ISBN:

BUKU AJAR
DASAR PEMROGRAMAN

Dewi Pramudi Ismi, S.T., M.CompSc
Supriyanto, S.T., M.T.
Bambang Robiin, S.T., M.Eng
Fitri Indra Indikawati, S.Kom., M.Eng

Program Studi Teknik Informatika

HAK CIPTA

BUKU AJAR DASAR PEMROGRAMAN

Dewi Pramudi Ismi, S.T., M.CompSc
Supriyanto, S.T., M.T.
Bambang Robiin, S.T., M.Eng
Fitri Indra Indikawati, S.Kom., M.Eng

Diterbitkan oleh:
Program Studi Teknik Informatika
Fakultas Teknologi Industri
Universitas Ahmad Dahlan
Jalan Ring Road Selatan, Tamanan, Banguntapan, Bantul Yogyakarta 55166

Penulis	: Dewi Pramudi Ismi, S.T., M.CompSc Supriyanto, S.T., M.T. Bambang Robiin, S.T., M.Eng Fitri Indra Indikawati, S.Kom., M.Eng
Editor	: Laboratorium Teknik Informatika, Universitas Ahmad Dahlan
Desain sampul	: Laboratorium Teknik Informatika, Universitas Ahmad Dahlan
Tata letak	: Laboratorium Teknik Informatika, Universitas Ahmad Dahlan
Ukuran/Halaman	: 21 x 29,7 cm / 127 halaman

Didistribusikan oleh:

Laboratorium Teknik Informatika
Universitas Ahmad Dahlan
Jalan Ring Road Selatan, Tamanan, Banguntapan, Bantul Yogyakarta 55166
Indonesia

KATA PENGANTAR

Puji Syukur kehadirat Allah SWT yang telah melimpahkan Rahmat serta KaruniaNya sehingga kami dapat menyelesaikan buku ajar mata kuliah Dasar Pemrograman ini. Buku ajar ini dibuat sebagai panduan mahasiswa dalam mendalami mata kuliah Dasar Pemrograman dan mempelajari pemrograman dalam bahasa C/C++. Di dalam buku ajar ini, materi dasar dalam pemrograman C/C++ disampaikan dan diberikan contoh penggunaanya. Penulis menyadari bahwa dalam penyusunan buku ajar ini masih terdapat banyak kekurangan, oleh karenanya saran dan kritik yang membangun sangat kami harapkan. Akhir kata, semoga buku ini dapat bermanfaat bagi mahasiswa

Yogyakarta, Agustus 2019

Penulis

DAFTAR ISI

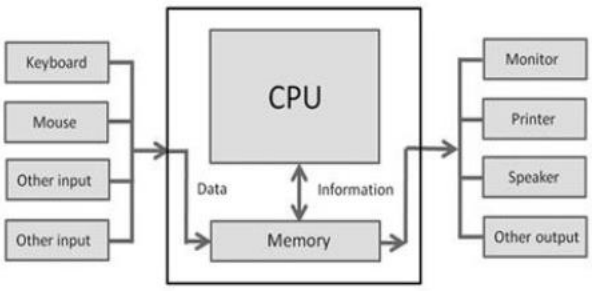
HAK CIPTA ii
KATA PENGANTAR iii
DAFTAR ISI iv
BAB 1. PENGENALAN DASAR PEMROGRAMAN vi
1.1 Pendahuluan vi
1.2 Instruksi, Program dan Software viii
1.3 Bahasa Mesin vs Bahasa Pemrograman viii
BAB II. INPUT DAN OUTPUT x
2.1. OUTPUT x
2.2 Input xi
BAB III. TIPE DATA dan VARIABEL xix
3.1 Tipe Data Native xix
3.2 Variabel xix
3.3. Konstanta xx
BAB IV. OPERATOR xxvi
BAB V. PERNYATAAN KONDISIONAL 38
5.1 Kondisional if 38
5.2 Kondisional if-else 40
5.3 Kondisional `switch-case` 42
5.4. Pernyataan Kondisional Bertingkat 46
Latihan 49
BAB VI. PERULANGAN 51
6.1 Perulangan dengan for 51
6.2 Perulangan dengan while 53
Latihan 56
BAB VII PERULANGAN BERTINGKAT 58
7.1 Perulangan for di dalam for 58
7.2 Perulangan while di dalam while 59
7.3 Perulangan for di dalam while 60
BABVIII FUNGSI 64
8.1 PENDAHULUAN 64
BAB IX FUNGSI REKURSIF 78
9.1 Pendahuluan 78
9.2 Pengertian Fungsi Rekursif 78
9.3 Karakteristik Fungsi Rekursif 79
9.4 Latihan 81
BAB X. ARRAY 83
10. 1 Definisi Array 83
10.2. Deklarasi Array 1 Dimensi 83
10.3. Pengisian Nilai Pada Array 84
10.4. Pembacaan Array 1 Dimensi 86
10.5 Array Dengan Tipe Data Huruf (char, string, dll) 89
10. 6. Pencarian Elemen Array 1 Dimensi 91
Metode Pencarian Linier 91
BAB XI. ARRAY 2 DIMENSI 93

11.1. Array 2 Dimensi 93
11.2. Deklarasi Array 2 Dimensi 94
11.3 Pengisian Nilai Pada Array 95
11.4. Pembacaan Array 2 Dimensi 97
BAB XII. STRUCT 100
12.1 Definisi Struct 100
12.2. Deklarasi Struct 101
12.3. Struct Sebagai User Defined Type 102
12.4. Pemanggilan Elemen Struct 103
12.5. Array of Struct 105
12.4 Nested Struct 107
BAB XIII POINTER 110
13.1. Pengenalan Pointer 110
13.2 Deklarasi Pointer 111
13.3 Pointer Untuk Alokasi Memori 113
13.4. malloc() dan free() 114
13.5 new dan delete 115
13.6 Pointer dan Array 117
13.7. Pointer ke Fungsi 119
BAB XIV INPUT/OUTPUT FILE 121
14.1 Pendahuluan 121
14.2 Membaca Text File 121
14.3 Menulis pada File 124
DAFTAR PUSTAKA 127

BAB 1. PENGENALAN DASAR PEMROGRAMAN

1.1 Pendahuluan

Komputer adalah perangkat elektronik yang mampu menerima input data dari penggunanya, memproses data tersebut sesuai instruksi, menampilkan output hasil pemrosesan data, serta menyimpan output tersebut untuk keperluan masa yang akan datang. Sehingga secara umum komputer memiliki prosesor, *storage* untuk menyimpan data, serta perangkat-perangkat untuk menerima input maupun menampilkan output. Gambar 1.1 di bawah ini menampilkan bagian-bagian dari sebuah komputer.



Gambar 1.1 Arsitektur Komputer

Perangkat untuk menerima input (*input devices*) terdiri dari keyborad, mouse, dan perangkat lain seperti scanner, kamera, joystick, microphone dll. Perangkat untuk menampilkan output (*output devices*) meliputi monitor, printer, speaker, proyektor, dll. Gambar 1.2 menunjukkan beberapa contoh input devices.

Gambar 1.2 Input Devices (www.examplesof.net)

Gambar 1.3 di bawah ini merupakan contoh-contoh output devices pada komputer.

Gambar 1.3 Output Devices pada komputer (examplesof.net)

Keseluruhan perangkat komputer yang telah disebutkan sebelumnya (input devices, output devices, processor, dan storage) merupakan perangkat keras (*hardware*). Hardware atau perangkat keras meliputi semua perangkat komputer yang bersifat *tangible,* dapat dilihat/disentuh bentuk fisiknya. Seperti keyboard, mouse, harddisk, processor, dll. Komputer **tidak** dapat bekerja hanya dengan *hardware* saja. Komputer memerlukan *software* (perangkat lunak) supaya dapat melakukan fungsi yang diinginkan oleh manusia penggunanya.

Apa saja yang termasuk *software* dalam komputer Anda? Mudahnya, segala sesuatu yang diinstall di dalam komputer/smartphone Anda adalah software. Untuk mengetahuinya Anda dapat membuka laptop Anda, jika Anda menggunakan Windows Operating System, Anda dapat mengecek semua software yang terinstall pada komputer Anda dengan membuka Control Panel → Programs → All Programs and Features. Berbagai macam contoh perangkat lunak yang biasa digunakan sehari-hari adalah aplikasi teks prosesor misalnya Microsoft Word, aplikasi media player seperti Winamp atau Windows Media Player, aplikasi browser misalnya Mozilla Firefox atau Google Chrome, dll. Berbagai aplikasi (*apps*) yang terinstall dalam smartphone juga merupakan software.

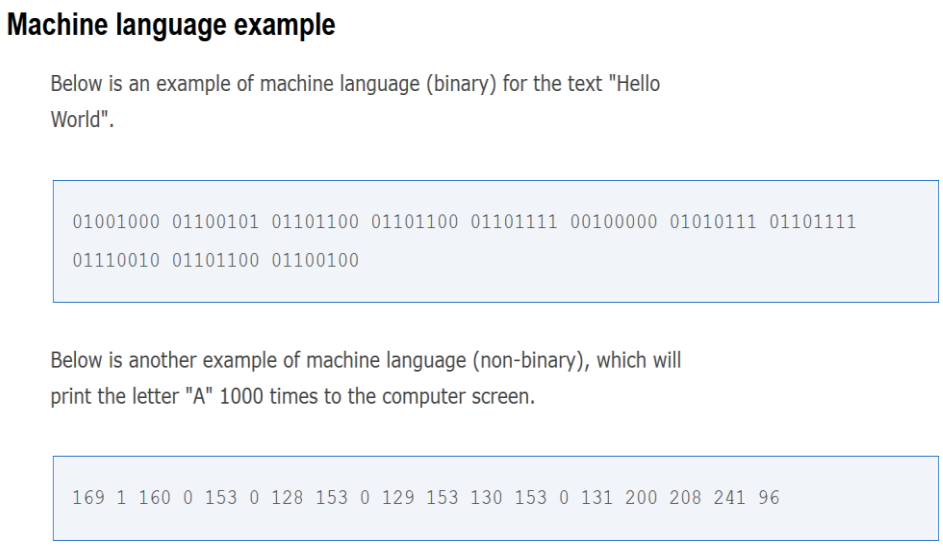
1.2 Instruksi, Program dan Software

Agar komputer dapat menjalankan fungsi yang diinginkan oleh manusia/penggunanya, maka manusia/penggunanya perlu memberikan instruksi kepada komputer. Instruksi kepada komputer diwujudkan dalam bentuk program. Program adalah sekumpulan instruksi yang harus dijalankan oleh komputer untuk menghasilkan suatu output tertentu atau untuk menjalankan suatu tugas tertentu.

Apa perbedaan program dengan software? Program adalah suatu unit tunggal yang dapat dieksekusi (*single executable unit*). Sedangkan software mengacu pada hal yang lebih kompleks dari pada program. Sebuah software melibatkan banyak program di dalamnya. Sebagai contoh, Aplikasi sosial media adalah sebuah software. Di dalamnya terdapat banyak program-program kecil seperti untuk mengupdate status, untuk menambah teman, untuk mengupload foto, untuk menulis komentar, dll.

1.3 Bahasa Mesin vs Bahasa Pemrograman

Komputer, dalam hal ini prosesor, dapat mengeksekusi/menjalankan instruksi yang ditulis dalam bahasa mesin. Bahasa mesin adalah bahasa satu-satunya yang dapat dimengerti oleh hardware komputer untuk menjalankan instruksi. Seperti apakah bahasa mesin itu?
Bahasa mesin dapat berupa bilangan biner, dan dapat pula berupa non-biner (perhatikan Gambar 1.4.).



Gambar 1.4. Contoh bahasa mesin

Manusia sulit untuk memahami bahasa mesin. Lantas bagaimana manusia memberikan instruksi kepada komputer? Untuk memudahkan penulisan instruksi yang diberikan kepada komputer, ilmuwan dalam bidang pemrograman komputer telah menciptakan bahasa pemrograman. Bahasa pemrograman adalah bahasa yang digunakan oleh manusia untuk menulis kode program. Bahasa pemrograman lebih '*readable*' (lebih mudah dibaca oleh manusia) daripada bahasa mesin. Bahasa pemrograman memiliki himpunan sintaks yaitu aturan-aturan berupa kata-kata serta notasi-notasi khusus untuk menuliskan instruksi. Ada macam-macam bahasa pemrograman yang dikenal saat ini misalnya C,C++, Java, Pascal, Python, dll.

Bahasa pemrograman terbagi menjadi dua kategori yaitu Compiled Language dan Interpreted Language. Bahasa pemrograman yang berjenis compiled language diantaranya adalah C, C++, Java, Pascal, Basic, dll. Sedangkan bahasa pemrograman yang berjenis interpreted language diantaranya adalah Python, Perl, dll. **Apa perbedaan Compiled Language dan Interpreted Language?** Pada compiled language, kode program yang telah ditulis diubah menjadi kode berbahasa mesin. Proses pengubahan ini disebut sebagai proses kompilasi. Hasil dari proses kompilasi adalah kode berbahasa mesin yang siap untuk dieksekusi (*executable file*), contoh file .exe dan .jar. Pada interpreted language, kode program yang telah ditulis akan dijalankan langsung oleh interpreter tanpa diubah dalam kode berbahasa mesin. Tidak ada proses kompilasi pada *interpreted language*. Eksekusi program pada *interpreted language* langsung dilakukan mulai dari instruksi pada baris teratas program sampai instruksi pada baris terbawah pada kode program yang telah ditulis.

BAB II. INPUT DAN OUTPUT

Suatu perangkat lunak memerlukan mekanisme untuk berinteraksi dengan pengguna (manusia). Saat proses pembuatan perangkat lunak atau programming harus dapat menyediakan mekanisme interaksi tersebut. Hal ini dapat diamati pada penggunaan aplikasi Calculator. Pengguna harus memasukkan angka untuk melakukan perhitungan, mekanisme ini disebut Input. Ketika aplikasi Calculator menampilkan angka dan hasil di layer, mekanisme ini disebut output. Bahasa pemrograman biasanya sudah menyediakan mekanisme Input dan Output. Bab ini akan membahasnya dalam dua Bahasa yaitu Bahasa C dan C++.

2.1. OUTPUT

2.1.1 Bahasa C

Bahasa C menyediakan Function untuk menampilkan data/informasi ke layar. Standar Input/Output pada Bahasa C harus menyatakan file header standar IO yaitu stdio. Function yang disediakan diantaranya adalah prinf() dan putchar(). Function printf() paling umum digunakan untuk menampilkan data/informasi. Berbagai jenis data dapat ditampilan melalui Function ini. Bentuk umum prinf() berisi 2 bagian yaitu diawali dengan string control diikuti dengan argument-argumen berikutnya.

```
printf("string control", argument);
```

String control adalah keterangan format string yang akan ditampilkan ke layer. Format tipe data diperlukan oleh compiler untuk dapat diterjemahkan. Tipe data lebih lanjut akan dibahas pada BAB 4. Namun pada BAB 3 ini, format tipe data sudah akan mulai digunakan seperti %d, %f dan %c. Format %d dan %f adalah format yang digunakan untuk menampilkan tipe data berupa angka. Berikut contoh penulisan sintak printf():

```
printf("%d", 50);
```

Contoh di atas akan menampilkan angka 50 ke layar. Format yang ditampilkan adalah format bilangan bukan character. Format character menggunakan format %s atau %c. Berikut adalah penjelasan format-format yang dapat digunakan:

%u	untuk menampilkan data bilangan tak bertanda (unsigned) dalam bentuk desimal.

%d dan %i	untuk menampilkan bilangan integer bertanda (signed) dalam bentuk desimal
%o	untuk menampilkan bilangan bulat tak bertanda dalam bentuk oktal
%x	untuk menampilkan bilangan bulat tak bertanda dalam bentuk heksadesimal
%X	(%x notasi yang dipakai : a, b, c, d, e dan f sedangkan %X notasi yang dipakai : A, B, C, D, E dan F)
%f	untuk menampilkan bilangan real dalam notasi : dddd.dddddd
%e dan %E	untuk menampilkan bilangan real dalam notasi eksponensial
%g	untuk menampilkan bilangan real dalam bentuk notasi seperti %f,%E atau %F %G bergantung pada kepresisian data (digit 0 yang tak berarti tak akan ditampilkan)
I	Merupakan awalan yang digunakan untuk %d,%u,%x,%X,%o untuk menyatakan long int (misal %ld). Jika diterapkan bersama %e,%E,%f,%F,%g atau %G akan menyatakan double
L	Merupakan awalan yang digunakan untuk %f,%e,%E,%g dan %G untuk menyatakan long double
h	Merupakan awalan yang digunakan untuk %d,%i,%o,%u,%x, atau %X, untuk menyatakan short int

Fungsi putchar() digunakan khusus untuk menampilkan character di layar. Contoh penulisannya adalah sebagai berikut:

```
putchar('A');
```

Hasil luarannya akan sama dengan printf() berikut:

```
printf("%c",'A');
```

2.1.2 Bahasa C++

Output pada Bahasa C++ memiliki mekanisme yang berbeda. Bahasa C++ perlu menyertakan header file yaitu iostream yang berisi standar Function yang digunakan dalam proses Input/Output. Cara menyertakan fie ini dengan menggunakan #include. Tanda (#) menrupakan pernyataan untuk menyertakan pre-processor. Sintak untuk menampilkan ke layar adalah sebagai berikut:

```
cout<<"Argumen yang akan ditampilkan";
```

Bahasa C++ tidak memerlukan format khusus yang digunakan untuk menampilkan ke layar. Compiler akan langsung menyesuaikan dengan Function yang ada dala iostream.

2.2 Input

2.2.1 Bahasa C

Bahasa C menyediakan Function untuk mekanisme user Input. Data dimasukkan melalui keyboard saat program dieksekusi. Function yang disediakan diantaranya adalah scanf() dan getchar(). Sama seperti sintak printf(), penulisan scanf() harus menyertakan format input. Perintah scanf() memerlukan variable (akan dibahas pada BAB berikutnya) untuk menampung data input yang dimasukkan yang kemudan nantinya dapat diproses dalam program yang dijalankan:

```
printf("string kontrol",argumen);
```

Daftar code string kontrol pada scanf adalah sebagai berikut:

%c	Membaca sebuah karakter
%s	Membaca sebuah string
%i atau %d	Membaca sebuah integer desimal
%e atau %f	Membaca sebuah bilangan real
%o	Membaca sebuah integer oktal
%x	Membaca sebuah integer heksadesimal
%u	Membaca sebuah integer tak bertanda
l	Awalan untuk membaca data long int (missal: %l d) atau untuk membaca data double (missal: %lf)
L	Awalan untuk membaca data long double (missal: %Lf)
h	Awalan untuk membaca data short int

Scanf() menempatkan argument dalam suatu alamat (dapat berupa variable) seperti pada contoh berikut:

```
printf("%f",&jarak);
```

Contoh berikut memerintahkan komputer untuk membaca sebuah bilangan real (%f) dan ditempatkan ke alamat dari variable jarak (&jarak).

Function getchar() khusus digunakan untuk memerima input berupa karakter/string dari keyboard. Contoh penggunaan getchar() sama dengan putchar(), tidak memerlukan kode format.

```
nilai = getchar();
```

Hasil eksekusinya akan sama dengan scanf() berikut:

```
scanf("%c",&nilai);
```

2.2.2 Bahasa C++

Input pada Bahasa C++ hampir sama dengan output, harus menyertakan file header iostream. Sintak yang digunakan adalah cin>>, seperti halnya cout<< tanpa memerlukan kode format data input.:

```
cin>>jarak;
```

Sintak cin>> tetap memerlukan variable untuk menampung data yang dimasukkan.

LATIHAN

Latihan 1

Memasukkan angka dan menampilkannya ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   Int angka;
   prinf("Masukkan angka 1: ");
   scanf("%d",&angka);
   printf("Angka yang dimasukkan= %d",angka);
}
```

Sintak cin>> tetap memerlukan variable untuk menampung data yang dimasukkan.

Latihan 2

Memasukkan angka dan menampilkan bil. real ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   float angka;
   printf(" Masukkan angka  = ");
   scanf("%f", &angka);
   printf("Angka yang dimasukkan %f\n",angka); }
```

Latihan 3

Memasukkan angka dan menampilkan bil. real (dengan aturan 2 angka dibelakang koma) ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   float angka;
   printf(" Masukkan angka  = ");
   scanf("%f", &angka);
   printf("Angka yang dimasukkan %.2f\n",angka); }
```

Latihan 4

Memasukkan angka dan menampilkan huruf ke layar dengan Bahasa C

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

```
main()
{
   char huruf;
   printf(" Masukkan huruf  = ");
   scanf("%c", &huruf);
   printf("Huruf yang dimasukkan %c\n",huruf);}
```

Latihan 5

Memasukkan angka dan menampilkan kata ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
      char kata[]="";
      printf(" Masukkan kata  = ");
      scanf("%s", &kata);
      printf("Kata yang dimasukkan %s\n",kata); }
```

Latihan 6

Memasukkan angka dan menampilkan huruf ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
      float angka= 3.1415926535897;
      printf("Angka yang ditampilkan %e \n",angka);
 }
```

Latihan 7

Memasukkan angka dan menampilkan huruf ke layar dengan Bahasa C

```
#include <stdio.h>

main()
{
      float angka= 3.1415926535897;
      printf("Angka yang ditampilkan %g \n",angka);
 }
```

Latihan 8

Memasukkan angka dan menampilkan jumlahnya ke layar dengan Bahasa C

```
#include <stdio.h>
main()
{
      int bil1, bil2;
      printf(" Masukkan angka bilangan 1 = ");
      scanf("%d", &bil1);
      printf(" Masukkan angka bilangan 2 = ");
```

```
      scanf("%d", &bil2);
printf("Hasil Penjumlahan bilangan 1 + bilangan 2 = %d ", bil1+bil2);
 }
```

Latihan 9

Memasukkan angka dan menampilkan jumlahnya ke layar dengan Bahasa C

```
#include <stdio.h>
main()
{
      int bil1;
     float bil2; //bilangan real
      printf(" Masukkan angka bilangan 1 = ");
      scanf("%d", &bil1);
      printf(" Masukkan angka bilangan 2 = ");
      scanf("%f", &bil2);
printf("Hasil Penjumlahan bilangan 1 + bilangan 2 = %f ", bil1+bil2);
 }
```

Latihan 10

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa C

```
#include <stdio.h>
main()
{
      float bil1, bil2; //bilangan real
      printf(" Masukkan angka bilangan 1 = ");
      scanf("%f", &bil1);
      printf(" Masukkan angka bilangan 2 = ");
      scanf("%f", &bil2);
printf("Hasil Penjumlahan bilangan 1 : bilangan 2 = %f ", bil1/bil2);
 }
```

Latihan 11

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
     float bil1;
      cerr<<"Masukkan angka = ";
      cin>>bil1;
      cerr<<"Angka yang dimasukkan = "<<bil1;
}
```

Latihan 12

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa

C++

```
#include <iostream>
using namespace std;

main()
{
      int bil1;
      cout<<"Masukkan angka = ";
      cin>>bil1;
      cout<<"Angka yang dimasukkan = "<<bil1;
 }
```

Latihan 13

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa

C++

```
#include <iostream>
using namespace std;

main()
{
      float bil1;
      cout<<"Masukkan angka = ";
      cin>>bil1;
      cout<<"Angka yang dimasukkan = "<<bil1;
}
```

Latihan 14

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa

C++

```
#include <iostream>
using namespace std;

main()
{     int bil1;
      float bil2;
      cout<<"Masukkan angka bilangan 1 = ";
      cin>>bil1;
      cout<<"Masukkan angka bilangan 2 = ";
      cin>>bil2;
      cout<<"Hasil Bilangan 1 - Bilangan 2 = "<<float(bil1-bil2);
 }
```

Latihan 14

Memasukkan angka (bilangan real) dan menampilkan hasil bagi keduanya ke layar dengan Bahasa C++

```
#include <iostream>
using namespace std;
main()
{     int bil1;
      int bil2;
      cout<<"Masukkan angka bilangan 1 = ";
      cin>>bil1;
      cout<<"Masukkan angka bilangan 2 = ";
      cin>>bil2;
      cout<<"Hasil Bilangan 1 - Bilangan 2 = "<< bil1+bil2;
 }
```

Latihan 16

Konversikan source code soal **latihan 4** ke dalam Bahasa C++

Latihan 17

Konversikan source code soal **latihan 5** ke dalam Bahasa C++

Latihan 18

Konversikan source code soal **latihan 8** ke dalam Bahasa C++

Latihan 19

Konversikan source code soal **latihan 9** ke dalam Bahasa C++

Latihan 20

Konversikan source code soal **latihan 10** ke dalam Bahasa C++

BAB III. TIPE DATA dan VARIABEL

3.1 Tipe Data Native

Data adalah suatu nilai yang dapat dinyatakan sebagai sebuah konstanta atau variable. Konstanta memiliki nilai tetap sedangnkan variable nilainya dapat berubah-ubah. Komputer dapat mengenali data dalam berbagai tipe atau bentuk. Terdapat 5 tipe data native/dasar yaitu:

1. **void** □ tanpa tipe data.
2. **int** □ bilangan bulat (integer)
3. **float** □ bilangan pecahan (floating point)
4. **double** □ bilangan pecahan dengan jangkauan data lebih luas
5. **char** □ karakter

Pada Bahasa C++ terdapat dua tipe data tambahan yaitu:

6. **bool** □ **bilangan** Boolean (True dan False)
7. **wchar_t** □ wide character

Setiap tipe data memiliki jangkauan dan total alokasi memori untuk menampungnya, seperti ditunjukkan pada tabel berikut:

Tipe	Ukuran (bit)	Jangkauan	
unsigned char	8	0	s/d 255
char	8	-128	s/d 127
short int	16	-32.768	s/d 32.767
unsigned int	32	0	s/d 4.294.967.295
int	32	-2.147.483.648	s/d 2.147.483.647
unsigned long	32	0	s/d 4.294.967.295
long	32	-2.147.483.648	s/d 2.147.483.647
float	32	3,4 e-38	s/d 1,7 E + 38
double	64	1,7 E-308	s/d 3,4 E + 308
long double	80	3,4 E-4932	s/d 1,1 E + 4932

3.2 Variabel

Variable digunakan untuk menyimpan data dengan tipe data tertentu. Agar variable dapat dikenali, maka perlu diperkenalkan atau dalam Bahasa pemrograman biasa disebut dengan deklarasi. Namun tidak semua Bahasa pemrograman harus mendeklarasikan variable baru yang

akan digunakan. Bahasa C dan C++ adalah salah satu Bahasa yang mewajibkan deklarasi nama variable yang akan digunakan sekaligus dengan tipe data dari data yang akan ditampung.

Deklarasi variable diawali dengan menuliskan code tipe data diikuti dengan nama variable dengan format sebagai berikut:

```
tipe nama_variable;
```

Variable dengan tipe data yang sama dapat dideklarasikan dalam satu baris perintah dengan dipisahkan dengan tanda koma.

```
tipe nama_variable1, nama_variable2;
```

Pemberian nama variable harus sesuai dengan aturan dasar variable sebagai berikut:

- Harus diawali dengan huruf (A..Z, a..z) atau karakter aris bawah (_)
- Karakter selanjutnya dapat berupa huruf, digit(0..9) atau karakter garis bawah (_) dan tanda dollar ($)
- Panjang boleh melebihi 31 karakter, namun hanya 31 karakter pertama yang akan dianggap berarti
- Nama variable tidak boleh menggunakan kata-kata yang sudah digunakan oleh Bahasa pemrograman seperti kode tipe data (int, float, dll), nama function standar (for, while, cin, cout, dll) dan lain sebagainya.
-

Pemberian nilai pada suatu variable dilakukan dengan memberikan tanda sama dengan (=) diikuti dengan nilai yang akan diberikan.

```
nama_variable = nilai;
```

Perhatikan contoh berikut beberapa cara deklarasi dan pemberian nilai suatu variable.

```
//Contoh pertama, deklarasi pada baris pertama
//Dilanjutkan pemberian nilai pada baris berikutnya
int angka;
angka = 10;

//Contoh kedua, deklarasi sekaigus pemberian nilai
int bilangan = 10;
```

3.3. Konstanta

Berbeda dengan variable, cara deklarasi konstanta menggunakan sintak define didahului pre-processor (#define). Format penulisan konstanta diawali tanda pagar diikuti define spasi nama konstanta spasi nilai, pemberian nilai tanpa menggunakan tanda asama dengan (=).:

```
#define nama_konstanta nilai;

//Contoh deklarasi konstanta
#define Angka 100
#define Phi 3.14
```

Konstanta berupa string merupakan deretan karakter, cara penulisan nilainya menggunakan tanda kutip seperti contoh berikut:

```
#define Judul "Dasar Pemrograman";
```

LATIHAN

Latihan 1

Pendeklarasian variable, pemberian nilai dan menampilkan layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   float angka;
   angka = 3.14;
   printf("Angka yang dimasukkan %.2f",angka); }
```

Latihan 2

Pendeklarasian variable, pemberian nilai dan menampilkan layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   int angka;
   angka = 100;
   printf("Angka yang dimasukkan %d",angka);
}
```

Latihan 3

Pendeklarasian variable, pemberian nilai dan menampilkan layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   double angka = 100;
   printf("Angka yang dimasukkan %f",angka);
}
```

Latihan 4

Pendeklarasian variable, pemberian nilai dan menampilkan layar dengan Bahasa C

```
#include <stdio.h>

main()
{
   char huruf = A;
   printf("Huruf yang dimasukkan %c",huruf);
}
```

Latihan 5

Pendeklarasian variable global, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

#define Phi 3.14
main()
{
   cout<<"Nilai dari Phi = "<<Phi;
}
```

Latihan 6

Pendeklarasian variable global, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

#define Phi 3.14
main()
{
   int r = 7;
   cout<<"Rumus Luas Lingkaran Phi x r x r = "<<float(Phi*r*r);
 }
```

Latihan 7

Pendeklarasian variable, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   char nim [] = "1900018XXX";
   cout<<"Nim Mahasiswa = "<<nim; }
```

Latihan 8

Pendeklarasian variable global, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int bil1 = 7;
   int bil2 = 8;
   cout<<"Hasil penjumlahan bil1 + bil2 ="<<bil1+bil2;
 }
```

Latihan 9

Pendeklarasian variable global, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
```

```
using namespace std;

main()
{
   double bil1 = 7;
   long bil2 = 3;
   cerr<<"Hasil penjumlahan bil1 + bil2 ="<<bil1+bil2;
 }
```

Latihan 10

Pendeklarasian variable global, pemberian nilai dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   float sisi;
   cout<<"Masukkan nilai sisi = ";
   cin>>sisi;
   cout<<"Luas persegi = "<<float(sisi*sisi);
}
```

Latihan 11

Buatlah program sederhana untuk mencari luas persegi panjang, dimana panjangnya 6, dan lebarnya 3 (dengan inputan statis) !

Latihan 12

Buatlah program sederhana untuk mencari keliling dari sebuah lingkaran dengan Phi =3.14 dan r = 7 (dengan inputan statis) !

Latihan 13

Buatlah program sederhana untuk mencari luas dari sebuah segitiga dengan alas =10 dan tinggi = 8 (dengan inputan statis) !

Latihan 14

Buatlah program sederhana untuk mencari nilai dari jari-jari atau r dari sebuah lingkaran jika diketahui luas lingkaran tersebut adalah 1384.74 (dengan inputan statis) !

Latihan 15

Buatlah program sederhana untuk mencari tinggi atau t dari sebuah segitiga jika diketahui luas segitiga tersebut adalah 31.5 (dengan inputan statis) !

Latihan 16

Buatlah program sederhana untuk mencari luas persegi panjang (dengan inputan dinamis) !

Latihan 17

Buatlah program sederhana untuk mencari keliling dari sebuah lingkaran dengan Phi =3.14 (dengan inputan dinamis) !

Latihan 18

Buatlah program sederhana untuk mencari luas dari sebuah segitiga (dengan inputan dinamis) !

Latihan 19

Buatlah program sederhana untuk mencari nilai dari jari-jari atau r dari sebuah lingkaran jika yang diketahui hanya luas lingkaran tersebut (dengan inputan dinamis) !

Latihan 20

Buatlah program sederhana untuk mencari luas dari sebuah trapesium jika yang diketahui luas trapesium tersebut (dengan inputan dinamis) !

BAB IV. OPERATOR

Operator merupakan karakter yang dapat dilibatkan diterjemahkan sebagai suatu perintah di dalam pemrograman. Bahasa pemrograman C/C++ mensupport beberapa jenis operator yaitu operator aritmatika, operator logika, operator relasional, serta operator bitwise.

1. Operator Aritmatika (+, -, x, /, %)

Operator Aritmatika dibagi menjadi dua kategori yaitu operator binary dan operator unary. Operator Binary adalah operator yang dikenakan pada dua buah nilai (operand).

Contoh: a + b

Operator (+) digunakan untuk menjumlahkan dua buah operand yaitu a dan b. Sedangkan Operator Unary adalah operator yang melakukan operasi terhadap satu operand.

Contoh: -c

Operator (-) di depan c melakukan operasi pengurangan dengan 1 operand yaitu c. Operator aritmatika secara rinci ditunjukkan tabel berikut:

Operator	Arti	Contoh
+	Penambahan	x + y
-	Pengurangan	x - y
*	Perkalian	x * y
/	Pembagian	x / y
%	Modulo - Sisa Pembagian	x % y (sisa pembagian dari x/y)

Eksekusi operator mengikuti tabel prioritas berikut:

Prioritas	Operator	
1	+, -	unary
2	*, /, %	binary

3	+, -	binary

2. Operator Compound Assignment (+=, -=, *=, /=, >>=, <<=, &=, ^=)

Operator ini melakukan perintah modifikasi nilai variable saat ini dengan meakukan operasi di atasnya.

Operator	Keterangan	Contoh Penggunaanya
+=	Penjumlahan	z += x sama dengan, z = z + x
-=	Pengurangan	z -= x sama dengan, z = z - x
*=	Perkalian	z *= x sama dengan, z = z * x
/=	Pembagian	z /= x sama dengan, z = z / x
>>=	Right shift AND	z >>= 2 sama dengan, z = z >> 2
<<=	Left shift AND	z <<= 2 sama dengan, z = z << 2
&=	Bitwise AND	z &= 2 sama dengan, z = z & 2
^=	Bitwise exclusive OR	z ^= 2 sama dengan, z = z ^ 2

Contoh Program Operator Compound Assignment

```
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a += 2;  // sama dengan a = a + 2
  cout << a;
}
```

3. Operator Increment dan Decrement (++, --)

Operator Increment (++) merupakan operator yang menjumlahkan nilai suatu operand dengan 1, sedangkan Decrement (--) merupakan operator yang mengurangkn nilai suatu operand dengan 1.

```
//code sebelum menggunakan increment dan decrement
x = x + 1 ;
y = y - 1 ;
```

```
//setelah menggunakan operator increment dan decrement

++ x ;
-- y ;

//atau

x ++ ;
y -- ;
```

Terdapat dua jenis operasi Increment maupun Decrement:

1. Pre Increment (++angka)
2. Post Increment (angka++)

Pre-Increment menambhakan nilai sebesar 1 sebelum operasi lain dijalankan, sedangkan Post-Decrement menambahkan nilai sebesar 1 setelah operasi lain dijalankan.

Contoh Program Operator Increment/Decrement

```
#include <iostream>
using namespace std;

int main ()
{
   int a, b;
   a = 3;
   b = ++a;

   cout << "a:" << a;
   cout << " b:" << b;
}
```

```
#include <iostream>

using namespace std;

int main(){
     // Mendeklarasikan Variabel KD
     int KD;

     // Mengisi nilai kedalam Variabel KD
     // dengan nilai 10
     KD = 10;

     // Melakukan Pre-Increment
     cout<<"Nilai KD awal : "<<KD<<endl;
     cout<<"Nilai ++MD    : "<<++KD<<endl;
     cout<<"Nilai KD      : "<<KD<<endl;
```

```
    // Mengubah nilai yang terdapat dalam variabel KD
    // dengan nilai 20
    KD = 20;

    // Melakukan Post-Increment
    cout<<"\nNilai KD awal : "<<KD<<endl;
    cout<<"Nilai KD++    : "<<KD++<<endl;
    cout<<"Nilai KD      : "<<KD<<endl;

    return 0;
}
```

4. Operator Relasional (==, !=, >, <, >=, <=)

Operator Relasional dan Komparasi akan selalu menghasilkan nilai Boolen (True atau Flase).

Berikut daftar operator tersebut:

Operator	Keterangan
==	Sama dengan
!=	Tidak sama dengan
>	Lebih dari
<	Kurang dari
>=	Lebih dari atau sama dengan
<=	Kurang dari atau sama dengan

Contoh Program Operator Relasional

```
#include <iostream>
using namespace std;

void main() {
    bool nilai;
    nilai = 3 > 2 ; // hasil ungkapan : benar
    cout << "Nilai = " << nilai;
    nilai = 2 > 3 ; // hasil ungkapan : salah
    cout << "Nilai = " << nilai;
}
```

5. Operator Logika (!, &&, ||)

Operator logika digunakan untuk membandingkan nilai dua variable atau lebih. Hasil operasinya selalu menghasilkan nilai Boolean (True dan False). Berikut daftar operator Logika dalam Bahasa C++:

Operator	Keterangan
&&	**AND** - Jika semua operand bernilai benar (TRUE) maka kondisi bernilai benar.
||	**OR** - Jika salah satu operand bernilai benar (TRUE) maka kondisi bernilai benar.
!	**NOT** - Digunakan untuk membalik kondisi. Jika kondisi benar (TRUE) maka akan berubah menjadi salah (FALSE), begitu pula sebaliknya.

Contoh Program Operator Logika

```
( (5 == 5) && (3 > 6) )  // Hasil = false, karena ( true && false )
( (5 == 5) || (3 > 6) )  // Hasil = true, karena ( true || false )
```

6. Operator Bitwise (&, |, ^, <<, >>)

Operator bitwise digunakan untuk melakukan manipulasi nilai yang ada dalam memori. Operator ini hanya dapat digunakan untuk tipe data char, int, dan long int. Operator bitwise dilakukan dalam bit per bit (binary)

```
&   AND  Bitwise AND
|   OR   Bitwise inclusive OR
^   NOT  Unary complement (bit inversion)
<<  SHL  Shift bits left
>>  SHR  Shift bits right
```

Tabel kebenaran operasi bitwise:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Contoh Program Operator Bitwise

```
A = 0011 1100
B = 0000 1101

//Hasilnya

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011
```

Latihan

Latihan 1

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = a+2;
   cout << "a:" << a;

   b += a;
   cout << " b:" << b;
}
```

Latihan 2

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = a+2;
   cout << "a:" << a;

   b += a;
   cout << " b:" << b; }
```

Latihan 3

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a,b;
   a = 5;
   b = a;
   a -= b;

   cout << "a:" << a;
   cout << " b:" << b;
}
```

Latihan 4

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   cout << "a:" << a;

   a += a;
   b = a;
   cout << " b:" << b;
}
```

Latihan 5

Penggunaan operator logika dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 3;
   cout << "a:" << a;

   b = ++a;
   cout << " b:" << b;
}
```

Latihan 6

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
    int a = 10;
    float b = 10.00;

     bool nilai;
    nilai =(( a == b ) && (a != b )) ; // hasil ungkapan : salah
    cout << "Nilai = " << nilai;
}
```

Latihan 7

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
```

```
using namespace std;

main()
{
    int a = 10;
    float b = 10.00;

    bool nilai;
    nilai =(( a == b ) || (a != b )) ; // hasil ungkapan : benar
    cout << "Nilai = " << nilai;
}
```

Latihan 8

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   cout << "a:" << a;

   a = --a;
   b = a;
   cout << " b:" << b;

   bool nilai;
   nilai = ( a == b )  ; // hasil ungkapan : bener
   cout << "Nilai = " << nilai;
}
```

Latihan 9

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;
main()
{
   int a, b;
   a = 2;
   b = a;
   a -= a;

   cout << "a:" << a;
   cout << " b:" << b;

   bool nilai;
   nilai = ( a > b )  ; // hasil ungkapan : salah
   cout << "Nilai = " << nilai;
```

```
}
```

Latihan 10

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;
main()
{
   a = 2;
   b = a;
   a -= a;

   cout << "a:" << a;
   cout << " b:" << b;

   bool nilai;
   nilai = (!( a > b ) && ( a < b ) ) ; // hasil ungkapan : bener
   cout << "Nilai = " << nilai;
}
```

Latihan 11

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   cout << "a:" << a;

   a = --a;
   b = a;
   cout << " b:" << b;

   bool nilai;
   nilai = (!( a == b ) && !( a >= b))  ; // hasil ungkapan : salah
   cout << "Nilai = " << nilai;
}
```

Latihan 11

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   b = 8;
```

```
   cout << "a:" << a;
   b = 8/4;
   cout << " b:" << b;

   bool nilai;
   nilai = !( a >= b ); // hasil ungkapan : salah
   cout << "Nilai = " << nilai;
}
```

Latihan 12

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   b = 4;
   b -= a;
   cout << "a:" << a;
   cout << " b:" << b;

   bool nilai;
   nilai = ( a <= b ); // hasil ungkapan : bener
   cout << "Nilai = " << nilai;
}
```

Latihan 13

Penggunaan operator dan menampilkan layar dengan Bahasa C++

```
#include <iostream>
using namespace std;

main()
{
   int a, b;
   a = 2;
   b = 5;
   cout << "a :" << a;
   cout << " b :" << b;
   cout << " b % a :" << b%a; //syntax modulus

   cout<<endl;
   bool nilai;
   nilai = a < ( b%a ); // hasil ungkapan : salah
   cout << "Nilai = " << nilai;  }
```

Latihan 14

Buatlah program penjumlahan sederhana menggunakan syntax operator penjumlahan binary dengan inputan statis!

Latihan 15

Buatlah program pengurangan sederhana menggunakan syntax operator pengurangan binary dengan inputan dinamis!

Latihan 16

Buatlah program penjumlahan sederhana menggunakan syntax operator penjumlahan unary dengan inputan dinamis!

Latihan 17

Buatlah program pengurangan sederhana menggunakan syntax operator pengurangan unary dengan inputan dinamis!

Latihan 18

Buatlah program sederhana menggunakan syntax operator dengan inputan dinamis yang dapat mendeteksi bilangan tersebut kecil dari 10!

Latihan 19

Buatlah program sederhana menggunakan syntax operator dengan inputan dinamis yang dapat mendeteksi bilangan tersebut merupakan bilangan positif!

Latihan 20

Buatlah program kalkulator sederhana menggunakan syntax operator dengan inputan dinamis!

BAB V. PERNYATAAN KONDISIONAL

Dalam paradigma pemrograman terstruktur terdapat tiga macam kontrol struktur: struktur sekuensial, struktur kondisional, dan struktur perulangan [1]. Pada dasarnya program komputer yang kita tuliskan mengikuti struktur sekuensial. Komputer akan mengeksekusi program secara sekuensial, artinya setiap pernyataan akan dieksekusi sesuai urutan kode program ditulis. Struktur kondisional dan struktur perulangan memerlukan sintaks atau bentuk pernyataan khusus untuk diimplementasikan. Struktur kondisional dibentuk oleh **pernyataan kondisional** yang akan dibahas pada bab ini. Sementara struktur perulangan dibentuk oleh pernyataan perulangan yang akan dibahas lebih lanjut pada bab 7 dan 8.

Pernyataan kondisional membuat program berjalan berdasarkan syarat (kondisi) tertentu. Apabila sebuah kondisi terpenuhi maka pernyataan berikutnya dijalankan. Apabila kondisi tersebut tidak terpenuhi maka terdapat dua kemungkinan, pernyataan berikutnya dilewati atau menjalankan pernyataan lain yang berbeda. Eksekusi pernyataan kondisional tersebut akan bergantung pada sintaks yang digunakan. Sintaks pernyataan kondisional pada pemrograman C++ dapat dibagi menjadi 3 macam yaitu: if, if-else, dan switch-case.

Pernyataan kondisional if merupakan **pernyataan kondisional pilihan tunggal** karena akan memilih atau mengabaikan suatu aksi atau pernyataan di dalam pernyataan kondisional. Pernyataan kondisional if-else merupakan **pernyataan kondisional dua pilihan** karena program akan dieksekusi dengan memilih salah satu dari dua pernyataan. Sementara, pernyataan kondisional switch-case disebut **pernyataan kondisional pilihan ganda/banyak** karena dapat memilih dari beberapa kelompok pernyataan [2]. Penjelasan lebih lanjut ada pada masing-masing sub-bab.

5.1 Kondisional if

---

[1] Corrado Böhm and Giuseppe Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Communications of the ACM* 9, no. 5 (1966): 366–71, https://doi.org/10.1145/355592.365646.

[2] Pual Deitel and Harvey Deitel, *C++ How to Program 9th Edition*, *Igarss 2014*, 2014, https://doi.org/10.1007/s13398-014-0173-7.2.

Pernyataan kondisional menggunakan sintaks if digunakan untuk mengeksekusi sebuah pernyataan atau blok pernyataan hanya jika kondisi tersebut terpenuhi. Apabila tidak terpenuhi maka eksekusi pernyataan atau pernyataan akan dilewati. Dasar penulisannya adalah:

```
if(kondisi)
  pernyataan
```

apabila pernyataan yang akan dieksekusi hanya satu, atau

```
if(kondisi){
  blok_pernyataan
}
```

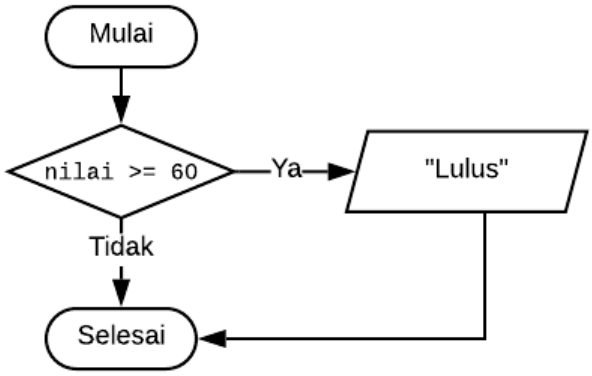
apabila pernyataan yang akan dieksekusi lebih dari satu pernyataan (berupa blok pernyataan)

Jika kondisi bernilai benar (*true*) atau bukan nol (*non-zero*) maka pernyataan atau blok pernyataan yang mengikuti kondisi if tersebut dieksekusi. Contoh sederhana adalah menampilkan teks "Lulus" apabila nilai mahasiswa >= 60, kode pernyataan kondisionalnya dapat dilihat pada listing berikut:

```
if(nilai >= 60)
    cout << "Lulus";
```

Listing Pernyataan kondisional `if` dengan satu pernyataan untuk dieksekusi

Apabila variabel nilai yang berisi nilai mahasiswa berisi angka 60 atau lebih artinya hasil evaluasi kondisi pada pernyataan kondisional tersebut bernilai benar (*true*), sehingga pernyataan yang mengikuti kondisional tersebut dijalankan yaitu menampillan teks "Lulus". Sebaliknya, apabila variabel nilai berisi angka kurang dari 60, maka pernyataan cout << "Lulus" dilewati karena tidak memenuhi kondisi nilai >= 60. Alur eksekusi pernyataan kondisionalnya dapat dilihat pada Gambar 0.1.



Gambar 0.1 flowchart pernyataan kondisional if

Kita dapat menambahkan tanda kurung { } apabila lebih dari satu pernyataan atau blok pernyataan dieksekusi. Contohnya adalah menampilkan teks "Lulus" dan "Nilai akhir A" apabila kondisi yang dievaluasi adalah variabel nilai berisi angka 80 atau lebih seperti contoh berikut:

```
if(nilai >= 80){
    cout << "Lulus";
    cout << ". Nilai Akhir A";
}
```

Listing Pernyataan kondisional `if` dengan blok pernyataan untuk dieksekusi

Contoh pada Listing dan Listing menggunakan evaluasi kondisi benar (*true*). Seperti telah dijelaskan sebelumnya bahwa selain evaluasi kondisi bernilai *true*, dapat digunakan evaluasi dengan nilai integral bukan 0 (*non-zero*). Artinya, semua evaluasi kondisi yang bernilai bukan nol dianggap bernilai *true* sehingga pernyataan dalam kondisional dapat dieksekusi. Contohnya bisa dilihat pada Listing berikut:

```
if(angka%pembagi)
    cout << angka << " tidak dapat dibagi oleh " << pembagi ;
```

Listing Evaluasi kondisi `if` dengan nilai integral bukan 0

Misalkan variabel `angka` berisi `11` dan `pembagi` berisi `3` maka nilai `angka%pembagi` adalah `2`. Karena hasil evaluasi kondisi `angka%pembagi` berupa angka `2` yang merupakan angka integral bukan nol, maka evaluasi kondisi dianggap bernilai benar sehingga pernyataan setelah sintaks if dieksekusi dan menghasilkan keluaran teks `11 tidak dapat dibagi oleh 3.`

5.2 Kondisional if-else

Pernyataan kondisional `if-else` terdiri dari dua pernyataan: pernyataan atau blok pernyataan yang dijalankan saat kondisi terpenuhi dan pernyataan atau blok pernyataan yang dijalankan saat kondisi tidak terpenuhi. Pernyataan pada sintaks `if` dijalankan apabila evaluasi kondisi terpenuhi seperti yang telah dijelaskan pada bab 5.1, sementara pernyataan sintaks `else` dijalankan apabila evaluasi kondisi pada `if` tidak terpenuhi. Bentuk kondisional `if-else` adalah sebagai berikut:

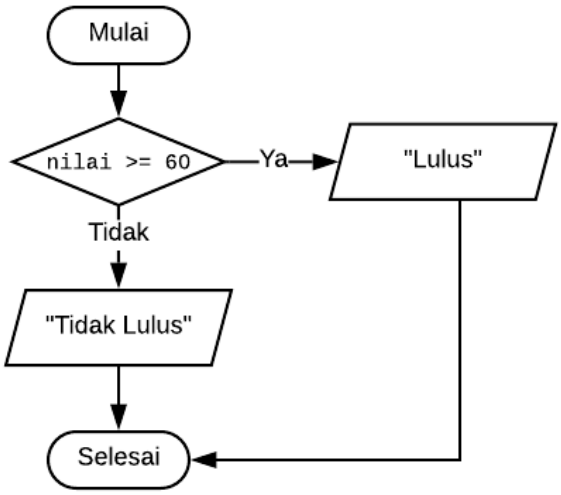
```
if(kondisi)
    pernyataan1
else
    pernyataan2
```

Kita ambil contoh Listing yang menampilkan teks `"Lulus"` apabila `nilai` adalah `60` atau lebih. Apabila variabel `nilai` berisi angka kurang dari `60` maka tidak ada baris program yang dijalankan karena evaluasi kondisi pada `if` bernilai salah (*false*). Penggunaan sintaks `else` dapat digunakan untuk menangani evaluasi kondisi yang tidak tertangkap oleh evaluasi kondisi pada `if`. Contohnya kita ingin menampilkan teks `"Tidak Lulus"` apabila `nilai` kurang dari `60`, maka kode pada Listing dapat kita tambahi `else` sehingga menjadi kondisional `if-else` seperti pada Listing 0.4.

```
if(nilai >= 60)
    cout << "Lulus";
else
    cout << "Tidak Lulus";
```

Listing 0.4 Pernyataan kondisional `if-else`

Jalannya eksekusi kode pada Listing 0.4 dapat dilihat pada Gambar 0.2. Pengecekan dilakukan pertama kali pada kondisional `if`. Apabila kondisi memenuhi yaitu `nilai` lebih dari sama dengan nol maka teks `"Lulus"` ditampilkan. Apabila `nilai` kurang dari 60 maka kondisi tidak terpenuhi, sehingga eksekusi langsung diterukan pada bagian kondisional `else`.



Gambar 0.2 Flowchart pernyataan pada kondisional `if-else`

5.3 Kondisional `switch-case`

Pada sub bab 5.1 telah dijelaskan bahwa kondisional if disebut **pernyataan kondisional pilihan tunggal** karena hanya dilakukan satu kali pengecekan kondisi. Sedangkan pada sub bab 5.2, kondisional else disebut **pernyataan kondisional dua pilihan** karena hanya ada dua kemungkinan kondisi yaitu memenuhi kondisi di dalam `if` atau tidak memenuhi sehingga otomatis masuk ke dalam pernyataan `else`. Pada sub bab ini, akan dibahas kondisional `switch-case` yang merupakan **pernyataan kondisional pilihan ganda/banyak** dimana kita dapat menentukan beberapa pernyataan kondisional dalam blok kode. Bentuk pernyataan kondisional `switch-case` adalah sebagai berikut:

```
switch(ekspresi_kondisi){
    case konstanta1:
        pernyataan1;
        ...
        break;
    case konstanta2:
        pernyataan2;
        ...
        break;
    default:
        pernyataan3;
        ...
        break;
```

```
}
```

Dalam pernyataan `switch-case` sebuah ekpresi kondisi atau variabel di dalam `switch` dicek kemudian dicocokkan dalam beberapa konstanta yang terdapat pada bagian `case`. Apabila ada konstanta yang memenuhi ekspresi di dalam `switch`, maka pernyataan yang ada dalam `case` dieksekusi. Misalnya, `konstanta1` memenuhi `ekspresi_kondisi`, maka `pernyataan1` dieksekusi. Sedangkan apabila `konstanta2` yang memenuhi `ekspresi_kondisi`, maka `pernyataan2` yang dieksekusi. Apabila tidak ada kondisi pada `case` yang memenuhi, maka pernyataan di dalam akan `default` dieksekusi.

Konstanta pada bagian `case` ini dapat berupa `string`, `integer`, `boolean` atau bentuk lain. Kita dapat membuat beberapa ekspresi kondisional `case` dalam satu blok `switch` dan hanya satu ekspresi kondisional `default`. Setiap blok `case` atau `default` harus ditutup dengan `break`. Jika tidak ditutup dengan `break` maka eksekusi program akan dilanjutkan (*fallthrough*) ke semua pernyataan `case` setelahnya dengan **tidak menghiraukan nilai kondisinya** hingga menemukan `break` atau sampai akhir blok `switch`.

Contoh sederhana adalah sebuah program untuk menampilkan keterangan nilai berdasarkan nilai huruf (A, B, C, D, E) pada Listing 0.5. Ekspresi kondisi berupa variabel yang bernama `nilai_huruf`. Isi dari variabel tersebut dicocokkan dengan beberapa konstanta pada `case`. Apabila terdapat konstanta yang sesuai dengan nilai variabel `nilai_huruf` maka pernyataan di dalam `case` dijalankan. Misalnya `nilai_huruf` adalah `B` maka pernyataan di dalam `case 'B':` dijalankan sehingga teks `"Lulus dengan nilai baik"` ditampilkan. Pernyataan `case 'B'` ditutup dengan `break` sehingga eksekusi program dihentikan atau dengan kata lain keluar dari blok `switch`.

```
switch(nilai_huruf){
    case 'A':
        cout << "Lulus dengan nilai memuaskan";
        break;
    case 'B':
        cout << "Lulus dengan nilai baik";
        break;
    case 'C':
        cout << "Lulus dengan nilai cukup";
        break;
    case 'D':
```

```
        cout << "Tidak lulus";
        break;
    case 'E':
        cout << "Tidak lulus";
        break;
    default:
        cout << "Nilai huruf tidak valid";
        break;
}
```

Listing 0.5 Pernyataan kondisional `switch-case`

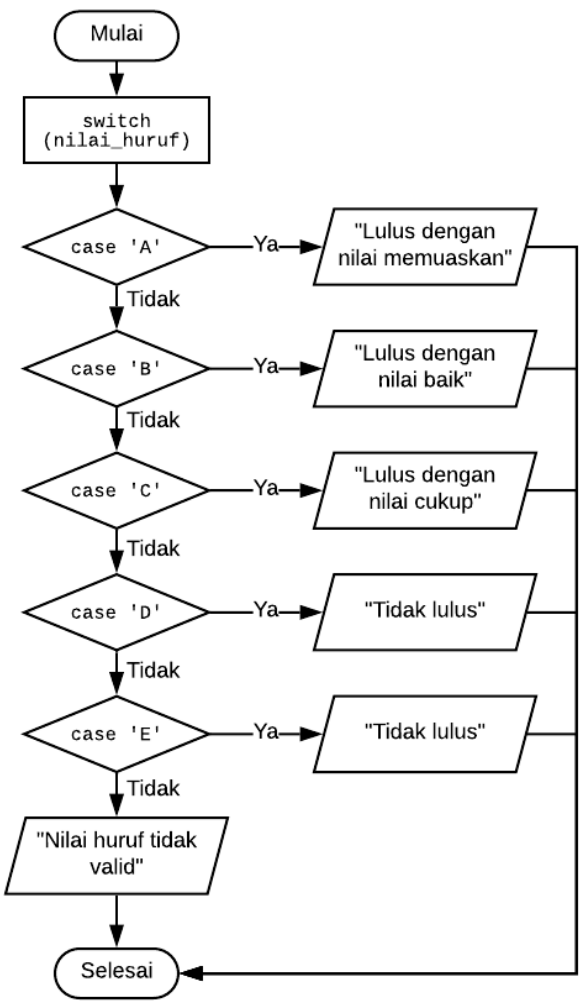
Fallthrough pada pernyataan kondisional switch-case

Pada contoh di atas terdapat kasus apabila `nilai_huruf` adalah `D` atau `E` maka teks `"Tidak lulus"` ditampilkan. Kode pada bagian tersebut dapat disederhanakan dengan memanfaatkan konsep *fallthrough* yang terjadi saat sintaks `break` dihilangkan. Kode yang telah disederhanakan dapat dilihat pada Listing 0.6. `case 'D'` tidak memiliki pernyataan dan tidak diakhiri dengan `break` sehingga `case 'D'` dan `case 'E'` tergabung atau *fallthrough*.

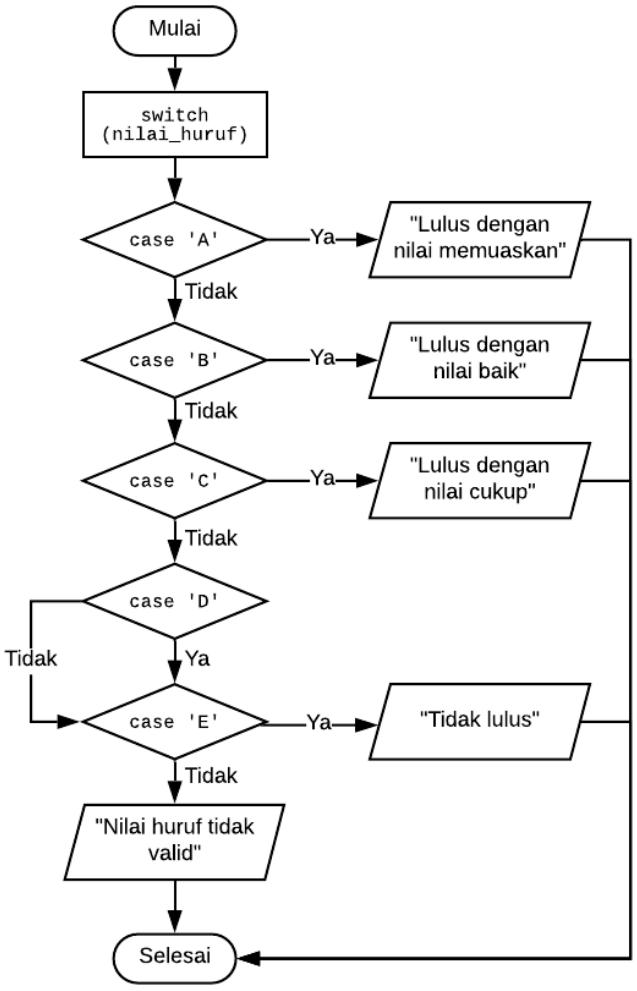
```
switch(nilai_huruf){
    case 'A':
        cout << "Lulus dengan nilai memuaskan";
        break;
    case 'B':
        cout << "Lulus dengan nilai baik";
        break;
    case 'C':
        cout << "Lulus dengan nilai cukup";
        break;
    case 'D':
    case 'E':
        cout << "Tidak lulus";
        break;
    default:
        cout << "Nilai huruf tidak valid";
        break;
}
```

Listing 0.6 Pernyataan kondisional `switch-case` dengan fall through

Apabila `nilai_huruf` berisi huruf `D` maka eksekusi program akan dilanjutkan ke `case` selanjutnya yaitu `case 'E'` dan pengecekan kondisi diabaikan sehingga teks **`"Tidak lulus"`** ditampilkan. Setelah itu ditemukan sintaks `break` sehingga eksekusi program dihentikan atau keluar dari blok `switch`.



1. tanpa *fallthrough*



(b) dengan *fallthrough*

Gambar 0.3 flowchart program menggunakan kondisional `switch-case`

Meskipun memiliki keluaran yang sama, kode pada Listing 0.5 dan Listing 0.6 mempunyai perbedaan dalam alur eksekusi kodenya. Perbedaan alur eksekusi program dapat dilihat pada Gambar 0.3. Gambar 0.3 (a) menunjukkan alur program Listing 0.5 tanpa *fallthrough* dimana semua sintaks `case` ditutup dengan `break` dan Gambar 0.3 (b) menunjukkan alur program Listing 0.6 dimana terdapat *fall through* pada `case 'D'`.

Berdasarkan flowchart pada Gambar 0.3 (a), apabila terdapat konstanta dalam `case` yang memenuhi ekspresi kondisi pada `switch` maka pernyataan yang mengikuti `case` tersebut

dijalankan dan eksekusi program langsung dihentikan atau langsung keluar dari blok `switch`. Misalkan `nilai_huruf` adalah `D` maka teks `"Tidak lulus"` ditampilkan dan eksekusi program dihentikan dan keluar dari blok `switch`. Sementara dalam flowchart di Gambar 0.3 (b), apabila `nilai_huruf` adalah `D` maka tidak ada pernyataan yang dijalankan. Alur eksekusi kode akan langsung masuk ke dalam `case 'E'` dan pengecekan kondisinya tidak dihiraukan atau dianggap benar sehingga pernyataan setelahnya langsung dijalankan dan menampilkan teks `"Tidak lulus"`. Setelah menampilkan teks `"Tidak lulus"` ditemukan sintaks `break` sehingga eksekusi program dihentikan dan keluar dari blok `switch`.

Penggunaan *fallthrough* berguna untuk menangani beberapa kondisi dalam pernyataan kondisional `switch-case` yang mempunyai aksi atau eksekusi pernyataan yang sama dalam dua atau lebih `case`. Akan tetapi, penggunaan kondisi *fallthrough* harus dilakukan dengan sangat hati-hati karena kelalaian dalam menempatkan sintaks `break` dapat menyebabkan kesalahan logika akibat pengabaian pengecekan kondisional pada *fallthrough*.

5.4. Pernyataan Kondisional Bertingkat

Pernyataan kondisional dapat dituliskan secara bertingkat artinya di dalam suatu pernyataan kondisional dapat disisipkan blok pernyataan kondisional lainnya. Contohnya kode untuk mengecek apakah sebuah angka adalah positif, negatif, atau merupakan bilangan nol. Dengan menggunakan pernyataan kondisional `if-else` maka program dapat ditulis seperti pada Listing 5.7.

```
if(angka > 0)
    cout << "bilangan positif";
else
    if(angka < 0)
        cout << "bilangan positif";
    else
        cout << "bilangan adalah nol";
```

Listing 5.7 pernyataan kondisional bertingkat `if-else`

Pada kode tersebut terdapat 2 tingkat pernyataan kondisional. Tingkat pertama atau terluar digunakan untuk mengecek apakah nilai variabel `angka` lebih dari nol atau tidak. Apabila evaluasi kondisi bernilai salah, maka akan masuk ke dalam level kedua yang terdiri dari blok pernyataan kondisional `if-else` lain. Pada tingkat kedua dilakukan pengecekan

variabel angka apakah kurang dari nol. Apabila evaluasi kondisi bernilai benar maka bilangan adalah positif sedangkan apabila salah maka bilangan adalah nol.

Struktur if dan else seperti pada Listing 5.7 dapat disederhanakan dan digabungkan menjadi pernyataan else if untuk menyeleksi beberapa jangkauan nilai sekaligus. Bentuk pernyataan kondisionalnya adalah sebagai berikut:

```
if(kondisi1)
    pernyataan1
else if(kondisi2)
    pernyataan2
else
    pernyataan3
```

Pengecekan dilakukan satu per satu dari atas ke bawah hingga ditemukan kondisi yang memenuhi persyaratan kondisional. Pertama dilakukan pengecekan pada kondisi1, apabila kondisi1 memenuhi maka pernyataan1 dijalankan dan apabila salah akan dilanjutkan pada pengecekan kondisi2. Apabila kondisi2 memenuhi maka pernyataan2 dijalankan. Jumlah pernyataan pada else if bisa lebih dari satu dan masing-masing pernyataan tersebut akan dicek satu per satu sampai menemukan kondisi yang memenuhi persyaratan kondisional. Apabila sampai akhir pernyataan else if tidak ada yang memenuhi, maka pernyataan setelah kondisional else dijalankan, yaitu pernyataan3.

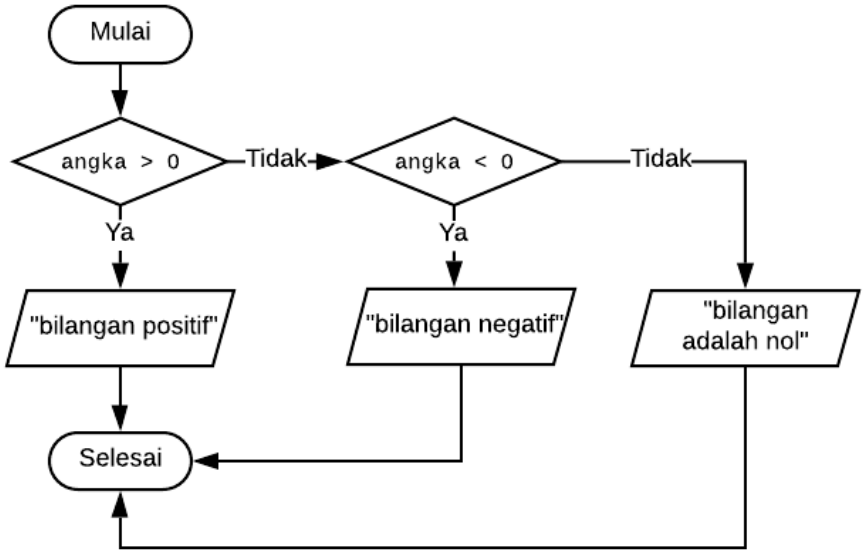
```
if(angka > 0)
    cout << "bilangan positif";
else if( angka < 0)
    cout << "bilangan negatif";
else
    cout << "bilangan adalah nol";
```

Listing 5.8 Pernyataan kondisional if - else if - else

Kode pada contoh Listing 5.8 merupakan kode yang identik dengan kode pada Listing 5.7. Pengecekan awal dilakukan pada kondisional pertama yaitu pengecekan angka > 0. Apabila angka lebih dari nol maka teks "bilangan positif" akan ditampilkan. Apabila salah pengecekan dilanjutkan pada kondisional else if dan apabila angka bernilai kurang dari 0 maka teks "bilangan negatif" ditampilkan. Apabila angka tidak memenuhi kedua

kondisi lebih dari nol dan kurang dari nol maka hanya ada satu kemungkinan yaitu angka tersebut bernilai nol. Karena tidak memenuhi kedua kondisi yang telah ditentukan, maka eksekusi pada kondisional `else` dijalankan sehingga teks yang ditampilkan adalah `"bilangan adalah nol".`

Pada dasarnya kode pada Listing 5.7 dan Listing 5.8 identik kecuali pada perbedaan spasi dan indentasi baris. Perbedaan indentasi ini diabaikan oleh kompiler sehingga kode bisa disederhanakan dari bentuk pernyataan kondisional bertingkat `if-else` pada Listing 5.7 menjadi pernyataan `if - else if - else` pada Listing 5.8. Penulisan kode seperti pada Listing 5.8 biasanya lebih banyak digunakan karena lebih mudah dibaca dan dikelola apabila terdapat perubahan logika. Alur eksekusi pada contoh Listing 5.7 dan Listing 5.8 sama seperti dapat dilihat pada Gambar 0.4.



Gambar 0.4 Flowchart kode program pengecekan bilangan

Latihan

Kerjakan latihan berikut dengan menuliskan kode menggunakan pernyataan kondisional:

1. Tulislah kode untuk mengecek apakah nilai dua bilangan sama atau tidak!
   Contoh
   Input : bilangan pertama = 3
   bilangan kedua = 2
   Output : kedua bilangan tidak sama
2. Tulislah kode untuk mencari nilai maksimum dari dua bilangan!
   Contoh
   Input : bilangan pertama = 5
   bilangan kedua = 3
   Output : bilangan terbesar adalah 5
3. Tulislah kode untuk menampilkan nama bulan Masehi dengan memasukkan bulan dalam bentuk angka (1 – 12)!
   Contoh
   Input : bulan ke = 5
   Output : bulan ke-5 adalah bulan Mei
4. Tulislah kode untuk menampilkan jumlah hari dalam suatu bulan!
   Contoh
   Input : bulan = juli
   Output : di bulan juli ada 31 hari
5. Tulislah kode untuk mengecek apakah sebuah huruf/karakter merupakan vokal atau konsonan!
   Contoh
   Input : huruf = 'B'
   Output : huruf B adalah huruf konsonan
6. Buatlah kode program untuk kalkulator sederhana dengan operasi penjumlahan, pengurangan, perkalian, dan pembagian!
7. Tulislah kode untuk memasukkan panjang sisi-sisi sebuah segitiga dan mengecek apakah panjang masing-masing sisi segitiga valid atau tidak!
   Contoh
   Input : sisi pertama = 5
   sisi kedua = 9

sisi ketiga = 6

Output : segitiga tersebut valid

8. Tulislah kode untuk memasukkan panjang sisi-sisi segitiga dan mengecek apakah segitiga tersebut termasuk segitiga siku, segitiga sama sisi, segitiga sama kaki, atau segitiga sembarang!

   Contoh

   Input : sisi pertama = 5

   sisi kedua = 9

   sisi ketiga = 5

   Output : segitiga sama kaki

9. Tulislah kode untuk menampilkan jumlah lembar uang kertas yang diberikan berdasarkan harga total yang diberikan!

   Contoh:

   Nilai uang : 500, 100, 50, 10, 5, 2, 1

   Input : 574

   Output : 5 lembar dengan rincian 500 = 1, 50 = 1, 20 = 1, 2 = 2

10. Tuliskan kode untuk menghitung total tagihan listrik berdasarkan pemakaian (kWH) dengan aturan pembayaran sebagai berikut:

    50 kWH pertama Rp. 1100/kWH

    100 kWH selanjutnya Rp. 1200/kWH

    100 kWH selanjutnya Rp. 1300/kWH

    Untuk pemakaian di atas 250 kWH Rp. 1500/kWH dan dikenakan biaya tambahan sebesar 5% dari total pembayaran

BAB VI. PERULANGAN

Iterasi atau perulangan adalah eksekusi pernyataan yang dilakukan secara berulang. C++ mempunyai tiga bentuk perulangan: `for`, `while`, dan `do-while`. Masing-masing bentuk perulangan mempunyai ciri atau komponen dann alur eksekusi yang berbeda. Sebagian besar pernyataan perulangan dapat dituliskan ke dalam ketiga bentuk `for`, `while`, dan `do-while` . Akan tetapi, terdapat beberapa kasus yang hanya bisa diselesaikan dengan bentuk perulangan tertentu. Penjelasan lebih lengkap akan dibahas pada masing-masing sub-bab.

6.1 Perulangan dengan for

Perulangan menggunakan sintaks `for` merupakan bentuk perulangan yang terkontrol oleh *counter*. Terdapat beberapa elemen dalam perulangan terkontrol *counter* antara lain[3]:

1. Nama dari **variabel kontrol** (*counter* perulangan)
2. **Nilai awal** atau inisial dari variabel kontrol
3. **Kondisi kelanjutan perulangan** yang mengecek **nilai akhir** dari variabel kontrol (pengecekan apakah perulangan harus dilajutkan atau dihentikan)
4. Perubahan berupa **kenaikan** atau **penurunan** nilai variabel kontrol yang dimodifikasi setiap iterasi

Elemen-elemen tersebut dapat diterjemahkan ke dalam bentuk perulangan menggunakan `for` dengan format seperti berikut:

```
for (inisialisasi; kondisi; perubahan_nilai)
    pernyataan
```

Fungsi utama dari bentuk perulangan `for` di atas adalah untuk mengulang `pernyataan` (berupa pernyataan tunggal ataupun blok pernyataan) selama `kondisi` bernilai benar. Jalannya perulangan dapat dijabarkan dalam langkah-langkah berikut:

---

[3] Pual Deitel and Harvey Deitel, *C++ How to Program 9th Edition*, *Igarss 2014*, 2014, https://doi.org/10.1007/s13398-014-0173-7.2.

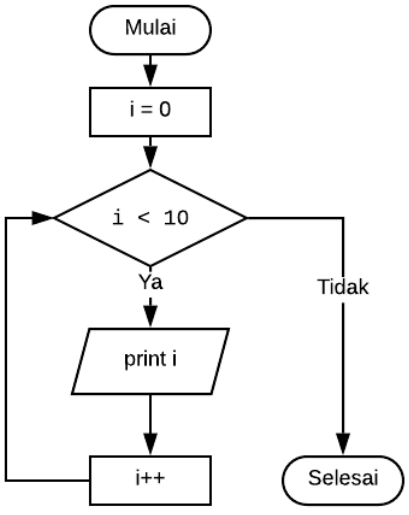
a. Iterasi atau perulangan dimulai dari eksekusi pernyataan `inisialisasi` yang biasanya dilakukan untuk menentukan nilai awal dari variabel kontrol. Eksekusi pernyataan `inisialisasi` ini hanya dilakukan sekali di awal perulangan.
b. Pada setiap iterasi, `kondisi` selalu dicek apakah bernilai benar atau salah.
c. Apabila `kondisi` bernilai benar maka iterasi dilanjutkan dan `pernyataan` dieksekusi. Apabila `kondisi` bernilai salah maka iterasi dihentikan, `pernyataan` tidak lagi dijalankan dan keluar dari blok perulangan.
d. Perubahan variabel kontrol dengan eksekusi pernyataan `perubahan_nilai` yang menambah atau mengurangi nilainya pada setiap iterasi.
e. Mengulangi langkah 2-4 hingga pengecekan pada langkah 2 bernilai salah.

Contoh perulangan sederhana untuk menampilkan bilangan 0 sampai 10 pada Listing 0. berikut:

```
for (int i = 0; i <= 10; i++)
   cout << i << endl;
```

Listing 0. perulangan menggunakan sintaks `for`

Iterasi dilakukan dengan inisialisasi variabel kontrol `i` dengan nilai `0`. Pada iterasi pertama variabel `i=0` sehingga kondisi `i <= 10` bernilai benar sehingga pernyataan untuk menampilkan variabel `i` dijalankan dan angka `0` ditampilkan. Selanjutnya, nilai variabel `i` ditambah `1` dengan sintaks `i++`. Pada iterasi kedua, nilai variabel `i` sudah menjadi `1`. Kondisi `i <= 10` tetap bernilai benar sehingga angka 1 ditampilkan, nilai variabel `i` ditambah `1` dan perulangan dilanjutkan pada iterasi ketiga. Perulangan akan diteruskan sampai nilai `i <= 10` bernilai salah yaitu setelah iterasi ke-11. Pada iterasi ke-11 nilai `i=10` sehingga setelah menampilkan variabel `i` dan variabel `i` bertambah menjadi `11`. Saat akan melanjutkan iterasi, kondisi `i <= 10` dicek dan bernilai salah karena nilai `i=11` sehingga iterasi dihentikan dan keluar dari blok perulangan. Alur program dapat dilihat pada Gambar 0.2.



Gambar 0.2 flowchart perulangan menggunakan `for` untuk menampilkan angka 0-10

Bentuk khusus perulangan menggunakan for

Bagian dari sintaks perulangan menggunakan `for` yaitu `inisialisasi, kondisi,` dan `perubahan_nilai` bisa dihilangkan. Akan tetapi, harus ada dua titik koma (;) dalam sintaks `for`. Sehingga sintaks `for` yang paling pendek adalah:

```
for( ; ; )
```

Pernyataan di atas valid karena sesuai format dari sintaks `for` tetapi menyebabkan perulangan yang tidak dapat dihentikan (infinite loop). Infinite loop terjadi karena `kondisi` yang kosong akan dianggap bernilai benar saat eksekusi sehingga iterasi akan terus dilakukan.

Untuk menghindari infinite loop kita dapat mendeklarasikan `kondisi` pada sintaks `for`. Bentuk perulangannya menjadi seperti berikut:

```
for( ; kondisi ; )
```

Sintaks tersebut ekuivalen dengan perulangan menggunakan `while` dengan bentuk `while(kondisi)` yang akan dibahas pada sub bab setelah ini. Perulangan akan dilakukan selama `kondisi` bernilai benar.

6.2 Perulangan dengan while

Sintaks dari perulangan menggunakan `while` adalah sebagai berikut:

```
while(kondisi)
  pernyataan
```

Dalam perulangan menggunakan `while`, bagian `kondisi` dicek kebenarannya saat perulangan pertama kali dijalankan. Jika bernilai benar maka pernyataan atau blok pernyataan yang ada di dalam perulangan akan dieksekusi. Iterasi akan terus dilakukan hingga `kondisi` bernilai salah maka perulangan dihentikan dan keluar dari perulangan. `kondisi` biasanya berupa ekspresi logika atau pernyataan yang bisa dikonversi menjadi tipe `bool`. Seperti yang telah dijelaskan pada bab pernyataan kondisional bahwa nilai nol dapat dikonversi menjadi bernilai salah atau `bool=false` dan nilai selain nol dapat dikonversi menjadi bernilai benar atau `bool=true`.

Kita dapat merubah bentuk perulangan menggunakan `for` menjadi bentuk perulangan menggunakan `while`. Kode pada Listing 0. dapat kita ubah menjadi bentuk perulangan `while` sebagai berikut:

```
int i = 0;                  //inisialisasi
while (i <= 10){            //kondisi
   cout << i << endl;       //pernyataan
   i++;                     //perubahan_nilai
}
```

Listing 6.2 Perulangan menggunakan `while`

Perulangan menggunakan `for` pada Listing 0. dan perulangan menggunakan `while` Listing 6.2 mempunyai alur eksekusi yang sama. Terdapat `kondisi` untuk pengecekan kondisi perulangan berdasarkan variabel kontrol, `inisialisasi` untuk variabel kontrol, `pernyataan` yang akan dieksekusi pada setiap iterasi, dan `perubahan_nilai` untuk merubah nilai variabel kontrol.

Pada kode tersebut variabel kontrol yang digunakan untuk pengecekan kondisi perulangan adalah variabel `i`. Dalam perulangan menggunakan `for`, variabel `i` diinisialisasi di dalam sintaks `for`. Sementara pada perulangan menggunakan `while` , inisialisasi dilakukan sebelum sintaks `while`. Ketika menuliskan perulangan menggunakan `while`, perlu didefinisikan sebuah aksi atau metode untuk memaksa `kondisi` menjadi bernilai salah. Apabila tidak dilakukan, maka iterasi akan diulang selamanya (*infinite loop*). Pada kasus di atas, pernyataan `i++` digunakan untuk memaksa keluar dari perulangan pada suatu titik dalam iterasi. Nilai `i` akan terus bertambah sampai saat dimana `i` mencapai angka `11` dan kondisi `i<=10` bernilai salah sehingga perulangan dihentikan dan eksekusi keluar dari blok perulangan.

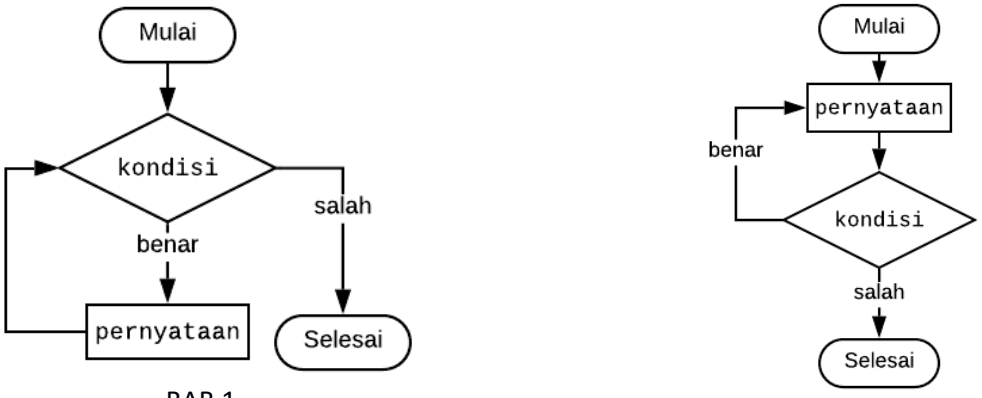
Dalam bahasa pemrograman C, perulangan `for` dapat dirubah bentuknya menjadi perulangan `while`, dan sebaliknya. Akan tetapi, penggunaan masing-masing perulangan sebaiknya dibedakan berdasarkan kondisi dan tujuan dari perulangan. Perulangan `for` biasanya digunakan untuk melakukan iterasi sebanyak jumlah yang dapat diukur (ada titik awal dan titik akhir), misalnya iterasi sebanyak 10 kali atau perulangan sebanyak panjang sebuah array. Sementara perulangan `while` biasanya digunakan untuk melakukan iterasi berdasarkan suatu kondisi perulangan dan biasanya tidak diketahui berapa jumlah iterasi yang dibutuhkan dalam sebuah perulangan. Selain alasan kemudahan dalam membaca kode (*readibility*), ada beberapa kasus dalam bahasa pemrograman lain dimana bentuk perulangan tidak bisa dirubah bentuknya.

6.3 Perulangan dengan do-while

Bentuk dasar perulangan menggunakan adalah sebagai berikut:

```
do{
   pernyataan
}
while(kondisi);
```

Pada dasarnya perulangan menggunakan `do-while` memiliki bentuk dan fungsionalitas yang mirip dengan perulangan menggunakan `while`. Bedanya adalah pada alur eksekusi pengecekan kondisi kelanjutan perulangannya. Dalam perulangan menggunakan `while`, `kondisi` dievaluasi sebelum eksekusi pernyataan dalam perulangan. Apabila bernilai salah maka perulangan tidak dilanjutkan. Sementara dalam perulangan menggunakan `do-while`, `kondisi` dievaluasi setelah eksekusi pernyataan dalam perulangan. Perbedaan alur eksekusi pada perulangan menggunakan `while` dengan perulangan menggunakan `do-while` dapat dilihat pada Gambar 6.2 .



BAB 1.

(b)

Gambar 6.2. perbedaan alur eksekusi (a) perulangan `while` dan (b) perulangan `do-while`

Latihan

1. Tulislah kode menggunakan perulangan untuk menampilkan 10 bilangan positif pertama!
2. Tulislah kode menggunakan perulangan untuk menampilkan angka-angka di antara dua bilangan!

Contoh

Input : angka pertama = 4
angka kedua = 9

Output : 5 6 7 8 9

3. Tulislah kode menggunakan perulangan untuk menampilkan penjumlahan dari 10 bilangan positif pertama!
4. Tulislah kode menggunakan perulangan untuk
5. Tulislah kode menggunakan perulangan untuk mengecek apakah suatu bilangan yang diinputkan adalah bilangan prima atau bukan!

Contoh

Input : angka = 7

Output : angka 7 adalah bilangan prima

6. Tulislah kode menggunakan perulangan untuk menghitung nilai faktorial dari sebuah bilangan!

Contoh

Input : angka = 5

Output : faktorial 5 adalah 120

7. Tulislah kode menggunakan perulangan untuk mencari nilai FPB dari dua bilangan!

Contoh

Input : bilangan pertama = 18
bilangan kedua = 24

Output : FPB dari 18 dan 24 adalah 6

8. Tulislah kode menggunakan perulangan untuk menghitung hasil pangkat dari bilangan!

Contoh

Input : angka = 2
pangkat = 3

Output : hasil 2 pangkat 3 adalah 8

9. Tulislah kode untuk menampilkan ejaan suatu bilangan!

Contoh

Input : 4258

Output : empat dua lima delapan

10. Tulislah kode untuk mencari angka yang pertama kali berulang!

Contoh

Input : 2413648

Output : angka yang pertama kali berulang adalah 4

BAB VII PERULANGAN BERTINGKAT

Seperti pernyataan kondisional, perulangan dapat dibuat secara bertingkat. Standar ANSI (American National Standards Institute) menetapkan perulangan dapat dilakukan dengan kedalaman maksimal 256 perulangan bertingkat.

Perulangan bertingkat terdiri dari perulangan dalam (*inner loop*) dan perulangan luar (*outer loop*). Untuk setiap iterasi pada *outer loop*, maka *inner loop* akan dieksekusi secara menyeluruh satu siklus. Misalnya jumlah iterasi *outer loop* sebanyak 3 kali dan *inner loop* iterasinya sebanyak 5 kali. Untuk setiap eksekusi iterasi pada *outer loop*, *inner loop* dieksekusi sebanyak 5 kali sehingga untuk 3 kali eksekusi iterasi pada *outer loop* maka *inner loop* dieksekusi sebanyak 15 kali (3 x 5). Perulangan dapat dibuat lebih dari dua tingkat. Karena jumlah eksekusi pernyataan akan semakin bertambah sesuai jumlah tingkat perulangan, maka waktu yang dibutuhkan untuk eksekusi perulangan bertingkat akan bertambah dengan signifikan (bertambah secara kuadratik) dengan meningkatnya jumlah tingkatan perulangan.

7.1 Perulangan for di dalam for

Bentuk umum perulangan bertingkat `for` di dalam `for` adalah sebagai berikut:

outer loop / *inner loop*

```
for (inisialisasi_1; kondisi_1; perubahan_nilai_1){
  pernyataan
  for (inisialisasi_2; kondisi_2; perubahan_nilai_2){
     pernyataan
  }
  ...
}
```

Listing 0 digunakan untuk menampilkan dua buah angka dalam variabel `i` dan `j` yang dimulai dari `1` sampai `3`. Dalam kode tersebut iterasi akan dilakukan sebanyak 9 kali (3 iterasi *outer* loop x 3 iterasi *inner loop*). Variabel angka `j` digunakan pada *inner loop* dan variabel i pada *outer loop*.

```
for(int i = 1; i <= 3; i++){
   for(int j = 1; j <= 3; j++){
     cout << i << " " << j << endl;
   }
}
```

Listing 0 perulangan bertingkat `for` di dalam `for`

Detail langkah-langkah dan *output* dalam perulangan bertingkat pada Listing 0 dapat dilihat pada Tabel 0.1. Pada iterasi pertama *outer loop*, nilai `i` adalah `1`. Karena tidak ada pernyataan yang dieksekusi pada *outer loop*, maka iterasi dilanjutkan ke level di dalamnya (*inner loop*). Saat memasuki *inner loop* nilai `i` dan `j` adalah sama-sama `1`. Selanjutnya *inner loop* dieksekusi sampai nilai `j` menjadi 3. Saat variabel `j` mencapai nilai `>3` maka proses iterasi dikeluarkan dari *inner loop* dan kembali ke *outer loop.* P*roses ini diulangi hingga nilai* `i` dan `j` mencapai `3` dan iterasi dihentikan.

Tabel 0.1 detail langkah dalam contoh perulangan bertingkat

i	j	*output*
1	1	1 1
	2	1 2
	3	1 3
2	1	2 1
	2	2 2
	3	2 3
3	1	3 1
	2	3 2
	3	3 3

7.2 Perulangan while di dalam while

Bentuk umum perulangan `while` di dalam `while` adalah sebagai berikut:

outer loop

inner loop

```
while(kondisi_1){
  pernyataan_1
  while(kondisi_2){
     pernyataan_2
     ...
  }
  ...
}
```

Proses iterasi dalam perulangan bertingkat `while` di dalam `while` sama dengan perulangan bertingkat `for` di dalam `for`. *Outer loop* mengecek `kondisi_1`, apabila bernilai benar maka `pernyataan_1` dijalankan dan proses dilanjurkan ke *inner loop*. `kondisi_2` dievaluasi dan apabila bernilai benar maka `pernyataan_2` dijalankan sampai `kondisi_2` bernilai salah dan iterasi pada *inner loop* dihentikan. Proses diulangi hingga `kondisi_1` bernilai salah.

Kode pada Listing 0 dapat diterjemahkan ke dalam perulangan bertingkat `while` di dalam `while` seperti pada Listing 7.2 berikut:

```
int i = 1;
while(i <= 3){
   int j = 1;
   while(j <= 3){
     cout << i << " " << j << endl;
     j++;
   }
   i++;
}
```

Listing 7.2. perulangan bertingkat `while` di dalam `while`

7.3 Perulangan for di dalam while

Bentuk umum perulangan `for` di dalam `while` adalah sebagai berikut:

```
while(kondisi_1){
  pernyataan_1
  for (inisialisasi; kondisi_2; perubahan_nilai){
      pernyataan_2
  }
  ...
}
```

Kode pada Listing 0 dapat diterjemahkan ke dalam perulangan bertingkat `while` di dalam `while` seperti pada Listing 7.2. berikut:

```
int i = 1;
while(i <= 3){
   for(int j = 1; j <= 3; j++){
     cout << i << " " << j << endl;
   }
   i++;
}
```

Listing 7.2 perulangan bertingkat `for` di dalam `while`

Latihan

1. Tulislah kode menggunakan perulangan bertingkat `for` untuk menampilkan tabel perkalian yang berisi perkalian dari 1 sampai 10!
2. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan perhitungan faktorial dari 1! sampai 5!
3. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

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

4. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

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

5. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

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

6. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

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

7. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

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

*****

***

*

8. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan seperti berikut:

```
    *
   * + *
  * + + + *
* * * * * * *
```

9. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan sebagai berikut:

```
5
54
543
5432
54321
```

10. Tulislah kode menggunakan perulangan bertingkat untuk menampilkan tampilan sebagai berikut:

```
1     1
 2   2
  3 3
   4 4
    5
   4 4
  3 3
 2   2
1     1
```

BABVIII FUNGSI

8.1 PENDAHULUAN

Suatu program dalam menangani suatu permasalahan pada dasarnya sangat tergantung pada jenis dan kompleksitas masalah tersebut. Program yang menangani masalah dengan kompleksitas tinggi akan mengandung banyak perintah-perintah yang ada didalamnya. Pada saat pembuatannya dibutuhkan waktu dan ketelitian yang cukup tinggi. Untuk kemudahan dalam pembuatan dan pengecekan maka masalah-masalah tersebut dapat dipisah-pisahkan kedalam persoalan-persoalan yang lebih kecil dan lebih sederhana. Pembagian ini kemudian dinyatakan dalam sebuah bagian program yang disebut dengan sub program.

Pemrograman modular merupakan suatu metode untuk membagi program menjadi sub program yang berupa prosedur (*procedure*) dan fungsi (*function*). Dengan membuat suatu prosedur dan fungsi maka permasalahan didalam program dapat dibagi menjadi bagian-bagian yang lebih kecil. Selain itu pemrogram tidak perlu menuliskan kode-kode yang sama beberapa kali sehingga program menjadi lebih sistematis dan singkat.

Beberapa Bahasa pemrograman memiliki syntaks yang berbeda-beda untuk menuliskan sebuah prosedur dan fungsi. Di dalam Bahasa C sendiri tidak tertulis secara explisit apakah suatu subprogram tersebut merupakan sebuah prosedur atau fungsi. Kita dapat memmbedakan sebuah sub program tersebut merupakan sebuah fungsi atau prosedur setelah memahamai pengertian dari jenis sub program tersebut.

Pemrograman c++ merupakan pemrograman yang mendukung pembuatan program dengan system modular. Dalam pembuatan program dalam Bahasa C terdapat dua kelompok sub program (fungsi) yaitu fungsi yang sudah ada dalam Bahasa C++ dan fungsi yang dibuat sendiri oleh pemrogram. Fungsi-fungsi yang sudah ada dan disediakan oleh Bahasa C++ terdapat dalam kamus yang disebut dengan library.

Prosedur atau fungsi adalah suatu program yang terspisah dalam blok sendiri yang berfungsi sebagai sub program. Prosedur atau fungsi biasanya berupa sub program yang digunakan untuk menyelesaikan tugas tertentum sebagai bagian dari program

utama seperti menampilkan data atau membaca masukan data. Sub program dalam pemrograman C++ tersedia dalam bentuk prosedur dan fungsi tetapi keduanya lebih banyak disebut sebagai fungsi karena pada dasarnya fungsi maupun prosedur memiliki pengertian yang sama. Perbedaan mendasar antara prosedur dan fungsi adalah prosedur tidak memiliki nilai pengembalian, sedangkan fungsi memiliki nilai pengembalian.

8.2 Fungsi Pada Library C++

Bahasa C++ memiliki berbagai fungsi yang telah didefinisikan dan di sematkan di dalam program. Fungsi-funsgi tersebut sudah disiap digunakan oleh pemrogram. Fungsi-fungsi bawaan dari Bahasa pemrograman C++ ini berada pada library. Fungsi-fungsi tersebut disimpan dalam sebuah file yang berekstensi .h (*.h) atau yang disebut juga dengan file header. Contoh file header yang umum digunakan adalah stdio.h (standart input ouput). Sesuai dengan namanya, file stdio.h ini adalah file header yang didalamnya terdapat fungsi-fungsi umum untuk operasi input dan out yang digunakan oleh pemrogram. Salah satu fungsi yang ada dalam file ini adalah printf() yaitu fungsi yang digunakan untuk mencetak ke layar monitor. Selain stdio.h Bahasa C++ masih menyediakan fungsi-fungsi yang lain yang terdapat dalam file header yang berbeda-beda. Tabel 9.1 berikut ini adalah daftar file header pada C++ dan kegunaanya.

Tabel 8.1. File Header pada C++

File Header	Kegunaan
Assert.h	Untuk melakukan diagnose program
ctype.h	untuk penanganan character
errno.h	untuk menangani error
float.h	untuk menangani type floating point
iso646	untuk alternative ejaan operator
Limits.h	Untuk menentukan batas tipe integral
Locale.h	Berfungsi untuk seting lokalisasi seperti format tanggal mata, mata uang, dan lainnya
Math.h	Untuk melakukan komputasi matematik secara umum
Setjmp.h	Untuk melakukan bypass pemanggilan fungsi secara umum
Signal.h	Digunakan untuk menginformasikan proses yang sedang berjalan dengan suatu signal
Stdio.h	Untuk operasi standart input dan output
Stdlib.h	Untuk fungsi-fungsi umum termasuk yang melibatkan memori dinamis seperti membangkitkan bilangan acak.
String.h	Untuk operasi-operasi umum pada tipe data string
Time.h	Untuk memanipulasi tangal dan waktu
Uchar.h	Untuk operasi karakter yang support pada 16 bit dan 32 bit

Selain file-file header yang tersebut pada tabel 8.1 diatas, masih banyak lagi file header yang lain yang dapat digunakan oleh pemrogram.

Untuk dapat menggunakan file header yang berisi fungsi-fungsi pada library seorang pemrogram harus menyertakanya dengan ditandai dengan symbol # dan pernyataan include di depan file header tersebut. Sebagai contoh #include <stdio.h>. Pernyataan ini menunjukkan bahwa program yang sedang ditulis menyertakan file header stdio.h yang didalamnya berisi fungsi-fungsi yang siap digunakan seperti printf(), scanf(), dan lain sebagainya. Listing program berikut ini adalah contoh penggunaan file header dan fungsi yang ada pada file header tersebut.

```
#include <iostream.h>    /*menyertakan file header iostream.h*/
#include <conio.h>       /*menyertakan file header conio.h*/
#include <math.h>        /*menyertakan file header math.h*/

main(){
 int a = 25;
 int b = 3;
 int c = 2;
 int d = 90;
 float pangkat, akar, sin ,cos;
 akar = sqrt(a);
          /*sqrt() adalah fungsi yang tersedia pada header math.h*/
 cout<<"Akar dari "<<a<<" adalah"<<akar<<endl;
         /* cout adalah perintah yang ada pada header iostream.h */
 pangkat = pow(b,c);
         /* pow() adalah fungsi yang ada pada header math.h */
 cout<<b<<" pangkat "<<c<<" adalah"<<ipangkat<<endl;

 sin = sin(d);
         /* sin() adalah fungsi yang ada pada header math.h */
 cout<<"sin sudut "<<a<<" derajat adalah"<<isin<<endl;

 cos = cos(d);
         /* cos() adalah fungsi yang ada pada header math.h */
```

```
 cout<<"cos sudut "<<a<<" derajat adalah"<<icos<<endl;

getch();   /*fungsi yang ada pada header conio.h*/
}
```

8.3 Fungsi yang Dibuat User (User Defined Function)

Fungsi yang dibuat oleh user adalah fungsi yang dibuat sendiri oleh pemrogram. Fungsi ini dibuat untuk memudahkan pemrogram dalam membagi program manjadi sub program-sub program. Ketika pemrogram akan membuat sebuah fungsi, maka terdapat beberapa hal yang harus diperhatikan agar program yang dibuat memiliki struktur yang baik. Ciri-ciri dari fungsi yang baik adalah sebagai berikut:

1. Sebuah fungsi sebaiknya dibuat hanya focus pada satu tujuan atau hanya memecahkan sebuah masalah dalam program dan tidak bercampur dengan tujuan lain. Dengan demikian fungsi tersebut akan lebih mudah dipahami.
2. Suatu fungsi tidak bergantung pada fungsi yang lain atau bersifat mandiri. Sebuah fungsi yang dibuat secara mandiri tanpa bergantung pada fungsi yang lain akan dapat dijalankan dan diuji tanpa harus menunggu bagian yang lain selesai. Selain itu, parameter-parameter yang digunakan dalam suatu fungsi tidak akan mempengaruhi parameter yang digunakan pada bagian yang lain.
3. Suatu fungsi sebaiknya berukuran kecil. Ukuran disini menunjukan panjangnya kode program dari sungsi tersebut. Hal ini akan menjadikan suatu fungsi mudah untuk dipahami dan bila terjadi kesalahan akan lebih mudah dalam memperbaiki.

8.4 Struktur Fungsi

Untuk membuat sebuah fungsi maka harus dipahami bagaimana struktur penulisan fungsi secara umum. Bentuk umum penulisan fungsi terbagi menjadi dua yaitu fungsi dengan nilai pengembalian dan fungsi tanpa nilai pengembalian atau yang disebut juga prosedur.

Struktur fungsi tanpa nilai kembalian atau sering dsebut juga sebagai prosedur dapat dituliskan dengan struktur sebagai berikut:

```
void nama_fungsi (parameter1, parameter2 ,…)
    {
       Pernyataan bagian utama fungsi
    }
```

Keterangan:

1. `Void` adalah pernyatan bahwa bagian tersebut adalah fungsi tanpa nilai pengembalian
2. `Nama_fungsi` adalah identifier atau nama sebagai identitas dari suatu fungsi.
3. `Parameter` adalah argument yang berisi nilai yang digunaan saat pemanggilan fungsi. Parameter tersebut bersifat local, artinya hanya dapat digunakan dalam ruang lingkup fungsi tersebut dan tidak dapat digunakan di luar fungsi.
4. `Pernyataan` adalah kode-kode program yang menjadi bagian dari fungsi.

Untuk lebih jelas dalam memahami bagaimana fungsi tanpa nilai pengembalian ini digunakan, listing program berikut ini adalah contoh penggunaan fungsi tanpa nilai pengembalian.

```
#include <iostream>
using namespace std;

void salam(){ // Fuction tanpa parameter
    cout<<"Selamat datang "<<endl;
    cout<<"Ini di tulis dari fungsi"<<endl;
    cout<<"Terimakasih"<<endl;
}

int main(){
    salam(); //pemanggilan fungsi
    return 0;
}
```

Pada listing program diatas, fungsi `salam()` merupakan fungsi tanpa nilai kembalian atau disebut juga sebagai prosedur. Fungsi `salam` merupakan fungsi tanpa parameter yang digunakan untuk menampilkan pesan salam. Contoh program lainya yang menggunakan fungsi tanpa nilai pengembalian adalah sebagai berikut:

```
#include <iostream>
using namespace std;

void LuasPersegi()
{
 int panjang,lebar;

 cout << "Masukkan panjang: "; cin >> panjang;
 cout << "Masukkan lebar: "; cin >> lebar;

 cout<<"Luas persegi: " << panjang*lebar;
}

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

Fungsi diatas merupakan fungsi tanpa nilai pengembalian. Struktur fungsi dengan nilai pengembalian ditulis dengan cara yang berbeda. Penulisan struktur fungsi dengan nilai kembalian adalah sebagai sebagai berikut:

```
Tipe_data nama_fungsi(parameter1, parameter 2, …)
    {
       Pernyataan bagian utama fungsi
    }
```

Keterangan:

1. `Tipe data` merupakan tipe data untuk nilai kembalian yang dihasilkan setelah fungsi tersebut dijalankan.
2. `Nama_fungsi` adalah identifier atau nama sebagai identitas dari suatu fungsi.
3. `Parameter` adalah argument yang berisi nilai yang digunaan saat pemanggilan fungsi. Parameter tersebut bersifat local, artinya hanya dapat digunakan dalam ruang lingkup fungsi tersebut dan tidak dapat digunakan di luar fungsi.
4. `Pernyataan` adalah kode-kode program yang menjadi bagian dari fungsi.

Untuk lebih lebih bagaimana menggunakan fungsi tersebut didalam program, listing program berikut ini adalah contoh program.

```
#include <iostream>
using namespace std;

int LuasPersegi (int p, int l){
    int luas;
    luas = p*l;
    return luas;
}

int main()
{
    int a,b;

    cout << "masukkan panjang: "; cin >> a;
    cout << "masukkan lebar: "; cin >> b;

    cout << "Luas Persegi: " << LuasPersegi(a,b);
    return 0;
}
```

Penulisan fungsi pada kedua contoh diatas adalah sebelum fungsi utama (fungsi main). Penulisan badan fungsi secara lengkap dapat ditulis setelah fungsi utama. Jika fungsi lengkap ditulis setelah fungsi utama, maka perlu dibuat prototype fungsi sebelum fungsi utama karena compiler c++ membaca program berdasar urutan dari baris

program yang dibuat. Prototipe fungsi adalah pernyataan deklarasi fungsi dengan menyebutkan nama fungsi dan parameternya tanpa menuliskan badan fungsi. Prototipe ini berfungsi untuk menyatakan bahwa suatu fungsi tersbut ada sehingga compiler dapat mengenalinya ketika terdapat pemanggilan pada program utama (fungsi main). Contoh program diatas dapat ditulis dengan prototype fungsi sebagai berikut:

```
#include <iostream>
using namespace std;
int LuasPersegi (int p, int l); /* prototype fungsi */

int main()
{
    int a,b;

    cout << "masukkan panjang: "; cin >> a;
    cout << "masukkan lebar: "; cin >> b;

    cout << "Luas Persegi: " << LuasPersegi(a,b);
    return 0;
}

int LuasPersegi (int p, int l){
    int luas;
    luas = p*l;
    return luas;
}
```

8.5 Parameter pada Fungsi

Pada beberapa contoh diatas baik fungsi yang memiliki pengembalian nilai maupun tidak, terdapat fungsi yang menggunakan parameter atau tidak. Parameter adalah tempat menyimpan data sementara yang berada pada memori atau disebut juga sebagai variabel. Parameter digunakan untuk pemberian nilai pada saat suatu fungsi dipanggil. Parameter terletak pada bagian diantara tanda (dan) setelah identitas penamaan fungsi. Sebuah fungsi bisa memiliki satu parameter, dua parameter atau lebih,

bahkan sebuah mungkin juga tidak memiliki parameter. Jika parameter fungsi memiliki lebih dari satu parameter maka dipisahkan dengan tanda koma (,).

Dalam melakukan operasi, sebuah fungsi biasanya membutuhkan data sebagai data awal atau data input. Sebagai contoh fungsi yang bertugas untuk menghitung luas persgi panjang, fungsi tersebut membutuhkan data awal berupa panjang dan lebar. Untuk menampung data lebar dan panjang maka diperlukan parameter gungsi. Berikut ini adalah contoh fungsi dengan 3 parameter.

```
#include <iostream>
using namespace std;
int VolumeBalok (int p, int l, int T); /* p, l,dan t adalahparameter */

int main()
{
    int a,b,c;
    cout << "masukkan panjang: "; cin >> a;
    cout << "masukkan lebar: "; cin >> b;
    cout << "masukkan Tinggi: "; cin >> c;


    cout << "Volume Balok : " << VolumeBalok(a,b,c);
    return 0;
}

int LuasPersegi (int p, int l, int t){
    int volume;
    luas = p*l*t;
    return volume;
}
```

Fungsi `LuasPersegi (int p, int l)` merupakan fungsi dengan dua paremeter sedangkan fungsi `VolumeBalok(int p, int l, int t)` merupakan fungsi dengan 3 parameter. Jumlah parameter dari sebuah fungsi ini sangat bergantung pada kebutuhan dari fungsi tersebut. Parameter yang yang menyertai fungsi ini disebut dengan parameter fungsi atau

parameter frormal yaitu parameter yang berada pada deklarasi fungsi. Parameter ini menentukan bagaimana fungsi tersebut dipanggil. Ketika memanggil fungsi, perlu menyebutkan suatu nilai untuk mewakili parameter-parameter fungsi tersebut. sebagai conton unutk memanggil fungsi `LuasPersegi (int p, int l)` dapat dilakukan dengan cara `LuasPersegi(10,4)` atau dengan menggunakan variable lain seperti `LuasPersgi(a,b)`. nilai yang digunakan untuk memanggil fungsi ini disebut dengan argument atau parameter actual.

Parameter fungsi menentukan bagaimana sebuah fungsi dipanggil. Terdapat dua cara dalam pemanggilan fungsi yaitu ***passing by value*** dan *passing by refennce*. Penjelasan beriku ini adalah bagaimana melakukan pemanggilan fungsi dengan pasing by value dan passing by reference serta perbedaan dari kedua pemanggilan tersebut.

Passing by Value

Passing by Value adalah suatu cara pemanggiln fungsi dengan memberikan argument pad fungsi parameter. Argument ini dapat berupa nilia secara langsung atau variable yang memiliki nilai. Berikut ini adalah contoh penggunaan passing by value.

```
#include <iostream>
using namespace std;

int kali(int a, int b){
    return a*b;
}

int main () {
    int x,y;
    x=4;  y=8;
    cout<<kali(4,6)<<endl; //pemangilan dengan passing by value
    cout<<kali(12,12)<<endl; //pemanggilan dengan passing by value
    cout<<kali(x,y)<<endl; //pemanggilan dengan passing by value

    return 0;
```

```
}
```

Pada contoh program diatas, fungsi tambah merupakan fungsi dengan dua buah parameter. Pada program tersebut terlihat bahwa fungsi tambah di panggil berulang sebanyak tiga kali dengan argument yang berbeda-beda. Pada saat pemanggilan fungi, nilai pada argument tersebut akan dikirim sebagai nilai untuk parameter fungsi. Contoh lain dari pemanggilan dengan passing by value adalah sebagai berikut:

```
#include <iostream>
using namespace std;

void tukar(int a, int b);



int main(){
    int a=5;
    int b=10;

    cout<<"a sebelum ditukar = "<<a<<endl;
    cout<<"b sebelum ditukar= "<<b<<endl;

    tukar(a,b);

    cout<<"a setelah fungsi tukar = "<<a<<endl;
    cout<<"b setelah fungsi tukar = "<<b<<endl;

    return 0;
}

void tukar(int a,int b){
    int temp;
    temp=a;
    a=b;
    b=temp;
    cout<<"nilai a setelah ditukar pada fungsi="<<a<<endl;
    cout<<"nilai b setelah ditukar  pada fungi="<<b<<endl;

}
```

Pada program diatas, terdapat fungsi yang bertugas untuk menukar nilai pada variabel a dan variabel b. parameter fungsi dipanggil dengan cara passing by value. Perubahan yang terjadi pada parameter fungsi tidak akan berpengaruh pada argument yang digunakan karena masing variable baik argument maupun parameter fungsi adalah variable yang berbeda dan berdiri sendiri. Program diatas apabila dijalan akan menghasilkan keluaran sebagai berikut:

```
a sebelum ditukar = 5
b sebelum ditukar= 10
nilai a setelah ditukar pada fungsi=10
nilai b setelah ditukar  pada fungi=5
a setelah fungsi tukar = 5
b setelah fungsi tukar = 10
```

Passing by Reference

Passing by reference adalah pemanggilan fungsi dengan menggunakan variable referensi untuk parameter fungsi sehingga perubahan nilai yang terjadi pada parameter akan mempengaruhi nilai argumenya. Hal ini berbeda dengan pemanggilan passing by value. Antara parameter dengan argumen saling terkait yang di sebut dengan referensi. Operasi yang terjadi pada sebuah fungsi dan merubah nilai dari parameter akan mengakibatkan juga terjadi perubahan nilai pada argumen yang digunakan. Fungsi dengan model ini biasanya digunakan untuk fungsi yang memilikiki nilai kembalian lebih dari satu. Sebagai contoh fungsi untuk menukar dua buah bilangan, fungsi ini diharapkan mengembalikasi dua buah nilai yang telah di tukar.

Fungsi dengan pemanggilan passing by reference memiliki penulisan yang berbeda. Penulisan fungsi ini perlu ditmbahkan tanda & pada awal nama parameter misalnya `fungsi tukar(int a, int b)` maka ditulis menjadi fungsi `tukar(int &a, int &b)`. Contoh program berikut ini adalah contoh program menukar dua buah bilangan yang dilakukan dengan fungsi tukar.

```
#include <iostream>
using namespace std;

void tukar(int &x, int &y); /*parameter untuk passing by reference*/
```

```
int main(){
    int a=5;
    int b=10;

    cout<<"a sebelum ditukar = "<<a<<endl;
    cout<<"b sebelum ditukar= "<<b<<endl;

    tukar(a,b);

    cout<<"a setelah fungsi tukar = "<<a<<endl;
    cout<<"b setelah fungsi tukar = "<<b<<endl;

    return 0;
}

void tukar(int &x,int &y){
    int temp;
    temp=x;
    x=y;
    y=temp;
       }
```

Fungsi pada program diatas di deklarasikan dengan menambahkan & diawal nama parameter yaitu `void tukar(int &x, int &y)`. Pada program diatas terlihat bahwa variable yang digunakan untuk parameter fungsi adalah x dan y. Pada saat pemanggilan fungsi menggunakan argumen berupa variable a dan b. nama variable a dan b Nampak jelas berbeda dengan variable x dan y. setelah fungsi dipanggil, nilai x dan y akan berubah. Hal ini mempengaruhi nilai a dan b sebagai argument yang digunakan untuk memanggil fungsi tersebut karena antara a dan b sebagai argument saling terkait dengan x dan y sebagai parameter fungsi. Hasil keluaran dari program diatas adalah seperti berikut:

```
a sebelum ditukar = 5
b sebelum ditukar= 10
a setelah fungsi tukar = 10
b setelah fungsi tukar = 5
```

Latihan

1. Dengan menggunakan file header math.h buatlah program kalkulator sederhana yang dapat melakukan operasi penambahan, pengurangan, perkalian, pembagian, perpangkatan, dan akar pangkat. Operasi-operasi tersebut menggunakan fungsi-fungsi yang terdapat pada header math.h!
2. Buatlah fungsi untuk menghitung jumlah suatu bilangan mulai dari nilai tertentu sampai dengan nilai tertentu. Program dibuat dengan fungsi tanpa nilai kembalian dan fungsi dengan nilai kembalian. Misalnya jumlah bilangan dari 5 sampai 10 maka hasilnya adalah 5+6+7+8+9+10.

BAB IX FUNGSI REKURSIF

9.1 Pendahuluan

Sub program dalam Bahasa pemrograman c++ disebut dengan fungsi. Secara garis besar fungsi ini terbagi menjadi dua yaitu fungsi yang tidak memiliki nilai pengembalian (disebut juga sebagai prosedur) dan fungsi yang memiliki nilai pengembalian. Kedua jenis fungsi tersebut telah dibahas pada bab sebelumnya. Fungsi seringkali digunakan untuk melakukan tugas yang berulang-ulang. Fungsi yang didalamnya memiliki operasi yang di ulang-ulang terhadap sekelompok instruksi, biasanya disebut dengan iterative. Cara menerapkan fungsi iteratif adalah sebagaimana fungsi yang telah dijelaskan sebelumnya.

Selain fungsi iteratif sebagamana dua jenis fungsi tersebut yang telah dijelaskan sebelumnya, terdapat satu model fungsi yang sering digunakan dalam program yaitu fungsi rekursif. Fungsi rekursif ini adalah fungsi yang didalam terdapat perintah untuk memanggil dirinya sendiri. Dengan memanggil dirinya sendiri, maka secara otomatis fungsi ini menjadi berulang terus menerus sampai pada satu titik yang ditentukan untuk berhenti. Lebih detil bagaimana konsep dan cara pembuatan fungsi rekursif ini akan dibahas pada bagian berikut ini.

9.2 Pengertian Fungsi Rekursif

Fungsi rekursif adalah sebuah fungsi yang didalamnya terdapat perintah untuk memanggil dirinya sendiri atau sebuah fungsi yang memanggil fungsi yang lain, dan fungsi lain tersbut terdapat perintah yang memanggil fungsi dirinya sendiri. Sebagai contoh sebuah fungsi `f()` disebut sebagai fungsi rekursif apabila memanggil lain missal fungsi `g()` dan didalam fungsi G tersebut terdpat perintah yang memanggil fungsi `f()`.

Pada fungsi biasa, proses berulang biasanya dinyatakan dengan iteratif yang dinyatakan dengan perintah perulangan seperti for atau while. Fungsi rekursif dapat dipandang sebagai sebuah operator. Sebagai contoh dalam kasus perhitungan nilai faktorial secara iteratif n faktorial didesinisikan sebagai:

$$n!=1 \text{ X } 2 \text{ X } 3 \text{ X } 4 \ldots. \text{ X } N = \prod_{i=1}^{n} i$$

n faktorial secara rekursif dapat didefinisikan sebagai berikut:

$$n!= n * (n-1)!$$

$$(n-1)!=n-1 * (n-2)!$$

(n-2)!=n-2 * (n-3)!

….

1!=1

0!=1

maka diperoleh n!=n*n-1*n-2*n-3 …. 2 * 1 sehingga secara ringkas dapat didefinisikan sebagai:

n!=1 untuk n=0 atau n=1

n!=n * (n-1)! Untuk n>2.

Dapat dilihat pada posisi sebelah kanan masih terdapat operator yang sama dengan sebelah kiri yaitu faktorial. Dengan demikian n faktorial baru dapat diperoleh hasilnya apabila n-1 faktorial telah diperoleh dan seterusnya. Pencarian ini berhenti bila sudah sampai pada nilai konstan yaitu telah mencapai n=0 atau n=1.

9.3 Karakteristik Fungsi Rekursif

Selain memiliki ciri pemanggilan fungsi terhadap dirinya sendiri, fungsi rekursif biasanya digunakan untuk menyelesaikan permasalahan-permasalahan yang memiliki beberapa karakteristik diantaranya adalah sebagari berikut:

1. Kasus sederhana dari permasalahan memiliki jawaban langsung yang disebut dengan base case. Pada contoh kasus factorial, base case dari factorial tersebut adalah 0!=1.
2. Kasus yang lebih komplek dapat didefinisikan secara sama tetapi dalam ukuran yang lebih kecil yang disebut dengan recursive case. Pada contoh kasus factorial, rekursif case adalah n!=n * (n-1)!
3. Dengan menerapkan karakteristik dari yang kedua yaitu bagian recursif case, apabila dilakukan secara berulang maka akan semakin mendekati dan sampai pada base case. Sebagai contoh adalah n! -> (n-1)! -> (n-2)!-> (n-3)! -> … ->1! -> 0!.
4. Base case merupakan stoping point yang artinya sebagai titik berhentinya perulangan dalam memanggil dirinya sendiri. Pada contoh kasus factorial, pemanggilan kembali dirinya sendiri kaan berhenti jika sudah sampai pada factorial(0).

Dengan memperhatikan karakteristik dari permasalahan pada fungsi rekursif maka secara umum fungsi rekursif memiliki bentuk sebagai berikut:

```
If kasus_sederhana
Selesaikan dengan jawaban langsung
 Else
Selesaikan dengan menggunakan rekursif
```

Dari bentuk diatas dapat dijelaskan bahwa cabang if merupakan base case sedangkan cabang else merupakan rekursif case. Cabang recursif case merupakan perulangan yang terdapat bagian pemanggilan dirinya sendiri untuk menyederhanakan permaslahan dan base case berfungsi sebagai stoping poin. Agar fungsi rekursif ini dapat berheti maka setiap pemanggilanya pada rekursif case harus mendekati nilai basecase.

Untuk melihat perbadingan antara fungsi yang menggunakan iteratif dan fungsi rekursif, fungsi dengan iteratif telah dibahas pada bagian sebelumnya dan berikut ini adalah program factorial dengan fungsi rekursif. Berikut ini adalah contoh program factorial dengan menggunakan fungsi iterative dan menggunakan fungsi rekursif.

```
#include <iostream>
using namespace std;

int faktorial(int n);

int main(){
    int n=6;
    cout<<"Nilai factorial”<<n<<” adalah = "<<factorial(n)<<endl;

    return 0;
}

void faktorial(int n){
    int I, kali=1;
    for (i=n; i>1; --i)
      kali=kali*i;
     return kali;
       }
```

Program diatas adalah program menghitung nilai factorial dengan menggunakan fungsi secara iterative. Apabila digunakan fungsi rekursif maka penelisan program akan menjadi seperti berikut ini.

```
#include <iostream>
using namespace std;

int faktorial(int n);

int main(){
    int n=6;
    cout<<"Nilai factorial”<<n<<” adalah = "<<factorial(n)<<endl;

    return 0;
}

void faktorial(int n){
      if (n==0)
        return 1;
      else
        return n*factorial(n-1);
       }
```

Pada program fungsi dengan rekursif diatas, terlihat bahwa pada pernyatan else terdapat perintah `factorial(n-1)`, bagian ini yang berfungsi untuk memanggil dirinya sendiri. Pada pernyataan if tidak terdapat pemanggilan dirinya sendiri. Kondisi ini menjadikan pernyataan if sebagai stoping poin. Pada pemanggilan fungsi dirinya sendiri, nilai n akan berkurang 1 dan terjadi secara berulang-ulang sehingga nilai n akan sampai dan kondisi n=0 sehingga pemanggilan terhadap dirinya berhenti.

9.4 Latihan

1. Buatlah fungsi rekursif untuk menghitung nilai deret Fibonacci.
2. Buatlah fungsi rekursif untuk menghitung jumlah akumulasi pengunjung suatu museum dalam beberapa bulan terakhir, misalnya 10 bulan. Fungsi rekursif digunakan untuk

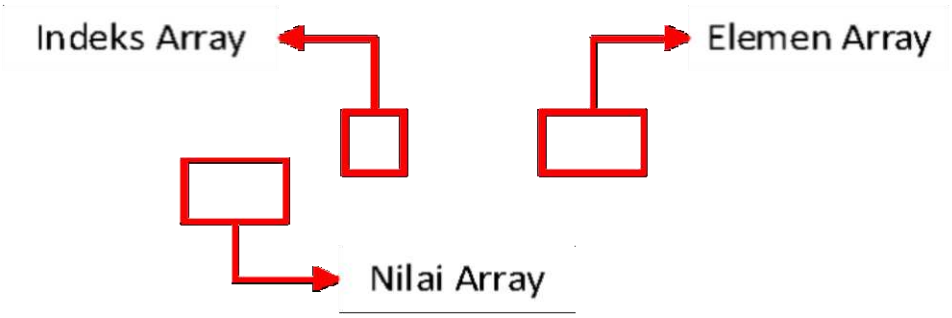
menghitung jumlah akumulasi pengunjung pada bulan-bulan sebelumnya dari 10 bulan terakhir.

BAB X. ARRAY

10. 1 Definisi Array

Array atau larik adalah sekumpulan variable yang bertipe sama dan diacu dengan menggunakan satu nama. Array memiliki variable di dalamnya yang disebut dengan elemen array yang dapat diakses melalui indeks array. Array dapat kita gunakan untuk menggantikan penggunaan banyak variable seperti ketika kita ingin menampung nilai, sebagai contoh: nilai1, nilai2, nilai3, nilai4, nilai5, dst. Dengan menggunakan array kita tidak perlu melakukan banyak deklarasi variable. Kita cukup mendeklarasikan sebuah array untuk menampung nilai-nilai tersebut

Indeks array selalu bertipe int dan dimulai dari 0, bukan 1. Indeks array berfungsi untuk menunjukkan posisi dari masing-masing elemen dari array. Gambar 10.1 menunjukkan ilustrasi dari sebuah array yang kita sebut dengan nama array a yang berisi 6 elemen dengan tipe int.



Gambar 0.1 Ilustrasi Array

Gambar di atas menjelaskan mengenai ilustrasi dari elemen array, indeks array, dan nilai dari array a. a[0], a[1], a[2], a[3], a[4], dan a[5] disebut sebagai elemen array a. 0, 1, 2, 3, 4, dan 5 disebut sebagai indeks array a. 100, 200, 300, 400, 500, dan 600 disebut sebagai nilai-nilai yang berada pada setiap elemen array a.

10.2. Deklarasi Array 1 Dimensi

Array dideklarasikan dengan menggunakan tanda kurung siku (`[ ]`). Untuk mendeklarasikan array 1 dimensi, kita dapat menggunakan bentuk umum penulisan sebagai berikut:

```
TipeData NamaArray [JumlahElemen];
```

```
/* contoh penulisan array a yang memiliki 5 elemen dengan tipe int */
int a[4];
```

Pada contoh di atas kita melakukan alokasi 5 alamat pada memori untuk tipe data int. Perlu kita ketahui bahwa pada contoh di atas kita hanya mendeklarasikan array dengan nama 'a' saja. Nilai dari array tersebut belum kita inisialisasi atau kita isi. Pada contoh di atas, masing-masing nilai yang akan dimasukkan akan diberi nomor elemen dengan indeks dimulai dari 0 hingga 4, yaitu a[0], a[1], a[2], dan a[3]. Perlu kita ingat bahwa indeks array selalu dimulai dari angka 0. Tipe data array tidak harus selalu bertipe int. Kita dapat menggunakan tipe data lainnya seperti float atau double. Untuk tipe data huruf (char, string, dll) akan kita bahas pada bab selanjutnya.

10.3. Pengisian Nilai Pada Array

Untuk mengisi nilai pada sebuah array, kita dapat memasukkan nilai array satu per satu pada masing-masing elemen array atau dengan memasukkan nilai array secara bersamaan. Kode program berikut merupakan bentuk umum untuk mengisi nilai array satu per satu.

```
NamaArray [indeks] = NilaiArray;
```

```
/* contoh inisialisasi nilai array secara satu per satu */

int a[5];               //deklarasi array a dengan tipe int
int a[0]=100;           //mengisi indeks 0 dengan nilai 100
int a[1]=200;           //mengisi indeks 1 dengan nilai 200
int a[2]=300;           //mengisi indeks 2 dengan nilai 300
int a[3]=400;           //mengisi indeks 3 dengan nilai 400
int a[4]=500;           //mengisi indeks 4 dengan nilai 500
```

Apabila kita ingin mendeklarasikan array dan menginisialisasi nilai-nilai dari elemen array secara bersamaan, kita dapat menggunakan bentuk umum sebagai berikut:

```
TipeData NamaArray[JumlahElemen]={nilai1, nilai2, dst};
```

```
/* contoh deklarasi & inisialisasi nilai array secara bersamaan */
int a[5]={100, 200, 300, 400, 500};
```

Kode di atas mendeklarasikan array a yang bertipe int dan memiliki 5 elemen array. Kelima elemen tersebut adalah a[0] yang memiliki nilai 100, a[1] yang memiliki nilai 200, a[2] yang memiliki nilai 300, a[3] yang memiliki nilai 400, dan a[4] yang memiliki nilai 500.

Kita juga dapat melakukan inisialisasi array tanpa menyertakan ukuran atau jumlah elemen dari array, seperti berikut:

TipeData NamaArray [] = {nilai1, nilai2, nilai3, dst};

```
/* contoh penulisan array a tanpa menuliskan jumlah elemen array */
int b[]={100, 200};
```

Pada contoh kode di atas, kita tidak menuliskan jumlah elemen dari array b. Pada kondisi seperti itu, jumlah elemen dari array akan ditentukan dengan melihat banyaknya elemen yang digunakan untuk inisialisasi. Dari contoh di atas, array b akan memiliki 2 elemen array dengan nilai 100 akan ditempatkan pada elemen b[0] dan nilai 200 akan ditempatkan pada elemen b[1].

Ketika array yang dideklarasikan memiliki sejumlah elemen tertentu dan jumlah nilai pada inisialisasi tidak sama dengan jumlah elemen, maka elemen-elemen sisa yang tidak diinisialisasi umumnya diisi dengan nilai 0. Kita juga dapat menginisialisasi sebuah array dengan seluruh elemen array bernilai 0.

```
int c[5]={100, 200};          //array c berisi {100, 200, 0, 0, 0}
int d[5]={0};                 //array d berisi {0, 0, 0, 0, 0}
```

Program di atas mendeklarasikan array c yang memiliki 5 elemen sekaligus menginisialisasi nilai dari array c. Nilai yang diinisialisasi sebanyak 2, yaitu: a[0] dengan nilai 100 dan a[1] dengan nilai 200. Ketiga elemen array lainnya yang tidak diinisialisasi (a[2], a[3], dan a[4]) diisi dengan nilai 0.

Pada situasi tertentu kita perlu untuk memasukkan nilai array yang baru ke dalam sebuah elemen array yang telah memiliki nilai. Pada kode program di atas, array a yang bertipe int pada awalnya memiliki nilai 100, 200, 300, 400, dan 500. Kemudian kita memasukkan nilai 900 ke dalam indeks 3 dari array a. Elemen array a[3] akan memiliki nilai baru 900 sehingga array a akan memiliki nilai 100, 200, 300, 900, dan 500.

```
int a[5]={100, 200, 300, 400, 500}; //inisialisasi nilai array a
a[3] = 900;             //mengisi nilai ke indeks 3 dari array a
                        // array a akan berisi {100, 200, 300, 900, 500}
```

Terkadang kita akan berada pada situasi dimana kita diminta untuk memasukkan nilai dari sebuah array yang memiliki banyak elemen. Apabila kita menuliskan program tersebut dengan kode di atas, maka kode yang ditulis akan sangat panjang dan rentan terhadap kemungkinan terjadinya error. Untuk mengatasi hal tersebut kita dapat menggunakan loop atau

perulangan ketika melakukan inisialisasi array. Pada contoh kode program di bawah, kita melakukan deklarasi array 5 yang bertipe int dan memiliki 5 elemen array. Inisialisasi nilai array dilakukan sesuai dengan kondisi perulangan yang diberikan.

```
int a[5];                              //deklarasi array a
cout<<"Masukkan nilai ke dalam array: "<<endl;
for(int i=0; i<5 ; i++)       //loop untuk memasukkan nilai ke dalam array
      {
      cout<<"a["<<i<<"]: ";
      cin>>a[i];
      }
return 0;
}
```

10.4. Pembacaan Array 1 Dimensi

Untuk membaca nilai dari sebuah elemen array, kita dapat menggunakan bentuk umum penulisan sebagai berikut:

`NamaArray [IndeksArray];`

Kode program di bawah akan membaca dan menampilkan elemen array a yang berada pada indeks 0, 1, dan 2.

```
/* contoh sederhana membaca nilai array a */
cout<<a[0];
cout<<a[1];
cout<<a[2];
```

Contoh Program

```
/* contoh program membaca nilai array */
#include <iostream>
using namespace std;

int main()
{
int a[5] = {100, 200, 300, 400, 500};          //deklarasi array a
int b[] = {600, 700, 800};                     //deklarasi array b
```

```
cout<<"Nilai elemen 1 array a adalah: "<<a[0]<<endl;
cout<<"Nilai elemen 2 array a adalah: "<<a[1]<<endl;
cout<<"Nilai elemen 3 array a adalah: "<<a[2]<<endl;
cout<<"Nilai elemen 4 array a adalah: "<<a[3]<<endl;
cout<<"Nilai elemen 5 array a adalah: "<<a[4]<<endl<<endl;
cout<<"Nilai indeks 0 array b adalah: "<<b[0]<<endl;
cout<<"Nilai indeks 1 array b adalah: "<<b[1]<<endl;
cout<<"Nilai indeks 2 array b adalah: "<<b[2]<<endl;

return 0;
}
```

Hasil program:

```
Nilai elemen 1 array a adalah: 100
Nilai elemen 2 array a adalah: 200
Nilai elemen 3 array a adalah: 300
Nilai elemen 4 array a adalah: 400
Nilai elemen 5 array a adalah: 500

Nilai indeks 0 array b adalah: 600
Nilai indeks 1 array b adalah: 700
Nilai indeks 2 array b adalah: 800
```

Dari contoh program di atas dapat kita lihat bahwa pemanggilan nilai dari setiap elemen array menggunakan indeks dari masing-masing elemen. Kita juga dapat menggunakan loop atau perulangan untuk mengakses nilai array. Dengan menggunakan perulangan untuk mengakses nilai pada sebuah array untuk membuat program menjadi lebih singkat dan sederhana, sebagaimana pada contoh berikut:

Contoh Program

```
/* contoh program membaca nilai array menggunakan for loop */
#include <iostream>
using namespace std;

int main()
{
int a[5] = {100, 200, 300, 400, 500};          //deklarasi array a
```

```
int b[] = {600, 700, 800};                    //deklarasi array b

for(int i=0; i<5; i++)  //mengakses array a menggunakan loop for
{
      cout<<"Nilai indeks "<<i<<" array a adalah: "<<a[i]<<endl;
}
cout<<endl;
for(int i=0; i<3; i++)  //mengakses array b menggunakan loop for
{
      cout<<"Nilai indeks "<<i<<" array b adalah: "<<b[i]<<endl;
}
return 0;
}
```

Hasil program:

```
Nilai indeks 1 array a adalah: 100
Nilai indeks 2 array a adalah: 200
Nilai indeks 3 array a adalah: 300
Nilai indeks 4 array a adalah: 400
Nilai indeks 5 array a adalah: 500

Nilai indeks 0 array b adalah: 600
Nilai indeks 1 array b adalah: 700
Nilai indeks 2 array b adalah: 800
```

Contoh Program

```
/* Contoh program array sederhana */
#include <iostream>
using namespace std;

int main()
{
      int a[5], total;
      int jumlah[5];
```

```
	cout<<"Masukkan nilai pada array"<<endl;
	for(int i=0;i<5;i++)
	{
		cout<<"Nilai indeks ["<<i<<"] = ";
		cin>>a[i];
	}
	cout<<endl;
	for(int i=0;i<5;i++)
	{
		cout<<"Nilai indeks ["<<i<<"] adalah "<<a[i]<<endl;
		total+=a[i];
	}
	cout<<"Jumlah nilai elemen pada array a adalah: "<<total;
	return 0;
}
```

Hasil program:

```
Masukkan nilai pada array
Nilai indeks [0] = 1
Nilai indeks [1] = 2
Nilai indeks [2] = 3
Nilai indeks [3] = 4
Nilai indeks [4] = 5

Nilai indeks [0] adalah 1
Nilai indeks [1] adalah 2
Nilai indeks [2] adalah 3
Nilai indeks [3] adalah 4
Nilai indeks [4] adalah 5
Jumlah nilai elemen pada array a adalah: 15
```

10.5 Array Dengan Tipe Data Huruf (char, string, dll)

Untuk array dengan tipe data huruf, terdapat beberapa pendekatan tergantung tipe data yang kita gunakan. Disini kita akan membahas mengenai 2 tipe data huruf, yaitu : char dan

string. Tipe data char dan string sekilas memiliki fungsi yang sama, yaitu menyimpan nilai huruf. Tetapi ternyata terdapat perbedaan dari char dan string. Kode program di bawah menunjukkan deklarasi array dengan tipe data char dan tipe data string.

```
/* contoh deklarasi array dengan tipe data char */

char huruf[5]={'a', 'b', 'c', 'd', 'e'};

/* contoh deklarasi array dengan tipe data string */

string huruf[5]={"saya", "mahasiswa", universitas", "ahmad", "dahlan"};
```

Perhatikan contoh kode program di atas. Dari contoh kode program di atas dapat kita lihat perbedaan dari array dengan tipe data char dan array dengan tipe data string. Array dengan tipe data char hanya dapat menampung satu huruf dalam setiap indeks nya. Sementara array dengan tipe data string dapat menampung banyak huruf dalam setiap indeksnya. Untuk menampilkan nilai array dengan tipe data char atau string, kita menggunakan cara yang sama untuk menampilkan nilai array bertipe int.

Contoh Program

```
#include <iostream>
using namespace std;

int main()
{
      char contohChar[3] = {'a','b','c'};
      string contohString[3]={"Universitas Ahmad Dahlan",
                              "Teknik Informatika"
                              "Saya suka c++"};

      //Menampilkan nilai array bertipe char menggunakan perulangan
      cout<<"Menampilkan nilai char"<<endl;
      for(int i=0;i<3;i++)
      {
            cout<<"Nilai pada indeks ["<<i<<"] adalah:
"<<contohChar[i]<<endl;
      }
            //Menampilkan nilai array bertipe string dengan perulangan
```

```
	cout<<"\nMenampilkan nilai string"<<endl;
	for(int i=0;i<3;i++)
	{
		cout<<"Nilai pada indeks ["<<i<<"] adalah:
"<<contohString[i]<<endl;
	}
	return 0;
}
```

Hasil Program:

```
Menampilkan nilai char
Nilai pada indeks [0] adalah: a
Nilai pada indeks [1] adalah: b
Nilai pada indeks [2] adalah: c

Menampilkan nilai string
Nilai pada indeks [0] adalah: Universitas Ahmad Dahlan
Nilai pada indeks [1] adalah: Teknik Informatika
Nilai pada indeks [2] adalah: Saya suka c++
```

10. 6. Pencarian Elemen Array 1 Dimensi

Salah satu permasalahan yang sering kita temukan adalah pencarian elemen dari sebuah array. Pada umumnya terdapat 2 metode pencarian elemen array, yaitu: metode pencarian linier (linier search) dan metode pencarian bagi dua (binary search).

Metode Pencarian Linier

Metode pencarian linier juga sering disebut sebagai pencarian beruntun (*sequential search*) dimana nilai yang dicari akan dibandingkan dengan nilai dari setiap elemen array, mulai dari elemen pertama sampai dengan nilai yang dicari ditemukan atau sampai dengan elemen terakhir dari array. Kode program di bawah menunjukkan cara menggunakan metode pencarian linier pada sebuah array.

Contoh Program

```
/* Contoh metode pencarian linier */
#include <iostream>
using namespace std;

int main()
{
      int x, arrA[10] = {1, 2, 4, 6, 8, 2, 10, 1, 9, 7};

      cout<<"Isi array: "<<endl;
      for(int i=0; i<10; i++)       //menampilkan isi array
      {
            cout<<"Indeks ["<<i<<"] = "<<arrA[i]<<endl;
      }
      cout<<"\nMasukkan data yang ingin anda cari: ";
      cin>>x;
      cout<<endl;
      for(int i=0; i<10; i++)       //pencarian data pada array
      {
            if (arrA[i] == x)
            cout<<"Data yang anda cari berada pada indeks ke "<<i<<endl;
      }
      return 0;
}
```

Hasil Program:

```
Isi array:
Indeks [0] = 1
Indeks [1] = 2
Indeks [2] = 4
Indeks [3] = 6
Indeks [4] = 8
Indeks [5] = 2
Indeks [6] = 10
Indeks [7] = 1
```

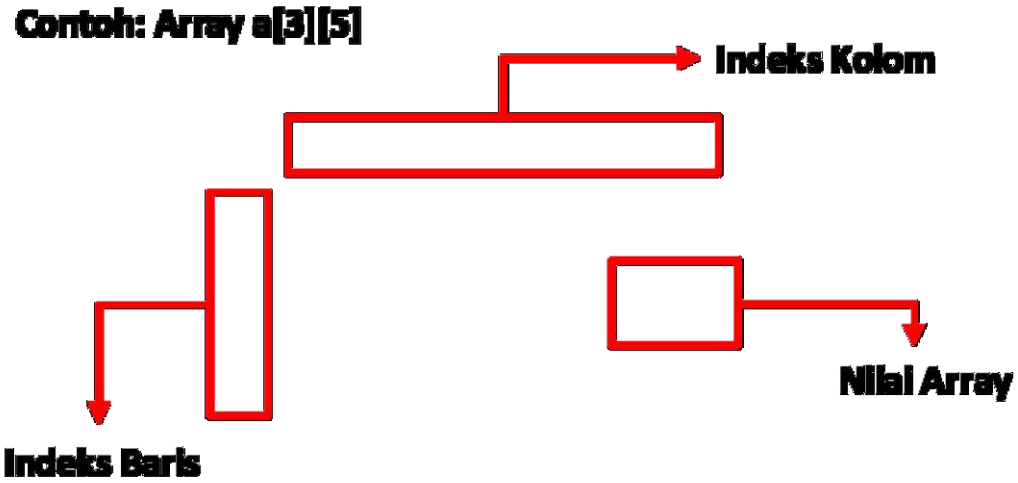
```
Indeks [8] = 9
Indeks [9] = 7

Masukkan data yang ingin anda cari: 1

Data yang anda cari berada pada indeks ke 0
Data yang anda cari berada pada indeks ke 7
```

BAB XI. ARRAY 2 DIMENSI

Pada bab sebelumnya kita telah membahas mengenai array dalam 1 baris atau yang juga dikenal sebagai array 1 dimensi. Selain berbentuk 1 dimensi, array juga dapat berbentuk multidimensi atau memiliki dimensi lebih dari 1. Array dapat memiliki 2 dimensi, 3 dimensi, atau bahkan lebih. Pada bab ini kita akan membahas mengenai array multidimensi, khususnya array 2 dimensi.

11.1. Array 2 Dimensi

Array 2 dimensi adalah array yang memiliki 2 dimensi atau 2 indeks: baris dan kolom. Sama seperti array 1 dimensi, nilai array pada array 2 dimensi harus memiliki tipe data yang sama. Array 2 dimensi dapat dikatakan berbentuk seperti matriks atau tabel. Gambar berikut menjelaskan ilustrasi dari array 2 dimensi.



GGambar 10.1 Ilustrasi Array 2 Dimensi

Dari gambar di atas dapat kita lihat bahwa array a merupakan array 2 dimensi yang merupakan sebuah matriks dengan ordo 3x5. Contoh array di atas memiliki 3 elemen array dimana masing-masing elemennya berupa array dengan 5 elemen, atau bisa kita katakan memiliki 3 baris dan 5 kolom. Oleh karena itu, array multidimensi juga sering disebut dengan "array dari array" atau "kumpulan array".

11.2. Deklarasi Array 2 Dimensi

Array 2 dimensi dapat kita deklarasikan dengan menggunakan bentuk umum sebagai berikut:

```
TipeData NamaArray [JumlahBaris][JumlahKolom];
```

Sebagai contoh, untuk mendeklarasikan array a yang merupakan array 2 dimensi dengan ukuran 3 baris dan 4 kolom yang masing-masing elemennya bertipe int, kita akan menuliskan array a sebagai berikut:

```
/* contoh deklarasi array 2 dimensi */

Int a[3][4];
```

Gambar di bawah merupakan ilustrasi dari kode program di atas. Dapat kita lihat bahwa kode program di atas akan membuat array 2 dimensi yang memiliki 3 baris dan 4 kolom dengan elemen: a[0][0], a[1][0], a[2][0], a[0][1], a[1][1], a[2][1], a[0][2], a[1][2], a[2][2], a[0][3], a[1][3], dan a[2][3].

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Gambar 10.2 Array 2 dimensi dengan 3 baris dan 4 kolom

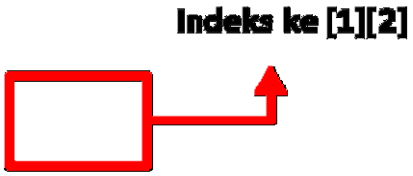
11.3 Pengisian Nilai Pada Array

Untuk mengisi nilai ke dalam sebuah elemen array 2 dimensi, kita dapat menggunakan bentuk umum sebagai berikut:

```
NamaArray [Baris] [Kolom] = Nilai;
```

```
/* mengisi nilai ke dalam array 2 dimensi */

Int a[2][3];            //deklarasi array 2 dimensi berukuran 2 x 3
Int a[0][0]=100;
Int a[0][1]=200;        //mengisi nilai indeks baris o, indeks kolom 1
Int a[0][2]=300;
Int a[1][0]=100;
Int a[1][1]=200;
Int a[1][2]=300;
```

Kode program di atas menunjukkan cara mengisi nilai ke dalam sebuah elemen array. Perhatikan kode “Int a[1][2]=300;”. Dengan menggunakan kode tersebut, elemen array pada baris indeks ke-1 dan kolom indeks ke-2 akan kita isi dengan nilai 300 seperti yang diperlihatkan pada gambar berikut.



Gambar 0.5 Mengisi Elemen Array 2 Dimensi

Kita juga dapat melakukan inisialisasi nilai array 2 dimensi ketika melakukan deklarasi. Apabila kita ingin mendeklarasikan array dan menginisialisasi nilai-nilai dari elemen array secara bersamaan, kita dapat menggunakan bentuk umum sebagai berikut:

`TipeData NamaArray [JumlahElemen] = {{nilai elemen baris 1},{nilai elemen baris 2}, … dst };`

```
/* deklarasi & inisialisasi nilai array 2 dimensi secara bersamaan */

Int siswa[5][3] = {{1,1,1},{2,2,2},{3,3,3},{4,4,4},{5,5,5}};
```

Kode program di atas mendeklarasikan sekaligus menginisialisasi nilai dari array siswa yang bertipe int dan memiliki 5 baris dan 3 kolom. Gambar di bawah memberi ilustrasi hasil dari kode program di atas.

1	**1**	**1**
2	**2**	**2**
3	**3**	**3**
4	**4**	**4**
5	**5**	**5**

Gambar 0.6 Ilustrasi Array 2 Dimensi Berukuran 5x3

Sama seperti pada array 1 dimensi, kita dapat melakukan pengisian nilai pada elemen-elemen array 2 dimensi menggunakan perulangan. Dengan menggunakan perulangan, kode program yang kita gunakan akan menjadi lebih singkat dan sederhana. Kode program berikut menunjukkan cara melakukan pengisian nilai array ke dalam elemen array 2 dimensi menggunakan perulangan bersarang atau nested loop.

```
/* mengisi nilai ke dalam array 2 dimensi menggunakan nested loop */

Int arrayA[2][5];                    //deklarasi array 2 dimensi
cout<<"Masukkan nilai array: "<<endl;
```

```
      for(int i=0;i<2;i++)                //elemen baris array 2 dimensi
      {
            for(int j=0;j<5;j++)          //elemen kolom array 2 dimensi
            {
                  cout<<"Nilai elemen ["<<i<<"]["<<j<<"]= ";
                  cin>>arrayA[i][j];
            }
      }
```

Pada kode program di atas, kita membuat array 2 dimensi dengan jumlah elemen baris 2 (indeks baris ke-0 dan indeks baris ke-1) dan elemen kolom berjumlah 5 (indeks baris ke-0 sampai dengan indeks baris ke-4). Program akan membaca perulangan elemen baris 1 dan elemen kolom 1 sampai dengan 5. Kemudian program akan lanjut membaca perulangan elemen baris 2 dan elemen kolom 1 sampai dengan 5.

11.4. Pembacaan Array 2 Dimensi

Berbeda dengan array 1 dimensi yang hanya menyertakan indeks kolom untuk menampilkan isi array, pada array 2 dimensi kita perlu untuk menyertakan indeks baris. Untuk menampilkan isi dari array 2 dimensi, kita dapat menggunakan bentuk umum sebagai berikut:

```
NamaArray [IndeksBaris][IndeksKolom];
```

```
/* menampilkan nilai array 2 dimensi satu per satu */

cout<<arrayA[0][0]=100;
cout<<arrayA[0][1]=200;//menampilkan nilai indeks baris 0, indeks kolom 1
cout<<arrayA[0][2]=300;
cout<<arrayA[1][0]=100;
cout<<arrayA[1][1]=200;
cout<<arrayA[1][2]=300;
```

Dari contoh kode program di atas dapat kita lihat bahwa kita menampilkan nilai array 2 dimensi dengan memanggil satu per satu menurut indeks elemen arraynya masing-masing. Apabila kita ingin menampilkan nilai array 2 dimensi yang memiliki ukuran besar atau dengan kata lain memiliki banyak baris dan banyak kolom maka kode program yang kita buat akan sangat panjang dan rumit. Untuk mengatasi hal tersebut kita dapat menampilkan nilai array 2

dimensi dengan menggunakan perulangan, lebih tepatnya perulangan bersarang. Kode program di bawah menunjukkan cara menggunakan perulangan untuk menampilkan nilai array 2 dimensi.

```
/* menampilkan nilai array berukuran 2x3 menggunakan nested loop */

for(int i=0;i<2;i++)
      {
            for(int j=0;j<3;j++)
            {
                  cout<<"Nilai array arrayA["<<i<<"]["<<j<<"]=
"<<arrayA[i][j]<<endl;
            }
      }
```

Contoh Program

```
/* Memasukkan dan menampilkan nilai array menggunakan nested loop */
#include <iostream>
using namespace std;

int main()
{
      int siswa[2][5];              //deklarasi array 2 dimensi

      cout<<"Masukkan nilai array: "<<endl;
      for(int i=0;i<2;i++)          //memasukkan nilai ke array 2 dimensi
      {
            for(int j=0;j<5;j++)
            {
                  cout<<"Nilai elemen ["<<i<<"]["<<j<<"]= ";
                  cin>>siswa[i][j];
            }
      }
      cout<<endl;
      for(int i=0;i<2;i++)          //melihat nilai array 2 dimensi
      {
            for(int j=0;j<5;j++)
```

```
            {
                  cout<<"Nilai array siswa["<<i<<"]["<<j<<"]=
"<<siswa[i][j]<<endl;
            }
      }
      return 0;
}
```

Hasil Program:

```
Masukkan nilai array:
Nilai elemen [0][0]= 1
Nilai elemen [0][1]= 2
Nilai elemen [0][2]= 3
Nilai elemen [0][3]= 4
Nilai elemen [0][4]= 5
Nilai elemen [1][0]= 1
Nilai elemen [1][1]= 2
Nilai elemen [1][2]= 3
Nilai elemen [1][3]= 4
Nilai elemen [1][4]= 5

Nilai array siswa[0][0]= 1
Nilai array siswa[0][1]= 2
Nilai array siswa[0][2]= 3
Nilai array siswa[0][3]= 4
Nilai array siswa[0][4]= 5
Nilai array siswa[1][0]= 1
Nilai array siswa[1][1]= 2
Nilai array siswa[1][2]= 3
Nilai array siswa[1][3]= 4
Nilai array siswa[1][4]= 5
```

BAB XII. STRUCT

12.1 Definisi Struct

Struct atau struktur adalah kumpulan dari variable yang dinyatakan dengan sebuah nama. Konsep struktur mirip dengan array. Apabila array hanya dapat menampung variable dengan tipe data sama, struktur dapat menampung beberapa variabel, baik yang memiliki tipe

data sama atau berbeda. Variabel dalam struktur sering disebut dengan komponen, field, elemen, atau anggota. Contoh data yang dapat dinyatakan ke dalam tipe struktur adalah data buku yang memiliki komponen: kode buku, judul, penulis, penerbit, dll. Contoh lainnya adalah data mahasiswa yang terdiri dari komponen: nama mahasiswa, NIM, tempat & tanggal lahir, alamat, dll. Untuk program sederhana struktur mungkin belum terlalu diperlukan karena jumlah variable yang digunakan masih sedikit. Struktur akan sangat berguna untuk program yang kompleks yang menggunakan banyak variable dan banyak tipe data. Dengan menggunakan struktur, kita akan dapat membedakan variable satu dengan yang lain karena kita dapat mengelompokkan nama dan tipe data dari variable.

12.2. Deklarasi Struct

Struktur dideklarasikan dengan menggunakan kata kunci "struct" dan selalu ditutup dengan titik koma (;). Berikut merupakan bentuk umum penulisan struktur:

```
struct NamaStruktur
{
    TipeData NamaAnggota1;
    TipeData NamaAnggota2;
    ...
}NamaObjek;
```

Nama objek merupakan deklarasi yang menggunakan struct tersebut menjadi tipe data dari deklarasi , Kita dapat menggunakan lebih dari 1 objek yang dipisahkan dengan menggunakan tanda koma (,). Penulisan objek dapat adalah opsional, dalam artian dapat dilakukan di luar struktur atau di dalam deklarasi struktur.

```
/* contoh deklarasi struktur */
//contoh 1
struct Mahasiswa     //contoh penulisan struktur pertama
{
    char nama[20];   //deklarasi anggota struktur
    int nim;
    char alamat[20];
};
```

```
Mahasiswa mhs1, mhs2;    //deklarasi objek di luar struktur

//contoh 2
struct Mahasiswa     //contoh penulisan struktur kedua
{
    char nama[20];
    int nim;
    char alamat;
}mhs1, mhs2;                  //deklarasi objek di dalam struktur
```

Kode program di atas adalah contoh bagaimana cara untuk melakukan deklarasi struktur. Struktur Mahasiswa akan memiliki anggota nama, nim, dan alamat. Dari contoh di atas dapat juga kita lihat pendeklarasian objek struktur mhs1 dan mhs2. Untuk dapat menggunakan anggota struktur, objek yang dideklarasikan di luar deklarasi struktur harus dideklarasikan dengan menggunakan nama strukturnya. Dengan deklarasi objek di atas, maka mhs1 dan mhs2 masing-masing memiliki 3 anggota, yaitu: nama, nim, dan alamat.

12.3. Struct Sebagai User Defined Type

UDT atau User Defined Type adalah tipe data yang dibuat sendiri oleh programmer. Dalam bahasa pemrograman C++, untuk membuat tipe data baru sebagai wujud untuk implementasi tipe data UDT dapat menggunakan kata kunci **`typedef`**. Fungsi typedef adalah membuat nama baru bagi sebuah tipe data yang didefinisikan. Fungsi ini dapat dikatakan bertujuan untuk membuat nama alias atau nama lain. Ingat, typedef tidak membuat tipe data yang baru tetapi membuat alias atau nama lain dari sebuah tipe data. Bentuk penulisan penggunaan typedef adalah sebagai berikut:

```
typedef TipeData NamaAlias;
```

Contoh Program

```
/* contoh penggunaan typedef */
#include <iostream>
using namespace std;

int main()
{
```

```
      typedef int contoh;           //membuat alias dari tipe data int
      contoh nilai1, nilai2;        //deklarasi dgn tipe data alias
      int hasil;                    //deklarasi dgn tipe data asli
      hasil = nilai1 + nilai2;
      cout<<"Masukkan nilai pertama: ";
      cin>>nilai1;
      cout<<"\nMasukkan nilai kedua: ";
      cin>>nilai2;
      cout<<"\nHasil penjumlahan adalah "<<hasil<<endl;
      return 0;
}
```

Hasil Program:

```
Masukkan nilai pertama: 20
Masukkan nilai kedua: 10
Hasil penjumlahan adalah 30
```

Perhatikan contoh kode program di atas. Pada kode program di atas, tipe data “int” kita beri nama alias “contoh”. Kemudian kita menggunakan nama alias tersebut untuk mendeklarasikan variable nilai1 dan nilai2. Meskipun tipe data int telah kita beri nama alias, kita masih tetap dapat menggunakan tipe data aslinya. Pada contoh di atas kita menggunakan tipe data int untuk deklarasi variable hasil.

12.4. Pemanggilan Elemen Struct

Elemen struktur atau anggota struktur adalah variable yang didirikan di dalam struktur dan dijadikan sebagai bagian dari struktur. Kita tidak dapat menggunakan anggota-anggota dari struktur sebelum membuat objek yang dibuat dengan menggunakan data struktur tersebut. Setelah kita berhasil membuat deklarasi objek, kita dapat mengakses anggota struktur dari objek tersebut dengan menggunakan tanda titik (**.**) di antara nama objek dan nama anggota variable struktur. Pemanggilan anggota struktur dapat menggunakan bentuk umum sebagai berikut:

```
NamaObjek.NamaAnggota;
```

Contoh Program

```
/* contoh sederhana penggunaan struktur */
#include <iostream>
using namespace std;

int main()
{
      struct mahasiswa              //deklarasi struktur
      {
            char nama[20];
            char alamat[30];
            int nim;
      }
      mahasiswa mhs;                        //deklarasi objek di dalam
struktur
      cout<<"Masukkan Nama\t\t: ";
      gets(mhs.nama);
      cout<<"Masukkan Alamat\t      : ";
      gets(mhs.alamat);
      cout<<"Masukkan NIM\t\t: ";
      cin>>mhs.nim;
      cout<<"\nData Mahasiswa"<<endl<<endl;
      cout<<"Nama\t\t: "<<mhs.nama<<endl; //memanggil anggota struktur
      cout<<"Alamat\t\t: "<<mhs.alamat<<endl;
      cout<<"NIM\t\t: "<<mhs.nim<<endl;
      return 0;
}
```

Hasil Program:

```
Masukkan Nama         : Ali Budimansyah
Masukkan Alamat       : Jl, Gejayan No.11
Masukkan NIM          : 123123

Data Mahasiswa

Nama          : Ali Budimansyah
```

Alamat : Jl, Gejayan No.11 NIM : 123123

12.5. Array of Struct

Array of struct atau array dari struktur adalah ketika kita mendeklarasikan sebuah array yang anggota-anggotanya berupa struktur. Untuk mengakses anggota pada array dari struktur, kita dapat menggunakan bentuk umum penulisan kode sebagai berikut:

```
NamaObjek [Indeks].NamaAnggota;
```

Contoh Program

```
/* contoh penggunaan array of structure */
#include <iostream>
using namespace std;

int main()
{
	struct mahasiswa		//deklarasi struktur
	{
		char nama[30];
		char alamat[30];
		int nim;
	};
	mahasiswa mhs[3];	//jumlah perulangan
	for(int i=0; i<3; i++)
	{
		cout<<"Masukkan Nama\t\t: ";
		cin>>mhs[i].nama;
		cout<<"Masukkan NIM\t\t: ";
		cin>>mhs[i].nim;
		cout<<"Masukkan Alamat\t\t: ";
		cin>>mhs[i].alamat;
		cout<<endl;
	}
	cout<<"\nData yang sudah dimasukkan"<<endl;
```

```
	for(int i=0; i<3; i++)
	{                          //mengakses anggota
		cout<<"Nama\t\t: "<<mhs[i].nama<<endl;
		cout<<"NIM\t\t: "<<mhs[i].nim<<endl;
		cout<<"Alamat\t\t: "<<mhs[i].alamat<<endl;
		cout<<endl;
	}
	return 0;
}
```

Hasil Program:

```
Masukkan Nama        : Ali
Masukkan NIM         : 123
Masukkan Alamat      : Yogya

Masukkan Nama        : Sita
Masukkan NIM         : 456
Masukkan Alamat      : Sleman

Masukkan Nama        : Roni
Masukkan NIM         : 321
Masukkan Alamat      : Bantul

Data yang sudah dimasukkan
Nama        : Ali
NIM         : 123
Alamat      : Yogya

Nama        : Sita
NIM         : 456
Alamat      : Sleman

Nama        : Roni
NIM         : 321
```

Alamat : Bantul

12.4 Nested Struct

Nested struct atau struktur bersarang adalah struktur di dalam struktur. Untuk lebih jelasnya dapat kita lihat kode program untuk mendeklarasikan struktur bersarang berikut ini:

```
/* contoh deklarasi nested struct */
struct anak       //deklarasi struct anak
{
      string nama;
      int umur;
};
struct siswa       //deklarasi struct siswa
{
      anak info_anak;    //nested struct
      int noInduk;
      float nilaiUjian;
};
```

Kode program di atas menunjukkan 2 buah struktur dimana struktur anak bersarang di dalam struktur siswa. Untuk dapat mengakses data anggota struktur anak kita harus mengakses struktur siswa terlebih dahulu. Perhatikan contoh program sebagai berikut:

Contoh Program

```
/* contoh penggunaan nested struct */
#include <iostream>
using namespace std;

int main()
{
      struct Nilai                  //deklarasi struktur Nilai
      {
            int kuis;
            int uts;
            int uas;
      };
```

```
      struct Mahasiswa        //deklarasi struktur Mahasiswa
      {
            char nama[20];
            int nim;
            struct Nilai nilai_akhir;     //nested struct
      };
      Mahasiswa data_mahasiswa;     //deklarasi objek di luar struktur
      cout<<"Pendataan Nilai Mahasiswa"<<endl;
      cout<<"Masukkan nama Mahasiswa\t\t: ";
      gets(data_mahasiswa.nama);
      cout<<"Masukkan NIM Mahasiswa\t\t: ";
      cin>>(data_mahasiswa.nim);
      cout<<"Masukkan Nilai Kuis\t\t: ";
      cin>>data_mahasiswa.nilai_akhir.kuis;
      cout<<"Masukkan Nilai UTS\t\t: ";
      cin>>data_mahasiswa.nilai_akhir.uts;
      cout<<"Masukkan Nilai UAS\t\t: ";
      cin>>data_mahasiswa.nilai_akhir.uas;
      cout<<endl;
      cout<<"Data Mahasiswa"<<endl;
      cout<<"Nama Mahasiswa\t: "<<data_mahasiswa.nama<<endl;
      cout<<"NIM Mahasiswa\t: "<<data_mahasiswa.nim<<endl;
      cout<<"Nilai Kuis\t: "<<data_mahasiswa.nilai_akhir.kuis<<endl;
      cout<<"Nilai UTS\t: "<<data_mahasiswa.nilai_akhir.uts<<endl;
      cout<<"Nilai UAS\t: "<<data_mahasiswa.nilai_akhir.uas<<endl;
      cout<<"Nilai Akhir\t: "<<nilai(data_mahasiswa.nilai_akhir.kuis,
                              data_mahasiswa.nilai_akhir.uts,
                              data_mahasiswa.nilai_akhir.uas)<<endl;
      return 0;
}
```

Hasil Program:

```
Pendataan Nilai Mahasiswa
Masukkan nama Mahasiswa        : Ali
Masukkan NIM Mahasiswa         : 123456
Masukkan Nilai Kuis            : 90
```

```
Masukkan Nilai UTS          : 68
Masukkan Nilai UAS          : 89

Data Mahasiswa
Nama Mahasiswa  : Ali
NIM Mahasiswa   : 123456
Nilai Kuis      : 90
Nilai UTS       : 68
Nilai UAS       : 89
Nilai Akhir     : 82
```

Kode program di atas menjelaskan mengenai struktur bersarang. Struct Nilai adalah anak dari struct Mahasiswa. Jadi semisal kita ingin mengakses data dari struct Nilai maka kita harus meminta ijin akses ke struct Mahasiswa. Untuk mengakses data kuis, uts, dan uas yang merupakan anggota dari struct Nilai, kita harus mengakses struct Mahasiswa terlebih dahulu. Contoh: **`data_mahasiswa.nilai_akhir.uts;`**.

BAB XIII POINTER

Pointer adalah fitur C++ yang dibawa dari bahasa C dan tidak banyak dimiliki oleh bahasa pemrograman yang lain. Pointer dibuat untuk dapat melakukan alokasi memori secara dinamis. Ketika sebuah program menggunakan nilai tertentu, nilai tersebut akan disimpan pada memori. Dan ketika nilai tersebut tidak diperlukan, kita dapat membuangnya. Penggunaan pointer secara tepat akan dapat meningkatkan efisiensi program. Di sisi yang lain, penggunaan pointer yang tidak tepat akan dapat menyebabkan terjadinya kebocoran memori atau memory leak. Bab ini akan mencoba untuk membahas mengenai pointer, bagaimana cara menggunakan pointer, dan alokasi memori pointer.

13.1. Pengenalan Pointer

Pointer adalah variable yang menunjuk ke alamat memori dari variable lain. Karena pointer menunjuk alamat dari sebuah variable, maka pointer akan berisi alamat memori dan bukan berisi nilai. Setiap variable atau fungsi pada kode program menunjuk alamat tertentu pada memori. Pointer bertugas untuk menunjuk alamat-alamat pada memori tersebut.

Sebagai contoh: kita mendeklarasikan variable a dengan tipe data int. Untuk mendapatkan alamat dari suatu variabel a, kita dapat menggunakan address of operator yang

menggunakan tanda "**&**" di depan variable a tersebut. Kode program di bawah menunjukkan cara mendapatkan alamat dari sebuah variable.

Contoh Program

```
/* Menampilkan alamat dari variable */
#include<iostream>
using namespace std;

int main()
{
      int a = 10;
      cout<<"Nilai a adalah: "<<a<<endl;
      cout<<"Alamat memori a adalah: "<<&a<<endl;
      return 0;
}
```

Hasil Program:

```
Nilai a adalah: 10
Alamat a adalah: 0x6ffe3c
```

13.2 Deklarasi Pointer

Pointer dideklarasikan dengan menggunakan dereference operator atau dereferencing yang menggunakan tanda asterisk (*****). Untuk mendeklarasikan pointer, kita dapat menggunakan bentuk umum sebagai berikut:

```
TipeData *NamaPointer;
```

Kita tetap menggunakan tipe data untuk deklarasi pointer. Pointer yang dideklarasikan menggunakan tipe data int dapat menunjuk ke alamat-alamat dari memori yang didalamnya berisi nilai dengan tipe int. Contoh: pointer p (*p) yang bertipe data int akan dapat menunjuk ke alamat memori yang memiliki nilai dengan tipe data int, tetapi tidak dapat digunakan untuk menunjuk ke alamat memori yang berisi nilai dengan tipe data double atau float. Berikut merupakan contoh pointer yang menunjuk ke alamat dari sebuah variable.

```
/* Contoh pointer menunjuk ke alamat variabel */
int a = 10;
```

```
int *p = &a;        //p berisi alamat memori a
int b = *p;        //b memiliki nilai a, yaitu 10
```

Pada kode program di atas dilakukan deklarasi variable a yang bertipe int dengan nilai 10. Kemudian dilakukan deklarasi pointer p yang bertipe int. Kemudian pointer p menunjuk ke alamat variable a. Pada saat pointer p menunjuk ke alamat variable a, maka pointer p akan memiliki nilai yang sama dengan variable a dan p akan sama dengan &a. p dan &a sama-sama berisi alamat memori. Kemudian variabel b menunjuk ke pointer p yang memiliki nilai dari variable a (10). Dengan menggunakan cara di atas, kita dapat mengakses dan memanipulasi nilai variable b melalui pointer p. Proses mengakses nilai a melalui pointer p ini yang kita sebut dengan dereference pointer.

Pointer dapat kita deklarasikan tanpa menggunakan tipe data atau pointer tanpa tipe dan digantikan dengan void. Bentuk umum penulisannya sebagai berikut:

```
void *NamaPointer;
```

```
/* Contoh void pointer */
void *p =NULL;
```

Dengan menggunakan kode di atas, pointer p dapat digunakan untuk menunjuk ke alamat memori yang memiliki tipe data apa saja, seperti: int, float, double, char, dll.

Contoh Program

```
/* contoh penggunaan void pointer */
#include <iostream>
using namespace std;

int main()
{
      int i = 10;
      float f = 0.25;
      double d = 123.123;

      void *p =NULL;                        //deklarasi void pointer

      p = &i;                               //menunjuk ke alamat int i
      cout<<"p menunjuk ke alamat i"<<endl;
      cout<<"Nilai *p: "<<*((int*)p)<<endl;
```

```
      cout<<"Nilai p: "<<p<<endl<<endl;
      p = &f;                              //menunjuk ke alamat float f
      cout<<"p menunjuk ke alamat f"<<endl;
      cout<<"Nilai *p: "<<*((float*)p)<<endl;
      cout<<"Nilai p: "<<p<<endl<<endl;
      p = &d;                              //menunjuk ke alamat double d
      cout<<"p menunjuk ke alamat d"<<endl;
      cout<<"Nilai *p: "<<*((double*)p)<<endl;
      cout<<"Nilai p: "<<p<<endl<<endl;
      return 0;
}
```

Hasil Program:

```
p menunjuk ke alamat i
Nilai *p: 10
Nilai p: 0x6ffe34

p menunjuk ke alamat f
Nilai *p: 0.25
Nilai p: 0x6ffe30

p menunjuk ke alamat d
Nilai *p: 123.123
Nilai p: 0x6ffe28
```

13.3 Pointer Untuk Alokasi Memori

C++ memungkinkan kita untuk mengatur ruang memori secara dinamis sesuai dengan kebutuhan program. Kita dapat mengalokasikan ruang memori untuk menyimpan nilai tertentu. Setelah nilai tersebut selesai digunakan, kita dapat membuang atau dealokasi memori tersebut. Dengan demikian ruang memori tersebut dapat digunakan untuk keperluan yang lain.

Terdapat 2 cara untuk melakukan alokasi dan dealokasi memori, yaitu:

- Menggunakan fungsi malloc() dan free()

- Menggunakan operator new dan delete

13.4. malloc() dan free()

Fungsi malloc() dan free() merupakan fungsi yang diturunkan dari bahasa C. Kedua fungsi ini dapat diakses dengan menggunakan file header <cstdlib>. Fungsi malloc() berguna untuk melakukan alokasi ruang memori dan mengembalikan pointer ke tipe void. Sedangkan free() berguna untuk melakukan dealokasi ruang memory. Berikut adalah contoh penulisan fungsi malloc() dan free():

```
/* Contoh penggunaan malloc() dan free() */
void*malloc(size_t, size);
void free (void*ptr);
```

Contoh Program

```
/* Contoh penggunaan malloc() dan free() */
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
      int *p;                 //deklarasi pointer tipe int
      p = (int *) malloc(5 * sizeof(int));     //mengalokasikan 5 ruang
                                                    memori tipe int
      cout<<"Isi Ruang Memori Yang Dialokasikan"<<endl;
      cout<<endl;
      for (int i=0; i<5; i++)
      {
            p[i] = (i+1)*10;        //mengisi ruang memori yang telah
                                    dialokasikan
            cout<<p[i]<<endl;       //menampilkan isi memori
      }
      free (p);                           //dealokasi ruang memori

      cout<<endl;
      for (int i=0; i<5; i++)
```

```
	{
		cout<<p[i]<<endl;	//menampilkan isi memori setelah
					free()
	}
	return 0;
}
```

Hasil Program:

```
Isi Ruang Memori Yang Dialokasikan

10
20
30
40
50

11016720
0
11010384
0
50
```

13.5 new dan delete

Operator new dan delete merupakan fitur yang hanya ada pada bahasa C++. Operator new digunakan untuk alokasi memori, sedangkan operator delete digunakan untuk dealokasi memori. Untuk mengalokasikan satu ruang memori, kita dapat menggunakan bentuk umum sebagai berikut:

```
NamaPointer = new TipeData;
```

Jika yang dialokasikan lebih dari 1, maka bentuk umumnya menjadi seperti berikut:

```
NamaPointer = new TipeData [JumlahElemen];
```

```
/* Contoh penggunaan operasi new */
int *p;
p = new int[5];		//mengalokasikan 5 ruang memori
```

Untuk mendealokasikan 1 ruang memori, kita dapat menggunakan bentuk umum sebagai berikut:

```
delete NamaPointer;
```

Sedangkan untuk mendealokasikan lebih dari 1 ruang memori, kita dapat menggunakan bentuk umum berikut ini.

```
delete [] NamaPointer;
```

```
/* Contoh penggunaan operasi delete */
int *p;
delete [] p;              //dealokasi lebih dari 1 ruang memori
```

Contoh Program

```
/* contoh penggunaan new dan delete */
using namespace std;
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
      int *p=NULL;
      p = new int[5];
      for (int i=0; i<5; i++)
      {
            p[i] = (i+1) * 10;
            cout<<p[i]<<" ";
      }

      delete [] p;
      return 0;
}
```

Hasil Program:

```
10 20 30 40 50
```

13.6 Pointer dan Array

Pointer dapat digunakan untuk menunjuk ke sebuah array dan sebaliknya dengan catatan memiliki tipe data yang sama. Ketika pointer digunakan untuk menunjuk array, pointer secara otomatis akan menunjuk ke alamat memori dari elemen pertama array. Perhatikan kode program di bawah ini:

Contoh Program

```
/* contoh penggunaan pointer ke array */
#include <iostream>
using namespace std;

int main()
{
      int a[5]={10, 20, 30, 40, 50};      //array bertipe int dengan 5
                                          elemen
      int *p;                             //pointer tipe int

      p = a;                              //p menunjuk ke &a[0]
      for(int i=0; i<5; i++)  //akses elemen array menggunakan pointer
      {
            cout<<p[i]<<" ";
      }
      return 0;
}
```

Hasil Program:

```
10 20 30 40 50
```

Kode di atas mendeklarasikan array a dan pointer p yang bertipe int. Karena memiliki jenis tipe data yang sama, kita dapat menggunakan pointer p untuk menunjuk ke array a. Kemudian p akan menunjuk ke alamat dari a[0].

Pointer dapat dijadikan sebagai elemen dari sebuah array. Untuk membuat array yang memiliki elemen bertipe pointer, kita dapat menggunakan bentuk umum penulisan seperti berikut:

```
TipeData *NamaArray [JumlahElemen];
```

Contoh Program

```
/* contoh penggunaan array dari pointer */
#include <iostream>
using namespace std;

int main()
{
      int *ptr[5];

      cout<<"Sebelum dirubah"<<endl;      //menampilkan nilai sebelum
                                          dirubah
      for (int i=0; i<5; i++)
      {
            ptr[i] = new int;             //alokasi memori tipe int
            *ptr[i] = (i+1) *10;          //mengisi nilai ke alamat
            cout<<"Nilai: "<<*ptr[i]<<" tersimpan pada alamat:
"<<ptr[i]<<endl;
      }
      *ptr[2] = 99;                       //mengubah nilai indeks 2
      cout<<"\nSetelah dirubah"<<endl;    //menampilkan nilai setelah
                                          dirubah
      for (int i=0; i<5; i++)
      {
            cout<<"Nilai: "<<*ptr[i]<<" tersimpan pada alamat:
"<<ptr[i]<<endl;
      }

      for (int i=0; i<5; i++)
      {
            delete ptr [i];         //menghapus alamat yg telah
                                    dialokasikan
      }
      return 0;
}
```

Hasil Program:

```
Sebelum dirubah
Nilai: 10 tersimpan pada alamat: 0x1a1510
Nilai: 20 tersimpan pada alamat: 0x1a1530
Nilai: 30 tersimpan pada alamat: 0x1a1550
Nilai: 40 tersimpan pada alamat: 0x1a1b80
Nilai: 50 tersimpan pada alamat: 0x1a1af0

Setelah dirubah
Nilai: 10 tersimpan pada alamat: 0x1a1510
Nilai: 20 tersimpan pada alamat: 0x1a1530
Nilai: 99 tersimpan pada alamat: 0x1a1550
Nilai: 40 tersimpan pada alamat: 0x1a1b80
Nilai: 50 tersimpan pada alamat: 0x1a1af0
```

13.7. Pointer ke Fungsi

Dalam pemrograman terkadang kita menemukan sebuah fungsi yang memiliki nama yang panjang. Untuk menyederhanakan pada saat pemanggilan fungsi tersebut, kita dapat menggunakan pointer yang menunjuk ke alamat dari memori fungsi tersebut. Berikut merupakan contoh kode program untuk pemanggilan fungsi dengan menggunakan pointer.

Contoh Program

```
/* contoh penggunaan pointer ke fungsi */
#include <iostream>
using namespace std;

double perkalian_2_bilangan (double a, double b)
{
      return a*b;
}
```

```
int main()
{
      double x, y;      //deklarasi 2 variabel bertipe double
      double (*ptr) (double, double);     //deklarasi pointer ke fungsi
                                          yang memiliki 2 parameter bertipe
                                          double
      ptr = &perkalian_2_bilangan;  //ptr menunjuk ke alamat fungsi
      cout<<"Masukkan bilangan pertama: ";
      cin>>x;
      cout<<"\nMasukkan bilangan kedua: ";
      cin>>y;
      double hasil = ptr(x, y);           //memanggil fungsi
      cout<<endl;
      cout<<x<<" * "<<y<<" = "<<hasil;
      return 0;
}
```

Hasil Program:

```
Masukkan bilangan pertama: 2

Masukkan bilangan kedua: 3

2 * 3 = 6
```

BAB XIV INPUT/OUTPUT FILE

14.1 Pendahuluan

Dalam pemrograman sebuah program dapat membaca input yang diketikkan oleh pengguna (*user*) program melalui *keyboard*. Bahasa C memfasilitasi mekanisme untuk membaca input dari *keyboard* tersebut. Tidak jarang sebuah program juga membaca input dari suatu file yang tersimpan di harddisk. Begitu pula sebuah program dapat menuliskan outputnya di suatu file sehingga output tersebut dapat disimpan dan diakses kembali sewaktu-waktu. Pada bab ini akan dibahas mekanisme untuk membaca file serta menulis file oleh program yang ditulis dalam bahasa C. Fungsi-fungsi untuk pembacaan dan penulisan (input/output) pada bahasa C terdapat pada library `stdio.h`

14.2 Membaca Text File

Untuk membaca sebuah file teks dari program C, mula-mula perlu dideklarasikan sebuah pointer yang menunjuk pada suatu File. Deklarasi pointer yang menunjuk pada suatu File dilakukan sebagai berikut.

File *fp;

Selanjutnya, sebelum dilakukan pembacaan isi file, sebuah file harus dibuka terlebih dahulu. Untuk membuka suatu file, digunakan fungsi `fopen`. Fungsi `fopen` memiliki dua parameter yaitu nama file serta mode yang dipilih. Contoh pemanggilan fungsi `fopen` adalah sebagai berikut.

```
fp = fopen("myfile.txt", "r");
```

Parameter `"myfile.txt"` adalah nama file yang akan dibaca. File ini harus berada pada lokasi yang sama dengan program C atau jika tidak maka *fullpath* nya harus disertakan seperti pada contoh di bawah ini.

```
fp = fopen("D:\MyProgram\myfile.txt", "r");
```

Parameter kedua pada fungsi fopen adalah mode dalam membaca file. Ada beberapa mode pembacaan file yang disupport oleh bahasa C. Mode-mode tersebut ditampilkan dalam Tabel 14.1.

Tabel 14.1 Mode pembukaan (open) file dan deskripsinya

Mode	Description
r	Membuka file teks dalam mode “reading” (membaca)
w	Membuka file teks dalam mode “writing” (menulis) Jika file belum ada, program akan membuat file tsb dan menulis mulai dari awal file
a	Membuka file teks dalam mode “append” (menggabung) Program akan menulis pada file teks dengan menggabungkan dengan isi file yang sudah ada. Jika file belum ada, program akan membuat file tsb
r+	Membuka file teks dalam mode “reading and writing”
w+	Membuka file teks dalam mode “reading and writing”. Apabila file teks sudah terisi sebelumya maka program akan menghapus isi file, kemudian menulis pada file tersebut dari awal file
a+	Membuka file teks dalam mode “reading and writing”, tetapi penulisan pada file dilakukan dengan append (menggabung dengan isi file yang sudah ada). Pembacaan file dapat dimulai dari awal file.

Fungsi fopen akan menghasilkan null jika file tidak berhasil dibuka (ada error saat membuka file). Setelah file dibuka dengan fungsi fopen, langkah selanjutnya adalah pembacaan isi file teks. Pembacaan dapat dilakukan dengan cara membaca satu demi satu karakter isi file teks atau langsung membaca keseluruhan isi file teks.

Listing 14.1 di bawah ini menunjukkan cara melakukan pembacaan pada isi file teks.

```
FILE * pFile;
int c;
```

```
 int n = 0;
 pFile=fopen ("myfile.txt","r");
 if (pFile==NULL)
 {
        printf("Error when opening file!!");
 }
 else
 {
  do {
        c = fgetc (pFile);
        printf("%c", c);
  } while (c != EOF);
  fclose (pFile);
 }
 return 0;
}
```

Listing 14.1 Pembacaan isi file teks per karakter

Pada listing di atas terdapat sebuah kondisional `if-else` untuk mengecek apakah file teks berhasil dibuka atau tidak. Jika file tidak berhasil dibuka pesan `Error when opening file!!` akan ditampilkan. Jika file berhasil dibuka, perulangan dilakukan untuk membaca setiap karakter di dalam file teks mulai dari karakter pertama hingga karakter terakhir. Pembacaan karakter ini dilakukan dengan memanggil fungsi `fgetc`. Akhir dari isi file teks ditandai dengan karakter `EOF` (End of File). Pada listing kode 14.1 di atas, karakter yang dibaca dari file teks disimpan dalam variabel `c`, kemudian setiap karakter `c` ditampilkan dengan fungsi `printf`. Setelah pembacaan file selesai dilakukan, file teks ditutup dengan fungsi `fclose`.

Selain menggunakan fungsi `fgetc`, pembacaan isi file teks dapat dilakukan juga menggunakan fungsi `fgets`. Fungsi `fgets` tidak hanya membaca sebuah karakter, melainkan membaca sejumlah karakter dan menyimpannya dalam string. Perhatikan Listing 14.2 berikut ini.

```
/* fgets example */
#include <stdio.h>

int main()
{
   FILE * pFile;
   char mystring [100];

   pFile = fopen ("myfile.txt" , "r");
   if (pFile == NULL)
   {printf("Error opening file");}
   else {
     if ( fgets (mystring , 100 , pFile) != NULL )
       puts (mystring);
      fclose (pFile);
   }
   return 0;
}
```

Listing 14.2 Pembacaan isi file teks dengan fgets

Pada listing 14.2 fungsi `fgets` dipakai untuk membaca isi file teks. Fungsi `fgets` memiliki tiga parameter yaitu pointer variabel string untuk menyimpan string yang dibaca, ukuran string yang dibaca (termasuk sebuah karakter terminasi, -null sehingga pada kode ukuran string 100 namun karakter yang dibaca hanya 99), serta pointer file teks yang dibaca. Fungsi `fgets` mengembalikan null jika terdapat error saat pembacaan string. Fungsi `puts` mengoutputkan string pada console.

14.3 Menulis pada File

Selain melakukan pembacaan file, program C juga dapat melakukan penulisan pada file teks. Seperti pada pembacaan file teks, penulisan pada file teks didahului dengan pemanggilan fungsi `fopen` namun mode yang digunakan adalah mode `w` (*writing*). Listing 14.3 di bawah ini menampilkan kode program untuk menulis karakter pada file teks alphabet.txt.

```
/* fputc example: alphabet writer */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  char c;

  pFile = fopen ("alphabet.txt","w");
  if (pFile!=NULL) {

    for (c = 'A' ; c <= 'Z' ; c++)
      fputc ( c , pFile );

    fclose (pFile);
  }
  return 0;
}
```

Listing 14.3 Penulisan karakter pada file alphabet.txt

Untuk penulisan karakter pada file teks digunakan fungsi `fputc` yang menerima dua buah input yaitu karakter yang akan dituliskan dan pointer ke file teks yang akan ditulisi. Contoh kode pada Listing 14.3 di atas menuliskan karakter 'A' sampai karakter 'Z' pada file teks alphabet.txt. Selain untuk menulis sebuah karakter pada file teks, dalam bahasa C kita dapat memanggil fungsi `fputs` untuk menuliskan string ke dalam file teks. Listing 14.4 menunjukkan kode untuk menulis string ke dalam file teks mylog.txt.

```
/* fputs example */
#include <stdio.h>

int main ()
{
   FILE * pFile;
   char sentence [256];
```

```
   printf ("Enter sentence to append: ");
   fgets (sentence,256,stdin);
   pFile = fopen ("mylog.txt","a");
   fputs (sentence,pFile);
   fclose (pFile);
   return 0;
}
```

Listing 14.4 Menuliskan string ke file teks

DAFTAR PUSTAKA

1. www.cplusplus.com
2. www.tutorialspoint.com
3. www.cppreference.com


LABORATORIUM
TEKNIK INFORMATIKA
UNIVERSITAS AHMAD DAHLAN


UNIVERSITAS AHMAD DAHLAN