



Dasar C

Modul Pembelajaran



10 Oktober 2022

Table of Contents

Persyaratan Sistem.....	ii
Bab 01. Dasar C.....	1
A. Pengantar	1
B. Pengenalan IDE (<i>Integrated Development Environment</i>).....	3
C. Tipe Data dan Variabel.....	4
D. Sintaks <i>input/output</i>	5
E. Operator	11
Latihan Soal Input, Output, Tipe Data, Variabel, dan Operator	16
F. Seleksi (Selection).....	17
Latihan Soal Seleksi	21
G. Perulangan (Repetition).....	22
Latihan Soal Repetisi	25
Bab 02. Array dan Pointer.....	26
A. Array Satu Dimensi.....	26
B. Array Banyak Dimensi	27
C. Pointer	28
Latihan Soal Array	29

Persyaratan Sistem

Perangkat Lunak

- Dev-C++ 5.11

Bab 01. Dasar C

Materi

Dasar C:

- Pengantar
 - Pengenalan IDE
 - Tipe data dan variabel
 - Sintaks *Input/Output*
 - Operator
 - Repetisi
 - Seleksi
-

A. Pengantar

Secara etimologis, **Algoritma** berasal dari kata *Al Khwarizmi* / *algorism* yang dalam bahasa Arab berarti proses menghitung menggunakan angka. Namun dalam pengertian lain, **algoritma** adalah kumpulan instruksi yang mendefinisikan suatu urutan dalam penyelesaian suatu masalah yang disusun secara sistematis.

Pada bidang pemrograman juga terdapat algoritma yang dipakai dan sering sekali sebelum algoritma tersebut dituangkan ke bentuk kode, akan dibuat dalam bentuk **pseudocode** terlebih dahulu. **Pseudocode** adalah serangkaian kode-kode yang dapat kita mengerti yang nantinya akan diolah dan diubah ke dalam suatu bahasa pemrograman. Selain itu, **pseudocode** dapat juga diartikan sebagai cara untuk menuliskan sebuah algoritma secara *high-level* (*level* tingkat tinggi). Untuk penulisan, **pseudocode** biasanya dituliskan dengan kombinasi bahasa manusia dan notasi matematika dan biasanya sebuah **pseudocode** tidak terlalu detail jika dibandingkan dengan kode program. Isu-isu detail dalam kode program yang sifatnya teknis tidak dibahas di dalam **pseudocode**.

Contoh kasus: kita ingin membuat program yang dapat meminta *input* nama dari *user*. Setelah itu program akan *print* nama tersebut dengan format yang sudah disediakan. Maka untuk **pseudocode**-nya akan menjadi seperti ini:


1. Read Name
2. Print Name

Untuk bentuk bahasa C-nya akan menjadi seperti ini:

```
char nama[50]; // Deklarasi Variabel
scanf("%s", nama); getchar(); // Input Nama
printf("Perkenalkan nama saya %s\n", nama); // Print Nama dengan format
```

Gambar 1. Pseudocode menjadi bahasa C

B. Pengenalan IDE (*Integrated Development Environment*)

IDE (*Integrated Development Environment*) merupakan suatu perangkat lunak yang menyediakan fasilitas bagi pengguna untuk membuat atau mengembangkan suatu program. Biasanya IDE terdiri dari *source code editor*, *build automaton tools*, dan *debugger*. Untuk materi kali ini kita menggunakan Dev-C++ 5.11 yang dapat di unduh di <https://bit.ly/slc-devcpp> .

```
int main() {  
    return 0;  
}
```

Gambar 2. Struktur utama bahasa pemrograman C

Ketika kita menjalankan program dalam bahasa pemrograman C, program akan mencari fungsi utama yang akan dijalankan, yaitu fungsi **int main()**. Dalam potongan kodingan di atas terdapat **return 0**, bagian kode ini digunakan untuk memberikan indikasi bahwa program yang kita buat berjalan tanpa ada kesalahan ketika nilai yang dikembalikan saat program berakhir adalah nilai 0.

Catatan: Untuk *compile* dan *run* program bisa menekan F11 di IDE DEV C++.

C. Tipe Data dan Variabel

Pada bahasa pemrograman C, terdapat beberapa tipe data untuk meng-kategorikan data-data yang ada kepada setiap tipe yang telah disediakan. Tipe-tipe data tersebut ada:

No	Tipe Data	Ukuran Memori	Jangkauan	Format Specifier	Keterangan
1.	char	1 byte	-128 s/d 127	%c	Karakter
2.	int	2 bytes	-32768 s/d 32767	%d	Integer/bilangan bulat
3.	float	4 bytes	-3.4E-38 s/d 3.4E+38	%f	Float/bilangan desimal
4.	double	8 bytes	-1.7E-308 s/d 1.7E+308	%lf	Bilangan decimal presisi ganda
5.	array of char	1 byte * char count	-128 s/d 127 for each character	%s	String/kumpulan karakter

Untuk menyimpan suatu nilai atau data, kita memerlukan sebuah penampung yang biasa disebut dengan variabel. Berikut format untuk mendeklarasikan suatu variabel:

Format: (Tipe_data) (Nama_variabel);

```
char nama[50];  
int umur;
```

Gambar 3. Contoh Deklarasi variabel

Pada potongan koding di atas, variabel di depan 'nama' memiliki [50], hal ini berarti untuk variabel tersebut dapat menampung 50 data untuk tipe data karakter (50 karakter) dan hal ini biasa di sebut dengan string/array of char. Untuk memberikan nilai awal pada suatu variabel kita dapat menginisialisasinya dengan cara:

```
char nama[50] = "Andrew";  
int umur = 21;
```

Gambar 4. Inisialisasi variable

D. Sintaks *input/output*

Dalam bahasa pemrograman C, operasi *input output* dapat ditemukan dalam *header (stdio.h)*. *stdio* merupakan singkatan dari *standard input output*. Jadi untuk menggunakan *standard input output* yang terdapat di ***stdio.h***, kita perlu meng-*import*-nya terlebih dahulu.

```
#include<stdio.h>
int main() {
    return 0;
}
```

Gambar 5. Import header standard input output

Untuk *import* sebuah *header* kita dapat menggunakan simbol '#' yang disebut dengan *preprocessor* diikuti dengan *keyword include*. Setelah *#include* kita dapat *import header* yang kita inginkan.

a. Input

Dalam bahasa pemrograman C, *user* atau pengguna program dapat memberikan masukan/*input* melalui perangkat *keyboard* untuk dapat berinteraksi dengan program. Berikut beberapa cara untuk membuat *input*:

1. *scanf*

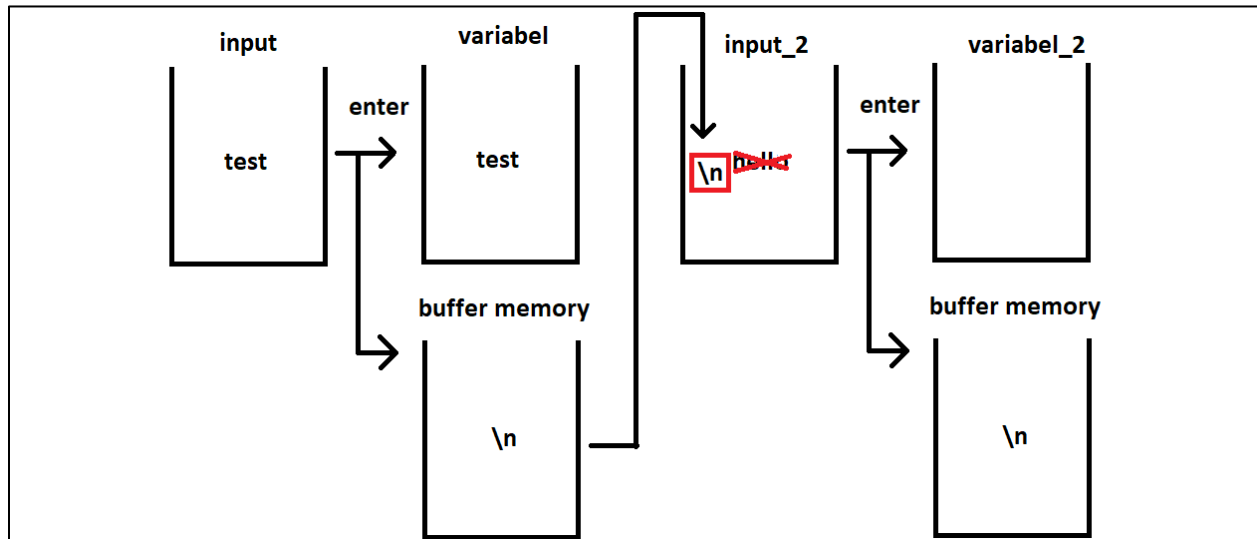
scanf digunakan untuk meminta *input* data dengan format (tipe data) yang sudah ditentukan.

Berikut merupakan daftar dari *format specifier* yang ada:

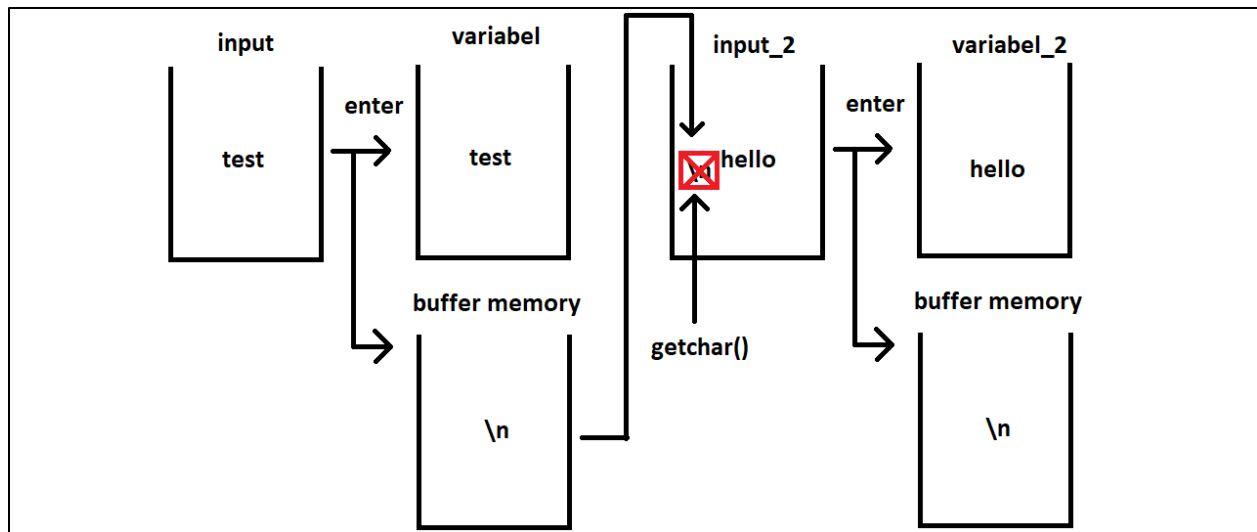
No	Format	Keterangan
1.	%x	Hexadecimal Integer
2.	%f	Float/bilangan desimal
3.	%lf	Double/bilangan desimal presisi ganda
4.	%e	Float tipe exponent dengan menggunakan e
5.	%c	Karakter
6.	%s	String/kumpulan karakter
7.	%d	Integer/bilangan bulat
8.	%ld	Long Integer/bilangan bulat dengan jangkauan lebih besar
9.	%u	Unsigned Integer/bilangan bulat positif

Catatan: Untuk pemakaian sintaks *scanf*, apabila data yang di-*input* adalah *array of character/string*, maka tanda "&" dapat dihilangkan, selain itu harus pakai. Tanda ampersand (&) diperlukan karena program perlu mengetahui alamat dari variabel yang dituju, tempat di mana menyimpan nilai yang di-*input*. Namun untuk *array of char* tidak perlu karena sebuah *array* sudah menjadi sebuah *pointer* dari alamat variabel tersebut, sehingga secara tidak langsung *array of char* sudah menunjukan tempat untuk menyimpan nilai.

Catatan: Setelah melakukan *scanf*, perlu diberi *getchar()* untuk menghilangkan "\n" yang menyangkut di *buffer memory*. *Buffer memory* merupakan tempat penyimpanan sementara untuk menyimpan nilai untuk *input* maupun *output*. Ketika *user* memberikan *input* dan menekan enter untuk mengakhiri *input*, *input*-an yang dihasilkan adalah nilai *input* yang dimasukkan oleh *user*, ditambah dengan **escape sequence** "\n" yang dihasilkan ketika *user* menekan *enter*. Nilai selain "\n" akan masuk ke variabel yang dituju, dan "\n" akan masuk ke *buffer memory*. Nilai yang menyangkut di *buffer memory* akan mengganggu proses *input* atau *output* yang selanjutnya. Karena ketika proses *input/output* selanjutnya dilakukan, *buffer memory* akan dipanggil terlebih dahulu untuk mengisi *input/output* selanjutnya jika *buffer memory* tidak kosong. Maka dari itu sebelum melakukan proses *input/output* yang selanjutnya, kita perlu mengosongkan *buffer memory* terlebih dahulu.



Gambar 6. Ilustrasi buffer memory



Gambar 7. Ilustrasi membersihkan buffer memori menggunakan getchar()

```
#include<stdio.h>

int main() {
    char nama[50];
    int umur;
    float ipk;

    printf("Masukkan nama: ");
    scanf("%s", nama);
    printf("Masukkan umur: ");
    scanf("%d", &umur);
    printf("Masukkan ipk: ");
    scanf("%f", &ipk);

    return 0;
}
```

Annotations in the image:

- `char nama[50];` → Deklarasi Variabel
- `printf("Masukkan nama: ");` → Input Prompt
- `scanf("%d", &umur);` → Input ke dalam variabel
- `scanf("%f", &ipk);` → Membersihkan '\n' dari buffer memory

Gambar 8. Input menggunakan scanf

```
Masukkan nama: Andrew
Masukkan umur: 21
Masukkan ipk: 3.90
```

Gambar 9. Contoh input

2. gets

gets digunakan untuk meminta *input* berupa kata atau kalimat sampai akhir baris “\n” (menekan tombol enter).

```
#include<stdio.h>

int main() {

    char nama[50];
    char alamat[100];

    printf("Masukkan nama: ");
    gets(nama);
    printf("Masukkan alamat: ");
    gets(alamat);

    return 0;
}
```

Gambar 10. Input menggunakan gets

Catatan: Kita dapat menggunakan “[^\\n]” untuk menggantikan **gets**. “\n” merupakan salah satu *escape sequence* yang berarti *line feed/enter*. Dengan scanf ini akan mengambil sampai karakter “\n”. Karakter “\n” tidak diambil. (**dibaca: ambil sampai negasi “\n”**). Contoh: scanf(“%[^\\n]”, &variable);

Selain dari “\n” terdapat beberapa *escape sequence* yang terdapat di bahasa pemrograman C, diantaranya:

Karakter	Arti/Nilai
\\a	Alert
\\b	Backspace
\\f	Form feed
\\n	Newline
\\t	Horizontal tab
\\r	Carriage return
\\v	Vertical tab
\\\\	Backslash
\\'	Single quote
\\"	Double quote
\\?	Question mark

3. getchar

getchar digunakan untuk memasukkan sebuah nilai karakter ke variabel yang bertipe karakter.

```
#include<stdio.h>
int main() {
    char huruf;
    printf("Masukkan satu huruf: ");
    huruf = getchar();
    return 0;
}
```

Gambar 11. Input menggunakan getchar

b. Output

Dalam bahasa pemrograman C, program dapat menampilkan suatu data/nilai pada konsol (layar). Istilah yang dipakai untuk menampilkan data keluar adalah *output*. Berikut beberapa cara untuk membuat *output* di C:

1. printf

Printf digunakan untuk menampilkan data sesuai dengan format yang diberikan. Berikut beberapa variasi format:

- %s: menampilkan seluruh karakter pada string.
- %Ns: menampilkan semua karakter rata kanan dengan lebar N posisi.
- %-Ns: menampilkan semua karakter rata kiri dengan lebar N posisi.
- %N.Ms: menampilkan rata kanan hanya M karakter pertama saja dengan lebar N posisi.

Catatan:

N & M: angka/jumlah

Catatan: untuk format pencetakan (%s, %d, %c, dan lainnya), perhatikan format yang sudah ada pada input-scanf.

```
#include<stdio.h>

int main() {

    char nama[50];
    int umur;
    float ipk;

    printf("Masukkan nama: ");
    scanf("%[^\\n]", nama); getchar(); // Pakai [^\\n] agar bisa input > 1 kata
    printf("Masukkan umur: ");
    scanf("%d", &umur); getchar();
    printf("Masukkan ipk: ");
    scanf("%f", &ipk); getchar();

    printf("Perkenalkan nama saya %s, umur saya %d, dan ipk saya %f\\n", nama, umur, ipk);

    return 0;
}
```

Gambar 12. Output menggunakan printf

2. putchar

Untuk menampilkan hanya 1 karakter. Biasanya digunakan ketika yang ingin diprint adalah tipe data *char*.

```
#include<stdio.h>
int main() {
    char nilai;
    printf("Masukkan nilai: ");
    scanf("%c", &nilai); getchar();
    putchar(nilai);
    return 0;
}
```

Gambar 13. Output menggunakan putchar

3. puts

Untuk menampilkan nilai string dengan disertai *line feed* atau *escape sequence* ganti baris ("**\\n**").

```
#include<stdio.h>

int main() {

    puts("Halo semua!");
    puts("Saya sedang belajar Bahasa Pemrograman C!");

    return 0;
}
```

Gambar 14. Output menggunakan puts

E. Operator

Pada bahasa pemrograman C juga terdapat operator yang bisa digunakan. Opeartor dalam bahasa pemrograman merupakan sebuah simbol yang digunakan untuk melakukan operasi matematik, relasi, logika dan lain-lain. Berikut beberapa jenis operator yang terdapat dalam bahasa pemrograman C:

a. Operator Aritmatika

Operator	Kegunaan
+	Untuk menjumlahkan dua buah operan atau variabel
-	Untuk mengurangi satu operan atau variabel dengan operan atau variabel
*	Untuk mengkalikan dua buah operan atau variabel
/	Untuk membagi satu operan atau variabel dengan operan atau variabel
%	Untuk menentukan menentukan nilai sisa dari suatu pembagian

b. Operator Increment & Decrement

1. Increment (++)

i. *Pre-increment*

Untuk menambahkan sebuah variabel sebanyak satu nilai sebelum menjalankan suatu proses. Operator *pre-increment* didefinisikan dengan meletakkan dua operator (+) tepat sebelum variable tersebut.

ii. *Post-increment*

Untuk menambahkan sebuah variabel sebanyak satu nilai setelah menjalankan suatu proses. Operator *pre-increment* didefinisikan dengan meletakkan dua operator (+) tepat setelah variable tersebut.

2. Decrement (--)

i. *Pre-decrement*

Untuk mengurangi sebuah variabel sebanyak satu nilai sebelum menjalankan suatu proses. Operator *pre-decrement* didefinisikan dengan meletakkan dua operator (-) tepat sebelum variable tersebut.

ii. *Post-decrement*

Untuk mengurangi sebuah variabel sebanyak satu nilai setelah menjalankan suatu proses. Operator *post-decrement* didefinisikan dengan meletakkan dua operator (-) tepat setelah variable tersebut.

```
#include<stdio.h>

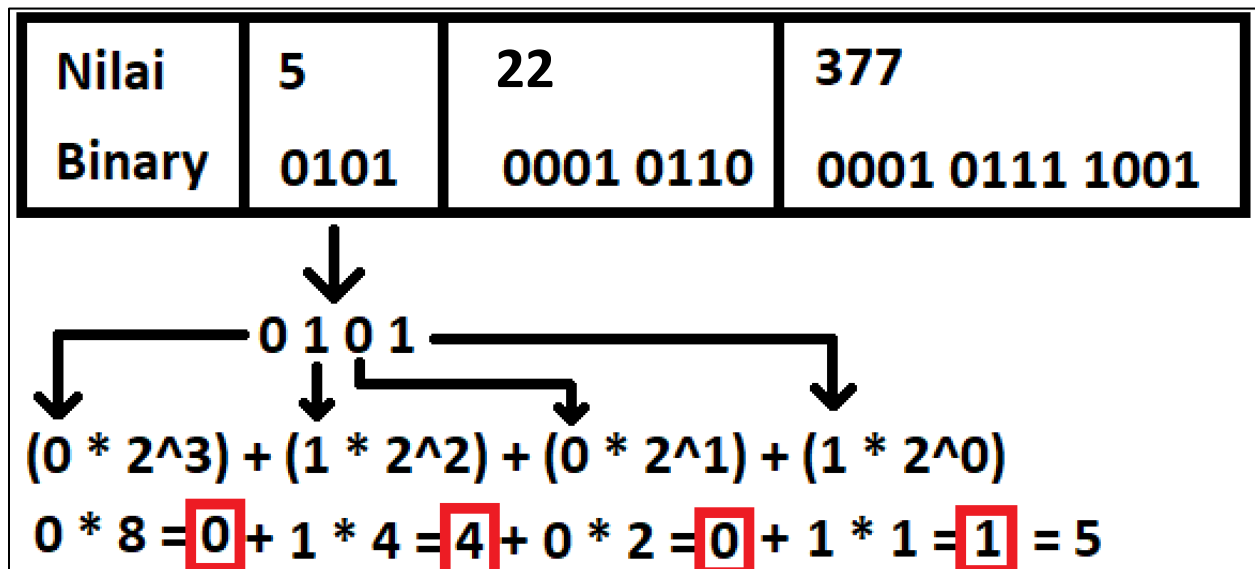
int main() {
    int a = 10, b = 3;
    printf("Penjumlahan: %d\n", a + b); // Penjumlahan: 13
    printf("Pengurangan: %d\n", a - b); // Pengurangan: 7
    printf("Perkalian: %d\n", a * b); // Perkalian: 30
    printf("Penjumlahan: %d\n", a / b); // Pembagian: 3
    printf("Penjumlahan: %d\n", a % b); // Modulus: 1
    printf("Postincrement: %d\n", a++); // Postincrement -> a = 13
    printf("Preincrement: %d\n", ++a); // Preincrement -> a = 12
    printf("Postdecrement: %d\n", a--); // Postdecrement -> a = 12
    printf("Predecrement: %d\n", --a); // Predecrement -> a = 10

    return 0;
}
```

Gambar 15. Operasi Aritmatika

c. Operator *Bitwise*

Dalam bahasa pemrograman C, setiap bilangan bulat memiliki bentuk binernya (*binary*). Sebelum masuk ke penjelasan mengenai operator bitwise, tentu kita harus memahami mengenai *binary* terlebih dahulu.



Gambar 16. Contoh bilangan biner

Bentuk biner merupakan notasi sebuah angka dengan angka 0 dan 1 saja. Jika dilihat dari gambar di atas, angka 5 dinotasikan dalam bentuk biner menjadi 0101. Cara membaca bilangan biner adalah

dari kanan ke kiri. 0101 berarti sama dengan $(1 * 2^0) + (0 * 2^1) + (1 * 2^2) + (0 * 2^3) = 5$. Untuk membantu dalam memanipulasi bilangan biner tersebut, kita bisa menggunakan operator *bitwise* yang tersedia di bahasa pemrograman C. Ada hal yang perlu kita pahami lagi sebelum membahas operator *bitwise*, yaitu logika matematika secara dasar yang terdiri dari:

p	q	$p \wedge q$ (p and q)	$p \vee q$ (p or q)	$p \underline{\vee} q$ (p xor q)
T	T	T	T	F
T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

Catatan:

T: True

F: False

Untuk sebagian besar operator pada logika matematika mirip penggunaannya pada operator *bitwise*. Berikut beberapa jenis operator *bitwise*:

Operator	Kegunaan
&	Hasil dari bitwise AND adalah 1 jika semua bit dari dua operan adalah 1. Jika salah satu bit dari operan adalah 0, maka hasilnya adalah 0
	Hasil dari bitwise OR adalah 1 jika salah satu bit dari dua operan adalah 1. Jika salah satu bit dari operan adalah 1, maka hasilnya adalah 1
^	Hasil dari bitwise XOR adalah 1 jika kedua bit dari dua operan berbeda. Jika kedua bit dari operan adalah sama, maka hasilnya adalah 0
~	Hasil dari bitwise complement adalah mengubah binari 1 menjadi 0 dan binary 0 menjadi 1 pada satu operan
>>	Digunakan untuk menggeser semua bit ke kanan dengan jumlah yang telah ditentukan

<<	Digunakan untuk menggeser semua bit ke kiri dengan jumlah yang telah ditentukan. Posisi yang kosong akan diisi dengan 0
----	---

```
#include<stdio.h>

int main() {
    int a = 19, b = 7;
    printf("Bitwise AND: %d\n", a & b); // Bitwise AND: 3
    printf("Bitwise OR: %d\n", a | b); // Bitwise OR: 23
    printf("Bitwise XOR: %d\n", a ^ b); // Bitwise XOR: 20
    printf("Bitwise Complement: %d\n", ~a); // Bitwise Complement: -20
    printf("Left Shift: %d\n", a << 2); // Left Shift: 76
    printf("Right Shift: %d\n", a >> 2); // Right Shift: 4

    return 0;
}
```

Gambar 17. Operator Bitwise

d. Operator Logika

Operator	Kegunaan
&&	Operator 'dan'
	Operator 'atau'
!	Operator 'negasi'

Untuk operator logika, biasanya digunakan untuk memberikan kondisi pada struktur kendali pemilihan dan perulangan yang akan dijelaskan di bagian selanjutnya. Penggunaan logika nya sama dengan logika matematika yang sudah kita pelajari sebelumnya di operator *bitwise*.

e. Operator Relasi

Operator	Kegunaan
<	Operator kurang dari
>	Operator lebih dari
<=	Operator kurang dari sama dengan
>=	Operator lebih dari sama dengan
==	Operator sama dengan

!=	Operator tidak sama dengan
----	----------------------------

Untuk operator relasi, juga biasanya digunakan untuk memberikan kondisi pada struktur kendali pemilihan dan perulangan yang akan dijelaskan di bagian selanjutnya. Operator relasi digunakan untuk membandingkan dua buah operan dan akan mengembalikan nilai benar (*true*) atau salah (*false*).

f. Operator *Shorthand*

Operator *shorthand* merupakan cara lebih pendek untuk mendeklarasikan suatu operasi, namun menghasilkan hasil yang sama.

```
#include<stdio.h>

int main() {
    int a = 10, b = 3;
    a += b; // Notasi ini sama saja hasilnya dengan a = a + b;

    return 0;
}
```

Gambar 18. Operator *Shorthand*

Catatan: Operator *shorthand* bisa diaplikasikan ke semua jenis operator (tidak hanya operator penjumlahan)

Latihan Soal Input, Output, Tipe Data, Variabel, dan Operator

```
=====
|| Simulasi Jual Barang ||
=====
Masukkan nama barang [ >= 1 kata]: Bandeng Presto
Masukkan harga barang: 35000
Masukkan jumlah barang: 4

Jadi barang yang terjual adalah 4 Bandeng Presto dan mendapat keuntungan 133000!
Tekan enter untuk mengakhiri...
```

Gambar 19. Latihan Soal Input, Output, Tipe Data, Variabel, dan Operator

Buatlah program seperti gambar di atas, dengan ketentuan:

1. Program akan meminta 3 buah *input*, yaitu *input* nama barang dan bisa menerima lebih sama dengan satu kata, kedua adalah *input* harga satuan barang, terakhir *input* jumlah barang yang akan dijual.
2. Jika sudah ter-*input* semuanya, program akan mengkalkulasi total keuntungan yang akan didapat dari perkalian harga satuan barang dan jumlah barang yang dijual.
3. Terakhir program akan memberikan *output* dengan format:

```
"Jadi barang yang terjual adalah [jumlah_barang] [nama_barang] dan mendapat keuntungan
[total_keuntungan]!"
"Tekan enter untuk mengakhiri..."
```

F. Seleksi (Selection)

Dalam bahasa pemrograman C, terdapat struktur kendali pemilihan yang digunakan untuk memilih dan mengeksekusi suatu *statement* yang dideklarasikan berdasarkan kondisi yang diberikan. Untuk jenis seleksi yang terdapat di C ada empat, yaitu *if*, *else*, *else if* dan *switch case*. Berikut penjelasan untuk setiap jenis repetisi:

a. if

If digunakan untuk melakukan pengecekan atau menyeleksi apakah suatu kondisi terpenuhi atau tidak. Jika terpenuhi, amaka *statement* yang ada di dalamnya akan di eksekusi.

```
if (condition) {  
    <block of statements>  
}
```

```
#include<stdio.h>  
int main() {  
  
    int umur = 21;  
  
    if(umur >= 21) {  
        printf("Saya sudah dewasa!");  
    }  
  
    return 0;  
}
```

Gambar 20. Struktur kendali pemilihan menggunakan if (relasi lebih dari sama dengan)

```
#include<stdio.h>  
int main() {  
  
    int umur = 21;  
  
    if(umur != 15) {  
        printf("Sekarang saya tidak berumur 15 tahun.");  
    }  
  
    return 0;  
}
```

Gambar 21. Struktur kendali pemilihan menggunakan if (relasi tidak sama dengan)

```
#include<stdio.h>
int main() {

    int umur = 19;
    char statusMahasiswa = 'Y';

    if(umur >= 18 && statusMahasiswa == 'Y') {
        printf("Sekarang saya sedang berkuliah.\n");
    }

    if(umur >= 21 || statusMahasiswa == 'Y') {
        printf("Saya beranjak dewasa.\n");
    }

    return 0;
}
```

Gambar 22. Struktur kendali pemilihan menggunakan if (logika and dan or)

b. else

Else digunakan untuk memberikan alternatif terakhir ketika kondisi-kondisi yang sebelumnya tidak terpenuhi tanpa perlu adanya kondisi lain untuk melakukan *statement* yang di dalamnya. Jadi jika kondisi *if* diawal tidak terpenuhi, maka program akan mengeksekusi *statement* yang ada di dalam *else*.

```
if (condition) {
    <block of statements>
} else {
    <block of statements>
}
```

```
#include<stdio.h>
int main() {

    int umur = 15;

    if(umur >= 21) {
        printf("Saya sudah dewasa!");
    } else {
        printf("Saya belum dewasa!");
    }

    return 0;
}
```

Gambar 23. Struktur kendali pemilihan menggunakan else

Dari potongan koding di atas akan menghasilkan "Saya belum dewasa!". Ini dikarenakan kondisi *if* di awal tidak terpenuhi dan akan masuk ke kondisi *else* dan mengeksekusi *statement* yang ada di dalamnya.

c. else if

Else if digunakan untuk memberikan alternatif lain ketika kondisi-kondisi yang sebelumnya tidak terpenuhi dengan memberikan kondisi lain sebelum melakukan *statement* yang di dalamnya. Jadi jika kondisi *if* diawal tidak terpenuhi, maka program akan mengeksekusi *statement* yang ada di dalam *else if* jika kondisi yang terdapat di *else if* terpenuhi.

```
#include<stdio.h>
int main() {

    int nilai = 70;

    if(nilai >= 90) {
        printf("Saya mendapatkan nilai A!");
    } else if (nilai >= 75) {
        printf("Saya mendapatkan nilai B!");
    } else if (nilai >= 65) {
        printf("Saya mendapatkan nilai C!");
    } else {
        printf("Saya mendapatkan nilai D!");
    }

    return 0;
}
```

Gambar 24. Struktur kendali pemilihan menggunakan *else if*

Jika melihat dari potongan koding di atas, kondisi pertama dan kedua tidak terpenuhi. Dan nilai yang memenuhi ada di kondisi yang ketiga. Maka dari itu potongan koding ini akan mengeksekusi *statement* yang ada di kondisi nilai ≥ 65 .

d. switch case

Switch case digunakan untuk menyeleksi suatu kumpulan syarat tertentu terhadap suatu variabel/*expression* yang diberikan.

```

switch (expression) {
    case constant-expression-1: {statements-1; break;}
    [case constant-expression-2: {statements-2; break;}]
    ...
    [case constant-expression-n: {statements-n; break;}]
    [default: {default-statements; break;}]
}

```

Catatan:

[]: *statement* opsional

{}: *scope* untuk setiap *case*

Expression diatas biasanya diisi dengan sebuah variabel yang akan dicek. Lalu untuk *constant-expression* merupakan kasus-kasus yang tersedia. Ketika nilai dari variabel/*expression* sama dengan salah satu *constant-expression*, maka program akan mengeksekusi *statement* yang ada di dalamnya. Keyword **break** pada format di atas digunakan untuk menandakan bahwa setelah mengeksekusi salah satu *statement* maka *statement-statement* yang ada di bawahnya tidak akan dieksekusi. Untuk keyword **default** berfungsi untuk memberikan *statement* yang akan dieksekusi jika tidak ada satupun kondisi yang terpenuhi.

```

#include<stdio.h>
int main() {

    char nilai = 'B';
    switch(nilai) {
        case 'A': {
            printf("Nilai anda sangat bagus!"); break;
        }
        case 'B': {
            printf("Nilai anda cukup bagus!"); break;
        }
        case 'C': {
            printf("Nilai anda cukup!"); break;
        }
        default: {
            printf("Nilai anda tidak ada dalam kategori yang disediakan!"); break;
        }
    }
    return 0;
}

```

Gambar 25. Struktur kendali pemilihan menggunakan switch case

Potongan koding di atas akan menghasilkan “Nilai anda cukup bagus!”.

Latihan Soal Seleksi

```
=====
|| Cek Angka Ganjil Genap ||
=====
Masukkan angka: 13

Angka 13 merupakan bilangan ganjil!
Tekan enter untuk mengakhiri...
```

Gambar 26. Latihan Soal Seleksi (Angka Ganjil)

```
=====
|| Cek Angka Ganjil Genap ||
=====
Masukkan angka: 128

Angka 128 merupakan bilangan genap!
Tekan enter untuk mengakhiri...
```

Gambar 27. Latihan Soal Seleksi (Angka Genap)

Buatlah program seperti gambar di atas, dengan ketentuan:

1. Program akan meminta 1 buah *input*, yaitu *input* sebuah angka.
2. Jika sudah *input* sebuah angka, program akan cek apakah angka tersebut bilangan ganjil atau genap.
3. Terakhir program akan memberikan *output* dengan format:

"Angka [angka] merupakan bilangan [genap/ganjil]!"
"Tekan enter untuk mengakhiri..."

G. Perulangan (Repetition)

Dalam bahasa pemrograman C, terdapat struktur kendali perulangan yang digunakan untuk mengulang suatu *statement* yang dideklarasikan berdasarkan kondisi yang diberikan. Pada Bahasa C, terdapat 3 jenis repetisi, yaitu *for*, *do ... while*, dan *while*. Ketiganya memiliki ciri khasnya tersendiri, dan cara deklarasinya pun berbeda. Berikut penjelasan untuk setiap jenis repetisi:

a. for

For merupakan struktur kendali perulangan yang menerima 3 parameter untuk struktur dasarnya, dimana itu terdiri dari inialisasi variabel awal, kondisi perulangan, dan aksi yang dilakukan setelah *statement* yang terdapat di dalam repetisi *for* dieksekusi, yang biasanya adalah *increment/decrement* variabel yang diinisialisasi. Untuk ketiga parameter tersebut bersifat opsional. Format untuk *for*:

```
for ( initialization ; condition ; statement ) {  
    <block of statements>  
}
```

Alur untuk repetisi *for* adalah inialisasi variabel, cek kondisi (jika kondisi tidak terpenuhi, maka *statement* yang ada di dalam *for* tidak akan dijalankan), menjalankan *block of statements*, terakhir baru menjalankan *after statement*, yang biasanya diisi dengan *increment/decrement*.

```
#include<stdio.h>  
  
int main() {  
    for(int i = 0; i < 10; i++) {  
        printf("%d\n", i);  
    }  
    return 0;  
}
```

Gambar 28. Struktur Kendali Perulangan menggunakan *for*

Untuk potongan kode di atas akan menghasilkan *output* angka 0 hingga 9 ke bawah. Dengan menggunakan struktur kendali perulangan kita dapat mempersingkat kode yang perlu kita tulis. Jika dengan cara manual kita dapat `printf()` kira-kira sebanyak 10 baris untuk mencetak angka 0 hingga 9, kita dapat memperpendeknya menjadi 3 baris.

b. while

While digunakan untuk mengulang suatu *statement* selama kondisi yang diberikan di dalamnya bernilai benar, jika bernilai salah, maka *statement* yang dideklarasikan di dalamnya tidak akan diulang.

```
while (condition) {  
    <block of statements>  
}
```

```
#include<stdio.h>  
  
int main() {  
    int i = 0, j = 0;  
    while(i < 10) {  
        printf("%d\n", i++);  
    }  
    while(j < 0) {  
        printf("%d\n", j++);  
    }  
    return 0;  
}
```

Gambar 29. Struktur Kendali Perulangan menggunakan *while*

Pada operasi *while* yang pertama akan menghasilkan angka 0 hingga 9 ke bawah. Namun operasi *while* yang kedua tidak akan menghasilkan apapun karena kondisi diawal tidak terpenuhi.

c. do ... while

Do ... while digunakan untuk mengulan suatu *statement* yang dideklarasikan sesudah *keyword do* dan setelah menjalankan *statement* tersebut, dilakukan pengecekan terhadap kondisi yang diberikan, apakah bernilai benar/salah. Jika bernilai benar, maka *statement* tesebut akan diulang kembali. Jika bernilai salah, maka *statement* tersebut hanya akan dijalankan satu kali tanpa dilakukan pengulangan.

```
do {  
    <block of statements>  
} while(condition);
```

```

#include<stdio.h>

int main() {
    int i = 0, j = 0;
    do {
        printf("%d\n", i++);
    } while(i < 10);
    do {
        printf("%d\n", j++);
    } while(j < 0);

    return 0;
}

```

Gambar 30. Struktur kendali perulangan menggunakan *do ... while*

Pada operasi *do ... while* yang pertama akan menghasilkan angka 0 hingga 9 ke bawah. Namun, untuk operasi *do ... while* yang kedua hanya akan menghasilkan angka 0 saja. Selain itu, *do ... while* biasanya digunakan untuk validasi *input* atau pembuatan menu, karena *do ... while* memungkinkan pengguna untuk melakukan *input* terlebih dahulu baru mengecek kondisi.

Catatan: Jadi perbedaan antara *while* dengan *do ... while* adalah minimal *do ... while* menjalankan *statement* yang ada di dalam sebanyak 1 kali meskipun kondisi awal tidak terpenuhi, sedangkan *while* tidak akan menjalankan *statement* sama sekali jika kondisi awal tidak terpenuhi.

Latihan Soal Repetisi

```
=====
|| Validasi Input ||
=====
Masukkan umur [>= 0]: -1
Masukkan umur [>= 0]: 21
Masukkan jenis kelamin [L atau P]: L

Saya seorang laki-laki berumur 21!
Tekan enter untuk mengakhiri...
```

Gambar 31. Latihan Soal Repetisi

Buatlah program seperti gambar di atas, dengan ketentuan:

1. Program akan meminta 2 buah *input*, yaitu *input* umur dengan ketentuan umur harus di atas sama dengan 0 (jika kurang dari 0 maka program akan meminta *input* ulang) dan *input* jenis kelamin 'L' (laki-laki) atau 'P' (perempuan) (jika input selain karakter 'L' atau 'P' maka program akan meminta *input* ulang).
2. Jika sudah *input* sebuah angka, program akan cek apakah *input* jenis kelamin 'L' atau 'P'.
3. Terakhir program akan memberikan *output* sesuai jenis kelamin dengan format:

"Saya seorang [laki-laki/perempuan] berumur [umur]!"

Bab 02. Array dan Pointer

Materi

Array dan Pointer:

- Array satu dimensi
 - Array banyak dimensi
 - Pointer
-

A. Array Satu Dimensi

Array adalah suatu tipe data terstruktur yang bertipe data sama dan berjumlah tetap (berdasarkan apa yang ditentukan) dan diberi suatu nama tertentu (sesuai variabel). Elemen-elemen array tersusun secara berurutan. Susunan tersebut membuat array memiliki alamat yang bersebelahan/berdampingan dalam memori sesuai dengan besar pemakaian memori tipe data yang digunakan. Format array:

```
tipe_data nama_array[n];  
n = Ukuran array
```

```
#include<stdio.h>  
int main() {  
    int a[5] = {4, 15, 7, 9, 10}; // Inisialisasi array  
    a[2] = 27; // Mengganti nilai dari sebuah array  
    printf("Nilai array indeks ke-2: %d\n", a[2]); // Akses nilai array  
    return 0;  
}
```

Gambar 32. Array satu dimensi

Untuk mengakses dan mengganti isi dari sebuah bagian array akan dimulai dari index ke 0. Jadi ketika sebuah array berukuran 5 seperti kasus di atas dan kita mau mengakses nilai 27, kita perlu mengakses array index ke-2.

Index	0	1	2	3	4
Nilai	4	15	7	9	10

Gambar 33. Ilustrasi array

B. Array Banyak Dimensi

Dalam bahasa pemrograman C, array tidak hanya bisa memiliki satu dimensi, namun juga dapat memiliki banyak dimensi. Salah satu contohnya adalah array dua dimensi yang sering digunakan dalam konsep matriks. Format array dua dimensi:

```
tipe_data nama_array[n][m];
```

n = Ukuran array untuk banyak baris

m = Ukuran array untuk banyak kolom

```
#include<stdio.h>
int main() {

    int m[4][3] =
    {
        {11, 12, 13},
        {21, 22, 23},
        {31, 0, 33},
        {41, 42, 43}
    }; // Inisialisasi array 2D
    m[2][1] = 32; // Mengganti nilai dari sebuah array 2D
    printf("Nilai array 2D baris ke-3, kolom ke-2: %d\n", a[2][1]); // Akses nilai array
    2D

    return 0;
}
```

Gambar 34. Array dua dimensi

Selain untuk matriks, masih banyak kegunaan *array* 2D, seperti membuat peta untuk permainan konsol, menyimpan kumpulan nama mahasiswa, dan masih banyak lagi.

```

#include<stdio.h>
int main() {
    char peta[10][11] =
    {
        "#####",
        "# #      F ",
        "# # #####",
        "# #      #",
        "# # # # #",
        "#      # #",
        "## # # # #",
        "# #      #",
        "# # # # #",
        "#S#####",
    }; // Membuat peta labirin

    return 0;
}

```

Gambar 35. Contoh lain dari penggunaan array 2D (peta labirin)

C. Pointer

Pointer adalah suatu variabel yang berisi alamat memori dari suatu variabel lain. Pointer dilambangkan dengan operator “*” dan biasanya digunakan untuk memanipulasi isi variabel melalui alamat memori yang disimpan. Untuk membuat suatu pointer dari sebuah variabel, pointer tersebut perlu memiliki tipe data yang sama. Dan untuk mengambil alamat memori dari sebuah variabel kita dapat menggunakan tanda “&” di depan nama variabelnya. Setiap pointer juga memiliki alamat memori nya sendiri.

```

#include<stdio.h>
int main() {

    int a = 10;
    int *p = &a;

    *p = 15; // Mengganti isi nilai variabel a melalui pointer p

    printf("Nilai variabel a: %d\n", a);
    printf("Alamat variabel a: %d\n", &a);
    printf("Akses nilai variabel a menggunakan pointer p: %d\n", *p);
    printf("Akses nilai alamat a menggunakan pointer p: %d\n", p);
    printf("Alamat pointer p: %d\n", &p);

    return 0;
}

```

Gambar 36. Pointer

Latihan Soal Array

```
Pilih ukuran matriks:
1. 3x3
2. 4x4
Pilih: 3
Pilih ukuran matriks:
1. 3x3
2. 4x4
Pilih: 1
Masukkan matriks baris ke-1 kolom ke-1: 1
Masukkan matriks baris ke-1 kolom ke-2: 2
Masukkan matriks baris ke-1 kolom ke-3: 3
Masukkan matriks baris ke-2 kolom ke-1: 4
Masukkan matriks baris ke-2 kolom ke-2: 5
Masukkan matriks baris ke-2 kolom ke-3: 6
Masukkan matriks baris ke-3 kolom ke-1: 7
Masukkan matriks baris ke-3 kolom ke-2: 8
Masukkan matriks baris ke-3 kolom ke-3: 9

Hasil matriks:
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |
Sum matriks: 45
```

Gambar 37. Latihan Soal Array

Buatlah program seperti gambar di atas, dengan ketentuan:

1. Program akan meminta 1 buah *input* utama, yaitu *input* untuk memilih ukuran matriks dan ada validasi untuk input tersebut antara angka 1 atau 2 saja (jika selain itu maka program akan minta *input* ulang).
2. Jika sudah memilih ukuran matriks, program akan meminta *input* setiap sel dari matriks yang ada mulai dari baris pertama hingga baris terakhir seperti Gambar 37.
3. Lalu program akan menghitung jumlah total dari semua angka yang ada di setiap sel.
4. Terakhir program akan mengeluarkan *output* dengan format:

"Hasil matriks:

```
[  
| [angka_1] [angka_2] [angka_3] |  
| [angka_4] [angka_7] [angka_6] |  
| [angka_7] [angka_8] [angka_9] |  
/  
| [angka_1] [angka_2] [angka_3] [angka_4] |  
| [angka_5] [angka_7] [angka_6] [angka_8] |  
| [angka_9] [angka_10] [angka_11] [angka_12] |  
| [angka_13] [angka_14] [angka_15] [angka_16] |  
]
```

Sum matriks: [total_semua_angka]"