

# **BAHASA C**



**LABORATORIUM SISTEM INFORMASI**

2009

## DAFTAR ISI

Halaman	
PENGENALAN PEMROGRAMAN BAHASA C .....	1
Sekilas Bahasa C .....	1
Struktur Bahasa C .....	2
Variable, Statement dan Preprocessor .....	5
Tipe data Bahasa C .....	7
OPERATOR BAHASA C .....	9
Operator Aritmatika .....	9
Operator Unary .....	10
Operator Pengerjaan .....	11
Operator Logika .....	11
Operator Koma .....	11
Operator Bitwise .....	12
FORMAT UNTUK SETIAP TIPE DATA .....	13
FUNGSI .....	15
Data Tidak Terformat .....	15
Data Terformat .....	17
FUNGSI OUTPUT .....	18
Hasil Tidak Terformat .....	19
Hasil Terformat .....	19
PENGAMBILAN KEPUTUSAN DALAM BAHASA C .....	23
Statement IF .....	23
Statement IF... Else .....	23
Statement Switch Case .....	26

PERULANGAN DALAM BAHASA C .....	29
Statement For .....	29
Statement While .....	30
Statement Do While .....	31
STATEMENT CONTINUE DAN GO TO .....	31
ARRAY DAN DATA TERSTRUKTUR DALAM BAHASA C .....	33
FUNCTION DALAM BAHASA C .....	42
POINTER DALAM BAHASA C .....	54

## **KATA PENGANTAR**

C adalah bahasa yang standar artinya suatu program yang ditulis dengan versi bahasa C tertentu akan dapat di kompilasi dengan versi bahasa C yang lain dengan sedikit modifikasi. Bahasa C tersedia hamper disetiap jenis computer, Kode bahasa C sifatnya adalah portable dan didukung dengan pustaka yang banyak.

Melalui praktikum C, diharapkan mahasiswa dapat merasakan kemudahan dan kelebihan dalam membuat suatu program.

Modul Bahasa C berisi tentang pengertian bahasa C, kemudian diawali mulai dari awal penulisan program, fungsi – fungsi input dan output bahasa C hingga dihasilkan output dari program tersebut.

# PENGENALAN PROGRAM DALAM BAHASA C

1

## **Obyektif :**

1. Mengetahui Sejarah Bahasa C
  2. Mengerti tentang Struktur Bahasa C
  3. Mengenal dan dapat menggunakan Variable
- 

## **Sekilas Bahasa C**

Akar dari bahasa C adalah BCPL yang dikembangkan oleh Martin Richards pada tahun 1967. Selanjutnya bahasa ini memberikan ide kepada Ken Thompson, yang kemudian mengembangkan bahasa yang disebut bahasa B pada tahun 1970. Perkembangan selanjutnya dari bahasa B adalah bahasa C oleh Dennis Ritchie sekitar tahun 1970-an di Bell

Telephone Laboratories Inc. (sekarang adalah AT&T Bell Laboratories).

Bahasa C pertama kali digunakan di computer Digital Equipment Corporation PDP-11 yang menggunakan system operasi UNIX. C adalah bahasa standar, artinya suatu program ditulis dengan versi bahasa C tertentu akan dapat dikompilasi dengan versi bahasa C yang lain dengan sedikit modifikasi. Kepopuleran C membuat versi-versi yang banyak untuk komputer mikro. Untuk menstandarisasikannya maka dibentuk komite pada tahun 1983 yang menetapkan standar ANSI untuk bahasa C.

Hingga saat ini penggunaan bahasa C telah merata di seluruh dunia. Hampir semua perguruan tinggi di dunia menjadikan bahasa C sebagai salah satu mata kuliah wajib. Selain itu, banyak bahasa pemrograman populer seperti PHP dan Java menggunakan sintaks dasar yang mirip bahasa C.

Adapun Karakteristik Bahasa C yaitu:

a. Kelebihan

- Bahasa C tersedia hampir di semua jenis Komputer
- Kode bahasa C sifatnya adalah Portabel (dapat digunakan di semua jenis komputer).
- Bahasa C hanya menyediakan sedikit kata-kata kunci
- Proses executable program lebih cepat
- Dukungan pustaka yang banyak
- C bahasa pemrograman yang terstruktur dan merupakan bahasa tingkat tinggi, namun dapat pula dikatakan sebagai bahasa tingkat menengah karena mempunyai kemampuan seperti bahasa *low level* (tingkat rendah).

b. Kelemahan

- Bahasa C merupakan bahasa yang cukup sulit dimengerti terutama dalam hal sintaks-sintaksnya

## Struktur Program Bahasa C

Bahasa C dikatakan sebagai bahasa pemrograman terstruktur, karena strukturnya menggunakan fungsi-fungsi sebagai program-program bagian (sub rutin). Fungsi utama yang pertama kali diproses adalah fungsi yang bernama "***main()***", selain itu adalah merupakan fungsi-fungsi bagian yang dapat ditulis setelah fungsi "***main()***" atau diletakan di file pustaka (***library***).

Jika fungsi-fungsi yang tersedia pada file pustaka (library) dan akan dipakai di suatu program maka file judulnya (***Header***) nya harus disertakan/ditulis pada program dengan menggunakan preprocessor directive ***#include***, contohnya ***#include <stdio.h>*** yang berarti akan mengambil fungsi-fungsi yang tersedia pada library standar IO, diantaranya ***clrscr()***, ***printf***, ***scanf()***, dll.

Untuk komentar / penjelasan program harus diapit diantara (*/\* ...\*/*).

*Penulisan program bahasa C bersifat **Case Sensitive** artinya penamaan fungsi, variabel maupun keyword-nya membedakan huruf kecil dan besar.*

Adapun struktur bahasa C sebagai berikut:

**#include < file-file header>**

*/\* directive #include digunakan jika mengambil fungsi-fungsi library\*/*

**main()** */\* fungsi utama sbg awal dimulainya pemrosesan program \*/*

```
{           /*blok awal*/
    statemen-statemen;
}           /*blok akhir*/
```

**Fungsi-fungsi lain()**

```
{
    statemen-statemen;
}
```

Pemakaian #include pustaka (library) yang umum

- conio.h = Tampilan Layar : clrscr(), textcolor(), textbackground(), textattr(), dll.
- stdio.h = standart input/output : clrscr(), printf(), scanf(), puts(), gets(), getch(), cprintf(), dll.
- stdlib.h = standart library (akses file / stream) : fopen(), flose(), fread(), fwrite(), fprintf(), fscanf(), dll.

- `alloc.h/malloc.h` = pengalokasian memori (memori dinamis) : `malloc()`, `free()`, `calloc()`, dll.

Pemakaian Escape Sequence pada fungsi output (`printf()`, `gets()`, dll) Karakter Escape Sequence sering digunakan untuk menampilkan hasil output, seperti mengganti baris, membunyikan bel, dll).

Karakter Escape	Arti
<code>\a</code>	Bunyi bel
<code>\b</code>	Mundur satu spai
<code>\f</code>	Ganti halaman
<code>\n</code>	Ganti baris baru
<code>\r</code>	Ke kolom awal, baris yang sama
<code>\t</code>	Tabulasi horisontal
<code>\v</code>	Tabulasi vertikal
<code>\0</code>	Null
<code>\'</code>	Petik tunggal
<code>\"</code>	Kutip
<code>\\</code>	Garis miring terbaik



Contoh program sederhana:

```
#include <stdio.h>
main()
{
    clrscr();
    printf("Hallo kawan-kawan....!\n"); /* \n : ganti baris baru */
    printf("Saya sedang belajar bahasa C...nih..susah looh...!");
}
```

### **Variable, Statement dan Preprocessor**

Mendeklarasikan **variabel** dapat dituliskan di dalam maupun di luar fungsi. Jika deklarasi variabel ditulis pada luar fungsi, maka variabel tersebut dapat di akses oleh semua fungsi-fungsi yang lain ( **Variabel Global** ) sedangkan jika di dalam fungsi, maka variabel tersebut hanya dapat diakses oleh fungsi itu sendiri ( **Variabel Lokal** ). Penamaan variabel tidak boleh mengandung spasi, tidak sama dengan nama keyword, tidak dimulai dengan angka serta tidak mengandung simbol-simbol yang telah ditentukan oleh BAHASA C.

```
char A;
int X, Y, my_var; /* X bertipe integer, Y bertipe integer, my_var
bertipe integer*/
float bil_pecahan=9.56; /* memberikan nilai awal = 9.56 untuk
variabel bil_pecahan */
```

**Statement** adalah pernyataan yang menyebabkan suatu tindakan dilakukan oleh komputer. Statement dalam bahasa C diakhiri dengan tanda titik koma ( ; ) . Jenis statement diantaranya :

- a. Empty Statement / Null Statement

Statemen kosong adalah statement yang hanya terdiri dari pengakhiran titik koma saja, sehingga tidak melakukan tindakan apapun. Digunakan untuk membuat perulangan kosong yang dimaksudkan untuk memberi jarak ke proses selanjutnya.

b. Expression Statement

Statemen ungkapan merupakan statemen yang dibentuk dari ungkapan yang diakhiri dengan titik koma.

c. Control Statement

Statemen kendali merupakan statemen yang berfungsi untuk mengendalikan proses dari program, dapat berupa proses seleksi kondisi, perulangan atau lompatan. Statemen ini dibentuk dengan menggunakan kata kunci `if`, `switch`, `do-while`, `goto`, `break` dan `continue`.

d. Coumpound Statement/Block Statement

Statemen jamak adalah statemen yang terdiri dari beberapa statemen tunggal yang ditulis diantara tanda kurung kurawal ( { } )

**Konstanta** merupakan nilai yang tidak dapat diubah selama proses dari program. Untuk menentukan konstanta dapat dilakukan dengan menggunakan variabel atau macro.

Jika menggunakan perantara variabel digunakan keyword **const** :

```
const my_konstanta = 306; /* nilai pada variabel my_konstanta
adalah tetap 306 */
```

Jika secara langsung menggunakan macro atau directive **#define**:

```
#define my_konstanta = 306 /* MACRO tidak perlu diakhiri titik
koma (;) */
```

```
float nilai = my_konstanta; /* memberikan variabel nilai dengan
my_konstanta (nilai=306) */
```

Contoh :

```
#include <stdio.h>
#define PI=3.14
/* Perhatikan huruf kecil dan besarnya */
main()
{
    int a = 2, B, niLAI;
    float bil_pecahan;

    B = 4;
    bil_pecahan = 2.0;
    niLAI = a * (2 + B);
    printf(" Hasil dari %d * (2 + %d) = %d \n",a,b,niLAI);
    printf("Perkalian dari %4.2f * %4.2f = %4.2f \n", bil_pecahan,
    PI, bil_pecahan*PI);

    /* Format specifier untuk %d dan %f akan dijelaskan pada
    sesion berikutnya */
}
```

### **Tipe Data Bahasa C**

Bahasa C menyediakan 5 macam tipe data dasar, yaitu tipe data integer (numerik bulat dideklarasikan dengan int), floating point (numerik pecahan ketepatan tunggal dideklarasikan dengan float), double precision (numerik pecahan ketepatan ganda dideklarasikan dengan double), karakter (dideklarasikan dengan char) dan kosong (dideklarasikan dengan void). Untuk int, float, double dan char dapat dikombinasikan dengan pengubah (*modifier*) signed, unsigned, long, short, maka hasilnya menjadi seperti pada tabel berikut.

**Tabel1. Tipe Data Bahasa C**

TIPE	Lebar	Jangkauan Nilai
int	16 bit	- 32768 s/d 32767
signed int		
short int		
signed short int		
unsigned int	16 bit	0 s/d 65535
unsigned short int		
long int	32 bit	- 2147483648 s/d 2147483649
signed long int		
unsigned long int	32 bit	0 s/d 4294967296
float	32 bit	3.4E-38 s/d 3.4E+38
double	64 bit	1.7E-308 s/d 1.7E+308
long double	80 bit	3.4E-4932 s/d 3.4E+4932
char	8 bit	- 128 s/d 127
signed char		
unsigned char	8 bit	0 s/d 255

## OPERATOR BAHASA C

2

### Obyektif :

4. Mengetahui macam-macam operator dalam Bahasa C.
  5. Mengetahui dan dapat menggunakan format pada tiap tipe data..
- 

Operator adalah suatu tanda atau simbol yang digunakan untuk suatu operasi tertentu. Bahasa C menyediakan operator Pengerjaan, operator Aritmatika, operator tipe, operator hubungan, operator logika, operator bitwise, operator dan operator koma.

### Operator Aritmatika (Arithmetic operator)

Operator	Fungsi
*	Perkalian
/	Pembagian
%	Pembagian modulo (Sisa pembagian)
+	Penjumlahan
-	Pengurangan

Operator aritmatika melibatkan 2 buah operand, terkadang operand yang digunakan berbeda tipenya. Untuk menghindari hal-hal yang tidak diinginkan maka kompiler C mempunyai pedoman untuk operand yang berbeda tipe :

1. Tipe char akan dikonversikan ke tipe int
2. Tipe float akan dikonversikan ke tipe double
3. Jenjang tertinggi adalah mulai dari long double, double, long int, unsigned int, dan int. ini berarti tipe double dioperasikan dengan tipe int akan menghasilkan tipe double.

### Operator Unary (Unary Operator)

Operator unary merupakan operator yang hanya menggunakan sebuah operand saja. Operator-operator unary mempunyai jenjang 2.

Operator	Fungsi
-	Unary minus
++	Increase dgn penambahan nilai 1
--	Decrease dengan pengurangan nilai 1
(tipe)	Cast
sizeof	Ukuran operand dalam byte
!	unary NOT
~	Komplemen 1 (bitwise NOT)
&	Menghasilkan alamat memori operand(operator pointer)
*	Menghasilkan nilai pengenalan dialamatnya(operator pointer)

### Operator Pengerjaan (Assignment Operator)

Operator pengerjaan digunakan untuk memindahkan nilai dari suatu ungkapan kesuatu pengenalan. Operator pengerjaan mempunyai jenjang 14.

Operator	Contoh	Ekuivalen dengan
=	$A = B + C$	Mengerjakan $B + C$ ke $A$
+=	$A += 1$	$A = A + 1$
-=	$A -= B$	$A = A - B$
*=	$A *= B$	$A = A * B$
/=	$A /= B$	$A = A / B$
%=	$A \% = B$	$A = A \% B$

### **Operator Hubungan (Relational Operator)**

Operator hubungan digunakan untuk menunjukkan hubungan antara 2 buah operand. Banyak digunakan untuk penyeleksian kondisi dengan statement if, do-while, atau while.

Operator	Fungsi	Jenjang
<	Lebih kecil dari	6
<=	Lebih kecil atau sama dengan	6
>	Lebih besar dari	6
>=	Lebih besar atau sama dengan	6
==	Sama dengan	7
!=	Tidak sama dengan	7

### **Operator Logika (Logical Operator)**

Operator logika digunakan untuk membandingkan logika hasil dari operator-operator hubungan.

Operator	Fungsi	Jenjang
&&	Logika DAN (AND)	11
	Logika ATAU (OR)	12

### **Operator Koma (Comma Operator)**

Operator koma digunakan untuk menggabungkan beberapa ungkapan dengan proses yang berurutan dari ungkapan sebelah kiri koma ke ungkapan sebelah kanan koma. Operator koma mempunyai jenjang 16.

### **Operator Bitwise**

Operator bitwise digunakan untuk memanipulasi bit-bit nilai data yang ada di memori. Operator-operator ini hanya dapat digunakan untuk tipe data char, int, dan long int.

Operator	Fungsi	Jenjang
<<	Pergeseran bit ke kiri	5
>>	Pergeseran bit ke kanan	5
&	Bitwise AND	8
^	Bitwise XOR (Exclusive OR)	9
	Bitwise OR	10
~	Bitwise NOT	1

### Operator Pengerjaan Bitwise

Operator	Contoh	Ekuivalen dengan
<<=	A <<= 2	A = A << 2
>>=	A >>= 2	A = A >> 2
&=	A &= 0x1b	A = A & 0x1b
^=	A ^= 0x1b	A = A ^ 0x1b
=	A  = 0x1b	A = A   0x1b



## Format untuk Setiap Tipe Data

Untuk memasukan nilai data menggunakan Spesifikai *format* yaitu : "%  
*type*" dimana *type* bisa diganti dengan salah satu dari sbb:

Kode Format	Fungsi
%c	Membaca sebuah karakter
%s	Membaca nilai string
%d	Membaca nilai desimal integer
%i	Membaca nilai desimal integer
%x	Membaca nilai heksa desimal integer
%o	Membaca nilai oktal integer
%f	Membaca nilai pecahan
%e	Membaca nilai pecahan
%g	Membaca nilai pecahan
%h	Membaca nilai short integer desimal
[...]	Membaca karakter string yg diakhiri dengan karakter yg tidak ada didalam [...]
[^..]	Membaca karakter string yg diakhiri dengan karakter yg ada didalam [..]

Contoh Program:

```
#include <stdio.h>

main()
{
    int a, b, c = 10;
    float bil = 10.56;

    a = 5; b = 8;
    clrscr();
    printf ("Hasil tampilan numerik terformat %d, %d, %f \n", a, 78,
bil);
    printf ("Hasil tampilan string dan karakter : %s dan %c \n","coba-
coba",'A');
}
```

Hasil:

Hasil tampilan numerik terformat 5, 78, 10.56789

Hasil tampilan string dan karakter : coba-coba dan A

## FUNGSI INPUT DAN OUTPUT

3

### Obyektif :

6. Mengetahui fungsi input dan fungsi output bahasa C
  7. Dapat mengerti dan menggunakan fungsi input
  8. Dapat mengerti dan menggunakan fungsi output
- 

### Fungsi Input

Fungsi-fungsi pustaka yang digunakan untuk memasukkan data melalui keyboard, prototypenya ada di file judul `stdio.h` dan `conio.h`. Fungsi-fungsi yang menggunakan file judul `stdio.h` yaitu `gets()` dan `scanf()`. Sedangkan fungsi yang menggunakan file judul `conio.h`, yaitu `getche()`, `getchar()`, dan `getch()`.

### Fungsi Input Data Tidak Terformat

Untuk memasukkan nilai karakter tidak terformat digunakan `getchar()` `getch()`, `getche()` dan memasukan nilai string tidak terformat yaitu `gets`, tergantung dari karakteristik masing-masing.

#### ❖ `getchar()`

Sintak: : `int getchar(void):`

Fungsi:

- mengembalikan sebuah karakter (nilai ASCII) berikutnya dari buffer keyboard.
- Karakter ditampilkan di layar monitor
- Menunggu sampai ada ENTER
- Header file ada di `stdio.h`

#### ❖ `getch()`

Sintak: `int getch(void):`

## Fungsi

- mengembalikan satu karakter dari buffer keyboard
- karakter tidak ditampilkan di layar monitor (no echo)
- Tidak menunggu sampai ada ENTER
- Cocok untuk membuat password
- Header file ada di conio.h

### ❖ getch()

Sintak :        int getch(void)

Fungsi :

- mengembalikan satu karakter dari keyboard
- Karakter ditampilkan di layar (echo)
- Tidak menunggu sampai ada ENTER
- Header file ada di conio.h

Contoh:

```
#include <stdio.h>
#include<conio.h>
main()
{
    char A;
    printf("Masukan Nilai Sebuah Karakter?");
    A=getch();
    printf("\nNilai yang dimasukan adalah:%c\n",A);
}
```

### ❖ gets()

Sintak :        char \*gets(char \*buffer)

Fungsi:

- membaca string dari keyboard sampai ketemu new-line dan disimpan pada buffer.
- Kemudian new-line di replace dengan null character

- Mengembalikan nilai NULL jika ada error dan mengembalikan argument-nya (buffer) jika sukses.

## Fungsi Input Data Terformat

Untuk meg-input nilai data terformat digunakan perintah scanf(), Spesifikasi *format* adalah : "% type" dimana *type* bisa diganti dengan salah satu dari sbb:

Kode Format	Fungsi
%c	Membaca sebuah karakter
%s	Membaca nilai string
%d	Membaca nilai desimal integer
%i	Membaca nilai desimal integer
%x	Membaca nilai heksa desimal integer
%o	Membaca nilai oktal integer
%f	Membaca nilai pecahan
%e	Membaca nilai pecahan
%g	Membaca nilai pecahan
%h	Membaca nilai short integer desimal
[...]	Membaca karakter string yg diakhiri dengan karakter yg tidak ada didalam [...]
[^..]	Membaca karakter string yg diakhiri dengan karakter yg ada didalam [..]

Fungsi scanf mengembalikan tipe integer, dimana nilai nya menyatakan jumlah field yang sukses di assigned. Contoh:

```
int x,y,z,w;
x=scanf("%d %d %d",&y,&z,&w);
```

maka :

- Jika di input dari keyboard 3 buah nilai interger 6 7 8, maka nilai x = 3;

- Jika di input dari keyboard 4 buah nilai interger 6 7 8 9 maka nilai x = 3 (karena 3 nilai yg sukses di- assigned masing-masing ke variabel y, z dan w)

Karakter Space, tab, linefeed, carriage-return, formfeed, vertical-tab, dan newline disebut "*white-space characters*". Contoh :

```
char ss[40];
scanf("%s",ss);
```

Pada potongan program diatas, jika dimasukkan string "Selamat Pagi Pak" dari keyboard maka yg dimasukkan ke variabel ss hanya "Selamat" saja. Untuk mengambil string yang diakhiri karakter tertentu (misalnya ENTER), dengan scanf, menggunakan format [^\n]. Menjadi :

```
char ss[40];
scanf("%[^\n]",ss);
```

### **Fungsi Output**

Prototype dari fungsi-fungsi untuk menampilkan hasil terdapat pada file judul stdio.h bersifat standar yaitu putchar(), puts(), printf(), printf() dan conio.h bersifat tidak standar, dalam arti tidak semua kompiler C menyediakan yaitu clrscr(), gotoxy().

### **Fungsi Output Hasil Tidak Terformat**

Untuk menampilkan hasil tidak terformat digunakan putchar(), putch() dan puts(). Maksudnya tidak terformat adalah lebar dan bentuk tampilannya tidak dapat diatur.

#### **❖ putchar( )**

Sintak:        int putchar(int c)

Fungsi :

- untuk menampilkan karakter tidak terformat

- Menampilkan karakter ke layar monitor pada cursor, kemudian setelah ditampilkan cursor bergerak ke posisi berikutnya.
- Mengembalikan EOF jika error, dan mengembalikan karakter yang ditampilkan jika sukses
- Puchar adalah macro yang sama artinya dengan: `putc(c, stdout)`
- Header File : `stdio.h`

#### ❖ **putch( )**

Sintak : `int putch(int ch)`

Fungsi :

- menampilkan karakter ascii di `ch` di monitor tanpa memindahkan kursor ke posisi berikutnya
- Header file : `conio.h`
- Mengembalikan EOF jika error, dan mengembalikan karakter yang di tampilkan jika sukses.

#### ❖ **puts( )**

Sintak : `int puts(const char *str);`

Fungsi:

- Menampilkan string ke layar monitor dan memindahkan kursor ke baris baru.
- Header file: `stdio.h`
- Mengembalikan nilai non-negative jika sukses dan EOF jika ada error.

CONTOH :

```
puts("Selamat Datang");
```

```
puts("Di GUNDAR");
```

Tampilan di layar monitor :

Selamat Datang  
Di GUNDAR

Penempatan kursor

- Layar dapat dihapus dengan menggunakan fungsi: `clrscr()`;
- Kursor dapat dipindahkan ke posisi manapun di dalam layar monitor dengan menggunakan fungsi : `gotoxy(col,row)`; dimana col = kolom dan row = baris
- Sebagian dari baris, mulai posisi kursor hingga akhir baris (end of line), dapat dihapus dengan fungsi: `clreol()`;
- Function prototype untuk fungsi `gotoxy()`, `clrscr()`, `clreol()` pada bahasa C terdapat pada header file : `<conio.h>`

**Fungsi Output Hasil Terformat**

Sedangkan untuk hasil terformat digunakan perintah **printf** dengan spesifikai format sbb: `%[flags][width][.precision] type;`

Kode Format	Fungsi
<code>%c</code>	Menampilkan sebuah karakter
<code>%s</code>	Menampilkan nilai string
<code>%d</code>	Menampilkan nilai desimal integer
<code>%i</code>	Menampilkan nilai desimal integer
<code>%u</code>	Menampilkan nilai desimal integer tidak bertanda
<code>%x</code>	Menampilkan nilai heksa desimal integer
<code>%o</code>	Menampilkan nilai oktal integer
<code>%f</code>	Menampilkan nilai pecahan
<code>%e</code>	Menampilkan nilai pecahan dalam notasi scientific
<code>%g</code>	Sebagai pengganti ' <code>%f</code> ' atau ' <code>%e</code> ' tergantung mana yang terpendek
<code>%p</code>	Menampilkan suatu alamat memori untuk pointer



*width* : menentukan jumlah kolom yang disediakan

*precision* : menentukan jumlah angka dibelakang koma (untuk bilangan pecahan)

*flags* dapat diganti sbb:

none : right justify (rata kanan)

- : left justify (rata kiri)

+ : untuk bilangan dimulai dgn  
tanda – jika negatip atau +  
jika positif

CONTOH 1:

```
printf(“%6d”, 34);           ....34
printf(“%-6d”, 34);         34....
```

CONTOH 2 :

```
printf(“%10s”, “GUNDAR”);    ...GUNDAR
printf(“%-10s”, “GUNDAR”);   GUNDAR ...
printf(“%8.2f”, 3.14159 );    ....3.14
printf(“%-8.3f”, 3.14159 );   3.141...
printf(“%c\n”,65);           //akan ditampilkan A
printf(“%x\n”,'A');          // akan ditampilkan 41
printf(“%o\n”,65);           // akan ditampilkan 101
printf(“%+d\n”,34);          // akan ditampilkan +34
printf(“%+d\n”,-45);         // akan ditampilkan -45
printf(“%e\n”,3.14);         // akan ditampilkan 3.140000e+000
```

CONTOH 3:

```
#include <stdio.h> Laboratorium Sistem Informasi
int main(){
    char ss[]="Selamat Datang";
    printf("123456789012345678901234567890\n");
```

```

printf("%.10s di Gundar\n",ss);
printf("%10s di Gundar\n",ss);
printf("%-10s di Gundar\n",ss);
printf("%.20s di Gundar\n",ss);
printf("%20s di Gundar\n",ss);
printf("%-20s di Gundar\n",ss);
printf("%20.10s di Gundar\n",ss);
printf("%-20.10s di Gundar\n",ss);
return 0;
}

```

Output Program di atas sbb:

123456789012345678901234567890

Selamat Da di Gundar

Selamat Datang di Gundar

Selamat Datang di Gundar

Selamat Datang di Gundar

    Selamat Datang di Gundar

Selamat Datang    di Gundar

        Selamat Da di Gundar

Selamat Da        di G

## PENGAMBILAN KEPUTUSAN DALAM BAHASA C

4

### Obyektif :

9. Mengetahui macam-macam pengambilan keputusan dalam bahasa C
  10. Mengetahui dan mengerti tentang statemen kondisi IF
  11. Mengetahui dan mengerti tentang statemen kondisi Switch
- 

### Statement If

- a. Bentuk If tunggal sederhana

Sintaks :

```
if ( kondisi ) statement ;
```

Bentuk ini menunjukkan jika kondisi bernilai benar, maka statement yang mngikutinya akan di-eksekusi. Jika tidak maka statement selanjutnya yang akan diproses.

- b. Bentuk If tunggal blok statement

Sintaks :

```
if ( kondisi ) {  
    blok statement;  
}
```

Perbedaan dengan bentuk sebelumnya statement yang akan dilaksanakan ada dalam satu blok kurung kurawal.

### Statement If...Else

- a. Bentuk If..Else

sintaks :

```
if ( kondisi )
```

```
        statement1;
    else
        statement2;
```

Statement setelah kondisi atau statement sesudah else dapat berupa statement kosong, statement tunggal maupun blok statement. statement1 akan dijalankan jika kondisi benar, jika salah maka statement2 yang akan diproses.

contoh :

```
//Program menentukan ganjil atau genap
#include<stdio.h>

int main(){
    int Bilangan;
    char Lagi;
    printf("Mencari Bilangan Ganjil atau Genap\n\n");
    printf("Input Bilangan : ");
    scanf("%d", &Bilangan);
    if(Bilangan %2 == 1)
        printf("\n\nIni Bilangan Ganjil");
    else
        printf("\n\nIni Bilangan Genap");
    return 0; }
```

Output :      Mencari Bilangan Ganjil atau Genap

                  Input Bilangan : 15

                  Ini Bilangan Ganjil

b. Bentuk If..else if...else

Sintaks :

```
if ( kondisi 1)
    statement1;
else if ( kondisi 2 )
    statement2;
else if ( kondisi 3)
    statement3;
.
else
    statement default;
```

Proses akan mulai dari penyeleksian kondisi 1, jika benar maka statement yang mengikutinya akan dieksekusi, jika salah maka akan masuk proses seleksi kondisi 2, begitu seterusnya. Jika semua kondisi tidak ada yang terpenuhi, maka program akan menjalankan statement default.

contoh :

```
//Program Mencari Mutu Nilai
#include<stdio.h>

int main(){
    int Nilai; char Mutu;
    printf("Mencari Mutu Nilai\n\n");
    printf("Input Nilai Mahasiswa : ");scanf("%d", &Nilai);
    if (Nilai<50) Mutu = 'E';
        else if(Nilai<65) Mutu = 'D';
            else if(Nilai<75) Mutu ='C';
                else if (Nilai<85) Mutu ='B';
                    else Mutu = 'A';
```

```
printf("\n\nNilai Mahasiswa yang diinput = %d", Nilai);  
printf("\nMutu Nilai = %c", Mutu);  
return 0; }
```

Output :                    Mencari Mutu Nilai

Input Nilai Mahasiswa : 78

Nilai Mahasiswa yang diinput = 78

Mutu Nilai = B

c. Bentuk If bersarang ( nested if )

Sintaks :

```
if ( kondisi 1)  
    if ( kondisi 2)  
        .  
        .  
        if (kondisi n )  
            statement;  
        else  
            statement;  
        .  
        .  
    else  
        statement  
else statement;
```

Kondisi yang akan diseleksi pertama kali adalah kondisi yang paling luar (kondisi 1). Jika bernilai tidak benar maka statement setelah else yang terluar ( pasangan dari if yang bersangkutan ) yang akan diproses.

- d. Bentuk If dengan kondisi berupa variable

Contoh :

```
if ( D == 0 )
    printf ("Nilai D sama dengan Nol \n");
else
    printf ("Nilai D tidak sama dengan Nol \n");
```

- e. Bentuk If dengan kondisi Jamak

Beberapa kondisi dapat diseleksi sekaligus dalam statement if dengan menggunakan operator logika AND ( && ), OR ( || ), atau NOT ( ! )

### **Statement Switch**

- a. Statement Switch tunggal

Sintaks :

```
switch ( kondisi ) {
case konstanta1 :
    statement-statement;
    break;
case konstanta2 :
    statement-statement;
    break;
.
.
default :
    statement-statement;
}
```

Statement switch akan menyeleksi kondisi yang diberikan dan akan membandingkan hasilnya dengan konstanta-konstanta yang berada di

**case. Break** digunakan sebagai peloncat keluar blok switch, sedangkan **default** merupakan pilihan selain dari konstanta-konstanta yang ada pada case.

Apabila telah selesai memproses sebuah bagian dari case dan jika belum ditemukan statemen break, maka proses akan masuk ke bagian case berikutnya.

contoh :

```
//Program dengan switch Case
#include<stdio.h>

int main(){
    int Pilih;
    printf("----MENU BUAH----\n");
    printf("\n1. APEL");
    printf("\n2. MANGGA");
    printf("\n3. JERUK");
    printf("\n4. KELUAR");
    printf("\n\nPilihan Anda [1-4] : ");
    scanf("%d",&Pilih);
    switch(Pilih){
        case 1 : printf("\n\nANDA PILIH APEL"); break;
        case 2 : printf("\n\nANDA PILIH MANGGA"); break;
        case 3 : printf("\n\nANDA PILIH JERUK"); break;
        case 4 : exit(0);
        default : printf("\n\nANDA SALAH INPUT...");
    }
    return 0; }
```

b. Statement nested switch

Yaitu statement switch yang berada didalam switch lainnya.



Sintaks :

```
switch ( kondisi ) {  
  case konstanta 1 :  
    statement-statement ;  
    switch ( kondisi x ) {  
      case konstanta 1a :  
        statement-statement ;  
        break;  
      case konstanta 1b :  
        statement-statement ;  
        break;  
    }  
    break;  
  case konstanta 2 :  
    statement-statement;  
    break;  
}
```

## PERULANGAN DALAM BAHASA C

5

### Obyektif :

12. Mengetahui macam-macam perulangan dalam bahasa C
13. Mengetahui dan mengerti tentang statemen perulangan For
14. Mengetahui dan mengerti tentang statemen While dan Do While
15. Mengetahui dan mengerti tentang statemen Continue dan Go To

---

### PERULANGAN

#### a. Statement for

Sintaks :

```
for ( inisialisasi; terminasi; iterasi ) statement;
```

*inisialisasi* adalah pemberian nilai awal variable untuk perulangan, *terminasi* adalah pemberian nilai akhir atau batas perulangan, *iterasi* adalah perubahan variable kontrol (*counter*).

contoh :

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

```
main()
```

```
{
```

```
    int i,
```

```
    clrscr();
```

```
    for(i=0;i<5;i++)
```

```
        /*pengulangan diproses sebanyak 0 sampai 4, kenapa bisa  
        begitu...? */
```

```
    {
```

```
        printf("%d\n",i);
```

```
    }
```

```
}
```

output:

0

1

2

3

4

b. Statement while

Sintaks :

```
while (kondisi ) statement;
```

Statement dapat berupa statement kosong, statement tunggal maupun blok statement. Proses perulangan akan terus dilaksanakan jika kondisi dalam while masih bernilai benar.

Contoh :

```
#include <stdio.h>
main()
{
    int i=0;
    while(i<5)
    {
        printf("%d\n",i);
        /*hasil output contoh ini sama dengan contoh pada
for( )*/
        i++;
    }
}
```

c. Statement do...while

Sintaks :

```
do
    statement
while ( kondisi )
```

Sedikitnya statement akan diproses sebanyak 1 kali karena seleksi kondisi dilaksanakan diakhir statement.

Contoh:

```
#include <stdio.h>
main()
{
    int i=0;
    do
    {
        printf("%d\n",i);
        i++;
    }while (i < 5);
}
```

## STATEMENT COUNTINUE DAN GO TO

### Statement Countinue

Statement continue akan menyebabkan proses perulangan kembali ke awal perulangan dengan mengabaikan statement setelah statement continue.

contoh :

```
//Program dengan for & continue
```

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

```
int main(){
```

```
    int X;
```

```
    for (X=0; X<10; X++){
```

```
        if (X==5) continue;
```

```
        printf("%d ", X);
```

```
    }
```

```
    return 0;}
```

Output : 0 1 2 3 4 6 7 8 9

### **Statement GO TO**

Digunakan untuk melompat dari satu proses ke proses tertentu didalam program.

Sintaks :

```
goto label;
```

Proses lain yang ditunjuk sebagai lompatan akan ditulis

```
label
```

## ARRAY DAN DATA TERSTRUKTUR DALAM BAHASA C

6

### Obyektif :

- 16. Mengetahui dan mengerti tentang Array
  - 17. Mengetahui dan mengerti tentang Data Terstruktur
- 

### Array

Larik adalah kumpulan nilai-nilai data bertipe sama dalam urutan tertentu yang menggunakan sebuah nama yang sama. Nilai-nilai data di suatu larik disebut dengan elemen larik yang letak urutannya ditunjukkan oleh suatu *subscript* atau suatu *index* yang dimulai dengan index nol.

Larik dapat berdimensi satu (one dimensional array) yang mewakili suatu vektor, larik berdimensi dua (two dimensional array) mewakili bentuk suatu matrik atau tabel, larik berdimensi tiga (three dimensional array) mewakili suatu bentuk ruang atau berdimensi lebih dari tiga.

### Array Dimensi 1

Suatu larik dapat dideklarasikan dengan menyebutkan jumlah dari elemennya yang dituliskan diantara tanda '[' ]'. Contoh :

```
Int X[5];
```

Berarti variabel X bertipe integer dan merupakan larik dimensi satu.

Contoh larik berdimensi Satu :

```
#include <stdio.h>
main()
{
    float X[3] = {5,3,7}, Total = 0;
    int I;
    for(I=0;I<=2;I++) Total = Total + X[I];
    printf("Total = %f \n",Total);
}
```

Output :

Total = 15.000000

## Array Dimensi 2

Pendeklarasian larik dimensi dua :

int X[3][4]; → berarti akan membentuk matrik  
dengan ukuran 3 baris X 4 kolom

int X[2][3]={1,2,3,4,5,6} → matrik 2X3

atau larik tidak berukuran seperti :

int X[ ][4] = {1,2,3,4,5,6,7,8} → matrik 2X4

Contoh larik dimensi dua :

```
#include <stdio.h>
main()
{
    int I,J;
    float X[3][4] = { 12.34, 34.56, 56.78, 78.90,
                    23.45, 45.67, 67.89, 89.01,
                    34.56, 56.78, 78.90, 90.12};

    /*Menampilkan dalam bentuk matrik*/
    for(I=0;I<3;I++){
        for(J=0;J<4;J++){
            printf("%8.2f", X[I][J]);
            printf("\n");
        }
        printf("\n"); }
}
```

Output :

```
12.34  34.56  56.78  78.90
23.45  45.67  67.89  89.01
34.56  56.78  78.90  90.12
```

Larik String

Hubungan antara nilai larik string dengan nilai larik karakter

String	Character
Nilai string tunggal	Larik karakter dimensi satu
Larik string dimensi satu	Larik karakter dimensi dua
Larik string dimensi dua	Larik karakter dimensi tiga
Larik string dimensi N	Larik karakter dimensi N + 1



Contoh 1 :

```
#include <stdio.h>
main()
{
    int I,J;
    char Hari[7][10] = {"Minggu", "Senin", "Selasa", "Rabu", "Kamis",
    "Jum'at", "Sabtu"};
    for(I=0;I<7;I++)
    {
        for(J=0;J<10;J++)
            printf("%c", Hari[I][J]);
        printf("\n");
    }
}
```

Output :

```
Minggu
Senin
Selasa
Rabu
Kamis
Jum'at
Sabtu
```

Contoh 2 :

```
#include <stdio.h>
main()
{
    int I;
    char Hari[7][10] = {"Minggu", "Senin", "Selasa", "Rabu", "Kamis",
    "Jum'at", "Sabtu"};
    for(I=0;I<7;I++)
        printf("%s \n", Hari[I]);
}
```

Output :

Minggu

Senin

Selasa

Rabu

Kamis

Jum'at

Sabtu

Contoh 3 :

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

```
main()
```

```
{
```

```
    int I,J;
```

```
    char Nama[5][3][10] = {"Adit", "Bayu", "Coki", "Dhani", "Erwin",  
"Fenty", "Gunarto", "Henry", "Ibrahim", "Joko", "Kemal", "Lukman",  
"Meny", "Nony", "Onie"};
```

```
    for(I=0;I<5;I++)
```

```
    {    printf("Nama-nama dikelas %1d adalah : \n", I+1);
```

```
        for(J=0;J<3;J++)
```

```
            printf("%s \n", Nama[I][J]);
```

```
        }    }
```

Output :

Nama-nama dikelas 1 adalah :

Adit

Bayu

Coki

Nama-nama dikelas 2 adalah :

Dhani

Erwin

Fenty

Nama-nama dikelas 3 adalah :

Gunarto

Henry

Ibrahim

Nama-nama dikelas 4 adalah :

Joko

Kemal

Lukman

Nama-nama dikelas 5 adalah :

Meny

Nony

Onie

### **Data Terstruktur**

Record digunakan untuk mengklasifikasikan field-field untuk dijadikan satu rekaman data. Sintaks record dalam bahasa C ini adalah :

*Struct* merupakan suatu struktur data yang menggabungkan beberapa data dengan berbagai tipe data yang memiliki ukuran yang berbeda (heterogen) di kelompokkan dalam satu deklarasi unik dan saling berkaitan, dengan format sbb :

```
struct model_name {  
    type1 element1;  
    type2 element2;  
    type3 element3;  
    .  
    .  
} object_name;
```

dimana *model\_name* adalah nama untuk model tipe stukturanya dan parameter optional *object\_name* merupakan identifier yang valid untuk objek struktur. Diantara kurung kurawal { } berupa tipe dan sub-identifier yang mengacu ke elemen pembentuk struktur. Jika pendefinisian stuktur menyertakan parameter *model\_name* (optional), maka parameter tersebut akan menjadi nama tipe yang valid ekuivalen dengan struktur. Contoh :

```
struct products {
    char name [30];
    float price;
};

products apple;
products orange, melon;
```

Didefinisikan model struktur **products** dengan dua field : **name** dan **price**, dengan tipe yang berbeda. Kemudian tipe struktur tadi (**products**) digunakan untuk mendeklarasikan tiga objek : **apple**, **orange** dan **melon**.

Ketika dideklarasikan, **products** menjadi nama tipe yang valid seperti tipe dasar *int*, *char* atau *short* dan dapat mendeklarasikan objects (variables) dari tipe tersebut. Optional field yaitu *object\_name* dapat dituliskan pada akhir deklarasi struktur untuk secara langsung mendeklarasikan object dari tipe struktur. Contoh :

```
struct products {
    char name [30];
    float price;
} apple, orange, melon;
```

Sangat penting untuk membedakan antara structure **model**, dan structure *object*. *model* adalah *type*, dan *object* adalah *variable*. Kita dapat membuat banyak *objects* (variables) dari satu *model* (type).

*Struct* dapat dideklarasikan secara bertingkat, yaitu salah satu *field struct* bertipe *struct* lainnya (*nested Structure*) . Selain itu *struct* juga dapat digabungkan dengan array, *struct* yang field-nya berupa array atau array yang setiap elemennya berupa *structure*.

Structure sebagai parameter

Structure dapat dikirim ke *function* sebagai parameter, jika *structure* ini hanya merupakan data masukan, maka dapat dikirimkan dengan *call by value*, tetapi bila *structure* yang dikirimkan akan mengalami perubahan nilai maka pengiriman parameter harus dengan *call by reference* dengan mengirimkan *pointer to struct*. *Struct* juga dapat menjadi *return type* sebuah *function*.

```
struct <nama struct > {  
    <deklarasi data type>;  
} [<nama variabel struct>]
```

contoh:

```
struct record_saya  
{  
    char  nama[20];  
    char  alamat[30];  
    int   jumlah_anak;  
} data_rec[100];
```

atau

```

struct record_saya
{
    char  nama[20];
    char  alamat[30]; /* mendeklarasikan tipe record*/
    int   jumlah_anak;
};

```

```

struct record_saya data_rec[100];

/*mendeklarasikan variabel record sebanyak 100 data*/

```

```

#include <stdio.h>
struct record_saya
{
    char  nama[20];
    char  alamat[30]; /* mendeklarasikan tipe record*/
    int   jumlah_anak;
};

```

```

main()
{
    struct record_saya data_saya[100];
    int i,jumlah_data;
    clrscr();
    printf("Masukan jumlah data yang di input      :
");scanf("\n%d",&jumlah_data);
    for (i=0;i<jumlah_data;i++)
    {
        printf("\nInput Data Ke - %d",i+1);
    }
}

```

```

        printf("Nama Karyawan   :
");scanf("\n%s",data_saya[i].nama);
        printf("Alamat Karyawan : ");gets(data_saya[i].alamat);
        printf("Jumlah Anak     :                ");scanf("\n%d",
&data_saya[i].jumlah_anak);

    }
    clrscr();
    for (i=0;i<jumlah_data;i++)
    {
        printf("\nOutput Data Ke - %d",i+1);
        printf("Nama Karyawan   : %s",data_saya[i].nama);
        printf("Alamat Karyawan : %s",data_saya[i].alamat);
        printf("Jumlah Anak     : %d",data_saya[i].jumlah_anak);
    }
}

```

## FUNCTION DAN POINTER DALAM BAHASA C

7

### Obyektif :

18. Mengetahui dan mengerti tentang Function

19. Mengetahui dan mengerti tentang Pointer

---

### Function

*Function* adalah satu blok instruksi yang akan dieksekusi ketika dipanggil dari bagian lain dalam suatu program. Format penulisan *function*

:

*type name ( argument1, argument2, ...) statement;*

Dimana :

- *type*, adalah tipe data yang akan dikembalikan/dihasilkan oleh *function*.
- *name*, adalah nama yang memungkinkan kita memanggil function.
- *arguments* (dispesifikasikan sesuai kebutuhan). Setiap argumen terdiri dari tipe data diikuti identifier, seperti deklarasi variable (contoh, int x) dan berfungsi dalam function seperti variable lainnya. Juga dapat melakukan *passing parameters* ke function itu ketika dipanggil. Parameter yang berbeda dipisahkan dengan koma.
- *statement*, merupakan bagian badan suatu *function*. Dapat berupa instruksi tunggal maupun satu blok instruksi yang dituliskan diantara kurung kurawal {}.



Contoh *function* 1 :

```

// function example
#include <stdio.h>
int addition (int a, int b){
    int r;
    r=a+b;
    return (r);
}

int main ()
{
    int z;
    z = addition (5,3);
    printf("The result is %d", z);
    return 0;
}

```

### Output

Dari program diatas, ketika eksekusi akan dimulai dari fungsi main. main function memulai dengan deklarasi variabel z bertipe int. Setelah itu instruksi pemanggilan fungsi addition. Jika diperhatikan, ada kesamaan antara struktur pemanggilan dengan deklarasi fungsi itu sendiri, perhatikan contoh dibawah ini :

```

int addition (int a, int b)
      ↑           ↑
z = addition ( 5 , 3 );

```

Instruksi pemanggilan dalam fungsi main untuk fungsi addition, memberikan 2 nilai : 5 dan 3 mengacu ke parameter int a dan int b yang dideklarasikan untuk fungsi addition.

Saat fungsi dipanggil dari main, kontrol program beralih dari fungsi main ke fungsi addition. Nilai dari kedua parameter yang diberikan (5 dan 3) di-copy ke variable local ; int a dan int b.

Fungsi addition mendeklarasikan variable baru (int r;), kemudian ekspresi r=a+b;, yang berarti r merupakan hasil penjumlahan dari a dan b, dimana a dan b bernilai 5 dan 3 sehingga hasil akhirnya 8. perintah selanjutnya adalah :  
return (r);

Merupakan akhir dari fungsi addition, dan mengembalikan kontrol pada fungsi main. Statement return diikuti dengan variabel r (return (r);), sehingga nilai dari r yaitu 8 akan dikembalikan :

```
int addition (int a, int b)
  ↓ 8
z = addition ( 5 , 3 );
```

Dengan kata lain pemanggilan fungsi (addition (5,3)) adalah menggantikan dengan nilai yang akan dikembalikan (8).

Contoh *function 2* :

```
// function example
#include <stdio.h>
int subtraction (int a, int b){
    int r;
    r=a-b;
    return (r);
}

int main (){
    int x=5, y=3, z;
    z = subtraction (7,2);
    printf("The first result is %d\n", z);
    printf("The second result is %d\n", subtraction (7,2));
    printf("The third result is %d\n", subtraction (x,y));
    z= 4 + subtraction (x,y);
    printf("The fourth result is %d\n", z);
    return 0; }
```

Output :

The first result is 5

The second result is 5

The third result is 2

The fourth result is 6

Fungsi diatas melakukan pengurangan dan mengembalikan hasilnya. Jika diperhatikan dalam fungsi main, dapat dilihat beberapa cara pemanggilan fungsi yang berbeda.

Perhatikan penulisan pemanggilan *function*, format penulisan pada dasarnya sama.

Contoh 1 : `z = subtraction (7,2);`  
`printf("The first result is %d\n", z);`

Contoh 2 : `printf("The second result is %d\n", subtraction (7,2));`

Contoh 3 : `printf("The third result is %d\n", subtraction (x,y));`

Hal lain dari contoh diatas, parameter yang digunakan adalah variable, bukan konstanta. Contoh diatas memberikan nilai dari x dan y, yaitu 5 dan 3, hasilnya 2.

contoh 4 : `z = 4 + subtraction (x,y);`

Atau dapat dituliskan : `z = subtraction (x,y) + 4;`

Akan memberikan hasil akhir yang sama. Perhatikan, pada setiap akhir ekspresi selalu diberi tanda semicolon (;).

*Function* tanpa tipe (Kegunaan *void*)

Deklarasi fungsi akan selalu diawali dengan tipe dari fungsi, yang menyatakan tipe data apa yang akan dihasilkan dari fungsi tersebut. Jika tidak ada nilai yang akan dikembalikan, maka dapat digunakan tipe void.

Contoh *function 3* :

```
// void function example
#include <stdio.h>
void dummyfunction (void) {
    printf("I'm a function!");
}

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



Output :  
I'm a function!

Argument passed *by value* dan *by reference*.

Parameter yang diberikan ke fungsi masih merupakan *passed by value*. Berarti, ketika memanggil sebuah fungsi, yang diberikan ke fungsi adalah nilainya, tidak pernah men-spesifikasikan variabelnya. Sebagai Contoh, pemanggilan fungsi addition, menggunakan perintah berikut :

```
int x=5, y=3, z;
z = addition ( x , y );
```

Yang berarti memanggil fungsi addition dengan memberikan nilai dari x dan y, yaitu 5 dan 3, bukan variabelnya.

```
int addition (int a, int b)
z = addition (  5 ,  3 );
```

Tetapi, dapat juga memanipulasi dari dalam fungsi, nilai dari variable external. Untuk hal itu, digunakan argument *passed by reference*.

Contoh *function* 3 :

```
// passing parameters by reference
#include <stdio.h>

void duplicate (int& a, int& b, int& c) {
    a*=2; b*=2; c*=2;
}

int main () {
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    printf("x = %d, y = %d, z = %d", x, y, z);
    return 0; }
```

Output :

```
x=2, y=6, z=14
```

Perhatikan deklarasi *duplicate*, tipe pada setiap argumen diakhiri dengan tanda *ampersand* (&), yang menandakan bahwa variable tersebut biasanya akan *passed by reference* dari pada *by value*.

Ketika mengirimkan variable *by reference*, yang dikirimkan adalah variabelnya dan perubahan apapun yang dilakukan dalam fungsi akan berpengaruh pada variable diluarnya.

```
void duplicate (int& a,int& b,int& c)
                ↑x   ↑y   ↑z
duplicate ( x , y , z );
```

Atau dengan kata lain, parameter yang telah ditetapkan adalah a, b dan c dan parameter yang digunakan saat pemanggilan adalah x, y dan z, maka perubahan pada a akan mempengaruhi nilai x, begitupun pada b akan mempengaruhi y, dan c mempengaruhi z.

Itu sebabnya mengapa hasil output dari program diatas adalah nilai variable dalam main dikalikan 2. jika deklarasi fungsi tidak diakhiri dengan tanda ampersand (&), maka variable tidak akan *passed by reference*, sehingga hasilnya akan tetap nilai dari x, y dan z tanpa mengalami perubahan.

*Passing by reference* merupakan cara efektif yang memungkinkan sebuah fungsi mengembalikan lebih dari satu nilai.

Contoh *function 4* :

```
// more than one returning value
#include <stdio.h>

void prevnext (int x, int& prev, int& next){
    prev = x-1;
    next = x+1;
}

int main (){
    int x=100, y, z;
    prevnext (x, y, z);
    printf("Previous= %d", next = %d", y, z);
    return 0;}
```

Output :

Previous=99, Next=101

Contoh *function* 5 :

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

```
void exchange ( int *a, int *b ){  
    int temp;  
    temp = *a; *a = *b; *b = temp;  
    printf("From function exchange: ");  
    printf("a = %d, b = %d\n", *a, *b);  
    }
```

```
void main(){  
    int a = 5, b = 7;  
    printf("From main: a = %d, b = %d\n", a, b);  
    exchange(&a, &b);  
    printf("Back in main: ");  
    printf("a = %d, b = %d\n", a, b);          }  
}
```

Output :

From main: a = 5, b = 7

From function exchange: a = 7, b = 5

Back in main: a = 7, b = 5

### Recursivity Function

Rekursif merupakan kemampuan sebuah fungsi untuk memanggil dirinya sendiri. Sangat berguna untuk pengerjaan sorting atau perhitungan factorial. Contoh, format perhitungan factorial :

$$n! = n * (n-1) * (n-2) * (n-3) \dots * 1$$

Misalkan, 5! ( 5 faktorial), akan menjadi :

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$



Contoh *function* 6 :

```
// factorial calculator
#include <stdio.h>
long factorial (long a){
    if (a > 1) return (a * factorial (a-1));
    else
        return (1);
}

int main () {
    long l;
    printf("Type a number: ");scanf("%l", &l);
    printf("! %l = %l ", l, factorial(l));
    return 0;
}
```

Output :

```
Type a number: 9
!9 = 362880
```

Prototyping function.

Format :

```
type name ( argument_type1, argument_type2, ...);
```

Hampir sama dengan deklarasi fungsi pada umumnya, kecuali :

- Tidak ada *statement* fungsi yang biasanya dituliskan dalam kurung kurawal {}.
- Diakhiri dengan tanda semicolon (;).
- Dalam argumen dituliskan tipe argumen, bersifat optional.

Contoh *function* 7 :

```
// prototyping
```

```

#include <stdio.h>

void odd (int a);
void even (int a);

int main (){
    int i;
    do {
        printf("Type a number: (0 to exit)"); scanf("%d", &i);
        odd (i);
    } while (i!=0);
    return 0;
}

void odd (int a){
    if ((a%2)!=0) printf("Number is odd.\n");
    else even (a);
}

void even (int a){
    if ((a%2)==0) printf("Number is even.\n");
    else odd (a);
}

```

Output :

```

Type a number (0 to exit): 9
Number is odd.
Type a number (0 to exit): 6
Number is even.
Type a number (0 to exit): 1030
Number is even.

```

Type a number (0 to exit): 0

Number is even.

Contoh diatas tidak menjelaskan tentang efektivitas program tetapi bagaimana prototyping dilaksanakan. Perhatikan prototype dari fungsi odd dan even:

```
void odd (int a);
```

```
void even (int a);
```

Memungkinkan fungsi ini dipergunakan sebelum didefinisikan. Hal lainnya mengapa program diatas harus memiliki sedikitnya 1 fungsi prototype, karena fungsi dalam odd terdapat pemanggilan fungsi even dan dalam even terdapat pemanggilan fungsi odd. Jika tidak satupun dari fungsi tersebut dideklarasikan sebelumnya, maka akan terjadi error.

Array sebagai parameter

Adakalanya array diberikan kedalam fungsi sebagai parameter. Dalam C++ tidak memungkinkan untuk *pass by value* satu blok memory sebagai parameter kedalam suatu fungsi. Untuk menggunakan array sebagai parameter maka yang harus dilakukan saat pendeklarasian fungsi adalah spesifikasi tipe array pada argumen, Contoh :

```
void procedure (int arg[])
```

Contoh *function 8* :

```

// arrays as parameters
#include <iostream.h>
void printarray(int arg[], int length)
{
    for (int n=0; n<length; n++)
        cout << arg[n] << " ";
        cout << "\n";
}

int main ()
{
    int firstarray[] = {5, 10, 15};
    int secondarray[] = {2, 4, 6, 8, 10};
    printarray (firstarray,3);
    printarray (secondarray,5);
    return 0;
}

```

Output :

```

5 10 15
2 4 6 8 10

```

Dari contoh diatas, instruksi (int arg[]) menjelaskan bahwa semua array bertipe int, berapapun panjangnya. oleh sebab itu dideklarasikan parameter kedua dengan sifat yang sama seperti parameter pertama.

## POINTER

### 1. Konsep Pengalamatan Langsung

```
char nilai1;
```

```
char nilai2;
```

```
nilai1=5;
```

```
nilai2=nilai1;
```

Contoh di atas merupakan pemberian nilai dari suatu variabel ke variabel lain secara langsung atau istilah lainnya adalah memberikan nilai pada suatu alamat variabel secara langsung. Variabel nilai1 pada mulanya diberikan nilai 5, kemudian nilai2 secara langsung diberikan nilai yang sama dengan variabel nilai1.

```
char nilai1=5;
```

```
char nilai2;
```

Alamat Memori

```
-----  
SEG : OFF      VARIABEL  DATA  
-----  
777F : 0000    nilai1  5  
777F : 0001    nilai2  0  
...   : ...  
-----
```

```
nilai2=nilai1;
```

```
-----  
SEG : OFF      VARIABEL  DATA  
-----
```

```
777F : 0000      nilai1  5
```

```
777F : 0001      nilai2  5
```

```
...      : ...  
-----
```

Tabel variabel memori

```
-----  
&nilai1      nilai1  &nilai2      nilai2  
-----
```

```
777F:0000  2      777F:0001  2  
-----
```

tabel di atas menunjukkan gambaran bagaimana pemberian nilai tersebut berlangsung di dalam RAM. SEGMENT dan OFFSET merupakan bentuk pengalamatan memori untuk memproses data yang dilakukan dalam sistem operasi mode REAL (real mode), contohnya DOS.

Variabel nilai1 kita anggap beralamat memori di 777F:0000 sedangkan nilai2 beralamat 777F:0001. Kedua variabel ini mempunyai tipe yang sama, yaitu char (1 byte) dan tiap-tiap alamat memori menyimpan data dalam 1 byte.

## 2. Konsep Pengalamatan Tidak Langsung

Pointer merupakan tipe variabel data yang berisi alamat dari variabel lain .

```
char nilai1=2;
char *ptr_nilai;
```

#### Alamat Memori

```
-----
SEG  : OFF      VARIABEL  DATA
-----
777F : 0000      nilai1  2
777F : 0001 *0)  ptr_nilai  0
777F : 0002 *1) ptr_nilai  0
777F : 0003 *2) ptr_nilai  0
777F : 0004 *3) ptr_nilai  0
-----
```

```
ptr_nilai=&nilai1; /* & = referensi alamat memori variabel
```

```
-----
SEG  : OFF      VARIABEL  DATA
-----
777F : 0000      nilai1  2
777F : 0001 *0)  ptr_nilai  00
777F : 0002 *1) ptr_nilai  00
777F : 0003 *2) ptr_nilai  7F
777F : 0004 *3) ptr_nilai  77
-----
```

#### Tabel Variabel memori

```
-----
&nilai1      nilai1  &ptr_nilai  ptr_nilai  *ptr_nilai
-----
777F:0000  2      777F:0001  777F:0000  2
-----
```

```
nilai1=5;
```

Tabel Variabel memori

```
-----  
&nilai1      nilai1  &ptr_nilai  ptr_nilai   *ptr_nilai  
-----  
777F:0000   5           777F:0001  777F:0000  5  
-----
```

contoh di atas menunjukkan bahwa menggunakan tipe pointer pada variabel, menggunakan tanda (\*) di depan nama variabelnya. Pada awalnya variabel nilai1 diberikan nilai 2 sedangkan variabel pointer ptr\_nilai berisi alamat yang belum jelas dan mempunyai panjang data sebanyak 4 byte untuk menyimpan suatu alamat.

Pada saat ptr\_nilai direferensikan ke nilai1 (ptr\_nilai=&nilai1;) maka ptr\_nilai kini telah menunjuk ke variabel nilai1. Jika variabel nilai1 mengalami perubahan maka ptr\_nilai juga akan mengalami perubahan.

Untuk dapat mengambil nilai/isi yang ditunjuk oleh ptr\_nilai gunakan (\*) : \*ptr\_nilai;



```

#include <stdio.h>
#include <conio.h>
main()
{
    char *ptr_nilai;
    unsigned char cetak;
    int nilai1;
    clrscr();
    printf("Masukan sebuah nilai : ");scanf("\n%d",&nilai1);
    printf("Nilai HexaDesimal   : %X H\n",nilai1);
    ptr_nilai=(char*)&nilai1;

    cetak=0;
    cetak=*((char*)ptr_nilai+0);
    printf("Nilai Byte Terendah   : %X H\n",cetak);
    cetak=0;
    cetak=*(ptr_nilai+1);
    printf("Nilai Byte Tertinggi   : %X H\n\n",cetak);

    (ptr_nilai+0)+=5; /*penambahan 5 pada byte terendahnya*/
    printf("Nilai Desimal+5       : %d\n",nilai1);
    printf("Nilai HexaDesimal+5   : %X H\n",nilai1);
    cetak=0;
    cetak=*(ptr_nilai+0);
    printf("Nilai Byte Terendah   : %X H\n",cetak);
    cetak=0;
    cetak=*(ptr_nilai+1);
    printf("Nilai Byte Tertinggi   : %X H\n",cetak);
    *(ptr_nilai+1)+=5;

}

```

OutPut:

Masukan sebuah nilai : 6374

Nilai HexaDesimal : 18E6 H

Nilai Byte Terendah : E6 H

Nilai Byte Tertinggi : 18 H

Nilai Desimal+5 : 6379

Nilai HexaDesimal+5 : 18EB H

Nilai Byte Terendah : EB H

Nilai Byte Tertinggi : 18 H