga pointer tail dideklarasikan dan selalu diperbaharui isinya saat penambahan dan penghapusan simpul dari kanan.
2. Buat sebuah program untuk menampilkan output di bawah ini menggunakan double linked list!

Input the number of nodes : 3  
Input data for node 1 : 2  
Input data for node 2 : 5  
Input data for node 3 : 8

Data entered in the list are :

node 1 : 2

node 2 : 5

node 3 : 8

Input data for the first node : 1

After insertion the new list are :

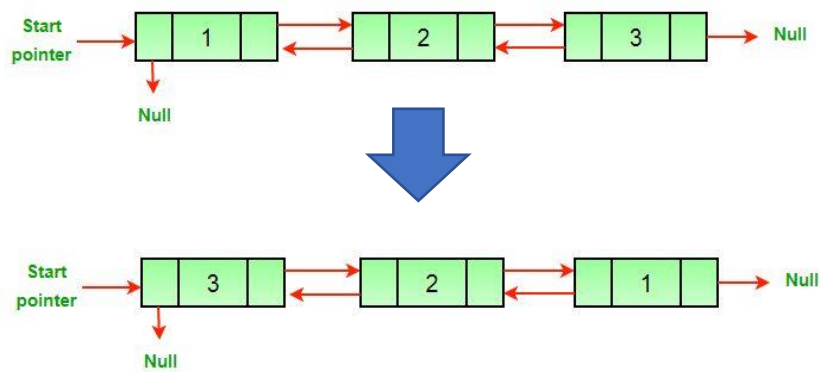
node 1 : 1

node 2 : 2

node 3 : 5

node 4 : 8

3. Bagaimana untuk membalik nilai-nilai dalam double linked list (tail ke head)?



## MODUL 6: STACK (TUMPUKAN)

---

### 6.1 Deskripsi Singkat

Stack atau tumpukan adalah suatu struktur data berbentuk list atau daftar yang proses penambahan atau penghapusan datanya hanya melalui satu posisi yaitu item teratasnya atau biasa disebut top. Tumpukan dapat diimplementasikan dengan array atau linked list.

### 6.2 Tujuan Praktikum

- 1) Mahasiswa mampu mengimplementasikan berbagai persoalan tumpukan dengan menggunakan bahasa C
- 2) Mahasiswa dapat melakukan operasi penyisipan (push) maupun penghapusan (pop) item dari tumpukan dengan menggunakan bahasa C

### 6.3 Material Praktikum

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

### 6.4 Kegiatan Praktikum

Pada praktikum kali ini, kita akan membuat sebuah program konversi bilangan desimal ke biner dengan menggunakan tumpukan yang diimplementasikan dengan menggunakan array satu dimensi. Dalam proses pembuatannya, ada 4 tahap, yaitu:

1. Persiapan,
2. Membuat fungsi push,
3. Membuat fungsi pop,
4. Finalisasi.

#### A. Persiapan

Dalam tahapan ini kita harus memahami bagaimana cara mengkonversi bilangan desimal ke biner. Dalam modul 6 ini, kita akan menggunakan algoritma yang sudah umum dalam mengkonversi bilangan desimal ke biner, yaitu bilangan desimal yang akan dikonversi dibagi 2 secara berulang sampai hasil baginya 0.

Contoh 1: Konversi bilangan desimal nilai 50 menjadi bilangan biner.

$$50/2 = 25 \text{ sisa bagi adalah } 0$$

$$25/2 = 12 \text{ sisa bagi adalah } 1$$

$$12/2 = 6 \text{ sisa bagi adalah } 0$$

$$6/2 = 3 \text{ sisa bagi adalah } 0$$

$3/2 = 1$  sisa bagi adalah 1

$1/2 = 0$  sisa bagi adalah 1

Hasil pembagian tersebut kemudian diurutkan dari yang paling akhir hingga paling awal menjadi 110010.

Jadi Hasil Konversi bilangan desimal 50 menjadi bilangan biner adalah 110010.

Contoh 2: Konversi bilangan desimal 105 menjadi bilangan biner.

$105/2 = 52$  sisa bagi adalah 1

$52/2 = 26$  sisa bagi adalah 0

$26/2 = 13$  sisa bagi adalah 0

$13/2 = 6$  sisa bagi adalah 1

$6/2 = 3$  sisa bagi adalah 0

$3/2 = 1$  sisa bagi adalah 1

$1/2 = 0$  sisa bagi adalah 1

Hasil pembagian tersebut kemudian diurutkan dari yang paling akhir hingga paling awal menjadi 1101001.

Jadi Hasil Konversi bilangan desimal 105 menjadi bilangan biner adalah 1101001.

Menggunakan tumpukan, setiap kali bilangan desimal dibagi dengan 2, sisa bagi disimpan ke dalam tumpukan (push). Setelah selesai, tumpukan di hapus itemnya satu persatu (pop) dan ditampilkan. Itulah hasil konversi bilangan desimal ke biner. Setelah memahami algoritmanya, maka perlu dipersiapkan struktur data tumpukan serta fungsi lain yang akan digunakan dalam proses berikutnya.

```

#include<stdio.h>

typedef struct
{
    int item[50];
    int jml_item;
} stack;

//menyiapkan tumpukan kosong
void initializestack(stack *s)
{
    s->jml_item = 0;
}

//jika tumpukan kosong maka nilai fungsinya 1 (true),
//jika tidak 0 (false)
int isEmpty(stack *s)
{
    return (s->jml_item == 0);
}

//jika tumpukan sudah full(dlm deklarasi stack, field
item //adalah array dg jumlah elemen sebanyak 50)maka
nilai
//nilai fungsinya 1, jika tidak 0
int isFull(stack *s)
{
    return (s->jml_item == 50);
}

```

sampai dengan potongan program di atas, kita sudah menyiapkan struktur dari tumpukan dan beberapa fungsi yang nanti akan digunakan dalam proses selanjutnya.

ketik potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum6.c**.

## B. Pembuatan Fungsi push

Untuk membuat fungsi push, ada dua kondisi yang harus diperhatikan. Kondisi pertama yaitu saat tumpukan sudah penuh maka artinya bilangan yang diinputkan terlalu besar (lebih dari 49 digit bilangan biner). Saat kondisi penuh, fungsi tidak menambahkan item ke dalam tumpukan dan menampilkan pesan bahwa bilangan terlalu besar. Sedangkan kondisi kedua adalah tumpukan belum penuh. Saat seperti ini, fungsi menambahkan item dan jml\_item bertambah 1.

```
void push(int x, stack *s)
{
    if(isFull(&s))
        printf("Bilangan terlalu besar !\n");
    else {
        s->item[s->jml_item]=x; ++(s->jml_item);
    }
}
```

## C. Pembuatan Fungsi pop

Fungsi pop adalah fungsi yang prosesnya berlawanan dengan fungsi push. Seperti fungsi push, fungsi ini juga harus memperhatikan 2 kondisi. Pertama saat tumpukan kosong, maka fungsi akan mengembalikan nilai 0. Kedua saat tumpukan tidak kosong. Dalam kondisi seperti ini, fungsi akan mengembalikan sesuai nilai item dan jml\_item berkurang 1.

```
int pop(stack *s)
{
    if(isEmpty(&s)) return (0);
    else {
        --(s->jml_item);
        return (s->item[s->jml_item]);
    }
}
```

#### D. Finalisasi

Langkah terakhir adalah mengatur fungsi-fungsi tersebut sehingga tumpukan bisa menyimpan hasil bagi bilangan yang ingin dikonversi. Setelah itu tumpukan di pop satu persatu itemnya yang merupakan hasil konversi bilangan desimal tersebut. Fungsi akan mengembalikan nilai 0 saat pengguna menginputkan nilai 0 untuk bilangan desimal yang ingin dikonversi.

```
void main(){
    stack tumpukan;
    int bil_desimal;
    int sisa_bagi;

    printf("Program konversi Desimal ke Biner\n\n");
    initializestack(&tumpukan);
    printf("Masukkan bilangan desimal = ");
    scanf("%d", &bil_desimal);

    if (bil_desimal==0) printf("Hasil konversi ke biner = 0");
    else{
        int n;
        for(n=bil_desimal;n>0;n=n/2){
            sisa_bagi=n%2;
            push(sisa_bagi,&tumpukan);
        }

        int i;
        printf("Hasil konversi ke biner = ");
        for(i=tumpukan.jml_item;i>0;i--){
            printf("%d", pop(&tumpukan));
        }
    }
}
```

#### 6.5 Penugasan

Modifikasi program konversi bilangan desimal ke biner yang telah kita buat, dengan ketentuan sebagai berikut:



1. Buatlah sebuah fungsi konversi, agar fungsi utama/ main dalam program di atas menjadi lebih sederhana.
2. Ubah program konversi desimal ke biner di atas menjadi menggunakan linked list.
3. Modifikasi program tersebut agar bisa juga mengkonversi bilangan desimal negatif.
4. Tambahkan sebuah fungsi yang dapat mengkonversi bilangan desimal menjadi bilangan oktal. Sehingga pertama kali program dijalankan, user dapat memilih ingin mengkonversi bilangan desimal menjadi biner atau oktal.

## MODUL 7: Antrian

---

### 7.1 Deskripsi Singkat

Queue atau antrian adalah suatu struktur data berbentuk list atau daftar yang proses pengelolaan datanya menggunakan konsep FIFO(First In First Out) atau LIFO(Last In Last OUT), sehingga proses menambah data dan penghapusannya berbeda dengan tumpukan, menggunakan dua pintu. Menambahkan data ke rear atau tail dan menghapus data mulai dari head atau front. Antrian dapat diimplementasikan dengan array atau linked list.

### 7.2 Tujuan Praktikum

- 1) Mahasiswa mampu mengimplementasikan berbagai persoalan antrian dengan menggunakan bahasa C
- 2) Mahasiswa dapat melakukan operasi penyisipan (Enqueue) maupun penghapusan (Dequeue) item dari antrian dengan menggunakan bahasa C

### 7.3 Material Praktikum

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

### 7.4 Kegiatan Praktikum

Pada praktikum kali ini, kita akan membuat sebuah program antrian berprioritas yang akan memilih 5 orang perwakilan mahasiswa yg akan mengikuti lomba pemrograman. Kelima orang perwakilan ini dipilih pertama berdasarkan nilai UTS Alpro, kemudian berdasarkan nilai UTS Kalkulus dan terakhir berdasarkan urutan pendaftaran.

Dalam proses pembuatannya, ada 4 tahap, yaitu:

1. Persiapan,
2. Membuat fungsi dequeue,
3. Membuat fungsi enqueue,
4. Finalisasi.

#### A. Persiapan

Dalam tahapan ini kita harus mempersiapkan struktur data yang dibutuhkan serta fungsi lain yang akan digunakan dalam proses berikutnya.

```

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

// Node
typedef struct node {

char nama[20];

int alpro;

int kalkulus;
    struct node* next;
} mhs;

int count=0;

// Function to Create A New Node
mhs* newmhs(char a[], int alp, int kal)
{

mhs* temp = (mhs*)malloc(sizeof(mhs));

strcpy(temp->nama, a);

temp->alpro = alp;

temp->kalkulus = kal;

temp->next = NULL;

return temp;
}

```

sampai dengan potongan program di atas, kita sudah menyiapkan struktur data serta beberapa fungsi yang nanti akan digunakan dalam proses selanjutnya.

ketik potongan program tersebut pada IDE Anda, lalu simpan dengan nama **praktikum7.c**.

### **B. Pembuatan Fungsi dequeue**

Untuk membuat fungsi dequeue, ada dua kondisi yang harus diperhatikan. Kondisi pertama yaitu saat antrian masih kosong dan selain itu. Jika masih kosong maka tidak bisa menghapus, jika tidak baru menghapus antrian dari depan.

```
// menghapus pendaftar
void dequeue(mhs** head)
{
    if ((*head) != NULL) {
        mhs* temp = *head;
        (*head) = (*head)->next;
        free(temp);
    }
}
```

### **C. Pembuatan Fungsi enqueue**

Fungsi enqueue adalah fungsi utama dari program ini. Ada beberapa kondisi yang perlu diperhatikan berkaitan dengan penempatan antrian baru.

```
// Function to push according to priority
void enqueue(mhs** head, char n[], int alp, int kal)
{
    mhs* temp = newmhs(n, alp, kal);

    if ((*head) == NULL) {
        (*head) = temp;
    }
}
```

```

else if ((*head)->alpro > alp) {
temp->next = *head;
(*head) = temp;
}

else if ((*head)->alpro==alp) {
    mhs*start=(*head);
    while (start->next != NULL && start->next->kalkulus <
kal) {
start = start->next;
}

temp->next = start->next;
start->next = temp;
}

else
{
    mhs* start = (*head);
while (start->next != NULL && start->next->alpro < alp) {
start = start->next;
}

if (start->next->alpro==alp){
    while (start->next != NULL && start->next->kalkulus <
kal) {

start = start->next;

}
}
}

```

```

temp->next = start->next;
start->next = temp;
}

count++;
}

```

#### D. Finalisasi

Langkah terakhir adalah mengatur fungsi-fungsi tersebut sehingga antrian bisa menyimpan hasil yang sesuai.

```

void display(mhs* head)
{
    if(head == NULL)
    {
        printf("Belum ada yang daftar\n");
    }
    else
    {
        printf("Nama:%s Alpro: %d Kalkulus: %d\n", head-
>nama,head->alpro,head->kalkulus);
        display(head->next);
    }
}

int main()
{
    mhs* wakil;

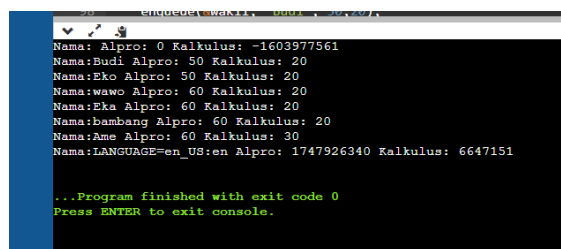
    enqueue(&wakil, "Eko", 50,20);
    enqueue(&wakil, "Budi", 50,20);
    enqueue(&wakil, "bambang", 60,20);
    enqueue(&wakil, "Eka", 60,20);
    enqueue(&wakil, "wawo", 60,20);
    enqueue(&wakil, "Ame", 60,30);
}

```

```
    display(wakil);  
  
return 0;  
}
```

## 7.5 Penugasan

1. Output dari program yang dibuat adalah seperti gambar di bawah. Perbaiki program agar yang muncul hanya yang diinputkan saja



The screenshot shows a console window with the following output:

```
Nama: Alpro: 0 Kalkulus: -1603977561  
Nama:Budi Alpro: 50 Kalkulus: 20  
Nama:Eko Alpro: 50 Kalkulus: 20  
Nama:wawo Alpro: 60 Kalkulus: 20  
Nama:Eka Alpro: 60 Kalkulus: 20  
Nama:bambang Alpro: 60 Kalkulus: 20  
Nama:Ame Alpro: 60 Kalkulus: 30  
Nama:LANGUAGE=en_US:en Alpro: 1747926340 Kalkulus: 6647151  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

2. Modifikasi program agar secara otomatis antrian hanya maksimal berisi 5 pendaftar terbaik sesuai kriteria yang ditentukan.
3. Modifikasi program tersebut agar menampilkan mulai dari urutan teratas.
4. Buat yang versi arraynya.

## MODUL 8: TREE BAGIAN 1

---

### 8.1 Deskripsi Singkat

Tree atau pohon merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki antara elemen-elemennya. Definisi tree yaitu sekumpulan simpul (node) yang saling dihubungkan oleh sebuah vektor (edge) yang tidak membentuk sirkuit. Terdapat banyak jenis tree. Salah satunya yang paling penting adalah pohon biner (*binary tree*) karena banyak aplikasinya. Pada praktikum ini kita akan mempelajari Binary Tree dan Binary Search Tree.

### 8.2 Tujuan Praktikum

- 1) Mahasiswa mampu mengimplementasikan binary tree dengan menggunakan bahasa C
- 2) Mahasiswa mampu mengimplementasikan binary search tree dengan menggunakan bahasa C

### 8.3 Material Praktikum

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

### 8.4 Kegiatan Praktikum

#### A. Binary Tree

Binary Tree adalah tree yang pada masing-masing simpulnya hanya dapat memiliki maksimum 2 (dua) node child, tidak boleh lebih. Terdapat beberapa operasi yang dapat dilakukan pada binary tree, diantaranya adalah penambahan simpul baru dan peneluruan atau traversal pohon. Kita akan mencoba membuat program yang menjalankan kedua operasi tersebut.

1. Simpul-simpul pada tree, dapat dibuat menggunakan structure. Structure tersebut dapat menyimpan data tertentu, serta menyimpan informasi lokasi anak kanan dan anak kiri. Contohnya seperti ini.

```
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
```



2. Untuk membuat simpul baru, yang kita lakukan adalah mengalokasikan memori untuk structure simpul di atas dan melakukan assignment nilai pada `data`, `*left`, `*right`, dengan cara seperti ini.

```
struct node *newNode(int data)
{
    struct node *node = (struct
node*)malloc(sizeof(struct node));

    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}
```

3. Untuk mencoba membuat simpul baru, panggil fungsi **newNode** di atas pada fungsi **main**. Contohnya adalah sebagai berikut.

```
int main()
{
    struct node* root = newNode(8);

    root->left = newNode(3);
    root->left->right = newNode(1);
    root->left->left = newNode(6);
    root->left->right->right = newNode(4);
    root->left->right->left = newNode(7);

    root->right = newNode(10);
    root->right->left = newNode(14);
    root->right->left->right = newNode(13);

    return 0;
}
```

Simpan potongan program pada ketiga tahapan di atas dengan nama **Praktikum8A.c**. Pastikan tidak ada error yang muncul.

4. Untuk menampilkan isi dari pohon yang telah kita buat, kita dapat menelusuri satu-persatu simpul yang terdapat pada pohon (traversal). Terdapat tiga cara untuk melintasi sebuah tree, yaitu preorder, inorder, dan postorder. Ketik ulang dan pahami cara kerja potongan program untuk menelusuri pohon di bawah ini.

```
void displayPreorder(struct node* node)
{
    if(node == NULL)
        return;

    printf("%d ", node->data); //root
    displayPreorder(node->left); //subtree kiri
    displayPreorder(node->right); //subtree kanan
}

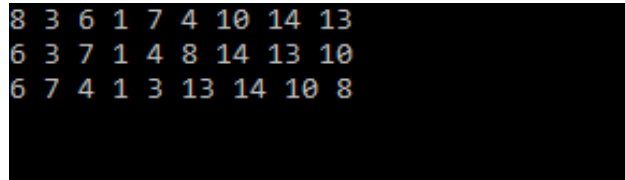
void displayInorder(struct node* node)
{
    if(node == NULL)
        return;

    displayInorder(node->left); //subtree kiri
    printf("%d ", node->data); //root
    displayInorder(node->right); //subtree kanan
}

void displayPostorder(struct node* node)
{
    if(node == NULL)
        return;

    displayPostorder(node->left); //subtree kiri
    displayPostorder(node->right); //subtree kanan
    printf("%d ", node->data); //root
}
```

Tambahkan potongan program di atas pada **Praktikum8A.c**, lalu panggil fungsi-fungsi traversal tersebut melalui fungsi main. Jika berhasil, maka data pada seluruh node yang ada pada tree akan muncul di layar seperti pada gambar berikut:



```
8 3 6 1 7 4 10 14 13
6 3 7 1 4 8 14 13 10
6 7 4 1 3 13 14 10 8
```

## B. Binary Search Tree

Kekurangan dari Binary Tree adalah data pada node-node yang ada tidak terurut. Sehingga akan sulit bila ingin melakukan pencarian/search node tertentu dalam binary tree karena bisa jadi program harus mengunjungi seluruh node pada tree untuk dapat menemukan node yang dicari. Binary Search Tree adalah Binary Tree yang isi nodenya sudah terurut. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree. Binary Search Tree memiliki sifat dimana semua left child harus lebih kecil dari pada right child dan parentnya. Pada dasarnya operasi dalam Binary Search Tree sama dengan Binary Tree biasa, hanya saja saat melakukan insert, update, dan delete, kita harus memastikan urutan node pada tree tetap terjaga. Pada praktikum ini kita akan mencoba memodifikasi fungsi insert pada **Praktikum8A.c** agar pohon yang terbentuk adalah Binary Search Tree.

5. Tambahkan fungsi insert berikut pada **Praktikum8A.c**. Setiap kali kita akan menambahkan simpul baru pada pohon, program akan mencari posisi yang tepat untuk simpul baru tersebut. Program akan membandingkan data pada simpul baru dengan data pada simpul-simpul yang sudah ada pada pohon.

```
struct node* insert(struct node* root, int newData)
{
    if(root == NULL)
    {
        root = newNode(newData);
    }
    else
    {
        int is_left = 0;
        struct node* cursor = root;
        struct node* prev = NULL;
```

```

while(cursor != NULL)
{
    prev = cursor;
    if(newData < cursor->data)
    {
        is_left = 1;
        cursor = cursor->left;
    }
    else if(newData > cursor->data)
    {
        is_left = 0;
        cursor = cursor->right;
    }
}

if(is_left == 1)
    prev->left = newNode(newData);
else
    prev->right = newNode(newData);
}

return root;
}

```

6. Modifikasi potongan program pada fungsi main. Sebelumnya kita menambahkan simpul baru secara langsung pada posisi yang kita inginkan. Sekarang, simpul-simpul baru ditambahkan dengan menggunakan fungsi **insert**.

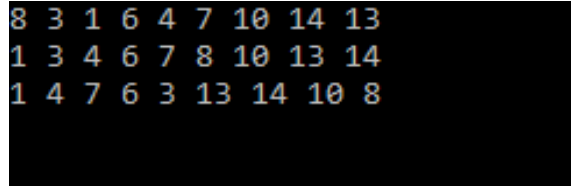
```

struct node* root = newNode(8);
root = insert(root, 3);
root = insert(root, 1);
root = insert(root, 6);
root = insert(root, 4);
root = insert(root, 7);
root = insert(root, 10);

```

```
root = insert(root, 14);  
root = insert(root, 13);
```

Setelah itu, coba jalankan program **Praktikum8A.c**. Jika berhasil, maka output yang ditampilkan di layar seperti gambar berikut:



```
8 3 1 6 4 7 10 14 13  
1 3 4 6 7 8 10 13 14  
1 4 7 6 3 13 14 10 8
```

7. Sekarang setelah node-node telah terurut, maka akan lebih mudah untuk melakukan pencarian sebuah node. Tambahkan potongan program untuk pencarian node berikut pada **Praktikum8A.c**. Ketik ulang sambil dipahami cara kerjanya.

```
void search_node(struct node* root, int data)  
{  
    struct node* cursor = root;  
  
    while(cursor->data != data){  
        if(cursor != NULL){  
            if(data > cursor->data){  
                cursor = cursor->right;  
            }  
            else {  
                cursor = cursor->left;  
            }  
  
            if(cursor == NULL){  
                printf("\nNode %d tidak ditemukan",  
data);  
            }  
        }  
    }  
  
    printf("\nNode %d ditemukan", data);  
    return;  
}
```

Cobalah untuk melakukan pencarian beberapa node, melalui fungsi main. Contoh:

```
search_node(root, 6);  
search_node(root, 100);
```

8. Jika sudah berhasil melakukan pencarian sebuah node, maka menghapus sebuah node dari pohon adalah sebuah hal yang mudah. Tambahkan potongan program berikut pada **Praktikum8A.c**.

```
struct node* delete_node(struct node* root, int  
deletedData)  
{  
    if(root == NULL)  
        return;  
  
    struct node* cursor;  
    if(deletedData > root->data)  
        root->right = delete_node(root->  
>right, deletedData);  
    else if(deletedData < root->data)  
        root->left = delete_node(root->left,  
deletedData);  
    else  
    {  
        //1 CHILD  
        if(root->left == NULL){  
            cursor = root->right;  
            free(root);  
            root = cursor;  
        }  
        else if(root->right == NULL) {  
            cursor = root->left;  
            free(root);  
            root = cursor;  
        }  
        //2 CHILDS NODE  
        else{  
            cursor = root->right;
```

```

        while(cursor->left != NULL) {
            cursor = cursor->left;
        }
        root->data = cursor->data;
        root->right = delete_node(root->right,
cursor->data);
    }
}
return root;
}

```

Node yang dihapus, digantikan dengan predecessor atau successor node tersebut pada urutan inorder. Pada fungsi `delete_node` di atas, node yang dihapus digantikan oleh predecessor ataukah successor? Pastikan Anda telah memahami cara kerja fungsi tersebut. Kemudian cobalah untuk menghapus beberapa node dari pohon melalui fungsi `main`. Contoh:

```

root=delete_node(root, 13);
root=delete_node(root, 3);

```

## 8.5 Penugasan

Sekarang, Anda telah memahami cara menyimpan data menggunakan struktur data tree dan melakukan operasi sederhana pada tree yang telah dibuat.

1. Modifikasi program pada **Praktikum8A.c** sehingga data yang disimpan pada Binary Search Tree, bukan lagi sebuah angka bertipe integer, melainkan data nama mahasiswa dengan tipe **char[30]** . Simpan hasil modifikasi Anda pada file **Praktikum8B.c**.

## MODUL 9: TREE BAGIAN 2

---

### 9.1 Deskripsi Singkat

Meskipun Binary Search Tree sederhana dan mudah dimengerti, ia memiliki satu masalah utama: tidak seimbang. Dalam praktikum kali ini kita akan mempelajari salah satu jenis binary tree lainnya, yaitu AVL Tree. AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi/ level maksimal 1 antara subtree kiri dan subtree kanan. AVL Tree muncul untuk menyeimbangkan Binary Search Tree. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan.

Selain AVL Tree, terdapat pula Height Balanced n Tree, yakni Binary Search Tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n. Sehingga, dengan kata lain AVL Tree adalah Height Balanced 1 Tree.

### 9.2 Tujuan Praktikum

Mahasiswa mampu memecahkan persoalan dengan menggunakan AVL Tree pada bahasa pemrograman C

### 9.3 Material Praktikum

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

### 9.4 Kegiatan Praktikum

Kita akan membuat program untuk menyimpan data ke dalam struktur data AVL Tree. Kita akan memodifikasi program Binary Search Tree (BST) yang telah kita buat pada praktikum sebelumnya (**Praktikum8A.c**). Simpan ulang **Praktikum8A.c** menjadi file baru dengan nama **Praktikum9A.c**.

#### A. Menambah Simpul Baru

1. Pertama kita akan memodifikasi structure yang digunakan untuk membuat simpul-simpul pada tree. Pada AVL Tree, kita memerlukan informasi height atau tinggi pohon untuk memeriksa keseimbangan pohon. Pada `struct node`, kita tambahkan satu elemen tambahan:

```
int height;
```

Saat kita ingin membuat sebuah simpul baru, kita menggunakan fungsi `struct node *newNode(int data)`. Pada fungsi tersebut, kita tambahkan satu baris program untuk menginisialisasi nilai `height`.

```
new_node->height = 1;
```



Simpul baru pada awalnya ditambahkan sebagai daun, oleh karena itu heightnya bernilai 1.

2. Kemudian, kita modifikasi fungsi insert. Perbedaan AVL Tree dibandingkan dengan BST adalah AVL Tree selalu memastikan keseimbangan pohon setiap kali ada simpul baru yang ditambahkan atau dihapus dari pohon (*self balancing binary search tree*). Tahapan untuk menambahkan simpul baru pada AVL Tree adalah sebagai berikut:

- 1) Lakukan insert seperti halnya pada BST
- 2) Perbarui height untuk seluruh node, mulai dari node yang terakhir ditambahkan hingga ke root. Height node adalah height subtree kiri atau subtree kanan yang ada di bawahnya ditambahkan dengan 1.
- 3) Hitung balance factor untuk seluruh node, mulai dari node yang terakhir ditambahkan hingga ke root, untuk menentukan apakah node balanced atau tidak
- 4) Jika node unbalanced, lakukan rotasi. Ada 4 kasus rotasi, bergantung kepada posisi node terbawah.
- 5) Hasil akhir fungsi adalah mengembalikan node yang telah balanced.

Jika kita perhatikan, tahapan-tahapan di atas dilakukan secara berulang-ulang untuk node-node yang ada pada tree. Oleh karena itu, tahapan tersebut akan diimplementasikan secara rekursif. Jika diimplementasikan, kelima tahapan di atas dapat dituliskan seperti berikut ini.

```
struct node* insert(struct node* root, int new_data)
{
    // 1. Lakukan BST insert biasa
    if (root == NULL)
        return(newNode(new_data));
    // asumsi tidak boleh ada nilai yang sama dalam BST
    if (new_data < root->data)
        root->left = insert(root->left, new_data);
    else if (new_data > root->data)
        root->right = insert(root->right, new_data);

    // 2. Update height dari node baru sampai root
    root->height = 1 + max(getHeight(root->left),
        getHeight(root->right));
}
```

```

// 3. Hitung balance factor untuk menentukan apakah
node unbalanced
int balance = getBalanceFactor(root);

// Jika tidak balanced, return hasil rotation
// Kasus 1: Left Left
if (balance > 1 && new_data < root->left->data)
    return rightRotate(root);
// Kasus 2: Right Right
if (balance < -1 && new_data > root->right->data)
    return leftRotate(root);

// Kasus 3: Right Left
if (balance < -1 && new_data < root->right->data)
{
    root->right = rightRotate(root->right);
    return leftRotate(root);
}

// Kasus 4: Left Right
if (balance > 1 && new_data > root->left->data)
{
    root->left = leftRotate(root->left);
    return rightRotate(root);
}

// return node jika balanced
return root;
}

```

Ganti fungsi `insert` yang sudah ada di **Praktikum9A.c** dengan fungsi `insert` di atas. Perhatikan bahwa pada fungsi `insert` di atas terdapat beberapa fungsi tambahan yang dipanggil untuk membantu proses insert simpul baru ke dalam AVL Tree, antara lain:

- 1) `max`, fungsi untuk mencari tinggi maksimum antara node kiri dan node kanan

- 2) getHeight, fungsi untuk mencari tinggi sebuah node
- 3) getBalanceFactor, fungsi untuk menghitung balance faktor sebuah node
- 4) rightRotate, fungsi untuk rotasi kanan
- 5) leftRotate, fungsi untuk rotasi kiri

```
int max(int a, int b)
{
    if (a > b) return a;
    else return b;
}

int getHeight(struct node* N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

// Hitung Balance factor untuk node N
int getBalanceFactor(struct node *N)
{
    if (N == NULL)
        return 0;
    return getHeight(N->left) - getHeight(N->right);
}

struct node* rightRotate(struct node *T)
{
    struct node *new_root = T->left;
    struct node *orphan = new_root->right;

    // Lakukan rotasi
    new_root->right = T;
    T->left = orphan;

    // Update height
```

```

    T->height = max(getHeight(T->left), getHeight(T->right))+1;
new_root->height = max(getHeight(new_root->left),
getHeight(new_root->right))+1;

    // Return root baru
    return new_root;
}

struct node *leftRotate(struct node *T)
{
    struct node *new_root = T->right;
    struct node *orphan = new_root->left;

    // Lakukan rotasi
    new_root->left = T;
    T->right = orphan;

    // Update height
    T->height = max(getHeight(T->left), getHeight(T->right))+1;
    new_root->height = max(getHeight(new_root->left),
getHeight(new_root->right))+1;

    // Return root baru
    return new_root;
}

```

Untuk menguji apakah fungsi insert tersebut bisa dijalankan, cobalah untuk menginisiasi tree dan menambahkan beberapa simpul, lalu menampilkannya secara preorder, inorder, atau postorder. Contoh:

```

struct Node *root = NULL;

root = insert(root, 9);
root = insert(root, 5);
root = insert(root, 10);

```

```

root = insert(root, 0);
root = insert(root, 6);
root = insert(root, 11);
root = insert(root, -1);
root = insert(root, 1);
root = insert(root, 2);

```

Jika berhasil, maka akan muncul tampilan seperti ini di layar Anda.

```

9 1 0 -1 5 2 6 10 11
-1 0 1 2 5 6 9 10 11
-1 0 2 6 5 1 11 10 9

```

## B. Menghapus Sebuah Simpul

Tahapan pada penghapusan simpul AVL Tree mirip dengan tahapan penambahan simpul baru yang telah dijelaskan di atas. Penghapusan simpul pada AVL Tree dilakukan dengan cara yang sama seperti penghapusan simpul pada BST. Namun, setelahnya kita harus mengecek kembali tinggi dan balance factor untuk setiap node agar keseimbangan pohon tetap terjaga. Tahapan penghapusan simpul pada AVL Tree adalah sebagai berikut:

- 1) Lakukan delete seperti halnya pada BST
- 2) Perbarui height untuk seluruh node, mulai dari node yang terakhir ditambahkan hingga ke root. Height node adalah height subtree kiri atau subtree kanan yang ada di bawahnya ditambahkan dengan 1.
- 3) Hitung balance factor untuk seluruh node, mulai dari node yang terakhir ditambahkan hingga ke root, untuk menentukan apakah node balanced atau tidak
- 4) Jika node unbalanced, lakukan rotasi. Ada 4 kasus rotasi, bergantung kepada posisi node terbawah.
- 5) Hasil akhir fungsi adalah mengembalikan node yang telah balanced.

Oleh karena itu, untuk penghapusan, kita tambahkan potongan program untuk menyeimbangkan pohon (tahap 2-5) pada fungsi `delete_node` yang sudah ada pada **Praktikum9A.c**, persisnya sebelum baris `return root;`.

```

// Jika setelah dilakukan delete, tree kosong maka
return root

    if (root == NULL)
        return root;

```

```

// 2. Update height dari node
root->height = 1 + max(getHeight(root->left),
getHeight(root->right));

// 3. Hitung balance factor untuk menentukan apakah
root unbalanced
int balance = getBalanceFactor(root);

// Jika tidak balanced, return hasil rotation

// Kasus 1: Left Left
    if (balance > 1 && getBalanceFactor(root->left) >=
0)
        return rightRotate(root);

// Kasus 2: Right Right
    if (balance < -1 && getBalanceFactor(root->right)
<= 0)
        return leftRotate(root);

// Kasus 3: Right Left
    if (balance < -1 && getBalanceFactor(root->right)
> 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

// Kasus 4: Left Right
    if (balance > 1 && getBalanceFactor(root->left) <
0)
    {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }

```

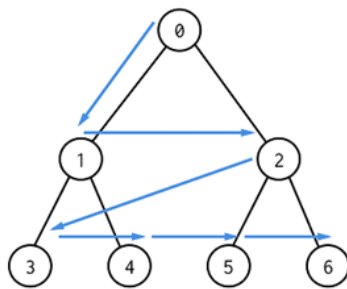
Untuk memastikan fungsi `delete_node` tersebut sudah berjalan, cobalah untuk menghapus salah satu node pada tree, lalu tampilkan kembali isi tree secara preorder, inorder, atau postorder. Contoh:

```
root=delete_node(root, 10);
```

## 9.5 Penugasan

Sekarang, Anda telah memahami cara membuat AVL Tree. Untuk memperdalam pemahaman Anda mengenai AVL Tree, modifikasi BST untuk menyimpan nama mahasiswa yang ada pada program **Praktikum8B.c** menjadi AVL Tree.

1. Simpan ulang **Praktikum9B.c** dengan nama **Praktikum9B.c**, lalu lakukan modifikasi pada fungsi insert dan delete seperti yang kita lakukan pada kegiatan praktikum di atas.
2. Kemudian, tambahkan sebuah fungsi untuk menampilkan nama-nama mahasiswa yang ada pada tree dengan alur:



## MODUL 10: Hashing

---

### 10.1 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.

### 10.2 Tujuan Praktikum

Mahasiswa mampu mengimplementasikan Hashing dengan menggunakan bahasa C.

### 10.3 Material Praktikum

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

### 10.4 Kegiatan Praktikum

#### Hashing

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

$$\text{key mod } 10$$

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.



## 5. Persiapan

Pada tahap ini akan dibuat hash table (dalam program akan diberi nama array) dengan bentuk struktur 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;
}
```

#### 6. 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 apakah 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 dilakukan 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 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);
}

```

## 7. 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");
}
```

## 8. 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;
}

```

Tambahkan potongan program di atas dengan nama **Praktikum10A.c**, kemudian coba jalankan dengan menggunakan data berikut: 45, 72, 39, 48, 56, 77, 91, 63, 84, 90

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

## 10.5 Penugasan



2. Modifikasi program **Praktikum10A.c** sehingga jumlah data yang disimpan pada tabel hash bisa fleksibel . Simpan hasil modifikasi Anda pada file **Praktikum10B.c**.
3. Modifikasi program **Praktikum10B.c** sehingga data pada baris/indeks yang dihapus (flag=2) tidak dapat diisi lagi. Simpan hasil modifikasi Anda pada file **Praktikum10C.c**.
4. Modifikasi program **Praktikum10C.c** dengan tanpa menggunakan pointer dan collision resolutionnya dengan menggunakan metode quadratic probing. Simpan hasil modifikasi Anda pada file **Praktikum10D.c**.

## MODUL 11: PENCARIAN

---

### 11.1 Deskripsi Singkat

Proses pencarian adalah proses menemukan data tertentu di dalam sekumpulan data yang bertipe sama. Pencarian adalah salah satu hal yang fundamental dalam pemrograman. Jika kita membuka aplikasi pada komputer, fitur yang hampir pasti selalu ada adalah fitur pencarian. Entah itu mencari sebuah kata dalam kumpulan teks, mencari data tertentu pada spreadsheet, mencari mahasiswa dengan nama tertentu pada aplikasi database dan lain sebagainya.

Pada modul praktikum ini akan dibahas 2 metode pencarian dasar yaitu pencarian sekuensial dan pencarian biner. Metode pencarian tersebut dapat diterapkan pada struktur data array dan linked list

### 11.2 Tujuan Praktikum

- 1) Mahasiswa mampu menerapkan pencarian sekuensial dan biner pada array menggunakan bahasa C
- 2) Mahasiswa dapat melakukan operasi pencarian sekuensial dan biner pada linked list 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

#### A. Pencarian Sekuensial

Pencarian sekuensial atau disebut juga pencarian berurutan atau pencarian linier merupakan metode pencarian yang paling sederhana. Pencarian sekuensial dilakukan dengan cara membandingkan nilai yang dicari pada sekumpulan data mulai dari awal data ke data berikutnya satu per satu hingga nilai yang dicari ditemukan atau tidak ditemukan pada sekumpulan data tersebut.

Dari deskripsi cara kerja pencarian sekuensial di atas, maka perulangan dilakukan mulai dari 1 hingga sejumlah  $n$  data. Jika beruntung maka nilai yang dicari ditemukan pada data pertama yang dicek. Jika tidak maka nilai yang dicari tidak ditemukan hingga akhir data. Proses pencarian dihentikan jika data yang dicari ditemukan, atau setelah pengecekan dilakukan hingga akhir data. Runtime pencarian sekuensial secara umum adalah  $O(n)$ .

#### A.1. Pencarian Sekuensial Pada Data Tidak Terurut (Array)

Berikut adalah listing program pencarian sekuensial pada data tidak terurut (array). Anda bisa menyimpannya pada file **modul11a1.c**.

```
#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

int seq_search(int data[], int size, int x){
    for (int i=0;i<size;i++){
        if(data[i]==x) return i;
    }
    return -1;
}

void main(){
    int data[MAX];
    int size; //ukuran array
    int x;
    fill_data(data,&size);
    printf("nilai yang mau dicari: ");
    scanf("%d", &x);
    if(seq_search(data,size,x)==-1) printf("tidak ditemukan");
    else printf("ditemukan pada indeks ke-
%d",seq_search(data,size,x));
}
```

Ketika dijalankan akan menghasilkan output sebagai berikut:

```
C:\Users\Panasonic\Documents\codeblocks\arraysearch.exe
Input ukuran array (max 100): 6
Input data: 2 4 3 6 5 1
nilai yang mau dicari: 6
ditemukan pada indeks ke-3
Process returned 26 (0x1A)    execution time : 22.368 s
Press any key to continue.
```

## A.2. Pencarian Sekuensial Pada Data Tidak Terurut (Linked List)

Berikut adalah contoh listing program pencarian sekuensial pada data tidak terurut (linked list). Anda bisa menyimpannya pada file **modul11a2.c**.

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

struct node{
    int value;
    struct node *next;
};

typedef struct node *ptrnode;

ptrnode head = NULL;
int jumnode; //jumlah node

ptrnode insert(int nilai){
    ptrnode p,q;
    p = (ptrnode)malloc(sizeof(struct node));
    p->value = nilai;
    p->next = NULL;
    if(head==NULL) {
        head=p;
        q=head;
    }
    else {
        q = head;
        while(q->next!=NULL) {
            q = q->next;
        }
    }
}
```

```

        }
        q->next = p;
    }
    return(head);
}

void isi_data(){
    int k;
    printf("input jumlah node: ");
    scanf("%d",&jumnode);
    for(int j=1;j<=jumnode;j++){
        printf("input data ke-%d :",j);
        scanf("%d", &k);
        head=insert(k);
    }
}

int search(int x){ //x adalah nilai yang dicari
    int j = 1;
    ptrnode tmp=head;
    while(tmp != NULL){
        if(x==tmp->value){
            return j;
        }
        else{
            tmp = tmp->next;
            j++;
        }
    }
    return -1; //jika tidak ada yang dicari return -1
}

void bersihkan_memori(){
    while(head != NULL){
        ptrnode tmp = head;

```

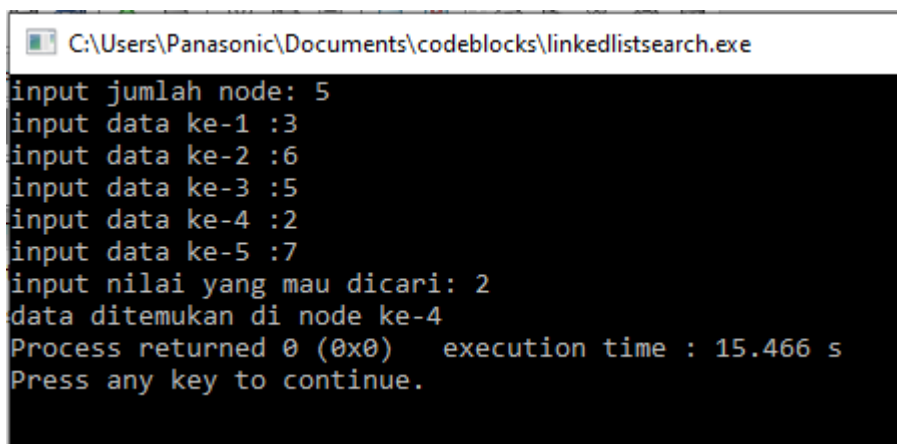
```

        head = head->next;
        tmp->next = NULL;
        free(tmp);
    }
}

void main(){
    isi_data();
    int x;
    printf("input nilai yang mau dicari: ");
    scanf("%d",&x);
    if (search(x) == -1) printf("data tidak
ditemukan");
    else printf("data ditemukan di node ke-
%d",search(x));
    bersihkan_memori();
}

```

Jika kode di atas dijalankan maka outputnya sebagai berikut:



```

C:\Users\Panasonic\Documents\codeblocks\linkedlistsearch.exe
input jumlah node: 5
input data ke-1 :3
input data ke-2 :6
input data ke-3 :5
input data ke-4 :2
input data ke-5 :7
input nilai yang mau dicari: 2
data ditemukan di node ke-4
Process returned 0 (0x0)   execution time : 15.466 s
Press any key to continue.

```

## A.2. Pencarian Sekuensial Pada Data Terurut

Pada pencarian sekuensial pada data yang terurut, proses pencarian adalah proses iterasi yang dimulai dari data indeks pertama pada array sampai data yang dicari ditemukan atau nilai yang dicari sudah melebihi atau kurang dari nilai data pada indeks array yang dikunjungi atau sampai data pada indeks terakhir. Jika diterapkan pada linked list, maka pencarian dimulai dari head atau node pertama hingga node terakhir atau tail, sampai data yang dicari ditemukan atau nilai yang dicari sudah melebihi atau kurang dari nilai data pada node yang dikunjungi atau sampai data pada node tail.

Dari pernyataan di atas, terdapat 3 kondisi dimana proses pencarian berhenti yaitu:

1. Jika data yang dicari ditemukan
2. Jika nilai data pada indeks array atau node yang sedang dikunjungi sudah melebihi atau kurang dari nilai yang dicari (tergantung data terurut menaik atau menurun).
3. Jika sudah sampai di data indeks array atau node terakhir.

Modifikasilah file **modul11a1.c** dan file **modul11a2.c** supaya dapat mengakomodasi pencarian sekuensial pada data yang terurut, baik urut menaik atau menurun.

## B. Pencarian Biner

Pencarian biner atau pencarian bagi dua hanya bisa dilakukan pada data yang terurut. Pencarian dilakukan dengan cara mengecek elemen tengah. Jika data di elemen tengah sama dengan yang dicari, maka ditemukan. Jika data di elemen tengah lebih besar dari data yang dicari, maka pencarian dilanjutkan di bilah kiri, jika data di elemen tengah lebih kecil dari data yang dicari, maka pencarian dilanjutkan di bilah kanan. Begitu seterusnya hingga data yang dicari ditemukan atau semua elemen sudah diperiksa.

Performa dari pencarian biner  $O(\log n)$  jauh lebih cepat dibandingkan pencarian sekuensial  $O(n)$ . Fungsi pencarian biner dapat dinyatakan sebagai fungsi rekursif atau iteratif. berikut ini adalah fungsi pencarian biner secara iteratif pada array. Simpanlah pada file **modul11b1.c**

```
#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data ascending: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

int binary_search(int data[], int size, int x){
    int L = 0;
```

```

    int H = size-1;
    int M = -1;
    int index = -1;

    while(L<=H) {
        M = (L+H)/2;
        if(data[M]==x) return M;
        else{
            if(data[M] < x) L = M + 1;
            else H = M - 1;
        }
    }
    return -1;
}

void main(){
    int data[MAX];
    int size; //ukuran array
    int x;
    fill_data(data,&size);
    printf("nilai yang mau dicari: ");
    scanf("%d", &x);
    if(binary_search(data,size,x)==-1) printf("tidak
ditemukan");
    else printf("ditemukan pada indeks ke-
%d",binary_search(data,size,x));
}

```

Jika program di atas dijalankan maka kurang lebih outputnya seperti ini:

```

C:\Users\Panasonic\Documents\codeblocks\arraysearchbinary.exe
Input ukuran array (max 100): 6
Input data ascending: 9 13 15 17 21 29
nilai yang mau dicari: 29
ditemukan pada indeks ke-5
Process returned 26 (0x1A)   execution time : 32.851 s
Press any key to continue.

```



Pada linked list, pencarian biner dapat diterapkan juga namun perlu runtime yang lebih lama pada pencarian node tengah. Berikut adalah contoh listing program untuk pencarian biner pada single linked list. Simpanlah pada file **modul11b2.c**

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

struct node{
    int value;
    struct node *next;
};

typedef struct node *ptrnode;

ptrnode head,tail = NULL;
int jumnode;

ptrnode insert(int nilai){
    ptrnode p;
    p = (ptrnode)malloc(sizeof(struct node));
    p->value = nilai;
    p->next = NULL;
    if(head==NULL) {
        head=p;
        tail=head;
    }
    else {
        tail = head;
        while(tail->next!=NULL) {
            tail = tail->next;
        }
        tail->next = p;
        tail = p;
    }
    return(head);
}
```

```

void isi_data(){
    int j,k;
    printf("input jumlah node (data harusurut
ascending): ");
    scanf("%d",&jumnode);
    for(j=1;j<=jumnode;j++){
        printf("input data ke-%d :",j);
        scanf("%d", &k);
        head=insert(k);
    }
}

ptrnode middle(ptrnode start, ptrnode last){
//untuk mendapatkan node tengah
    if (start == NULL) return NULL;
    ptrnode slow = start;
    ptrnode fast = start -> next;
    while (fast != last){
        fast = fast -> next;
        if (fast != last){
            slow = slow -> next;
            fast = fast -> next;
        }
    }
    return slow;
}

ptrnode binarySearch(int x){
    ptrnode start = head;
    ptrnode last = NULL;
    do{
        // temukan node tengah
        ptrnode mid = middle(start, last);
        // jika node tengah NULL

```

```

        if (mid == NULL) return NULL;
        // Jika x ditemukan di node tengah
        if (mid -> value == x) return mid;
        // Jika nilai x lebih dari node tengah
        else if (mid -> value < x) start = mid ->
next;

        // Jika nilai x kurang dari node tengah
        else last = mid;
    } while (last == NULL || last != start);
    // jika tidak ditemukan
    return NULL;
}

void bersihkan_memori() {
    while(head != NULL) {
        ptrnode tmp = head;
        head = head->next;
        tmp->next = NULL;
        free(tmp);
    }
}

void main() {
    isi_data();
    int x;
    printf("input nilai yang mau dicari: ");
    scanf("%d", &x);
    if(binarySearch(x)==NULL) printf("data tidak
ditemukan");
    else printf("data ditemukan");
    bersihkan_memori();
}

```

Jika kode di atas dijalankan maka outputnya seperti ini:

```
C:\Users\Panasonic\Documents\codeblocks\linkedlistsearchbinary.exe
input jumlah node (data harusurut ascending): 6
input data ke-1 :2
input data ke-2 :4
input data ke-3 :6
input data ke-4 :8
input data ke-5 :9
input data ke-6 :13
input nilai yang mau dicari: 9
data ditemukan
Process returned 0 (0x0)   execution time : 15.916 s
Press any key to continue.
```

pada doubly linked list arah pengunjungan node bisa ke kiri maupun ke kanan. Performa pencarian biner pada linked list tidak bisa secepat pada array karena penentuan node yang akan dikunjungi tidak menggunakan random access seperti array. Runtime pencarian biner pada doubly-linked list adalah  $O(n \log n)$ , dibandingkan runtime pencarian sekuensial pada array sebesar  $O(n)$ . Namun dengan sedikit modifikasi, performa pencarian biner pada doubly linked list dapat ditingkatkan menjadi  $O(n)$ .

### 11.5 Penugasan

1. Modifikasilah file **modul11a1.c** dan file **modul11a2.c** supaya dapat mengakomodasi pencarian sekuensial pada data yang terurut, baik urut menaik atau menurun.
2. Buat program untuk pencarian data students berisi **int nim, char nama[50]** dengan struktur array. pencarian bisa dilakukan secara sekuensial/biner dengan berdasarkan nim atau berdasarkan nama.
3. Buat program untuk pencarian data students berisi **int nim, char nama[50]** dengan struktur linked list. pencarian bisa dilakukan secara sekuensial/biner dengan berdasarkan nim atau berdasarkan nama.

## MODUL 12: PENGURUTAN

---

### 12.1 Deskripsi Singkat

Proses pengurutan adalah proses menyusun kembali data dengan aturan tertentu. Data dapat disusun kembali baik dengan urutan menaik (ascending) maupun urutan menurun (descending). Pada modul ini akan dibahas tentang pengurutan data pada data bertipe array menggunakan beberapa algoritma dasar.

### 12.2 Tujuan Praktikum

- 1) Mahasiswa mampu menerapkan pengurutan menggunakan metode insertion sort, selection sort, merge sort, bubble sort pada array menggunakan bahasa C
- 2) Mahasiswa mampu menerapkan pengurutan pada data yang lebih kompleks.

### 12.3 Material Praktikum

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

### 12.4 Kegiatan Praktikum

#### A. Insertion Sort

Pada insertion sort, proses pengurutan dilakukan seperti mengurutkan kartu pada satu tangan. Untuk menemukan posisi yang banar, maka satu persatu kartu yang ada di tangan harus dibandingkan secara berurutan. Berikut adalah contoh listing program untuk melakukan selection sort. Anda dapat menyimpannya pada file **modul12a.c**

```
#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

void tampil_data(int data[], int size){
    for(int i=0;i<size;i++) printf("%d ",data[i]);
```

```

        printf("\n");
    }

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void main(){
    int data[MAX];
    int size; //ukuran array yang dipakai
    fill_data(data,&size);
    insertionSort(data,size);
    printf("data setelah diurutkan:\n");
    tampil_data(data,size);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

C:\Users\Panasonic\Documents\codeblocks\insertionsortarray.exe
Input ukuran array (max 100): 10
Input data: 12 14 11 90 45 23 56 78 79 20
data setelah diurutkan:
11 12 14 20 23 45 56 78 79 90

Process returned 10 (0xA)   execution time : 45.122 s
Press any key to continue.

```

## B. Selection Sort

Pada selection sort, proses pengurutan dilakukan dengan cara mencari nilai data terkecil atau terbesar pada setiap perulangan dan menempatkannya pada posisi yang sesuai. Selection sort merupakan kombinasi dari searching dan sorting. Berikut adalah program untuk melakukan selection sort. Anda bisa menyimpannya pada file **modul12b.c**.

```
#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

void tampil_data(int data[], int size){
    for(int i=0;i<size;i++) printf("%d ",data[i]);
    printf("\n");
}

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void selection_sort(int data[], int size){
    for(int step = 0; step < size - 1; step++){
        int min_idx = step;
        for(int i = step + 1; i < size; i++)
            if(data[i] < data[min_idx])
                min_idx = i;
        swap(&data[min_idx], &data[step]);
    }
}
```

```

    }
}

void main() {
    int data[MAX];
    int size; //ukuran array yang dipakai
    fill_data(data,&size);
    selection_sort(data,size);
    printf("data setelah diurutkan:\n");
    tampil_data(data,size);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

C:\Users\Panasonic\Documents\codeblocks\selectionsortarray.exe
Input ukuran array (max 100): 8
Input data: 1 6 3 9 8 2 5 4
data setelah diurutkan:
1 2 3 4 5 6 8 9

Process returned 10 (0xA)   execution time : 33.434 s
Press any key to continue.

```

### C. Merge Sort

Merge sort dirancang untuk memenuhi kebutuhan pengurutan data yang tidak memungkinkan untuk ditampung dalam memori komputer yang terbatas karena ukuran data sangat besar. Merge sort menggunakan prinsip divide and conquer yaitu dengan memecah data menjadi 2 bagian, mengurutkan setiap bagian, kemudian menggabungkan kembali hasil pengurutan kedua bagian (merge). Berikut adalah contoh listing program merge sort pada array. Anda bisa menyimpannya pada file **modul12c.c**.

```

#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

```



```

    }
}

void tampil_data(int data[], int size){
    for(int i=0;i<size;i++) printf("%d ",data[i]);
    printf("\n");
}

void merge(int arr[], int l, int m, int r){
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2){
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1){
        arr[k] = L[i];
        i++;
        k++;
    }

```

```

    }
    while (j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r){
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void main(){
    int data[MAX];
    int size; //ukuran array yang dipakai
    fill_data(data,&size);
    mergeSort(data,0,size-1);
    printf("data setelah diurutkan:\n");
    tampil_data(data,size);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

C:\Users\Panasonic\Documents\codeblocks\mergesortarray2.exe
Input ukuran array (max 100): 10
Input data: 12 10 11 90 100 103 99 110 70 65
data setelah diurutkan:
10 11 12 65 70 90 99 100 103 110

Process returned 10 (0xA)   execution time : 39.663 s
Press any key to continue.

```

## Bubble Sort

Pada bubble sort, proses pengurutan dilakukan dengan cara membandingkan satu data dengan data berikutnya, jika lebih kecil maka akan ditukar urutannya. Berikut adalah fungsi pengurutan dengan bubble sort. Anda bisa menyimpannya pada file **modul12d.c**.

```
#include <stdio.h>
#define MAX 100 //ukuran maksimum array

void fill_data(int data[], int *size){ //mengisi data
    printf("Input ukuran array (max 100): ");
    scanf("%d", size);
    printf("Input data: ");
    for(int i=0;i<*size;i++){
        scanf("%d",&data[i]);
    }
}

void tampil_data(int data[], int size){
    for(int i=0;i<size;i++) printf("%d ",data[i]);
    printf("\n");
}

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void bubbleSort(int arr[], int n){
    int i, j;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}
```

```

void main() {
    int data[MAX];
    int size; //ukuran array yang dipakai
    fill_data(data,&size);
    bubbleSort(data,size);
    printf("data setelah diurutkan:\n");
    tampil_data(data,size);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

C:\Users\Panasonic\Documents\codeblocks\bubblesortarray.exe
Input ukuran array (max 100): 6
Input data: 12 15 1 43 90 77
data setelah diurutkan:
1 12 15 43 77 90

Process returned 10 (0xA)   execution time : 18.448 s
Press any key to continue.

```

## 12.5 Penugasan

1. Modifikasilah file **modul12a.c**, **modul12b.c**, **modul12c.c**, dan **modul12d.c** supaya dapat juga mengakomodasi pengurutan data menurun (descending).
2. Gabungkan keempat metode sorting tersebut, kemudian buatlah sebuah menu sehingga pengguna dapat memilih metode pengurutan yang diinginkan dan juga memilih urutan menaik atau menurun. Kira-kira tampilan menu sebagai berikut:

```

###PROGRAM SORTING DATA###
Input Jumlah data = 10
Metode Sorting yang tersedia
1. Insertion Sort
2. Selection Sort
3. Merge Sort
4. Bubble Sort
Pilih Metode Sorting (1/2/3/4): 4
Pilih pengurutan Naik/Turun(N/T): T
Input Data Anda: 1 3 2 4 5 7 6 8 10 9
Menjalankan sorting dengan metode Bubble Sort
Pilihan pengurutan Turun

```

Data setelah diurutkan: 10 9 8 7 6 5 4 3 2 1

3. Buat program untuk input dan pengurutan data students berisi **int nim**, **char nama[50]**, **int nilai** dengan struktur array. Pengurutan bisa berdasarkan nim, nama, atau nilai terserah pada pilihan pengguna.

## MODUL 13: GRAPH

---

### 13.1 Deskripsi Singkat

Graph adalah kumpulan node (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan edge (garis). Graph dapat digunakan untuk merepresentasikan objek-objek diskrit (direpresentasikan sebagai node) dan hubungan antara objek-objek tersebut (direpresentasikan sebagai edge). Secara matematis, graph dinyatakan sebagai  $G = (V, E)$  dimana  $G$  adalah Graph,  $V$  adalah Vertex atau Node atau Simpul, atau Titik dan  $E$  adalah Edge atau Busur atau Garis.

Terdapat berbagai jenis graph seperti Graph berarah (directed graph), Graph tak berarah (undirected graph), graph berbobot (weighted graph), dan graph tak berbobot (unweighted Graph). Baik Graph berarah maupun graph tak berarah dapat memiliki bobot maupun tidak memiliki bobot.

Graph dapat direpresentasikan dengan 2 cara yaitu dengan Adjacency Matrix dan Adjacency List. Adjacency Matrix mereprestasikan graph ke dalam sebuah matriks (array 2 dimensi) dan Adjacency List merepresentasikan graph dengan sebuah array dari linked list. Pada Praktikum kali ini akan dibahas representasi graph menggunakan adjacency list.

### 13.2 Tujuan Praktikum

- 1) Mahasiswa mampu merepresentasikan struktur data graph berarah tak berbobot dengan menggunakan adjacency list
- 2) Mahasiswa mampu merepresentasikan struktur data graph berarah berbobot dengan menggunakan adjacency list

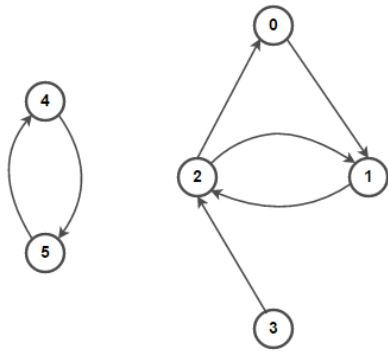
### 13.3 Material Praktikum

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

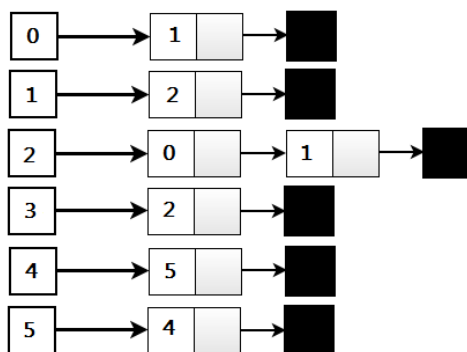
### 13.4 Kegiatan Praktikum

#### A. Graph Berarah tak berbobot

Pada representasi graph menggunakan adjacency list, setiap node pada graph diasosiasikan dengan kumpulan node lain yang terhubung dengan node tersebut. Dengan kata lain setiap node menyimpan list node lain yang terhubung. Sebagai contoh graph di bawah ini:



Memiliki gambaran representasi adjacency list sebagai berikut:



Berikut ini adalah contoh listing program untuk merepresentasikan graph tersebut. Anda dapat menyimpannya pada file **modul13a.c**.

```

#include <stdio.h>
#include <stdlib.h>
#define N 6 //misal maksimum node adalah 6

// Struktur data untuk menyimpan adjacency list dari
node pada graph
struct Node{
    int dest;
    struct Node* next;
};
typedef struct Node *ptrNode;

//Struktur data untuk menyimpan onjek graph
struct Graph{
    // array pointer ke node untuk representasi
adjacency list

```

```

    ptrNode head[N];
};
typedef struct Graph *ptrGraph;

// Struktur data untuk menyimpan edge graph
struct Edge {
    int src, dest;
};

// Fungsi untuk membuat adjacency list dari edge
tertentu
ptrGraph createGraph(struct Edge edges[], int n){
    // alokasi memori untuk menyimpan struktur data
    graph
    ptrGraph graph = (ptrGraph)malloc(sizeof(struct
    Graph));

    // inisialisasi semua pointer head ke null
    for (int i = 0; i < N; i++) {
        graph->head[i] = NULL;
    }

    // menambahkan edge satu demi satu
    for (int i = 0; i < n; i++)
    {
        // ambil source dan destination dari node
        int src = edges[i].src;
        int dest = edges[i].dest;

        // buat node baru dari src ke dest
        ptrNode newNode =
        (ptrNode)malloc(sizeof(struct Node));
        newNode->dest = dest;

        // point node baru ke head
        newNode->next = graph->head[src];
    }
}

```



```

        // point head ke node baru
        graph->head[src] = newNode;
    }

    return graph;
}

// Fungsi print representasi adjacency list
void printGraph(ptrGraph graph){
    int i;
    for (i = 0; i < N; i++){
        // print node dan semua yang terhubung
        ptrNode ptr = graph->head[i];
        while (ptr != NULL){
            printf("(%d -> %d)\t", i, ptr->dest);
            ptr = ptr->next;
        }

        printf("\n");
    }
}

void main(){
    //input array pasangan dari x ke y
    struct Edge edges[] =
        {{ 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 }, { 3, 2 },
        { 4, 5 }, { 5, 4 }};

    // menghitung jumlah edge
    int n = sizeof(edges)/sizeof(edges[0]);

    // membuat graph
    ptrGraph graph = createGraph(edges, n);
}

```

```

    // print graph
    printGraph(graph);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

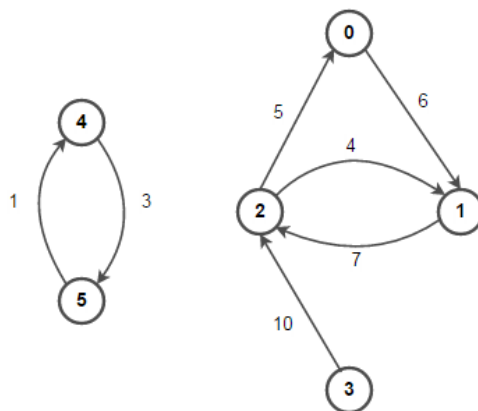
"D:\ibnu\Modul Strukdat\adjlist.exe"
(0 -> 1)
(1 -> 2)
(2 -> 1)      (2 -> 0)
(3 -> 2)
(4 -> 5)
(5 -> 4)

Process returned 10 (0xA)   execution time : 0.016 s
Press any key to continue.

```

## B. Graph berarah berbobot

Pada graph berbobot, setiap edge memiliki weight. Misal kita akan mereprestasikan graph di bawah ini:



Maka pada struktur node selain kita menyimpan destination, kita juga menyimpan weight. Berikut adalah contoh implementasinya. Anda bisa menyimpannya pada file **modul13b.c**.

```

#include <stdio.h>
#include <stdlib.h>
#define N 6 //misal maksimum node adalah 6

// Struktur data untuk menyimpan adjacency list dari
node pada graph

```

```

struct Node{
    int dest, weight;
    struct Node* next;
};
typedef struct Node *ptrNode;

//Struktur data untuk menyimpan onjek graph
struct Graph{
    // array pointer ke node untuk representasi
    adjacency list
    ptrNode head[N];
};
typedef struct Graph *ptrGraph;

// Struktur data untuk menyimpan edge graph
struct Edge {
    int src, dest, weight;
};

// Fungsi untuk membuat adjacency list dari edge
tertentu
ptrGraph createGraph(struct Edge edges[], int n){
    // alokasi memori untuk menyimpan struktur data
    graph
    ptrGraph graph = (ptrGraph)malloc(sizeof(struct
    Graph));

    // inisialisasi semua pointer head ke null
    for (int i = 0; i < N; i++) {
        graph->head[i] = NULL;
    }

    // menambahkan edge satu demi satu
    for (int i = 0; i < n; i++){
        // ambil source dan destination dari node
        int src = edges[i].src;

```

```

        int dest = edges[i].dest;
        int weight = edges[i].weight;

        // buat node baru dari src ke dest
        ptrNode newNode =
(ptrNode)malloc(sizeof(struct Node));
        newNode->dest = dest;
        newNode->weight = weight;

        // point node baru ke head
        newNode->next = graph->head[src];

        // point head ke node baru
        graph->head[src] = newNode;
    }

    return graph;
}

// Fungsi print representasi adjacency list
void printGraph(ptrGraph graph){
    int i;
    for (i = 0; i < N; i++){
        // print node dan semua yang terhubung
        ptrNode ptr = graph->head[i];
        while (ptr != NULL){
            printf("%d -> %d (%d)\t", i, ptr->dest,
ptr->weight);
            ptr = ptr->next;
        }

        printf("\n");
    }
}

void main(){

```

```

//input array pasangan dari x ke y
struct Edge edges[] =
    {{ 0, 1, 6 }, { 1, 2, 7 }, { 2, 0, 5 }, { 2, 1, 4 },
    { 3, 2, 10 }, { 4, 5, 1 }, { 5, 4, 3 }};

// menghitung jumlah edge
int n = sizeof(edges)/sizeof(edges[0]);

// membuat graph
ptrGraph graph = createGraph(edges, n);

// print graph
printGraph(graph);
}

```

Berikut adalah tampilan ketika program dijalankan:

```

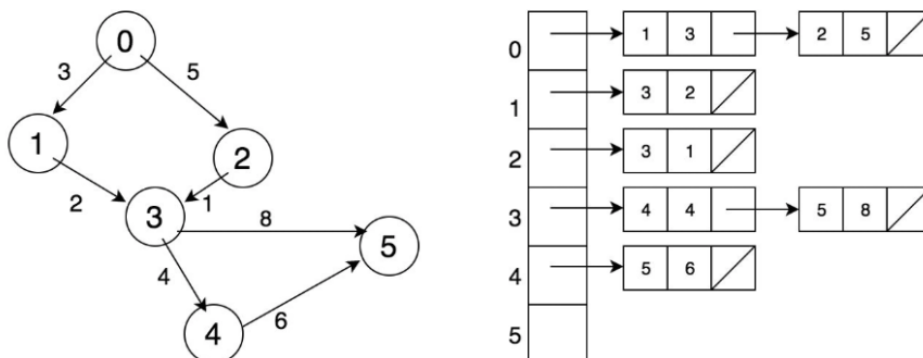
"D:\ibnu\Modul Strukdat\adjlistweighted.exe"
0 -> 1 (6)
1 -> 2 (7)
2 -> 1 (4)      2 -> 0 (5)
3 -> 2 (10)
4 -> 5 (1)
5 -> 4 (3)

Process returned 10 (0xA)   execution time : 0.018 s
Press any key to continue.

```

### 13.5 Penugasan

1. Buatlah program untuk untuk graph berarah dan berbobot dan representasinya berikut ini:



2. Buat program untuk contoh representasi graph tak berarah.

## MODUL 14: Tugas Akhir

Pada tugas akhir ini anda diminta untuk membuat sebuah project pemrograman. Pada project ini, anda diminta untuk menerapkan penggunaan struktur data dengan mengimplementasikan sebuah *contact book* atau buku kontak.

Anda boleh mendesain sendiri struktur data yang anda anggap baik, atau anda boleh menggunakan struktur data yang telah anda pelajari sebelumnya. Semua struktur data pada program anda harus didesain dan diimplementasikan sendiri. Jadi anda tidak diperbolehkan menggunakan library tertentu untuk struktur datanya. Pastikan source code dapat dikompilasi tanpa error.

### A. Contact Book

Sebuah *contact book* berisi data-data yang anda butuhkan ketika akan menghubungi sebuah teman. Data tersebut dapat berisi bermacam-macam atribut seperti nama, umur, jenis kelamin, nomor telepon, email, tanggal lahir, dan lain-lain. Lihat tabel 1 untuk contoh isi *contact book*. Tantangan utama dari tugas akhir ini adalah anda mampu menggunakan struktur data yang tepat dan efisien yang dapat digunakan untuk mendukung berbagai fungsi atau fitur sebuah *contact book*.

Tabel 1. Contoh struktur *contact book*

Nama	Umur	Jenis Kelamin	Nomor Telepon	Email	...
Andi Pawanari	17	L	081341780123	<a href="mailto:andip@gmail.com">andip@gmail.com</a>	...
Bebi Putri	16	P	085678912345	<a href="mailto:bebputri@yahoo.com">bebputri@yahoo.com</a>	...
Catur	15	L	085831124358	<a href="mailto:caturdoang@gmail.com">caturdoang@gmail.com</a>	...
...	...	...	...	...	...

### B. Fitur contact book

Fitur-fitur pada *contact book* pada project ini dibagi menjadi 2 kategori yaitu fungsi dasar dan fungsi opsional. Fungsi dasar adalah fitur yang wajib anda bangun sedangkan fitur opsional dapat anda kembangkan untuk menambah nilai akhir.

Berikut ini adalah deskripsi fungsi dasar:

1. **User Interface:** menampilkan menu utama yang dapat anda pilih dan daftar kontak
2. **Search:** Pencarian kontak dengan atribut tertentu, menampilkan seluruh atribut lain jika ditemukan
3. **Insert:** Menambah kontak baru pada buku kontak
4. **Delete:** Menghapus kontak dengan atribut tertentu
5. **Edit:** Mengupdate atribut kontak tertentu
6. **Sorting:** Melakukan pengurutan kontak berdasarkan atribut tertentu. Fungsi ini dapat dijalankan pada semua atribut sesuai kemauan pengguna.

Dan berikut ini adalah deskripsi fungsi opsional

1. **Save:** menyimpan data *contact book* yang ada di memori ke dalam sebuah file tertentu. Anda dapat menyimpan data kontak dengan format file anda sendiri.
2. **Load:** Membaca data kontak dari file yang telah tersimpan dengan fungsi *save*. Format file harus konsisten dengan fungsi *save*.
3. **Max (atau Min):** menemukan atribut maksimum atau minimum dari atribut tertentu
4. **Average:** menemukan rata-rata dari atribut tertentu yang bertipe numerik
5. **Undo:** melakukan undo sebuah action, baik itu undo insert, delete, atau edit
6. **Redo:** melakukan redo action yang di-undo sebelumnya
7. **And:** Pencarian dengan dua atribut atau lebih, temukan kontak yang match dengan semua kriteria
8. **Or:** Pencarian dengan dua atribut atau lebih, temukan kontak yang match dengan paling sedikit satu kriteria
9. **Wildcard search:** pencarian dengan wildcard (\*). Misal jika yang dicari nomor telepon 0813\* maka akan ditampilkan daftar kontak dengan nomor awal kontak 0813. Fungsi ini dapat diterapkan pada atribut bertipe string.



10. **Konektivitas:** Misal hubungan antar kontak juga tersimpan dalam *contact book*. Relasi ini dapat disimpan di struktur data lain/tambahan. Misal dapat dilihat pada ilustrasi tabel 2 berikut ini:

Tabel 2. Relasi antar kontak dalam *contact book*

Kontak	Anda	A	B	C	...
Anda	1	1	1	1	...
A	1	1	1	0	...
B	1	1	1	0	...
C	1	0	0	1	...
...	...	...	...	...	...

Misal angka 1 melambangkan memiliki hubungan dan 0 adalah tidak. Jika kontak C ingin menghubungi kontak B, karena C tidak memiliki kontak B, maka C harus menghubungi Anda dahulu untuk mendapatkan kontak B. Contoh lainnya B ingin menghubungi kontak C, namun karena B tidak memiliki kontak C, maka B bisa menghubungi A atau menghubungi Anda dahulu untuk mendapatkan kontak C.

Fungsi ini mengambil input 2 buah kontak, kemudian menentukan apakah kontak pertama dapat menghubungi kontak kedua tersebut secara langsung, jika ya maka selesai. Jika tidak bisa maka melalui kontak mana saja kontak lain tersebut dapat dihubungi.

### C. Penilaian

Grade penilaian tugas akhir ditunjukkan pada tabel 3.

Tabel 3. Penilaian tugas akhir

Kategori	Fitur	Grade
Fungsi Dasar	User Interface	10%
	Search	

	Insert	60%
	Delete	
	Edit	
	Sorting	
Fungsi Opsional	Save	5%
	Load	5%
	Max Min	3%
	Average	3%
	Undo	5%
	Redo	5%
	And	5%
	Or	5%
	Wildcard	10%
	Konektivitas	20%
Lain-lain		1-20%