MODUL PRAKTIKUM

PEMROGRAMAN KOMPUTER (FIS-134)

Denny Darmawan, M.Sc.
(darmawan@uny.ac.id)

PROGRAM STUDI FISIKA
FAKULTAS MATEMATIKA & ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI YOGYAKARTA

MODUL 1 : PENGANTAR BAHASA C

Pendahuluan

Dalam praktikum pemrograman komputer ini, kita akan menggunakan bahasa C untuk belajar menyusun program komputer berdasarkan algoritma yang telah disusun.

Terdapat banyak jenis bahasa pemrograman, seperti: Fortran, BASIC, Pascal, Java, dan lainnya. Bahasa-bahasa pemrograman ini digolongkan ke dalam bahasa pemrograman aras tinggi (*high level programming language*). Selain bahasa aras tinggi, terdapat juga bahasa aras rendah (*low level programming language*) yaitu bahasa *assembly* yang lebih dekat dengan bahasa mesin (kode yang dimengerti oleh komputer).

Bahasa C merupakan bahasa pemrograman yang berada di antara bahasa aras rendah dan aras tinggi (*medium level programming language)*. Selain memiliki keunggulan dari bahasa pemrograman aras tinggi, yaitu menggunakan perintah yang lebih mudah dipahami oleh manusia, bahasa C juga memiliki kecepatan eksekusi mendekati kecepatan eksekusi bahasa aras rendah dan juga kemampuan memanipulasi alamat data secara langsung. Keunggulan inilah yang kemudian menjadikan bahasa C banyak dipilih oleh programmer komputer untuk menyusun aplikasi (software) komputer, mulai dari sistem operasi (*operating system*) seperti Microsoft-Windows, Mac atau Linux hingga aplikasi yang berjalan di dalam sistem operasi seperti aplikasi pengolah kata (*word processor)* dan pengolah data.

Sekilas Bahasa C

Awalnya, program komputer harus dibuat dengan kode biner langsung. Untuk memudahkan, diciptakan bahasa assembly yang lebih mudah diingat dibandingkan susunan perintah dalam kode biner. Namun, bahasa assembly masih dirasa cukup rumit terutama untuk menyusun perintah yang kompleks. Untuk itu diciptakanlah bahasa Fortran (Formula Translator) pada tahun 1954 yang sudah dianggap sebagai bahasa pemrograman aras tinggi. Dari bahasa ini selanjutnya dikembangkan bahasa Algol (Algorithmic Language) pada tahun 1958, kemudian berkembang menjadi CPL (Combined Programing Language), pada tahun 1963. Dari CPL dikembangkan BCPL (Basic CPL) pada tahun 1967. Dua tahun berikutnya, muncul bahasa B yang dikembangkan dari BCPL (dan diyakini merupakan pemendekan dari nama BCPL).

Pada tahun 1972, Dennis Ritchie dari Laboratorium Bell di Amerika Serikat mengembangkan bahasa C yang didasarkan pada bahasa B (dan pemilihan nama C diyakini merupakan kelanjutan dari alfabet B). Meskipun awalnya bahasa C dikembangkan untuk keperluan menyusun sistem operasi, namun banyak aplikasi komputer yang juga ditulis dalam bahasa ini dan merupakan bahasa pemrograman paling populer di kalangan programmer komputer. Pada tahun 2007, muncul bahasa D yang dikembangkan menjadi penerus bahasa C. Namun hingga saat ini, bahasa C masih merupakan bahasa pemrograman yang paling banyak digunakan.

Apa yang diperlukan?

Untuk menyusun program komputer dalam bahasa C dan dapat menjalankannya, ada beberapa hal yang diperlukan: pengetahuan mengenai struktur bahasa C, *text editor* dan *compiler*. Text editor digunakan untuk menuliskan kode program (kemudian disebut *source-code*). Apabila Anda menggunakan sistem operasi Microsoft-Windows, Anda dapat menggunakan aplikasi Notepad sebagai *text editor*, atau dapat juga menggunakan aplikasi gratis Notepad++ yang memiliki kemampuan *syntax highlighting*. Ingat bahwa *text editor* berbeda dengan *word processor*, sehingga Anda tidak dapat menggunakan aplikasi MS-Office atau OpenOffice untuk menuliskan program Anda.

Agar kode program yang Anda tulis dapat dimengerti dan dijalankan oleh komputer, maka kode tersebut harus diterjemahkan (di-*compile*) ke dalam bahasa mesin oleh compiler. Compiler yang tersedia bebas (bahkan gratis) dan populer diantaranya adalah **gcc** (GNU C Compiler). Compiler ini berjalan pada sistem operasi Linux, namun dapat juga dijalankan dalam sistem operasi MS-Windows melalui aplikasi seperti MinGW atau Cygwin.

Alih-alih menggunakan *text editor* dan *compiler* secara terpisah, kita dapat menggunakan aplikasi gabungan yang dikenal sebagai IDE (*Integrated Development Environment*) yang selain memiliki *text editor*-nya sendiri juga dilengkapi *compiler* yang sudah terintegrasi di dalamnya. Di sistem operasi MS-Windows, IDE untuk pemrograman C yang cukup populer dan dapat digunakan secara gratis adalah Dev-C++ yang dapat diperoleh di www.bloodshed.net. Meskipun IDE ini lebih ditujukan untuk menyusun program dalam bahasa C++ (yang merupakan pengembangan bahasa C untuk pemrograman berorientasi objek), namun dapat juga digunakan untuk menyusun program dalam bahasa C. Dev-C++ memanfaatkan *compiler* **gcc** yang dijalankan di atas aplikasi MinGW. Ketika Anda meng-*install* aplikasi Dev-C++, *compiler* **gcc** dan aplikasi MinGW juga akan otomatis ter-*install*. Dengan menggunakan IDE, Anda dapat menuliskan kode program dan meng-

compile-nya langsung di dalam IDE tersebut.

Struktur Dasar Bahasa C

Perhatikan program sederhana dalam bahasa C berikut:

```
#include<stdio.h>

main()
{
      printf("Halo dunia \n");
}
```

Tuliskan program di atas dengan memanfaatkan IDE Dev-C++, simpan dengan nama **coba.c** di drive C dan *compile*. Jika tidak ada pesan kesalahan, buka konsole (command-prompt MS-Windows) dan masuk ke drive **C:** dengan mengetik perintah **cd** kemudian ketik perintah `coba` dan tekan tombol enter. Hasil yang muncul di layar adalah tulisan:

```
Halo dunia
```

Penjelasan program:

Program di atas diawali dengan perintah **#include<stdio.h>**. Perintah **#include** merupakan salah satu jenis *preprocessor* (pengarah). Perintah ini akan meminta komputer membaca file **stdio.h** (yang dikategorikan sebagai file *header*) sebelum mulai membaca perintah lainnya. File header ini berisi sekumpulan fungsi yang akan digunakan dalam program yang disusun. File **stdio.h** dipanggil karena fungsi **printf()** yang dinyatakan dalam file header **stdio.h** akan digunakan dalam program yang disusun.

Selanjutnya terdapat fungsi:

```
main()
{
}
```

Fungsi **main()** merupakan fungsi yang harus ada dalam setiap program yang menggunakan bahasa C karena komputer akan memulai membaca perintah dari fungsi ini.

Fungsi ini menjadi fungsi induk bagi fungsi-fungsi yang lain. Setelah nama fungsi yang diikuti sepasang tanda kurung (yang menunjukkan bahwa main merupakan sebuah fungsi), terdapat tanda kurung kurawal buka dan tutup. Semua perintah atau fungsi yang digunakan harus dinyatakan di dalam tanda kurung kurawal ini. Kumpulan perintah atau fungsi yang berada di antara tanda "{" dan" }" selanjutnya disebut sebagai *block*.

Di dalam fungsi **`main`** terdapat fungsi **`printf()`**. Fungsi ini meminta komputer untuk menampilkan string kontrol atau argumen fungsi tersebut ke layar. Pada contoh program di atas, string yang ditampilkan berupa kalimat: **`Halo dunia`**. String control harus diapit oleh sepasang tanda petik " ".

Pada string control di atas terdapat tanda **`\n`**. Tanda ini meminta pada komputer untuk berganti baris, sehingga kursor akan terletak di bawah kalimat **`Halo dunia`**. Untuk menampilkan tanda \ dan " maka harus didahului dengan tanda \ sehingga:

\n	meminta kursor untuk berganti baris
\"	menampilkan karakter " ke layar
\\	menampilkan karakter \ ke layar
\t	menampilkan karakter tab ke layar

Ingat bahwa semua perintah atau fungsi yang ada di dalam blok harus diakhiri dengan tanda ; (titik-koma).

Fungsi masukan

Selain fungsi **`printf()`** yang digunakan untuk menampilkan karakter ke layar, terdapat juga fungsi yang dapat digunakan untuk meminta masukan. Salah satu fungsi tersebut adalah **`scanf()`**. Perhatikan contoh berikut:

```
#include<stdio.h>
main()
{
      char namamu[10];
      printf("Masukkan nama Anda: ");
      scanf("%s",&namamu);
      printf("Hallo %s \n ", namamu);
}
```

Tuliskan program di atas, simpan dengan nama **coba2.c** dan *compile*. Jika tidak ada pesan kesalahan, buka konsole (command-prompt MS-Windows) dan ketik perintah **coba2** kemudian tekan tombol enter. Hasil yang muncul di layar adalah komputer akan meminta Anda memasukkan sebarang nama, kemudian menampilkan:

Hallo *nama*

dengan *nama* diganti karakter yang Anda masukkan.

Penjelasan program:

Pada program di atas, perintah **Char nama[10];** meminta komputer untuk menyiapkan alamat memori yang akan diisi oleh maksimal 10 karakter dan alamat ini akan diberi label **namamu** (yang selanjutnya akan disebut sebagai *penanda* atau *identifier*). **Char** merupakan salah satu tipe data yang memberi tahu komputer bahwa penanda **namamu** akan berisi data dengan tipe string (beberapa karakter).

Perintah **scanf("%s", &namamu);** berarti komputer akan meminta masukan berupa string (karakter) dan disimpan pada alamat yang telah diberi label *namamu*. Tanda **%** dinamakan penentu format, dan karena format yang diminta adalah string maka pada program diatas dipilih **%s.** Tanda **&** disebut operator alamat sehingga **&namamu** berarti data yang dimasukkan akan disimpan pada alamat dari namamu.

Terakhir, perintah **printf("Hallo %s \n", namamu);** meminta komputer untuk menampilkan string *Hallo* yang diikuti oleh karakter yang tersimpan pada **namamu** dan kemudian kursor berpindah baris.

Menuliskan Komentar

Ketika menuliskan kode program, adakalanya Anda perlu menyisipkan komentar agar orang lain yang membaca program Anda dapat memahami kode Anda dengan lebih mudah atau Anda juga memerlukannya untuk mengingat program apa yang Anda tulis. Di dalam bahasa C, komentar didahului oleh tanda // atau sepasang tanda /* dan */. Apabila komentar cukup pendek dan hanya menempati satu baris biasanya digunakan tanda // sedangkan jika komentar Anda cukup panjang dan menempati beberapa baris maka didahului oleh /* dan diakhiri oleh */. Komentar tidak akan diproses ketika kode program di-*compile*. Maka

program sebelumnya dapat dituliskan dengan komentar:

```
/* ---------------------------------------
              Program Pertama
--------------------------------------- */
#include<stdio.h>

main()
{
      char namamu[10];
      printf("Masukkan nama Anda: ");
      scanf("%s",&namamu);       // memasukkan nama
      printf("Hallo %s \n ", namamu);
}
```

Tugas

Buatlah program yang meminta masukan nama, NIM dan prodi (program studi) kemudian menampilkannya ke layar. Simpan file dengan nama **`kelas_1_NIM.c`** dimana kelas dan NIM diganti dengan kelas dan NIM Anda sendiri.

MODUL 2 : VARIABEL DAN TIPE DATA

Deklarasi Variabel

Dalam pemrograman, data yang akan diolah disimpan dalam variabel. Agar dapat digunakan, sebelumnya variabel dideklarasikan dahulu beserta tipe data yang akan disimpan dalam variabel tersebut. Format penulisan deklarasi variabel adalah:

```
tipe_variabel nama_variabel;
```

Penamaan variabel mengikuti aturan penulisan *identifier* (penanda), yaitu:

- Nama variabel harus diawali dengan huruf (tidak boleh diawali angka), selanjutnya dapat diikuti dengan angka atau underscore (_).
- Nama variabel tidak boleh mengandung operator aritmatika (+ - / *) dan karakter khusus lain selain huruf, angka dan underscore.
- Nama variabel tidak boleh mengandung spasi (disarankan diganti dengan underscore) dan maksimal hanya mengandung 32 karakter.
- Harap diingat bahwa dalam pemrograman bahasa C identifier bersifat *case sensitive* (huruf besar dan kecil dibedakan), sehingga identifier a_1 akan dianggap berbeda dari A_1.
- Dan terakhir, identifier tidak boleh memakai kata kunci (*keywords*) yang telah digunakan oleh bahasa C. Bahasa C memiliki 32 kata kunci, yaitu:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Terdapat beberapa macam tipe data dalam bahasa C:

- Tipe data integer (bilangan bulat), yaitu: *short, int, long*.
- Tipe data real (bilangan nyata), yaitu: *float, double*.
- Tipe data char (karakter), yaitu: *char*.
- Tipe data void (kosong)

Tipe data yang berbeda akan memiliki ukuran yang berbeda dan nilai batas minimal /maksimal yang berbeda pula yang ditunjukkan dalam tabel berikut:

Tipe data	Ukuran (byte)	Nilai minimal	Nilai maksimal	Format
short	2	-32768	32767	%i, %d
int	2	-32768	32767	%i, %d
long	4	-2147483648	2147483647	%i, %d
float	4	± 3,4 E-38	± 3,4 E+38	%f
double	8	± 1,7 E-308	± 1,7 E+308	%f
char	1			%c

Di dalam bahasa C tidak dikenal tipe string, namun demikian variabel dengan tipe string dapat dibentuk melalui array dari variabel dengan tipe char dan dengan format **`%s`** :

```
char nama_variabel[ukuran_string];
```

Untuk beberapa variabel yang memiliki tipe data yang sama, pendeklarasian tipe data dapat dilakukan dalam satu baris dengan dipisahkan tanda koma. Contoh pendeklarasian variabel adalah sebagai berikut:

```
int a;            //variabel a memiliki tipe int
float x, y, z;    // variabel x, y dan z memiliki tipe float
char n, m[10];    // variabel n dan m memiliki tipe char
```

Nilai atau data yang tersimpan dalam sebuah variabel dapat dinyatakan bersamaan dengan deklarasi variabelnya, atau bisa juga dimasukkan setelahnya. Untuk memasukkan nilai atau data ke dalam variabel, digunakan operator *assignment* berupa tanda sama

dengan. Sebagai contoh:

```
int   a = 30, b = 2, c = 100+50;
float x = 3.5;
char  m='Y', n[10] = "namaku";
```

Perhatikan bahwa untuk string, data yang dimasukkan harus diapit oleh tanda petik ganda (" "), sementara untuk karakter tunggal cukup diapit oleh tanda petik tunggal (' ').

Menyatakan konstanta

Adakalanya kita memerlukan data yang tidak boleh berubah (berupa konstanta). Dalam bahasa C, konstanta dinyatakan dengan preprocessor **`#define`**. Sintaksnya adalah:

```
#define nama_variabel nilai
```

Ingat bahwa untuk pernyataan preprocessor tidak diakhiri oleh tanda titik koma (;). Contoh pernyataan konstanta adalah sebagai berikut:

```
#define pi 3.14
```

Dengan menyatakan **`pi`** sebagai konstanta melalui preprocessor **`#define`**, maka tidak perlu lagi dilakukan deklarasi variabel untuk variabel **`pi`**.

Format data

Untuk menampilkan isi data yang terkandung dalam suatu variabel, perlu dinyatakan terlebih dahulu format datanya dengan sintaks:

```
printf("%format_data",nama_variabel);
```

Format data untuk tipe data integer dapat menggunakan **`%i`** atau **`%d`**, untuk tipe data float menggunakan **`%f`**, untuk tipe data char menggunakan **`%c`** sedangkan untuk string menggunakan **`%s`**.

Panjang ruang yang disediakan untuk menampilkan data dapat diatur dengan menyisipkan bilangan bulat yang menunjukkan seberapa panjang ruang yang diinginkan. Sebagai contoh untuk tipe data integer, perintah **`printf("a=%6d",10);`** akan memberikan tampilan:

a	=					1	0

Sedangkan untuk tipe data real, perintah **`printf("a=%6.2d",10.5);`** akan memberikan tampilan:

a	=		1	0	.	5	0

yaitu, lebar ruang yang ditempati adalah 6 field (termasuk tanda titik desimal) dan diberikan 2 field untuk angka di belakang tanda titik desimal.

Operator

Data yang tersimpan dalam variabel selanjutnya akan diolah. Untuk melakukan pengolahan data ini salah satunya dengan menggunakan operator. Ada beberapa macam operator yang dapat digunakan, diantaranya adalah: operator assignment (yang sudah ditunjukkan sebelumnya), operator logika, operator aritmatika, dan operator perbandingan. Operator logika dan perbandingan akan dipelajari pada modul berikutnya. Untuk operator aritmatika ditunjukkan pada tabel di bawah:

Operator	Simbol
Perkalian	*
Pembagian	/
Penjumlahan	+
Pengurangan	-
Sisa (modulus)	%

Konversi Tipe Data

Ketika dalam sebuah pernyataan ditemukan perbedaan tipe data, maka akan dilakukan penyamaan tipe data (terjadi konversi tipe data). Dalam operasi aritmatik, aturan yang digunakan dalam melakukan konversi ini adalah:

- Semua tipe data char dan short akan dikonversi ke tipe int, sedangkan tipe data float

akan dikonversi ke double.

- Apabila salah satu variabel dalam operasi aritmatik bertipe double, maka variabel lainnya (dalam operasi itu) akan dikonversi ke tipe double.
- Apabila salah satu variabel dalam operasi aritmatik bertipe long, maka variabel lainnya (dalam operasi itu) akan dikonversi ke tipe long.

Untuk operasi assignment, variabel atau data yang ada di kanan tanda '=' akan dikonversi ke tipe variabel yang ada di kiri '='. Sebagai contoh, untuk pernyataan berikut:

```
int  a;
float b, c;
c=a*b;
```

variabel a bertipe int, sedangkan b dan c bertipe float. Karena pada pernyataan **`c=a*b`** terjadi perbedaan tipe data, maka terjadi konversi. Dari aturan konversi untuk operasi aritmatik, variabel b yang bertipe float akan diubah ke tipe double. Selanjutnya, karena b bertipe double, pada operasi a*b variabel a akan diubah ke tipe double sehingga a dan b sekarang memiliki tipe data yang sama, dan hasil operasi a*b disimpan dengan tipe double. Terakhir, karena pada pernyataan **`c=a*b`** variabel c (yang ada di kiri tanda '=') bernilai float, maka hasil operasi **`a*b`** (yang ada di kanan '=') akan diubah menjadi tipe float juga.

Pengubahan tipe variabel ini dapat berpengaruh pada hasil perhitungan. Selain berbeda pada jangkauan nilainya (lihat tabel tipe data), terdapat juga perbedaan ketelitian data yang tersimpan. Sebagai contoh, amati perbedaan ketelitian tipe float dan double melalui contoh berikut:

```
#include<stdio.h>
main()
{
      float  a = 10.123456789123456789;
      double b = 10.123456789123456789;

     printf("a = %.16f \n",a);
     printf("b = %.16f \n",b);
}
```

Jika program di atas dijalankan, akan dihasilkan:

```
a = 10.1234569549560547
b = 10.1234567891234573
```

Nampak bahwa variabel a yang bertipe float hanya mampu memberikan ketelitian hingga digit ke-8, sedangkan variabel b yang bertipe double mampu memberikan ketelitian hingga digit ke-16.

Dengan adanya konversi tipe data dan perbedaan ketelitian tipe data ini maka diharapkan pemilihan tipe data yang akan dipakai dilakukan dengan cermat.

Pelajari contoh program berikut:

```
#include<stdio.h>
#define pi 3.14

main() {
      float radius, luas, keliling;

      printf("masukkan besarnya jejari lingkaran: ");
      scanf("%f",&radius);

      luas = pi*radius*radius;
      printf("lingkaran berjejari %10.2f memiliki luas %10.2f \n",
radius, luas);
}
```

Tugas

Buatlah program yang meminta masukkan radius lingkaran pada alas silinder dan tinggi silinder kemudian hitung volume dan luas permukaannya. Tampilkan dalam bentuk data detail silinder: radius alas, tinggi, volume dan luas. Simpan program Anda dengan nama file: **`kelas_2_NIM.c`**

MODUL 3: SYARAT (KONDISI)

Operator Relasi dan Logika

Dalam pemrograman, sering kali ditemui kasus dimana suatu perintah hanya akan dilaksanakan jika kondisi atau syarat tertentu telah dipenuhi sebelumnya. Dalam bahasa C, ada beberapa pernyataan yang bisa digunakan agar komputer mengecek dahulu apakah syarat sudah dipenuhi sebelum melakukan suatu pernyataan, yaitu: **`if`**, kombinasi **`if-else`** dan **`switch`**.

Untuk mengetahui apakah suatu syarat telah dipenuhi, diperlukan nilai benar atau salah yang dibangkitkan melalui operasi relasi atau melalui operasi logika. Operasi relasi (hubungan) memerlukan operator relasi berikut:

Operator	Keterangan
>	Lebih besar
<	Lebih kecil
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
= =	Sama dengan
!=	Tidak sama dengan

Contoh:

`x > 3` : bernilai benar jika **`x`** lebih dari 3 dan bernilai salah jika **`x`** kurang dari 3

`a == 5` : bernilai benar jika **`a`** bernilai 5 dan salah jika **`a`** bernilai selain 5

Selain menggunakan operator relasi, nilai benar atau salah dapat juga dibangkitkan melalui operator logika berikut:

Operator	Simbol	Keterangan
AND	&&	Bernilai benar hanya jika keduanya benar
OR	||	Bernilai salah jika keduanya salah
NOT	!	Bernilai benar jika ekspresi salah dan bernilai salah jika ekspresi benar
XOR	^	Bernilai benar jika kedua ekspresi berbeda nilainya

Contoh:

`a && b` : bernilai benar hanya jika **`a`** dan **`b`** keduanya bernilai benar, selain itu salah

`a || b` : bernilai salah hanya jika **`a`** dan **`b`** keduanya bernilai salah, selain itu benar

Pernyataan `if`

Melalui pernyataan **`if`**, maka suatu pernyataan (atau kumpulan pernyataan) akan dilaksanakan apabila syarat dipenuhi. Format pernyataan **`if`** adalah:

`if` (syarat) pernyataan ;

Apabila pernyataan yang harus dikerjakan lebih dari satu, maka kumpulan pernyataan itu dikalang oleh kurung kurawal buka dan tutup (disebut blok pernyataan):

```
if (syarat) {
pernyataan1 ;
pernyataan2 ;
pernyataan3; }
```

Pada pernyataan **`if`** di atas, kumpulan atau suatu pernyataan hanya akan dikerjakan apabila kondisi dipenuhi, dan jika tidak, maka kumpulan pernyataan tersebut diabaikan. Terkadang suatu program juga menginginkan komputer mengerjakan pernyataan yang lain apabila syarat tidak dipenuhi. Dalam hal ini, kita menggunakan pernyataan **`if-else`** dengan format berikut:

`if` (syarat) pernyataan_A **`else`** pernyataan_B ;

Apabila pernyataan yang harus dikerjakan lebih dari satu, maka digunakan blok pernyataan:

```
if (syarat) { pernyataan_A1;
              pernyataan_A2; }
else { pernyataan_B1;
       pernyataan_B2; }
```

Pernyataan **`if-else`** juga dapat dinyatakan dalam bentuk bertingkat :

```
if  (syarat_A) { pernyataan_A1;
                 pernyataan_A2; }
else if (syarat_B) { pernyataan_B1;
                     pernyataan_B2;}
else { pernyataan_C1;
       pernyataan_C2;}
```

Pada kasus tertentu, if-else bertingkat dapat digantikan dengan perintah switch dengan format sebagai berikut:

```
switch (ekspresi)
{
      case nilai1  :  pernyataan1 ;
                      break ;
      case nilai2  :  pernyataan2 ;
                      break ;
      case nilai3  :  pernyataan3 ;
                      break ;
      default      : pernyataan ;
}
```

Pada pernyataan **`switch`** di atas, ekspresi harus berupa pernyataan yang bernilai integer atau karakter. Ekspresi dapat diganti dengan suatu variabel bertipe integer atau karakter dengan nilai1, nilai2, nilai3 berupa nilai yang mungkin dari variabel tersebut.

Contoh Program:

```
#include<stdio.h>

main() {
      int angka;
      printf("Masukkan sembarang angka bulat (0-100) : ");
      scanf("%d",&angka);

      if (angka <= 100 && angka >= 80)
            printf("angka Anda berada di antara 80 dan 100 \n");
      else if (angka < 80 && angka >=60 )
            printf("angka Anda berada di antara 60 dan 80 \n");
      else if (angka < 60 && angka >=40)
            printf("angka Anda berada di antara 40 dan 60 \n");
      else if (angka < 40 && angka >=20)
            printf("angka Anda berada di antara 20 dan 40 \n");
      else if (angka < 20 && angka >=0)
            printf("angka Anda berada di antara 0 dan 20 \n");
      else
            printf("angka Anda berada di luar jangkauan 0-100 \n");
}
```

Tugas

Buatlah program yang meminta masukan berupa nilai antara 0 hingga 100 dan program memberikan keluaran berupa skor A, B, C, D dan E dengan kriteria berikut:

nilai >= 90 mendapat skor A

75 <= nilai < 90 mendapat skor B

50 <= nilai < 75 mendapat skor C

40 <= nilai < 50 mendapat skor D

nilai < 40 mendapat skor E

Perhatikan bahwa program yang Anda buat harus dapat menerima nilai pecahan. Simpan file Anda dengan nama: **`kelas_3_NIM.c`**

MODUL 4: PERULANGAN

Dalam membuat program, terkadang ada beberapa perintah atau pernyataan yang ingin diulang beberapa kali selama nilai suatu variabel belum memenuhi nilai tertentu. Alih-alih menuliskan pernyataan tadi secara berulang, kita dapat menyatakannya melalui pernyataan perulangan **`for`** atau **`while`**. Sebelumnya, harus dipahami terlebih dahulu istilah operator *decrement* (penurunan) dan *increment* (kenaikan).

Operator decrement dan increment

Untuk menambah nilai suatu variabel berisi data integer dengan nilai 1 dapat dinyatakan melalui ekspresi:

```
x = x + 1 ;
```

sehingga apabila nilai awalnya nilai **`x`** adalah 3, maka setelah pernyataan di atas nilai **`x`** menjadi 4.

Di dalam bahasa C, pernyataan tersebut dapat dituliskan secara ringkas sebagai

```
++x ;
```

yang dikenal dengan nama *pre increment operator*. Artinya, data pada suatu variabel yang dikenai operator tersebut akan ditambah nilai 1 terlebih dahulu sebelum digunakan pada ekspresi berikutnya.

Selain operator *pre increment*, terdapat juga operator *post increment*, yang dituliskan sebagai

```
x++ ;
```

Pada pernyataan di atas, isi variabel **`x`** akan dipakai terlebih dahulu pada suatu pernyataan sebelum ditambah dengan nilai 1. Operator *decrement* juga memiliki bentuk yang mirip, kecuali operasi penjumlahan diganti dengan operasi pengurangan. Selengkapnya ditunjukkan pada tabel berikut:

Operator	Nama	Keterangan
`var++`	Post increment	Nilai yang ada di sebuah variabel digunakan dahulu, kemudian baru ditambah dengan satu
`++var`	Pre increment	Nilai yang ada di sebuah variabel ditambah satu, kemudian baru digunakan

var--	Post decrement	Nilai yang ada di sebuah variabel digunakan dahulu, baru dikurang satu
--var	Pre decrement	Nilai yang ada di sebuah variabel dikurang satu, baru kemudian digunakan

Perulangan dengan *for*

Format perintah perulangan dengan menggunakan **for** adalah sebagai berikut:

```
for (ekspresi_1; ekspresi_2; ekspresi_3) pernyataan ;
```

Sedangkan bentuk perulangan dimana terdapat lebih dari satu pernyataan dapat dinyatakan dalam bentuk blok pernyataan:

```
for (ekspresi_1; ekspresi_2; ekspresi_3) {
      pernyataan1;
      pernyataan2;
      pernyataan3;
      }
```

dimana **ekspresi_1** merupakan nilai awal dari variabel yang mengontrol perulangan, **ekspresi_2** menyatakan syarat kapan perulangan akan berhenti, sedangkan **ekspresi_3** merupakan pengendali besarnya kenaikan (atau penurunan) nilai dari variabel yang mengendalikan operasi perulangan.

Contoh:

```
#include<stdio.h>

main() {
      int x;
      for (x=1; x<=3; x++) printf("isi x = %d \n", x);
}
```

Program di atas akan menampilkan hasil:

```
isi x=1
isi x=2
isi x=3
```

Perulangan dengan *while*

Selain dengan menggunakan pernyataan **for**, perulangan dapat juga memanfaatkan pernyataan **while**. Perbedaannya adalah, jika pada pernyataan **for** kita sudah mengetahui berapa cacah perulangannya, pada pernyataan **while** cacah perulangan akan ditentukan oleh syarat atau kondisi yang dijadikan kontrol perulangan.

Format penulisan perintah perulangan dengan **while** adalah sebagai berikut:

```
while (syarat) {
      pernyataan;
      pernyataan;
      }
```

Pada dasarnya, perulangan dengan for dapat diubah ke perulangan dengan **while**. Dengan menggunakan istilah yang dipakai pada format perulangan dengan **for**, maka jika dinyatakan ke bentuk perulangan dengan **while** adalah:

```
ekspresi_1;
while (ekspresi_2) {
      pernyataan1;
      pernyataan2;
      pernyataan3;
      ekspresi_3
      }
```

sehingga contoh program sebelumnya dapat dituliskan sebagai:

```
#include<stdio.h>

main() {
      int x;
```

```
      x=1;
      while (x<=3) {
      printf("isi x = %d \n", x);
      x++ ;
      }
}
```

Program di atas juga akan menghasilkan tampilan:

```
isi x=1
isi x=2
isi x=3
```

Pelajari Contoh Program berikut:

```
#include<stdio.h>

main() {
      int i, n, x;
      printf("Masukkan sebuah bilangan bulat: ");
      scanf("%d",&n);

      x = 0;
      for (i=1; i<=n; i++) {
      x=x+i; // hitungan ini berulang n kali
      }
      printf("\n Jumlahan dari 0 hingga %d adalah %d \n",n,x);
}
```

Tugas

Buatlah program untuk menghitung nilai faktorial suatu bilangan. Simpan dengan nama file **kelas_4_nim.c**.

MODUL 5 : PERULANGAN BERTINGKAT

Perintah perulangan (*looping*) atau iterasi dapat disusun secara bertingkat yang dikenal dengan istilah *loop* dalam *loop* (*nested loop*). Perhatikan contoh program berikut:

```
#include<stdio.h>

main() {

      int m, k;
      for (m=1; m<=3; m++) {
            printf("mulai iterasi ke-%d \n",m);
            for (k=1; k<=2; k++)
                  printf("ini sub-iterasi ke-%d dari iterasi ke-%d
\n",k,m);
      }
}
```

Pada program di atas, variabel **m** menjadi variabel kendali perulangan utama. Perulangan dilakukan sebanyak 3 kali. Untuk setiap perulangan, dilakukan 2 perulangan lagi (sub-iterasi) dimana variabel **k** menjadi variabel kendali perulangan di dalam perulangan utama. Apabila program dijalankan, maka di layar terminal akan diperoleh hasil:

```
mulai iterasi ke-1
ini sub-iterasi ke-1 dari iterasi ke-1
ini sub-iterasi ke-2 dari iterasi ke-1
mulai iterasi ke-2
ini sub-iterasi ke-1 dari iterasi ke-2
ini sub-iterasi ke-2 dari iterasi ke-2
mulai iterasi ke-3
ini sub-iterasi ke-1 dari iterasi ke-3
ini sub-iterasi ke-2 dari iterasi ke-3
```

Pelajari contoh berikut:

```
#include<stdio.h>

main() {
      int k, m, x;

      printf(" Nilai Faktorial Bilangan \n");
      printf(" -------------------------\n");

      for (k=1; k<=10; k++){
            x=1;
            for (m=1; m<=k; m++) {
                  x=x*m;
            }
            printf(" Nilai %d! adalah %d \n",k,x);
      }
}
```

Tugas

Buatlah program untuk menampilkan tabel perkalian dalam bentuk matriks 10x10. Contoh hasil program ditunjukkan sebagai berikut:

```
-------------------------------------------------
|    |   1   2   3   4   5   6   7   8   9  10   |
-------------------------------------------------
|  1 |   1   2   3   4   5   6   7   8   9  10   |
|  2 |   2   4   6   8  10  12  14  16  18  20   |
|  3 |   3   6   9  12  15  18  21  24  27  30   |
|  4 |   4   8  12  16  20  24  28  32  36  40   |
|  5 |   5  10  15  20  25  30  35  40  45  50   |
|  6 |   6  12  18  24  30  36  42  48  54  60   |
|  7 |   7  14  21  28  35  42  49  56  63  70   |
|  8 |   8  16  24  32  40  48  56  64  72  80   |
|  9 |   9  18  27  36  45  54  63  72  81  90   |
| 10 |  10  20  30  40  50  60  70  80  90 100   |
-------------------------------------------------
```

MODUL 6 : PERULANGAN DENGAN WHILE

Selain menggunakan pernyataan **for**, perulangan dapat juga dinyatakan dengan pernyataan **while**. Terdapat 2 macam cara menyatakan perulangan dengan **while**, yaitu melalui pernyataan **do-while** atau **while**. Pernyataan **do-while** memiliki format berikut:

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

sedangkan pernyataan **while** memiliki format berikut:

```
while (syarat) {
      pernyataan;
      pernyataan;
      }
```

Perbedaan dari kedua pernyataan tersebut adalah: pernyataan **while** menguji syarat di awal, sehingga apabila syarat tidak dipenuhi, maka blok pernyataan tidak akan dikerjakan. Sedangkan pernyataan **do-while** menguji syarat di belakang, sehingga meskipun syarat tidak dipenuhi, blok pernyataan tetap akan dikerjakan satu kali.

Perhatikan dua contoh program berikut:

```
#include<stdio.h>

main() {
        int m=6;

        while (m <= 5) {
              printf("Halo \n");
        }
}
```

```
#include<stdio.h>

main() {
        int m=6;

        do {
             printf(" Halo \n ");
        } while (m<=5);
}
```

Pada program di sebelah kiri, perintah mencetak karakter "**Halo**" di layar tidak akan dikerjakan, karena syarat yang diminta tidak dipenuhi. Sedangkan pada program di sebelah kanan, pada layar akan tercetak karakter "**Halo**" meskipun syarat tidak dipenuhi. Namun, karena syarat baru diuji di akhir blok perintah, maka perintah mencetak karakter ke layar tetap dikerjakan satu kali.

Pelajari contoh program berikut:

```
#include<stdio.h>

main() {
      int i, m, k;

      do {
            printf("\n Masukkan sebuah bilangan bulat: ");
            scanf("%d",&m);

            k = 0;
            for (i=1; i<=m; i++) k=k+i;
            printf("\n Jumlahan dari 0 hingga %d adalah %d \n",m,k);
      } while (m != 0);
}
```

Tugas

Buatlah program untuk terus meminta masukan bilangan dan baru berhenti jika diisikan bilangan nol. Selanjutnya, jumlahkan seluruh bilangan yang telah dimasukkan tadi.

MODUL 7 : SUB-PROGRAM (FUNGSI)

Dalam modul-modul sebelumnya, beberapa fungsi input-output telah digunakan dalam program, yaitu **`printf()`** dan **`scanf()`**. Selain fungsi input-output, terdapat juga fungsi matematik seperti **`sin()`**, **`cos()`** dan **`tan()`** yang akan dipelajari di modul berikutnya. Fungsi (dikenal juga dengan nama sub-program) merupakan blok perintah/pernyataan (kumpulan perintah/pernyataan) untuk mengerjakan tugas tertentu. Fungsi-fungsi seperti **`printf()`**, **`scanf()`**, **`sin()`**, **`cos()`** dan **`tan()`** merupakan fungsi-fungsi bawaan yang ada dalam bahasa C. Namun demikian, fungsi tersebut dapat juga dibuat sendiri. Dalam modul ini akan dipelajari bagaimana cara menyatakan sebuah fungsi dan menggunakannya dalam program.

Sebuah fungsi akan memerlukan masukan yang dikenal sebagai argumen atau parameter. Fungsi juga akan memberikan keluaran yang dalam bahasa C dinyatakan melalui pernyataan **`return()`**. Maka format dari sebuah fungsi adalah:

```
tipe_fungsi nama_fungsi() (argumen) {
      pernyataan1;
      pernyataan2;
      ...
      return(nilai);
}
```

Tipe fungsi dan nama fungsi memiliki aturan yang sama dengan tipe dan nama suatu variabel. Argumen dapat diisi oleh nilai dari variabel, begitu juga dengan nilai yang dikembalikan oleh fungsi **`return()`**.

Sebelum digunakan, fungsi harus dideklarasikan dahulu, lengkap dengan tipe fungsi dan tipe argumennya. Pendeklarasian fungsi ini disebut prototipe fungsi dan diletakkan setelah pernyataan praprosessor. Setiap pernyataan prototipe fungsi harus diakhiri dengan tanda titik-koma (;), namun untuk pernyataan fungsi itu sendiri tidak diberi tanda titik-koma.

Untuk jelasnya, coba pelajari contoh program berikut:

```
//program untuk menentukan nilai logaritma natural (ln)
#include<stdio.h>

float pangkat(float a,int b); //deklarasi prototype fungsi pangkat

main(){
      int n;
      float x, y, hasil=0;

      printf("masukkan sebuah bilangan (antara 0 dan 2): ");
      scanf("%f",&y);

      x = y - 1;
      for (n=1;n<=10;n++)
            hasil = hasil + pangkat(x,n)*pangkat(-1,n+1)/n;
      printf("nilai    logaritma    natural    dari    %.2f    adalah
%.4f\n",y,hasil);
}

float pangkat(float a, int b){
      //fungsi untuk menghitung nilai a pangkat b
      int i;
      float m=1.0;

      for (i=1;i<=b;i++) m=m*a;
      return(m);
}
```

Penjelasan:

Program di atas akan meminta pengguna untuk memasukkan sebuah bilangan, kemudian menghitung nilai logaritma naturalnya menggunakan bentuk deret:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n}$$

Namun, bentuk deret ini hanya konvergen pada interval nilai -1 < x < 1, sehingga hanya akurat untuk menentukan nilai logaritma natural antara 0 dan 2. Keakuratannya juga akan ditentukan oleh banyaknya suku deret yang dilibatkan dalam perhitungan. Semakin banyak akan memberikan nilai yang semakin akurat.

Tugas

Buatlah program untuk menentukan nilai pangkat exponensial (e^x). Gunakan bentuk deret

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots = 1 + \sum_{n=1}^{\infty} \frac{x^n}{n!}$$

MODUL 8 : FUNGSI MATEMATIK

Bahasa C menyediakan fungsi-fungsi matematik bawaan yang tersimpan dalam pustaka (*library*) **math.h**. Agar fungsi-fungsi matematik tersebut dapat digunakan dalam program, maka pustaka **math.h** harus disertakan dalam program melalui pernyataan:

```
#include<math.h>
```

Beberapa fungsi yang terdapat dalam library **math.h** adalah sebagai berikut:

Fungsi	Kegunaan
sin(x)	Menghitung nilai sinus sudut x yang dinyatakan dalam radian
cos(x)	Menghitung nilai cosinus sudut x yang dinyatakan dalam radian
tan(x)	Menghitung nilai tangen sudut x yang dinyatakan dalam radian
asin(x)	Menghitung nilai arcus sinus dari x yang harus bernilai di antara -1 dan 1
acos(x)	Menghitung nilai arcus cosinus dari x yang harus bernilai di antara -1 dan 1
atan(x)	Menghitung nilai arcus tangen dari x
sinh(x)	Menghitung nilai sinus hiperbolik dari x
cosh(x)	Menghitung nilai cosinus hiperbolik dari x
tanh(x)	Menghitung nilai tangen hiperbolik dari x
abs(x)	Menghitung nilai mutlak dari x
log(x)	Menghitung nilai logaritma natural dari x (yaitu ln x)
log10(x)	Menghitung nilai logaritma basis 10 dari x
pow(x,y)	Menghitung nilai x pangkat y
sqrt(x)	Menghitung nilai akar kuadrat dari x
exp(x)	Menghitung nilai exponen dari x (yaitu ex)

Berikut ini contoh program yang melibatkan fungsi matematik bawaan yang ada dalam bahasa C:

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

main() {
      float y0, v0, y, v, t, g=9.8;

      printf("-----------------------------------\n");
      printf("     Program Gerak Jatuh Bebas \n");
      printf("-----------------------------------\n\n");
      printf("Masukkan ketinggian awal : ");
      scanf("%f",&y0);
      printf("Masukkan kelajuan awal : ");
      scanf("%f",&v0);

      t = (v0 + sqrt(v0*v0 + 2*g*y0))/g;
      v = v0 - g*t;

      printf("\n Hasil Perhitungan: \n");
      printf("----------------------\n");
      printf("Waktu yang diperlukan untuk menyentuh tanah : %.2f
s\n",t);
      printf("Kelajuan benda saat menyentuh tanah : %.2f m/s2\n",v);
}
```

Penjelasan:

Program di atas merupakan program simulasi gerak jatuh bebas yang akan meminta masukan ketinggian dan kelajuan awal (y_0 dan v_0) dari benda yang mengalami gerak jatuh bebas. Persamaan yang digunakan adalah:

$$y = y_0 + v_0 t - \frac{1}{2} g t^2$$

Saat benda menyentuh tanah, maka y = 0 sehingga persamaan di atas dapat dituliskan sebagai:

$$\frac{1}{2} g t^2 - v_0 t - y_0 = 0$$

dengan salah satu nilai *t* adalah:

$$t = (v_0 + \sqrt{v_0^2 + 2g\,y_0}) / g$$

menyatakan lamanya benda di udara sebelum menyentuh permukaan tanah. Selanjutnya kelajuan akhir benda ketika menyentuh permukaan tanah ditentukan dari:

$$v = v_0 - g\,t$$

Tugas

Buatlah program untuk menghitung gerak parabolik (proyektil). Gunakan ketinggian awal, kelajuan awal dan sudut elevasi sebagai masukan. Keluarannya berupa waktu yang diperlukan untuk menyentuh tanah, jarak titik tembak dengan titik jatuh dan kelajuan ketika menyentuh tanah.

Persamaan gerak yang dapat digunakan:

$y = y_0 + v_{0y} t - \frac{1}{2} g\,t^2$ dengan $v_{0y} = v_0 \sin\theta$ dan $v_y = v_{0y} - g\,t$

serta

$x = x_0 + v_{0x} t$ dengan $v_{0x} = v_0 \cos\theta$ dan $v_x = v_{0x}\,t$

MODUL 9 : ARRAY (LARIK)

Bentuk array (larik) sebenarnya telah kita temui pada modul pertama. Dalam modul tersebut, bentuk ini dipakai ketika diinginkan variabel bertipe char yang mampu menampung lebih dari satu karakter (disebut string). Namun variabel array tidak hanya bertipe char saja, variabel array dapat juga diberi tipe int atau float. Dengan menggunakan array, maka akan dimiliki sebuah variabel yang memiliki tipe dan nama yang sama, namun setiap elemen array dapat memiliki nilai yang berbeda. Format deklarasi variabel berbentuk array satu dimensi adalah sebagai berikut:

```
tipe_variabel nama_variabel[ukuran];
```

Nilai yang terdapat pada elemen array akan dipanggil dengan menggunakan indeks array. Dalam bahasa C, indeks array akan diawali dari angka nol dan berakhir dengan indeks **`ukuran-1`**. Contoh:

```
int skor[25];
skor[2]=10;
```

Baris pertama pada pernyataan di atas akan meminta komputer untuk menyiapkan variabel bernama skor dan bertipe integer yang memiliki 25 elemen array. Baris berikutnya menyatakan bahwa elemen array ke-3 dari variabel skor diisi dengan nilai 10 (ingat bahwa indeks array dimulai dari indeks 0).

Selain dalam bentuk satu dimensi, array dapat juga dinyatakan dalam bentuk dua dimensi, biasanya digunakan dalam analisis yang melibatkan matriks. Format umumnya adalah sebagai berikut:

```
tipe_variabel nama_variabel[ukuran][ukuran];
```

Misal diperlukan variabel yang merepresentasikan matriks dengan orde 2x3, maka deklarasi variabelnya adalah:

```
int matriks[2][3];
```

sedangkan untuk mengakses nilai pada elemen (0,1) dari matriks di atas diperlukan pernyataan dengan indeks [0][1], misal:

```
matriks[0][1]=20;
```

Perhatikan contoh program berikut:

```
#include<stdio.h>

main() {

      int n, m, matriks[3][3];

      printf("    Program matriks 3x3 \n\n");

      for (n=0;n<3;n++){
            for (m=0;m<3;m++){
            printf("Masukkan elemen (%d,%d) dari matriks: ",n+1,m+1);
            scanf("%d",&matriks[n][m]);
            }
      }
      printf("\n Matriks Anda adalah: \n");
      for (n=0;n<3;n++){
            for (m=0;m<3;m++){
            printf("%4d ",matriks[n][m]);
            }
      printf("\n");
   }
}
```

Tugas

Buatlah program untuk menjumlahkan dua buah matriks yang masing-masing berorde 2x2.