I. PENDAHULUAN

1.1 C & C++

Berbicara tentang C++ biasanya tidak lepas dari C, sebagai bahasa pendahulunya. Pencipta C adalah Brian W. Kerninghan dan Dennis M. Ritchie pada sekitar tahun 1972, dan sekitar satu dekade setelahnya diciptakanlah C++, oleh Bjarne Stroustrup dari Laboratorium Bell, AT&T, pada tahun 1983. C++ cukup kompatibel dengan bahasa pendahulunya C. Pada mulanya C++ disebut " a better C ". Nama C++ sendiri diberikan oleh Rick Mascitti pada tahun 1983, yang berasal dari operator increment pada bahasa C.

Keistimewaan yang sangat berari dari C++ ini adalah karena bahasa ini mendukung pemrograman yang berorientasi objek (OOP / Object Oriented Programming).

1.2 PEMROGRAMAN BERORIENTASI OBJEK

Sampai saat ini, program dianggap sebagai sekumpulan procedure yang melakukan aksi terhadap data. Procedure, atau function, adalah suatu set instruksi khusus yang dieksekusi secara bergantian. Data terpisah dari procedure, dan trik pemrogramannya adalah menjaga urutan pemanggilan fungsi, dan data apa yang diubah. Dalam demikian terciptalah program yang terstruktur.

Ide dari pemrograman terstruktur adalah memecah program yang besar menjadi kecil sehingga lebih mudah dipahami. Program-program lama memaksa pengguna untuk melakukan langkah-perlangkah melalui layar monitor. Sedangkan program modern menyajikan semua pilihan sekaligus dan merespon aksi pengguna.

OOP berusaha untuk memenuhi kebutuhan itu, menyediakan teknik untuk mengelola kompleksitas, mencatat penggunaan ulang komponen software dsb. Inti dari OOP adalah memperlakukan data dan procedure sebagai sebuah objek berisi entitas dengan identitas dan ciri yang khusus.

1.3 C++ & OBJECT ORIENTED PROGRAMMING

Tiga karakteristik utama dari bahasa yang berorientasi objek adalah

1. Encapsulation
2. Inheritance
3. Polymorphisme.

Tiga ciri diatas mendukung reusability, yang merupakan salah satu factor penentu kualitas software.

C++ mendukung karakteristik encapsulation dengan menggunakan konsep class. Setelah terbentuk, maka class akan bertindak sebagai entitas yang tenkapsulasi.

Dengan adanya konsep inheritance, maka C++ mendukung ide penggunaan ulang suatu object.

Polymorphisme (Banyak Bentuk) merupakan suatu konsep yang menyatakan sesuatu yang sama dapat memiliki berbagai bentuk dan perilaku yang berbeda.

1.4 MENYIAPKAN PROGRAM

Source code C++ dapat ditulis pada text editor apapun. Baik itu digunakan pada sistem operasi Windows atau Unix (Linux, BSD, dsb).

Walaupun demikian, lebih disarankan apabila digunakan dengan editor khusus C++, agar penggunaan tools yang lain lebih mudah.

1.5 KOMPILASI

Untuk mengubah source code menjadi sebuah program, kita gunakan compiler. Setelah source code tercompile, terbentuklah sebuah file objek dengan ekstension " .obj ". File " .obj " ini belum merupakan sebuah program executable. Untuk membentuk program executable linker harus dijalankan. Jika program executable sudah diperoleh, walaupun di komputer anda tidak terinstall compiler C++ namun program masih tetap dapat dijalankan.

Saat ini banyak compiler C++ yang berada di pasaran, contohnyaa Borland C++, Turbo C++, Microsoft C++, C++ Builder, Visual C++ sampai pada compiler gratis seperti g++ di Unix.

II. **ELEMEN DASAR**

Untuk membuat suatu program ada baiknya kita mengenal terlebih dahulu apa yang disebut dengan *prepocessor directive.* Propocessor ditandai dengan adanya awalan # . Preprocessor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi.

Setiap program C++ mempunyai bentuk seperti di bawah , yaitu:

```
#prepocessor directive
main()
{
  // Batang Tubuh Program Utama
}
```

Melihat bentuk seperti itu dapat kita ambil kesimpulan bahwa batang tubuh program utama berada didalam fungsi main(). Berarti dalam setiap pembuatan program utama, maka dapat dipastikan seorang pemrogram menggunakan minimal sebuah fungsi. Pembahasan lebih lanjut mengenai fungsi akan diterangkan kemudian. Yang sekarang coba ditekankan adalah kita menuliskan program utama kita didalam sebuah fungsi main(). Jangan lupa bahwa C++ bersifat case sensitive, sehingga, nama hallo dan Hallo berbeda artinya.

2.1 CARA PENULISAN

- Komentar

Komentar tidak pernah dicompile oleh compiler. Dalam C++ terdapat 2 jenis komentar, yaitu:

Jenis 1 : /* Komentar anda diletakkan di dalam ini
Bisa mengapit lebih dari satu baris */

Jenis 2 : // Komentar anda diletakkan disini (hanya bisa perbaris)

- Semicolon

Tanda semicolon “;” digunakan untuk mengakhiri sebuah pernyataan. Setiap pernyataan harus diakhiri dengan sebuah tanda semicolon. Baris yang diawali dengan tanda #, seperti

#include <iostream.h>

tidak diakhiri dengan tanda semicolon, karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan prepocessor directive

2.2 MASUKAN DAN KELUARAN DASAR

Pada C++ terdapat 2 jenis I/O dasar, yaitu:

a. cout (character out), standard keluaran
b. cin (character in), standard masukan

Untuk dapat menggunakan keyword diatas, maka harus ditambahkan #include <iostream.h> pada prapocessor directive.

Contoh :

```
#include <iostream.h>
main()
{
    char nama[100];    // Dekalarasi variable nama
    cout<<"Masukkan nama Anda : ";
    cin>>nama;   // Meminta user untuk menginisialisasi variable nama
    cout<<"Nama anda adalah "<<nama;
    return 0;
}
```

2.3 KARAKTER & STRING LITERAL

String adalah gabungan dari karakter
Contoh : " Belajar " → Literal String
" B " → Karakter

Panjang String

strlen() → nama fungsi untuk menghitung panjang string

Fungsi strlen() dideklarasikan dalam file string.h
Jadi bila anda ingin menggunakan fungsi strlen(), maka prepocessor directive #include<string.h> harus dimasukkan dalam program diatas main().

Contoh :

```
#include <iostream.h>
#include <string.h>
main()
{
 cout<<strlen("Selamat Pagi.\n")<<endl;
 cout<<strlen("Selamat Pagi.")<<endl;
 cout<<strlen("Selamat")<<endl;
 cout<<strlen("S")<<endl;
 cout<<strlen("");
 return 0;
}
```

Keluarannya:
14
13
7
1
0

Perhatikan, bahwa disetiap akhir baris pernyataan diakhiri dengan tanda titik – koma (semicolon) “ ; “.
Perhatikan, bahwa ‘ \n ‘ dihitung satu karakter. \n disebut newline karakter endl juga merupakan newline karakter (sama kegunaannya seperti \n).
Dalam C++, selain \n terdapat juga beberapa karakter khusus yang biasa dsebut *escape sequence characters*, yaitu

Karakter	Keterangan
\0	Karakteeer ber-ASCII nol (karakter null)
\a	Karakter bell
\b	Karakter backspace
\f	Karakter ganti halaman (formfeed)
\n	Karakter baris baru (newline)
\r	Karakter carriage return (ke awal baris)
\t	Karakter tab horizontal
\v	Karakter tab vertikal
\\	Karakter \
\’	Karakter ‘
\”	Karakter “
\?	Karakter ?
\ooo	Karakter yang nilai oktalnya adalah ooo (3 digit octal)
\xhh	Karakter yang nilai heksadesimalnya adalah hh (2 digit heksadesimal)

2.4 KEYWORD & IDENTIFIER

Dalam bahasa pemrograman, suatu program dibuat dari elemen-elemen sintaks individual yang disebut token, yang memuat nama variable, konstanta, keyword, operator dan tanda baca.

Contoh :
```
#include <iostream.h>
main()
{
  int n=66;
  cout<<n<<endl;   // n sebagai variabel
  return 0;
```

```
}
```

Keluarannya:
66

Program diatas memperlihatkan 15 token, yaitu
main, (,), {, int, n, =, 66, ;, cout, <<, endl, return, 0 dan }
Token n adalah suatu variable
Token 66,0 adalah suatu konstanta
Token int, return dan endl adalah suatu keyword
Token = dan << adalah operator
Token(,), {, ;, dan } adalah tanda baca
Baris pertama berisi suatu prepocessor directive yang bukan bagian sebenarnya dari program

2.5 VARIABEL, DEKLARASI & INISIALISASI

Variabel adalah symbol dari suatu besaran yang merepresentasikan suatu lokasi di dalam memori komputer. Informasi yang tersimpan di dalam lokasi tersebut disebut nilai variable. Untuk memperoleh nilai dari suatu variable digunakan pernyataan penugasan (assignment statement), yang mempunyai sintaks sebagai berikut

variable = ekspresi ;

Yang akan diolah terlebih dahulu adalah ekspresi, baru hasilnya dimasukkan kedalam variable
Tanda “=” adalah operator penugasan.

Contoh :

```
#include <iostream.h>
main()
{
  int n;
  n=66;  // sama juga jika ditulis int n=66;
  cout<<n<<endl;   // n sebagai variabel
  cout<<’n’<<endl;   // end sebagai karakter
  return 0;
}
```

Keluarannya :
66
n

Deklarasi dari suatu variable adalah sebuah pernyataan yang memberikan informasi tentang variable kepada compiler C++. Sintaksnya adlah

type variable ;

dengan type adalah tipe data yang didukung oleh C++, beberapa contohnya yaitu:

Tipe Data	Ukuran Memori (byte)	Jangkauan Nilai	Jumlah Digit Presisi
char	1	-128 hingga +127	-
Int	2	-32768 hingga +32767	-
Long	4	-2.147.438.648 hingga 2.147.438.647	-
float	4	3,4E-38 hingga 3,4E38	6-7
double	8	1.7E-308 hingga 1.7E308	15-16
Long double	10	3.4E-4932 hingga 1.1E4932	19

NB : Untuk mengetahui ukuran memori dari suatu tipe digunakan fungsi sizeof(tipe)
Tipe data dapat diubah (type cast), misalkan:
float x = 3.345;
int p = int(x);
maka nilai p adalah 3 (terjadi truncating).

Contoh Deklarasi dan Inisialisasi

```
int a,b,c;
int p = 55;
```

Dalam contoh, kita mendeklarasikan tiga variable yaitu variable a,b dan c namun belum kita inisialisasi.
Sedangkan variable p kita inisialisasi (diberikan nilai).
Dalam C++, untuk dapat menggunakan suatu variable, variable tersebut minimal kita deklarasikan terlebih dahulu. Apa yang terjadi, jika suatu variable telah dideklarasikan namun belum kita inisialisasi lalu kita mencetak nilai variable tersebut ?

Contoh :

```
#include <iostream.h>
main()
{
  int n;
  cout<<n<<endl;   // n sebagai variabel
  return 0;
}
```

Keluarannya:
18125

Darimana angka 18125 diperoleh ?
>> Jika variable tidak diinisialisai, namun nilai keluarannya diminta, maka compiler dengan bijak akan menampilkan nilai acak yang nilainya tergantung dari jenis compilernya.

2.6 KONSTANTA

1. Konstanta Oktal, digit yang digunakan 0-7
2. Konstanta Heksadesimal, digit yang digunakan 0-9, A-F
3. Konstanta Bernama
   a. Menggunakan keyword const
      Contoh : const float PI = 3.14152965;
      Berbeda dengan variable, konstanta bernama tidak dapat diubah jika telah diinisialisasi
   b. Menggunakan #define
      Contoh : #define PI 3.14152965
      Keuntungan menggunakan #define apabila dibandingkan dengan const adalah kecepatan kompilasi, karena sebelum kompilasi dilaksanakan, kompiler pertama kali mencari symbol #define (oleh sebab itu mengapa # dikatakan prepocessor directive) dan mengganti semua PI dengan nilai 3.14152965.

III. OPERATOR

Operator adalah symbol yang biasa dilibatkan dalam program untuk melakukan sesuatu operasi atau manipulasi.

Contoh : a = b + c * d / 4
a, b, c, d → disebut operand
=, +, *, / → disebut operator

3.1 OPERATOR ARITMATIKA

Operator	Deskripsi	Contoh
+	Penjumlahan (Add)	m + n
-	Pengurangan (Substract)	m – n
*	Perkalian (Multiply)	m * n
/	Pembagian (Divide)	m / n
%	Sisa Pembagian Integer (Modulus)	m % n
-	Negasi (Negate)	-m

NB : Operator seperti operator negasi (-) disebut unary operator, karena membutuhkan hanya satu buah operand

Contoh :

```
#include <iostream.h>
void main()
{
 int m = 82, n = 26;
 cout<<m<<" + "<<n<<" = "<<m+n<<endl;
 cout<<m<<" - "<<n<<" = "<<m-n<<endl;
 cout<<m<<" * "<<n<<" = "<<m*n<<endl;
 cout<<m<<" / "<<n<<" = "<<m/n<<endl;
 cout<<m<<" % "<<n<<" = "<<m%n<<endl;
 cout<<"-"<<m<<" = "<<-m<<endl;
}
```

Keluarannya :

```
82 + 26 = 108
82 - 26 = 56
82 * 26 = 2132
82 / 26 = 3
82 % 26 = 4
-82 = -82
```

Karena tipe datanya adalah int, maka 82/26=3, supaya dapat merepresentasikan nilai yang sebenarnya, gunakan tipe data float.

Cara lain penulisan dengan menggunakan operator aritmatika :

m = m + n ⇔ m += n
m = m - n ⇔ m -= n
m = m * n ⇔ m *= n
m = m / n ⇔ m /= n
m = m % n ⇔ m %= n

3.2 OPERATOR NAIK DAN TURUN (INCREMENT DAN DECREMENT)

Operator increment → ++
Operator decrement → --

Contoh :

```
#include <iostream.h>
main()
{
  int m = 44, n = 66;
  cout<<"m = "<<m<<", n = "<<n<<endl;
  ++m; --n;
  cout<<"m = "<<m<<", n = "<<n<<endl;
  m++; n--;
  cout<<"m = "<<m<<", n = "<<n<<endl;
  return 0;
}
```

Keluarannya :
m = 44, n = 66
m = 45, n = 65
m = 46, n = 64

Terlihat bahwa operator pre-increment dan post-increment memiliki akibat yang sama, yaitu manambah nilai satu pada m dan memasukkan nilai tersebut kembali ke m (m = m+1). Hal yang sama juga terjadi pada operator pre-decrement dan post-decrement yang memberikan akibat yang sama, yaitu mengurangi nilai satu dari n (n = n - 1).

Tetapi bila digunakan sebagai sub-ekspresi, operator post-increment dan pre-increment menunjukkan hasil yang berbeda

Contoh :

```
#include <iostream.h>
main()
{
  int m = 66, n ;
        n = ++m;
  cout<<"m = "<<m<<", n = "<<n<<endl;
        n = m++;
```

```
  cout<<"m = "<<m<<", n = "<<n<<endl;
  cout<<"m = "<<m++<<endl;
        cout<<"m = "<<m<<endl;
  cout<<"m = "<<++m<<endl;
        return 0;
}
```

Keluarannya :
m = 67, n = 67
m = 68, n = 67
m = 68
m = 69
m = 70

Penjelasan :
Dalam penugasan yang pertama, m adalah pre-increment, menaikkan nilainya menjadi 67, yang selanjutnya dimasukkan ke n.
Dalam penugasan kedua, m adalah post-increment, sehingga 67 dimasukkan dahulu ke n baru kemudian nilai m-nya dinaikkan, itu sebabnya mengapa nilai m = 68 dan n = 67.
Dalam penugasan ketiga, m adalah post-increment, sehingga nilai m (= 68) ditampilkan dahulu (ke layar) baru kemudian nilai m dinaikkan menjadi 69.
Dalam penugasan keempat, m adalah pre-increment, sehingga nilai m dinaikkan dahulu menjadi 70 baru kemudian ditampilkan ke layar.
Supaya lebih paham, perhatikan pula contoh dibawah.

Contoh :

```
#include <iostream.h>
main()
{
  int m = 5, n;
  n = ++m * --m;
  cout<<"m = "<<m<<", n = "<<n<<endl;
  cout<<++m<<" "<<++m<<" "<<++m<<endl;
  return 0;
}
```

Keluarannya :
m = 5, n = 25
8 7 6

Penjelasan :
Dalam penugasan untuk n, pertama kali m dinaikkan (++m) menjadi 6, kemudian m diturunkan kembali menjadi 5, karena adanya --m. Sehingga

nilai m sekarang adalah 5 dan nilai m = 5 inilah yang dievaluasi pada saat penugasan perkalian dilakukan.
Pada baris terakhir, ketiga sub-ekspresi dievaluasi dari kanan ke kiri.

3.3 OPERATOR BITWISE

Operator	Deskripsi	Contoh
<<	Geser n bit ke kiri (left shift)	m << n
>>	Geser n bit ke kanan (right shift)	m >> n
&	Bitwise AND	m & n
|	Bitwise OR	m | n
^	Bitwise XOR	m ^ n
~	Bitwise NOT	~m

NB : Seluruh operator bitwise hanya bisa dikenakan pada operand bertipe data int atau char

Berikut ini diberikan tabel kebenaran untuk operator logika
P = A operator B

AND

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	P
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	P
0	0	0
0	1	1
1	0	1
1	1	0

Contoh :

```
#include <iostream.h>
void main()
{
 int m = 82, n = 26;
 cout<<m<<" << 2"<<" = "<<(m<<2)<<endl;
 cout<<m<<" >> 2"<<" = "<<(m>>2)<<endl;
 cout<<m<<" & "<<n<<" = "<<(m&n)<<endl;
 cout<<m<<" | "<<n<<" = "<<(m|n)<<endl;
 cout<<m<<" ^ "<<n<<" = "<<(m^n)<<endl;
```

```
 cout<<"~"<<m<<" = "<<~m<<endl;
 }
```

Keluarannya :
82 << 2 = 328
82 >> 2 = 20
82 & 26 = 18
82 | 26 = 90
82 ^ 26 = 72
~82 = 83

Penjelasan :
Nilai keluaran diatas, tergantung dari jenis compiler yang digunakan. Hasil diatas merupakan keluaran dari compiler Turbo C++.
Pada Turbo C++ besar dari integer adalah 2 byte atau sama dengan 16 bit, untuk mengetahuinya digunakan perintah

```
cout<<sizeof(int)<<endl; // Untuk mengetahui besar dari int
```

Maka :
82_{10} = 0000000001010010_2 dan
26_{10} = 0000000000011010_2
Sehingga :
82 << 2 → $0000000101001000_2 = 328_{10}$
82 >> 2 → $0000000000010100_2 = 20_{10}$
82 & 26 → 0000000001010010_2
0000000000011010_2 &

$0000000000010010_2 = 18_{10}$
dan begitu juga untuk operasi OR dan XOR.
~82 → digunakan 2's complement, yaitu
82_{10} = 0000000001010010_2 lalu dinegasikan tiap bitnya menjadi 1111111110101101_2 kemudian LSB ditambah 1 menjadi $1111111110101110 = 65454_{10}$ nilai ini melebihi jangkauan maksimum int yang berkisar di -32768 sampai 32767, sehingga nilai yang keluar yaitu 83.

Cara lain penulisan dengan menggunakan operator bitwise :
m = m << n ⇔ m <<= n
m = m >> n ⇔ m >>= n
m = m & n ⇔ m &= n
m = m | n ⇔ m |= n
m = m ^ n ⇔ m ^= n

3.4 OPERATOR RELASI

Operator relasi digunakan untuk membandingkan dua buah nilai. Operator ini biasa digunakan dalam instruksi percabangan.

Operator	Deskripsi
==	Sama dengan (bukan assignment)
!=	Tidak sama dengan
>	Lebih besar
<	Lebih kecil
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan

Contoh:

```
#include <iostream.h>
main()
{
  int m = 5, n =7;
  if(m == n) cout<<m<<" sama dengan "<<n<<endl;
  else  if(m != n) cout<<m<<" tidak sama dengan "<<n<<endl;
  else  if(m > n) cout<<m<<" lebih besar dari "<<n<<endl;
  else  if(m < n) cout<<m<<" lebih kecil dari "<<n<<endl;
  return 0;
}
```

Keluarannya :
5 lebih kecil dari 7

3.5 OPERATOR LOGIKA

Operator logika digunakan untuk menghubungkan dua atau lebih ungkapan menjadi sebuah ungkapan berkondisi.

Operator	Deskripsi	Contoh
&&	logic AND	m && n
||	logic OR	m ||n
!	logic NOT	!m

Contoh :

```
#include <iostream.h>
void main()
{
  int m = 166;
  cout<<"(m>=0 && m<=150) → "<<(m>=0 && m<=150)<<endl;
  cout<<"(m>=0 || m<=150) → "<<(m>=0 || m<=150)<<endl;
}
```

Keluarannya :
(m>=0 && m<=150) → 0
(m>=0 || m<=150) → 1

Penjelasan :
Hasil / keluaran dari operator logika adalah 0 dan 1.
0 jika keluarannya salah dan 1 jika keluarannya benar.

3.6 OPERATOR KONDISI

Operator kondisi digunakan untuk memperoleh nilai dari dua kemungkinan
ungkapan1 ? *ungkapan2* : *ungkapan3*
Bila nilai *ungkapan1* benar, maka nilainya sama dengan *ungkapan2*, bila tidak maka nilainya sama dengan *ungkapan3*

Contoh :
```
#include <iostream.h>
main()
{
  int m = 26, n = 82;
  int min = m < n ? m : n;
  cout<<"Bilangan terkecil adalah "<<min<<endl;
  return 0;
}
```

Keluarannya :
Bilangan terkecil adalah 26

Operator relasi, logika dan kondisi akan banyak digunakan pada pernyataan berkondisi

IV. PERNYATAAN (STATEMENTS)

Pernyataan digunakan untuk melakukan suatu tindakan, yaitu

4.1 PERNYATAAN UNGKAPAN

Pernyataan ini merupakan bentuk pernyataan yang paling sering digunakan. Pernyataan ini diakhiri dengan semicolon " ; ".

Contoh : var = 166;
var++;

4.2 PERNYATAAN DEKLARASI

Untuk menggunakan suatu variable minimal variable tersebut dideklarasikan terlabih dahulu

Contoh : int var;

Merupakan contoh deklarasi sebuah variable var dengan tipe data integer (int).

4.3 PERNYATAAN KOSONG

Pernyataan ini tidak melaksanakan apapun.

Contoh : while(ada);

4.4 PERNYATAAN MAJEMUK

Merupakan sejumlah pernyataan yang berada di dalam sebuah blok { }

Contoh :
```
for(var = 0 ; var <10 ; var++)
{
  nilai1 = 100;
  if(!nilai2) nilai2 = 0;
  nilai 3 = nilai1 + nilai2;
}
```

4.5 PERNYATAAN BERLABEL

Pernyataan **goto**, diperlukan untuk melakukan suatu lompatan ke suatu pernyataan berlabel yang ditandai dengan tanda " : ".

Contoh :
```
goto bawah;
pernyataan1;
pernyataan2;
```

:bawah pernyataan 3;

Pada contoh diatas, pada saat goto ditemukan maka program akan melompat pernyataan berlabel bawah dan melakukan pernyataan 3.

4.6 PERNYATAAN KONDISI (CONDITIONAL EXPRESSION)

Pertanyaan Kondisi dibagi menjadi,

4.6.1 Pernyataan *if*

Digunakan dalam pengambilan keputusan

Bentuk umum:

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

Pernyataan1 dilaksanakan jika dan hanya jika kondisi yang diinginkan terpenuhi, jika tidak, lakukan pernyataan2.

Jika anda tidak mempergunakan pernyataan else program tidak akan error, namun jika anda mempergunakan pernyataan *else* tanpa didahului pernyataan *if*, maka program akan error.

Jika pernyataan1 atau pernyataan2 hanya terdiri dari satu baris, maka tanda { } tidak diperlukan, namun jika lebih maka diperlukan.

Bentuknya menjadi :

```
if(kondisi)
 {
    pernyataan1;
    pernyataan1a;
    pernyataan1b;
 }
 else
 {
    pernyataan2;
    pernyataan2a;
    pernyataan2b;
 }
```

Contoh :

```
#include <iostream.h>
void main()
{
int m = 166;
   if(m == 0)cout<<"Nilainya sama dengan nol\n";
   else
    {
      cout<<"Nilainya tidak sama dengan nol\n";
      cout<<"Nilainya sama dengan "<<m<<endl;
```

```
  }
}
```

Selain dari if … else, juga dikenal bentuk if … else if. Adapun perbedaannya diilustrasikan oleh dua contoh dibawah ini.

Contoh 1 :

```
#include <iostream.h>
void main()
{
  int m = 166;
  if(m > 1000) cout<<m<<” lebih besar dari 1000\n”;
  if(m > 100) cout<<m<<” lebih besar dari 100\n”;
  if(m > 10) cout<<m<<” lebih besar dari 10\n”;
}
```

Keluarannya :

```
166 lebih besar dari 100
166 lebih besar dari 10
```

Contoh 2 :

```
#include <iostream.h>
void main()
{
  int m = 166;
  if(m > 1000) cout<<m<<” lebih besar dari 1000\n”;
  else if(m > 100) cout<<m<<” lebih besar dari 100\n”;
  else if(m > 10) cout<<m<<” lebih besar dari 10\n”;
}
```

Keluarannya :

```
166 lebih besar dari 100
```

Mengapa ? Karena contoh 2 sama saja jika ditulis seperti dibawah ini

```
#include <iostream.h>
void main()
{
      int m = 166;
  if(m > 1000) cout<<m<<" lebih besar dari 1000\n";
      else
      {
        if(m > 100) cout<<m<<" lebih besar dari 100\n";
        else if(m > 10) cout<<m<<" lebih besar dari 10\n";
      }
```

}

Contoh diatas disebut juga *nested conditional*

4.6.2 Pernyataan *switch*

Pernyataan *if…else if* jamak dapat dibangun dengan pernyataan *switch.* Bentuk umumnya adalah sebagai berikut.

```
switch(ekspresi)
{
  case konstanta1 : pernyataan1;
  case konstanta2 : pernyataan2;
  case konstanta3 : pernyataan3;
        :
        :
  case konstantaN : pernyataanN;
  default : pernyataanlain;
}
```

Hal – hal yang perlu diperhatikan adalah :

1. Dibelakang keyword case harus diikuti oleh sebuah konstanta, tidak boleh diikuti oleh ekspresi ataupun variable.
2. Konstanta yang digunakan bertipe int atau char
3. Jika bentuknya seperti diatas maka apabila *ekspresi* sesuai dengan konstanta2 maka pernyataan2, pernyataan3 sampai dengan pernyataanlain dieksekusi. Untuk mencegah hal tersebut, gunakan keyword *break;.* Jika keyword *break* digunakan maka setelah pernyataan2 dieksekusi program langsung keluar dari pernyataan *switch.* Selain digunakan dalam *switch,* keyword *break* banyak digunakan untuk keluar dari pernyataan yang berulang (looping).
4. pernyataanlain dieksekusi jika konstanta1 sampai konstantaN tidak ada yang memenuhi *ekspresi.*

Contoh :

```
// Program untuk melihat nilai akhir test
// Nilai A jika nilai diatas 80, B jika 70<= nilai <80
// C jika 50<= nilai <70, D jika 30<=nilai <50
// E jika nilai < 30
#include <iostream.h>
void main()
{
  int nilai;
  cout<<"Masukkan nilai test : ";
  cin>>nilai;
```

```
  switch(nilai/10)
  {
     case 10:
     case 9:
     case 8:cout<<'A'<<endl;break;
     case 7:cout<<'B'<<endl;break;
     case 6:
     case 5:cout<<'C'<<endl;break;
     case 4:
     case 3:cout<<'D'<<endl;break;
     case 2:
     case 1:
     case 0:cout<<'E'<<endl;break;
     default:cout<<"Salah, nilai diluar jangkauan.\n";
    }
}
```

Keluaran :
Masukkan nilai test : *45*
D
Masukkan nilai test : *450*
Salah, nilai diluar jangkauan.
Masukkan nilai test : *nilai_test*
Salah, nilai diluar jangkauan.

Ket : 45, 450 dan nilai_test adalah hasil input dari user

4.6.3 Pernyataan *while*

Digunakan untuk pengambilan keputusan dan looping.

Bentuk :

```
While(kondisi)
{
  pernyataan
 }
```

Jika kondisi tidak terpenuhi, maka *pernyataan* tidak akan dieksekusi.

Contoh:

```
#include <conio.h>
#include <iostream.h>
#define TINGGI 5
#define LEBAR 10
// Program menggambarkan karakter khusus pada sebuah
// koordinat yang ditentukan
```

```
void main()
{
 char matrix[TINGGI][LEBAR];
 int x,y;
 for(y=0;y<TINGGI;y++)
  for(x=0;x<LEBAR;x++)matrix[y][x]='.';
 cout<<"Ketik koordinat dalam bentuk x y(4 2).\n";
 cout<<"Gunakan bilangan negatif untuk berhenti.\n";

 while(x>=0 && y>=0)
 {
  for(y=0;y<TINGGI;y++)
  {
       for(x=0;x<LEBAR;x++)
        cout<<matrix[y][x];
       cout<<"\n\n";
  }
 cout<<"Koordinat : ";
 cin>>x>>y;
 matrix[y-1][x-1]='\xBO';
 }
 getch();
}
```

Penjelasan :
Program ini adalah program menggambar karakter [] jika dicompile di Turbo C++ atau mengambar ♂ jika dicompile di Borland C++. (ditunjukkan oleh karakter 'xBO').
Karena adanya pernyataan *while(x>=0 && y>=0),* maka program akan langsung mengeksekusi pernyataan

```
cout<<"Koordinat : ";
cin>>x>>y;
matrix[y-1][x-1]='\xBO';
```

jika user memasukkan bilangan negatif.
Pada program diatas terdapat fungsi getch(). Gunanya adalah untuk memberhentikan keluaran program sampai user menekan tombol keyboard. Untuk menggunakannya, file conio.h harus diinclude.

4.6.4 Pernyataan *do…while*

Pernyataan do…while mirip seperti pernyataan while, hanya saja pada do…while pernyataan yang terdapat didalamnya minimal akan sekali dieksekusi.

Bentuk :

```
do{
```

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

Terlihat, walaupun kondisi tidak terpenuhi, maka *pernyataan* minimal akan dieksekusi sekali.

Contoh :

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
// Program konversi bilangan desimal ke biner
void main()
{
 int p,n,i=0;
 cout<<"Masukkan bilangan desimal : ";
 cin>>p;


 double A[100];
 do
 {
  A[++i]=p%2;
  p=p/2;
  floor(p);
 } while (p>1);
 cout<<"Nilai binernya : ";
 cout<<p;
 for(n=i;n>=1;n--)
 {
  cout<<A[n];
 }

 getch();
}
```

Penjelasan :
Coba anda masukkan bilangan negatif.
Itulah letak kesalahan program ini (sekaligus untuk menunjukkan sifat dari *do…while*)
Jika anda memasukkan bilangan positif, maka program ini akan menghasilkan nilai biner yang bersesuaian dengan nilai desimal yang anda masukkan.

4.6.5 Pernyataan *for*

Pernyataan for digunakan untuk melakukan looping. Pada umumnya looping yang dilakukan oleh for telah diketahui batas awal, syarat looping dan perubahannya.

Bentuk :

```
for( inisialisasi ; kondisi ; perubahan)
{
pernyataan;
}
```

Selama *kondisi* terpenuhi, maka pernyataan akan terus dieksekusi. Bila pernyataan hanya terdiri atas satu baris pernyataan, maka tanda kurung { } tidak diperlukan.

Contoh :

```
//Program mencetak angka 1-100
#include <iostream.h>
void main()
 {
   for(int x=1 ; x<=100 ; x++) cout<<x<<endl;
 }
```

Bagaimana jika program diatas diubah menjadi

```
#include <iostream.h>
void main()
 {
   for(int x=1 ; ;x++) cout<<x<<endl;
 }
```

Program diatas akan menampilkan bilangan yang banyaknya tak terhingga sehingga dapat membuat komputer anda berhenti bekerja. Contoh diatas juga merupakan prinsip membuat bom program (contohnya : bom mail)

Pernyataan *for* dapat berada di dalam pernyataan *for* lainnya yang biasa disebut *nested for*

Contoh :

```
// Program menghasilkan segitiga pascal
#include <iomanip.h>
#include <conio.h>
#include <iostream.h>

main()
{
```

```
    unsigned int n,a,b,x,s[100],p[100];
    cout<<"Masukkan nilai n: "; cin>>n;

   for(a=0,x=0;a<=n;a++,x+=2)
    {
            cout<<setw(3*n-x);
            s[a]=1;
            p[a]=1;
            for(b=0;b<=a;b++)
            {


                    if (b<1||b==a) cout<<"1"<<setw(4);
                    else
                    {
                            s[b]=p[b];
                            p[b]=s[b-1]+s[b];
                            cout<<p[b]<<setw(4);
                    }

            }

            cout<<endl;
    }
    getch();
    return 0;

}
```

4.7 PERNYATAAN *BREAK*

Pernyataan *break* akan selalu terlihat digunakan bila menggunakan pernyataan *switch*. Pernyataan ini juga digunakan dalam loop. Bila pernyataan ini dieksekusi, maka akan mengkakhiri loop dan akan menghentikan ierasi pada saat tersebut.

4.8 PERNYATAAN *CONTINUE*

Pernyataan *continue* digunakan untuk pergi ke bagian awal dari blok loop untuk memulai iterasi berikutnya.

Contoh :

```
#include <iostream.h>
void main()
```

```
{
    int n;
    for(;;)
    {
      cout<<”Masukkan bilangan integer : “; cin>>n;
      if(n % 2 == 0) continue;
      else if(n % 5 == 0) break;
      cout<<”\tLanjutkan loop berikutnya.\n”;
     }
     cout<<”Akhiri Loop.\n”;
  }
```

Keluarannya :

```
Masukkan bilangan integer : 9
    Lanjutkan loop berikutnya
Masukkan bilangan integer : 8
Masukkan bilangan integer : 5
    Akhiri Loop
```

V. FUNGSI

Fungsi adalah sekumpullan perintah operasi program yang dapat menerima argumen input dan dapat memberikan hasil output yang dapat berupa nilai ataupun sebuah hasil operasi.

Nama fungis yang didefinisikan sendiri oleh pemrogram tidak boleh sama dengan nama build-in function pada compiler C++.

Fungsi digunakan agar pemrogram dapat menghindari penulisan bagian program (kode) berulang-ulang, dapat menyusun kode program agar terlihat lebih rapi dan kemudahan dalam debugging program.

5.1 FUNGSI, DEKLARASI DAN DEFINISI NYA

Pemrogram dapat membuat fungsi yang didefinisikan sendiri olehnya.

Contoh :

```
// Fungsi kuadrat
// tipe_return nama_fungsi (tipe_argument argumen)
float kuadrat ( float x )
{
  return x*x;
}
```

Fungsi yang didefinisikan oleh pemrogram terdiri atas dua bagian, yaitu judul (*header*) dan isi (*body*). Judul dari sebuah fungsi terdiri dari tipe return (float), nama fungsi (kuadrat) dan list parameter (float x). Jadi, judul untuk fungsi kuadrat adalah

```
float kuadrat ( float x )
```

Isi dari sebuah fungsi adalah blok kode yang mengikuti judulnya. Berisi kode yang menjalankan aksi dari fungsi, termasuk pernyataan *return* yang memuat nilai fungsi yang akan dikembalikan ke yang memanggilnya, Isi dari fungsi kuadrat() adalah

```
{
   return x*x;
}
```

Biasanya isi dari fungsi cukup besar. Meskipun demikian, judulnya tetap hanya berada dalam satu baris. Isi dari sebuah fungsi dapat memanggil fungsi itu sendiri (*disebut rekursif*) atau memanggil fungsi lainnya.

Pernyataan *return* dari sebuah fungsi mempunyai dua manfaat, yaitu akan mengakhiri fungsi dan mengembalikan nilainya ke program pemanggil. Bentuk umum pernyataan return adalah :

return *ekspresi*;

Dengan *ekspresi* adalah sebuah ekspresi yang nilainya dinyatakan untuk sebuah variable yang tipenya sama seperti tipe *return.* Terdapat juga fungsi yang tidak memberikan nilai return atau tipe *return*nya void.

Contoh :

```
#include <iostream.h>
void sayHello(char[])  ; // deklarasi fungsi sayHello()
void main()
{
 char n[50];
 cout<<"Masukkan nama anda : "; cin>>n;
 sayHello(n);
}

void sayHello(char nama[]) // definisi funsi sayHello()
{
 cout<<"Selamat datang "<<nama;
}
```

Pengertian deklarasi fungsi berbeda dengan dengan definisi fungsi. Suatu deklarasi fungsi adalah judul fungsi yang sederhana yang diikuti oleh tanda semicolon (;). Sedangkan definisi fungsi adalah fungsi yang lengkap, terdiri dari judul dan isinya. Suatu deklarasi fungsi disebut juga sebagai prototype fungsi.

Suatu deklarasi fungsi seperti layaknya suatu deklarasi variabel, yang memberitahu *compiler* semua informasi yang dibutuhkan untuk mengkompilasi file. *Compiler* tidak perlu mengetahui bagaimana fungsi bekerja, yang perlu diketahui adalah nama fungsi, jumlah dan tipe parameternya, dan tipe balikannya (*return*). Hal ini merupakan informasi yang dimuat secara lengkap dalam judul fungsi.

Juga seperti sebuah deklarasi variabel, suatu deklarasi fungsi harus muncul diatas semua nama fungsi yang digunakannya. Berbeda dengan definisi fungsi, yang dapat diletakkan terpisah dari deklarasinya, dan dapat muncul dimana saja diluar fungsi main() dan biasanya dituliskan setelah fungsi main() atau dalam file terpisah yang jika ingin digunakan tinggal menambahkan prepocessor *#include "nama_file"* pada file utama.

Jika definisi fungsi diletakkan diatas fungsi main() maka deklarsi fungsi tidak diperlukan.

Variabel-variabel yang di list di dalam parameter fungsi disebut parameter-parameter formal atau argumen-argumen formal. Variabel lokal

seperti ini hanya ada selama eksekusi fungsi yang bersangkutan. Dalam contoh dibawah, parameter-parameter formalnya adalah x dan y.

Variabel yang dilist dalam pemanggilan fungsi disebut parameter-parameter actual atau argumen-argumen aktual. Sama seperti variabel lainnya dalam program utama, variabel-variabel tersebut harus dideklarasikan sebelum digunakan dalam pemanggilan. Dalam contoh dibawah, parameter-parameter aktualnya adalah m dan n.

Contoh :

```
// Penggunaan Fungsi Rekursif :
// Program mengecek sebuah bilangan integer atau bukan
#include <iostream.h>
#include <conio.h>
#include <math.h>
void cekInt(double);
void main()
{
 double angka;
 cout<<"Masukan sebuah angka :";cin>>angka;
 cekInt(angka);
}

void cekInt(double n)
{
 if(n>1)cekInt(n-1);
 else if(n<1)cekInt(-n-1);
 else
  {
   if(n>0&&n<1)cout<<n<<"\t Bukan bilangan bulat\n";
   else cout<<n<<"\t Bilangan bulat\n";
  }
}
```

Keluaran :

```
Masukkan sebuah angka : 57
        Bilangan bulat
Masukkan sebuah angka : 0.57
        Bukan bilangan bulat
Masukkan sebuah angka : -24
        Bilangan bulat
```

5.2 NILAI BAWAAN UNTUK ARGUMEN FUNGSI

Salah satu keistimewaan C++ yang sangat bermanfaat dalam pemrograman adalah adanya kemampuan untuk menyetel nilai *default* Argumen fungsi. Argumen-argumen yang mempunyai nilai bawaan nantinya dapat tidak disertakan di dalam pemanggilan fungsi dan dengan sendirinya C++ akan menggunakan nilai bawaan dari argumen yang tidak disertakan.

Contoh :

```
#include <iostream.h>
#include <conio.h>
void sayHello(int);

void main()
{
  sayHello();
}

void sayHello(int n=1)
{
  for(int m=0;m<n;m++) cout<<"Halloo …☺\n";
}
```

Penjelasan :

Jika pada program, argumen sayHello tidak diberikan, maka program akan menampilkan

Halloo …☺

Sebanyak satu kali, namun jika argumen pada fungsi sayHello diberikan, misalkan *sayHello(4),* maka program akan menampilkan

Halloo …☺
Halloo …☺
Halloo …☺
Halloo …☺

Itulah yang disebut dengan nilai default pada fungsi.

5.3 MELEWATKAN ARGUMEN DENGAN REFERENSI

Lihat bab mengenai array dan pointer.

5.4 FUNGSI-FUNGSI BAWAAN C++

Anda dapat menggunakan fungsi-fungsi bawaan C++, misalkan fungsi-fungsi matematika, pengolah kata dan banyak lagi. Sebenarnya (mungkin tidak terasa bagi anda) main juga adalah fungsi, jadi tanpa anda sadari sebenarnya anda telah menggunakan fungsi.
Untuk dapat menggunakan fungsi-fungsi tersebut anda harus meng-include file dimana fungsi tersebut didefinisikan

Misalkan :

- Fungsi – fungsi matematika, anda harus meng-include file math.h
- Fungsi – fungsi pengolah string dan karakter, anda harus meng-include file string.h
- Fungsi clrscr(), getch(), getche() dalam file conio.h

VI ARRAY, STRING & POINTER

6.1 ARRAY

Array adalah kumpulan data-data beripe sama dan menggunakan nama yang sama. Dengan menggunakan rray, sejumlah variabel dapat memakai nama yang sama. Antara satu variabel dengan variabel yang lain di dalam array dibedakan berdasarkan *subscript*. Sebuah *subscript* berupa bilangan didalam tanda kurung siku. Melalui *subscript* inilah masing-masing elemen array dapat diakses. Nilai *subscribe* pertama secara default adalah 0.

C++ tidak mengecek array. Bila anda menyatakan int x[10], ini artinya 10 elemen yang dimulai dari 0. Karena itu elemen terakhir array adalah x[9]. Bila anda salah mereferensikannya dengan x[10], anda akan mendapatkan harga yang tidak terpakai. Akan lebih buruk lagi jika anda memberikan harga ke x[10], yang tidak dapat diterima.

6.1.1 Representasi Array

Misalkan kita memiliki sekumpulan data *ujian* seorang siswa, *ujian* pertama bernilai 90, kemudian 95,78,85. Sekarang kita ingin menyusunnya sebagai suatu data kumpulan *ujian* seorang siswa. Dalam array kita menyusunnya sebagai berikut

```
ujian[0] = 90;
ujian[1] = 95;
ujian[2] = 78;
ujian[3] = 85;
```

Perhatikan :

- Tanda kurung [ ] digunakan untuk menunjukkan elemen array
- Perhitungan elemen array dimulai dari 0, bukan 1

Empat pernyataan diatas memberikan nilai kepada array *ujian*. Tetapi sebelum kita memberikan nilai kepada array, kita harus mendeklarasikannya terlebih dahulu, yaitu :

```
int ujian[4];
```

Perhatikan bahwa nilai 4 yang berada didalam tanda kurung menujukkan
jumlah elemen array, bukan menunjukkan elemen array yang ke-4. Jadi elemen array *ujian* dimulai dari angka 0 sampai 3.

Pemrogram juga dapat menginisialisasi array sekaligus mendeklarasikannya, sebagai contoh :

```
int ujian[4] = {90,95,78,85};
```

Elemen terakhir dari array diisi dengan karakter '\0'. Karakter ini memberitahu kompiler bahwa akhir dari elemen array telah dicapai. Walaupun pemrogram tidak dapat melihat karakter ini secara eksplisit, namun kompiler mengetahui dan membutuhkannya.

Sekarang kita akan membuat daftar beberapa nama pahlawan di Indonesia

```
char pahlawan[3][15] ;
char pahlawan[0][15] = "Soekarno";
char pahlawan[1][15] = "Diponegoro";
char pahlawan[2][15] = "Soedirman";
```

Array diatas terlihat berbeda denga contoh array pertama kita. Perhatikan bahwa pada array *pahlawan* memilii dua buah tanda kurung [ ][ ]. Array seperti itu disebut array dua dimensi. Tanda kurung pertama menyatakan total elemen yang dapt dimiliki oleh array pahlawan dan tanda kurung kedua menyatakan total elemen yang dapat dimiliki setiap elemen array pahlawan. Dalam contoh diatas, tanda kurung kedua menyatakan karakter yang menyatakan nama pahlawan.

6.1.2 Menghitung Jumlah Elemen Array

Karena fungsi *sizeof()* mengembalikan jumlah byte yang sesuai dengan argumennya, maka operator tersebut dapat digunakan untuk menemukan jumlah elemen array, misalnya

```
int array[ ] = {26,7,82,166};
cout<<sizeof(array)/sizeof(int);
```

akan mengembalikan nilai 4, yaitu sama dengan jumlah elemen yang dimiliki array *array*.

6.1.3 Melewatkan Array Sebagai Argumen Fungsi

Array dapat dikirim dan dikembalikan oleh fungsi

- Pada saat array dikirim ke dalam fungsi, nilai aktualnya dapat dimanipulasi

Contoh :

```
#include <iostream.h>
void ubah(int x[]);
void main()
{
  int ujian[] = {90,95,78,85};
```

```
  ubah(ujian);
  cout<<" Elemen kedua dari array ujian adalah "<<ujian[1]<<endl;
}

void ubah(int x[])
{
  x[1] = 100;
}
```

Keluarannya : Elemen kedua dari array ujian adalah 100

6.2 POINTER

Poiter adalah variable yang berisi alamat memori variable lain dan sevara tidak langsung menunjuk ke variable tersebut.

Analoginya – sebagai contoh – Andi berteman dengan Budi, lalu anda ingin mengetahui jumlah keluarga Budi untuk keperluan sensus penduduk. Anda tidak mengetahui alamat Budi, tetapi anda mengenal Andi. Untuk mencari jumlah keluarga Budi, maka pertama-tama anda pergi kerumah Andi, misalnya dirumah no 8321. Sesampai di Andi, Andi membaritahukan kepada anda bahwa alamat Budi ada pada alamat 9821. Kemudian anda pergi ke rumah Budi lalu mencatat jumlah keluarga yang dimiliki Budi yaitu lima orang (misalkan).

Dalam contoh diatas, Andi bertindak sebagai pointer. Andi tidak memberitahukan jumlah keluarga Budi, tetapi Andi memberitahu alamat Budi, di alamat 9821 (alamat Budi) itulah anda mengetahui jumlah keluarga Budi.

Jika *alamat dari* ditunjukkan dengan simbol & dan *isi dari* ditunjukkan dengan symbol *, maka hubungan analogi diatas adalah:

Nama	Alamat	Isi
Andi	8321	9821 = &Budi
Budi	9821	5 = *(&Budi)

Dalam bentuk pointer, ditulis :

```
Andi = &Budi;  // baris 1
Budi = *(&Budi);   // baris 2
```

Subtitusi pernyataan di baris 2 :

```
 Andi = *Andi;
```

Contoh program yang menggambarkan hal tersebut :

```
#include <iostream.h>
void main()
{
   int *Andi; // Andi sebagai pointer
```

```
  int Budi = 5; // Budi bukan pointer, prhatikan perbedaan pada *
  Andi = &Budi // Isi dari Andi yaitu alamat Budi
  cout<<"Isi alamat memori Andi : "<<Andi<<endl;
  cout<<"Isi alamat memori Budi : "<<Budi<<endl;
  cout<<"Isi alamat memori Budi : "<<*Andi<<endl;
  cout<<"Alamat memori Andi : "<<&Andi<<endl;
  cout<<"Alamat memori Budi :"<<&Budi<<endl;
}
```

Keluarannya :
Isi alamat memori Andi : 0x6da72448
Isi alamat memori Budi : 5
Isi alamat memori Budi : 5
Alamat memori Andi : 0x6da7244a
Alamat memori Budi : 0x6da72448

Penjelasan :
Isi alamat memori Andi adalah alamat memori Budi, yaitu 0x6da72448
(alamat ini berbeda-beda tergantung dari komputernya dan ditulis dalam bentuk hexadesimal).
Sedangkan isi alamat memori Budi adalah 5. Cara mengakses isi dari alamat Budi ada dua cara, yaitu mengakses variabel Budi dan mengakses isi dari pointer Andi (*Andi). *Andi dapat juga disebut "*isi dari alamat memori yang ditunjuk oleh Andi"*. Karena alamat memori yang ditunjuk oleh Andi adalah alamat memori Budi, maka dapat dikatakan *"isi dari alamat memori Budi"*.

6.2.1 Pointer - Array

Dalam 6.1 kita telah membahas array, sekarang kita akan melihat bagaimana data disimpan di memori dalam sebuah array.

Contoh :

```
#include <iostream.h>
void main()
{
   int n;
   int array[4] = {10,20,30,40};
   for(n=0;n<4;n++)
   {
      cout<<"Array["<<n<<"] = "<<array[n]<<endl;
      cout<<"\tMenggunakan pointer = "<<*&array[n]<<endl;
      cout<<"\tDisimpan dalam "<<&array[n]<<endl;
    }
}
```

Keluarannya :

```
Array[0] = 10
        Menggunakan pointer = 10
  Disimpan dalam 0xdb72408
Array[1] = 20
        Menggunakan pointer = 20
  Disimpan dalam 0xdb7240a
Array[2] = 30
        Menggunakan pointer = 30
  Disimpan dalam 0xdb7240c
Array[3] = 40
        Menggunakan pointer = 40
  Disimpan dalam 0xdb7240e
```

Penjelasan :
Seperti yang anda lihat, setiap array disimpan dalam 2 byte memori karena kita menggunakan tipe data integer. Perhatikan pula penggunaan pointer dalam pengaksesan nilai setiap elemen array dan pengaksesan alamat setiap array.

- Alamat setiap elemen array dapat diperoleh dengan cara

  &array[n] atau array+n

- Isi dari setiap elemen array dapat diperoleh dengan cara

  array[n] atau *(array+n)

Dibawah ini adalah contoh pengaksesan memori dan isi memori dengan menggunakan cara kedua

Contoh :

```
#include <iostream.h>
void main()
{
  int n;
  int array[4] = {10,20,30,40};
  for(n=0;n<4;n++)
  {
    cout<<"Array["<<n<<"] = "<<array[n]<<endl;
    cout<<"\tMenggunakan pointer = "<<*(array+n)<<endl;
    cout<<"\tDisimpan dalam "<<array+n<<endl;
   }
```

```
}
```

Keluarannya :

```
Array[0] = 10
        Menggunakan pointer = 10
  Disimpan dalam 0xdb72408
Array[1] = 20
        Menggunakan pointer = 20
  Disimpan dalam 0xdb7240a
Array[2] = 30
        Menggunakan pointer = 30
  Disimpan dalam 0xdb7240c
Array[3] = 40
        Menggunakan pointer = 40
  Disimpan dalam 0xdb7240e
```

Mengapa hasil antara dua contoh diatas sama namun sintaksnya berbeda ? Karena array itu sebenarnya telah menunjuk ke alamat memori setiap elemennya, sehingga untuk mengetahui alamat memori setiap elemen array cukup dengan *array* + *n* dengan n bilangan bulat (integer).

6.2.2 Pointer - String

String merupakan bentuk khusus dari array. Oleh karena itu operasi pointer-array tidak jauh berbeda dengan operasi pointer-string

Contoh :

```
#include <iostream.h>
void main()
{
    char nama[5] = “Andi”;
    cout<<”Nama awal : “<<nama<<endl;
    char *ptr;
    ptr = nama;
    *(ptr+3) = ‘y’;
    cout<<”Nama menjadi : “<<nama<<endl;
}
```

Keluarannya :

```
Nama awal : Andi
Nama menjadi : Andy
```

Jadi :

- String adalah array (susunan) dari karakter-karakter
- String dapat diakses dan dimanipulasi lewat pointer
- Alamat awal dari string dapat diperoleh dari namanya

6.2.3 Pointer Sebagai Argumen String

Jika pointer dikirim sebagai argument, maka nilai aktualnya dapat dimodifikasi.

Contoh :

```
#include <iostream.h>
void ubah(char *);
void main()
{
    char *ptr,nama[5] = “Andi”;
    ptr = nama; // ptr sebagai pointer ke variable nama
    cout<<”Nama awal : “<<nama<<endl;
    ubah(ptr);
    cout<<”Nama menjadi : “<<nama<<endl;
 }

void ubah(char *x)
{
  *(x+3) = ‘y’;
}
```

Keluarannya :

```
Nama awal : Andi
Nama menjadi : Andy
```

6.2.4 Alias

Alias adalah nama lain dari suatu variable. Jika suatu perubaha terjadi pada variable alias maka akan berpengaruh kepada variable asli dan begitu juga sebaliknya.

Contoh :

```
#include <iostream.h>
void main()
{
   int uang =10000;
   int &duit = uang;
   cout<<”Nilai uang Rp.“<<uang<<endl;
   cout<<”Nilai duit Rp.“<<duit<<endl;
   uang = 9000;
     cout<<”Uang dibelikan es krim Rp.1000, nilainya menjadi
Rp.“<<uang<<endl;
   cout<<”Nilai duit juga berubah menjadi Rp.“<<duit<<endl;
}
```

Keluarannya :

```
Nilai uang Rp.10000
Nilai duit Rp.10000
```

Uang dibelikan es krim Rp.1000, nilainya menjadi Rp.9000
Nilai duit juga berubah menjadi Rp.9000

Penjelasan :
Perubahan pada *uang* menyebabkan perubahan pada *duit* karena duit memiliki alamat memori yang sama dengan uang. Jadi jika isi dari alamat memori uang atau duit berubah, maka nilai variable duit atau uang juga akan ikut berubah.

6.2.5 Argumen Baris Perintah

Seringkali kita menggunakan perintah *edit file.txt* pada DOS, atau perintah *vi file.txt* pada Unix. Yang dimaksud dengan argumen baris perintah yaitu *file.txt.* Hal seperti itu dapat dibuat dengan menggunakan C++ dengan menyertakan argumen berikut pada fungsi main()

```
void main(int argc, char *argv[])
        {
          …
        }

atau

main(int argc, char *argv[])
        {
          …
    return 0;
        }
```

Keterangan :

- Argc : Beisi jumlah parameter baris ditambah 1
- Argv : Berisi daftar nama argumen dan program, dengan rincian sebagai berikut :
  - argv[0] menunjuk nama program, lengkap dengan alamat path
  - argv[1] menunjuk argumen pertama (kalau ada)
  - argv[n[menunjuk argumen ke-n (kalau ada)

Contoh :

```
// beri nama tes.cpp
#include<iostream.h>
void main(int argc, char *argv[])
{
    for(int a=0;a<argc;a++)cout<<"argv["<<a<<"] = "<<argv[a]<<endl;
}
```

Penjelasan :
Setelah dicompile dan di link akan muncul file tes.exe, misalkan anda simpan di d:\tes.exe
Buka command prompt, pindah ke direktori d:\ ketikkan
Tes argumen1 argument2 argument3, maka akan muncul tampilan

```
argv[0] = D:\TES.EXE
argv[1] = argument1
argv[2] = argument2
argv[3] = argument3
```

Dibawah ini diberikan contoh penggunaan argumen baris perintah yang lain, supaya anda lebih memahami

Contoh :

```
// Program mengubah nilai desimal ke biner
// Simpan dengan nama dec2bin.cpp
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
void main(int argc, char *argv[])
{
 if(argc !=2)
 {
  cerr<<"Pemakaian : dec2bin angka";
  exit(1);
 }
 int p = atoi(argv[1]),n,i=0;
 double A[100];
 do
 {
  A[++i]=p%2;
  p=p/2;
  floor(p);
 } while (p>1);
 cout<<"Nilai binernya : ";
 cout<<p;
 for(n=i;n>=1;n--)
 {
  cout<<A[n];
 }
 getch();
}
```

Jika anda ketikkan *dec2bin 4*, maka outputnya *Nilai binernya : 100*

VII. OPERASI FILE

Ada kalanya seorang programmer harus berhubungan dengan file. Sebagai contoh pada saat pembuatan program database, seorang programmer menyimpan data pada sebuah file dan pada kemudian waktu data tersebut dikeluarkan untuk diolah. Pada prinsipnya operasi yang dilakukan pada file terdiri dari tiga tahap, yaitu :

1. Membuka file
2. Melakukan pemrosesan pada file
3. Menutup file

Dalam melakukan operasi file, programmer membutuhkan fungsi – fungsi yang berhubungan dengan operasi file yang terdapat pada file fstream.h. Oleh sebab itu, untuk dapat melakukan operasi file, maka prepocessor directive berikut ditambahkan.

#include <fstream.h>

7.1 MEMBUKA FILE

Pembukaan dari suatu file mempunyai dua tujuan, yaitu membaca isi file atau untuk menulis ke dalam file tersebut. Dalam C++ penanganan pembukaan file untuk membaca atau menulis ke dalam file berbeda.

7.1.1 Membuka file untuk dibaca

```
ifstream file_objek;
```

Setelah objek_input diciptakan, maka file dibuka dengan cara

```
file_objek.open("nama_file");
```

Adapun dua pernyataan diatas dapat disederhanakan sebagai berikut:

```
ifstream file_objek("nama_file");
```

7.1.2 Membuka file untuk ditulisi

```
ofstream file_objek;
```

Setelah objek_output diciptakan, maka file dibuka dengan cara

```
file_objek.open("nama_file");
```

Adapun dua pernyataan diatas dapat disederhanakan sebagai berikut:

ofstream file_objek(“nama_file”);

7.1.3 Membuka File dengan Modus Tertentu

Pada 7.1.1 dan 7.1.2 pembukaan file hanya bias dilakukan untuk keperluan membaca atau menulis saja, bukan untuk kedua-duanya, dan pada 7.1.2 jika isi dari nama_file sudah ada, maka isi yang lama akan dihapus dan digantikan dengan isi yang baru. Pada suatu waktu mungkin anda memerlukan cara supaya file yang anda buka dapat dipergunakan untuk membaca dan menulis sekaligus, atau isi file yang sudah ada tidak dihapus jika anda ingin menambah isi file yang baru. Untuk keperluan itu, anda harus memformat modus pembukaan file. Adapun modus pembukaan file yang disediakan oleh C++ adalah sebagai berikut :

Modus	Keterangan
ios::app	Membuka file dengan modus keluaran dan memungkinkan operasi penambahan data pada file yang telah ada. Jika file belum ada, maka membuat file baru.
ios::ate	Membuka file dengan modus masukan dan keluaran. Secara otomatis menempatkan pointer file ke posisi akhir file
ios::in	Membuka file dengan modus masukan. Penggunaannya sama dengan ifstream.
ios::out	Membuka file dengan modus keluaran. Penggunaannya sama dengan ofstream.
ios::nocreate	Membuka file yang sudah ada. Jika file yang akan dibuka belum ada, maka C++ tidak akan membuat file baru.
ios::noreplace	Membuka file baru. Jika file sudah ada maka operasi pembukaan menjadi gagal. Jika file belum ada, maka akan dibuat file baru. Hal ini bertentangan dengan ios::nocreate
ios::trunc	Menghapus file yang sudah ada dan menciptakan file baru (replace)

ios::binary	Membuka file dengan operasi baca-tulis secara binary.

Adapun contoh penggunaan dari modus – modus pembukaan file diatas adalah sebagai berikut :

```
fstream file_objek (“nama_file”,ios::in | ios::out);
```

Pernyataan diatas adalah dekalarasi file *nama_file* dengan sehinggan *nama_file* dapat dibaca dan ditulisi.

7.2 PEMROSESAN FILE

Setelah file dibuka, maka dilakukan pemrosesan pada file yang telah dibuka tersebut, antara lain :

7.2.1 Menulis ke File

Contoh :

```
#include<iostream.h>
#include<fstream.h>
void main()
{
      ofstream file_objek;
      file_objek.open("latihan.txt");
      cout<<"Latihan menulis ke dalam sebuah file\n";
      for(int i=1;i<11;i++)
      file_objek<<"Ini adalah baris ke "<<i<<endl;
      file_objek.close();
}
```

Pada direktori dimana anda men-save file tersebut akan terdapat sebuah file bernama latihan.txt

7.2.2 Mambaca Isi File

Contoh :

```
#include<iostream.h>
#include<fstream.h>
void main()
{
     const int MAX = 80;
     char buffer[MAX+1];
     ifstream file_objek;
     file_objek.open("latihan.txt");
```

```
        cout<<"Membaca isi file latihan.txt\n";
        while(file_objek)
        {
          file_objek.getline(buffer,MAX);
          cout<<buffer<<endl;
        }
    }
```

Program ini membaca isi file latihan.txt dan menampilkannya ke layar. File_objek.getline(buffer,MAX) digunakan untuk membaca teks dari file.

7.2.3 Memeriksa Operasi File

C++ menyediakan sejumlah fungsi yang berguna untuk memeriksa kondisi-kondisi pada operasi file, sehingga kejadian kesalahan pada saat eksekusi dapat dikendalikan.

Fungsi Anggota	Kegunaan
Good()	Untuk memeriksa keberhasilan dari suatu operasi file. Jika operasi berhasil dilakukan, maka fungsi ini akan mengembalikan nilai 1 (TRUE)
eof()	Untuk memeriksa apakah pointer telah mencapai akhir file. Jika ya fungsi ini akan mengembalikan nilai 1 (TRUE)
fail()	Untuk memeriksa suatu kesalahan. Fungsi ini dapat digantikan dengan fungsi good() yang dinegasikan.
bad()	Untuk memeriksa apakah ada operasi yang tidak absah. Jika ada, maka fungsi ini akan mengembalikan nilai 1 (TRUE)

Contoh :

```
/* Program ini menghasilkan output yang sama dengan program
   pada contoh 7.2.2 */
#include<iostream.h>
#include<fstream.h>
void main()
{
  const int MAX = 80;
  char buffer[MAX+1];
```

```
    ifstream file_objek;
    file_objek.open("latihan.txt");
    cout<<"Membaca isi file latihan.txt\n";
    while(!file_objek.eof())
    {
      file_objek.getline(buffer,MAX);
      cout<<buffer<<endl;
    }
}
```

Program pada contoh diatas sama saja dengan contoh program pada 7.2.2. Perbedaannya hanya pada

```
while(file_objek)      dan
while(!file_objek.eof())
```

while(file_objek) jika diartikan dalam bahasa sehari – hari adalah jika isi dari file_objek (dalam hal ini, latihan.txt) masih ada, maka baca satu baris pada file latihan.txt lalu pindahkan pointer satu baris ke bawah. Jika isi pada baris tersebut tidak ada, maka hentikan loop. Sedangkan pada while(!file_objek.eof()) dapat diartikan, jika pointer tidak terdapat di baris paling akhir dari file_objek, maka lanjutkan loop. Jika tidak, maka hentikan loop.

7.3 MENUTUP FILE

Setelah pemrosesan file berakhir, maka file perlu ditutup. Langkah ini dilakukan dengan cara

```
file_objek.close();
```

Pemakaian fungsi close() sifatnya optional. Bila anda tidak mempergunakannya, compiler tidak akan mengeluarkan pesan error.

Dibawah ini diberikan contoh – contoh penggunaan operasi file pada suatu system operasi.

Contoh :

1. Program menyalin suatu file

```
// Simpan dengan nama cp.cpp
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
#include <string.h>
#define max 80
void main(int argc,char *argv[])
{
  char buffer[max+1];

  if (argc!=3)
  {
   cerr<<"Pemakaian : cp file_yang_akan_dikopi file_baru\n";
   exit(1);
  }
  strupr(argv[1]);
  ifstream input(argv[1], ios::binary);
  ofstream output(argv[2], ios::binary);
  if (!input)
  {
   cerr<<"File yang akan dikopi tidak ada, periksa kembali !!!\n";
   exit(1);
  }

  for(;;)
  {
    input.read(buffer,max);
    output.write(buffer,max);
    if(input.eof())break;
  }
  input.close();
  output.close();
}
```

2. Program Membaca Isi Suatu File

```
// Simpan dengan nama baca.cpp
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
void main(int argc,char *argv[])
```

```
  {
  clrscr();
  const int max=84;
  char buffer[max+1];
  char namafile[64];
  if(argc!=2)
   {
    cerr<<"Penggunaan : Baca nama_file\n";
    exit(1);
   }
  strcpy(namafile,argv[1]);
  strupr(namafile);

  ifstream input;
  input.open(namafile);
  while(!input)
  {
   cerr<<"File Not Found !!!\n";
   exit(1);
  }
  while (!input.eof())
  {
   input.getline(buffer,max);
   cout<<buffer<<endl;
  }
  input.close();
  getch();
}
```

VIII. STRUKTUR & UNION

8.1 STRUKTUR

Array adalah kumpulan elemen yang bertipe sama. Tetapi struktur memudahkan Anda untuk mengumpulkan variable dengan tipe yang berbeda di dalam satu nama. Fasilitas ini memungkinkan kita untuk melayani sekumpulan data yang rumit sebagai satuan tunggal

Suatu struktur dinyatakan dengan kata kunci *struct*, daftar pernyataan variable yang disebut anggota, yang terlampir dalam tanda kurung { }. Tiap pernyataan anggota dan struktur harus berakhir dengan semicolon (;).

Bentuk deklarasi struktur :

```
Struct nama_struct
{
   anggota_struktur ;
} ;
```

Apabila suatu struktur telah dideklarasikan, struktur ini dapat digunakan untuk mendefinisikan suatu varibel, misalnya :

```
nama_struct  variabel_struktur;
```

merupakan pendefinisian varibel variabel_struktur dengan tipe structur nama_struct.

Anggota struktur dapat diakses dengan menggunakan bentuk :

```
variabel_struktur.anggota
```

Contoh :

```
#include <iostream.h>
void main()
{
  struct keluarga
  {
     char suami[15];
     char istri[15];
     int jumlah_anak;
  };

 keluarga Andi = {“Andi”,”Nina”,3};
 keluarga Budi = {“Budi”,”Ana”,5};
 cout<<”Jumlah anak Bapak Andi “<<Andi.jumlah_anak<<endl;
```

```
  cout<<"Istri Bapak Budi ialah "<<Budi.istri<<endl;
}
```
Keluarannya :
Jumlah anak Bapak Andi 3
Istri Bapak Budi ialah Ana

Perhatikan bahwa pada akhir dari struct diberi tanda semicolon.

8.1.1 Array dari Struktur

Array dari struktur dapat dideklarasikan seperti halnya pendeklarasian array biasa.

Bentuk : nama_array[index]. anggota_struktur;

Contoh :

```
#include <iostream.h>
void main()
{
struct mahasiswa
   {
      long nim;
      int nilai;
   };

   mahasiswa teknik[3];
   teknik[0].nim = 19500376;
   teknik[0].nilai = 78;
   teknik[1].nim = 19500378;
   teknik[1].nilai = 71;
   teknik[2].nim = 19500276;
   teknik[2].nilai = 76;

   cout<<"NIM                    NILAI \n";
   for(int n=0;n<3;n++)
   {
    cout<<teknik[n].nim<<"\t\t"<<teknik[n].nilai<<endl;
   }
 }
```

Keluarannya :

NIM	NILAI
19500376	78
19500378	71
19500276	76

8.1.2 Pointer Sebagai Anggota Struktur

Anda dapat mempunyai karakter, integer, float bahkan pointer untuk dapat dijadikan sebagai tipe data anggota struktur.

Contoh :

```
#include <iostream.h>
void main()
{
     struct kuliah
     {
     char kuliah1[30];
     char kuliah2[30];
      struct kuliah *ptr;
      };

     kuliah semester[3] =
       {{"Komputer 207","Matematika 217",&semester[1]},
       {"Electronic 210","Sistem Kendali 303",&semester[2]},
       {"Analisis Numerik 301","Telekomunikasi 367",&semester[0]}};

  int n;
  for(n=0; n<3;n++)
  {
      cout<<"Isi sedang menujuk ke : ";
      cout<<semester[n].ptr->kuliah1<<endl;
  }
}
```

Keluarannya :

```
Isi sedang menunjuk ke : Electronic 210
Isi sedang menunjuk ke : Analisis Numerik 301
Isi sedang menunjuk ke :  Komputer 207
```

8.1.3 Struktur Sebagai Anggota Struktur

Struktur juga dapat menjadi anggota struktur lain. Contoh diatas juga salah satu contoh struktur sebagai anggota contoh. Untuk dapat lebih memahami dibawah ini diberikan sebuah contoh :

Contoh :

```
#include <iostream.h>
void main()
{
   struct anak
   {
```

```
    char pria[15];
    char wanita[15];
   };

   struct keluarga
   {
    char suami[15];
    char istri[15];
    struct anak ;
    }

    struct anak Andi = {“Tedi”,”Lisa”}
    struct keluarga Budi = {“Budi”,”Ana”,”Ryu”,”Caecilia”};

    cout<<”Anak laki-laki Andi “<<Andi.pria<<endl;
    cout<<”Putri Bapak Budi “<<Budi.wanita<<endl;
 }
```

Keluarannya :
Anak laki-laki Andi Tedi
Putri Bapak Budi Caecilia

8.2 UNION

Union menyerupai struktur, namun mempunyai perbedaan yang nyata. Union biasa dipakai untuk menyatakan suatu memori dengan nama lebih dari satu. Sebagai gambaran, sebuah union dideklarasikan sebagai berikut :

```
union bila_bulat
{
  unsigned int di;
  unsigned char dc[2];
}
```

Pada pendeklarasian seperti ini, di dan dc menempati memori yang sama. Untuk lebih jelasnya, perhatikan contoh berikut :

Contoh :

```
#include <iostream.h>
#include <conio.h>
void main()
{
  union bil_bulat
  {
```

```
    unsigned int di;
    unsigned char dc[2];
  };


  bil_bulat bil;
  bil.di = 0x2345;
  cout<<setiosflags(ios::showbase);
  cout<<hex<<"di      : "<<bil.di<<endl;
  cout<<hex<<"dc[0] : "<<bil.dc[0]<<endl;
   cout<<hex<<"dc[1] : "<<bil.dc[1]<<endl;
  }
```

Keluarannya :

```
di      : 0x2345
dc[0] : 0x45
dc[1] : 0x23
```

Keterangan :

setiosflags(ios::showbase) mengembalikan basis hitung yang digunakan. Hex membuat basis hitung menjadi heksadesimal.

IX. KELAS

9.1 KELAS & STRUKTUR

Kelas merupakan struktur data dari objek. Untuk menjelaskan tentang kelas, akan kita bandingkan bentuk antara struktur dan kelas.

BENTUK	
KELAS	**STRUKTUR**
class nama_class { private : anggota_data; public : fungsi_anggota; } ;	struct nama_struct { anggota_data; } ;

Terlihat perbedaan antara kelas dan struktur, yaitu :

PERBEDAAN	
KELAS	STRUKTUR
Terdapat anggota data dan fungsi anggota. Anggota data biasanya berupa variabel dan fungsi anggota biasanya berupa fungsi.	Hanya terdapat anggota data
Terdapat kata-kata kunci *private* dan *public* yang menentukan hak akses bagi anggota-anggota di dalam kelas. • Private dapat digunakan di dalam kelas untuk memproteksi anggota-anggota tertentu dari kelas, agar tidak dapat diakses dari luar kelas secara langsung. Private merupakan default dari kelas. • Public mengizinkan anggota-anggota yang berada didalamnya bebas di akses dari luar kelas	Hak akses pada struktur sama seperti hak akses *public* pada kelas

9.2 KELAS SECARA UMUM

Konsep penggabungan data dan fungsi seperti diatas disebut encapsulasi, yang diterapkan dalam C++ dengan tipe turunan.

Contoh Kelas :

1. Fungsi didefinisikan di dalam kelas

```
#include <iostream.h>
#include <string.h>
class penduduk
{
  private:
    int id;
    char nama[80];
  public:
    void tampilkan(void)
     {
       cout<<"No. KTP : "<<id<<endl;
       cout<<"Nama    : "<<nama<<endl;
     }

   void set(int idn, char *n)
      {
        id = idn;
        strcpy(nama,n);
       }
};

void main()
{
  penduduk saya;
  saya.set(1234,"Andi");
  saya.tampilkan();
}
```

2. Fungsi anggota didefinisikan diluar kelas

```
#include <iostream.h>
#include <string.h>
class penduduk
{
  private:
    int id;
```

```
    char nama[80];
  public:
    void tampilkan();
    void set(int idn, char *n);
};

void main()
{
  penduduk saya;
  saya.set(1234,"Andi");
  saya.tampilkan();
}

void penduduk :: tampilkan(void)
 {
    cout<<"No. KTP : "<<id<<endl;
     cout<<"Nama    : "<<nama<<endl;
  }
 void penduduk :: set(int idn, char *n)
  {
     id = idn;
     strcpy(nama,n);
   }
```

Keluaran kedua contoh diatas sama saja, yaitu :
No. KTP : 1234
Nama   : Andi

Penjelasan :
Saya merupakan objek dari *class penduduk*.Pada contoh 2 fungsi didefinisikan diluar, oleh karena itu pada pendefinisian fungsi harus memiliki bentuk :

tipe_return_fungsi nama_kelas :: nama_fungsi (parameter)

Hal ini untuk memberitahu kompiler bahwa fungsi tersebut merupakan anggota dari kelas *nama_kelas.* Simbol (::) merupakan operator resolusi lingkup.

9.3 KONSTRUKTOR

Konstruktor adalah fungsi anggota yang mempunyai nama yang sama dengan nama kelas. Kegunaannya :

- Mengalokasikan ruang bagi sebuah objek
- Memberikan nilai awal terhadap anggota data suatu objek

- Membentuk tugas-tugas umum lainnya

Contoh :

```
#include <iostream.h>
class jumlah
{
   public:
    int jumlah1;
    int jumlah2;
    jumlah();
};

jumlah objek1,objek2;
void main()
{
   cout<<"Didalam main() \n";
   cout<<"objek1.jumlah1 adalah "<<objek1.jumlah1<<endl;
   cout<<"objek1.jumlah2 adalah "<<objek1.jumlah2<<endl;
   cout<<"objek2.jumlah1 adalah "<<objek2.jumlah1<<endl;
   cout<<"objek2.jumlah2 adalah "<<objek2.jumlah2<<endl;
}

jumlah::jumlah()
{
   cout<<"Didalam jumlah() \n";
}
```

Keluarannya :

```
Didalam jumlah()
Didalam jumlah()
Didalam main()
objek1.jumlah1 adalah 0
objek1.jumlah2 adalah 0
objek2.jumlah1 adalah 0
objek2.jumlah2 adalah 0
```

Kesimpulan :

- Nama konstruktor sama dengan nama kelas
- Konstruktor tidak mempunyai nilai balik
- Konstruktor harus diletakkan di bagian public, coba saja anda meletakkan konstruktor dalam contoh diatas dibagian private.
- Konstruktor dijalankan dengan sendirinya pada saat objek diciptakan (dalam contoh diatas yaitu objek1 dan objek2). Bahkan konstruktor dijalankan sebelum fungsi main() dijalankan.

9.4 DESTRUKTOR

Destruktor adalah fungsi anggota yang mempunyai nama yang sama dengan nama kelas ditambah symbol tilde (~) didepannya.

Contoh :

```
#include <iostream.h>
class jumlah
{
  public:
   int jumlah1;
   int jumlah2;
   ~jumlah();
};

jumlah objek1,objek2;
void main()
{
  cout<<"Didalam main() \n";
  cout<<"objek1.jumlah1 adalah "<<objek1.jumlah1<<endl;
  cout<<"objek1.jumlah2 adalah "<<objek1.jumlah2<<endl;
  cout<<"objek2.jumlah1 adalah "<<objek2.jumlah1<<endl;
  cout<<"objek2.jumlah2 adalah "<<objek2.jumlah2<<endl;
}

jumlah::~jumlah()
{
  cout<<"Didalam jumlah() \n";
}
```

Keluarannya :

```
Didalam main()
objek1.jumlah1 adalah 0
objek1.jumlah2 adalah 0
objek2.jumlah1 adalah 0
objek2.jumlah2 adalah 0
Didalam jumlah()
Didalam jumlah()
```

Kesimpulan :

- Nama konstruktor sama dengan nama kelas ditambah tanda tilde (~) di depannya
- Destruktor tidak mempunyai nilai balik
- Destruktor harus diletakkan di bagian public, coba saja anda meletakkan destruktor dalam contoh diatas dibagian private.

- Destruktor dijalankan dengan sendirinya pada saat objek akan sirna (dalam contoh diatas yaitu objek1 dan objek2).

9.5 INHERITANCE (PEWARISAN)

C++ memungkinakan suatu kelas mewarisi data ataupun fungsi anggota kelas lain. Sifat seperti ini disebut pewarisan. Kelas uyang mewarisi sifat kelas lain disebut kelas turunan (derived class). Sedangkan kelas yang mewariskan sifat ke kelas lain lain disebut kelas dasar (base class).

Untuk memahami tentang konsep pewarisan, marilah kita lihat contoh berikut ini.

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

class Basis
{
   private :
    int alpha;
    int bravo;
   public :
    void info_basis()
    {
       cout<<"info_basis() dijalankan…"<<endl;
    }
  };

class Turunan : public Basis
{
   public :
    void info_turunan()
    {
        cout<<"info_turunan() dijalankan…"<<endl;
     }
  };

void main()
{
   clrscr();
   Turunan anak;
    anak.info_basis();
    anak.info_turunan();
}
```

Keluarannya :
info_basis() dijalankan…
info_turunan() dijalankan…

Pada contoh diatas, terdapat kelas bernama *Basis* dan *Turunan*. Dalam hal ini :

- *Basis* adalah kelas dasar
- *Turunan* adalah kelas turunan

Kelas *Turunan* mewarisi sifat-sifat dari kelas *Basis* .
Perhatikan pernyataan pada main() :

```
Turunan anak;
Anak.info_basis();
```

Sekalipun info_basis() dideklarasikan pada kelas Basis, ia juga diwariskan pada kelas Turunan. Namun bagaimana mekanisme pewarisannya? Jawaban dari pertanyaan ini terletak pada topik berikut.

Pada pendeklarasian kelas Turunan terdapat baris sebagai berikut :

```
Class Turunan : public Basis
```

Pada baris ini terdapat kata-kata kunci tersebut pada konteks ini ? Untuk melihat kegunaan kata-kata kunci ini, perhatikan terlebih dulu program berikut.

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

class Basis
{
  private :
   int alpha;
   int bravo;
  public :
   void info_basis()
   {
     cout<<"info_basis() dijalankan…"<<endl;
   }
 };

class Turunan : Basis
{
  public :
   void info_turunan()
   {
```

```
        cout<<"info_turunan() dijalankan…"<<endl;
      }
  };

  void main()
  {
     clrscr();
     Turunan anak;
      anak.info_basis();
      anak.info_turunan();
  }
```

Perbedaan program di atas dengan program sebelumnya terletak pada kata-kunci public. Pada program diatas, kata kunci public pada baris yang berisi class Turunan dibuang.

Apabila program dikompilasi, kesalahan akan terjadi, yakni pada pernyataan :

```
anak.info_basis();
```

Kesalahan menyatakan bahwa Basis::info_basis() tidaklah dapat diakses pada main(). Lalu, apa artinya?

Bentuk seperti :

```
class Turunan : Basis
```

sebenarnya mempunyai makna yang sama dengan :

```
class Turunan : private Basis
```

Maksudnya yaitu semua anggota yang bersifat public (dan juga protected) pada kelas dasar (Basis) diwariskan ke kelas turunan (Turunan) sebagai anggota yang bersifat private.

Sedangkan kalau pewarisan dilakukan dengan public, semua anggota yang bersifat public pada kelas dasar diwariskan ke kelas turunan seperti apa adanya pada kelas basis.

MODUL PELATIHAN C++
DIVISI KOMPUTER HME ITB

DISUSUN OLEH : MOCH IRWAN HARAHAP
2002