



Modul Pelatihan Pemrograman  
**Pemberdayaan Umat Berkelanjutan**

# Logika & Algoritma (Bahasa C)



أَظْلُبُ الْعِلْمَ مِنَ الْمَهْدِ إِلَى اللَّحْدِ

"Tuntutlah ilmu sejak dari buaian sampai ke liang lahad"

# Kata Pengantar

Assalamu'alaikum warahmatullahi wabarakatuh

Puji syukur kehadiran Allah SWT karena atas izinnya kami berkesempatan untuk menyusun modul pelatihan ini. Shalawat serta salam kepada Nabi Muhammad SAW yang telah membawa kita dari zaman kebodohan menuju zaman terang-benderang yang penuh dengan ilmu pengetahuan. Saat ini perubahan di dunia pemrograman berjalan begitu cepat.

Perubahan ini menuntut kita untuk segera menyesuaikan kurikulum pelatihan di PUB agar tidak ketinggalan zaman. Kalau tidak, itu akan menjadi bom waktu bagi Program Beasiswa PUB. Mau tidak mau, kita harus mengikuti perubahan itu.

Kami memutuskan untuk menyusun modul baru ini sebagai penunjang bagi mahasiswa PUB dalam mengikuti kurikulum baru tersebut. Kami juga memohon dukungan dari semua pihak untuk lebih peduli pada sistem pendidikan di PUB yang sedang sangat membutuhkan perhatian, sehingga bisa menjadi lebih baik dan berkualitas. Terima kasih kepada semuanya yang telah bersedia membantu dalam misi ini.

Wassalamu'alaikum warahmatullahi wabarakatuh

Bandung, 2024  
Koordinator Divisi Pendidikan  
Pemberdayaan Umat Berkelanjutan

**Muhamad Alwan Fadhlurrohmam**

## Silabus

Pertemuan	Bab	Materi
Pertemuan 1	Bab 1: Pemahaman Dasar Logika dan Algoritma	<ol style="list-style-type: none"> <li>1. Pengertian algoritma</li> <li>2. Sejarah algoritma</li> <li>3. Contoh algoritma</li> <li>4. Kriteria algoritma yang baik</li> </ol>
	Bab 2: Flowchart dan Pseudocode	<ol style="list-style-type: none"> <li>1. Pengertian flowchart</li> <li>2. Simbol-simbol flowchart</li> <li>3. Contoh-contoh flowchart</li> <li>4. Pengertian pseudocode</li> <li>5. Contoh-contoh pseudocode</li> </ol>
	Bab 3: Pemrograman	<ol style="list-style-type: none"> <li>1. Pemrograman</li> <li>2. Mengapa harus belajar pemrograman?</li> <li>3. Bahasa pemrograman</li> <li>4. Struktur dasar pemrograman</li> </ol>
Pertemuan 2	Bab 4: Konsep Dasar Pemrograman Bahasa C	<ol style="list-style-type: none"> <li>1. Fitur-Fitur dalam Bahasa C</li> <li>2. Menyiapkan compiler</li> <li>3. Media Pemrograman Bahasa C Melalui IDE</li> <li>4. Menyiapkan Visual Studio Code</li> </ol>
	Bab 5: Struktur Pemrograman Bahasa C	<ol style="list-style-type: none"> <li>1. Preprocessor</li> <li>2. Fungsi Main</li> <li>3. Comment</li> </ol>
	Bab 6: Tipe Data dan Variabel	<ol style="list-style-type: none"> <li>1. Tipe data</li> <li>2. Variable dan Konstanta</li> <li>3. Escape Sequence</li> <li>4. Input Output Program</li> </ol>

Pertemuan 3	Bab 7: Operator	<ol style="list-style-type: none"> <li>1. Pengertian</li> <li>2. Jenis-jenis <ol style="list-style-type: none"> <li>a. Penugasan</li> <li>b. Perbandingan</li> <li>c. Logika</li> </ol> </li> </ol>
Pertemuan 4 & 5	Bab 8: Percabangan (Decision)	<ol style="list-style-type: none"> <li>1. Aturan Pembuatan Kondis</li> <li>2. Jenis-jenis <ol style="list-style-type: none"> <li>a. If - else</li> <li>b. If - else if - else</li> <li>c. Nested if (if dalam if)</li> <li>d. Switch case</li> <li>e. Ternary operator</li> </ol> </li> </ol>
Pertemuan 6	Bab 9: Loop	<ol style="list-style-type: none"> <li>1. Pengertian</li> <li>2. Jenis-jenis <ol style="list-style-type: none"> <li>a. For</li> <li>b. While</li> <li>c. Do-while</li> </ol> </li> <li>3. Nested loop</li> </ol>
Pertemuan 7	Bab 10: Jump Statement	<ol style="list-style-type: none"> <li>1. Pengertian</li> <li>2. Jenis-jenis</li> </ol>
Pertemuan 8	Ujian Pertengahan (pilihan ganda dan essay)	
Pertemuan 9	Bab 11: Array	<ol style="list-style-type: none"> <li>1. Pengertian</li> <li>2. Contoh-contoh</li> </ol>
Pertemuan 10	Bab 12: Function	<ol style="list-style-type: none"> <li>1. Pengertian</li> <li>2. Parameter &amp; Argumen</li> <li>3. Pengembalian</li> </ol>
Pertemuan 11	Bab 12 : Function	<ol style="list-style-type: none"> <li>1. Penulisan Prosedur</li> <li>2. Penulisan Function</li> </ol>
Pertemuan 12	Bab 13: Input & Output File	<ol style="list-style-type: none"> <li>1. Pengenalan</li> <li>2. Membaca &amp; menambahkan file Txt Dalam bahasa c</li> </ol>
Pertemuan 13	Bab 13: Input & Output File	<ol style="list-style-type: none"> <li>1. Update file Txt dalam bahasa c</li> <li>2. Delete file Txt dalam bahasa c</li> </ol>
Pertemuan 14	Bab 14: Windows API	Manipulasi Tampilan

Pertemuan 15	Bab 14: Windows API	Manipulasi Warna
Pertemuan 16	Ujian Akhir (project CRUD berbasis konsol menggunakan Windows API)	

### **Bobot Penilaian Pelatihan Logika & Algoritma (Bahasa C)**

Nilai	Persentase bobot
Tugas	10%
Kuis	20%
Ujian Tengah Semester	30%
Ujian Akhir Semester	40%

## Hak cipta

Hak cipta © 2021 pada Divisi Pendidikan PUB periode 2021/2022.

### **Divisi Pendidikan PUB 2021/2022**

Koordinator : Romi Kusuma Bakti

Anggota :

- |                           |                                |
|---------------------------|--------------------------------|
| 1. Dani Hidayat           | 6. Nushrotummillah Nurul 'Aini |
| 2. Edwar Maulana Sitompul | 7. Seli Deslia                 |
| 3. Hanafi Abdullah        | 8. Tiara Agustin               |
| 4. Jamil Hamdi Harahap    | 9. Yoga Hendrapratama          |
| 5. Mita Zuriati           |                                |

Hak cipta © 2022 pada Divisi Pendidikan PUB periode 2022/2023.

### **Divisi Pendidikan PUB 2022/2023**

Pelatih Instruktur : Romi Kusuma Bakti

Koordinator : Muhammad Fadli Fathurrahman

Anggota :

- |                         |                        |
|-------------------------|------------------------|
| 1. Achmad Farhan Fauzan | 5. Dimas Firmasnsyah   |
| 2. Aliya Rohaya Siregar | 6. M. Azriel Pajarudin |
| 3. Nailly Rina Pribawa  | 7. Tigana Reymansya    |
| 4. Sawaluddin           |                        |

Hak cipta © 2023 pada Divisi Pendidikan PUB periode 2023/2024.

### **Divisi Pendidikan PUB 2023/2024**

Pelatih Instruktur : Anggi Permana

Koordinator : Anwar Juniansyah Harahap

Anggota :

- |                             |                            |
|-----------------------------|----------------------------|
| 1. Aisyah Romaito Siregar   | 5. Peni Julianti           |
| 2. Almia Kusnariati Harahap | 6. Ragil Sadewa Pasaribu   |
| 3. Amelia Siregar           | 7. Raja Salsabila Anatasya |
| 4. Anggi Nauval             | 8. Siti Nurhaliza          |
|                             | 9. Siti Pitriani           |

Hak cipta © 2024 pada Divisi Pendidikan PUB periode 2024/2025.

**Divisi Pendidikan PUB 2024/2025**

Pelatih Instruktur : Anggi Nauval

Koordinator : Muhamad Alwan Fadhlurrohman

Anggota :

- |                               |                              |
|-------------------------------|------------------------------|
| 1. Ahmad Wahyudi Tanjung      | 6. Marseliya Salsa Nur Rizky |
| 2. Arjun Zebua                | 7. Putri Ayu Sahara Lubis    |
| 3. Anwar Juniansyah Harahap   | 8. Puspita Dewi              |
| 4. Dwi Putri Melati Sikumbang | 9. Salim Hidayat             |
| 5. Iqbal Nafis                | 10. Vira Narulita            |

**Hak cipta dilindungi oleh undang-undang.**

**Dilarang menyalin atau memperbanyak sebagian atau seluruh isi buku tanpa izin tertulis dari pemegang hak cipta. Setiap pelanggaran hak cipta akan dituntut dan berhadapan dengan undang-undang yang berlaku**

# Bab 1:

## Pemahaman Dasar Logika dan Algoritma

### A. Pengertian Logika dan Algoritma

#### 1) Logika

Adalah suatu upaya berpikir secara cermat, tepat, dan logis. Maksudnya logis adalah dapat diterima oleh akal.

#### 2) Algoritma

Algoritma adalah langkah – langkah yang terstruktur dan logis untuk memecahkan suatu masalah. Maka dari itu, sebuah algoritma harus logis (masuk akal), sehingga dibutuhkan logika untuk dapat menghasilkan algoritma yang benar.

### B. Sejarah Algoritma

Kata *algoritma* berasal dari kata nama Abu Ja'far Muhammad bin Musa Al-  
(يَمَزْرُؤُ, يُهَيِّو, يُؤَمِّم), ilmuwan Persia (sekarang Iran) yang menulis buku Al-  
من Khawarizmi  
Jabr wal-Muqabalah (Rules of Restoration and Reduction), terbit 825 M.

Beliau hidup di masa **khilafah Abbasiyah**, diperkirakan pada masa khalifah Al-Ma'mun, Al-Mu'tashim, dan Al-Watsiq yang dikenal sebagai masa keemasan ilmu pengetahuan umat Islam berkat translasi buku dan ilmu pengetahuan ke dalam bahasa Arab. Pada masa itu terdapat Baitul Hikmah yang menjadi pusat penelitian, penerjemahan buku ke dalam bahasa Arab, dan juga publikasi ilmu pengetahuan yang dilakukan oleh para cendekiawan muslim tak terkecuali Al-Khawarizmi.

Muhammad bin Musa Al-Khawarizmi dikenal sebagai matematikawan yang **menemukan aljabar** dan juga merupakan **Bapak dari Algoritma**. Bagi sebagian orang yang berprofesi sebagai programmer atau developer tentunya algoritma ini digunakan saat melakukan pengembangan program.

Di belahan Eropa, karyanya banyak ditranslasikan ke dalam bahasa Latin sebagai Algorithmi, Algorismi, Alchawarizmi sehingga di literatur barat Al-Khawarizmi dikenal sebagai Algorizm. Sebutan inilah yang kemudian digunakan untuk menyebutkan konsep algoritma yang ditemukannya perhitungan logaritma yang sekarang banyak dipergunakan secara luas terutama di bidang komputer atau sains dan engineering yang berasal dari hasil pemikiran beliau.



Selain itu matematika biner yang digunakan dalam pemrograman juga didasari oleh konsep algoritma Al-Khawarizmi. Perkembangan yang semakin maju bagi komputer digital dan pemrogramannya tak terlepas dari pemikiran beliau yang menjadi gerbang kemajuan. Kata algoritma sendiri yang kita kenal sekarang merupakan kata yang diambil dari kata algorismi yang dilatinisasi dari namanya.

Di samping algoritma, salah satu kontribusi yang dilakukan oleh Al-Khawarizmi yang cukup besar untuk perkembangan bidang komputer adalah memperkenalkan **angka 0** dalam sistem penomoran Arab, yang nantinya diadaptasi pada bidang komputer. Angka nol sendiri merupakan bagian yang ada dalam kode biner dan merupakan dasar dari pembentukan program komputer.

Angka nol sendiri digunakan kembali oleh George Boole seorang ahli matematika dan logika asal Inggris untuk merumuskan **Aljabar Boolean**. Bahkan aljabar sendiri merupakan salah satu konsep yang ditemukan oleh Al-Khawarizmi. Aljabar boolean memiliki peran penting dalam evolusi digital untuk mewakili bentuk-bentuk logis dan silogisme dengan simbol-simbol aljabar dan logika melalui formula yang beroperasi pada 0 dan 1.

## C. Contoh Algoritma

### **Contoh algoritma yang logis dalam membuat segelas kopi:**

1. Siapkan gelas, sendok, kopi, gula, dan air panas
2. Masukkan kopi, gula, dan air panas ke dalam gelas
3. Aduk dengan sendok sampai rata
4. Kopi siap diminum

Algoritma di atas logis selama tidak dikurangi atau diubah urutannya.

### **Contoh algoritma yang salah:**

1. Masukkan kopi, gula, dan air panas ke dalam gelas
2. Aduk dengan sendok sampai rata
3. Kopi siap diminum

Algoritma di atas tidak logis karena gelas, kopi, gula, dan air panas tidak ada.

### **Contoh algoritma yang salah:**

1. Aduk dengan sendok sampai rata
2. Kopi siap diminum

Algoritma di atas tidak logis karena sendok dan yang diaduk tidak ada.

### **Contoh algoritma yang salah:**

1. Kopi siap diminum

Algoritma di atas tidak logis karena kopi tidak ada.

### **Contoh algoritma yang logis tapi hasilnya salah:**

1. Siapkan gelas, sendok, kopi, gula, dan air panas
2. Masukkan air panas ke dalam gelas
3. Aduk dengan sendok sampai rata
4. Kopi siap diminum

Algoritma di atas **tetap logis** karena proses bisa berjalan sampai selesai meskipun tidak jadi kopi karena kopi dan gula tidak dimasukkan.

### **Contoh algoritma komputer menghitung luas lingkaran:**

1. Komputer menetapkan bahwa pi adalah bilangan 3,14 yang tetap (tidak akan diubah)
2. Komputer menetapkan bahwa jari-jari dan luas adalah bilangan yang belum diketahui
3. Komputer meminta pengguna (user) untuk memasukkan jari-jari (input)
4. Komputer menetapkan bahwa luas adalah  $\pi \times \text{jari-jari} \times \text{jari-jari}$
5. Komputer menampilkan luas kepada pengguna (output)

Seluruh proses di atas berjalan dengan sangat cepat kecuali saat komputer meminta pengguna memasukkan jari-jari, komputer akan menunggu sampai pengguna menekan Enter atau tombol tertentu, baru akan melanjutkan prosesnya.

### **Contoh algoritma yang salah:**

1. Komputer menetapkan bahwa luas adalah  $\pi \times \text{jari-jari} \times \text{jari-jari}$
2. Komputer menampilkan luas kepada pengguna (output)

Proses di atas akan error karena pi dan jari-jari tidak ditetapkan.

## D. Kriteria Algoritma yang Baik

Kriteria algoritma yang baik menurut Donald E. Kanuth:

1. Memiliki input

Program minimal harus memiliki 1 input, baik dari pengguna maupun sudah dideklarasikan dalam kodenya.

2. Memiliki output

Program minimal harus memiliki 1 output.

3. Finite (terbatas)

Program harus ada berakhirnya, bukan berjalan terus-menerus.

4. Definite (pasti)

Program harus jelas arah dan tujuannya, kapan mulai dan kapan berakhirnya, dan jelas logikanya.

5. Efektif dan efisien

Program tidak memakan banyak memori, tidak melakukan hal-hal yang tidak perlu.

## Bab 2:

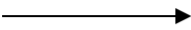


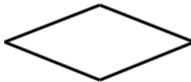


# Flowchart dan Pseudocode





Algoritma dapat disajikan dalam bentuk gambar (simbol-simbol) maupun tulisan. Penyajian dalam bentuk gambar biasanya menggunakan flowchart (diagram alir), sedangkan penyajian algoritma dalam bentuk tulisan biasanya menggunakan pseudocode.

### A. Flowchart

Flowchart (diagram alir) adalah diagram yang menggambarkan langkah-langkah pemecahan masalah dalam bentuk simbol-simbol standar.

#### Simbol-Simbol

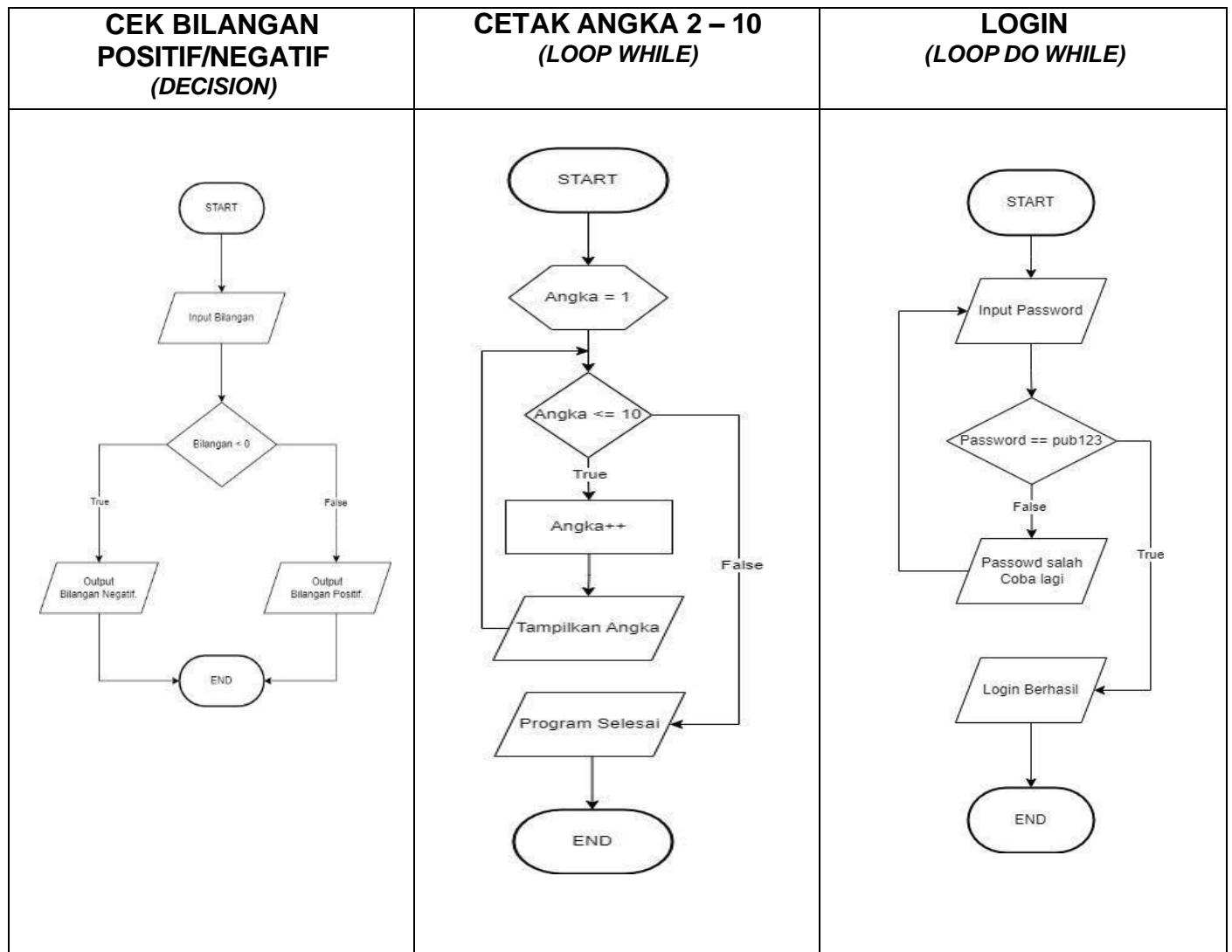
Bentuk	Nama	Deskripsi
	Flowline/arrow (Garis alir/panah)	Menunjukkan arah aliran algoritma, dari satu proses ke proses berikutnya.
	Terminal/terminator	Menunjukkan awal (start) atau akhir (end) sebuah proses.
	Process (Proses/langkah)	Merupakan operasi yang mengubah nilai, bentuk, atau lokasi data.
	Decision (Titik keputusan)	Menunjukkan operasi bersyarat (kondisi) yang menentukan salah satu dari dua jalur yang akan diambil program. Operasi biasanya berupa pertanyaan ya/tidak atau benar/salah.
	Input/output (Masukan/keluaran)	Menunjukkan proses input dan output data, seperti memasukkan data atau menampilkan hasil.
	Annotation(comment) (Anotasi (komentar))	Menunjukkan informasi tambahan tentang langkah dalam program.

	Predefined process (Proses terdefinisi)	Menunjukkan proses yang sudah didefinisikan di tempat lain.
	On-page connector (Penghubung dalam halaman)	Menghubungkan satu proses ke proses lainnya pada halaman yang sama.
	Off-page connector (Penghubung luar halaman)	Menghubungkan satu proses ke proses lainnya pada halaman yang berbeda.
	Preparation or initialization (Penyiapan atau inisialisasi)	Digunakan untuk menunjukkan langkah persiapan atau inisialisasi dalam suatu proses.

#### Ketentuan Pembuatan Flowchart:

1. Harus Memiliki Judul.
  2. Tidak Menggantung (Setiap simbol harus terhubung dan menghubungkan).
- Note:** Kecuali Terminal, On-Page, dan Off-Page.
3. Posisi Start Harus center dan End boleh center atau right (*asalkan tetap di paling bawah*).
  4. Decision -> Memiliki True dan False (True/False rapat dengan arrow atau mengenainya).
  5. Setiap Simbol Max 4 Terhubung/Menghubungkan kecuali Decision (Max 3), Terminal Start/On-Page/Off-Page (Max 1) dan Terminal END (Max 2).
  6. Arrow harus mengenai simbol.

Contoh Flowchart:



## B. Pseudocode

Pseudocode adalah cara penulisan algoritma yang hampir menyerupai bahasa pemrograman tetapi ditulis lebih sederhana dengan menggunakan bahasa yang mudah dipahami oleh manusia.

Contoh pseudocode menghitung luas lingkaran:

1.  $\text{pi} \leftarrow 3,14$
2. Input jari-jari
3.  $\text{luas} \leftarrow \text{pi} \times \text{jari-jari} \times \text{jari-jari}$
4. Output luas

Tanda  $\leftarrow$  artinya menetapkan.

# Bab 3:

## Pemrograman

Pemrograman adalah proses merancang dan membangun program komputer yang dapat dijalankan untuk melakukan tugas tertentu.

Yang dilakukan dalam pemrograman meliputi: menganalisis, menyiapkan algoritma, dan mengimplementasikan algoritma itu dalam bahasa pemrograman yang dipilih (biasanya disebut sebagai *programming* atau *ngoding*).

### A. Mengapa harus belajar pemrograman?

1. Meningkatkan cara berpikir, kreativitas, dan cara memecahkan masalah.
2. Memiliki peluang kerja yang sangat tinggi.
3. Sebagai sarana untuk berdakwah, menegakkan agama Allah Swt., dan membangun kembali kejayaan Islam dalam bidang ilmu pengetahuan dan teknologi.
4. Dapat memudahkan segala urusan umat dengan cara menciptakan aplikasi yang bermanfaat bagi umat.
5. Bisa bekerja dari mana saja, sehingga bisa ke masjid kapan saja tanpa ada atasan yang mengintimidasi.

### B. Bahasa pemrograman

Bahasa pemrograman adalah bahasa yang dimengerti oleh komputer yang digunakan untuk berinteraksi dan memberi perintah-perintah kepada komputer di dalam program yang didefinisikan sesuai kebutuhan.

Ada berbagai macam bahasa pemrograman. Berdasarkan tingkatan, ada 3 tingkatan bahasa pemrograman: bahasa tingkat tinggi, bahasa tingkat menengah, dan bahasa tingkat rendah. Orang yang membuat program komputer disebut dengan *programmer*.



Contoh bahasa pemrograman (urut berdasarkan popularitas):

- ☐ Python
- ☐ Java
- ☐ JavaScript
- ☐ C#
- ☐ PHP
- ☐ C/C++

Untuk memecahkan suatu masalah, ada 3 komponen utama yang komputer akan lakukan: input → proses → output.

Input : Masukan data yang akan diolah menjadi informasi

Proses : Pengolahan data dengan berbagai rumus atau langkah-langkah

Output : Hasil dari proses dan input data

Contoh masalah:

Penjumlahan antara 5 dan 2

Penyelesaian:

Input : 5 dan 2

Proses : 5 ditambah 2

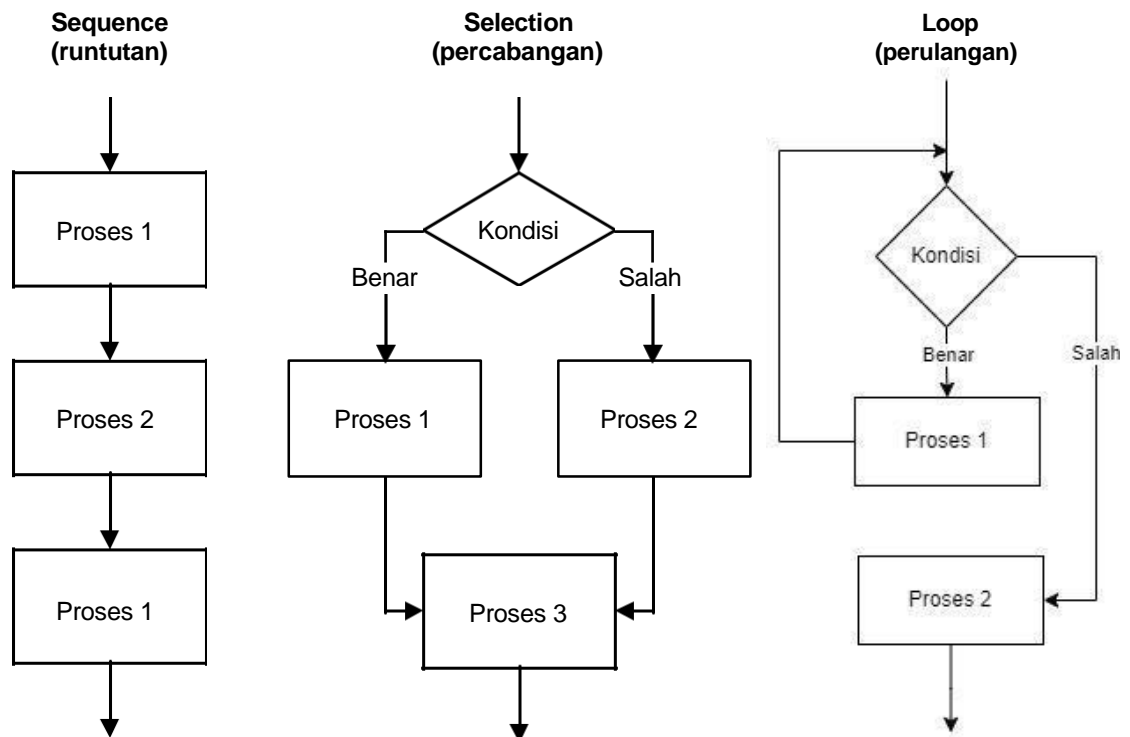
Output : 7

Dalam bahasa pemrograman:

- ☐ **Tanda “\*”** (*asterisk* atau bintang) digunakan untuk perkalian menggantikan tanda x
- ☐ **Tanda “/”** (*slash* atau garis miring) digunakan untuk pembagian menggantikan tanda ÷
- ☐ **Tanda “.”** (*point* atau titik) digunakan sebagai pemisah desimal seperti tanda koma dalam bahasa Indonesia.

## C. Struktur dasar

Di dunia pemrograman, terdapat 3 prinsip yang dikenal yang selalu digunakan oleh *programmer* dalam proses pembuatan suatu program: *sequence* (runtutan), *selection* (percabangan), dan *loop* (perulangan). Tiga hal mendasar inilah yang membentuk konsep segala jenis pemrograman.



# Bab 4:

## Konsep Dasar Pemrograman Bahasa C

### A. Fitur-Fitur dalam Bahasa C

#### 1. Bahasa Pemrograman yang Prosedural

Prosedural merupakan salah satu **konsep dari bahasa pemrograman**. Dalam ilmu pemrograman, konsep prosedural berarti setiap perintah/statement (kode) yang kita tulis akan dijalankan secara urut dari perintah paling atas sampai perintah yang paling bawah.

Oleh karena itu, saat kita menulis kode (*ngoding*) dalam Bahasa C, kita harus menuliskan setiap perintah kodenya secara urut supaya alur dari program yang kita buat dapat berjalan sesuai yang kita inginkan.

#### 2. Bahasa Tingkat Menengah

Bahasa C adalah bahasa pemrograman tingkat menengah yang berada di antara bahasa tingkat rendah (Bahasa Mesin) dan bahasa tingkat tinggi (bahasa pemrograman modern seperti Python, Java, dll. ).

Contoh instruksi dalam Bahasa Mesin hingga bahasa tingkat tinggi:

Intruksi	Bahasa
B402 or 1011 0100 0000 0010 B22A or 1011 0010 0010 1010 CD21 or 1100 1101 0010 0001	Bahasa Mesin
Write('*');	Pascal
DISPLAY("*").	COBOL
Printf("*");	C
Cout<<"*";	C++
System.out.print("*");	Java
Cat('*')	R
Print('*',end = "")	Python

### 3. Cepat dan Efisien

Bahasa C merupakan bahasa tingkat menengah, maka dari itu Bahasa C dapat memberikan programmer akses manipulasi yang lebih cepat ke mesin tanpa ada pemrosesan tambahan seperti pada bahasa tingkat tinggi (Python, Java, dll.).

### 4. Portable

Kode dalam Bahasa C bisa *\*dcompile* berulang kali untuk dijalankan di berbagai sistem operasi tanpa perlu kamu mengubah kode-kode yang ada. Program bahasa C yang dibuat di Windows dengan bahasa C bisa dipindahkan ke Linux dengan sedikit/tanpa adanya modifikasi.

*\*Compile adalah sebuah proses pemeriksaan code yang kita buat untuk diterjemahkan ke bahasa mesin dan berjalan sesuai tujuan yang kita inginkan.*

### 5. Populer

Perusahaan besar seperti Microsoft, Google, Adobe, dll. menggunakan bahasa C dalam mengembangkan software mereka. Banyak software terkenal yang ditulis dalam bahasa C, mulai dari sistem operasi (Windows, Android, dll.) hingga program grafik 3D (3ds Max, Blender, Maya, dll.).

### 6. Inti dari Berbagai Bahasa

Bahasa pemrograman C banyak menjadi inspirasi dari pembuatan bahasa pemrograman lain, seperti C++, C#, PHP, JAVA, JavaScript dan masih banyak lagi. Dengan mempelajari bahasa C, maka kita akan merasa familiar dan lebih mudah saat pindah ke bahasa pemrograman lain yang merupakan turunan dari bahasa C itu sendiri.

## B. Menyiapkan compiler


Compiler adalah program untuk mengompilasi, yaitu proses menerjemahkan kode yang kita tulis menjadi kode yang dimengerti oleh komputer.

Compiler bahasa C yang paling populer adalah GCC, yang sebenarnya untuk Linux. Karena kita menggunakan sistem operasi Windows, jadi kita gunakan MinGW yang merupakan port GCC untuk Windows.

### 1. Mendownload compiler

Terdapat banyak build MinGW yang tersedia di internet, salah satunya WinLibs. Kami merekomendasikan WinLibs karena WinLibs yang paling up-to-date (selalu menggunakan versi terbaru GCC).

Download WinLibs: <https://winlibs.com/>



The screenshot displays the 'Release versions' section of the WinLibs website. It lists various compiler builds categorized by runtime (UCRT and MSVCRT) and architecture (Win32 and Win64). Each entry includes the GCC version, LLVM/Clang/LLD/LLDB version, MinGW-w64 version, and release number. Links for downloading the archives are provided for each build.

**Release versions**

**UCRT runtime**

- GCC 11.2.0 + LLVM/Clang/LLD/LLDB 13.0.0 + MinGW-w64 9.0.0 - UCRT - release 2 **(LATEST)**
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)

**MSVCRT runtime**

- GCC 11.2.0 + LLVM/Clang/LLD/LLDB 12.0.1 + MinGW-w64 9.0.0 - release 1
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
- GCC 11.1.0 + LLVM/Clang/LLD/LLDB 12.0.0 + MinGW-w64 9.0.0 - release 3
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
- GCC 10.3.0 + LLVM/Clang/LLD/LLDB 12.0.0 + MinGW-w64 9.0.0 - release 2
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#) - without LLVM/Clang/LLD/LLDB: [7-Zip archive\\*](#) | [Zip archive](#)
- GCC 9.4.0 + MinGW-w64 9.0.0 - release 1
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#)
- GCC 8.5.0 + MinGW-w64 9.0.0 - release 1
  - Win32: [7-Zip archive\\*](#) | [Zip archive](#)
  - Win64: [7-Zip archive\\*](#) | [Zip archive](#)
- GCC 7.5.0 + MinGW-w64 7.0.0 - release 1
  - Win32: [7-Zip archive\\*](#)
  - Win64: [7-Zip archive\\*](#)

**Snapshot versions**

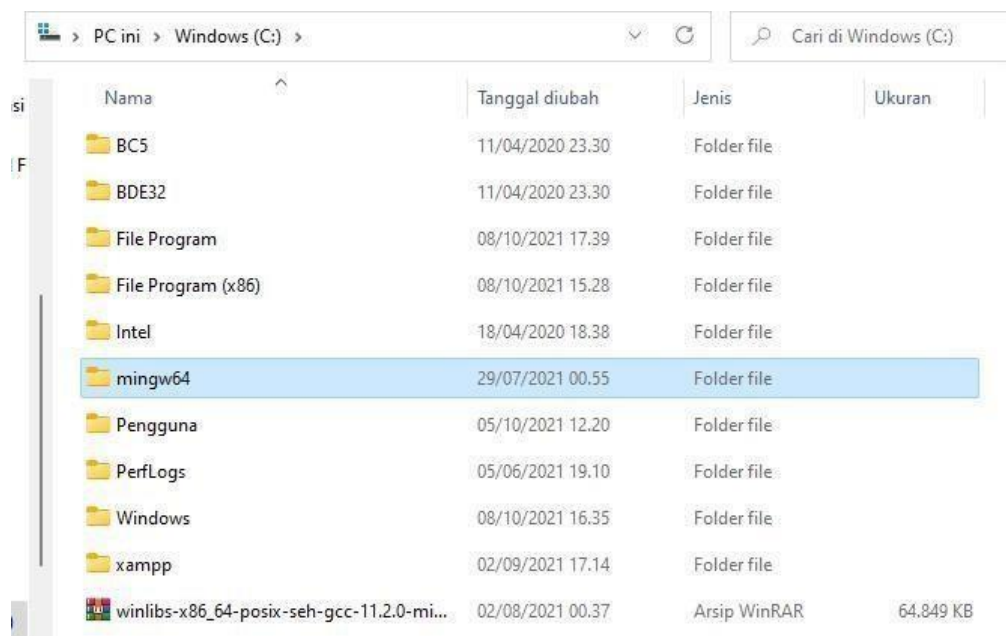
Pilih sesuai sistem operasi yang Anda gunakan, misalnya Windows 11 64-bit, maka download yang bagian Win64. Biasanya komputer-komputer sekarang menggunakan sistem operasi 64-bit.

Saran:

- Pilih yang versi terbaru, yaitu yang paling atas
- Pilih yang tanpa LLVM/Clang/LLD/LLDB yaitu link bagian kanan
- Pilih yang format 7-Zip agar ukuran download lebih kecil

## 2. Mengekstrak compiler

Compiler yang telah kita download tadi berbentuk arsip yang berisi folder “mingw64”, jadi kita harus mengekstraknya. Folder “mingw64” biasanya disimpan di C:, jadi kita ekstrak di C: saja. Maka akan ada folder “C:\mingw64” yang merupakan hasil ekstrak tadi.

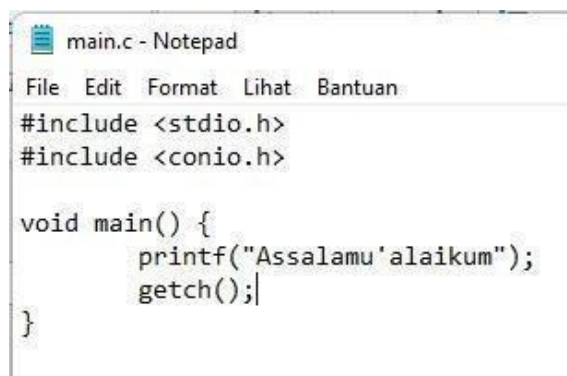


Compiler sebenarnya terletak di “C:\mingw64\bin”. Di situ ada 2 compiler, yaitu gcc.exe dan g++.exe. gcc adalah compiler untuk C, sedangkan g++ adalah compiler untuk C++. Kita akan menggunakan gcc karena kita menggunakan bahasa C.



langsung menutup sebelum ada tombol keyboard yang ditekan. Fungsi getch() ada di library conio.h.

Menulis kode sebenarnya bisa dilakukan di text editor apapun, termasuk Notepad.



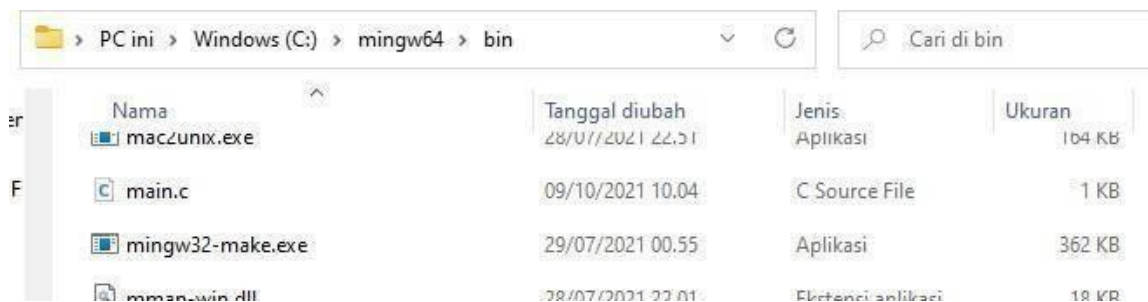
```
main.c - Notepad
File Edit Format Lihat Bantuan
#include <stdio.h>
#include <conio.h>

void main() {
    printf('Assalamu'alaikum');
    getch();
}
```

Simpan kode di atas sebagai file C (file dengan nama berakhiran “.c”), misalnya “main.c”. File ini disebut juga file sumber, karena berisi kode sumber.

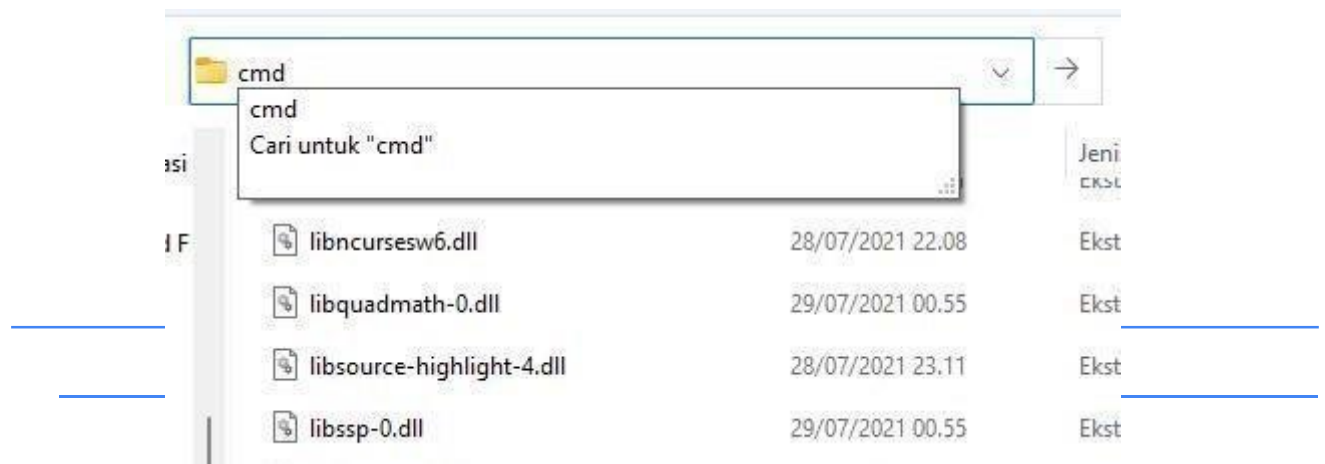
## 4. Mengompilasi dan menjalankan program

Kompilasi adalah proses menerjemahkan kode yang kita tulis menjadi kode yang dimengerti oleh komputer. Saat ini kita hanya bisa memanggil compiler dari folder compiler itu sendiri, jadi file sumber yang akan kita kompilasi harus terletak di folder tempat compiler berada, yaitu di “C:\mingw64\bin”.



Buka Command Prompt dengan cara ketik “cmd” di kotak path lalu enter.

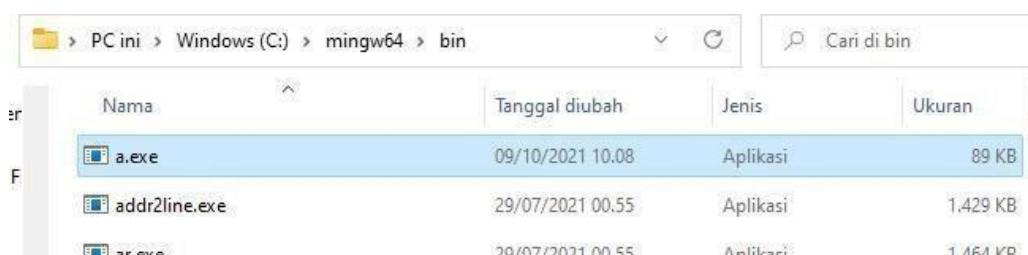




Untuk mulai mengompilasi, ketik “gcc nama\_file\_sumber.c”, lalu enter.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.194]
(c) Microsoft Corporation. All rights reserved.
C:\mingw64\bin>gcc main.c
```

Jika tidak ada kesalahan, maka compiler akan menghasilkan file “a.exe”. File ini disebut executable file (file yang dapat dieksekusi/dijalankan).



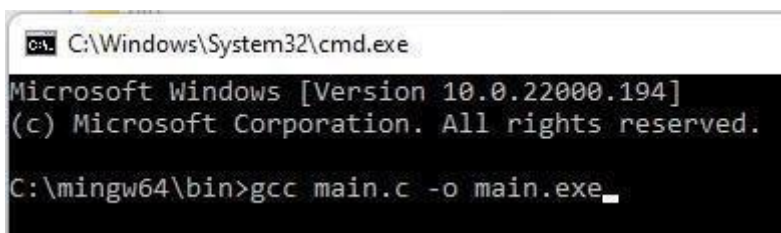
File itu adalah program hasil kompilasi dari kode kita. Sekarang coba jalankan program itu, maka akan keluar output seperti di bawah ini.

## 5. Mengompilasi dengan nama kustom

Sebelumnya setiap kita mengompilasi, file hasil kompilasi selalu bernama "a.exe". Untuk menggunakan nama kustom (sesuai keinginan kita), gunakan perintah berikut.

```
gcc nama_file_sumber.c -o nama_file_output.exe
```

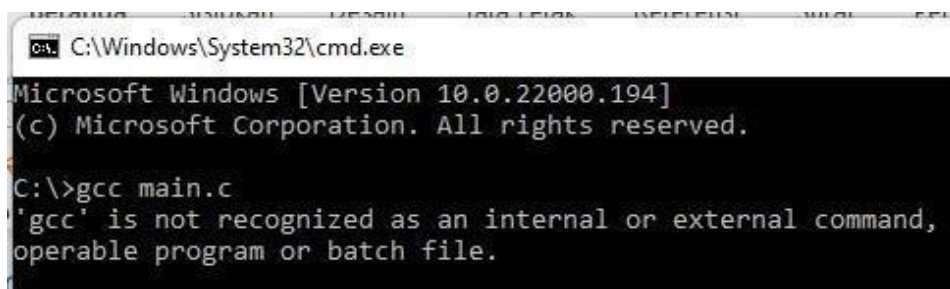
Contoh:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.194]
(c) Microsoft Corporation. All rights reserved.
C:\mingw64\bin>gcc main.c -o main.exe_
```

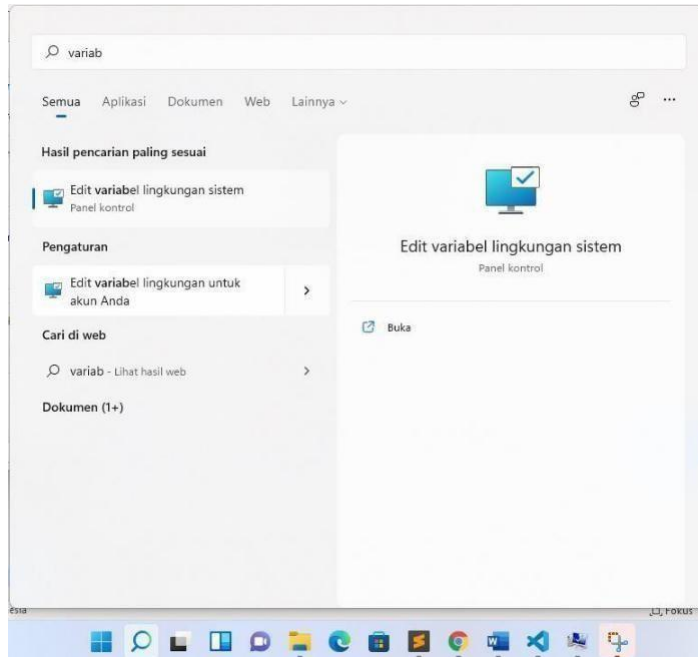
## 6. Mengompilasi dari folder mana pun

Sebelumnya kita hanya bisa mengompilasi dari folder tempat compiler berada, sehingga file sumber harus berada di folder itu untuk mengompilasinya. Karena kita tidak bisa memanggil compiler dari tempat lain.

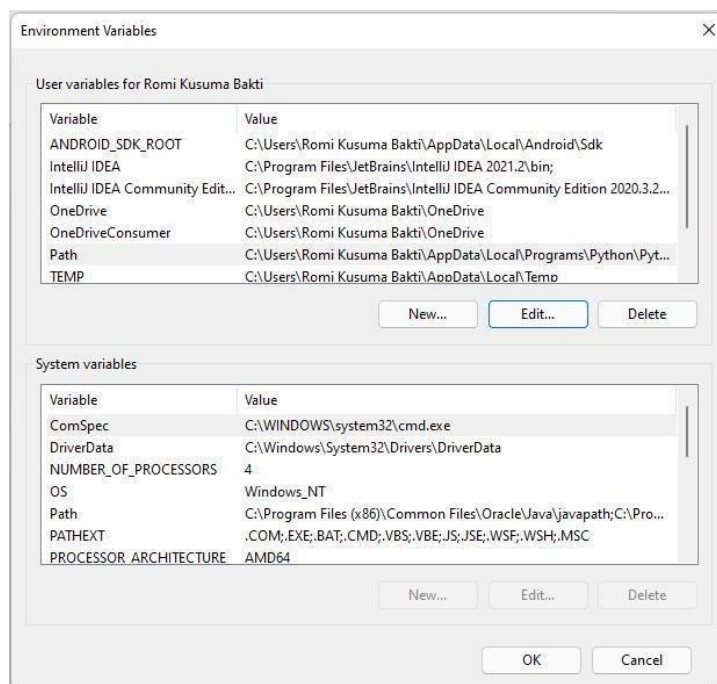


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.194]
(c) Microsoft Corporation. All rights reserved.
C:\>gcc main.c
'gcc' is not recognized as an internal or external command,
operable program or batch file.
```

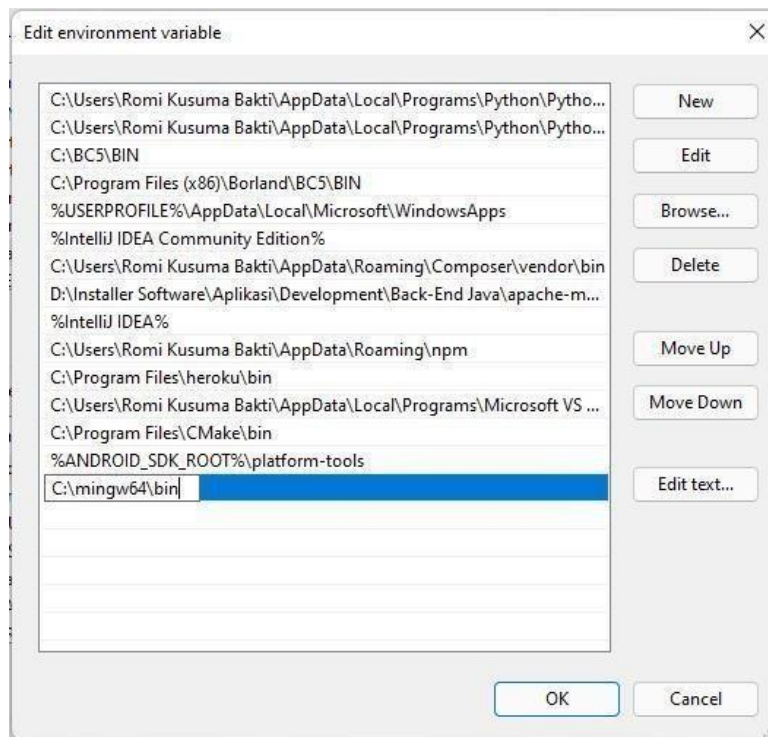
Untuk mengatasi hal ini, kita bisa menambahkan path (jalur/alamat) folder lokasi compiler kita ke Environment Variables. Kita bisa mencarinya di Windows Search.



Pilih “Edit environment variables for your account”. Di bagian “User variables for (user name)” klik “Edit...”.



Tambahkan path folder compiler lalu klik OK.



Klik OK untuk menyimpannya. Untuk memeriksa apakah berhasil, buka Command Prompt baru, ketik “gcc --version” lalu enter.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.194]
(c) Microsoft Corporation. All rights reserved.

D:\>gcc --version
gcc (MinGW-w64 x86_64-posix-seh, built by Brecht Sanders) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Jika berhasil, maka akan muncul versi GCC yang kita gunakan.

Mulai sekarang, Anda bisa mengompilasi kode Anda dari folder mana pun.

## C. Media Pemrograman Bahasa C Melalui IDE (Integrated Development Environment)

### 1. Pengertian IDE (Integrated Development Environment)

IDE (Integrated Development Environment) adalah aplikasi perangkat lunak yang menyediakan fasilitas lengkap bagi programmer untuk mengembangkan sebuah perangkat lunak.

Dengan adanya IDE, kita bisa melakukan semua itu dalam satu aplikasi. Kompilasi dan eksekusi bisa dilakukan dengan sekali klik. Selain itu banyak sekali fitur IDE yang dapat mempermudah kita dalam memrogram.

### 2. Contoh IDE

IDE yang populer digunakan untuk bahasa C saat ini adalah Visual Studio Code dan Code: Blocks.

#### 1. Visual Studio Code



Visual Studio Code adalah IDE yang dibuat oleh Microsoft yang tersedia untuk Windows, Linux, dan macOS. Ini adalah IDE yang paling populer di dunia, IDE ini bisa digunakan untuk hampir semua bahasa pemrograman. Ke depannya teman-teman akan banyak menggunakan IDE ini di hampir semua pelatihan di PUB.

#### 2. Code::Blocks



Code::Blocks merupakan sebuah IDE yang lebih berorientasi pada C, C++, dan Fortran.

#### 3. PUB Code

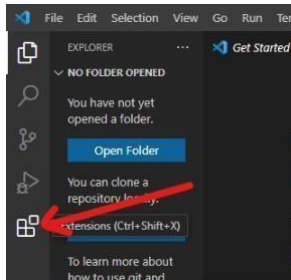


PUB Code adalah IDE C/C++ yang sederhana, gratis, dan open source untuk Windows. IDE ini cocok untuk pemula karena kesederhanaannya.

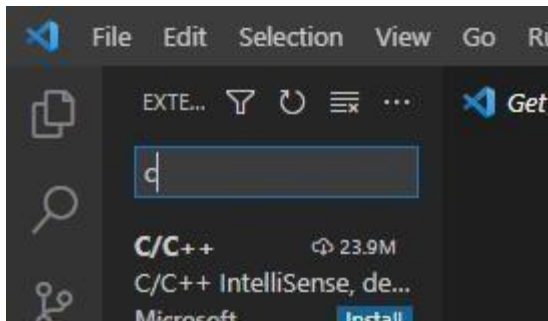
## D. Menyiapkan Visual Studio Code

### Menginstal ekstensi C/C++

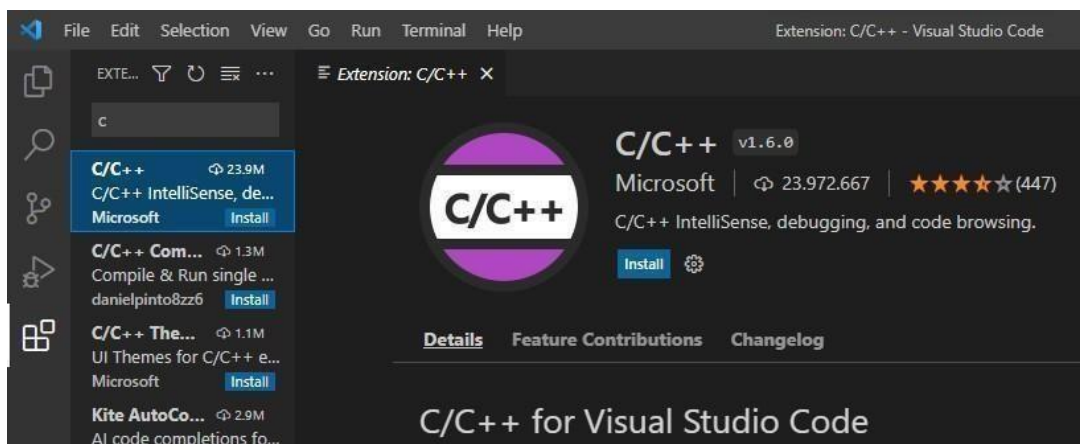
1. Di Visual Studio Code, pilih “Extensions”



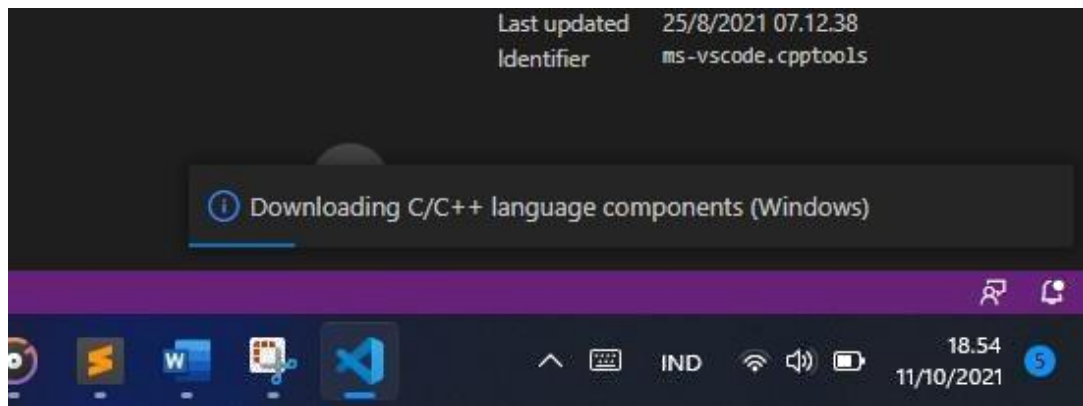
2. Cari “c”



3. Pilih ekstensi yang bernama “C/C++” dari Microsoft, lalu klik “Install”

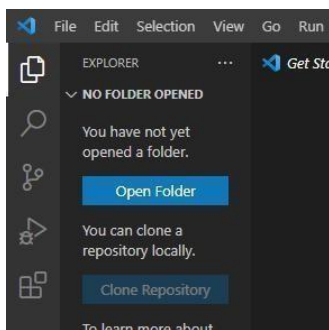


Tunggu sampai proses download semua komponen selesai

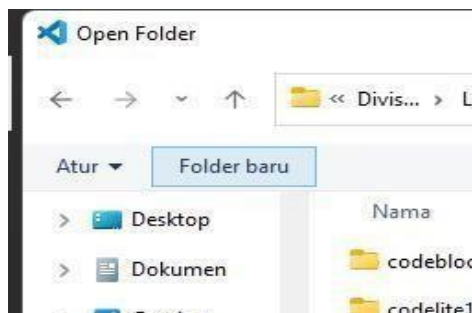


## Menyiapkan project baru

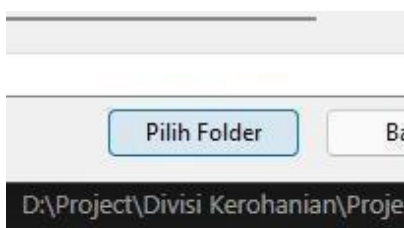
1. Di Visual Studio Code, pilih “Open folder”



2. Buat folder baru untuk project sekarang dengan mengklik “New folder”, beri nama dengan nama project kita, misalnya “Latihan 1”.



3. Klik tombol “Select folder”

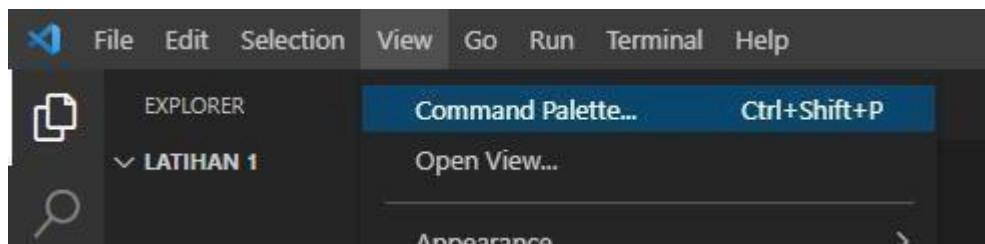




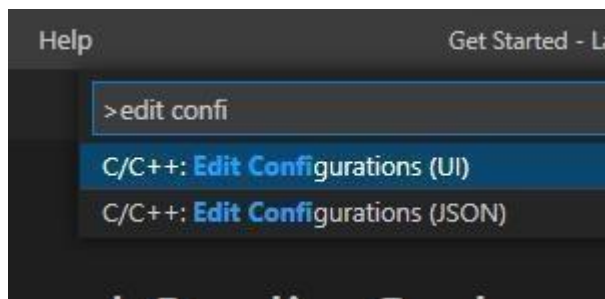
4. Jika muncul pertanyaan “Do you trust the authors of the files in this folder?” maka pilih “Yes, I trust the authors”, jika tidak ingin muncul pertanyaan ini lagi, centang checkbox di atasnya.



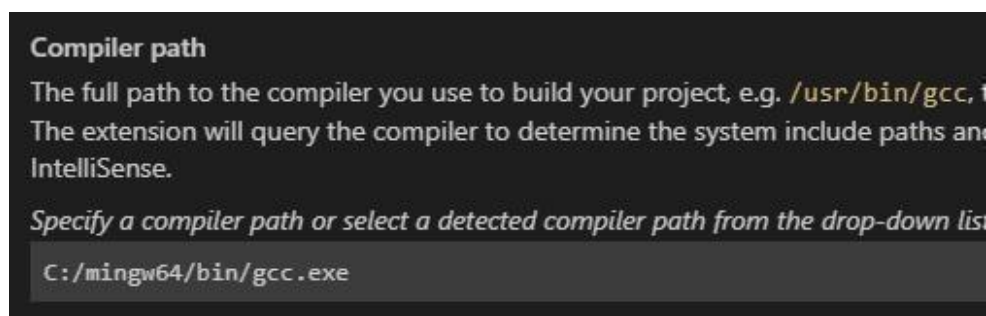
5. Pilih “View”, lalu “Command Palette...”, atau tekan Ctrl+Shift+P



6. Cari dan pilih “Edit Configurations (UI)”



7. Di bagian “Compiler path”, isi dengan path compiler kita (gcc.exe)





## E. Langkah-Langkah Membuat Program

1. Menentukan tujuan dari program yang akan kita buat
2. Menulis kode (*ngoding*) menggunakan Bahasa C dan menyimpannya sebagai file C (dengan nama berakhiran “.c”, contoh : *ini\_file\_saya.c*)
3. Mengompilasi (*compile*) file C yang telah kita buat, proses ini akan menghasilkan file berupa program yang dapat dijalankan (file exe)
4. Menjalankan program hasil kompilasi/*compile* tersebut
5. Menguji jalannya program tersebut
6. Memelihara dan memodifikasi program jika diperlukan

# Bab 5:

## Struktur Pemrograman Bahasa C

Saat kita menulis kode (*ngoding*), kita **tidak bisa sembarangan menuliskan kode secara acak** karena saat program kita berjalan, kode/perintah yang dilaksanakan oleh komputer adalah perintah paling atas dulu lalu urut ke bawah.

Berikut ini adalah **gambaran umum** dari struktur suatu program dengan menggunakan Bahasa C :

```
#preprocessor_directive <nama_standard_library.h>
#preprocessor_directive "nama_file_lain.c"
#preprocessor_directive variable nilai_yang_akan_menggantikan

tipe_data_pengembalian main ()
{
    pernyataan 1;
    pernyataan 2;
    ...
}
```

Keterangan :

Warna Latar Belakang	Nama Bagian
	Preprocessor
	Fungsi main (program utama)

Tidak perlu bingung untuk memahami semua bagian-bagian dari struktur pemrograman Bahasa C di atas, kita akan membahas dan memahami satu per satu bagian- bagiannya :

### A. Preprocessor

```
#preprocessor_directive <nama_standard_library.h>
#preprocessor_directive "nama_file_lain.c"
#preprocessor_directive variable
```

Preprocessor adalah program yang memanipulasi *source code* (kode yang sedang kita buat) sebelum dikompilasi oleh komputer. Kita membutuhkan kode untuk memanggil program preprocessor yang akan kita gunakan, yaitu dengan *preprocessor directive*

**Preprocessor directive** adalah sintak kode untuk memanggil preprocessor. Dalam pelatihan ini, preprocessor directive yang akan sering kita pakai dalam adalah **#include** yang berada pada bagian *header* dan **#define** yang berada pada bagian *deklarasi variable global*.

Letak penggunaan preprocessor directive :

### 1. Preprocessor Directive pada Header

Bagian header berfungsi untuk **mendeklarasi (menyertakan) library** (artinya: perpustakaan) yang berisi perintah-perintah (kode-kode) untuk menjalankan program yang kita buat. Untuk mendeklarasi library yang akan kita gunakan, maka kita harus menggunakan preprocessor directive **#include**. Contoh :

```
#include <nama_library.h>
#include "nama_file_lain.c"
```

Library dibagi menjadi dua, yaitu **standard library (library bawaan Bahasa C)** dan **library yang kita buat**.

Untuk menyertakan standard library, kita harus menulis nama *file header* (file library) diantara tanda kurung siku (< . . . >) dan nama filenya diakhiri dengan (.h) untuk menandakan bahwa library tersebut berbentuk file header.

```
#include <nama_library.h>
```

Berikut ini adalah beberapa standard library Bahasa C yang sering digunakan dalam pelatihan pemrograman PUB :

#### a. Standard Input Output <stdio.h>

Library ini berfungsi sebagai standar input output operasi yang digunakan oleh bahasa C.

Fungsi	Keterangan	Cara penulisan
printf	Mencetak (sebagai output)	printf("");

<b>scanf</b>	Menginputkan Variable dengan semua tipe data tanpa membaca spasi	<code>scanf("%format_specifier", &amp;nama_variable);</code>
<b>gets</b>	Menginputkan variable yang tipe datanya <b>string</b> dengan spasi	<code>gets(nama_variable);</code>
<b>getchar</b>	Menginputkan <b>char</b>	<code>nama_variable = getchar();</code>
<b>fgets</b>	Sama dengan <b>gets</b> tetapi <b>fgets</b> lebih aman digunakan karena <b>fgets</b> dapat membaca ukuran dari <b>string</b>	<code>fgets(nama_variable, ukuran_string / sizeof(ukuran), stdin);</code>

b. Control Input Output **<conio.h>**

Library ini berfungsi untuk menampilkan hasil inputan dalam character kepada pengguna.

Fungsi	Keterangan	Cara penulisan
<b>getch</b>	Menginput satu <b>char</b> dan bisa digunakan untuk menahan output program	<code>nama_variable = getch();</code>
<b>getche</b>	Menginput satu <b>char</b> tanpa perlu diakhiri dengan ENTER	<code>nama_variable = getche();</code>

c. String **<string.h>**

Library ini berfungsi untuk memanipulasi variable maupun nilai yang bertipe data **string**

Fungsi	Keterangan	Cara penulisan
<b>strcpy</b>	Menyalin <b>string</b> dari suatu variable ke variable lain	<code>strcpy(variabel_tujuan, string);</code>
<b>strlen</b>	Mengetahui panjang string	<code>strlen(nama_variabel);</code>
<b>strupr</b>	Membuat bentuk <b>string</b> menjadi UPPER CASE	<code>strupr(nama_variabel);</code>

<b>strlwr</b>	Membuat bentuk <b>string</b> menjadi lower case	<b>strlwr</b> (nama_variabel);
<b>strcmp</b>	Membandingkan dua buah <b>string</b> . Jika kedua <b>string</b> tersebut sama akan mengembalikan nilai 0. Jika kedua <b>string</b> tersebut berbeda maka akan mengembalikan nilai 1.	<b>strcmp</b> (string1,string2);
<b>strcat</b>	Menggabungkan dua buah <b>string</b>	<b>strcat</b> (string1,string2);
<b>strrev</b>	Mengembalikan <b>string</b>	<b>strrev</b> (nama_variabel);

d. Standard Boolean **<stdbool.h>**

Library ini berfungsi untuk pendeklaran variable yang menggunakan tipe data boolean (**bool**)

e. Standard Library **<stdlib.h>**

Library ini berfungsi sebagai operasi konversi tipe data dan fungsi kontrol proses

Fungsi	Keterangan	Cara penulisan
<b>atof</b>	Mengonversi nilai string menjadi bilangan bertipe data <b>float</b>	<b>atof</b> (nama_variable);
<b>atoi</b>	Mengonversi nilai string menjadi bilangan bertipe data <b>integer</b>	<b>atoi</b> (nama_variable);
<b>system("cls")</b>	Membersihkan layar <i>command prompt</i>	<b>system</b> ("cls");

<code>system("pause")</code>	Menjeda jalannya program sampai kita menginputkan sesuatu / menekan tombol apapun	<code>system("pause");</code>
<code>exit</code>	Keluar dari program	<code>exit(0);</code>

f. Mathematic `<math.h>`

Library ini berfungsi sebagai operasi matematika

Fungsi	Keterangan	Cara penulisan
<code>sqrt</code>	Menghitung nilai akar dari suatu bilangan	<code>sqrt(nama_variable);</code>
<code>pow</code>	Menghitung pangkat dari suatu bilangan	<code>pow(bilangan, pangkat);</code>
<code>sin</code>	Digunakan untuk menghitung nilai <i>sinus</i> , <i>cosinus</i> , dan <i>tangen</i>	<code>sin(nama_variable);</code>
<code>cos</code>		<code>cos(nama_variable);</code>
<code>tan</code>		<code>tan(nama_variable);</code>

g. Windows `<windows.h>`

Library ini berisi deklarasi untuk semua fungsi di Windows API (pemrograman aplikasi yang dibuat oleh Microsoft dalam inti sistem operasi Microsoft Windows buatannya.)

Fungsi	Keterangan	Cara penulisan
<code>Sleep</code>	Menjeda jalannya program selama jumlah detik yang kita tentukan	<code>Sleep(detik);</code>

Sedangkan untuk **library yang kita buat** harus berupa file yang berbentuk C (.c) dan dituliskan diantara tanda kutip dua ("").

```
#include "nama_file_lain.c"
```

## 2. Preprocessor Directive pada Deklarasi Global

Preprocessor directive yang terletak pada deklarasi global adalah preprocessor directive **#define**

```
#define variable nilai_yang_akan_menggantikan
```

Pada prosesnya **#define** akan menggantikan setiap **variable** tertentu dalam program dengan suatu **nilai**.

Contoh:

```
#define sisi 4
```

Dengan pernyataan di atas, preprocessor akan mengganti setiap **sisi** yang ada di program tersebut dengan bilangan **4** sebelum program itu berjalan.

## B. Fungsi main

```
tipe_data_pengembalian main ()  
{  
    pernyataan 1;  
    pernyataan 2;  
    ...  
}
```

**tipe\_pengembalian** artinya **jenis data yang akan dikembalikan** (data yang didapatkan setelah fungsi itu selesai berjalan). Bisa berupa angka, karakter, teks, dan sebagainya.

Karena kita belum ingin fungsi itu mengembalikan data, jadi tulis saja **void**, yang artinya kosong, jadi tidak mengembalikan data apapun.

Agar fungsi itu dijalankan otomatis saat program dijalankan, maka fungsi harus bernama **main**, yang artinya utama, berarti **yang dijalankan pertama kali**.

## C. Comment

*Comment* (komentar) adalah teks yang diabaikan oleh compiler dan tidak dijalankan oleh program. Komentar biasanya digunakan untuk memberi **keterangan untuk referensi** dari kode yang kita buat, terkadang juga digunakan agar kode yang kita buat diabaikan sementara oleh compiler agar kode kita tidak dijalankan.

Terdapat dua jenis kode *comment* dalam Bahasa C yaitu :

## 1. Single-line Comment

*Single-line comment* (komentar satu baris) adalah komentar yang hanya memiliki satu baris.

Penulisannya diawali dengan *double slash* (garis miring ganda) (*//*)

Contoh:

```
void main()
{
    // ini komentar satu baris yang tidak akan ditampilkan
    printf("Assalamu'alaikum");

    getch();
}
```

## 2. Multi-line Comment

*Multi-line comment* (komentar banyak baris) adalah komentar yang terdiri lebih dari satu baris.

Penulisannya diawali dengan *slash asterisk* (garis miring bintang) (*/\**) dan diakhiri dengan *asterisk slash* (bintang garis miring) (*\*/*). Contoh:

```
void main()
{
    /* Komentar ini terdiri
    lebih dari
    satu baris
    dan tidak akan dijalankan
    oleh komputer*/
    printf("Assalamu'alaikum");
}
```



# Bab 6:

## Tipe Data dan Variabel

### A. Tipe data

Tipe data dapat dikatakan sebagai **pengenal/identitas** dari suatu variable. Tipe data yang merupakan bagian program yang paling penting karena mempengaruhi setiap intruksi yang akan dilaksanakan oleh komputer.

Kita juga harus mengetahui *format specifier*, apa itu *format specifier*? *Format specifier* adalah sebuah kode yang digunakan **untuk membaca variabel** ketika akan dicetak. Kode penulisan **Format specifier** diawali dengan menuliskan tanda persen (%) lalu diikuti dengan huruf sesuai tipe datanya.

Dalam Bahasa C terdapat banyak tipe data, tetapi secara garis besar hanya ada beberapa tipe data yang wajib diketahui oleh *programmer*, diantaranya :

#### 1. Integer

Integer (bilangan bulat) adalah tipe data berupa bilangan yang tidak mendukung pecahan.

#### 2. Floating point

Floating point (bilangan pecahan) adalah tipe data berupa bilangan yang mendukung pecahan.

#### 3. Character

*Character* (karakter) adalah tipe data yang berupa karakter.

#### 4. String

String adalah tipe data yang berupa sekumpulan *character*.

#### 5. Boolean

Boolean adalah tipe data yang hanya berisi **True** atau **False** yang biasa digunakan dalam operator logika. Boolean juga bisa dikondisikan sebagai angka misalnya 1 sebagai kondisi True dan 0 sebagai kondisi False.

Setiap tipe data memiliki format specifiernya dan penggunaan secara umumnya masing-masing, berikut penjelasannya :

Tipe data	Format specifier	Penggunaan umum
char	%c	Karakter
char [] (string)	%s	Sekumpulan karakter
short	%hi atau %hd	Bilangan bulat pendek
int	%i atau %d	Bilangan bulat sedang
long	%li atau %ld	
float	%f	Bilangan pecahan
double	%lf	Bilangan pecahan panjang
bool	%i atau %d	Menghasilkan nilai 1 untuk benar Menghasilkan nilai 0 untuk salah

Setiap tipe data memiliki jumlah byte dan rentang nilainya masing-masing, berikut penjelasannya :

Tipe data	Jumlah byte	Rentang nilai
char	1 byte	kode ASCII 0 sampai 255
short	2 byte	-32768 sampai 32767
int	4 byte	-2147483648 sampai 2147483647
long		
float	4 byte	Bilangan pecahan dari 1.2E-38 sampai 3.4E+38 dengan presisi 6 di belakang koma.
double	8 byte	Bilangan pecahan dari 2.3E-308 sampai 1.7E+308 dengan presisi 15 di belakang koma.
bool	1 byte	false atau true

## B. Variabel dan Konstanta

### 1. Variabel

Variabel adalah wadah yang digunakan untuk **menampung data**. Data dalam variabel disebut **nilai/value**.

Bayangkan ada di sebuah meja makan terdapat gelas, piring, susu, dan nasi. Analoginya, gelas dan piring tempat (sebagai variable) untuk menampung susu atau nasi (sebagai value) tersebut. Jadi, dalam membuat program kita sangat membutuhkan variabel.

Untuk mengenali variabel tersebut maka harus ada tipe data.

Gelas adalah tempat susu, sedangkan piring adalah tempat untuk nasi.

Jika kita memilih gelas maka gunakan gelas itu untuk menampung susu, tidak mungkin kan gelas untuk menampung nasi.

Oleh karena itu dalam implementasi ke dalam bahasa pemrograman juga harus menggunakan tipe data sebagaimana harusnya.

#### a) Aturan Penulisan Variable

Berikut adalah aturan dalam penulisan / penamaan variabel:

1. Tidak boleh menggunakan simbol khusus (!, @, #, \$, %, ^, &, \*, (, ), -, +, =) kecuali *underscore* atau garis bawah (\_)
2. Karakter pertama tidak boleh angka
3. Tidak boleh menggunakan spasi

Contoh nama variabel yang **tidak valid**:

Penulisan variable	Alasan tidak valid
<code>int 6i14ng4n;</code>	karena diawali angka, yaitu angka 6
<code>int total bilangan;</code>	karena mengandung spasi
<code>int total-bilangan;</code>	karena mengandung simbol khusus selain _ (garis bawah), yaitu simbol – (tanda hubung)

Contoh nama variabel yang **valid** tapi **tidak sesuai standar** konvensi/aturan penulisan:

Penulisan variable	Alasan tidak sesuai standar
<code>int BILANGAN;</code>	Ini adalah konvensi variabel di bahasa pemrograman lain
<code>int TotalBilangan;</code>	
<code>int totalBilangan;</code>	

Contoh nama variabel yang **valid** dan **sesuai standar** konvensi/aturan penulisan:

`int bilangan;`

`int bilangan_1;`

`int total_bilangan;`

Di bahasa C, penamaan variabel **selalu menggunakan huruf kecil** dan setiap kata **dipisahkan dengan tanda garis bawah/underscore** (\_).

Bahasa C bersifat **case sensitive**, artinya **huruf besar dan kecil dianggap berbeda**.

Jadi, variable *NAMA*, *nama* dan *Nama* itu dianggap berbeda.

## b) Pembuatan dan Manipulasi Variable

### 1) Declaration

Declaration (deklarasi) adalah menyatakan sebuah variabel tanpa memberinya value (nilai).

```
int bilangan;
```

### 2) Initialization

Initialization (inisialisasi) adalah menyatakan sebuah variabel sekaligus memberinya nilai awal.

```
int bilangan = 7;
```

Jika ingin melakukan declaration dan initialization variable yang tipe datanya sama, maka bisa digabung dalam satu statement dengan dipisahkan tanda koma.

```
int a = 5, b = 2, c;
```

Kode di atas menginisialisasi variabel a dan variable b, serta mendeklarasi variabel c.

### 3) Assignment

Assignment (penugasan) adalah mengubah value dari sebuah variabel.

Assignment dilakukan dengan operator assignment.

Misalnya = (mengganti), += (menambahkan), dll.

```
int bilangan = 7;  
bilangan = 3;  
bilangan += 2;
```

*\*operator assignment akan dipelajari lebih lanjut di materi tentang operator*

## c) Jenis Variable

### 1. Variabel lokal

Variabel lokal adalah variabel yang dinyatakan di dalam fungsi, variabel lokal hanya bisa diakses dari dalam fungsi itu sendiri.

Contoh:

```
#include <stdio.h>  
#include <conio.h>  
  
void main()  
{  
    int bilangan = 7;  
    printf("%d", bilangan);  
  
    getch();  
}
```

Variabel *bilangan* hanya bisa diakses dari dalam fungsi *main()* saja.

### 2. Variabel global

Variabel global adalah variabel yang dinyatakan di luar fungsi, variabel global bisa diakses dari mana saja.

Contoh:

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

int bilangan = 7;

void cetak_1()
{
    printf("%d", bilangan);
}

void cetak_2()
{
    printf("%d", bilangan);
}
```

Variabel `bilangan` bisa diakses dari dalam fungsi `cetak_1()` maupun fungsi `cetak_2()`.

## 2. Konstanta

Konstanta adalah **variabel yang nilainya tetap** (tidak dapat diubah, kecuali dengan inputan saat program berjalan). Konstanta tidak dapat dikenai assignment (pengubahan nilai) yang kita coding dalam program. Contoh:

```
const int bilangan = 7;
```

Kode di atas menunjukkan bahwa variabel `bilangan` nilainya tetap, sehingga tidak dapat dilakukan assignment terhadap variabel itu.

## C. Escape Sequence

Escape sequence adalah urutan karakter yang tidak mewakili dirinya sendiri saat digunakan di dalam karakter atau string literal, tetapi diterjemahkan ke dalam karakter lain atau urutan karakter yang mungkin sulit atau tidak mungkin untuk direpresentasikan secara langsung.

Kumpulan karakter ini selalu dimulai oleh karakter *backslash* (\) dan diikuti oleh karakter.

Berikut ini escape sequence dan karakter yang diwakilinya :

Escape sequence	Karakter yang diwakili
<code>\a</code>	Memberikan peringatan (suara bip/bell)
<code>\b</code>	Backspace (menimpa 1 karakter sebelumnya dengan 1 karakter sesudahnya)
<code>\n</code>	Newline (baris baru)
<code>\r</code>	Carriage return (mencetak <i>string</i> setelahnya di konsol paling kiri)
<code>\t</code>	Memberikan <i>horizontal tab</i> / tab spasi
<code>\\</code>	Mencetak backslash
<code>\'</code>	Mencetak tanda petik
<code>\"</code>	Mencetak tanda kutip

Contoh penggunaan escape sequence:

```
// membuat baris baru menggunakan \n (new line)
printf("Baris pertama\nBaris kedua");
```

Output:

```
Baris pertama
Baris kedua_
```

## D. Input Output Program

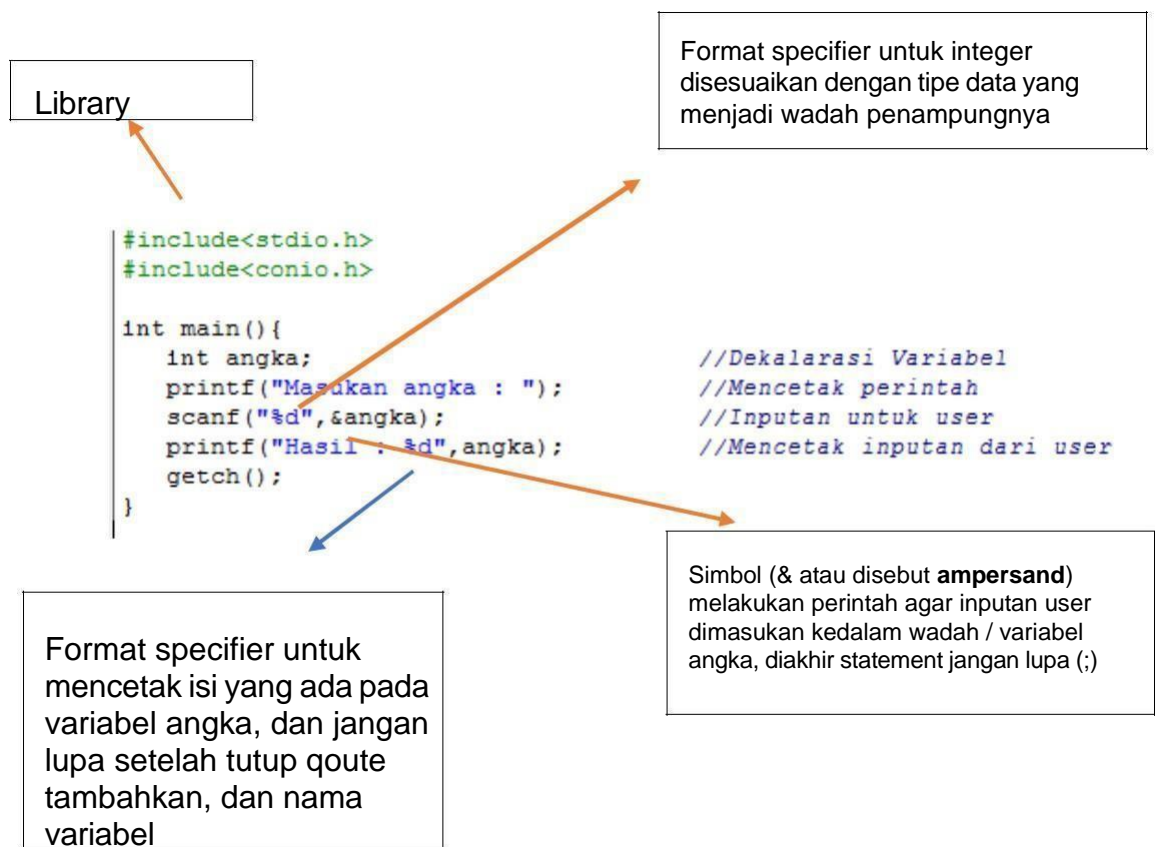
Input dan output adalah hal yang sangat dasar dalam program, dalam input dan output harus menggunakan format specifier.

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

void main()
{
    int bilangan; // deklarasi variabel

    printf("Input: ");
    scanf("%d", &bilangan); // menerima input dari pengguna
    printf("Output: %d", bilangan); // mencetak output

    getch();
}
```



Hasil outputnya akan seperti ini :

*Program akan meminta input dari kita.*

Input: \_

*Program akan mencetak kembali bilangan yang tadi diinput.*



Input: 5  
Output: 5\_

## Input string yang memiliki spasi

Fungsi `scanf()` hanya bisa membaca teks tanpa spasi. Jika terdapat spasi pada teks yang diinput maka `scanf()` hanya akan membaca sampai karakter sebelum spasi, artinya jika kita memasukkan kalimat maka `scanf()` hanya akan membaca kata pertama.

Untuk membaca teks yang mengandung spasi, kita bisa menggunakan `gets()`.

Sintak :

```
gets(variabel);
```

Contoh :

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

void main()
{
    char kalimat[20];

    printf("Input: ");
    gets(kalimat);
    printf("Output: %s", kalimat);

    getch();
}
```

# Bab 7:

## Operator

Operator secara umum adalah **sesuatu yang melakukan operasi**. Operator dalam Bahasa C dikenal sebagai suatu karakter atau simbol khusus yang berguna untuk melakukan operasi seperti *penjumlahan*, *perkalian*, *pembagian*, *pengurangan*, *perbandingan*, dan *inisialisasi (pengisian data)*.

Operator yang paling kita kenal adalah operator perkalian, pembagian, penjumlahan, pengurangan, dan modulus (sisa hasil bagi) yang dalam Bahasa C dikenal sebagai operator **aritmatika**. Namun dalam bahasa C masih banyak jenis operator yang tersedia dan harus kita pahami. Adapun jenis-jenis operator bahasa C adalah sebagai berikut.

### A. Operator Aritmatika

No.	Simbol	Nama	Contoh penulisan
1.	+	Penjumlahan	<code>a + b;</code>
2.	-	Pengurangan	<code>a - b;</code>
3.	*	Perkalian	<code>a * b;</code>
4.	/	Pembagian	<code>a / b;</code>
5.	%	Modulus (sisa hasil bagi)	<code>a % b;</code>

Contoh:

```
int a, b;

printf("Bilangan A: ");
scanf("%d", &a);

printf("Bilangan B: ");
scanf("%d", &b);

printf("\n");
printf("Hasil penjumlahan : %d\n", a + b);
printf("Hasil pengurangan : %d\n", a - b);
printf("Hasil perkalian      : %d\n", a * b);
printf("Hasil pembagian      : %d\n", a / b);
printf("Hasil modulus        : %d\n", a % b);
```

## B. Operator Penugasan

Operator penugasan adalah operator yang digunakan untuk mengisi variabel dengan suatu nilai.

Contoh:

```
int bilangan;  
bilangan = 7;  
  
printf("%d", bilangan);
```

Penjelasan:

- `int bilangan;` artinya kita membuat variabel bertipe data integer bernama `bilangan`
- `bilangan = 7;` artinya kita memasukkan angka 7 ke dalam variabel `bilangan`

Operator penugasan yang lain:

No.	Simbol	Nama	Kegunaan
1.	=	Sama dengan	Mengisi variabel dengan nilai baru
2.	+=	Plus sama dengan	Mengisi variabel dengan nilai sebelumnya ditambah suatu nilai
3.	-=	Min sama dengan	Mengisi variabel dengan nilai sebelumnya dikurangi suatu nilai
4.	*=	Kali sama dengan	Mengisi variabel dengan nilai sebelumnya dikali suatu nilai baru
5.	/=	Bagi sama dengan	Mengisi variabel dengan nilai sebelumnya dibagi suatu nilai baru
6.	%=	Modulus sama dengan	Mengisi variabel dengan nilai sebelumnya dimodulus (sisanya pembagian) suatu nilai baru

Contoh:

```
int bilangan = 5;  
bilangan += 3;  
bilangan--;  
printf("%d", bilangan);
```

Output : 7

karena awalnya bilangan bernilai 5, lalu ditambah 3, lalu dikurangi 1.

## C. Operator Perbandingan

Comparison operator (operator perbandingan) adalah operator yang berfungsi untuk membandingkan suatu variabel dengan variabel lainnya atau suatu kondisi dengan kondisi lainnya.

Operator perbandingan biasanya digunakan dalam kondisi pilihan (if else). Dalam bahasa C operator perbandingan ditulis dengan simbol-simbol sebagai berikut:

No.	Simbol	Deskripsi
1.	==	Sama dengan
2.	!=	Tidak sama dengan
3.	<>	
4.	>	Lebih dari
5.	<	Kurang dari
6.	>=	Lebih dari sama dengan
7.	<=	Kurang dari sama dengan

Catatan:

Bedakan antara **operator penugasan (=)** dengan **operator perbandingan (==)**.

## D. Operator Logika

Berikut adalah cara penulisan dan ketentuan operator dalam bahasa C.

No.	Simbol	Nama	Deskripsi
1.	&&	And (dan)	Membandingkan 2 atau lebih kondisi di mana kondisi akan True (benar) <b>jika semua syarat terpenuhi</b>
2.		Or (atau)	Membandingkan 2 atau lebih kondisi dimana kondisi akan True <b>jika salah satu syarat terpenuhi</b> /benar maka akan menghasilkan kondisi True
3.	!	Not (not/negasi)	Menandakan <b>negasi</b> /awan dari suatu kondisi/pernyataan

Berikut tabel perbandingan yang hanya akan membandingkan 2 kondisi Operator logika && (and).

### 1. && (and)

Kondisi 1	Kondisi 2	Hasil
true	true	true
true	false	false
false	true	false
false	false	false

### 2. || (or)

Kondisi 1	Kondisi 2	Hasil
true	true	true
true	false	true
false	true	true
false	false	false

### 3. ! (not)

Kondisi	Hasil
true	false
false	true

# Bab 8:

## Percabangan (Decision)

Dalam ilmu pemrograman, percabangan disebut dengan *decision*. Percabangan adalah suatu perintah (*statement*) dalam Bahasa C yang digunakan untuk menentukan perintah mana yang akan dijalankan berdasarkan sebuah kondisi. Setiap kondisi pasti menghasilkan pernyataan benar (*true*) atau salah (*false*).

Jika hasil dari kondisinya benar, maka perintah yang berada di dalam blok percabangan tersebut akan dijalankan. Tetapi jika hasil dari kondisinya salah, maka perintah yang berada di dalam blok percabangan tersebut tidak akan dijalankan.

### A. Aturan Pembuatan Kondisi

Kondisi tidak hanya digunakan dalam percabangan (*decision*), tetapi akan selalu digunakan juga dalam perulangan (*looping*) yang akan kita bahas di bab selanjutnya. Maka dari itu, pemahaman kita dalam pembuatan sebuah kondisi sangatlah penting dan membutuhkan pemahaman logika yang kuat.

#### Struktur Penulisan Kondisi :

```
kode_percabangan (kondisi)
{
    // perintah yang akan dijalankan jika kondisi terpenuhi;
}
```

Pembuatan suatu kondisi tidak terlepas dari penggunaan operator yang sudah kita bahas pada bab sebelumnya. Jadi kita harus bisa memahami setiap penggunaan operator berdasarkan jenis-jenisnya (Aritmatika, Penugasan, Perbandingan, dan Logika).

#### Contoh kasus untuk penggunaan kondisi

Misalkan dalam suatu program terdapat sebuah inputan untuk mengisi nilai dari variable angka yang sudah dideklarasikan sebelumnya,

```
int angka;

printf("Masukkan angka = ");
scanf("%d", &angka);
```

maka dari program tersebut dapat menghasilkan kasus-kasus untuk membuat sebuah kondisi :

- Apakah angka tersebut adalah nol ?

Kondisi :

```
...(angka == 0)
```

- Memeriksa angka apakah angka tersebut positif atau negatif ?

Kondisi angka positif :

```
...(angka > 0)
```

Kondisi angka negatif :

```
...(angka < 0)
```

- Memeriksa angka apakah angka tersebut angka ganjil atau genap ?

Kondisi angka ganjil :

```
...(angka % 2 != 0)
```

atau

```
...(angka % 2 == 1)
```

Kondisi angka genap :

```
...(angka % 2 == 0)
```

- Memeriksa angka apakah angka tersebut merupakan kelipatan angka tertentu ?

Kondisi untuk menentukan angka yang berkelipatan 3 :

```
...(angka % 3 == 0)
```

Kondisi untuk menentukan angka yang berkelipatan 7 :

```
...(angka % 7 == 0)
```



- Memeriksa suatu karakter, apakah karakter tersebut sama dengan karakter tertentu ?

Misalkan dalam suatu program terdapat sebuah inputan untuk mengisi nilai berupa karakter untuk variable grade yang sudah dideklarasikan sebelumnya,

```
char grade;  
  
printf("Masukkan grade : ");  
scanf("%c", &grade);
```

Kondisi untuk memeriksa grade, apakah grade itu adalah A ?

```
... (grade == 'A')
```

- Memeriksa suatu string, apakah string tersebut sama dengan kata atau kalimat tertentu ?

Misalkan dalam suatu program terdapat sebuah inputan untuk mengisi nilai berupa string untuk variable nama yang sudah diinisialisasi sebelumnya,

```
char nama[100] = "Fathurrahman";  
char input_nama[100];  
  
printf("Masukkan nama : ");  
gets(input_nama);
```

Kondisi untuk memeriksa variable input\_nama, apakah variable input\_nama **sama dengan** nilai dari variable nama?

```
... (strcmp(input_nama, nama) == 0)
```

atau

```
... (strcmp(input_nama, "Fathurrahman") == 0)
```

Kondisi untuk memeriksa variable input\_nama, apakah variable input\_nama **berbeda** dengan nilai dari variable nama?

```
... (strcmp(input_nama, nama) != 0)
```

atau

```
... (strcmp(input_nama, "Fathurrahman") == 1)
```

## Contoh kasus untuk penggunaan kondisi lebih dari satu perbandingan

Misalkan dalam suatu program terdapat sebuah inputan untuk mengisi nilai dari variable angka yang sudah dideklarasikan sebelumnya,

```
int angka;  
  
printf("Masukkan angka = ");  
scanf("%d", &angka);
```

maka dari program tersebut dapat menghasilkan kasus-kasus untuk membuat sebuah kondisi :

- ☐ Apakah angka tersebut merupakan angka positif **dan** genap ?  
Kondisi :

```
...(angka > 0 && angka % 2 == 0)
```

- ☐ Apakah angka tersebut merupakan angka positif **atau** lebih dari 10 ?  
Kondisi :

```
...(angka > 0 || angka > 10 == 0)
```

## B. Jenis – Jenis Perintah dalam Percabangan

Di dalam pelatihan Logika dan Algoritma (Bahasa C) ini, terdapat tiga jenis perintah yang digunakan untuk membuat sebuah percabangan yaitu :

1. if
2. switch-case
3. Operator ternary

Pada dasarnya ketiga jenis percabangan tersebut memiliki fungsi yang sama yaitu menggambarkan sebuah percabangan. Ketiga jenis perintah percabangan tersebut memiliki aturan penulisannya masing – masing. Maka ketiga jenis perintah percabangan tersebut akan kita bahas satu persatu.

### 1. if

Di dalam percabangan *if* terdapat tiga perintah untuk membuat sebuah program yang menggunakan percabangan *if*, yaitu :

- 1) if
- 2) else if
- 3) else

Perintah – perintah tersebut memiliki perannya masing – masing dalam menggambarkan sebuah program yang menggunakan percabangan *if*. Maka ketiga jenis perintah dalam percabangan *if* tersebut akan kita bahas satu persatu.

#### 1) if

*if* merupakan salah satu perintah yang menggambarkan percabangan *if* itu sendiri (selain *else* dan *else if*) sehingga perintah *if* **wajib ada di dalam sebuah percabangan**. *if* selalu diartikan sebagai **cabang pertama** dari sebuah percabangan. Setiap cabang *if* **harus mempunyai sebuah kondisi** untuk menentukan apakah kode di dalamnya dapat dieksekusi atau tidak.

### Penulisan Struktur Percabangan dengan *if* :

Penulisan struktur sebuah percabangan dengan **satu cabang** menggunakan percabangan *if* :

```
if (kondisi)
    // perintah yang akan dijalankan jika kondisi terpenuhi;
```

Penulisan struktur **dua buah percabangan** yang dituliskan dengan dua *if* :

```
if (kondisi cabang 1)
    // perintah 1 akan dijalankan tergantung kondisi cabang 1;
if (kondisi cabang 2)
    // perintah 2 akan dijalankan tergantung kondisi cabang 2;
```

Jika terdapat lebih dari satu perintah di dalam sebuah percabangan *if*, maka perintah-perintah yang ada di dalam percabangan *if* tersebut harus berada di dalam tanda kurung kurawal ({}).

```
if (kondisi)
{
    // perintah;
    // perintah;
    // perintah lainnya dan seterusnya;
}
```

Jika terdapat sebuah percabangan *if* dan terdapat lebih dari satu perintah, maka perintah yang akan dijalankan berdasarkan kondisi dari percabangan *if* tersebut adalah perintah pertama setelah percabangan *if* tersebut

```
if (kondisi)
    // perintah ini akan dijalankan jika kondisi terpenuhi;
// perintah ini langsung dijalankan tanpa tergantung
// kondisi;
```

## Contoh Program Menggunakan *if* dalam Percabangan *if*

### Contoh 1 :

```
int bilangan;

printf("Masukkan bilangan = "); scanf("%d", &bilangan);

if (bilangan > 20)
    printf("\nHallo, perintah ini ada di dalam percabangan\n");

    printf("\nHai,perintah ini ada di luar percabangan \n");

getch();
```

#### Output Contoh 1 :

- ☐ Jika menginputkan bilangan yang > 20

Masukkan bilangan = 90

Hallo, perintah ini ada di dalam percabangan

Hai, perintah ini ada di luar percabangan

- ☐ Jika menginputkan bilangan yang < 20

Masukkan bilangan = 10

Hai, perintah ini ada di luar percabangan

### Contoh 2 :

```
int bilangan;

printf("Masukkan bilangan = ");
scanf("%d", &bilangan);

if (bilangan > 20)
    printf("\nHallo, perintah ini ada di dalam cabang pertama\n");
    if (bilangan > 10)
        printf("\nHai, perintah ini ada di dalam cabang kedua\n");

    printf("\nHai, perintah ini ada di luar percabangan \n");

getch();
```

### Output Contoh 2 :

- Jika menginputkan bilangan yang  $> 20$  dan  $> 10$

```
Masukkan bilangan = 30

Hallo, perintah ini ada di dalam cabang pertama

Hai, perintah ini ada di dalam cabang kedua

Hai, perintah ini ada di luar percabangan
```

- Jika menginputkan bilangan yang  $< 20$  dan  $> 10$

```
Masukkan bilangan = 15

Hai, perintah ini ada di dalam cabang kedua

Hai, perintah ini ada di luar percabangan
```

- Jika menginputkan bilangan yang  $< 10$

```
Masukkan bilangan = 4

Hai, perintah ini ada di luar percabangan
```

### Contoh 3 :

```
int bilangan;
printf("Masukkan bilangan = "); scanf("%d", &bilangan);
if (bilangan < 10)
{
    printf("\nHallo, perintah ini ada di dalam cabang pertama\n");
    printf("Bilangan yang kamu masukkan kurang dari 10\n");
    printf("Bilangan yang kamu masukkan adalah %d\n", bilangan);
}
if (bilangan < 20)
{
    printf("\nHai, perintah ini ada di dalam cabang kedua\n");
    printf("Bilangan yang kamu masukkan kurang dari 20\n");
    printf("Bilangan yang kamu masukkan adalah %d\n", bilangan);
}
printf("\nPerintah ini berada di luar percabangan");
getch();
```

### Output Contoh 3 :

- Jika menginputkan bilangan yang  $< 10$  dan  $< 20$

```
Masukkan bilangan = 5

Hallo, perintah ini ada di dalam cabang pertama
Bilangan yang kamu masukkan kurang dari 10
Bilangan yang kamu masukkan adalah 5

Hai, perintah ini ada di dalam cabang kedua
Bilangan yang kamu masukkan kurang dari 20
Bilangan yang kamu masukkan adalah 5

Perintah ini berada di luar percabangan
```

- Jika menginputkan bilangan yang  $> 10$  dan  $< 20$

```
Masukkan bilangan = 15

Hai, perintah ini ada di dalam cabang kedua
Bilangan yang kamu masukkan kurang dari 20
Bilangan yang kamu masukkan adalah 15

Perintah ini berada di luar percabangan_
```

- Jika menginputkan bilangan yang  $> 10$  dan  $> 20$

```
Masukkan bilangan = 25

Perintah ini berada di luar percabangan_
```

## 2) else

*else* adalah perintah percabangan yang **selalu digunakan sebagai cabang terakhir** dari sebuah percabangan *if* sehingga **tidak bisa dijadikan cabang pertama** dari sebuah percabangan.

Perintah – perintah yang ada di cabang *else* akan otomatis dijalankan jika kondisi dari cabang – cabang sebelumnya tidak terpenuhi sehingga perintah percabangan *else* bisa dibuat **tanpa kondisi**

Penggunaan percabangan *else* **tidak wajib digunakan** di dalam sebuah percabangan *if*.

## Penulisan Struktur Percabangan dengan *if-else* :

```
if (kondisi)
    // perintah yang akan dijalankan jika kondisi terpenuhi;
else
{
    // perintah yang akan dijalankan jika kondisi pada if di
    // atas tidak terpenuhi;
}
```

## Contoh Program Menggunakan *else* dalam Percabangan *if*

### Contoh 1 :

```
int bilangan;

printf("Masukkan bilangan : ");
scanf("%d", &bilangan);

if (bilangan >= 0)
    printf("%d adalah bilangan positif", bilangan);
else
    printf("%d adalah bilangan negatif", bilangan);

getch();
```

### Output Contoh 1 :

☐ Jika menginputkan bilangan  $\geq 0$

```
Masukkan bilangan : 35
35 adalah bilangan positif
```

☐ Jika menginputkan bilangan yang tidak  $\geq 0$

```
Masukkan bilangan : -89
-89 adalah bilangan negatif
```



### Contoh 2 :

```
int bilangan;

printf("Masukkan bilangan : ");
scanf("%d", &bilangan);

if (bilangan % 2 == 0)
    printf("%d adalah bilangan genap", bilangan);
else
    printf("%d adalah bilangan ganjil", bilangan);

getch();
```

#### Output Contoh 2 :

- Jika memasukkan bilangan yang genap

```
Masukkan bilangan : 68
68 adalah bilangan genap
```

- Jika memasukkan bilangan yang ganjil

```
Masukkan bilangan : 5
5 adalah bilangan ganjil_
```

### Contoh 3 :

```
int bilangan;
printf("Masukkan bilangan : "); scanf("%d", &bilangan);

if (bilangan % 5 == 0)
    printf("%d adalah bilangan kelipatan 5", bilangan);
else
    printf("%d adalah bukan bilangan kelipatan 5", bilangan);
```

#### Output Contoh 3 :

- Jika memasukkan bilangan yang merupakan kelipatan 5

```
Masukkan bilangan : 55
55 adalah bilangan kelipatan 5
```

- Jika memasukkan bilangan yang bukan merupakan kelipatan 5

```
Masukkan bilangan : 46
46 adalah bukan bilangan kelipatan 5
```

### 3) else if

*else if* adalah sebuah perintah dalam percabangan *if* yang **harus memiliki kondisi** untuk menentukan apakah kode di dalamnya dapat dieksekusi atau tidak.

Percabangan *else if* **tidak bisa menjadi cabang pertama** dari sebuah percabangan karena cabang pertama hanya bisa dibuat dengan perintah *if*. Tetapi cabang *else if* **bisa dijadikan cabang terakhir** dari sebuah percabangan.

Jika kondisi dari suatu percabangan *else if* sudah benar atau terpenuhi, maka program akan **berhenti memeriksa setiap kondisi dari cabang – cabang selanjutnya** meskipun masih ada cabang lain yang kondisinya bisa terpenuhi.

#### Penulisan Struktur Percabangan dengan *if-(else-if)-else* :

```
if (kondisi ke-1)
    // statement jika kondisi ke-1 benar; else if
(kondisi ke-2)
    // statement jika kondisi ke-1 salah dan kondisi ke-2 benar; else if
(kondisi ke-3)
    // statement jika kondisi ke-2 salah dan kondisi ke-3 benar;
```

Percabangan *if* dan *else if* bisa dikombinasikan dengan percabangan *else*. Sesuai dengan yang dijelaskan sebelumnya, percabangan *if* sebagai cabang dan percabangan *else* menjadi cabang yang terakhir dari sebuah percabangan.

```
if (kondisi 1)
    // statement jika kondisi 1 benar; else if
(kondisi 2)
    // statement jika kondisi 1 salah tapi kondisi 2 benar;
else
    // statement jika semua kondisi di atas salah;
```

## Contoh Program Menggunakan *else if* dalam Percabangan *if*

### Contoh 1 :

```
int bilangan;

printf("Masukkan bilangan : ");
scanf("%d", &bilangan);

if (bilangan > 0)
    printf("%d adalah bilangan positif", bilangan); else if
(bilangan < 0)
    printf("%d adalah bilangan negatif", bilangan);
else
    printf("%d adalah angka nol", bilangan);

getch();
```

### Output Contoh 1 :

- Jika memasukkan bilangan yang > 0

```
Masukkan bilangan : 78
78 adalah bilangan positif
```

- Jika memasukkan bilangan yang < 0

```
Masukkan bilangan : -5
-5 adalah bilangan negatif
```

- Jika memasukkan 0

```
Masukkan bilangan : 0
0 adalah angka nol
```

## 2. switch-case

Percabangan *switch-case* adalah sebuah perintah untuk memilih hasil dari suatu kondisi yang memiliki nilai-nilai konstan (tetap/pasti). Oleh karena itu fungsi *switch-case* biasanya digunakan untuk variabel yang bertipe integer atau karakter.

### Penulisan Struktur Percabangan *switch-case* :

```
switch(kondisi/variabel)
{
    case hasil/nilai ke-1:
        // perintah jika hasil/nilai dari kondisi/variable ke-1 true;
        // perintah jika hasil/nilai dari kondisi/variable ke-1 true;
        break;
    case hasil/nilai ke-2:
        // perintah jika hasil/nilai dari kondisi/variable ke-2 false;
        break;
    default:
        // perintah jika nilainya bukan hasil dari kondisi;
        // perintah jika nilainya bukan hasil dari kondisi;
        // perintah jika nilainya bukan hasil dari kondisi;
}
```

### Catatan :

- **break;** adalah sebuah perintah yang merupakan *jump statement*. Fungsi dari *break* dalam struktur percabangan *switch-case* ini adalah untuk memberhentikan program agar tidak menjalankan perintah yang ada di dalam case lain di bawah case yang nilainya memenuhi kondisi di dalam *switch()*.
- **default:** tidak harus digunakan saat menggunakan percabangan *switch-case* karakteristiknya sama seperti percabangan *else* pada percabangan *if*.
- Perintah – perintah yang ada di cabang *default* kan otomatis dijalankan jika hasil kondisi atau nilai variable dari cabang – cabang (*case*) sebelumnya tidak terpenuhi

## Contoh Program Menggunakan Percabangan *switch-case*

### Contoh 1 :

```
int nomor;

printf("Masukkan nomor hari : ");
scanf("%d", &nomor);

switch(nomor)
{
    case 1:
        printf("Nomor %d adalah hari senin", nomor);
        break;
    case 2:
        printf("Nomor %d adalah hari selasa", nomor);
        break;
    case 3:
        printf("Nomor %d adalah hari rabu", nomor);
        break;
    case 4:
        printf("Nomor %d adalah hari kamis", nomor);
        break;
    case 5:
        printf("Nomor %d adalah hari jumat", nomor);
        break;
    case 6:
        printf("Nomor %d adalah hari sabtu", nomor);
        break;
    case 7:
        printf("Nomor %d adalah hari minggu", nomor);
        break;
    default :
        printf("Nomor %d invalid", nomor);
}

getch();
```

### Output Contoh 1 :

- ☐ Jika menginputkan nomor di antara 1 sampai 9

```
Masukkan nomor hari : 4
Nomor 4 adalah hari kamis_
```

- ☐ Jika menginputkan nomor selain nomor di antara 1 sampai 9

```
Masukkan nomor hari : -4
Nomor -4 invalid_
```

## Contoh 2 (tanpa menggunakan *break*) :

```
char huruf_kecil;

printf("Masukkan huruf kecil : ");
scanf("%c", &huruf_kecil);

switch(huruf_kecil)
{
    case 'a':
        printf("a adalah huruf vokal\n");
    case 'i':
        printf("i adalah huruf vokal\n");

    case 'u':
        printf("u adalah huruf vokal\n");
    case 'e':
        printf("e adalah huruf vokal\n");
    case 'o':
        printf("o adalah huruf vokal\n");
    default :
        printf("%c bukan huruf vokal\n", huruf_kecil);
}

getch();
```

### Output Contoh 2 :

- ☐ Jika menginputkan huruf kecil vokal

```
Masukkan huruf kecil : a
a adalah huruf vokal
i adalah huruf vokal
u adalah huruf vokal
e adalah huruf vokal
o adalah huruf vokal
a bukan huruf vokal
```

- ☐ Jika menginputkan huruf kecil bukan vokal

```
Masukkan huruf kecil : j
j bukan huruf vokal
```

### Contoh 3 (tanpa menggunakan *break*) :

```
int angka_1, angka_2, pilih_opt, hasil;
char opt;

printf("Operator :\n1. penjumlahan (+)\n2. pengurangan (-)\n");
printf("3. perkalian (x)\n4. pembagian (/)\n5. modulus (%%)\n");

printf("Pilih nomor operator = ");
scanf("%d", &pilih_opt);

printf("\nMasukkan angka pertama = ");
scanf("%d", &angka_1);

printf("Masukkan angka kedua = ");
scanf("%d", &angka_2);

switch(pilih_opt) {
    case 1 :
        opt = '+';
        hasil = angka_1 + angka_2;
    case 2 :
        opt = '-';
        hasil = angka_1 - angka_2;
    case 3 :
        opt = 'x';
        hasil = angka_1 * angka_2;
    case 4 :
        opt = '/';
        hasil = angka_1 / angka_2;
    case 5 :
        opt = '%';
        hasil = angka_1 % angka_2;
    default :
        printf("\nOperator salah!!!\n");
}

printf("\nHasil :\n%d %c %d = %d", angka_1, opt, angka_2, hasil);

getch();
```

### Output Contoh 3 :

- Jika menginputkan angka 1

```
Operator :  
1. penjumlahan (+)  
2. pengurangan (-)  
3. perkalian (x)  
4. pembagian (/)  
5. modulus (%)  
Pilih nomor operator = 3  
  
Masukkan angka pertama = 12  
Masukkan angka kedua   = 12  
  
Operator salah!!!  
  
Hasil :  
12 % 12 = 0_
```

Coding ulang contoh 2 dan contoh 3, lalu gunakanlah *break* pada sebagai *statement* terakhir dari masing-masing *case* di contoh 2 dan contoh 3.

Bandingkan jika kita menggunakan *break* dan tidak.



### 3. Operator *Ternary*

Operator *ternary* adalah salah satu jenis percabangan yang kondisi dan kedua hasil perintah berdasarkan hasil kondisinya (*true* dan *false*) ditulis dalam satu baris.

## Penulisan Struktur Percabangan dengan Operator *Ternary*

kondisi ? hasil perintah jika kondisi benar : hasil perintah jika kondisi salah

Agar kita dapat lebih mudah memahami struktur dari percabangan operator *ternary*, kita harus memahami posisi kode-kode yang ada di dalam operator *ternary*.

kondisi      ?      hasil perintah jika kondisi benar      :      hasil perintah jika kondisi salah

Keterangan latar warna :

- **Hijau** : Bagian ini digunakan untuk menulis kondisi yang dibutuhkan untuk menentukan hasil perintah yang akan dijalankan.
- **Tanda tanya (?)** : Digunakan sebagai pemisah antara bagian penulisan kondisi dengan hasil perintah yang akan dijalankan berdasarkan hasil dari kondisi tersebut (*true* dan *false*).
- **Biru** : Bagian ini digunakan untuk menuliskan hasil perintah yang akan dijalankan jika hasil dari kondisi tersebut *true*.
- **Titik dua (:)** : Bagian ini digunakan untuk memisahkan antara hasil perintah yang akan dijalankan jika hasil dari kondisi tersebut *true* dengan hasil perintah yang akan dijalankan jika hasil dari kondisi tersebut *false*.
- **Oranye** : Bagian ini digunakan untuk menuliskan hasil perintah yang akan dijalankan atau didapatkan jika hasil dari kondisi tersebut *false*.

Di dalam penggunaannya, operator *ternary* dapat digunakan untuk berbagai hal yang dapat membantu kita untuk membuat sebuah percabangan, antara lain untuk :

- 1) Menjalankan sebuah perintah berdasarkan kondisi
- 2) Menghasilkan nilai yang akan dicetak

Untuk memperdalam fungsi atau penggunaan dari percabangan operator *ternary* maka kita akan membahas penggunaan dari percabangan operator *ternary* satu persatu :

### 1) Menjalankan sebuah perintah berdasarkan kondisi

Hasil perintah dari percabangan operator *ternary* dapat berupa sebuah **perintah atau pernyataan** yang hanya bisa dijalankan berdasarkan kondisi yang kita tulis.

#### Contoh 1 :

```
int bilangan;  
  
printf("Masukkan bilangan : ");  
scanf("%d", &bilangan);  
  
bilangan % 2 == 0 ? printf("Genap") : printf("Ganjil");  
  
getch();
```

#### Output Contoh 1 :

- ☐ Jika menginputkan angka 5

```
Masukkan bilangan : 5  
Ganjil
```

- ☐ Jika menginputkan angka 8

```
Masukkan bilangan : 8  
Genap_
```

## 2) Menghasilkan nilai yang akan dicetak

Hasil perintah dari percabangan operator *ternary* dapat berupa sebuah **nilai yang akan dicetak** melalui fungsi `printf()` yang hanya bisa dihasilkan berdasarkan kondisi yang kita tulis.

### Contoh 1 :

```
int stok;  
  
printf("Masukkan jumlah stok : ");  
scanf("%d", &stok);  
  
printf("Keterangan stok : %s", stok > 0 ? "Tersedia" : "Habis");  
  
getch();
```

#### Output Contoh 1 :

- Jika menginputkan jumlah stok = 100

```
Masukkan jumlah stok : 100  
Keterangan stok = Tersedia
```

- Jika menginputkan jumlah stok = 0

```
Masukkan jumlah stok : 0  
Keterangan stok = Habis_
```

## C. Percabangan Bersarang (*Nested Decision*)

Percabangan bersarang atau dalam bahasa Inggris disebut *nested decision* adalah sebuah struktur percabangan yang memiliki percabangan lagi di dalamnya. Setiap jenis dari percabangan (*if*, *switch-case*, *ternary*) dapat digunakan menggunakan struktur percabangan bersarang ini.

Struktur dari percabangan bersarang akan kita bahas satu persatu berdasarkan jenis-jenis percabangan yang sudah kita pelajari sebelumnya

### Struktur Percabangan Bersarang pada Berbagai Jenis

#### Percabangan :

##### 1. Percabangan *if*

Contoh struktur percabangan bersarang menggunakan percabangan *if* secara lengkap (*if*, *else-if*, dan *else*) :

```
if (kondisi)
{
    // perintah atau statement;
    if (kondisi)
    {
        // perintah atau statement;
    }
    else if (kondisi)
    {
        // perintah atau statement;
    }
    else
    {
        // perintah atau statement;
    }
    // perintah atau statement;
}
else if (kondisi)
{
    // perintah atau statement
    if (kondisi)
    {
        // perintah atau statement;
    }
    else if (kondisi)
    {
        // perintah atau statement;
    }
}
```

```

else
{
    // perintah atau statement;
}
}

// perintah atau statement
else
{
    // perintah atau statement
    if (kondisi)
    {
        // perintah atau statement;
    }
    else if (kondisi)
    {
        // perintah atau statement;
    }
    else
    {
        // perintah atau statement;
    }
    // perintah atau statement
}

```

## 2. Percabangan *switch-case*

Contoh struktur percabangan bersarang menggunakan percabangan *switch-case* secara lengkap (*case, default*) :

```

switch(kondisi/variable)
{
    case hasil/nilai :
        // perintah atau statement;
        switch(kondisi/variable)
        {
            case hasil/nilai :
                // perintah atau statement;
                break;
            default :
                // perintah atau statement;
                break;
        }
        // perintah atau statement;
        break;

    default :
        // perintah atau statement;
        switch(kondisi/variable)
        {
            case hasil/nilai :
                // perintah atau statement;
                break;
            default :
                // perintah atau statement;
        }
}

```

```

        break;
    }
    //perintah atau statement;
    break;
}

```

### 3. Percabangan Operator *Ternary*

Contoh struktur percabangan bersarang menggunakan percabangan operator *ternary* secara lengkap :

```

kondisi ke-1 ? benar ke-1 :          kondisi ke-2 ? benar ke-2: salah ke-2

```

## Penggunaan Percabangan Bersarang pada Berbagai

### Jenis Percabangan :

#### 1. Percabangan *if*

Contoh penggunaan percabangan bersarang menggunakan percabangan *if* (*if* dan *else*) :

```

int angka_1 = 7, angka_2 = 7;

if (angka_1 == 7)
{
    if (angka_2 == 7)
        printf("Variable angka_1 dan angka_2 bernilai 7");
    else
        printf("Hanya variable angka_1 yang bernilai 7");
}
else
    printf("Variable a tidak bernilai 7");

getch();

```

**Output:**

```

Variable angka_1 dan angka_2 bernilai 7

```

## 2. Percabangan *switch-case*

Contoh penggunaan percabangan bersarang menggunakan percabangan *switch-case* secara lengkap (*case*, *default*) :

```
int angka = 9;
switch(angka % 2)
{
    case 0 :
        printf("Angka genap ");
        switch(angka > 0 == 0)
        {
            case 0 :
                printf("dan positif");
                break;
            default :
                printf("dan negatif");
                break;
        }
        break;
    default :
        printf("Angka ganjil ");
        switch(angka > 0 == 0)
        {
            case 0 :
                printf("dan positif");
                break;
            default :
                printf("dan negatif");
                break;
        }
        break;
}
getch();
```

**Output:**

```
Angka ganjil dan positif_
```

### 3. Percabangan Operator *Ternary*

Contoh program percabangan bersarang menggunakan percabangan operator *ternary* secara lengkap :

```
int berat = 55, tinggi = 170;

printf("Kriteria = %s", tinggi >= 160 && tinggi <= 180 ? "Ideal"
      : berat > 55 ? "Ideal" : "Jaga Asupan");

getch();
```

atau

```
int berat = 55, tinggi = 170;

printf("Kriteria = ");

tinggi >= 160 && tinggi <= 180 ? printf("Ideal") : berat > 55 ?
    printf("Ideal") : printf("Jaga Asupan");

getch();
```

**Output :**

```
Kriteria = Ideal_
```

Dalam perulangan bersarang, sebenarnya **tidak hanya dapat dilakukan oleh jenis percabangan yang sama saja**. Tetapi kita bisa menggunakan jenis percabangan yang berbeda dengan percabangan yang ada di dalamnya contoh :

```
if (kondisi)
{
    // perintah;
    switch(kondisi/variable)
    {
        case hasil/nilai :
            // perintah;
            break;
        default :
            // perintah;
    }
    // perintah;
}
```



## Bab 9:

# Perulangan (Looping)

Dalam ilmu pemrograman, perulangan disebut dengan *looping*. Perulangan adalah suatu perintah (*statement*) dalam Bahasa C yang digunakan untuk menentukan perintah mana yang akan dijalankan berdasarkan sebuah kondisi secara berulang-ulang. Setiap kondisi pasti menghasilkan pernyataan benar (*true*) atau salah (*false*). Jadi hanya perintah tertentu saja yang akan dijalankan berulang-ulang sebanyak kondisi yang kita buat.

Penggunaan perulangan atau *looping* dalam pemrograman sangat penting karena dapat mempersingkat penulisan kode yang kita buat hanya dengan beberapa perintah—perintah dari perintah perulangan. Kita juga harus bisa menentukan sebanyak apa perintah yang akan dijalankan berulang-ulang dan sampai kapan perulangan itu akan berhenti.

### A. Bagian - Bagian Dasar Perintah Perulangan

Dalam membuat sebuah program yang menggunakan perulangan, terdapat bagian–bagian yang wajib ada di dalam sebuah perulangan agar program perulangan tersebut dapat berjalan sesuai yang kita inginkan. Bagian–bagian ini wajib kita tulis di sebuah program perulangan karena merupakan **standar yang harus ada di dalam sebuah program perulangan**. Struktur atau bagian–bagian dari sebuah perulangan :

1. *Initialization*
2. *Condition*
3. *Final*

Bagian–bagian ini akan kita bahas pengertiannya terlebih dahulu satu persatu, sedangkan untuk letak posisinya di dalam sebuah program perulangan akan dibahas sekaligus bersamaan dengan jenis–jenis perintah pada perulangan di sub-bab selanjutnya.

## 1. *Initialization*

Inisialisasi dalam sebuah program perulangan adalah proses atau sebagai langkah awal dalam pembuatan program perulangan, yaitu membuat sebuah variable untuk membantu kita dalam menjalankan sebuah perulangan. Nilai awal yang diinisialisasi biasanya dimulai dari 0 agar perulangan berjalan dari angka yang paling kecil.

```
int a = 0;
```

## 2. *condition*

Kondisi dalam sebuah program perulangan adalah batas dari sebuah perulangan agar perulangan tersebut dapat berhenti sesuai dengan yang kita tentukan. Dalam penulisan kondisinya, biasanya akan membandingkan variable yang sudah diinisialisasi untuk membantu jalannya perulangan dengan batas nilai yang kita tentukan.

```
...(a < 5)
```

atau

```
int batas = 5;
```

```
...(a < batas)
```

atau

```
int batas;
```

```
printf("Masukkan batas : "); scanf("%d", &batas);
```

```
...(a < batas)
```

## 3. *final*

*final* dalam sebuah program perulangan adalah perintah terakhir dari sebuah perulangan yang mengubah nilai dari variable yang sudah diinisialisasi untuk membantu jalannya perulangan agar perulangan bisa berjalan.

```
a++;
```

variable++ disebut *increment* atau kenaikan

variable-- disebut *decrement* atau penurunan

## B. Jenis – Jenis Perintah dalam Perulangan

Dalam pemrograman bahasa C terdapat tiga jenis perintah yang digunakan untuk membuat sebuah program perulangan yaitu :

1. *while*
2. *do-while*
3. *for*

Ketiga jenis perintah perulangan tersebut memiliki aturan penulisannya masing–

masing. Setiap jenis–jenis perintah dalam perulangan memiliki **standar penulisan bagian-bagian yang sama**, yang membedakan hanyalah posisi peletakannya saja. Kita akan membahas jenis-jenis perulangan dan bagian-bagian dasarnya satu per satu.

### 1. *while*

*while* adalah perintah perulangan yang paling sederhana. Bagian perintah perulangan yang ada di dalam fungsi (yang ada di dalam kurung ()) *while* hanya membutuhkan kondisi atau *condition* sebagai syarat untuk berapa kali perulangan tersebut akan berjalan.

#### **Penulisan Struktur Perulangan dengan *while* :**

Jika hanya ada satu perintah di dalam perulangan *while* :

```
initialization;  
  
while (condition)  
    // perintah yang dijalankan jika kondisi terpenuhi;
```

Jika ada lebih dari satu perintah di dalam perulangan *while* :

```
initialization;  
  
while (condition)  
{  
    // perintah yang dijalankan jika kondisi terpenuhi;  
    // perintah lainnya;  
    // ...  
    final;  
}
```

## Contoh Program Menggunakan Perulangan *while*

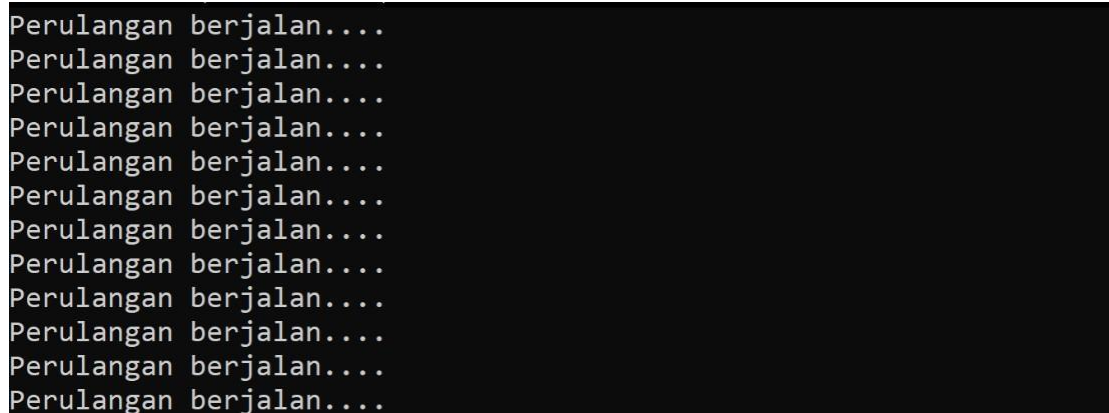
### Contoh 1 :

```
int i = 0;

while(i < 10)
{
    printf("Perulangan berjalan ....\n");
}

getch();
```

### Output Contoh 1 :



```
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
Perulangan berjalan....
```

Ouput dari Contoh 1 akan menyebabkan ***infinity loop*** yaitu **perulangan yang tidak akan pernah berakhir**. Hal ini disebabkan karena nilai variabel *i* selalu 0 dan tidak pernah berubah, sehingga nilai variabel *i* selalu kurang dari 10.

Jadi kita perlu menambahkan perintah untuk mengubah nilai *i* selama program perulangan sedang berjalan sebagai *final*.

### Contoh 2 :

```
int i = 1;
while (i < 10)
{
    printf("Perulangan berjalan...\n");
    i++;
}

getch();
```

Nilai *i* akan bertambah +1 setiap perulangan tersebut berjalan. Jadi ketika nilai *i* mencapai 10 maka perulangan akan berhenti.

### Output Contoh 2:

```
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
Perulangan berjalan...
```

## 2. *do-while*

*do-while* adalah salah satu perintah perulangan yang akan menjalankan perintahnya terlebih dahulu, kemudian kondisinya akan dijalankan di akhir. Jika hasil perbandingan dari kondisi tersebut terpenuhi maka akan mengulang kembali ke bawah kode *do*.

### Penulisan Struktur Perulangan dengan *do-while* :

Jika hanya ada satu perintah di dalam perulangan *do-while* :

```
initialization;
do
    // perintah;
while (condition);
```

Jika ada lebih dari satu perintah di dalam perulangan *do-while* :

```
initialization;
do
{
    // perintah 1;
    // perintah 2;
    // ...
    final;
} while (condition);
```

Setiap perintah perulangan *do-while* selalu diakhiri titik koma atau *semicolon* (;).

## Contoh Program Menggunakan Perulangan *while*

### Contoh 1 :

```
int s = 0;

do
{
    printf("Saya sedang belajar do-while\n");
    s++;
} while(s <

5); getch();
```

### Output Contoh 1 :

```
Saya sedang belajar do-while
Saya sedang belajar do-while
Saya sedang belajar do-while
Saya sedang belajar do-while
Saya sedang belajar do-while
_
```

## Contoh 2 :

```
int u = 0;

do
    printf("%d ", ++u);
while(u < 5);

getch();
```

### Output Contoh 2 :

```
1 2 3 4 5
```

## 3. *for*

*for* adalah salah satu perintah perulangan yang bagian perulangannya (*initialization*, *condition*, dan *finalnya*) berada di dalam satu baris, yaitu di dalam tanda kurung (()) setelah kode *if*.

### Penulisan Struktur Perulangan dengan *do-while* :

Jika hanya ada satu perintah di dalam perulangan *for* :

```
for (initialization; condition; final)
    // perintah;
```

atau

```
initialization;

for (; condition; final)
    // perintah;
```

atau

```
initialization;

for (; condition;)
    final;
```

Jika ada lebih dari satu perintah di dalam perulangan *for* :

```
for (initialization; condition; final)
{
    // perintah 1;
    // perintah 2;
    // perintah 3;
}
```

atau

```
initialization;

for (; condition; final)
{
    // perintah 1;
    // perintah 2;
    // perintah 3;
}
```

atau

```
initialization;

for (; condition;)
{
    // perintah 1;
    // perintah 2;
    // perintah 3;
    final;
}
```

atau

```
for (initialization; condition;)
{
    // perintah 1;
    // perintah 2;
    // perintah 3;
    final;
}
```



## Contoh Program Menggunakan Perulangan *for*

### Contoh 1 :

```
for(int i = 0; i < 10; i++)  
{  
    printf("%d", i);  
    printf(" Saya sedang belajar loop\n");  
}  
  
getch();
```

### Output Contoh 1 :

```
0 Saya sedang belajar loop  
1 Saya sedang belajar loop  
2 Saya sedang belajar loop  
3 Saya sedang belajar loop  
4 Saya sedang belajar loop  
5 Saya sedang belajar loop  
6 Saya sedang belajar loop  
7 Saya sedang belajar loop  
8 Saya sedang belajar loop  
9 Saya sedang belajar loop
```

### Contoh 2 :

```
int i = 0;  
  
for(; i < 10; i++)  
    printf("%d ", i);  
  
getch();
```

### Output Contoh 2 :

```
0 1 2 3 4 5 6 7 8 9
```

## C. Perulangan Bersarang (Nested Looping)

Perulangan bersarang atau dalam bahasa Inggris disebut *nested looping* adalah sebuah program perulangan yang di dalamnya terdapat perulangan lagi. Setiap jenis dari perulangan (*while*, *do-while*, *for*) dapat digunakan menggunakan struktur perulangan bersarang ini.

Struktur dari perulangan bersarang akan kita bahas satu persatu berdasarkan jenis-jenis perulangan yang sudah kita pelajari sebelumnya

### Struktur Percabangan Bersarang pada Berbagai Jenis Perulangan :

#### 1. Perulangan *while*

```
initialization_1;
while (condition_1)
{
    // perintah;
    initialization_2;
    while (condition_2)
    {
        // perintah;
        final_2;
    }
    // perintah;
    final_1;
}
```

#### 2. Perulangan *do-while*

```
initialization_1;
do
{
    // perintah;
    initialization_2;
    do
    {
        // perintah;
        final_2;
    } while (condition_2);
    // perintah; final_1;
} while (condition_1);
```

### 3. Perulangan *for*

```
for (initialization_1; condition_1; final_1)
{
    // perintah;
    for (initialization_2; condition; final_2)
    {
        // perintah;
    }
    // perintah;
}
```

## Penggunaan Percabangan Bersarang pada Berbagai Jenis Perulangan :

### 1. Perulangan *while*

```
int x=0;
while (x<5)
{
    int y=0;
    while (y<5)
    {
        if(x == 0 || x == 4 || y == 0 || y == 4)
            printf("x ");
        else
            printf(" ");
        y++;
    }
    printf("\n");
    x++;
}
```

## 2. Perulangan *do-while*

```
int x=0;
do
{
    int y=0;
    do
    {
        if(x == 0 || x == 4 || y == 0 || y == 4)
            printf("* ");
        else
            printf(" ");
        y++;
    }while (y<5);
    printf("\n");
    x++;
}while (x<5);
```

## 3. Perulangan *for*

```
for(int x=0; x<5; x++)
{
    for(int y=0; y<5; y++)
    {
        if(x == 0 || x == 4 || y == 0 || y == 4)
            printf("* ");
        else
            printf(" ");
    }
    printf("\n");
}
```

**Output :**

```
* * * * *
*       *
*       *
*       *
*       *
* * * * *
```

# Bab 10:

## Perintah Pelompatan (Jump Statement)

Perintah pelompatan atau dalam bahasa Inggris disebut *jump statement* pada umumnya digunakan pada saat kita membuat sebuah program perulangan (*looping*) kecuali *goto* dan *exit* yang tidak harus ditulis dalam program perulangan. Secara umum, fungsi dari penggunaan perintah pelompatan adalah untuk memindahkan, melewati, ataupun memberhentikan alur perulangan tersebut. Di dalam pelatihan Logika dan Algoritma (Bahasa C) kita akan mempelajari empat perintah pelompatan, yaitu :

1. *break*
2. *continue*
3. *goto*
4. *exit*

### A. Jenis-Jenis Perintah Pelompatan

#### 1. *break*

kata “*break*” dalam bahasa Indonesia dapat diartikan sebagai merusak ataupun pemutusan. Sedangkan dalam pemrograman bahasa C, *break* hanya bisa digunakan di dalam struktur percabangan *switch-case* dan di struktur semua jenis perulangan (*while*, *do-while*, dan *for*) untuk **menghentikan atau keluar dari alur sebuah perulangan** dan percabangan *switch-case*.

#### Penggunaan *break*

```
for (int i = 0; i < 5; i++)  
{  
    printf("Perintah ke-%d\n", i);  
    if (i == 2)  
        break;  
}  
getch();
```

### Output :

```
Perintah ke-0  
Perintah ke-1  
Perintah ke-2  
_
```

Ketika nilai variable perulangan *i* sudah mencapai 2, maka program akan memasuki percabangan *if* yang di dalamnya terdapat perintah peloncatan *break* sehingga perulangan akan berhenti atau keluar dari alur perulangan

## 2. *continue*

Perintah pelompatan *continue* adalah perintah untuk melewati atau mengabaikan perintah yang ada dibawahnya dan akan **kembali keatas untuk melanjutkan perulangan**. Sehingga dapat dikatakan jika di dalam sebuah perulangan dan program menjalankan perintah peloncatan *continue*, maka perintah peloncatan *continue* tersebut berfungsi **sebagai bagian final** dari perulangan tersebut.

### Penggunaan *continue* dengan Perulangan *for*

```
for (int i = 1; i <= 5; i++)  
{  
    printf("Hallo %d\n", i);  
    continue;  
    printf("Hai %d\n", i);  
}  
  
getch();
```

## Output :

```
Hallo 1
Hallo 2
Hallo 3
Hallo 4
Hallo 5
_
```

Perintah yang ada di bawah atau setelah perintah peloncatan *continue* tidak akan dijalankan sesuai dengan pengertian perintah peloncatan *continue* yang kita pelajari.

## Penggunaan *continue* dengan Perulangan *while*

```
int x = 0;
while(x < 5)
{
    printf("Awal while\n");
    if (x == 2)
    {
        printf("Awal if\n");
        x++; // final
        continue;
        printf("Akhir if\n");
    }
    printf("Akhir while\n\n");
    x++; // final
}
printf("Di luar while\n");

getch();
```

## Output :

```
Awal while
Akhir while

Awal while
Akhir while

Awal while
Awal if
Awal while
Akhir while

Awal while
Akhir while

Di luar while
```

### 3. *goto*

Perintah pelompatan *goto* terdiri dari dua buah kata dalam bahasa Inggris yaitu “*go*” yang artinya **pergi** dan “*to*” yang artinya untuk atau **ke**. Jadi, perintah pelompatan *goto* ini digunakan untuk **memindahkan alur program** dari baris tertentu ke baris lain yang kita pilih di baris mana saja.

#### Penggunaan *goto* dengan Perulangan *for*

```
for (int i = 0; i < 5; i++)
{
    printf("cek 1\n");
    if (i == 2)
        goto x;

    printf("cek 2\n");

    if (i == 3)
        goto h;

    x:
    printf("cek 3\n\n");
}

h :
printf("cek 4\n");

getch();
```



### Output :

```
cek 1
cek 2
cek 3

cek 1
cek 2
cek 3

cek 1
cek 3

cek 1
cek 2
cek 4
```

## 4. *exit*

Perintah pelompatan *exit* digunakan untuk **memberhentikan program** dimanapun yang kita mau. Perintah pelompatan *exit* adalah salah satu fungsi dari *library* : *standard library* (<stdlib.h>), sehingga jika kita ingin menggunakan perintah pelompatan *exit* ini maka kita harus menyertakan *library*-nya.

```
int angka = 0;

do
{
    printf("Masukan angka 1 untuk keluar program\n");
    printf("Masukan angka : "); scanf("%d", &angka); if
(angka == 1)
        exit(0);
    else
        printf("Anda menginputkan angka %d\n\n", angka);
} while(angka !=
1); getch();
```

### Output :

- Jika menginput angka selain 1

```
Masukan angka 1 untuk keluar program
Masukan angka : 89
Anda menginputkan angka 89

Masukan angka 1 untuk keluar program
Masukan angka :
```

- Jika menginput angka 1

*Output akan langsung ditutup karena sudah keluar dari program*

# Bab 11:

## Array

### A. Pengertian array

Jika diartikan secara bahasa, maka array dapat diartikan sebagai himpunan, susunan, atau jajaran. Sedangkan di dalam dunia pemrograman, array adalah variabel yang dapat menyimpan lebih dari 1 data dengan tipe yang sama. Misalnya sekumpulan bilangan, sekumpulan karakter, dan sebagainya.

### B. Struktur Array

```
tipe_data nama_variable_array [ukuran_array] = {elemen};
```

### C. Operasi umum

Dalam penggunaannya kita dapat melakukan beberapa manipulasi atau operasi terhadap variable array yang akan kita buat sesuai yang kita butuhkan, diantaranya:

#### a. Membuat Array

```
tipe_data nama_variabel[ukuran];
```

ukuran adalah berapa banyak data yang dapat disimpan oleh variabel itu.

Ukuran array tidak dapat diubah setelah ditentukan.

Contoh:

```
int sekumpulan_bilangan[10];
```

Variabel di atas dapat menyimpan 10 data dengan tipe integer.

```
char sekumpulan_huruf[26];
```

Variabel di atas dapat menyimpan 26 data dengan tipe karakter.

## b. Inisialisasi Array

Nilai-nilai dalam array disebut dengan “elemen”.

```
tipe_data nama_variabel[ukuran] = {elemen 1, elemen 2, ...};
```

Contoh:

```
int bilangan_prima[7] = {2, 3, 5, 7, 11, 13, 17};
```

```
char huruf_vokal[5] = {'a', 'i', 'u', 'e', 'o'};
```

## c. Inisialisasi Tanpa Menentukan Ukuran

Inisialisasi array dapat dilakukan tanpa menuliskan ukurannya. Compiler secara otomatis menentukan ukurannya sesuai dengan berapa banyak data yang dimasukkan saat inisialisasi.

Contoh:

```
char huruf[] = {'a', 'b', 'c'};
```

Array di atas akan memiliki ukuran 3.

## d. Menggunakan Indeks

```
nama_variabel[indeks]
```

Indeks adalah bilangan penunjuk yang dapat digunakan untuk mengakses atau mengubah elemen array.

**Indeks dimulai dari 0, bukan 1.** Artinya indeks elemen pertama adalah 0, indeks elemen kedua adalah 1, dan indeks elemen terakhir adalah jumlah elemen dikurangi 1.

Contoh:

```
char huruf[26];
```

Array di atas dapat menyimpan data dengan indeks dari **0 sampai 25**.

Contoh penggunaan indeks:

```
char huruf[] = {'a', 'b', 'c'};

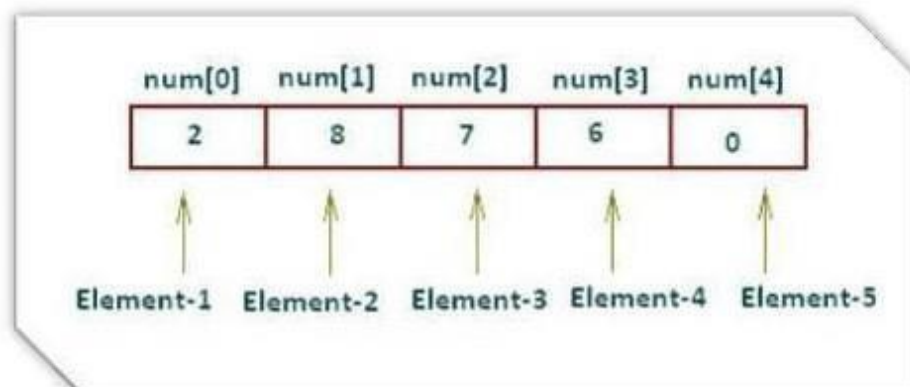
// mencetak huruf a
printf("%c", huruf[0]);

// mengganti huruf b dengan huruf d
huruf[1] = 'd';

// mengganti huruf c dengan huruf yang diinput oleh pengguna
scanf("%c", &huruf[2]);
```

## e. Mencetak Dengan Indeks Yang Melebihi Indeks Maksimal

Ketika kita mencetak dengan indeks yang melebihi indeks maksimal, maka yang akan tampil adalah bilangan acak yang merupakan lokasi memori di komputer yang biasa disebut dengan **garbage value** (nilai sampah).



## f. Mencetak Semua Elemen Array Menggunakan Perulangan

Untuk mencetak semua elemen array, indeks kita tentukan menggunakan variabel counter dari perulangan, sehingga data dari indeks 0 sampai indeks terakhir akan tercetak.

```
int bilangan_prima[7] = {2, 3, 5, 7, 11, 13, 17};

for (int i = 0; i < 7; i++)
    printf("%d ", bilangan_prima[i]);
```

Output:

```
2357111317_
```

## g. Mengisi Semua Elemen Array Menggunakan Perulangan

```
int bilangan[7];

for (int i = 0; i < 7; i++)
{
    printf("Indeks ke-%d: ", i);
    scanf("%d", &bilangan[i]);
}
```

## h. Mengisi Sebagian Elemen Array

Ketika kita mengisi sebagian elemen array, maka elemen sisanya akan bernilai 0.

Contoh:

```
int bilangan[7] = {5, 2, 3};
```

Kode di atas akan mengisi 3 elemen pertama dan elemen berikutnya akan bernilai 0.

```
{5, 2, 3, 0, 0, 0, 0}
```

Jadi kita bisa membuat semua elemennya bernilai 0 hanya dengan kode singkat berikut.

```
int bilangan[7] = {0};
```

Semua elemen pada array di atas akan bernilai 0.

```
{0, 0, 0, 0, 0, 0, 0}
```

## D. Array Multidimensi

Array multidimensi adalah array yang berisi sekumpulan array lagi. Penulisan ukuran array dimulai dari kiri,

Ukuran array anak ditulis di sebelah kanan ukuran induknya.

```
tipe_data nama_variabel[ukuran induk][ukuran anak];
```

Contoh array 2 dimensi:

```
int bilangan[3][4];
```

Array di atas berisi 3 array yang masing-masing berisi 4 elemen.

Contoh array 3 dimensi:

```
int bilangan[3][4][2];
```

Array di atas berisi 3 array yang masing-masing berisi 4 array yang masing-masing berisi 2 elemen.

Contoh inisialisasi:

```
int bilangan[3][4] = {  
    {1, 2, 3, 4},  
    {5, 6, 7, 8},  
    {9, 10, 11, 12}  
};
```

## E. Mencetak semua elemen array multidimensi menggunakan perulangan

```
int bilangan[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 4; j++)
        printf("%d ", bilangan[i][j]);
    printf("\n");
}
```

Output:

```
1234
5678
9101112
—
```

## F. String Adalah Array Karakter

String yang kita pelajari sebelumnya sebenarnya adalah array karakter yang elemen terakhirnya adalah NULL. Itulah mengapa ukurannya harus kita lebihkan minimal 1 dari jumlah karakter.

```
char nama[5] = "Umar";
```

Statement di atas sama seperti:

```
char nama[5] = {'U', 'm', 'a', 'r', NULL};
```



## G. Array String

Array string artinya array yang berisi sekumpulan array karakter. Jadi sama saja seperti array 2 dimensi.

```
char sekumpulan_nama[4][10] = {"Abu Bakar", "Umar", "Usman", "Ali"};
```

4 adalah jumlah nama,

10 adalah jumlah maksimal karakter per nama.

## H. Pertanyaan yang Sering Ditanyakan

### ☐ Apa tujuan mempelajari array?

Array berguna untuk menampung banyak data nantinya akan kita manfaatkan. Seperti data nilai satu kelas, daftar nama, daftar grade kuliah, mencari total keseluruhan nilai, dll.

### ☐ Apakah array bisa diisi dengan data yang tipenya bermacam-macam?

Tidak. Array hanya bisa diisi dengan data yang tipenya sesuai dengan tipe data variabelnya.

### ☐ Apakah kita bisa menambahkan elemen ke array?

Tidak. Kita hanya bisa mengganti nilai elemennya.

### ☐ Apakah array bisa diubah ukurannya?

Tidak. Tidak ada cara untuk mengubah ukuran array selama program berjalan.

# I. Contoh Program

Input nama anggota per kelompok ke dalam array

```
char kelompok[3][2][10];

for (int k = 0; k < 3; k++) {
    printf("Kelompok %d\n", k + 1);
    for (int a = 0; a < 2; a++) {
        printf("Anggota %d: ", a + 1);
        fgets(kelompok[k][a], 10, stdin);
    }
    printf("\n");
}
```

Hasil:

```
Kelompok 1
Anggota 1: Anggi
Anggota 2: Fadli

Kelompok 2
Anggota 1: Riyan
Anggota 2: Maulina

Kelompok 3
Anggota 1: Farhan
Anggota 2: Ali
```

# Bab 12:

## Fungsi

Program yang besar pasti memiliki algoritma yang kompleks. Algoritma kompleks itu perlu dipecah menjadi tugas-tugas sederhana yang masing-masing dikelompokkan dalam fungsi.

Fungsi juga akan mempersingkat kode yang kita tulis. Misalnya terdapat 50 proses yang menggunakan algoritma yang sama, setiap proses membutuhkan 10 baris kode, sehingga totalnya ada 500 baris. Dengan menggunakan fungsi kita hanya perlu menulis 10 baris itu 1 kali saja di dalam fungsi, dan untuk menjalankan 10 baris itu kita hanya perlu memanggil fungsi dengan 1 baris kode.

Fungsi juga akan membuat program lebih efisien dan hemat memori.

### A. Pengertian fungsi

Fungsi merupakan suatu blok program (sub program) yang digunakan untuk melakukan tugas-tugas tertentu yang letaknya terpisah dari program utamanya (main). Tugas tersebut bisa berupa perintah input, output maupun melakukan penyelesaian/perhitungan.

Sintak:

```
tipe_pengembalian nama_fungsi()
{
    statement 1;
    statement 2;
    ...
}
```

tipe\_pengembalian adalah tipe data yang akan dikembalikan (data yang didapatkan setelah fungsi itu selesai berjalan), untuk fungsi yang tidak mengembalikan data apapun, maka menggunakan void, yang artinya kosong.

Fungsi harus ada sebelum dipanggil. Compiler membaca program dari atas ke bawah, jadi fungsi harus berada di atas dari tempat pemanggilannya.

Contoh:

```
#include <stdio.h>

void baris_baru()
{
    printf("\n");
}

void main()
{
    printf("Baris pertama.");
    baris_baru();
    printf("Baris kedua.");
    baris_baru();
    printf("Baris ketiga.");
}
```

Output:

```
Baris pertama.
Baris kedua.
Baris ketiga._
```

## B. Parameter dan argumen

Parameter adalah variabel yang menerima nilai sehingga dapat diakses dari dalam fungsi yang memiliki parameter tersebut.

```
tipe_pengembalian nama_fungsi(parameter 1, parameter 2, ...)
{
    statement 1;
    statement 2;
    ...
}
```

Argumen adalah nilai yang dikirim saat memanggil fungsi.

```
nama_fungsi(argumen 1, argumen 2, ...);
```

Contoh:

```
#include <stdio.h>

void cetak_hasil_kali(int bilangan_a, int bilangan_b)
{
    printf("%d", bilangan_a * bilangan_b);
}

void main()
{
    cetak_hasil_kali(7, 3);
}
```

int bilangan\_a dan int bilangan\_b adalah parameter. Sedangkan 7 dan 3 adalah argumen.

Output:

```
21_
```

## C. Pengembalian

Fungsi dapat mengembalikan data ke tempat pemanggilannya menggunakan return.

Contoh:

```
#include <stdio.h>

int dapatkan_hasil_kali(int bilangan_a, int bilangan_b)
{
    return bilangan_a * bilangan_b;
}

void main()
{
    int hasil = dapatkan_hasil_kali(7, 3);
    printf("%d", hasil);
}
```

Atau langsung mencetaknya tanpa menggunakan variabel penampung:

```
void main()
{
    printf("%d", dapatkan_hasil_kali(7, 3));
}
```

Output:

21\_

Sebelum adanya standar C, fungsi yang tidak mengembalikan nilai disebut prosedur.

Fungsi yang mengembalikan nilai    Fungsi yang tidak mengembalikan nilai

Fungsi akan mengembalikan nilai hasil pengolahan kepada pemanggilnya

Prosedur tidak mengembalikan nilai kepada pemanggilnya

Return type-nya berupa tipe data seperti int, float, double, char, dll. Tergantung pada nilai apa yang akan kita kembalikan kepada si pemanggil Return bertipe void atau kosong (tidak ada nilai kembalian)

Terdapat statement/tulisan 'return' (tanpa tanda petik) dalam blok programnya

Tidak terdapat statement/tulisan 'return' (tanpa tanda petik) dalam blok programnya

Dalam pemanggilan fungsi dibutuhkan penampung nilai hasil kembalian dari fungsi yang dipanggilnya Dalam pemanggilan prosedur tidak dibutuhkan penampung karena tidak ada nilai yang dikembalikan

## D. Penulisan Prosedur

### 1. Prosedur Tanpa Parameter

Prosedur tanpa parameter adalah sebuah kode blok yang bisa mengeksekusi apa yang ada didalam prosedur tanpa bisa menerima lemparan dari prosedur lain. Jadi akan dieksekusi ketika dipanggil. Cara umum penulisan prosedur adalah sebagai berikut :

```
void NamaProsedur ( ) {  
    Statement (pernyataan);  
}
```

#### Penjelasan :

- void artinya kosong, maksudnya prosedur ini tidak mengembalikan nilai apa-apa (kosong).
- NamaProsedur adalah tempat untuk menamai prosedur yang kita buat.
- Tanda ' ( ) ' buka dan tutup kurung ini untuk menandakan bahwa kode tersebut adalah sebuah prosedur yang bisa dipanggil.
- Tanda '{ }' buka dan tutup kurawal ini adalah tempat untuk menandakan bagian / statement tersebut adalah milik prosedur tersebut.

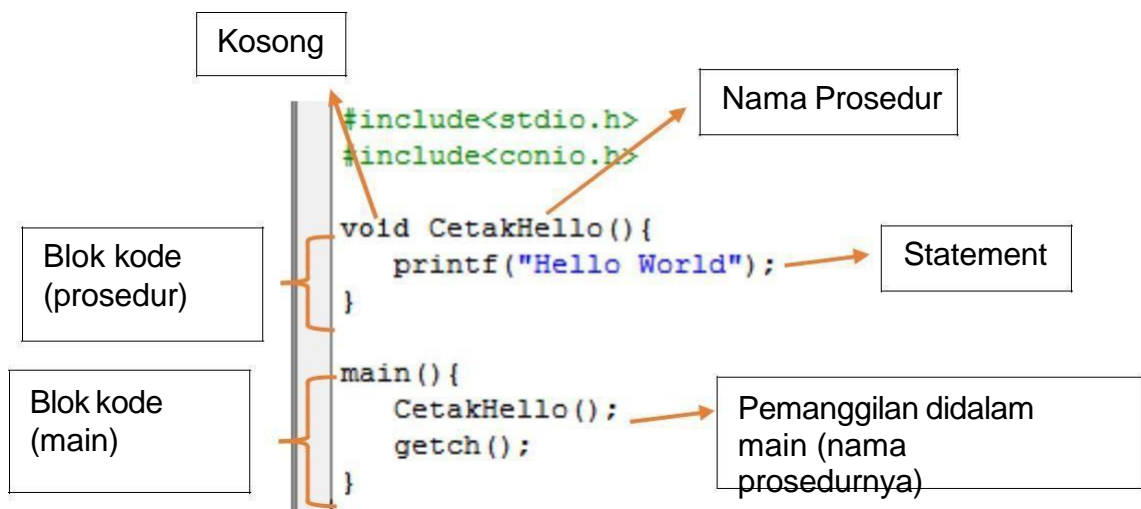
## Cara pemanggilan :

NamaProsedur();

## Penjelasan :

- Didalam main kita cukup menyebutkan NamaProsedur dan diakhiri ( ) juga semicolon (;) karena didalam main,prosedur tersebut sudah merupakan statement yang dipanggil.

Agar lebih jelas perhatikan kode dibawah ini :



**DIINGAT !** Pemanggilan juga bisa didalam sebuah prosedur lain, tidak hanya dalam main namun prosedur yang memanggil prosedur lain pun harus dipanggil didalam main()

Jadi program tidak akan menjalankan prosedur jika tidak dipanggil oleh main, namun tetap dieksekusi dan menyebabkan error jika salah satu prosedur memiliki syntax eror.



## 2. Prosedur Dengan Parameter

Prosedur dengan parameter adalah blok kode yang menerima lemparan data dari main atau fungsi lain untuk dikelola lagi dalam prosedur tersebut. Tetapi sama seperti Prosedur sebelumnya prosedur ini juga hanya menerima lemparan parameter saja tidak bisa mengembalikan nilainya. Cara umum penulisan adalah sebagai berikut :

```
void NamaProsedur (parameter) {  
    statement;  
}
```

### Penjelasan :

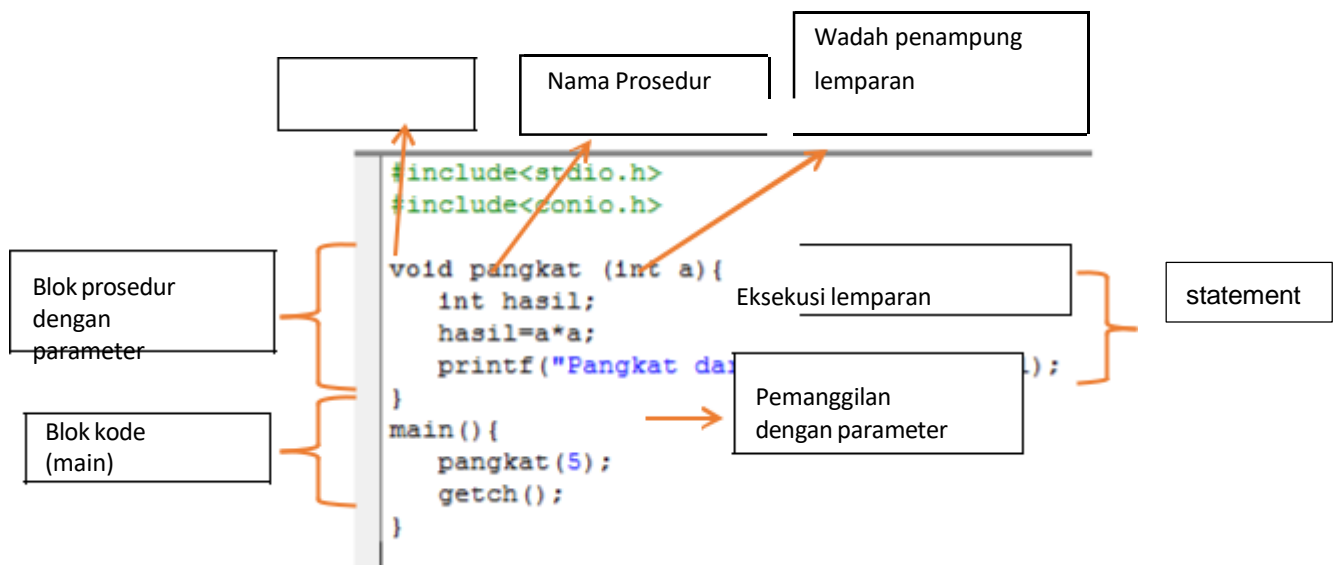
- Void artinya kosong, maksudnya prosedur ini tidak mengembalikan apa-apa (kosong).
- NamaProsedur adalah tempat untuk menamai prosedur yang kita buat.
- (parameter) adalah wadah untuk menerima lemparan dari kode yang memanggilnya.
- Tanda '{ }' buka dan tutup kurawal ini adalah tempat untuk menandakan bagian/ statement tersebut adalah milik prosedur tersebut.

### Cara pemanggilan :

NamaProsedur(lemparan);

### Penjelasan :

- Pemanggilan Prosedur dengan parameter sama seperti prosedur tanpa parameter hanya saja didalam tanda '( )' kurung , kita harus menyertakan lemparan, jika kita memanggil prosedur yang memiliki parameter tapi dalam main kita memanggil tanpa menyertakan lemparan atau menyertakan lemparan yang tidak sesuai tipe datanya maka akan muncul laporan eror seperti berikut :



### Perbedaan Parameter dengan Argumen:

- **Parameter** -> nama variabel yang ada di dalam prosedur/fungsi.  
Misal: (void tambah(int a, int b)) .
- **Argumen** -> nilai yang dilempar ke parameter prosedur/fungsi pada saat pemanggilan. Misal: (tambah(4,6); ).

## E. Penulisan Fungsi (Fungsi)

### 1. Fungsi tanpa parameter

Fungsi tanpa parameter adalah sebuah blok kode yang tidak bisa menerima lemparan karena tidak memiliki parameter namun fungsi ini bisa mengembalikan nilai karena fungsi memiliki return type. Return type adalah sebuah tipe data yang menampung nilai yang dikembalikan (di-return) oleh fungsi, tentunya return type ini harus sesuai dengan apa yang ingin kita kembalikan. Bentuk umum penulisan fungsi adalah :

```
returnType NamaFungsi() {  
    statement;  
    return;  
}
```

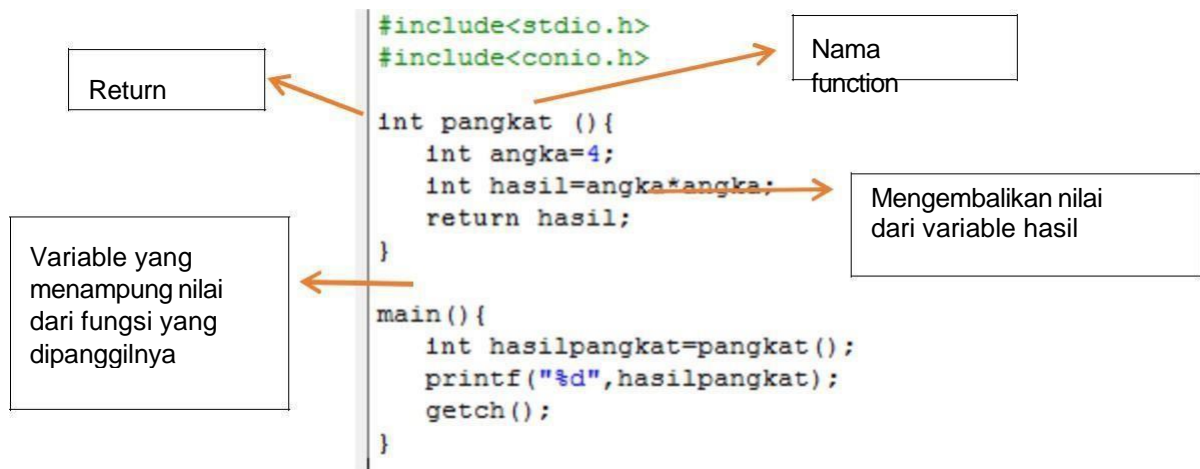
#### Penjelasan :

- ReturnType adalah sebuah type data seperti (int,float,long,double dll) disesuaikan dengan kebutuhan pengembalian nilai.
- NamaFungsi adalah tempat untuk memberi nama fungsi tersebut
- Return kata kunci untuk mengembalikan nilai.

Cara pemanggilan fungsi juga sama seperti prosedur, hanya jika kita ingin nilai yang dikembalikan oleh fungsi tersebut maka kita harus membuat tampungan juga, contoh :

```
void main() {  
    NamaVariabel = NamaFungsi();  
}
```

Untuk lebih jelasnya pahami kodingan dibawah ini !



## 2. Fungsi dengan parameter

Fungsi dengan parameter adalah blok kode yang menerima lemparan (parameter) dari siapa yang memanggилnya juga siap mengembalikan nilai tersebut. Analoginya seorang bos adalah sebagai pemanggil fungsi dan fungsi adalah anak buahnya yang siap mengerjakan dengan bahan yang dikirimkan oleh si bos (lemparan) kemudian diterima oleh anak buahnya (parameter) dan data tersebut diolah oleh anak buah (fungsi) ketika selesai data tersebut dikembalikan kepada si bos (pemanggil). Penulisan dalam kodingan bahasa C adalah :

```
ReturnType NamaFungsi(parameter) {
    Statement;
    return statement;
}
```

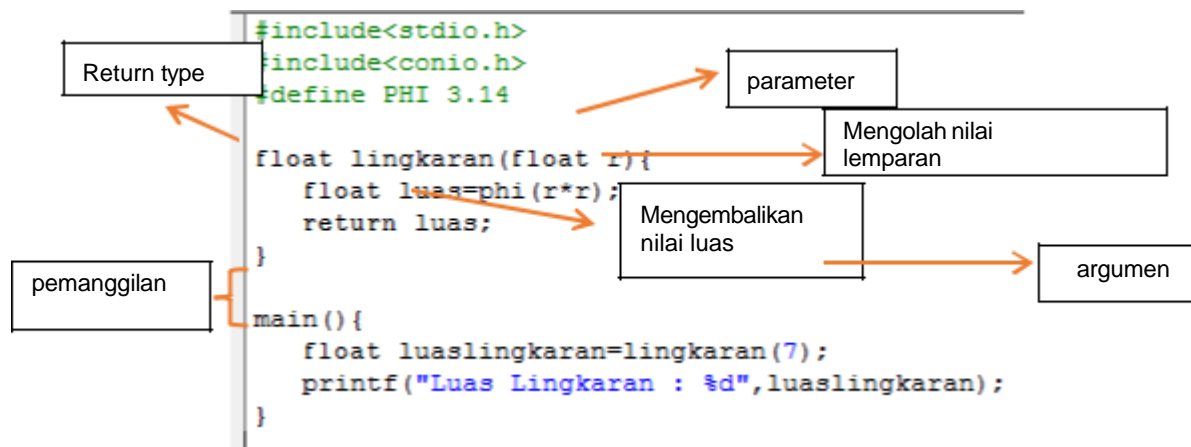
### Penjelasan :

- Sama hal dengan prosedur parameter adalah wadah penampung lemparan.

### Pemanggilan :

```
void main() {
    NamaVariabel NamaFungsi(argumen);
}
```

Perhatikan kode di bawah ini:



**Catatan:**

Dalam fungsi dan prosedur kita bebas menjabarkan apa saja, seperti menggunakan perulangan, pilihan if else disesuaikan dengan kebutuhan. Dan untuk lebih mengefisienkan program perhatikan kapan kita harus menggunakan fungsi dan kapan menggunakan prosedur.

# BAB 13 :

## Input Output file Text

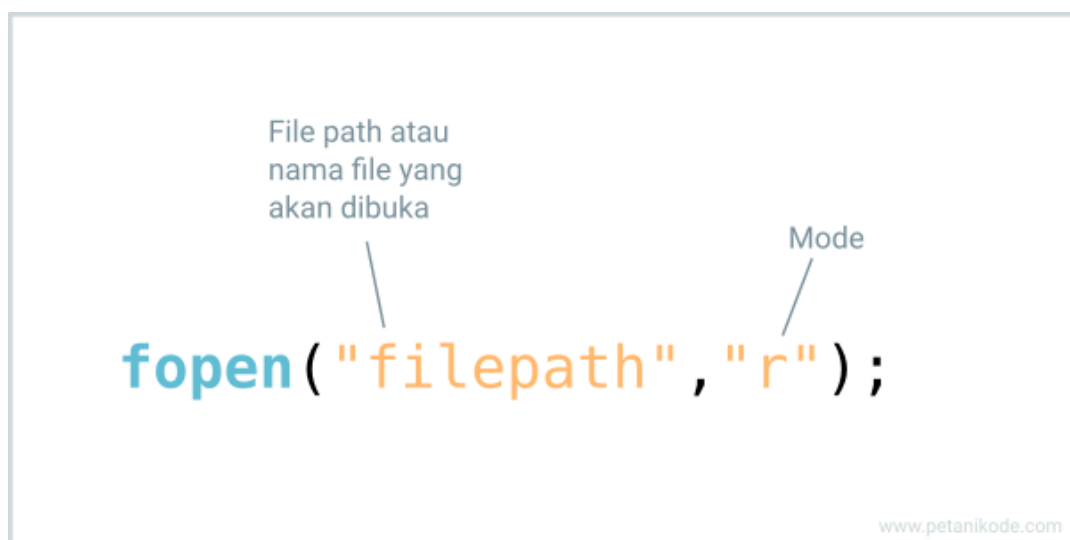
Dalam dunia pengembangan perangkat lunak, seringkali aplikasi perlu berinteraksi dengan file untuk menyimpan dan membaca data. File dapat berisi konfigurasi, log, atau data lain yang diperlukan oleh aplikasi. Dalam bahasa pemrograman C, operasi I/O file dilakukan menggunakan fungsi-fungsi yang disediakan oleh pustaka standar C, seperti `stdio.h`.

File teks adalah file `.txt` normal . Anda dapat dengan mudah membuat file teks menggunakan editor teks sederhana seperti Notepad.

Saat membuka file tersebut, Anda akan melihat semua konten di dalam file sebagai teks biasa. Anda dapat dengan mudah mengedit atau menghapus isinya.

### A. Membuka dan Menutup File

Berikut ini adalah fungsi untuk membuka atau membaca file di C



Membuka file dilakukan menggunakan `fopen()` fungsi yang ditentukan dalam `stdio.h` file header.

Fungsi fopen() akan membuka file sesuai dengan mode yang kita berikan. Mode r artinya read atau baca saja. Selain mode r ada juga mode yang lain. Mode r artinya read atau baca saja. Selain mode r ada juga mode yang lain. Berikut ini daftar lengkap modenya:

Mode	Arti	Jika File Tidak ada
r	Buka Untuk di baca	Return NULL
w	Buka untuk di tulis	Return null
a	Buka untuk di tambahkan	Return null
w+	Buka untuk ditulis dan di baca	

Fungsi fopen() akan menghasilkan sebuah pointer yang menunjuk ke alamat memori dari file yang akan dibuka, karena itulah kita membutuhkan pointer untuk mengaksesnya.

```
// membuat pointer
File *fptr;

// membuka file
fptr = fopen("namafile.txt", "r");
```

Menutup file dilakukan menggunakan fclose()fungsi.

```
fclose(fptr);
```

Fptr adalah penunjuk file yang terkait dengan file yang akan ditutup.

Membaca file teks

Untuk membaca dan menulis ke file teks, kami menggunakan fungsi fgets()

Fungsi **fgets()** akan membaca isi file yang ditunjuk oleh pointer ke struktur FILE.Di sini,

```

#include <stdio.h>

void main()
{
    char buff[255];
    FILE *fptr;

    // membuka file
    if ((fptr = fopen("puisi.txt","r")) == NULL){
        printf("Error: File tidak ada!");
        // Tutup program karena file gak ada.
        exit(1);
    }

    // baca isi file dengan gets lalu simpan ke buff
    fgets(buff, 255, fptr);
    // tampilkan isi file
    printf("%s", buff);

    // tutup file
    fclose(fptr);
}

```

Coba jalankan program nya , program tersebut akan eror , kenapa eror ?

karna program tidak menemukan file yg akan di buka

Sekarang coba buat file puisi.txt dengan isi berikut

```

ini adalah puisi buat kamu
tapi isinya ga jelas

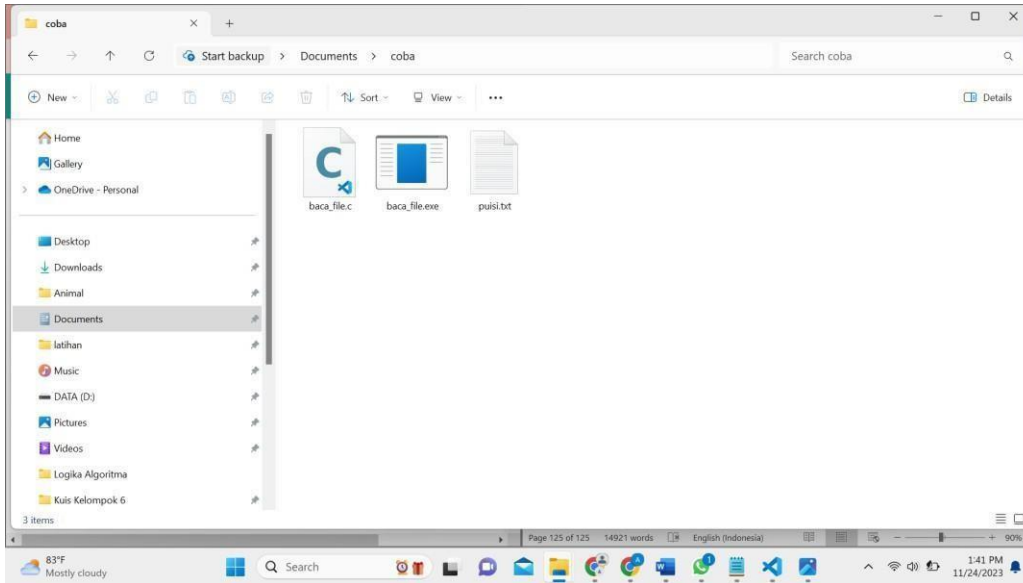
ah yang penting file ini ada isinya.

cuman buat ngetes aja sih hehe

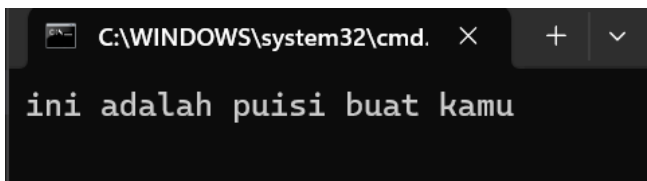
```



Simpan file di satu direktori dengan programnya



Setelah itu, coba jalankan lagi programnya  
maka hasilnya :



Isi File akan dibaca satu baris , karena kita menggunakan fungsi `fgets()`;

```
fgets(buff, 255, fptr);
```

Jika kita ubah menjadi seperti ini:

```
fgets(buff, 20, fptr);
```

Panjang karakter ini, bisa juga kita ambil dengan fungsi `sizeof()` agar mengikuti panjang karakter dari variabelnya.

```
sizeof(buff);
```

Fungsi `fgets()` hanya akan membaca satu baris saja. Angka 255 adalah panjang baris (karakter) yang akan dibaca.

Lalu bagaimana cara membaca semua baris teks yang ada di dalam file?

Gampang! Kita tinggal panggil fungsi `fgets()` berkali-kali.

Sekarang coba ubah kode programnya menjadi seperti ini:

```
#include <stdio.h>

void main()
{
    char buff[255];
    FILE *fptr;

    // membuka file
    if ((fptr = fopen("puisi.txt","r")) == NULL){
        printf("Error: File tidak ada!");
        // Tutup program karena file gak ada.
        exit(1);
    }

    // baca isi file dengan gets lalu simpan ke buff
    fgets(buff, sizeof(buff), fptr);
    printf("%s", buff);
    fgets(buff, sizeof(buff), fptr);
    printf("%s", buff);
    fgets(buff, sizeof(buff), fptr);
    printf("%s", buff);
    fgets(buff, sizeof(buff), fptr);
    printf("%s", buff);

    // tutup file
    fclose(fptr);
}
```

Outputnya :

```
ini adalah puisi buat kamu
tapi isinya ga jelas

ah yang penting file ini ada isinya.
|
```

Program akan membaca empat baris dari isi file, ini karena kita memanggil fungsi `fgets()` sebanyak empat kali.

Tapi masalahnya nanti:

“Gimana kalau isi filenya ada banyak, misal 1000 baris?”

“Kita gak mungkin kan harus mengetik perintah itu berulang-ulang”

Benar sekali...

Ini bisa kita atasi dengan perulangan. Contohnya seperti ini:

```
#include <stdio.h>

void main()
{
    char buff[255];
    FILE *fptr;

    // membuka file
    if ((fptr = fopen("puisi.txt", "r")) == NULL){
        printf("Error: File tidak ada!");
        // Tutup program karena file gak ada.
        exit(1);
    }

    // baca isi file dengan gets lalu simpan ke buff
    while(fgets(buff, sizeof(buff), fptr)){
        printf("%s", buff);
    }

    // tutup file
    fclose(fptr);
}
```

Outputnya :

```
C:\WINDOWS\system32\cmd. x 4 v
ini adalah puisi buat kamu
tapi isinya ga jelas

ah yang penting file ini ada isinya.

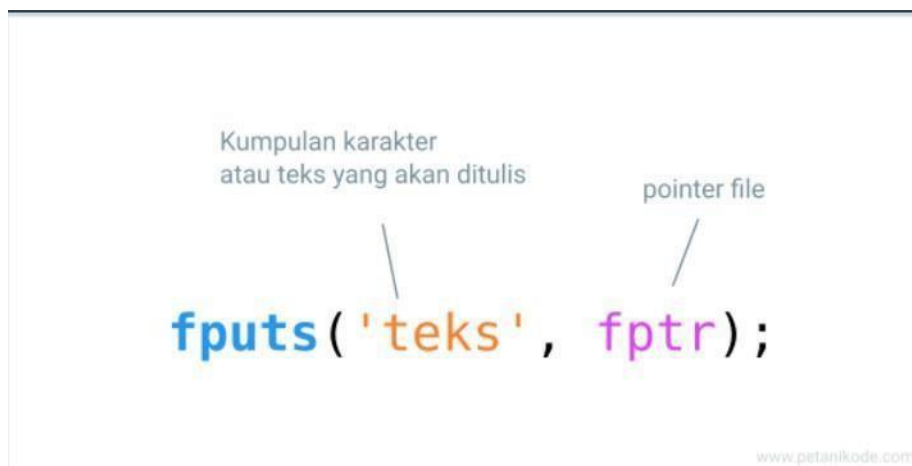
cuman buat ngetes aja sih hehe
```

Semua isi file akan dibaca. Ini karena kita memanggil fungsi `fgets()` di dalam perulangan `while`.

Perulangan `while` akan berhenti saat `fgets()` menghasilkan `null` atau sudah tidak ada lagi baris yang dibaca.

## B. Menulis File dengan Program C

Jika fungsi `fgets()` digunakan untuk membaca file, nah untuk menulisnya kita butuh fungsi `fputs()`.



Fungsi `fputs()` akan menulis teks ke dalam file yang sedang dibuka. Mari kita coba dalam program.

Buatlah program baru dengan nama `tulis_file.c`, kemudian isi dengan kode berikut:

```
#include <stdio.h>

void main()
{
    char buff[255];
    char text[255];
    FILE *fptr;

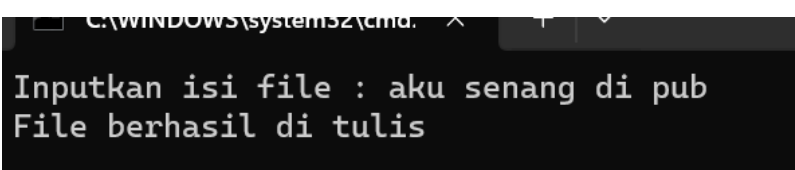
    // membuka file
    fptr = fopen("puisi.txt","w");

    // mengambil input dari user
    printf("Inputkan isi file: ");
    fgets(text, sizeof(text), stdin);

    // menulis ke text ke file
    fputs(text, fptr);

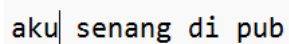
    printf("File berhasil ditulis\n");

    // tutup file
    fclose(fptr);
}
```



```
C:\WINDOWS\system32\cmd.exe
Inputkan isi file : aku senang di pub
File berhasil di tulis
```

Maka sekarang isi file `puisi.txt` akan seperti ini :



```
aku| senang di pub
```

Jika kamu membuka file untuk ditulis, maka mode yang digunakan adalah `w` atau `w+`,  
a. Silahkan cek kembali tabel mode di atas.

Kalau salah pakai mode gimana? Ya filenya tidak akan bisa ditulis. Lalu, gimana cara menulis dan sekaligus menampilkan isi file?

Caranya, kita harus membuka ulang filenya dengan mode yang berbeda-beda.

```
#include <stdio.h>

void main()
{
    char buff[255];
    char text[255];
    FILE *fptr;

    // membuka file untuk ditulis
    fptr = fopen("puisi.txt","w");

    // mengambil input dari user
    printf("Inputkan isi file: ");
    fgets(text, sizeof(text), stdin);

    // menulis ke text ke file
    fputs(text, fptr);

    printf("File berhasil ditulis\n");

    // tutup file setelah ditulis
    fclose(fptr);

    // buka kembali file untuk dibaca
    fptr = fopen("puisi.txt","r");

    // baca isi file dengan gets lalu simpan ke buff
    while(fgets(buff, sizeof(buff), fptr)){
        printf("Isi filenya sekarang: %s", buff);
    }

    // tutup file
    fclose(fptr);
}
```

kita hanya baru menulis dalam satu baris saja. Lalu gimana kalau mau menulis banyak baris?

Gampang, kita hanya tinggal memanggil fungsi fputs() berkali-kali.

```

#include <stdio.h>

void main()
{
    char buff[255];
    char text[255];
    FILE *fptr;

    // membuka file untuk ditulis
    fptr = fopen("puisi.txt","w");

    for(int i = 0; i < 5; i++){

        // mengambil input dari user
        printf("Isi baris ke-%d: ", i);
        fgets(text, sizeof(text), stdin);

        // menulis ke text ke file
        fputs(text, fptr);
    }

    printf("File berhasil ditulis\n");

    // tutup file setelah ditulis
    fclose(fptr);

    // buka kembali file untuk dibaca
    fptr = fopen("puisi.txt","r");

    // baca isi file dengan gets lalu simpan ke buff
    while(fgets(buff, sizeof(buff), fptr)){
        printf("Isi filenya sekarang: %s", buff);
    }

    // tutup file
    fclose(fptr);
}

```

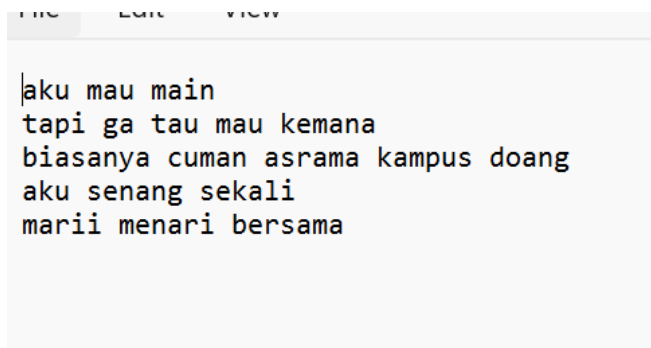
Maka hasilnya :

```

isi baris ke-0 : aku mau main
isi baris ke-1 : tapi ga tau mau kemana
isi baris ke-2 : biasanya cuman asrama kampus doang
isi baris ke-3 : aku senang sekali
isi baris ke-4 : marii menari bersama
file berhasil di tulis
isi filenya sekarang : aku mau main
isi filenya sekarang : tapi ga tau mau kemana
isi filenya sekarang : biasanya cuman asrama kampus doang
isi filenya sekarang : aku senang sekali
isi filenya sekarang : marii menari bersama

```

Maka sekarang isi file puisi.txt akan seperti ini :



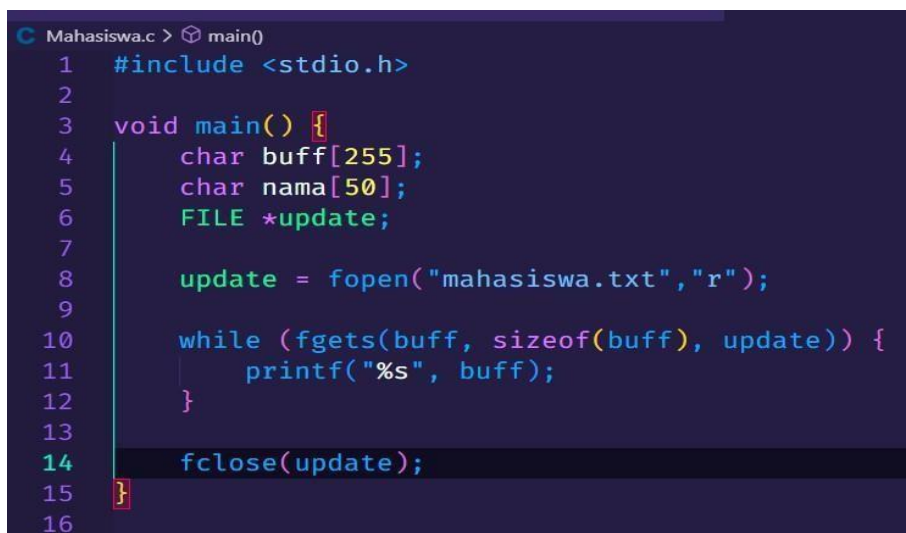
```
aku mau main
tapi ga tau mau kemana
biasanya cuman asrama kampus doang
aku senang sekali
marii menari bersama
```

Sebenarnya masih ada lagi fungsi lain untuk membaca dan menulis file seperti `scanf()`, `fwrite()`, `fread()`, dll. Namun, menurut fungsi-fungsi `fopen()`, `fget()`, dan `fputs()` lebih mudah dipakai.

### C. UPDATE FILE DALAM BAHASA C

Untuk Mengupdate Data Di Bahasa C Kita Perlu Membuka Kembali File Yang Sudah Kita Buat Dan Di Simpan. Sekarang Kita Bikin File Baru ,Contoh nya Saya Mempunyai File **“Mahasiswa.c”** Dan File **“Mahasiswa.txt”** Saya Ingin Mengedit Data **Mahasiswa.txt** Yang sudah Saya Berikan Sebelum nya. Berikut Langkah-langkah nya :

1. Buat Terlebih dulu File **Mahasiswa.c** Dan Isi Dengan Kode Berikut



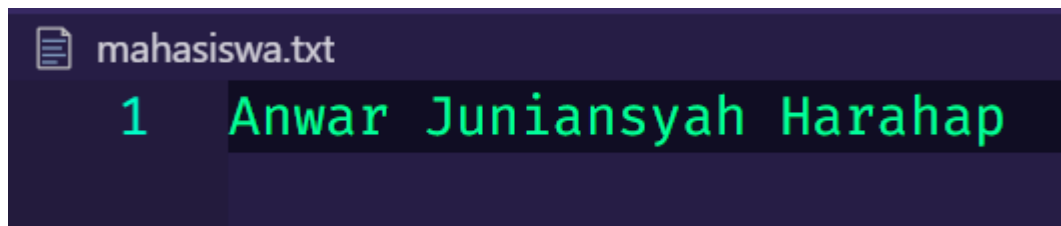
```
C Mahasiswa.c > main()
1  #include <stdio.h>
2
3  void main() {
4      char buff[255];
5      char nama[50];
6      FILE *update;
7
8      update = fopen("mahasiswa.txt","r");
9
10     while (fgets(buff, sizeof(buff), update)) {
11         printf("%s", buff);
12     }
13
14     fclose(update);
15 }
16
```



**Keterangan :**

- **FILE** adalah tipe data yang di gunakan untuk merepresentasikan(menggambarkan) aliran file. Tipe data ini memungkinkan program yang kita buat untuk berinteraksi dengan file **mahasiswa.txt**. contohnya : membaca file dan sebagainya.
- **\*update** berguna sebagai alamat memori yang merujuk ke **FILE** untuk mengakses **mahasiswa.txt**.

2. Buat File mahasiswa.txt Dan Isi Dengan Data Seperti Berikut :



```
mahasiswa.txt  
1 Anwar Juniansyah Harahap
```

Jika kita menjalankan program ini maka output nya sebagai berikut :



```
C:\WINDOWS\system32\cmd.exe  
Anwar Juniansyah Harahap_
```

3. Kemudian Ubah Kode pada Langkah No.1 Tadi Dengan Kode Berikut :

```
C:\Mahasiswa.c > ...
1  #include <stdio.h>
2
3  void main() {
4      char buff[255];
5      char nama[50];
6      FILE *update;
7
8      update = fopen("mahasiswa.txt","r");
9
10     while (fgets(buff, sizeof(buff), update)) {
11         printf("%s", buff);
12     }
13
14     fclose(update);
15     printf("\nMasukan Nama Baru Yang Akan Di Update : ");
16     fgets(nama, sizeof(nama), stdin);
17
18     fclose(update);
19     update = fopen("mahasiswa.txt", "w");
20     fprintf(update, "%s", nama);
21
22     fclose(update);
23     printf("Data berhasil diupdate.\n");
24
25     update = fopen("mahasiswa.txt","r");
26
27     while (fgets(buff, sizeof(buff), update)) {
28         printf("%s", buff);
29     }
30     fclose(update);
31 }
```

4. Kemudian Jalankan Program Nya Dan Isi Dengan Nama Yang Baru!

C:\WINDOWS\system32\cmd.exe

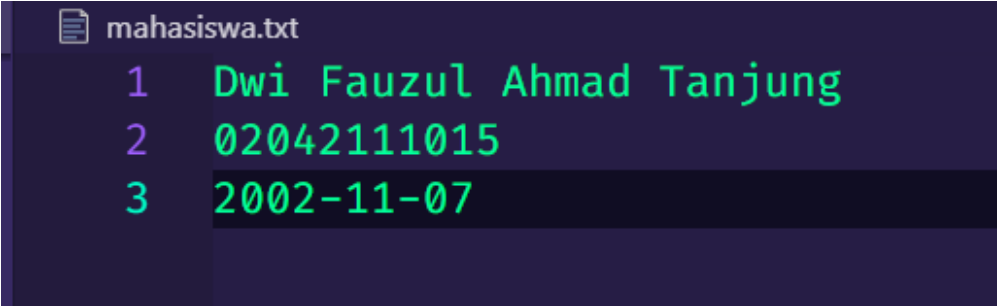
```
Anwar Juniansyah Harahap
Masukan Nama Baru Yang Akan Di Update : Anggi Nouval Tanjung
Data berhasil diupdate.
Anggi Nouval Tanjung
```

5. Data pada **mahasiswa.txt** Berhasil Di Update. Jika Kita Lihat Kembali Ke File **mahasiswa.txt** Maka Isinya Sudah Berubah Seperti Ini:



```
1 Anggi Nouval Tanjung
2
```

6. Isi ulang dulu file **mahasiswa.txt** dengan data berikut :



```
1 Dwi Fauzul Ahmad Tanjung
2 02042111015
3 2002-11-07
```

7. Lalu Isi Kode Berikut Ke **File Mahasiswa.c**

```
C Mahasiswa.c > main()
1  #include <stdio.h>
2
3  void main() {
4      FILE *update;
5
6      update = fopen("mahasiswa.txt", "r");
7
8      char Data[3][255];
9
10     fclose(update);
11     printf("Masukkan Nama Baru: ");
12     fgets(Data[0], sizeof(Data[0]), stdin);
13
14     printf("Masukkan NIM Baru: ");
15     fgets(Data[1], sizeof(Data[1]), stdin);
16
17     printf("Masukkan Tanggal Baru: ");
18     fgets(Data[2], sizeof(Data[2]), stdin);
19     update = fopen("mahasiswa.txt", "w");
20     for (int i = 0; i < 3; i++) {
21         fprintf(update, "%s", Data[i]);
22     }
23     fclose(update);
24     printf("File berhasil diubah.\n");
25 }
```

8. Jalankan Program,Dan Isi Data Baru.

C:\WINDOWS\system32\cmd.exe

```
Masukkan Nama Baru: Ragil Sadewa Pasaribu
Masukkan NIM Baru: 02032311034
Masukkan Tanggal Baru: 2004-03-09
File berhasil diubah.
```

Data **mahasiswa.txt** Sudah Di Update yeayyy 🥳🥳

### KESIMPULAN :

Untuk Mengupdate sebuah data kita harus membuka kembali file yang ingin kita ubah dengan fungsi **fopen** Yang ada pada **FILE** Kemudian Kita Perlu Membaca Data Yang Ada Pada File Yang Kita Tuju Menggunakan Mode “r” Yang Tersedia di **FILE**, Lalu Kita Tulis Ulang Data Yang Mau Kita Update Menggunakan Mode “w” Yang Tersedia di **FILE**

## D. DELETE FILE DALAM BAHASA C

Untuk Delete Sendiri Kita Perlu Memanipulasi Data Yang Sudah Kita Inputkan. Memang Agak Rumit Tapi Tenang Saja Anda Akan Semakin Pusing. Bercyanda 🤖. Pada Bagian Ini Kita Akan Belajar Memanipulasi Data Agar Bisa Menghapus nya Sesuai Dengan Keinginan Kita Berikut Langkah-Langkahnya :

1. Tuliskan Kode Berikut Ini di File Mahasiswa.c

```
C Mahasiswa.c > ...
2  #include <stdlib.h>
3  int main() {
4      FILE *update;
5
6      update = fopen("mahasiswa.txt", "w");
7
8      if (update == NULL) {
9          printf("Gagal membuka file untuk penulisan!\n");
10         return 1;
11     }
12     printf("Masukkan Nama: ");
13     char nama[255];
14     fgets(nama, sizeof(nama), stdin);
15     fprintf(update, "%s", nama);
16     printf("Masukkan NIM: ");
17     char nim[255];
18     fgets(nim, sizeof(nim), stdin);
19     fprintf(update, "%s", nim);
20     printf("Masukkan Tanggal: ");
21     char tanggal[255];
22     fgets(tanggal, sizeof(tanggal), stdin);
23     fprintf(update, "%s", tanggal);
24     fclose(update);
25     printf("Data berhasil dimasukkan ke dalam file.\n");
26     update = fopen("mahasiswa.txt", "r");
27     if (update == NULL) {
28         printf("Gagal membuka file untuk membaca!\n");
29         return 1;
30     }
31     char Data[3][255];
32     for (int i = 0; i < 3; i++) {
33         if (fgets(Data[i], sizeof(Data[i]), update) == NULL) {
34             printf("Error membaca file!\n");
35             fclose(update);
36             return 1;
37         }
38     }
39     fclose(update);
40     printf("Data yang baru dimasukkan:\n");
41     for (int i = 0; i < 3; i++) {
42         printf("%s", Data[i]);
43     }
44     return 0;
45 }
46
```

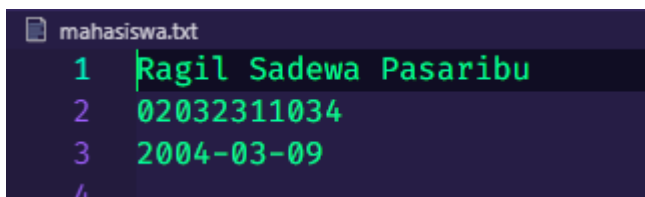
2. Kemudian Pada **File mahasiswa.txt** Tidak Usah Isi Data Apa pun

3. Jalan Kan Program **Mahasiswa.c**

C:\WINDOWS\system32\cmd.exe

```
Masukkan Nama: Ragil Sadewa Pasaribu
Masukkan NIM: 02032311034
Masukkan Tanggal: 2004-03-09
Data berhasil dimasukkan ke dalam file.
Data yang baru dimasukkan:
Ragil Sadewa Pasaribu
02032311034
2004-03-09
```

Kemudian Apabila Kita Lihat Kembali **mahasiswa.txt** Yang Awalnya Kosong Menjadi Ada Data nya Berupa :



```
mahasiswa.txt
1 Ragil Sadewa Pasaribu
2 02032311034
3 2004-03-09
4
```

4. Kemudian Tambahkan Kode Berikut Di File **Mahasiswa.c**

```
char pilih;
printf("\nApakah Anda ingin menghapus data tersebut? (y/n): ");
scanf(" %c", &pilih);

if (pilih == 'y' || pilih == 'Y') {
    update = fopen("mahasiswa.txt", "w");

    if (update == NULL) {
        printf("Gagal membuka file untuk penulisan!\n");
        return 1;
    }
}
```



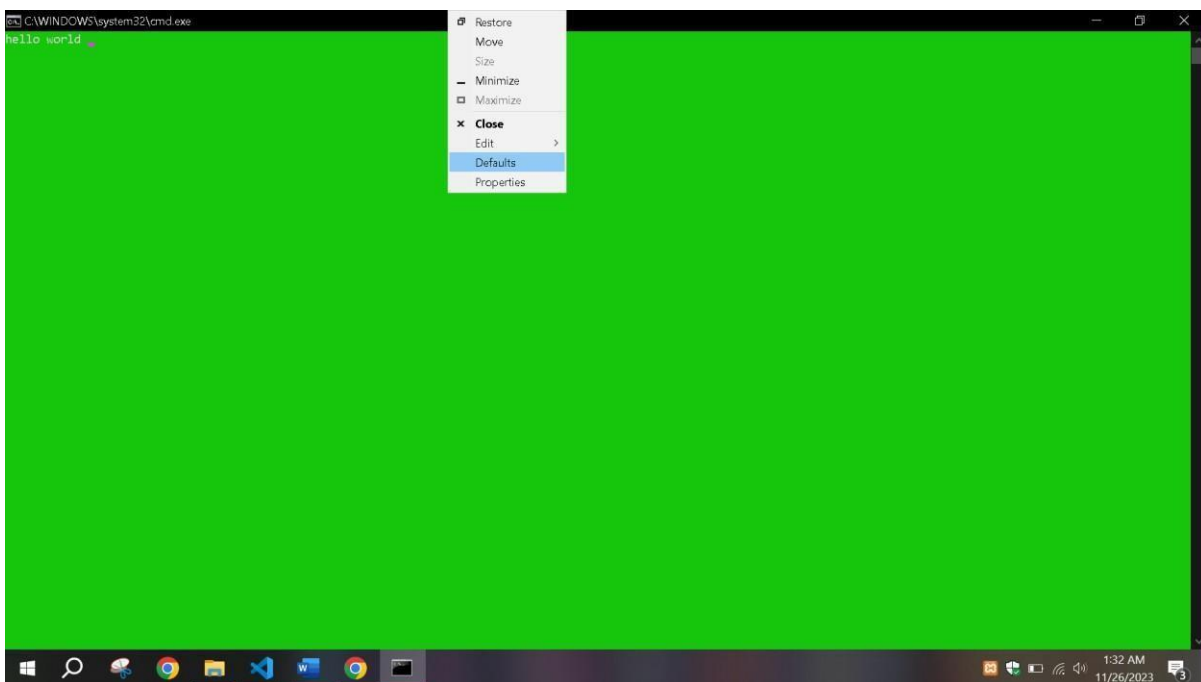
```
}  
  
fclose(update);  
  
printf("Data berhasil dihapus.\n");  
} else {  
    printf("Data tidak dihapus.\n");  
}
```

5. Data di **mahasiswa.txt** Sudah Di hapus Ketika Kita Inputkan 'y' dan tidak di hapus ketika kita inputkan 'n'.

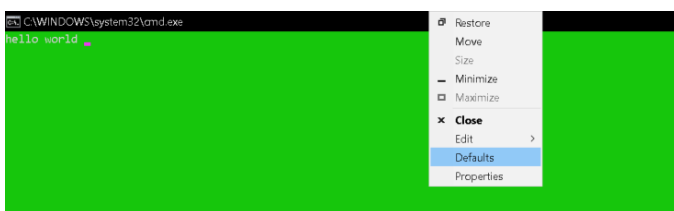
# BAB 14 :

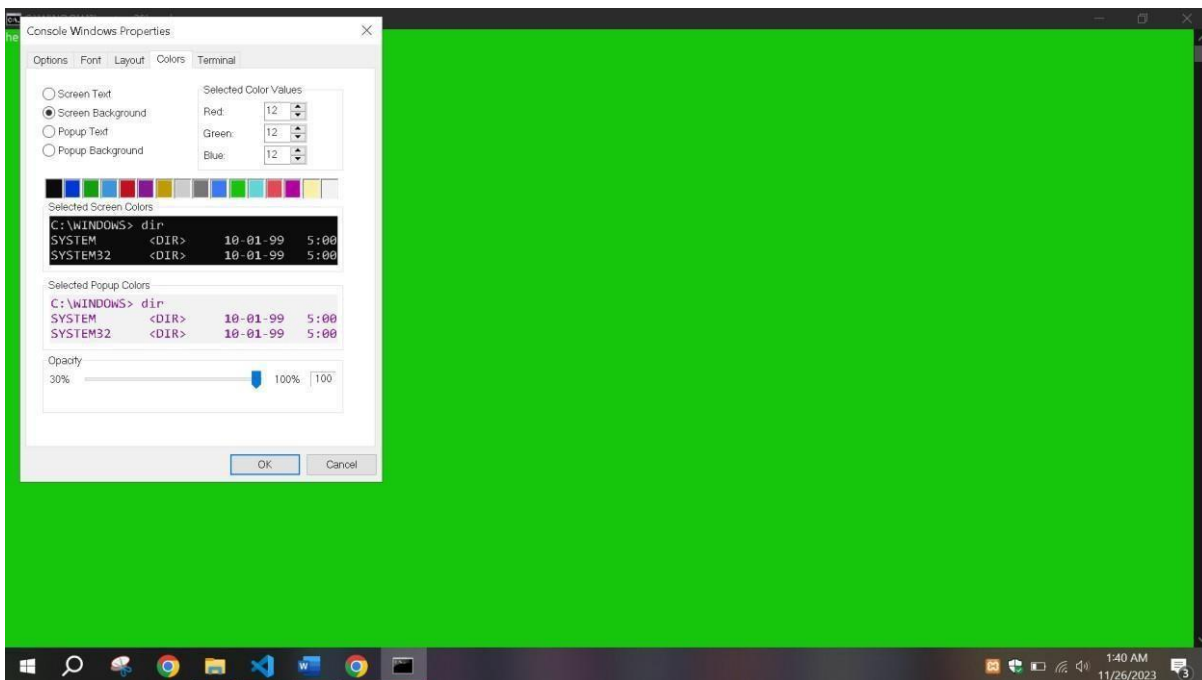
## Manipulasi Tampilan

Dimateri ini kita belajar tentang pembuatan front end(tampilan) di dalam sebuah program Bahasa c yang pertama kita harus mengerti default color di dalam cmd kalian, cmd bisa di manipulasi dengan kalian mengganti default terhadap cmd kalian perhatikan cara melihat di default cmd kalian pada gambar berikut ini



Klik kanan window kalian pada bagian yang di tunjuk oleh panah tersebut  
Lalu pilih defaults





Lalu pilih colors dan akan terlihat warna default pada cmd kalian itu dimulai Dari warna hitam yang berada pada index 0 dan sampai warna putih pada index 15,

## Cara memanggil color ke dalam kodingan

“(Akang / teteh) gimana sih cara kita bisa memanggil nya di dalam kodigan kita warna tersebut...?”

Oke cara memanggil nya kalian harus menulis kodingan seperti berikut..

```
#include<windows.h>
#include<stdio.h>
int text = 7, background = 0;
void textcolor(int new_color) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), new_color + background * 16);
    text = new_color;
}
void backgroundcolor(int new_color) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), text + new_color * 16);
    background = new_color;
}
```

Seperti biasa kita membutuhkan yang namanya library kodingan warna itu berada pada **#include<windows.h>** jadi kalian harus menambahkan library tersebut. Dimateri ini kalian harus mengerti dulu yang namanya fuction dengan baik

Kita membuat void yang berparameter bernama textcolor dan backgroundcolor yang didalam nya ada sintak **seperti gambar di atas!!**

## Cara memanggil color ke dalam kodingan

“(Akang / teteh) gimana cara nya agar kita bisa menggunakan warna terhadap tampilan kita agar lebih keren dan kece asekk!! Wkwkw”

Cara nya sangat sangat lah mudah kita **hanya perlu memanggil fuction** pada kodingan di atas contoh nya jika kita ingin mengganti warna tulisan(text) pada output, **kita hanya perlu memanggil nama function textcolor(index warna),**

Dan **jika kita ingin mengedit latar atau baground kita hanya perlu memanggil function backgroundcolor(index warna)**

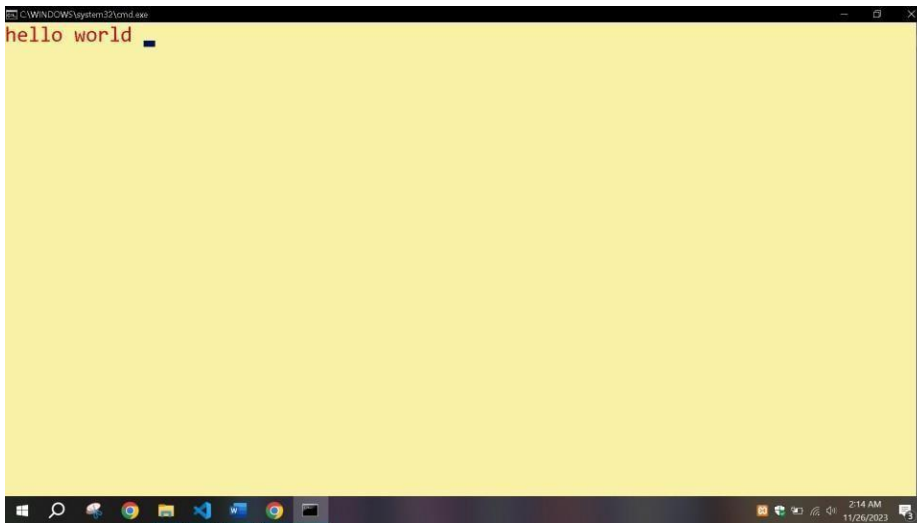
dan agar kita dapat menentukan warna yang kita ingin kan kita perlu mengetahui warna apa yang kita panggil dengan cara kita kita mengetahui index warna yang sudah di bahas di atas

“(Akang / teteh) coba contohin saya mau textnya berwarna merah dan baground nya berwarna kuning ”.

Oke baiklah kamu hanya perlu menulis kodingan ini !!

```
void main(){
    textcolor(4);
    backgroundcolor(14);
    printf("hello world ");
}
```

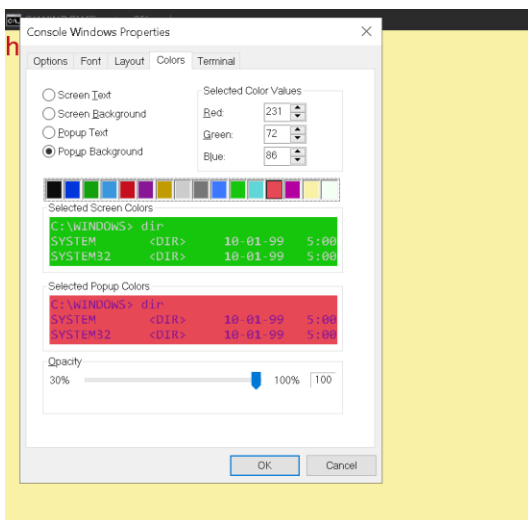
Output :



## Cara mengganti warna default

“(Akang / teteh) saya tidak suka dengan warna default yang itu itu aja saya ingin mengganti warna default nya menjadi warna pilihan saya”

Oke cara mengganti warna default kalian harus mengerti **RGB** jika kalian sudah mengerti cara menggantinya seperti berikut :



Kalian hanya perlu menaikkan RGB nya atau pun menurunkan RGB sampai kalian mendapat kan warna yang sesuai

*Catatan: jika warna sudah di ubah maka warna nya bakalan terus begitu sampai kalian menggantinya lagi*