

Pengenalan bahasa C

1. Pendahuluan

Bahasa C adalah bahasa pemrograman yang dapat dikatakan berada di antara bahasa beraras rendah dan beraras tinggi. Bahasa beraras rendah artinya bahasa yang berorientasi pada mesin dan beraras tinggi berorientasi pada manusia. Bahasa beraras rendah, misalnya bahasa assembler, bahasa ini ditulis dengan sandi yang dimengerti oleh mesin saja, oelha karena itu hanya digunakan bagi yang memprogram mikroprosesor. Bahasa beraras rendah merupakan bahasa yang membutuhkan kecermatan yang teliti bagi pemrogram karena perintahnya harus rinci, ditambah lagi masing-masing pabrik mempunyai sandi perintah sendiri. Bahasa tinggi relatif mudah digunakan, karena ditulis dengan bahasa manusia sehingga mudah dimengerti dan tidak tergantung mesinnya. Bahasa beraras tinggi biasanya digunakan pada komputer.

Pencipta bahasa C adalah Brian W. Kernighan dan Denis M. Ritchi, sekitar tahun 1972. Penulisan program dalam bahasa C dilakukan dengan membagi dalam blok-blok, sehingga bahasa C disebut dengan bahasa terstruktur. Bahasa C dapat digunakan di berbagai mesin dengan mudah, mulai dari PC sampai dengan mainframe, dengan berbagai sistem operasi misalnya DOS, UNIX, VMS dan lain-lain.

2. Penulisan Program Bahasa C

Program Bahasa C tidak mengenal aturan penulisan di kolom tertentu, jadi bisa dimulai dari kolom manapun. Namun demikian, untuk mempermudah pembacaan program dan untuk keperluan dokumentasi, sebaiknya penulisan bahasa C diatur sedemikian rupa sehingga mudah dan enak dibaca.

Berikut contoh penulisan Program Bahasa C:

```
#include <at89c51.h>
main ()
{
.....
.....
}
```

Program dalam bahasa C selalu berbentuk fungsi seperti ditunjukkan dalam **main ()**. Program yang dijalankan berada di dalam tubuh program yang dimulai dengan tanda kurung buka { dan diakhiri dengan tanda kurung tutup }. Semua yang tertulis di dalam tubuh program ini disebut dengan blok.

Tanda () digunakan untuk mengapit **argumen** suatu fungsi. Argumen adalah suatu nilai yang akan digunakan dalam fungsi tersebut. Dalam fungsi **main** diatas tidak ada argumen, sehingga tak ada data dalam (). Dalam tubuh fungsi antara tanda { dan tanda } ada sejumlah pernyataan yang merupakan perintah yang harus dikerjakan oleh prosesor. Setiap pernyataan diakhiri dengan tanda titik koma ;

Baris pertama **#include <...>** bukanlah pernyataan, sehingga tak diakhiri dengan tanda titik koma (;). Baris tersebut meminta kompiler untuk menyertakan file yang namanya ada di antara tanda <...> dalam proses kompilasi. File-file ini

(ber-ekstensi .h) berisi deklarasi fungsi ataupun variable. File ini disebut **header**. File ini digunakan semacam perpustakaan bagi pernyataan yang ada di tubuh program.

3. Tipe Data

Tipe data merupakan bagian program yang paling penting karena tipe data mempengaruhi setiap instruksi yang akan dilaksanakan oleh computer. Misalnya saja 5 dibagi 2 bisa saja menghasilkan hasil yang berbeda tergantung tipe datanya. Jika 5 dan 2 bertipe integer maka akan menghasilkan nilai 2, namun jika keduanya bertipe float maka akan menghasilkan nilai 2.5000000. Pemilihan tipe data yang tepat akan membuat proses operasi data menjadi lebih efisien dan efektif.

Tabel 1 Bentuk Tipe data:

No	Tipe Data	Ukuran	Range (Jangkauan)	Format	Keterangan
1	Char	1 byte	-128 s/d 127	%c	Karakter
2	Int	2 byte	-32768 s/d 32767	%i, %d	Bilangan bulat
3	Float	4 byte	-3.4E-38 s/d 3.4E+38	%f	Bilangan pecahan
4	Double	8 byte	1.7E-308 s/d 1.7E+308	% f	Pecahan presisi ganda
5	Void	0 byte	-	-	Tidak bertipe

4. Konstanta

Konstanta merupakan suatu nilai yang tidak dapat diubah selama proses program berlangsung. Konstanta nilainya selalu tetap. Konstanta harus didefinisikan terlebih dahulu di awal program. Konstanta dapat bernilai integer, pecahan, karakter dan string.

Contoh konstanta : 50; 13; 3.14; 4.50005; 'A'; 'Bahasa C'.

5. Variable

Variabel adalah suatu pengenalan (identifier) yang digunakan untuk mewakili suatu nilai tertentu di dalam proses program. Berbeda dengan konstanta yang nilainya selalu tetap, nilai dari suatu variabel bisa diubah-ubah sesuai kebutuhan. Nama dari suatu variabel dapat ditentukan sendiri oleh pemrogram dengan aturan sebagai berikut :

- Terdiri dari gabungan huruf dan angka dengan karakter pertama harus berupa huruf. Bahasa C bersifat case-sensitive artinya huruf besar dan kecil dianggap berbeda.
- Tidak boleh mengandung spasi.
- Tidak boleh mengandung simbol-simbol khusus, kecuali garis bawah (underscore). Yang termasuk simbol khusus yang tidak diperbolehkan antara lain : \$, ?, %, #, !, &, *, (,), -, +, = dsb
- Panjangnya bebas, tetapi hanya 32 karakter pertama yang terpakai.

6. Deklarasi

Deklarasi diperlukan bila kita akan menggunakan pengenalan (identifier) dalam program. Identifier dapat berupa variabel, konstanta dan fungsi.

6.1. Deklarasi Variabel

Bentuk umum pendeklarasian suatu variabel adalah :

Nama_tipe nama_variabel;

Contoh :

`int x; // Deklarasi x bertipe integer`

```

char y, huruf, nim[10]; // Deklarasi variable bertipe char

float nilai; // Deklarasi variable bertipe float

double beta; // Deklarasi variable bertipe double

int array[5][4]; // Deklarasi array bertipe integer

char *p; // Deklarasi pointer p bertipe char

```

6.2. Deklarasi Konstanta

Dalam bahasa C konstanta dideklarasikan menggunakan preprocessor #define.

Contohnya :

```

#define PHI 3.14                #define nim "0111500382"

#define nama "Sri Widhiyanti"

```

6.3. Deklarasi Fungsi

Fungsi merupakan bagian yang terpisah dari program dan dapat diaktifkan atau dipanggil di manapun di dalam program. Fungsi dalam bahasa C ada yang sudah disediakan sebagai fungsi pustaka seperti printf(), scanf(), getch() dan untuk menggunakannya tidak perlu dideklarasikan.

Fungsi yang perlu dideklarasikan terlebih dahulu adalah fungsi yang dibuat oleh programmer. Bentuk umum deklarasi sebuah fungsi adalah :

```
Tipe_fungsi nama_fungsi(parameter_fungsi);
```

Contohnya :

```

float luas_lingkaran(int jari);

void tampil();

int tambah(int x, int y);

```

7. Operator

2.7.1. Operator Penugasan

Operator Penugasan (*Assignment operator*) dalam bahasa C berupa tanda sama dengan (“=”).

2.7.2. Operator Aritmatika

Bahasa C menyediakan lima operator aritmatika, yaitu :

- `.*` : untuk perkalian
- `/` : untuk pembagian
- `%` : untuk sisa pembagian (modulus)
- `+` : untuk penjumlahan
- `-` : untuk pengurangan

2.7.2.1. Perkalian

```
//Program Perkalian
#include <at89x51.h>
void main()
{
    int bil1,bil2;
    bil1=4;
    bil2=2;
    P1=bil1*bil2;      //bil1    dikali    bil2    hasilnya
    dikeluarkan ke P1
}
```

2.7.2.2. Pembagian

```
//Program Pembagian
#include <at89x51.h>
void main()
{
    int bil1,bil2;
```

```

    bil1=10;
    bil2=2;
    P1=bil1/bil2;    //hasil pembagian dikeluarkan ke P1
}

```

2.7.2.3. Modulus

```

//Program modulus
#include <at89x51.h>
void main()
{
    int bil1,bil2;
    bil1=13;
    bil2=2;
    P1=bil1%bil2;    //hasil modulus dikeluarkan ke P1
}

```

2.7.2.4. Penjumlahan

```

//program Penjumlahan
#include <at89x51.h>    //deklarasi register at89c51
void main()
{
    char bil1,bil2;    //deklarasi variabel bil1 dan bil2
    bil1=0x30;    //bil1=0x30    dan    bil2=0x20    (bentuk
heksadesimal)
    bil2=0x20;
    P1=bil1+bil2;    //hasil penjumlahan dikeluarkan ke
Port1
}

```

2.7.2.5. Pengurangan

```

//Program Pengurangan
#include <at89x51.h>
void main()
{
    int bil1,bil2;
    bil1=0x30;

```

```

    bil2=0x20;
    P1=bil1-bil2;    // hasil pengurangan dikeluarkan ke
P1.
}

```

2.7.3. Operator Hubungan (Perbandingan)

Operator hubungan digunakan untuk membandingkan hubungan antara dua buah operand /sebuah nilai atau variable. Operasi majemuk seperti pada tabel dibawah ini:

Tabel 2.2 Operator Hubungan

Operator	Arti	Contoh	
<	Kurang dari	X<Y	Apakah X kurang dari Y
<=	Kurang dari sama dengan	X<=Y	Apakah X Kurang dari sama dengan Y
>	Lebih dari	X>Y	Apakah X Lebih dari Y
>=	Lebih dari sama dengan	X==Y	Apakah X Lebih dari sama dengan Y
==	Sama dengan	X==Y	Apakah X Sama dengan Y
!=	Tidak sama dengan	X!= Y	Apakah X Tidak sama dengan Y

2.7.4. Operator Logika

Jika operator hubungan membandingkan hubungan antara dua buah operand, maka operator logika digunakan untuk membandingkan logika hasil dari operator-operator hubungan.

Operator logika ada tiga macam, yaitu :

- && : Logika AND (DAN)
- || : Logika OR (ATAU)

- **! : Logika NOT (INGKARAN)**

Operasi AND akan bernilai benar jika dua ekspresi bernilai benar. Operasi OR akan bernilai benar jika dan hanya jika salah satu ekspresinya bernilai benar. Sedangkan operasi NOT menghasilkan nilai benar jika ekspresinya bernilai salah, dan akan bernilai salah jika ekspresinya bernilai benar.

```
//Program Operator Logika
#include <at89x51.h>
void main(void)
{
char in1;
char in2;
    in1=P2;
    in2=P3;
    if((in1==0xf0) && (in2==0x40))
        {P1 = 0x2A;}
}
```

2.7.5. Operator Bitwise (Manipulasi per bit)

Operator bitwise digunakan untuk memanipulasi bit-bit dari nilai data yang ada di memori.

Operator bitwise dalam bahasa C di SDCC adalah sebagai berikut :

- **<< : Pergeseran bit ke kiri**
- **>> : Pergeseran bit ke kanan**
- **& : Bitwise AND**
- **^ : Bitwise XOR (exclusive OR)**
- **| : Bitwise OR**
- **~ : Bitwise NOT**
- **Pertukaran Nibble dan Byte**
- **Mengambil Bit yang paling Berbobot**

2.7.5.1. Operasi Geser Kiri (<<)

Operasi geser kiri merupakan operasi yang akan menggeser bit-bit ke kiri sehingga bit 0 akan berpindah ke bit 1 kemudian bit 1 akan berpindah ke bit 2 dan seterusnya. Operasi geser kiri membutuhkan dua buah operan disebelah kiri tanda << merupakan nilai yang akan digeser sedangkan disebelah kanannya merupakan jumlah bit penggeseran.

Contohnya :

```
Datanya = 0x03 << 2 ; // 0x03 digeser ke kiri 2 bit hasilnya ditampung di datanya
a <<= 1      // Isi variabel A digeser ke kiri 1 bit hasilnya
              // kembali disimpan di A
```

```
//Program Operasi Geser Kiri
#include <at89x51.h>
void main()
{
    char a, led;
    led=0x01;
    for (a=0;a<8;a++) //melakukan loop sebanyak 8 kali
    {
        P1=led;      //variabel lampu dikeluarkan ke P1
        led=led <<1; //variabel lampu digeser kiri 1 bit
    }
}
```

2.7.5.2. Operasi Geser Kanan(>>)

Operasi geser kanan merupakan operasi yang akan menggeser bit-bit ke kanan sehingga bit 7 akan berpindah ke bit 6 kemudian bit 6 akan berpindah ke bit 5 dan seterusnya. Operasi geser kanan membutuhkan dua buah operan disebelah kanan tanda >> merupakan nilai yang akan digeser sedangkan disebelah kanannya merupakan jumlah bit penggeseran.

Contohnya :

```
Datanya = 0x03 >> 2 ; // 0x03 digeser ke kanan 2 bit hasilnya ditampung di datanya
a >>= 1      // Isi variabel A digeser ke kanan 1 bit hasilnya
              // kembali disimpan di A
```

```
//Program Operasi Geser Kanan
#include <at89x51.h>
void main()
{
    char a, led;
    led=0x80;          //bit ke-7 berlogika 1
    for (a=0;a<8;a++)  //diulang sebanyak 8 kali, hasil
    akhirnya= 0xFF
    {
        P1=led;
        led=led >>1;  //variabel lampu digeser kanan 1 bit
    }
}
```

2.7.5.3. Operasi Bitwise AND (&)

Operasi bitwise AND akan melakukan operasi AND pada masing-masing bit, sehingga bit 0 akan dioperasikan dengan bit 0 dan bit 1 dan seterusnya.

Contohnya :

Hasil = 0x03 & 0x31;	Operasinya	0x03 = 00000011 0x31 = 00110001
	Hasil	0x01 = 00000001

```
//Program 5.17
#include <at89x51.h>
void main(void)
{
    char a=0x03;
    char b=0x31;
    P1= a & b ;
}
```

2.7.5.4. Operasi Bitwise OR (|)

Operasi bitwise OR akan melakukan operasi OR pada masing-masing bit, sehingga bit 0 akan dioperasikan dengan bit 0 dan bit 1 dan seterusnya.

Contohnya :

Hasil = 0x05 0x31;	Operasinya	0x01 = 00000001 0x31 = 00110001
	Hasil	0x01 = 00110001

```
//Program 5.18
#include <at89x51.h>
void main(void)
{
    char a=0x01;
    char b=0x31;
    P1=a | b ;
}
```

2.7.5.5. Operasi Bitwise XOR(^)

Operasi bitwise XOR akan melakukan operasi XOR pada masing-masing bit, sehingga bit 0 akan dioperasikan dengan bit 0 dan bit 1 dan seterusnya.

Contohnya :

Hasil = 0x02 ^ 0xFA;	Operasinya	0x02 = 00000010 0xFA = 11111010
	Hasil	<hr/> 0x01 = 11111000

```
// Program Operasi Bitwise XOR
#include <at89x51.h>

void main(void)
{
    char a=0x02;
    char b=0xFA;
    P1=a ^ b ;
}
```

2.7.5.6. Operasi Bitwise NOT(~)

Operasi bitwise XOR akan melakukan operasi XOR pada masing-masing bit, sehingga bit 0 akan dioperasikan dengan bit 0 dan bit 1 dan seterusnya.

Contohnya :	Hasil = ~ 0x31;	0x31 = 00110001
	Hasil	~0x31 = 11001110

```
//Program Operasi Bitwise NOT
#include <at89x51.h>
void main(void)
```

```
{
    char a= 0x31;
    P1= ~a;
}
```

2.7.5.7. Pertukaran Nibble dan Byte

Pertukaran nibble dalam bahasa C dikenali SDCC dengan bentuk pernyataan sebagai berikut ini:

```
volatile unsigned char i;
i = (( i << 4) | ( i >> 4)); //pertukaran nibble
```

Dan pernyataan sebagai berikut ini sebagai pertukaran byte:

```
volatile unsigned char j;
j = (( j << 8) | ( j >> 8)); //pertukaran byte
```

```
//Program Nibble dan Byte
#include <at89x51.h>

union kint          //tipe data int agar dapat diambil
per byte
{
    unsigned char a[2];
    unsigned int b;
};

void main()
{
    union kint tmp;
    volatile unsigned char i=0x37;
    volatile unsigned int j=0x9973;
    P1=i;
    tmp.b=j;
    P3= tmp.a[1];
    P2=tmp.a[0];
    i= ((i<<4) | (i>>4)); //pertukaran nibble
    j= ((j<<8) | (j>>8)); //pertukaran byte
    P1=i;                //I dikeluarkan ke Port 1
    tmp.b=j;
    P2=tmp.a[0]; //byte rendah dari j dikeluarkan ke
Port 2
    P3=tmp.a[1]; //byte tinggi dari j dikeluarkan ke
Port 3
}
```

2.7.5.8. Mengambil Bit yang Paling Berbobot

Untuk mendapatkan bit yang paling berbobot (MSB) untuk tipe long, short, int, dan char maka dapat dilakukan dengan pertanyaan berikut:

```

Volatile unsigned char gint;
Unsigned char hob;
Hop = (gint >> 7) & 1    // mengambil MSB

```

```

//Program Berbobot
#include <at89x51.h>
void main()
{
    volatile unsigned char gint=0xaa;
    volatile unsigned char hob;
    unsigned a;
    for(a=0;a<8;a++)          //diulang 8 kali
    {
        hob=(gint>>7) &1;      //
        Pl=hob;
        gint=((gint<<1)|(gint>>7));
    }
}

```

2.7.6. Operator Unary

Operator Unary merupakan operator yang hanya membutuhkan satu operand saja.

Dalam bahasa C terdapat beberapa operator unary, yaitu :

Tabel 2.3 Operasi Unary

Operator	Arti/Maksud	Letak	Contoh	Equivalen
-	Unary minus	Sebelum operator	A + -B * C	A + (-B) * C
++	Peningkatan dengan penambahan nilai 1	Sebelum dan sesudah	A++	A = A + 1
--	Penurunan dengan pengurangan nilai 1	Sebelum dan sesudah	A--	A = A - 1
sizeof	Ukuran dari operand dalam byte	Sebelum	sizeof(I)	-
!	Unary NOT	Sebelum	!A	-
~	Bitwise NOT	Sebelum	~A	-
&	Menghasilkan alamat memori operand	Sebelum	&A	-
*	Menghasilkan nilai dari pointer	Sebelum	*A	-

Contohnya :

```
n = 0
Jum = 2 * ++n;
Jum = 2 * n++;
```

```
//Program Operator Unary
#include <at89x51.h>
void main(void)
{
    int a;
    for(a=0;a<20;a++) //selama a < 20, maka a dinaikkan 1
        P1=a;
}
```

2.7.7. Operator Majemuk

Operator majemuk terdiri dari dua operator yang digunakan untuk menyingkat penulisan. Operasi majemuk seperti pada tabel 2.3 dibawah ini

Tabel 2.4 Operasi majemuk

Operator	Contoh	Kependekan dari
+=	Counter +=1;	Counter = counter + 1
-=	Counter -=1	Counter = counter - 1
*=	Counter *=1	Counter = counter * 1
/=	Counter /=1	Counter = counter / 1
%=	Counter %=1	Counter = counter % 1
<<=	Counter <<=1	Counter = counter << 1
>>=	Counter >>=1	Counter = counter >> 1
&=	Counter &=1	Counter = counter & 1
=	Counter =1	Counter = counter 1
^=	Counter ^=1	Counter = counter ^ 1
~=	Counter ~=1	Counter = counter ~ 1

8. Komentar Program

Komentar program hanya diperlukan untuk memudahkan pembacaan dan pemahaman suatu program (untuk keperluan dokumentasi program). Dengan kata lain, komentar program hanya merupakan keterangan atau penjelasan program. Untuk memberikan komentar atau penjelasan dalam bahasa C digunakan pembatas `/*` dan `*/` atau menggunakan tanda `//` untuk komentar yang hanya terdiri dari satu baris. Komentar program tidak akan ikut diproses dalam program (akan diabaikan).

Contoh pertama :

```
// program ini dibuat oleh ....
```

Dibelakang tanda `//` tak akan diproses dalam kompilasi. Tanda ini hanya untuk satu baris kalimat.

Contoh kedua :

```
/* program untuk memutar motor DC atau  
motor stepper */
```

Bentuk ini berguna kalau pernyataannya berupa kalimat yang panjang sampai beberapa baris.

9. Penyeleksian Kondisi

Penyeleksian kondisi digunakan untuk mengarahkan perjalanan suatu proses. Penyeleksian kondisi dapat diibaratkan sebagai katup atau kran yang mengatur jalannya air. Bila katup terbuka maka air akan mengalir dan sebaliknya bila katup tertutup air tidak akan mengalir atau akan mengalir melalui tempat lain.

Fungsi penyeleksian kondisi penting artinya dalam penyusunan bahasa C, terutama untuk program yang kompleks.

2.9.1. STRUKTUR KONDISI “IF...”

Struktur if dibentuk dari pernyataan if dan sering digunakan untuk menyeleksi suatu kondisi tunggal. Bila proses yang diseleksi terpenuhi atau bernilai benar, maka pernyataan yang ada di dalam blok if akan diproses dan dikerjakan.

Bentuk umum struktur kondisi if adalah :

```
if(kondisi)
    pernyataan;
```

```
//Program IF
#include <at89x51.h>
void main(void)
{
char inpl;
inpl=P2;
if(inpl==0x40)
    {P1 = 0x20;}
}
```

2.9.2. STRUKTUR KONDISI “IF.....ELSE...”

Dalam struktur kondisi if.....else minimal terdapat dua pernyataan. Jika kondisi yang diperiksa bernilai benar atau terpenuhi maka pernyataan pertama yang dilaksanakan dan jika kondisi yang diperiksa bernilai salah maka pernyataan yang kedua yang dilaksanakan. Bentuk umumnya adalah sebagai berikut :

```
if(kondisi)
    pernyataan-1
else
    pernyataan-2
```

Contoh

```
IF
if (angka = fo)      /* bila angka sama dengan fo */
{                    /* kerjakan berikut ini */
```

```

        for (k = 0; k<4 ; k++)
        {
            i=tabel1(k);
            PORTA = i;           // pernyataan dalam blok ini bisa kosong
            tunda50(100);        // berarti tidak ada yang dikerjakan
        }
    }
    else                          //bila tidak sama kerjakan berikut ini
    {
        for (k = 0; k<4 ; k++)
        {
            i=tabel2(k);       // pernyataan dalam blok ini bisa kosong
            PORTA = i;         // berarti tidak ada yang dikerjakan
            tunda50(100);
        }
    }
}

```

```

//Program IF.....ELSE
#include <at89x51.h>
void main(void)
{
    char inpl;
    inpl=P2;
    if(inpl==0x01)    //jika P2 = 0x01 P1=0x20, selain
itu P2=0x80;
        {P1 = 0x20;}
    else
        {P1=0x80;}
}

```

2.9.3. STRUKTUR KONDISI “SWITCH...CASE... DEFAULT...”

Struktur kondisi switch....case....default digunakan untuk penyeleksian kondisi dengan kemungkinan yang terjadi cukup banyak. Struktur ini akan melaksanakan salah satu dari beberapa pernyataan ‘case’ tergantung nilai kondisi yang ada di dalam switch. Selanjutnya proses diteruskan hingga ditemukan pernyataan ‘break’. Jika tidak ada nilai pada case yang sesuai dengan nilai kondisi, maka proses akan diteruskan kepada pernyataan yang ada di bawah ‘default’.

Bentuk umum dari struktur kondisi ini adalah :

```

switch(kondisi)
{
    case 1 : pernyataan-1;
    break;
    case 2 : pernyataan-2;
    break;
    .....
    .....
    case n : pernyataan-n;
    break;
    default : pernyataan-m
}

```

contoh

```

SWITCH .... CASE ...
switch(fo)
{
    case 1:
    for (k = 0; k<4 ; k++)
    {
        i=tabel1(k);
        PORTA = i;
        tunda(100);
    }
    break;
    case 2:
    for (k = 0; k<4 ; k++)
    {
        i=tabel2(k);
        PORTA = i;
        tunda(100);
    }
    break;
}

```

```

//Program SWITCH
#include <at89x51.h>
void main(void)
{
    char a;
    a=P2;
    switch(a)
    {
        case 0: P1=5;break;    //jika a=0 P1=5
        case 1: P1=10;break;   //jika a=1 P1=10 dst
        case 2: P1=15;break;
        case 3: P1=20;break;
    }
}

```

```

        case 4: P1=40;break;
        case 5: P1=60;break;
        default: P1=0;break; //jika a bukan 0,1,2,3,4,5,
maka P1=0.
    }
}

```

10. Perulangan

Dalam bahasa C tersedia suatu fasilitas yang digunakan untuk melakukan proses yang berulang-ulang sebanyak keinginan kita. Misalnya saja, bila kita ingin menginput dan mencetak bilangan dari 1 sampai 100 bahkan 1000, tentunya kita akan merasa kesulitan. Namun dengan struktur perulangan proses, kita tidak perlu menuliskan perintah sampai 100 atau 1000 kali, cukup dengan beberapa perintah saja. Struktur perulangan dalam bahasa C mempunyai bentuk yang bermacam-macam.

2.10.1. STRUKTUR PERULANGAN “ WHILE ”

Perulangan WHILE banyak digunakan pada program yang terstruktur. Perulangan ini banyak digunakan bila jumlah perulangannya belum diketahui. Proses perulangan akan terus berlanjut selama kondisinya bernilai benar (true) dan akan berhenti bila kondisinya bernilai salah.

Bentuk umum dari struktur kondisi ini adalah:

```

While (ekspresi)
{
    Pernyataan_1
    Pernyataan_2
}

```

Contoh Program 1 :

```

while (!TF0);
{
    TF0 = 0;
    TR0 = 0;
}

```

```
}
```

```
//Program while
#include <at89x51.h>
void main(void)
{
    char a=10;
    while(a>=0)    //jika a>= 0 bernilai benar maka
    pernyataan dalam
    {              //dalam blok di bawahnya dieksekusi
        P1=a;
        a--;
    }
}
```

2.10.2. STRUKTUR PERULANGAN “DO.....WHILE...”

Pada dasarnya struktur perulangan do....while sama saja dengan struktur while, hanya saja pada proses perulangan dengan while, seleksi berada di while yang letaknya di atas sementara pada perulangan do....while, seleksi while berada di bawah batas perulangan. Jadi dengan menggunakan struktur do...while sekurang-kurangnya akan terjadi satu kali perulangan.

Bentuk umum dari struktur kondisi ini adalah:

```
Do
    {
        Pernyataan_1
        Pernyataan_2
    }
While (ekspresi)
```

```
//Program DO.....WHILE
#include <at89x51.h>
void main(void)
{
    char a=10;
    do              //pernyataan dalam blok di bawahnya
    dieksekusi
    {              //selama while bernilai benar
        P1=a;
```

```

    a--;
} while(a>=0);
}

```

2.10.3. STRUKTUR PERULANGAN “FOR”

Struktur perulangan for biasa digunakan untuk mengulang suatu proses yang telah diketahui jumlah perulangannya. Dari segi penulisannya, struktur perulangan for tampaknya lebih efisien karena susunannya lebih simpel dan sederhana. Bentuk umum perulangan for adalah sebagai berikut :

```

for(inisialisasi; syarat; penambahan)
    pernyataan;

```

Keterangan:

Inisialisasi : pernyataan untuk menyatakan keadaan awal dari variabel kontrol.

syarat : ekspresi relasi yang menyatakan kondisi untuk keluar dari perulangan.

penambahan : pengatur perubahan nilai variabel kontrol.

Contoh

```

for (k = 0; k<4 ; k++)
{
    i=tabel1(k);
    PORTA = i;
    tunda50(100);
}

```

```

//Program for
#include <at89x51.h>

void main(void)
{
    char a;
    for(a=10;a>=0;a--)        //inisialisasi a=10, a=10 >=0
    kondisinya
        P1=a;                // benar maka pernyataan a-akan
    dieksekusi
}

```

11. Array (Larik)

Array merupakan kumpulan dari nilai-nilai data yang bertipe sama dalam urutan tertentu yang menggunakan nama yang sama. Letak atau posisi dari elemen array ditunjukkan oleh suatu index. Dilihat dari dimensinya array dapat dibagi menjadi Array dimensi satu, array dimensi dua dan array multi-dimensi.

2.11.1. ARRAY DIMENSI SATU

□□Setiap elemen array dapat diakses melalui indeks. □□Indeks array secara default dimulai dari 0. □□Deklarasi Array Bentuk umum :

Deklarasi array dimensi satu:

```
Tipe_array nama_array[ukuran];
```

2.11.2. ARRAY DIMENSI DUA

Array dua dimensi merupakan array yang terdiri dari m buah baris dan n buah kolom. Bentuknya dapat berupa matriks atau tabel.

Deklarasi array dimensi dua :

```
Tipe_array nama_array[baris][kolom];
```

2.11.3. ARRAY MULTI-DIMENSI

Array multi-dimensi merupakan array yang mempunyai ukuran lebih dari dua. Bentuk pendeklarasian array sama saja dengan array dimensi satu maupun array dimensi dua.

Bentuk umumnya yaitu :

```
tipe_array nama_array[ukuran1][ukuran2]...[ukuranN];
```

```
//Program Larik
#include <at89x51.h>
#include <stdio.h>
#include <ser.h>
```

```

code char info[ ]={"Selamat Datang."}; //larik string
data char data_suhu [4]={20,30,40,50}; //larik
desimal

void main()
{
    char i;
    EA=1;
    ser_init();
    for (i=0;i<14;i++)
    printf ("%c",info[i]); //mengakses larik info
    printf ("\n");
    for (i=0;i<4;i++)
    {
        printf ("Data suhu ke %d adalah %d",i,data_suhu[i]);
        printf("\n");
    }
}

void putchar(char u) //io diarahkan
ke serial
{
    if(!TI)
        SBUF=u;
    do
        ;
    while (TI);
    TI=0;
}

```

12. Fungsi

2.12.1. PENGERTIAN FUNGSI

Fungsi merupakan suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya terpisah dari program yang memanggilya. Fungsi merupakan elemen utama dalam bahasa C karena bahasa C sendiri terbentuk dari kumpulan fungsi-fungsi. Dalam setiap program bahasa C, minimal terdapat satu fungsi yaitu fungsi main(). Fungsi banyak diterapkan dalam program-program C yang terstruktur. Keuntungan penggunaan fungsi dalam

program yaitu program akan memiliki struktur yang jelas (mempunyai readability yang tinggi) dan juga akan menghindari penulisan bagian program yang sama.

2.12.2. PENDEFISIAN FUNGSI

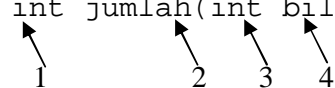
Sebelum digunakan fungsi harus didefinisikan terlebih dahulu. Bentuk definisi fungsi adalah:

```
Tipe_Nilai_Balik nama_fungsi(argumen1, argumen2)
{
    Pernyataan1;
    Pernyataan1;
    return(ekspresi);
}
```

Contoh:

```
int jumlah(int bil1,int bil2) //definisi fungsi jumlah
{
    int hasil;
    hasil = bil1 + bil2;
    return(hasil);
}

int jumlah(int bil1,int bil2)
1      2      3      4
```



Keterangan:

1. tipe data nilai balik fungsi
2. merupakan nama fungsi
3. tipe argumen
4. nama argumen

```
//program FUNGSI
#include <at89x51.h>
int jumlah(int bil1,int bil2) //definisi fungsi jumlah
{
    return(bil1+bil2);
}
void main()
{
```

```
P1=jumlah(20,50);           //pemanggilan fungsi jumlah
}
```

2.12.3. PROTOTYPE FUNGSI

Ketentuan pendefinisian fungsi yang mendahului fungsi pemanggil dapat merepotkan untuk program yang kompleks atau besar. Untuk mengatasi hal tersebut maka fungsi dapat dideklarasikan sebelum digunakan, terletak sebelum fungsi main. Deklarasi fungsi dikenal dengan prototype fungsi.

Cara mendeklarasikan fungsi sama dengan header fungsi dan diakhiri tanda titik koma (;)

```
//program PROTOTYPE FUNGSI
#include <at89x51.h>
int jumlah(int bil1,int bil2);      //prototype   fungsi
jumlah
void main()
{
    P1=jumlah(20,50);               //pemanggilan   fungsi
jumlah
}
int jumlah(int bil1,int bil2) //definisi fungsi jumlah
{
    return(bil1+bil2);
}
```

2.12.4. VARIABEL LOKAL DAN GLOBAL

Variabel lokal adalah variabel yang dideklarasikan di dalam suatu fungsi, variabel ini hanya dikenal fungsi tersebut. Setelah keluar dari fungsi ini maka variabel ini akan hilang.

Variabel global adalah variabel yang dideklarasikan di luar fungsi, sehingga semua fungsi dapat memakainya.

```

#include <at89x51.h>
#include <stdio.h>
#include <ser.h>

data char info; //variabel global
void tampil()
{
    data char info; //variabel lokal
    info=20;
    printf("Ini variabel lokal: %d\n",info);
}

void main()
{
    EA=1;
    ser_init();
    info=50;
    printf ("Ini adalah variabel global: %d\n",info);
    tampil();
    printf ("Ini adalah variabel global: %d\n",info);
}

void putchar(char u) //io diarahkan ke serial
{
    if(!TI)
        SBUF=u;
    do
        ;
    while (TI);
    TI=0;
}

```

2.12.5. KATA KUNCI *EXTERN* DAN *STATIC*

Kata kunci *extern* dan *static* digunakan untuk menyatakan sifat dari variabel atau fungsi. Suatu variabel atau fungsi yang didepannya ditambah dengan kata kunci *extern* maka artinya variabel atau fungsi tersebut didefinisikan di luar file tersebut.

Variabel global atau fungsi yang didepannya ditambah kata kunci *static* mempunyai arti bahwa variabel global atau fungsi tersebut bersifat *private* bagi file tersebut, sehingga tidak dapat diakses dari file yang lain.

Kata kunci *static* yang ditambahkan didepan variabel lokal (variabel di dalam suatu fungsi) artinya variabel tersebut dialokasikan pada memori statik. Nilai yang tersimpan dalam variabel statik tidak hilang walaupun sudah keluar dari fungsi.

```
//Program EXTERN DAN STATIC
#include <at89x51.h>
#include <stdio.h>
#include <ser.h>
data char info; //variabel global

void tampil()
{
    static data char info=20; //variabel local static
    info++;
    printf("Ini variabel lokal: %d\n",info);
}

void main()
{
    EA=1;
    ser_init();
    info=50;
    while(1)
    {
        printf ("Ini adalah variabel global: %d\n",info);
        tampil();
        printf ("Ini adalah variabel global: %d\n",info);
    }
}

void putchar(char u) //io diarahkan
ke serial
{
    if(!TI)
        SBUF=u;
    do
        ;
    while (TI);
    TI=0;
}
```

2.12.6. FUNGSI TANPA NILAI BALIK

Fungsi yang tidak mempunyai nilai balik menggunakan kata kunci void sedangkan fungsi yang tidak mempunyai argumen, setelah nama fungsi dalam kurung dapat kosong atau dengan menggunakan kata kunci void.

Contoh:

```
void tunda(void)
{
    for(i = 0; i < 10 ; i++);
}
```

atau

```
void tunda()
{
    for(i=0;i<10;i++);
    {}
}
/*-----*/
/*          fungsi tunda_panjang          */
/*-----*/
void tunda_panjang(int n)
{
    int i;
    for (i=0; i<n;i++)
        tunda();
}
```

2.12.7. FUNGSI DENGAN NILAI BALIK (*RETURN VALUE*)

Nilai balik dinyatakan dalam pernyataan return. Tipe nilai balik dapat berupa char, int, short, long, atau float

Contoh:

```
Char getch()
{
    if (RI)
    {
        RI = 0;
        Return(SBUF);
    }
}
```

2.12.8. ARGUMEN/ PARAMETER FUNGSI

Argumen dilewatkan ke dalam fungsi terdiri atas dua macam, yaitu:

a. Pelewatan secara nilai

Bentuk definisi pelewatan secara nilai adalah:

```
tipe nama_fungsi (tipe argumen1, tipe argumen2, ...)
{
    .....
    .....
}
```

```
//Program pelewatan_secara_nilai
#include <at89x51.h>
void Tambahv(int A)
{
    A=A+1;
}
void main()
{
    int B;
    B=4;
    Tambahv(B);
    P1=B;
}
```

b. Pelewatan secara pointer

Bentuk definisi pelewatan secara pointer adalah:

```
tipe nama_fungsi (tipe *argumen1, tipe *argumen2, ...)
{
    .....
    .....
}
```

```
//Program pelewatan_secara_pointer
#include <at89x51.h>
void Tambahp(int *A)
{
    *A=*A+1;
}
void main()
```

```
{
    int B;
    B=4;
    Tambahp (&B) ;
    P1=B;
}
```

13. STRUKTUR

Struktur merupakan sekelompok data (variabel) yang mempunyai tipe yang sama atau berbeda yang dikemas dalam satu nama.

2.13.1. DEKLARASI STRUKTUR

Deklarasi struktur dilakukan dengan format sebagai berikut:

```
Struct nama_struktur
{
    deklarasi variabel;
}
```

Contoh:

```
Struct kar_sensor
{
    unsigned char impedan;
    unsigned char koefi_suhu;
    unsigned char gain;
}
```

atau

```
typedef struct
{
    deklarasi variabel
} nama_struktur;
```

Contoh:

```
Typedef struct
{
    unsigned char impedan;
    unsigned char koefi_suhu;
    unsigned char gain;
}
```

2.13.2. PENDEFINISIAN VARIABEL STRUKTUR

Pada deklarasi struktur belum ada pengalokasian memori, oleh karena itu agar dapat digunakan maka perlu dilakukan pendefinisian variabel struktur.

Pendefinisian variabel struktur dilakukan dengan format sebagai berikut:

Bentuk 1:

```
Struct nama_struktur nama_variabel;
```

Contoh:

```
Struct kar_sensor sem_suhu;
```

Bentuk 2:

```
Nama_struktur nama_variabel
```

Contoh:

```
Kar_sensor sen_suhu;
```

2.13.3. MENGAKSES ANGGOTA STRUKTUR

Untuk mengakses anggota struktur dapat dilakukan dengan cara sebagai berikut:

```
Nama_variabel.anggota = data;    //untuk penulisan
Tampung = nama_variabel.anggota //untuk pembacaan
```

Contoh:

```
Sen_suhu.impedansi = 0x5;
Sen_suhu.kofi_suhu = 0x01;
Sen_suhu.gain = 0x04;
```

```
//Program MENGAKSES ANGGOTA STRUKTUR
#include <at89x51.h>

struct kar_sensor          //deklarasi struktur
{
    unsigned char impedan;
    unsigned char koefi_suhu;
    unsigned char gain;
```



```
};
void main()
{
    struct kar_sensor sen_suhu;    //definisi struktur

    sen_suhu.impedan=0x5;           //penulisan anggota
    struktur
    sen_suhu.koefi_suhu=0x01;
    sen_suhu.gain=0x04;
    P1=sen_suhu.impedan;           //pembacaan anggota
    struktur
    P2=sen_suhu.koefi_suhu;
    P3=sen_suhu.gain;
}
```

2.13.4. LARIK STUKTUR

Struktur dapat juga didefinisikan sebagai larik seperti berikut ini:

```
Struct kar_sensor dbase_sensor[4];
```

Untuk mengakses anggota struktur harus disertakan indeks lariknya.

Contoh:

Mengakses larik ke 0:

```
Dbase_sensor [0].impedan=0x5;
Dbase_sensor [0].koefi_suhu=0x01;
Dbase_sensor [0].gain=0x04;
```

Mengakses larik ke 1:

```
Dbase_sensor [1].impedan=0x6;
Dbase_sensor [1].koefi_suhu=0x05;
Dbase_sensor [1].gain=0x02;
```

```
//Program LARIK STUKTUR
#include <at89x51.h>

struct kar_sensor    //deklarasi struktur
{
    unsigned char impedan;
    unsigned char koefi_suhu;
    unsigned char gain;
};
```

```

void main()
{
    struct kar_sensor sen_suhu[2]; //definisi struktur
    sen_suhu[0].impedan=0x5;        //penulisan data
    anggota struktur
    sen_suhu[0].koefi_suhu=0x01;
    sen_suhu[0].gain=0x4;
    P1=sen_suhu[0].impedan;         //pembacaan data
    anggota struktur
    P2=sen_suhu[0].koefi_suhu;
    P3=sen_suhu[0].gain;
}

```

2.13.5. INISIALISASI STRUKTUR

Anggota struktur dapat diberi nilai ketika pendefinisian variabel struktur seperti pada berikut ini:

```
Struct kar_sensor sen_suhu = {0x05, 0x09, 0x01};
```

Contoh:

```

Struct kar_sensor dbase_sensor[4] =
{
    {0x05, 0x07, 0x09};
    {0x02, 0x04, 0x01};
    {0x04, 0x01, 0x03};
    {0x07, 0x03, 0x04};
}

```

```

//Program INISIALISASI STRUKTUR

#include <at89x51.h>
struct kar_sensor          //deklarasi struktur
{
    unsigned char impedan;
    unsigned char koefi_suhu;
    unsigned char gain;
};

void main()
{
    struct                                kar_sensor
    sen_suhu[2]={ {0x05,0x01,0x04}, {0x7,0x02,0x01}};
    P1=sen_suhu[0].impedan;               //pembacaan anggota
    struktur
    P2=sen_suhu[0].koefi_suhu;
}

```

```
P3=sen_suhu[0].gain;
}
```

2.13.6. POINTER STRUKTUR

Struktur dapat didefinisikan sebagai pointer seperti berikut ini:

```
Struct kar_sensor * dbase_sensor; //variabel
                                //pointer
Struct kar_sensor base;         //variabel
```

Untuk mengakses data anggota dilakukan dengan cara seperti berikut ini:

```
Dbase_sensor = &base;
Dbase_sensor -> impedan = 0x05;
Dbase_sensor -> koefi_suhu = 0x02;
Dbase_sensor -> gain = 0x03;
P1 = Dbase_sensor -> impedan
P2 = Dbase_sensor -> koefi_suhu;
P3 = Dbase_sensor -> gain
```

2.13.7. MELEWATKAN POINTER STRUKTUR KE FUNGSI

Penggunaan pointer struktur untuk melewati parameter ke fungsi dapat mencegah pelewatan data anggota struktur yang banyak, karena hanya parameter tertentu saja yang akan dilewatkan. Melewatkan pointer struktur ke fungsi dapat juga didefinisikan sebagai seperti berikut ini:

```
Struct kar_sensor base * dbase_sensor;
Dbase_sensor = &base;
Dbase_sensor -> impedan = 0x05;
Dbase_sensor -> koefi_suhu = 0x02;
Dbase_sensor -> gain = 0x03;
Kali_impedan(Struct kar_sensor *Structpointer)
{
    Structpointer -> impedan *= 2
}
P1 = Dbase_sensor -> impedan
P2 = Dbase_sensor -> koefi_suhu;
P3 = Dbase_sensor -> gain
```

14. POINTER

2.14.1. PENGERTIAN POINTER

Pointer (variabel penunjuk) adalah suatu variabel yang berisi alamat memori dari suatu variabel lain. Alamat ini merupakan lokasi dari obyek lain (biasanya variabel lain) di dalam memori. Contoh, jika sebuah variabel berisi alamat dari variabel lain, variabel pertama dikatakan menunjuk ke variabel kedua. Operator Pointer ada dua, yaitu :

- Operator &
 - Operator & bersifat unary (hanya memerlukan satu operand saja).
 - Operator & menghasilkan alamat dari operandnya.
- . Operator *
 - Operator * bersifat unary (hanya memerlukan satu operand saja).
 - Operator * menghasilkan nilai yang berada pada sebuah alamat.

2.14.2. DEKLARASI POINTER

Seperti halnya variabel yang lain, variabel pointer juga harus dideklarasikan terlebih dahulu sebelum digunakan.

Bentuk Umum :

Tipe_data *nama_pointer;

Tipe data pointer mendefinisikan tipe dari obyek yang ditunjuk oleh pointer. Secara teknis, tipe apapun dari pointer dapat menunjukkan lokasi (dimanapun) dalam memori. Bahkan operasi pointer dapat dilaksanakan relatif terhadap tipe dasar apapun yang ditunjuk. Contoh, ketika kita mendeklarasikan pointer dengan tipe `int*`, kompiler akan menganggap alamat yang ditunjuk menyimpan nilai integer - walaupun sebenarnya bukan (sebuah pointer `int*` selalu menganggap

bahwa ia menunjuk ke sebuah obyek bertipe integer, tidak peduli isi sebenarnya).
 Karenanya, sebelum mendeklarasikan sebuah pointer, pastikan tipenya sesuai dengan tipe obyek yang akan ditunjuk.

Contoh :

```
char *ptr;
data char *ptr;
```

Tabel 2.5 Ukuran Variabel Pointer

No	Kelas Memori	Lebar Pointer
1	generik	3
2	data	1
3	idata	1
4	xdata	2
5	code	2
6	pdata	1

Sdcc juga mendukung deklarasi pointer yang mengarahkan fisik pointer ke kelas memori tertentu.

Contoh:

```
/*secara fisik pointer berada di internal RAM yang
menunjukan ke RAM eksternal*/
xdata unsigned char *data p:

/* secara fisik pointer berada di eksternal RAM yang
menunjukan ke RAM internal*/
data unsigned char *data p:

/* secara fisik pointer berada di code ROM yang
menunjukan ke RAM eksternal*/
/* p harus diinisialisasi ketika dideklarasikan*/
xdata unsigned char *code p = 0x1000:

/* secara fisik pointer berada code ROM yang menunjukan
ke ROM*/
/* p harus diinisialisasi ketika dideklarasikan*/
code unsigned char *code p = 0x1000:
```

```
/* secara fisik pointer generic berada di eksternal
RAM*/
```

```
char *xdata p:
```

```
//Program POINTER
#include <stdio.h> //header fungsi printf
#include <at89x51.h> //
#include <serial.h>
#include <stdlib.h>

void main(void)
{
    char *aa; //pointer generik
    data char *ab; //menunjuk ke RAM internal
    (direct)
    idata char *ac; //menunjuk ke RAM internal
    (indirect)
    xdata char *ad; //menunjuk ke RAM eksternal
    code char *ae; //menunjuk ke memori program
    pdata char *af; //menunjuk ke memori page
    eksternal
    EA=0;
    serial_init();
    aa = malloc(4 *sizeof(char));
    printf("Ukuran pointer generik= %d\n",sizeof(aa));
    printf("Ukuran pointer (data) = %d\n",sizeof(ab));
    printf("Ukuran pointer (idata)= %d\n",sizeof(ac));
    printf("Ukuran pointer (xdata)= %d\n",sizeof(ad));
    printf("Ukuran pointer (code) = %d\n",sizeof(ae));
    printf("Ukuran pointer (pdata) = %d\n",sizeof(af));
}

void putchar(char c) //fungsi printf diarahkan ke
serial i/o
{
    if(!TI)
    {
        SBUF=c;
        TI=0;
    }
}
```

2.14.3. INISIALISASI POINTER

Setelah dideklarasikan pointer belum menunjuk ke suatu alamat tertentu, oleh karena itu perlu untuk diinisialisasi agar pointer menunjuk ke alamat tertentu sesuai dengan kebutuhan.

2.14.3.1. Menunjukkan Alamat Variabel

Menunjuk alamat variabel dilakukan dengan cara sebagai berikut:

```
int aku;           // deklarasi variabel
int *ptr;          // deklarasi pointer
ptr=&aku;           // inisialisasi pointer
                  // ptr = alamat variabel aku
```

atau

```
int aku;
int *ptr=&aku;
```

Tanda ‘&’ di depan variabel menyatakan alamat memori variabel tersebut.

2.14.3.2. Menunjukkan Alamat Memori Absolut

Disamping diarahkan untuk menunjukan alamat variabel, pointer juga dapat diinisialisasi untuk menunjukan alamat absolut dengan cara sebagai berikut:

```
int *ptrku;
ptrku= (int *) 0x8000;
```

atau

```
int *ptrku=(int *) 0x8000;
```

15. MENYISIPKAN INSTRUKSI ASSEMBLI

SDCC juga mendukung penyisipan instruksi dalam bahasa assembly. Instruksi assembly dituliskan diantara kata kunci `_asm` dan `_endasm` seperti berikut ini:

```
Void tunda()
{
    _asm
        mov r0, #20
```

```

        00001$: djnz r0, 00001$
    _endasm;
}

```

Jika suatu fungsi mempunyai parameter dituliskan menggunakan instruksi assembly maka parameter yang digunakan dituliskan sebelum kata kunci `_asm`, seperti berikut ini:

```

Void tunda(int mdetik)
{
    mdetik;
    _asm
        mov r0, #20
        00001$: djnz r0, 00001$
    _endasm;
}

```

```

//Program 7.1. LED Berjalan
#include <at89x51.h>
/*-----*/
/*  fungsi tunda */
/*-----*/

void tunda(int mdetik)
{
    mdetik;
    _asm
        mov r0, #0x0f5
        01$: mov r1, #0x0ff
        02$: mov r2, #0
            djnz r1, 02$
            djnz r0, 01$
    _endasm;
}

/*-----*/
/*          Program utama          */
/*-----*/

void main()
{
    char a;
    char k;
    while(1)
    {
        a=0x03;

```



```

        for (k=0;k<9;k++)
        {
            P1=a;                //mengeluarkan data LED
satu persatu
            tunda(100);          //sehingga efeknya seperti
LED berjalan
            a=a<<1;
        }
    }
}

```

2.15.1. PENGGUNAAN LABEL PADA INSTRUKSI ASSEMBLI

Label pada instruksi assembly berupa angka nnnnn\$ dengan nnnnn berupa angka di bawah 100. label pada instruksi assembly hanya dikenal oleh instruksi assembly, bahasa C tidak mengenal label pada penyisipan assembly dan juga sebaliknya.

Contoh:

```

Void conto()
{
    /*Pernyataan C*/
    _asm
        ; beberapa instruksi assembly
        ljmp 00003$
    _endasm;
    /*Pernyataan C*/
    clabel: /*instruksi assembly tidak mengenal*/
    _asm
        00003$: ; hanya dapat dikenal oleh assembly
    _endasm;
}

```

2.15.2. FUNGSI NAKED

Fungsi naked ini bermanfaat pada fungsi interupsi yang akhir definisinya ditambah kata junci “_naked”. Tambahan kata kunci “_naked” mengakibatkan kompiler tidak menambah prolog dan epilog pada fungsi.

Contoh:

```

Data unsigned char count;
Void tunda () interrupt 2 _naked
{

```

```
    _asm  
        inc count;  
        reti  
    _endasm;  
}
```