Catatan Singkat
Bahasa C

Oleh :

Inggriani Liem

Jurusan Teknik Informatika

Institut Teknologi Bandung

Versi Online, Agustus 2003

DAFTAR ISI

PENGANTAR 4
Sejarah Singkat dan Versi 4
Aplikasi dalam bahasa C 4
Istilah : 5
STRUKTUR PROGRAM DALAM BAHASA C 6
PEMROSESAN PROGRAM SUMBER DALAM BAHASA C 7
KARAKTER YANG DIPAKAI : 7
JENIS KALIMAT (Statement) dalam Bahasa C 7
Kalimat non-executable: 7
Kalimat *executable*: 8
NAMA dalam Bahasa C 8
Macam-macam nama : 8
Struktur Blok dan nama 8
Mengacu suatu Nama 8
Aturan nama 8
Aturan akses nama : 9
Name space dalam C: 9
LITERAL KONSTANTA 9
Konstanta Integer 9
Konstanta Karakter 9
Konstanta Floating 10
Konstanta Enumerasi 10
Konstanta String *(String Literal)* 10
INSTRUKSI BAHASA C 10
Assignment 10
Kondisional 10
Pengulangan 10
Pencabangan 11
DEKLARASI TYPE DAN VARIABEL 11
Deklarasi Tipe dan Kelas Penyimpanan Variabel 11
KONSTANTA BERNAMA 13
TIPE DASAR 13
POINTER (*) 14
Pointer sederhana 14
Pointer ke fungsi 15
Pointer sebagai sarana passing paramater output 15
Resume pointer 15
ARRAY 15
Array statik 16
Array dinamik 16
STRING 18
ARRAY MULTI DIMENSI & STRING 19
Aritmatika Pointer, *address arithmetic* dalam array 20
STRUKTUR KOMPOSISI (RECORD) 20
UNION 21
BIT FIELD 21
TYPEDEF : *USER DEFINED TYPE* 23
KONVERSI TIPE 24
OPERATOR 25
Operator Numerik 25
Operator Relasional 26
Operator Logika 27
Operator Alamat 27
Operator Assignment dan Compound Assignment 28
Operator Tipe 28
Operator Kondisional 29

Operator Ekspresi Postfix 29
Operator Koma 29
Presedensi Operator 29
PREPROSESOR BAHASA C 30
File Inclusion 30
Macro Substitution 31
Conditional Inclusion 31
FUNGSI DAN PROSEDUR 32
SATU PROGRAM UTUH DALAM BEBERAPA FILE 34
PROGRAM SANGAT BESAR DENGAN VARIABEL GLOBAL 37
INPUT/OUTPUT 38
Format Output 38
Format Input 41
FILE EKSTERNAL 42
PUSTAKA BAHASA C YANG STANDARD 43
TERJEMAHAN DARI NOTASI ALGORITMIK KE C : 44

Catatan :

1. Edisi bulan Agustus 1998 adalah edisi pertama (Draft) setelah catatan bahasa C yang pernah saya terbitkan pada tahun 1992. Jadi edisi ini masih mengandung banyak kesalahan, dan terutama urut-urutan (presedensi) bahan. Mohon koreksi dan komentar.
2. Urutan penyajian bahan adalah per topik, namun belum tentu pembahasan dalam kuliah per topik (bagian yang rumit ditunda dan dibahas setelah pemahaman akan bahan lain).
3. Terjemahan notasi algoritmik ke C hanya digunakan di perkuliahan dengan notasi algoritmik yang dipakai di jurusan IF ITB sesuai dengan diktat Algoritma dan pemrograman prosedural [Liem97]
4. Diktat kecil ini saling melengkapi dengan Contoh program kecil dalam bahasa C yang diterbitkan oleh Jurusan Informatika ITB [Liem98]

Referensi :

[Kern88] Brian W Kernighan & Dennis M. Ritchie : “The C Programming Language”. Prentice Hall, second ed, 1988.

[Liem98] Inggriani Liem : “Diktat pemrograman Prosedural”, bagian I dan II, Jurusan Teknik Informatika ITB, 1997

[Liem98] Inggriani Liem : “Contoh Porgram Kecil dalam Bahasa C”, Jurusan Teknik Informatika ITB, 1998

PENGANTAR

Sejarah Singkat dan Versi

Bahasa C dikembangkan oleh Dennis M. Ritchie dan Brian W. Kernighan pada awal tahun1970. Bahasa C berkembang di lingkungan UNIX (±90% sistem operasi UNIX ditulis dalam bahasa C).
Standar yang ada:

- Definisi Kernighan & Ritchie (K&R);
- ANSI-C (X-3.159 -1989-);
- Definisi AT&T (untuk superset C, C++).

Versi pada PC misalnya:

- Lattice C;
- Microsoft C/Microsoft QuickC;
- Turbo C/Borland C++;

Pada tahun 1986, dikembangkan superset C (kompatibel dengan C, namun dilengkapi dengan kemampuan pemrograman berorientasi objek) oleh Bjarne Stroustrup [Stroustrup-86], yaitu bahasa C++ *(C with Class)*.
Catatan:
Ringkasan ini memakai standar ANSI C. Contoh-contoh sedapat mungkin dipilih bebas dari implementasi kompilator tertentu. Jika ada contoh yang spesifik terhadap implementasi, implementasi kompilator yang dipakai akan disebutkan.

Aplikasi dalam bahasa C

Bahasa C banyak dipakai untuk:

1. membuat sistem operasi dan program-program sistem,
2. pemrograman yang "dekat" ke perangkat keras (misalnya untuk kontrol peralatan),
3. membuat tool kit,
4. menulis program aplikasi (misalnya dBase, WordStar, Lotus123).

Kelebihan bahasa C, sehingga terpilih untuk aplikasi-aplikasi tersebut, adalah kemampuannya untuk membuat kode yang compact, efisien tanpa mengorbankan readability (beda dengan bahasa assembly yang efisien namun susah dibaca, atau bahasa tingkat tinggi lain yang enak dibaca namun tidak efisien). Walaupun tak dapat diingkari bahwa program dalam bahasa C lebih sulit dibaca (karena compact) dibandingkan dengan bahasa tingkat tinggi yang lain.

Istilah :

Blok: sekumpulan kalimat C yang ditulis di antara { dan }.
Definisi:memberitahukan sifat (property) objek dan sekaligus mengalokasikan memori untuk objek.
Deklarasi: memberitahukan sifat (property) objek (terutama berkaitan dengan tipe).
Inisialisasi : memberikan nilai objek
Deklarasi Global : deklarasi yang berlaku dalam satu unit translasi (file).
Deklarasi Lokal : deklarasi yang hanya berlaku dalam blok tempat deklarasi.
Objek: daerah memori bernama (sama dengan variabel).
Lvalue: ekspresi yang me-referensi suatu objek. Secara mudah, lvalue adalah nilai di sebelah kiri ekspresi assignment Contoh: identifier dengan tipe dan kelas penyimpanan tertentu atau hasil indireksi dari pointer.
Prototipe: deklarasi fungsi, menyatakan nama, tipe return value, nama dan tipe parameter formal (argumen).
Body : realisasi dari fungsi
Scope: daerah program tempat suatu nama dikenal.

STRUKTUR PROGRAM DALAM BAHASA C

Berikut ini adalah struktur sebuah program utama dalam bahasa C. Contoh lengkap dapat dilihat pada Contoh program kecil

```
/* Nama File : ….. */
/* identitas perancang/penulis */
/* Deskripsi ringkas dari program */

<tipe> main([int argc, char** argv[, char** envp]])

/* Keterangan program  */


/* KAMUS */


/* Algoritma/deretan instruksi yang executable */


 return(<retval>);
}
```

Keterangan:

1 Tidak ada aturan penulisan ketat tentang posisi karakter seperti dalam bahasa-bahasa berorientasi kolom (misalnya FORTRAN). Walaupun demikian, disarankan agar pengetikan program dilakukan dengan indentasi, agar program mudah dibaca oleh manusia.Fungsi main() adalah nama fungsi yang menandai awal dan akhir eksekusi program. Suatu program dalam bahasa C harus mempunyai satu fungsi yang bernama main.

2 Return value main akan diberikan ke lingkungan yang menjalankan program (biasanya berupa angka integer, yang menyatakan tingkat kesalahan yang terjadi saat terminasi program). Kebanyakan program tidak mengembalikan nilai sehingga deklarasinya adalah

3 `void main () {        };`

4 Standard yang dipakai di kelas adalah bahwa main akan mengembalikan nilai integer 0 jika semua instruksinya berlangsung dengan baik

5 Parameter argc menyatakan jumlah argumen yang diberikan pada program pada saat dipanggil (nama program dianggap sebagai argumen, sehingga jumlah argumen minimum adalah 1).

6 Parameter argv adalah array string berakhiran '\0' (null-terminated). String pertama, argv[0], adalah nama program. String yang mengikuti adalah argumen-argumen berikutnya yang diberikan saat pemanggilan program.

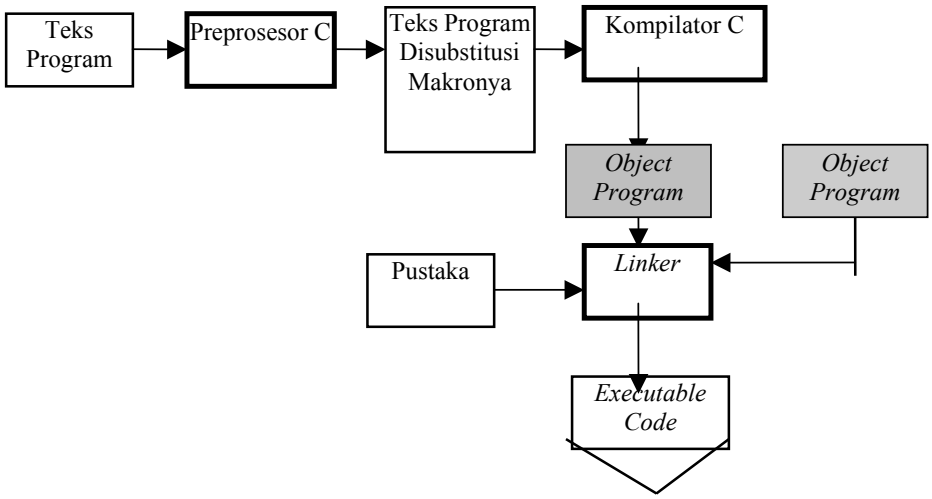
7 Parameter envp adalah pointer ke array string lingkungan. Akhir array ditandai dengan NULL.

8 Bentuk lain deklarasi parameter main (artinya sama dengan di atas, hanya char** <nama> diganti char* <nama>[]):

```
<tipe> main(int argc, char* argv[], char* envp[])
```

PEMROSESAN PROGRAM SUMBER DALAM BAHASA C

C didukung oleh suatu preprosesor yang melakukan substitusi makro dan manipulasi teks lain pada program. Perintah preprosesor selalu diawali dengan karakter "#" dan diakhiri dengan akhir baris. Teks program sumber diproses sebagai berikut:



KARAKTER YANG DIPAKAI :

```
Alphabet, besar dan kecil : A-Z dan a-z
 Angka                    : 0 - 9
 Karakter khusus          : + - * / = < > ( ) [ ] . , ; :
                            { } ? # ! ~ & | % \
 Karakter "blank"         : spasi, tabulasi horisontal/verti-
                            kal, CR, LF, FF
```

JENIS KALIMAT (STATEMENT) DALAM BAHASA C

Kalimat dalam bahasa C selalu diakhiri dengan tanda titik koma (';'). Kalimat dapat digolongkan menjadi dua yaitu kalimat yang tidak dieksekusi (komentar, assignement) dan yang dieksekusi (instruksi)

Kalimat non-executable:

Kalimat non-executable adalah kalimat yang bukan dieksekusi, melainkan sekedar komentar, atau kalimat untuk melakukan deklarasi nama (yang mungkin sekaligus melakukan inisialisasi nilai)

Komentar

- Dituliskan di antara tanda `/*` dan `*/`. Disarankan agar setiap komentar dituliskan dalam satu baris walaupun dalam bahasa C dimungkinkan untuk membuat komentar yang terdiri dari lebih dari satu baris
- Pada beberapa kompilator, di antara tanda `//` dan <eol> *(end of line)*.

Deklarasi

Bagian deklarasi mewakili "Kamus" yaitu semua nama yang didefinisikan dan akan dipakai.
Nama yang harus dideklarasikan sebelum dipakai dalam lingkup yang sesuai adalah :

- Deklarasi nama konstanta dan nilainya
- Deklarasi struktur dan union
- Deklarasi nama type yang didefinisikan
- Deklarasi nama variabel dan type yang sudah didefinisikan (baik oleh bahasa C atau didefinisikan sebelumnya) . Deklarasi nama variabel dapat diikuti dengan inisialisasi nilainya atau tidak.
- Deklarasi tipe turunan:

- Deklarasi fungsi (prototype)

Kalimat *executable*:

Kalimat *executable* adalah instruksi yang akan dikerjakan oleh komputer, meliputi pemberian harga, kondisional, pengulangan atau kalimat percabangan sebagai berikut:

```
-  Assignment (dengan operator =)
-  Kondisional
     if (<kondisi>) { };
      if () {  } else { };
      switch
-  Pengulangan
    while
    do while
    for
- Pencabangan
    goto
    continue
    break
    return
```

NAMA DALAM BAHASA C

Nama ***(identifier)*** dipakai untuk mengenali suatu objek dalam sebuah program.

Macam-macam nama :

. nama fungsi
. nama tipe data, struktur, union, enumerasi
. nama konstanta
. nama objek/variabel
. nama label

Struktur Blok dan nama

Sebuah "Blok" dalam bahasa C dituliskan di antara tanda kurung kurawal buka "{" dan kurung kurawal tutup "}". Sebuah blok dapat mengandung deklarasi data (kamus) dan instruksi. Bahasa C tidak mengenal deklarasi blok bertingkat *(nested)* seperti Pascal atau Ada. Deklarasi nama (fungsi, variabel, tipe, konstan) yang dilakukan di luar fungsi disebut deklarasi eksternal. Deklarasi di dalam fungsi disebut deklarasi internal. Variabel dengan deklarasi internal, lokal terhadap blok tempat ia dideklarasi. Nama variabel dengan deklarasi eksternal berlaku global dalam file tempat ia dideklarasi.

Mengacu suatu Nama

Dengan menyebutkan (mengacu) suatu nama maka berarti kita mengacu kepada nilainya. Nama yang diacu harus pernah dideklarasikan sebelumnya. Ini tidak berlaku untuk nama fungsi eksternal

- Fungsi eksternal yang belum dideklarasikan dianggap mempunyai *return value* dan parameter bertipe `int` atau `double` (tergantung pada tipe parameter aktual).
- Jika deklarasi implisit ini tidak sesuai akan timbul kesalahan pada saat kompilasi.
- Sebaiknya, setiap fungsi eksternal yang dipakai dideklarasikan dengan prototipe

Aturan nama

- terdiri dari huruf, angka, dan garis bawah "_" *(under score)*
- jumlah karakter penting dalam nama minimum 31
- huruf besar dan huruf kecil dibedakan
- dimulai dengan huruf
- tidak boleh *reserved word* , untuk C standar (ANSI C):

```
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
```

Aturan akses nama :

- Berdasarkan deklarasinya, dibedakan nama global (deklarasi global) dan nama lokal (deklarasi lokal)
- Nama global dapat diakses oleh semua fungsi dalam file yang sama (supaya nama ini dapat diakses oleh fungsi di file lain, nama ini harus dideklarasikan lagi di file tersebut)
- Nama yang dideklarasi pada suatu fungsi hanya dapat diakses dalam fungsi tersebut
- Jika ada nama yang sama, yang diacu adalah nama lokal.

***Name space* dalam C:**

Name space (ruang nama) adalah kategori nama yang dapat dipunyai oleh suatu nama yang dideklarasikan. Sebuah nama yang sama dapat dipakai untuk keperluan yang berbeda, asalkan *name space*-nya berbeda (walaupun pada kuliah ini tidak dianjurkan memakai nama yang sama untuk keperluan yang berbeda.
Ada lima *name space* dalam bahasa C :

- nama makro preprosesor, ini dipakai pada saat dilakukan preproses program sumber; setelah preproses selesai, nama ini tidak dikenal lagi;
- nama label tujuan perintah goto;
- nama tag struktur/union (nama yang mengikuti kata kunci struct atau union);
- nama anggota struktur/union; masing-masing struktur/ union mempunyai *name space* sendiri, nama yang sama dapat muncul sebagai anggota struktur/ union yang berbeda;
- nama yang tidak termasuk salah satu di atas, termasuk dalam *name space* untuk variabel, fungsi, tipe, dan enumerasi.

LITERAL KONSTANTA

Ada beberapa macam literal konstanta (penulisan nilai konstanta langsung di dalam teks program sesuai dengan type yang mewakili konstanta tsb) yaitu : integer, karakter, floating, enumerasi, dan string.

Konstanta Integer

- Konstanta integer terdiri dari deretan angka, boleh mempunyai prefiks dan/atau sufiks.
- Prefiks:
  O angka oktal [0..7], contoh `023 /* adalah nilai 19 dalam bil dasar 10*/`
  Ox, OX angka heksadesimal `[0..9, a..f, A..F],`
  contoh: `0x45 /* nilai 69 bil. dasar 10 */`
- Sufiks:
  `u, U` unsigned
  `l, L` long

Konstanta Karakter

- Konstanta karakter terdiri dari deretan satu/lebih karakter yang diapit petik tunggal, contoh `'r'`.
- Karakter yang tidak kelihatan di layar atau beberapa karakter khusus, tidak dapat dituliskan langsung sehingga harus menggunakan *escape sequence* yaitu penulisan nilai konstanta karakter tsb sesuai dengan tabel berikut (semua escape sequence selalu diawali dengan \)

Esc.Seq.	Nama	Esc.Seq.	Nama
\a	Alert (bell)	\v	Vertical tab
\b	Backspace	\'	Single quotation mark
\f	Form feed	\"	Double quotation mark
\n	Newline	\\	Backslash
\r	Carriage return	\ddd	ASCII character (in octal notation)
\t	Horizontal tab	\xdd	ASCII character (in hex notation)
\?	Question mark		

Konstanta Floating

- Konstanta floating terdiri atas bagian integer, titik desimal, bagian pecahan, dan bagian eksponen yang diawali huruf `'e'` atau `'E'`. Titik desimal atau bagian eksponen dapat tidak ada, namun salah satu harus tetap ada. Konstanta floating boleh mempunyai sufiks.
- Sufiks:
  `f, F  float`
  `l, L  longdouble`
  `(tanpa sufiks, tipe double)`
- Contoh: `3.141592654, 6.02217e23, 3E8`

Konstanta Enumerasi

- Dideklarasikan sebagai enumerator, representasi internalnya adalah konstanta dengan tipe `int` (lihat bagian deklarasi tipe enumerasi).

Konstanta String *(String Literal)*

- Konstanta *string* adalah deretan karakter yang dibatasi dengan petik ganda, contoh "IF-223".
- Bertipe *"array of character"* dengan kelas penyimpanan statik, terinisialisasi dengan karakter yang diberikan (berakhiran '\0'). Efek perubahan pada konstanta string tak terdefinisi.
- Bedakan antara konstanta string (misal `"I"`) dan konstanta karakter (misal `'I'`). Konstanta string `"I"` adalah array dengan dua elemen (karakter I dan '\0'). Konstanta karakter `'I'` mempunyai nilai integer sesuai dengan kode set karakter yang dipakai.

INSTRUKSI BAHASA C

Assignment

```
<nama-informasi> = <ekspresi>;
<nama-informasi> <operator>= <operand>;
```

Catatan:
Assigment ini “tidak boleh” dilakukan bila <nama-informasi> bertipe string (array of character)

Kondisional

```
if (<ekspresi>) <statement>;
if (<ekspresi>) <statement> else <statement>;

switch (<ekspresi>) {
      case <ekspresi-konstan-1>: <statement-1>; [break;]
      case <ekspresi-konstan-2>: <statement-2>; [break;]
      case <ekspresi-konstan-3>: <statement-3>; [break;]
       :
      default:
            <statement>;
}
```

Pengulangan

```
while (<ekspresi>) <statement>;
do <statement> while (<ekspresi>);
for (<ekspresi1>; <ekspresi2>; <ekspresi3>) <statement>;
```

Catatan:

```
for (<ekspresi1>; <ekspresi2>; <ekspresi3>) <statement>;
```

ekivalen dengan

```
<ekspresi1>;
while(<ekspresi2>) {
      <statement>;
      <ekspresi3>;
}
```

Pencabangan

```
goto <label>;
continue;
break;
return <ekspresi>;
```

Catatan:

- `continue` hanya boleh muncul dalam iterasi (`for, do, while`).
  Statement ini menyebabkan kontrol pindah ke loop terdalam yang melingkupinya.
  Contoh:
  ```
  while (...) {
  continue;
  }
  ```
- `break` hanya boleh muncul dalam iterasi atau switch. Statement ini menyebabkan iterasi/switch terdalam dihentikan, dan kontrol pindah ke statement sesudahnya.
- < label> untuk `goto` harus terletak dalam fungsi yang sama. Deklarasi label:
  ```
  <identifier>: <statement>
  ```

DEKLARASI TYPE DAN VARIABEL

Deklarasi Tipe dan Kelas Penyimpanan Variabel

Variabel adalah suatu lokasi di memori dan interpretasinya tergantung pada dua atribut utamanya: **kelas penyimpanan** dan **tipe**. Kelas penyimpanan menentukan *lifetime* variabel, yaitu berlakunya harga variabel tersebut. Tipe menentukan representasi internal data (yang menentukan harga/nilai) yang tersimpan lokasi memori tersebut.

Kelas penyimpanan

Kelas penyimpanan suatu objek ditentukan berdasarkan katakunci tertentu dan konteks deklarasinya. Ada dua kelas penyimpanan: otomatis dan statik.
Objek berkelas otomatis lokal terhadap suatu blok (fungsi) dan dibuang ketika eksekusi blok selesai (lokal terhadap fungsi,masa hidupnya hanya keetika fungsi tsb "aktif"). Jika tidak dispesifikasikan kelas penyimpanannya, deklarasi dalam suatu blok akan menghasilkan variabel otomatis. Objek berkelas statik mungkin dideklarasikan lokal/eksternal, namun dalam kedua kasus ini harga objek tetap walaupun blok/fungsi selesai dieksekusi/ dijalankan lagi *(exit / reentri)*.
Nilai variabel statik bersifat "seumur hidup program utama", namun lingkupnya lokal terhadap blok dimana variabel tsb didefinisikan. Bahasa C menginisialisasi variabel statik dengan 0. Variabel otomatis tidak diinisialisasi (berisi garbage), pemrogram harus menginisialisasinya sendiri.

Lingkup dan Kaitan

- Nama (variabel/fungsi) mempunyai lingkup *(scope),* daerah program tempat nama tersebut dapat dikenal. Lingkup variabel dapat dalam satu blok (fungsi) atau dalam satu unit translasi *(file)*.

- Selain itu, nama juga memiliki kaitan *(linkage)*, yang menentukan apakah nama yang sama di lingkup lain menunjuk ke objek/fungsi yang sama. Objek dengan kaitan internal hanya dikenal lokal dalam suatu file. Objek dengan kaitan eksternal (kata kunci `extern`) global terhadap seluruh program.

Jadi pada saat deklarasi, nama variabel dapat disertai dengan kata kunci sbb (yang dapat dikombinasikan) :

- Kata kunci `auto` menyatakan bahwa objek berkelas otomatis,
- Kata kunci `static` menyatakan bahwa objek berkelas statik : scopenya lokal, nilainya global (tidak diinisialisasi ketika blok dihidupkan pada saat *run time*)
- Kata kunci `extern` hanya mendeklarasikan objek. Objek yang dideklarasikan dengan atribut `extern` harus didefinisikan di *file* (unit translasi) lain (satu kali saja).
- Kata kunci `register` ekivalen dengan `auto`, namun menunjukkan bahwa objek disimpan di register dan sering diakses. Hati-hati dengan pemakaian register sebab terbatas pada register mesin

Berikut ini adalah resume dari S(Scope), K(kelas) dan L(lingkup) suatu nama variabel berdasarkan deklarasinya

Deklarasi	`auto`	`register`	`static`	`extern`	tidak ada
lokal	S: Blok K : otomatis L : -	S: Blok K : otomatis L : -	S: Blok K : statik L : -	S: Blok K : statik L : -	S: Blok K : otomatis L : -
global	tidak boleh auto	tidak boleh	S: File K : statik L : internal	S: File K : statik L : eksternal	S: File K : statik L : eksternal

Keterangan:

kolom : kata kunci yang dipakai sebagai specifier saat deklarasi.
baris : konteks deklarasi (lokal: dalam blok, global di luar blok).
S: *Scope*
K: Kelas Penyimpanan
L: *Linkage*

Suatu objek dapat memiliki type qualifier. Type qualifier yang ada:

- `const` : mendeklarasikan bahwa nilai objek tak akan berubah. Contoh: parameter fungsi yang tidak dikehendaki untuk diubah nilainya (parameter input) diberi type qualifier const. Lihat contoh pada bagian fungsi.
- `volatile` : mendeklarasikan sifat yang berkenaan dengan optimisasi (tergantung implementasi).

Pada saat deklarasi variabel, juga dapat juga sekaligus dilakukan inisialisasi.

Format deklarasi tipe variabel:

```
[<type qualifier>] [<kelas penyimpanan>][<tipe>]<NamaVar1>,<NamaVar2>, ...;
```

Contoh deklarasi sederhana:

```
/* deklarasi global/eksternal */
static i; /* i lokal terhadap file */
extern j; /* sama dengan extern int j, */
          /* j didefinisikan di file lain */
float r;  /* r dapat diakses dari file lain */

/* deklarasi lokal  */
/* awal sebuah blok */
     { /* Kamus */
       auto int i; /* sama dengan int i */
       register float f;
       static double d;
```

```
 static char* WalrusSaid[] = {"The", "time",
      "has", "come,", "to", "speak", "of", "many",
  "things"};
/* Algoritma : … */

}
```

KONSTANTA BERNAMA

Dalam program yang baik, dihindari kemunculan literal dalam teks. Untuk itu, bahsa biasanya menyediakan fasilitas untuk memberi nama kepada suatu nilai literal.
Dalam bahasa C, konstanta bernama dapat dibuat dengan dua cara, yaitu dengan memakai deklarasi `const`, atau dengan memanfaatkan fasilitas makro.
Format deklarasi konstanta bernama adalah sbb:

```
const [<tipe>] <nama> = <harga>;
```

Contoh:

```
const double pi = 3.141592654;
const float  rad = 57.29577951;
const umur = 40; /* sama dengan const int umur = 40 */
```

Definisi konstanta dengan kata kunci `const` baru ada dalam ANSI C. C "lama" (K&R) memakai preprosesor untuk mendefinisikan konstanta.

```
#define pi 3.141592654
#define rad 57.29577951
#define umur 40
```

Perhatikan bahwa preprosesor bekerja dengan melakukan substitusi makro (teks "umur" dalam diganti teks "40", dst.). Teks dalam string tidak disubstitusi.

TIPE DASAR

Tipe Dasar dalam bahasa C dapat dilihat pada tabel sebagai berikut:

- *Integral* type (type dengan suksesor dan predesesor yang terdefinisi):
  - karakter : `char, unsigned char`
  - integer bertanda : `int, short int, signed int, long int`
  - integer tak bertanda : `unsigned short int, unsigned int, unsigned long int`
- Enumerasi
- *Floating type* : `float, double, long double`

Nilai minimum dan maksimum yang dapat dikandung dalam setiap type dasar harus dilihat pada spesifikasi kompilator.

Catatan: Elemen tipe enumerasi merupakan konstanta bertipe integer, dengan harga mulai dari 0, dengan progresi satu untuk harga berikutnya. Pemrogram dapat menentukan harga ini. Jika ada harga yang telah ditetapkan, maka harga berikutnya jika tidak ditetapkan merupakan progresi dari harga terakhir yang ditetapkan.
Contoh definisi type dasar :

```
int  j; /* hanya deklarasi */
float ff; /* hanya deklarasi */
float rf=0.0;  /* deklarasi dan inisialisasi */
int i=2;       /* deklarasi dan inisialisasi */

enum numeral {nol,two,three,four, five} numero;
     /* nol = 0, two = 1, dst */
enum { Centaurus, Crux, Scorpio, Lupus, Pegasus, Perseus,
       Andromeda=31, Cassiopea } RasiBintang;
      /* Centaurus = 0 .. Perseus = 5, Andromeda = 31,
         Cassiopea = 32 */
```

Contoh pemakaian:

```
j := 99; ff:= 0.5;
i++; rf = rf*5.0; numero = three; RasiBintang = Centaurus;
```

Catatan:

- Objek bertipe `void` tidak dapat didefinisikan, karena ukuran tidak diketahui. Tipe ini dipakai untuk return value fungsi atau untuk mendeklarasikan/mendefinisikan pointer ke objek yang tipenya tidak diketahui.
- C tidak melakukan pemeriksaan saat ***run time*** untuk memastikan bahwa harga yang di-assign ke variabel sesuai dengan range yang ada.

POINTER (*)

Pointer berisi alamat mesin. Pointer menunjuk kepada nama yang diacunya, sehingga informasi yang disimpan pada nama dapat diakses. Pointer dapat menunjuk kepada pointer *(pointer to pointer)*.
Pointer memungkinkan alokasi dinamik, memori baru di-**alokasi** berdasarkan kontrol pemrogram, yaitu hanya jika diperlukan. Jika tidak dibutuhkan lagi, ruang memori yang dialokasi dapat di-**dealokasi** (dikembalikan) ke mesin. Dengan demikian, kontrol pemakaian memori pada saat *run time* berada di tangan program dan berdasarkan atas kondisi memori saat itu. Di samping keuntungannya, pemrogram harus berhati-hati dan cermat dalam melakukan alokasi/dealokasi. Jika tidak, maka suatu saat memori tidak cukup, atau address mengacu ke suatu yang tidak terdefinisi dan dapat menyebabkan komputer *"hang"*
Dalam bahasa C, walaupun nilai variabel bertype pointer adalah alamat mesin, nilai tersebut dapat dimanipulasi seperti halnya nilai numerik.

Pointer sederhana

Dekalarasi variabel bertype pointer
Format:

```
<type> *<name> ;
```

Contoh pendefinisian nama bertype pointer ke ….

```
int *i; /* pointer ke integer */
float *x; /* pointer ke float */
char *cc; /* pointer ke karakter */
FILE *FileKu; /* Handle sebuah file */
int *PointerKeInteger;
char **argv; /* pointer ke pointer ke karakter */
int (*tabel)[13]; /* pointer ke array dengan 13
                          elemen integer */
int *tabel[13]; /* pointer ke array dengan 13 elemen
                       yang bertipe pointer ke integer */
void* p; /* pointer ke objek yang tidak diketahui
              tipenya */
```

Contoh pendefinisian, kemudian alokasi dinamik, dan penentuan nilai yang ditunjuk

```
int *iptr; /* deklarasi dalam kamus */
/* Algoritma */
iptr=(int*) malloc(sizeof(int)); /* alokasi tempat
                                    untuk satu integer */
*iptr=999;  /* nilai yg ditunjuk iptr diisi */
            /* dapat dilakukan setelah alokasi ! */
```

Contoh pendefinisian dan inisialisasi nilai variabel bertype pointer

```
int *iptr= 100; /* deklarasi dan inisialisasi dalam kamus */
```

Perhatikanlah bahwa dengan instruksi tersebut, artinya nilai iptr adalah 100 (alamat absolut pada mesin. Di sini kita memanipulasi nilai iptr dan bukan nilai yang ditunjuk oleh iptr, karena itu tidak ada alokasi sebelumnya.

Pointer ke fungsi

Pointer ke fungsi dapat didefinisikan, disimpan di variabel/array, dan dipakai sebagai *return value*/ parameter fungsi. Lihat contoh program kecil.
Contoh penggunaan pointer ke fungsi sebagai parameter dapa dilihat di prototipe fungsi standar bsearch dan qsort.
Contoh deklarasi pointer ke fungsi:

```
int (*comp)(int, int);
 /* comp: pointer ke fungsi dengan dua argumen
          bertipe integer dan mempunyai return value
          integer */
char (*(*x[3])())
      /* x  array 3  elemen, masing-masing elemen
         adalah  pointer ke  fungsi  dengan  return
         value char* */

char (*(*y[3])())[5]
      /* y array 3 elemen, masing-masing elemen
          adalah  pointer  ke fungsi dengan return
         value pointer ke array karakter 5 elemen*/
```

Contoh pemanggilan:

```
if ((*comp)(a, b)) aksi();
printf("%s\n", x[0]);
```

Pointer sebagai sarana passing paramater output

Dalam kuliah ini, pointer sebagai type data (alamat mesin) dan pointer yang dimanfaatkan sebagai sarana *passing parameter*, dibedakan artinya (akan dibahas pada saat pembahasan topik yang terkait). Untuk setiap variabel bertype pointer, harus diperhatikan : deklarasi, alokasi dan inisialisasi nilai. Bagian ini akan dibahas lebih lanjut dalam prosedur dan fungsi

Resume pointer

Perhatikan dan pahamilah deklarasi type sebagai berikut

Deklarasi	Type
`int`	*integer*
`int *`	*pointer to integer*
`int *[3]`	*array of 3 pointer to integer*
`int(*)[]`	*pointer to an unspecified number of integer*
`int *()`	*function of unspecified parameters returning pointer to integer*
`int (*[]) (void)`	*array, of unspecified types, of pointers to functions with no parameters each returning an integer*

ARRAY

Array adalah sekumpulan elemen bertype sama, yang mempunyai sebuah nama (nama array) dan setiap elemen dapat diacu melalui indeksnya. Array dengan satu indeks disebut array berdimensi satu, vektor, larik atau tabel. array dengan dua indeks disebut array dua dimensi atau matriks. Array dapat mempunyai dimensi lebih dari dua. Yang harus diperhatikan adalah :

- nama array (seluruh elemen). Dalam bahasa C nama array mengacu ke elemen yang ke-0
- dimensi array (banyaknya indeks)
- ukuran array, atau batas nilai indeks. Dalam bahasa C, batas minimum nilai indeks selalu 0

Array dapat didefinisikan secara statik atau secara dinamik. Array statik adalah array yang ukurannya ditentukan saat kompilasi. Sedangkan array dinamik adalah array yang ukurannya didefinisikan pada saat run time dengan perintah alokasi memori.
Perhatikan presedensi operator dalam mengakses elemen dengan array.
Urutan akses array adalah *"row column order"*.

Array statik

-Definisi statik:

```
<type> <nama> [<uk-1>][<uk-2>]...[3]
```

- Contoh definisi array statik:

```
float TabX[200];
int MatA[3][3];
```

Indeks dan ukuran array:

<uk-n> menyatakan ukuran array di dimensi ke n
dengan indeks selalu mulai dari 0 hingga <uk-n>-1.

Isi/nilai dari <nama> sama dengan &<nama>[0],
*(<nama>+1) sama dengan <nama>[1], dst.

- Contoh deklarasi, inisialisasi dan cara mengacu:

```
   Deklarasi:                Cara mengacu:
----------------------      ----------------
  float TabX[200];              TabX[3]
  int MatA[3][3];              MatA[0][0]
```

```
- Inisialisasi array statik adalah per baris, contoh:
      int a[3][4] = { {1,7,1,2}, {1,9,3,9}, {1,6,0,1} };
```

Array dinamik

Array dinamik dideklarasi dengan hanya menyebutkan pointer ke elemen yang ke nol. Jadi definisinya adalah dinamik (dengan pointer):
Format deklarasi :

```
<type> *<name>
```

Contoh definisi array dinamik:
Perhatikanlah bahwa pada saat deklarasi seperti pada contoh sebagai berikut,, tidak ada bedanya dengan pointer ke integer. maka sangat disarankan agar deklarasi tabel statik selalu disertai komentar bahwa nama tsb akan menjadi array!

```
float *TabX; /* TabX adalah array dg elemen float */
int *Mata; /*Mata adalah matriks dg elemen integer*/
           /* perhatikanlah definisi ini, */
         /* tidak diketahui ukuran&dimensi matriks*/
         /* dan belum ada bedanya dengan pointer ke integer*/
```

Perhatikan bahwa dengan definisi tersebut, kita belum juga mengetahui ukurannya. Array dinamik harus dialokasi terlebih dahulu supaya ukurannya terdefinisi.

Jadi ada empat tahapan dalam pemakaian array dinamik:

- Deklarasi (pendefinisian nama)
- Alokasi (menentukan ukuran, alokasi memeori dinamik)
- Inisilisasi nilai
- Dealokasi (pengembalian memori dinamik)

Dalam bahasa C, ketiga hal tersebut dapat dilakukan sekaligus (dalam kamus) atau bertahap (dalam kamus dan dengan instruksi).

- Array dinamik (dengan pointer):
  - Contoh definisi dan cara mengacu:

```
   Deklarasi:          Cara mengacu:
----------------------  ----------------
  float *TabX;          *TabX     /* elemen 0 */
                         TabX[0]   /* elemen 0 */
                         *(TabX+1) /* elemen 1 */
                          TabX[1]   /* elemen 1 */
  int *mata;              *MatA     /* elemen 0 */
                           MatA[0]   /* elemen 0 */
```

```
*(MatA+100)  /* elemen 100*/
 MatA[100]   /* elemen 100*/
```

Catatan Penting:

- Perhatikan presedensi operator dalam mengakses elemen array.
- Urutan akses array adalah *"row column order"* (per baris).
- Perhatikan deklarasi, alokasi dan inisialisasi nilai untuk array dinamik lewat contoh berikut
- Contoh Deklarasi array dinamik dan sekaligus alokasi dan inisialisasi nilai

```
int *Tab[ ] = { 1,2,3,4 };
/* Deklarasi Tab */
/* alokasi berukuran 4 */
/* mendefinisikan nilai sbb: */
/*Tab[0]=1, Tab[1]=2, Tab[3]=4,Tab[4]=5 */
```

- Contoh Deklarasi array dinamik, kemudian alokasi dan menentukan isinya dengan assignment

```
int* list;           /* deklarasi dalam Kamus */
/* Algoritma */
list = (int*) malloc(5*sizeof(int));      /* alokasi */
*list = 1;      /* init elemen ke 0 dengan nilai 1 */
*(list+1) = 9;  /* init elemen ke 1 dengan nilai 9 */
*(list+2) = 5;  /* init elemen ke 2 dengan nilai 5 */
list[3] = 2;    /* init elemen ke 3 dengan nilai 2*/
list[4] = 0;    /* init elemen ke 4 dengan nilai 0*/
```

STRING

String adalah **pointer ke array dengan elemen karakter**. Perhatikanlah bahwa string berbeda dengan *array of char*.
Konstanta string dituliskan di antara tanda petik ganda. Contoh :

```
"Ini sebuah string"
```

Representasi internal string selalu diakhiri dengan karakter '\0'. Sedangkan array of char "biasa" tidak diakhiri dengan '\0'

Ada perbedaan mendasar pada dua buah definisi sebagai berikut:

```
char amessage[ ] = "kini saatnya tiba";
char *pmessage = "kini saatnya tiba";
```

amessage adalah sebuah array dengan elemen karakter, dengan ukuran yang cukup untuk menyimpan "kini saatnya tiba" dan '\0'. Setiap karakter anggota `amessage` dapat dimanipulasi/diubah, tetapi selalu menempati storage yang sama. `pmessage` adalah sebuah pointer yang diinisialisasi nilainya dengan konstanta string, maka pointer ini dapat diubah untuk menunjuk ke lokasi memori yang lain. .
Perhatikanlah beberapa versi program dalam bahasa C yang identik efeknya sbb. sebagai studi terhadap string :

```
/* strcpy : copy string t ke s, versi 1: array subscript */
void strcpy (char*s, char *t);
{ int i;
  i= 0;
  while ((s[i]=t[i]) != '\0') i++;
}
```

```
/* strcpy : copy string t ke s, versi 2: versi 1 dengan pointer */
void strcpy (char*s, char *t);
{ int i;
  i= 0;
  while ((*s= *t) != '\0') {s++; t++}
}

/* strcpy : copy string t ke s, versi 3: versi 2 dituliskan secara
lain */
void strcpy (char*s, char *t);
{ int i;
  i= 0;
  while ((*s++ = *t++) != '\0')
}


/* strcpy : copy string t ke s, versi 4: versi 3 dengan memanfaatkan
terminator string */
void strcpy (char*s, char *t);
{ int i;
  i= 0;
  while (*s++= *t++); }
```

Assignment secara langsung pada objek bertipe string tidak boleh dilakukan. Variabel bertipe string sebenarnya berisi pointer ke lokasi memori yang telah dialokasikan oleh C (sesuai dengan definisinya). Jika dilakukan *assignment* pada variabel ini, C tidak melakukan penyalinan isi string, melainkan mengubah nilai pointer yang tersimpan di variabel (berbeda dengan beberapa bahasa pemrograman yang lain). Manipulasi string dalam bahasa C harus dilakukan secara hati-hati!

Contoh:

```
char KalimatBagus[37];

/* contoh yang salah */
KalimatBagus = "Don't talk too much";  /* SALAH !!!! */
/* Setelah ini, KalimatBagus menunjuk ke string sepanjang
   20 karakter (ingat, string harus diakhiri dengan '\0').
   Jika kemudian dilakukan perintah:
strcpy(KalimatBagus, "Considers the lilies of the field...");
    mungkin dapat timbul bencana.
    (paling ringan: Null pointer assignment,
  paling berat: HANG!, ini tergantung daerah yang tertimpa) */
/* contoh yang benar */
strcpy(KalimatBagus, "Don't talk too much");
/* Perintah strcpy
strcpy(KalimatBagus, "Considers the lilies of the field...");
      akan berjalan seperti yang diharapkan */
```

ARRAY MULTI DIMENSI & STRING

Perhatikanlah definisi sebagai berikut

```
int a[10][20];
int *b[10]
```

Maka `a[3][4]` dan `b[3][4]` secara sintaks merupakan akses yang legal untuk mengakses sebuah elemen integer. a adalah array berdimensi dua, dengan 200 lokasi memori masing-masing bertype integer. Kalkulasi 20*row+col dipakai untuk mengacu elemen `a[row][col]`. Untuk b, definisi di atas hanya untuk alokasi 10 elemen dan tidak ada inisialisasi. Inisialisasi harus dilakukan secara eksplisit (pada saat deklarasi atau lewat instruksi). Jika setiap elemen b dialokasi ke 20 elemen integer, maka akan ada 200 elemen dialokasikan dan 10 elemen untuk pointernya. Keuntungan pointer array adalah bahwa ukuran setiap baris dapat mempunyai ukuran yang berbeda, seperti contoh berikut (yang sering digunakan untuk *array of string*)

```
char *name[] = {"tbd", "Jan", "Feb", Mar" };
```

```
yang berbeda alokasi memorinya dibandingkan
   char name[15] = {"tbd", "Jan", "Feb", Mar" };
```

Aritmatika Pointer, *address arithmetic* dalam array

Bahasa C sangat konsisten terhadap aritmatika, juga aritmatika address. Maka jika p adalah sebuah pointer dan mengacu ke elemen suatu array, kita dapat melakukan p++. maka p akan mengacu ke elemen berikutnya. Demikian pula `p--` akan mengacu ke elemen sebelumnya. `p+=i` akan mengacu ke i elemen berikutnya dalam array.

STRUKTUR KOMPOSISI (RECORD)

Struktur merepresentasi suatu type komposisi dalam konsep algoritmik, yaitu sebuah type yang terdiri dari komponen-komponen bertype tertentu (yang tentunya boleh rekursif, yaitu bertype sepreti definisi type tersebut)

Deklarasi/definisi:

```
struct [<nama struktur>] {
    /* definisi komponen struktur */
          <type-1> <anggota-1>;
          <type-2> <anggota-2>;
          :
          <type-n> <anggota-n>;
          } [<nama variabel struktur>];
```

Ada dua cara untuk mengakses elemen struktur :

- Pengaksesan elemen dengan operator .

```
<nama variabel struktur>.<anggota>
```

- Pengaksesan lewat pointer:

```
<pointer ke struktur>-><anggota>
(* <pointer ke struktur>).<anggota>
```

Contoh pendefinisian dan akses struktur komposisi

Berikut ini diberikan tiga buah contoh potongan deklarasi program yang "sama" efeknya, namun "berbeda" pada prinsipnya. Perhatikan baik-baik, mana nama type, nama variabel, nama "tag".
Contoh1: definisi nama variabel dengan struktur komposisi yang terdiri dari nama, kelas, beratbadan dan sekaligus mendefinisikan variabel Rocky, MikeTyson, Ali sebagai struktur yang mempunyai elemen nama, kelas, beratbadan :

```
/* Berikut ini adalah definisi variabel ber struktur komposisi */
          struct {
                char nama[20];
                char kelas[10];
                float beratbadan;
          } Rocky, MikeTyson, Ali;
```

Contoh2 : Pada contoh berikut, tiga buah field yang membentuk struktur dan terdiri dari nama, kelas dan berat badan diberi "nama" Petinju. Dalam kasus ini, tidak didefinisikan type baru .

```
          /*  definisi struktur Petinju */
           struct Petinju {
                 char nama[20];
                 char kelas[10];
                 float beratbadan;
           };
          /* deklarasi varibel dengan struktur Petinju */
           struct Petinju Rocky, MikeTyson, Ali;
```

Contoh Cara akses: Jika Rocky bertipe struct Petinju, maka:

```
            /* ingat, assignment ke string tidak dibolehkan */
               strcpy(Rocky.nama, "Rocky Balboa");
               strcpy(Rocky.kelas, "Berat");
               Rocky.beratbadan = 98.0;
```

Seringkali, untuk struktur yang rekursif, salah satu teknik adalah dengan memanfaatkan *tag* (cara lain lihat di contoh program kecil. Misalnya contoh untuk list linier). Contoh struktur rekursif (*node* pohon biner): pada contoh ini `tnode` adalah “tag”, yaitu "nama" yang diberikan untuk menggantikan sekumpulan *field* di antara `{` dan `}`

```
struct tnode {
            char* nama; /* nama node */
            struct tnode* kiri; /* anak kiri */
            struct tnode* kanan; /* anak kanan */
      } node;
struct tnode *left; /* variabel left :pointer ke sebuah tnode*/
```

UNION

Union memungkinkan sebuah struktur komposisi (record) mempunyai alternatif type elemennya
Deklarasi/definisi:

```
union [<nama union>] {
      /* definisi komponen struktur */
      <type-1> <anggota-11>, <anggota-12>, ...;
      <type-2> <anggota-21>, <anggota-22>, ...;
            :
      <type-n> <anggota-n1>, <anggota-n2>, ...;
} [<nama variabel union>];
```

Contoh:

```
/* mendefinisikan union kunci untuk mengakses */
/* kode ASCII dan scan code dari keyboard (Turbo C)*/
      union kunci {
            int i;     /* penampung ret. val. bioskey */
            char ch[2];/* ch[0] = scan code, ch[1] =
                          kode ASCII */
      };
```

Catatan: Elemen-elemen union memakai lokasi yang sama di memori. Pada deklarasi di atas, <anggota-1x>, <anggota-2x>, ..., <anggota-nx> memakai lokasi memori yang sama.
Pengaksesan elemen:

```
<nama variabel union>.<anggota>
```

Pengaksesan lewat pointer:

```
<pointer ke union>  ->  <anggota>
(* <pointer ke union>).<anggota>
```

Perhatikanlah bahwa pada pengaksesan lewat pointer dengan cara kedua, tanda kurung harus dituliskan akibat dari presedensi operator "." pada pengaksesan struktur. Lupa menuliskan tanda kurung akan berakibat fatal!
Contoh yang lengkap dari penggunaan union dapat dilihat pada program kecil, di mana sebuah struktur Gambar dapat terdiri dari beberapa macam bentuk (misalnya garis, segi empat, dsb)

BIT FIELD

Dalam pemrograman tingkat rendah, seringkali dibutuhkan akses sebuah struktur informasi berupa bit. Bitfield merepresentasi hal ini. Pengaksesan elemen sama dengan pengaksesan elemen struktur.
Bit field berlaku seperti integer (dengan nilai terbatas), dan dapat dipakai dalam ekspresi aritmatika.
Digunakan untuk menyimpan beberapa objek dalam 1 *byte/word*, jika :

- tempat penyimpanan sangat terbatas
- akan dilakukan manipulasi pada register perangkat keras

Contoh: return value status perangkat keras dari bios call/ DOS call pada IBM PC

Deklarasi **bit field** pada struktur:

```
struct [<nama struktur>] {
            /* definisi komponen struktur */
            <type-1> <anggota-1>: <jumlah-bit-1>;
            <type-2> <anggota-2>: <jumlah-bit-2>;
                    :
           <type-n> <anggota-n>: <jumlah-bit-n>;
       } [<nama variabel struktur>];
```

Harga total jumlah bit maksimum tergantung implementasi (misalnya, untuk Turbo C jumlah bit total maksimum 16).

- Contoh:

```
struct bagi_dua {
      unsigned int BagianKu: 4;
      unsigned int BagianMu: 4;
} CounterKita;
/* CounterKita.BagianKu mempunyai range 0..63,
   CounterKita.BagianMu mempunyai range 0..63,
   ukuran CounterKita 1 byte (8 bit) */
```

Contoh shift status (return value int 16h fungsi 2):

```
struct keyboard_flags {
      unsigned int rshift : 1; /* bit 0 (LSB) */
      unsigned int lshift : 1;
      unsigned int ctrl   : 1;
      unsigned int alt    : 1;
      unsigned int scroll : 1;
      unsigned int num    : 1;
      unsigned int caps   : 1;
      unsigned int insert : 1; /* bit 7 (MSB) */
} keystatus;
```

Contoh akses:

```
if (keystatus.ctrl) {
      /* tombol Ctrl ditekan, lakukan .... */
      keystatus.ctrl = 0;
      :
}
```

TYPEDEF : *USER DEFINED TYPE*

Dalam bahasa C, kita dapat mendefinisikan "nama type", yaitu dengan kata typedef. Typedef seringkali dipakai (dan sangat disarankan untuk dipakai) untuk *readability*, yaitu untuk memberikan kemudahan baca bagi struktur data yang rumit (misalnya pointer ke pointer atau struktur yang "membingungkan").

Format:

```
typedef <tipe diturunkan> <nama tipe>;
```

Contoh deklarasi dan efeknya :

```
typedef int bulat;
typedef struct {double r, theta;} Kompleks;
```

maka

```
bulat i; /*  deklarasi ini absah. i bertype int */
Kompleks z, *zp; /*z  adalah struktur */
                 /* yg merepresentasi bil.kompleks*/
                 /* zp adalah pointer ke bilangan kompleks */
```

Contoh berikut "ekivalen" dengan contoh 1) dan 2) pada pendefinisian petinju pada bagian struktur komposisi, namun memakai definisi tipe baru tPetinju . tPetinju adalah nama type yang didefinisikan, dan terdiri dai nama, kelas dan berat badan. Pada contoh ini kita mendefinisikan type "baru" yang belum didefinisikan dalam bahasa C

```
typedef struct {
      char nama[20];
      char kelas[10];
      float beratbadan;
} tPetinju;
tPetinju Rocky, MikeTyson, Ali;
```

- Contoh pengaksesan dengan pointer:

Jika pRocky = *tPetinju , maka:

```
pRocky = (*tPetinju) malloc(sizeof(tPetinju));
strcpy(pRocky->nama, "Rocky Balboa");
strcpy(pRocky->kelas, "Berat");
pRocky->beratbadan = 98.0;
```

Mendefinisikan tabel dengan ukuran efektifnya secara eksplisit.

```
typedef struct {int Tab[10];  /* tabelnya */
   int  Neff; /* ukurannya*/  } TabInt;
```

```
/* mendefinisikan matriks 3 x 3 */
            typedef int matriks3x3[3][3];
            matriks3x3 m1, m2, m3;

              :
            m1[0][0] = 1;



/* definisi antrian  */
          /* definisi antrian, yang direpresentasikan secara
               kontigu dengan tabel.
               Address didefinisikan sebagai tipe int,
               NMax konstanta berisi jumlah maksimum elemen
                  antrian,
               InfoType tipe informasi elemen antrian */

                typedef struct {
                        Address Head;
                        Address Tail;
                        InfoType TabElmt[NMax+1];
                      } Queue;
             /* perhatikan bahwa TabElmt mempunyai indeks
                [0..NMax]. Karena 0 dipakai sebagai tanda
                Queue kosong, maka TabElmt[0]
                tidak dipakai */

          /* definisi variabel */
           Queue Antrian1;
```

Perhatikan bahwa dengan mendefinisikan nama baru, maka teks program dapat lebih mudah dibaca seperti pada contoh elemen list sebagai berikut :

```
typedef struct tElmt;   /* forward declaration! */
typedef *tElmt address; /* address adalah sebuah TYPE, */
                        /* pointer ke tElmt */
typedef struct tElmt {int Info; /* informasi sebuah elemen list*/
                      address Next; /* alamat suksesor */
        } ELmtList;
address P; /* variabel P: alamat sebuah elemen */
address First; /* alamat elemen pertama list linier berkait */
```

Catatan :
Dua buah type dikatakan ekivalen jika mempunyai type specifier yang sama (dengan memperhatikan bahwa sebuah type specifier mungkin mengandung implikasi yang lain seperti long yang mencakup long int. Structures, union, enumeration dengan tag yang berlainan dianggap lain, dan structure, union, enumeration tanpa tag menspesifikasikan type yang unik.
Dua buah type dikatakan sama jika setelah ekspansi typedef dan penghapusan identifier parameter ekivalen dengan type specifier list. Ukuran array dan parameter fungsi penting artinya.

KONVERSI TIPE

- Konversi tipe yang dituliskan di bagian ini, adalah konversi tipe informal yang sederhana (aturan konversi yang lengkap dan detil dapat dilihat di bagian 6 Appendix A [Kernighan-88]).
- Secara general, konversi tipe otomatis dilakukan untuk mengubah operand yang lebih "sempit" ke operand yang lebih "lebar" tanpa kehilangan informasi. Ekspresi yang mungkin akan menimbulkan kehilangan informasi (seperti menyimpan angka floating point ke variabel bertipe integer) tidak dilarang (tetapi mungkin akan menimbulkan peringatan).

- Tipe karakter dianggap sama dengan integer (dengan range kecil), sehingga data bertipe karakter dapat dipakai di ekspresi aritmatika. Contoh:

```
/* lower: fungsi untuk mengubah karakter c ke huruf
   kecil */
int lower(int c) {
      if (c >= 'A' && c <= 'Z')
            return(c + 'a' - 'A');
      else
            return(c);
}
```

- Definisi C menjamin bahwa semua karakter tercetak tidak akan negatif. Namun, jika objek bertipe karakter akan dipakai untuk menyimpan data non-karakter, spesifikasikan tanda objek secara eksplisit (signed char atau unsigned char).
- Konversi implisit ekspresi aritmatika tanpa unsigned operand:
  - Jika salah satu operand long double, konversikan yang lain ke long double.
  - Jika tidak, jika salah satu operand double, konversikan yang lain ke double.
  - Jika tidak, jika salah satu operand float, konversikan yang lain ke float.
  - Jika tidak, konversikan char dan short ke int.
  - Jika salah satu operand long, konversikan yang lain ke long.
- Perhatikan bahwa tipe `float` dalam ekspresi tidak dikonversi secara otomatis ke tipe `double`.
- Aturan konversi yang melibatkan operand tak bertanda lebih rumit dan tergantung pada mesin. Contoh: jika int memakai 1 word dan long int memakai 2 word, maka `-1L < 1U` (karena 1U dikonversikan ke signed long). Sedangkan, `-1L > 1UL` (karena `-1L` dipromosikan ke `unsigned long`, dan menjadi bilangan positif besar).
- Konversi terjadi pada *assignment*, harga di sisi kanan diubah ke tipe `lvalue` di ruas kiri.
- "Pemaksaan" konversi tipe dapat dilakukan dengan operator uner **cast.** Perhatikanlah bahwa casting harus selalu dilakukan ketika kita melakukan **alokasi memori,** supaya jelas pointer menunjuk ke type apa. Casting dilakukan dengan menuliskannya dengan format sbb:

```
(<type>) <ekspresi>
```

  Contoh:

```
(double) 24 /* sama dengan 24.0 */
```

- Konversi pada parameter aktual fungsi tanpa prototipe: char dan short menjadi int, float menjadi double.
- Untuk fungsi dengan prototipe, parameter aktual di-cast sehingga tipenya sesuai dengan prototipe.
- Walaupun bahasa C memberikan banyak kebebasan terhadap type, pada kuliah ini diterapkan aturan **ketat terhadap type**. Sebuah ekspresi harus mempunyai komponen (operan) dengan type yang sama, dan jika disimpan dalam ruas kiri assignment juga harus bertype sama.

Contoh:

ekspresi `2*pi*r` dengan pi dan r adalah float, maka disarankan menulis `float(2)*pi*r`

Contoh representasi dan manipulasi type JAM :

```
typedef struct {int HH; /* jam */
                int MM; /* menit */
                int SS; /* detik*/ } JAM;
typedef long int Detik;
/* Konversi JAM menjadi Detik harus ditulis sbb:*/
JAM J;
Detik Dx;
/* Jika J sudah terdefinisi */
Dx = (long int) J.HH * (long int) (3600) +
       (long int) J.MM * (long int) (60) +
       (long int) J.SS ;
```

OPERATOR

Operator Numerik

Operator	Arti
*	kali
/	bagi
%	modulo
-	tambah atau plus uner
++	incremenet (plus 1)
©	decrement (minus 1)

Catatan:
Increment dan *decrement* dapat dilakukan sebelum ekspresi dievaluasi *(pre-increment)* atau setelah ekspresi dievaluasi *(post-decrement)*.
Contoh *pre-increment*:

```
x = --i + 1; /* ekivalen dengan --i; x = i + 1 */
```

Contoh *post-increment:*

```
x = i++ + 1; /* ekivalen dengan x = i + 1; ++i */
```

Operator Bit

Operator	Arti
<<	shift left
>>	shift right
&	and
|	or
^	xor
~	not

Catatan:
Operator ini bekerja bit per bit (jadi tergantung kepada representasi bilangan), bedakan dengan operator logika. Perhatikanlah contoh sebagai berikut

```
int a,b; /* ekspresi a&b akan berbeda hasilnya dengan a&&b */
```

Lihat contoh pemakaian pada program kecil dan perhatikan hasilnya

Operator Relasional

Operator	Arti
>	lebih besar
>=	lebih besar atau sama dengan
<	lebih kecil
≤	lebih kecil atau sama dengan
==	sama dengan
!=	tidak sama dengan

Catatan:
Perhatikan bahwa test kesamaan dilakukan dengan operator "==" bukan "=". Ini seringkali menimbulkan kesalahan program bagi yang terbiasa dengan bahasa pemrograman lain (misalnya Pascal). Operator = dalam bahasa C berarti *assignment* (lihat bagian operator assignment)
Perhatikan contoh berikut:
Kasus 1: Cek kesamaan nilai x dan y, kemudian increment x

```
if (x==y) x++;
```

Untuk kasus ini, x++ akan dilakukan jika x sama dengan y.
Kasus 2: *Assign* isi y ke x, kemudian *increment* nilai x

```
if (x=y) x++;
```

Di sini, x++ akan dilakukan jika nilai y tidak sama dengan nol (nilai ekspresi x=y sama dengan nilai y, dan true dalam bahasa C adalah jika tidak sama dengan 0).

Demi kemudahan membaca dan pengecekan kebenaran program, sebaiknya kondisi di atas diubah menjadi:

```
x=y;
if (x!=0) x++;
```

atau

```
x=y;
if (x) x++;
```

atau

```
 x=y;
if (y!=0) x++;
```

atau

```
  x=y;
if (y) x++;
```

Operator Logika

Operator	Arti
&&	and
||	or
!	not

Catatan:

- Operator logika adalah operator terhadap nilai boolean (true, false)
- C tidak mempunyai tipe data Boolean. Dalam pengetesan kondisi untuk `if-then`, `while, do-while,` dan `for`, nilai bukan nol dianggap "true" dan nilai nol dianggap "false".
- Emulasi data boolean dapat dilakukan dengan beberapa cara:
  - mendefinisikan nilai true dan false lewat `#define`,

```
#define true 1
#define false 0
#define boolean unsigned char
```

  - menggunakan enumerasi,

```
enum boolean {false, true}; /* perhatikan urutan,
      false harus pertama, karena false = 0 */
```

  atau

```
enum boolean {true = 1, false = 0};
```

- Cara memakai:

```
boolean x;
x = true;
while (x) {
      :
      dst.
      :
}
```

  atau

```
  if (!x) {….} else }
```

- Sebaiknya dibiasakan mempunyai sebuah ***header file*** untuk keseragaman type boolean yang dipakai (lihat contoh pendefinisian Boolean pada Program kecil)

Operator Alamat

Operator	Arti
&	address of (menghasilkan pointer)
*	indirection operator (mengakses isi memori yang ditunjuk pointer

Catatan:
Selain operasi address of atau indirection, pointer dapat dioperasikan secara numerik (dengan +, -, ++, --). Perhatikan bahwa hasil operasi numerik pada pointer tergantung tipe objek yang ditunjuknya. Contoh:

```
int *i=0;  /* pointer ke integer */
char *c=0; /* pointer ke karakter */
int **ii=0;/* pointer ke pointer ke integer */

i++; c++; ii++;
/* jika integer memerlukan 2 byte, karakter 1 byte,
    dan pointer 4 byte maka:
    i = 2, c = 1, ii = 4 */
```

Operator Assignment dan Compound Assignment

Catatan:
Secara umum, operator *compound assignment* mempunyai format:

```
x <op>= y
atau
x = x <op> y
```

Compound assignment hanya dapat dilakukan untuk operator:
*, /, %, +, -, <<, >>, &, ^, |.

Untuk tabel yang diberikan berikut ini, x dan y bertipe sembarang, dan diasumsikan bahwa tipe y dapat dikonversikan ke x.

Ekspresi	Arti
x = y	assignment nilai y ke x (x ← y)
x *= y	x ← x * y
x /= y	x ← x / y
x %= y	x ← x % y
x += y	x ← x + y
x -= y	x ← x - y
x <<= y	x ← x << y
x >>= y	x ← x >> y
x &= y	x ← x & y
x ^= y	x ← x ^ y
x |= y	x ← x | y

Operator Tipe

Operator	Arti
sizeof	ukuran dalam byte
(<type>)	casting tipe operand menjadi <type>

Catatan: sizeof biasanya dipakai untuk mengetahui ukuran alokasi memori, contoh:

```
struct Petinju *PetinjuKelasBerat;
/* alokasi array dinamik dengan 10 elemen */
PetinjuKelasBerat = (struct Petinju *)
                    malloc(10*sizeof(struct Petinju));
```

Sedapat mungkin -- terutama untuk pointer -- gunakan konversi tipe otomatis C (agar program lebih portabel), hindari penggunaan operator cast.

Operator Kondisional

Operator	Arti
(k)?t:f	Jika kondisi k benar (berharga != 0), harga ekspresi adalah t, jika tidak harga ekspresi adalah f

Catatan: Operator ini biasanya dipakai untuk "menghemat" pemakaian kalimat if-then dan/atau pemanggilan fungsi, dengan maksud agar dihasilkan kode yang lebih kompak dan efisien.
Contoh:

```
/* Untuk menggantikan fungsi lower */
            #define lower(c) ((c >= 'A' && c <= 'Z') ? \
                                    (c-'A'+'a') : (c) )
```

Operator Ekspresi Postfix

Operator	Arti
()	pemanggilan fungsi
[]	Elemen array
->	Pointer ke elemen struktur. Operator <pointer> -> <elemen> ekivalen dengan sbb: * <pointer>.<elemen> Lihat contoh definisi struktur Point pada Contoh program kecil
.	Elemen struktur atau union

Operator Koma

Ekspresi	Arti
<exp1>, <exp2>	Evaluasi <exp1>, lalu <exp2>. Tipe dan harga hasil ekspresi adalah <exp2>.

Contoh:

```
f(a, (t=3, t+2), c); /* sama dengan t=3;f(a, t+2, c) */
```

Presedensi Operator

Evaluasi suatu ekspresi dilakukan berdasarkan prioritas operator. Jadi prioritas operator menentukan urut-urutan pelaksanaan evaluasi suatu ekspresi. Dalam bahasa C, urut-urutan evaluasi ekspresi selain ditentukan oleh prioritas, juga ditentukan oleh asosiativitas (“arah”) evaluasi dilakukan yaitu dari kiri ke kanan atau kanan ke kiri. Kombinasi prioritas dan asosiativitas menentukan presedensi operator.

Contoh asosiativitas dari kanan ke kiri:

```
a = b = c; /* yang dikerjakan: a = (b = c) */
```

Contoh asosiativitas dari kiri ke kanan:

```
a = b + c + d; /* yang dikerjakan: a = (b + c) + d */
```

Jika kita tidak cermat dalam memahami presedensi, maka dapat menyebabkan “kesalahan” hasil evaluasi karena hasil tidak memberikan seperti yang diharapkan. Karena itu, sangat disarankan pemakaian evaluasi dengan tanda kurung lengkap.

Tabel presedensi operator dalam bahasa C
(paling atas prioritas paling tinggi, dan operator uner + dan - mempunyai prioritas lebih tinggi daripada sebagai operator biner)

Operator	Asosiativitas
`()  []  ->`	kiri ke kanan
`!  ~  ++  --  +  -  *  &  (<type>)  sizeof`	kanan ke kiri
`*  /  %`	kiri ke kanan
`+  -`	kiri ke kanan
`<<  >>`	kiri ke kanan
`<  <=  >  >=`	kiri ke kanan
`==  !=`	kiri ke kanan
`&`	kiri ke kanan
`^`	kiri ke kanan
`&&`	kiri ke kanan
`||`	kiri ke kanan
`?:`	kanan ke kiri
`|=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=`	kanan ke kiri

PREPROSESOR BAHASA C

Bahasa C menyediakan fasilitas preprosesor (pemrosesan program sumber sebelum kompilasi dilakukan). Dua kata kunci yang seringkali digunakan (dan sudah digunakan pada contoh-contoh sebelumnya) adalah `#include` dan `#define`.

File Inclusion

Semua baris dengan bentuk

```
#include "namafile"
```

atau

```
#include <namafile>
```

akan disubstitusi dengan isi dari file yang bernama namafile (perhatikan letak direktori)
Pada bentuk pertama, pencarian dilakukan terhadap file berupa source code yang didefinisikan pemrogram. Pada bentuk kedua, pencarian isi file dilakukan berdasarkan implementasi

File inclusion sering dimanfaatkan untuk memecah sebuah program besar dalam beberapa file (lihat bab yang bersangkutan).
Aturan dalam kuliah ini:

- File yang boleh di-include hanyalah file dengan ekstensi .h
- File dengan ekstensi .h tidak boleh mengandung variabel

Hal ini untuk mempermudah kompilasi terpisah, dan menghindari nama variabel (alokasi memori) terjadi dua kali.

Macro Substitution

Definisi makro dituliskan dengan format

```
    #define <nama> <teks_pengganti>
```

Semua <nama> dalam teks akan disubstitusi dengan <teks_pengganti>.
Hati-hati dengan format ini, sebab substitusi akan dilakukan secara "harfiah".
Perhatikan contoh berikut

```
 #define pi 3.1415 /* konstanta PI */
```

akan menimbulkan keanehan dalam program sebab semua kata "pi" dalam program disubstitusi menjadi "`3.1415 /* konstanta PI */`" yang mungkin akan menimbulkan kesalahan kompilasi jika dalam teks program sumber yang digantikan adalah

```
   luas = pi * r*r
```

sebaliknya, tidak akan timbul kesalahan jika teks program adalah

```
    konstpi= pi;
```

Semua nama dalam teks program sumber dapat disubstitusi dengan #define, misalnya

```
  #define forever for (;;)    /* loop terus menerus */
```

Substitusi dengan makro juga dapat mempunyai argumen

```
   #define max(a,b) ((a)>(b) ? (a) : (b))
   #define Info(P) (P)->Info
```

Pada contoh tersebut, semua parameter formal a dan b pada contoh pertama, dan P pada contoh kedua akan disubstitusi. Jadi jika teks program mengandung

```
  x = max(p+q), (r+s);
```

akan disubstitusi menjadi

```
 x = ((p+q)> (r+s) (p+q) : (r+s));
```

dan akan menimbulkan kesalahan misalnya pada teks

```
   x = max(i++),j++);
```

Conditional Inclusion

Substitusi teks dengan `#include` dapat mengandung kondisi dengan bentuk umum

```
  if-line text elif-part else-part #endif

dengan
if-line adalah :
  #if <ekspresi>
  #ifdef <identifier>
  #ifndef <identifier>
elif-part
  elif-line teks
  elif-part

elif-part
  #elif ekspresi

else-part
  #else
```

```
Contoh :
#if !defined(HDR)
#define HDR

/* isi hdr.h di-include di sini */

#end if
```

Contoh: untuk menjamin sebuah file hdr.h hanya pernah di-include satu kali, maka dituliskan

```
#if !defined HDR_H
#define HDR_H

/* isi hdr.h di-include di sini */

#endif
```

atau

```
#if ndef HDR_H
#define HDR_H

/* isi hdr.h di-include di sini */

#endif
```

Conditional inclusion seperti contoh di atas menjadi standard di kuliah (lihat contoh **boolean.h** pada program kecil)

FUNGSI DAN PROSEDUR

Bahasa C tidak mempunyai bentuk khusus untuk prosedur. Prosedur didefinisikan sebagai fungsi yang tidak mempunyai return value (return value bertipe void). Contoh:

```
void tukar(int* a, int* b) {
/* menukar isi *a dan *b */
      int tmp=*a;
      *a=*b;
      *b=tmp;
}
```

Fungsi yang dipanggil sebaiknya dideklarasikan terlebih dahulu (dengan menggunakan prototipe). Kompilasi dan link program yang terdiri dari beberapa file tergantung pada implementasi kompilator (untuk Turbo C: dengan file project, untuk Microsoft C dan Sun-OS C: dengan *makefile*). Standar pada kuliah ini adalah bahwa semua prosedur dan fungsi harus dideklarasi prototypenya.

Deklarasi Fungsi dan prosedur

Deklarasi fungsi dilakukan dengan menggunakan **prototipe fungsi**, contoh:

```
int strcmp(const char* str1, const char* str2); /* fungsi */
void rewind(FILE* stream);                      /* prosedur */
```

Dalam bahasa C, fungsi dapat mengembalikan type bentukan.

Nama-nama yang berasal dari pustaka telah ada deklarasinya dalam file include yang sesuai. Contoh: deklarasi fungsi-fungsi dan tipe untuk melakukan input/output ada di file include stdio.h. Agar deklarasi ini dikenali dalam program, file ini "dimasukkan" ke awal program dengan menambahkan direksi preprosesor:

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

Struktur sebuah program C dengan fungsi dan prosedur:

```
#include <header-files>
   :
/*direksi preprosesor lain*/
/*******************************************/
/* daftar nama eksternal yang dipakai program */
/* Kamus Umum: variabel global yang harus dikenal main dan semua */
```

```
/* fungsi dan prosedur, jika ada  */

/*********************************************/
/*prototipe fungsi yang dipakai*/
<tipe-f1> fungsi1(<tipe-pf1>, <tipe-pf2>, ...);
/* prototype prosedur yang dipakai */
void prosedur1(<tipe-pp1>, <tipe-pp2>, ...);

/*********** MAIN PROGRAM **************/

<tipe-m> main([int argc, char** argv[, char** envp]]){
        /* keterangan program */
        /* kamus lokal */

        /* program utama */
                    :
        <nama> = fungsi1(<parf1>, <parf2>, ...);
                    :
        prosedur1(<parp1>, <parp2>, ...);
                    :


        /* jika <tipe-m> bukan void, kirim ret. val. ke
        lingkungan */
        return(<retval>);
}

/************************************************/
/* Body/Realisasi dari fungsi dan prosedur */
<tipe-f> fungsi1(<tipe-pf1> param1, <tipe-pf2> param2, ...) {
        /* keterangan fungsi  */
        /* kamus lokal */
        /*prototipe fungsi/prosedur bantu */

        /* isi fungsi */
                    :
                    :
        /* kirim ret. val. ke pemanggil, tipe <retval> harus
           dapat dikonversi ke <tipe-f> */
        return(<retval>);
}

/*** lanjutan ****/
/* Body/realisasi dari prosedur  */

void prosedur1(<tipe-pp1> param1, <tipe-pp2> param2,...){
        /* keterangan prosedur */
        /* kamus lokal */
        /*prototipe fungsi/prosedur bantu */

        /* isi prosedur */
                    :
                    :
        [return;]

}
```

Contoh sebuah program lengkap dalam satu file yang mengandung sebuah fungsi untuk menentukan nilai maksimum dua buah variabel, dan sebuah prosedur untuk menukar isi nilai dari dua buah variabel:

```
#include <stdio.h>

/* prototipe */
int max(int a, int b);
void tukar(int*a, int*b);

 /* Variabel global : tidak perlu ada */

void main()
/* membaca dua harga dari keyboard */
/* menentukan maksimum, menuliskan ke layar */
/* menukar kedua harga, menuliskan ke layar */
{
      /* nama yang dipakai */
      int bil1, bil2, maxi;

      /* Baca */
      printf("Ketik dua bilangan ");
      scanf("%d %d", &bil1, &bil2);
      printf("Kedua bilangan: bil1 = %d, bil2 = %d\n",
      bil1, bil2);
      /* Maksimum */
      maxi = max(bil1, bil2);
      printf("Maksimum dari dua bilangan %d:\n", maxi);
      /* Tukar */
      tukar(&bil1, &bil2);
printf("Ditukar: bil1 = %d, bil2 = %d\n", bil1, bil2);
}
 /* realisasi fungsi max */
int max(int a, int b) {
/* mengirimkan harga maksimum dua bil. bulat */
      return((a>b)?a:b);
}
  /* realisasi prosedur tukar */
void tukar(int *a, int* b) {
/* menukar dua harga bilangan bulat */
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```

Parameter :

- Perhatikan cara menuliskan parameter formal Output dan parameter Input/Output. Pada bahasa C diimplementasi dengan pointer. Pemakaian pointer sebagai sarana "passing parameter" pada kuliah ini dibedakan dengan pemakaian pointer sebagai representasi informasi struktur data). Lihat contoh pendefinisian prosedur tukar dan bagian pointer.
- Parameter aktual dari parameter Output atau Input/Output dituliskan dengan operator *"address of"* (`&`). Lihat pada contoh pemanggilan prosedur tukar.

SATU PROGRAM UTUH DALAM BEBERAPA FILE

Sebuah program dalam bahasa C yang "utuh", seringkali terdiri dari beberapa modul program. Sebuah modul mungkin mewakili sekumpulan rutin sejenis, **ADT** (Abstract Data Type: definisi type dan primitifnya), atau **mesin** (definisi state variable dari mesin dan primiritfnya).
Pada hakekatnya, dalam hal ini program harus mempunyai sebuah `main()` dan memanfaatkan modul yang lain. Program yang dibagi-bagi menjadi beberapa file seharusnya dapat dikompilasi terpisah (setiap modul membentuk sebuah *object code*). Dengan demikian, pembuatan sebuah *executable code*

dapat dilakukan dengan melakukan link terhadap sejumlah object code yang sudah dikompilasi (penghematan waktu dan duplikasi usaha, reusability)
Supaya dapat dikompilasi dengan benar, modul yang lain dapat dilakukan dengan melakukan include terhadap file header. File header dalam bahasa C adalah sebuah file yang hanya berisi deklarasi type dan prorotype fungsi ("tidak boleh" ada deklarasi variabel). Standard penulisan file header dapat dilihat pada contoh program kecil.
Contoh program lengkap dengan beberapa modul yang diletakkan dalam beberapa file dapat dilihat pada Contoh Program kecil untuk pemanfaatan ADT JAM yang didefinisikan.

Skema program yang dipecah menjadi beberapa file.
Pada hakekatnya sebuah program utuh terdiri dari kelompok file sebagai berikut:

1. File header, dengan nama xxx.h. Untuk setiap type dan primitifnya, ada sebuah file xxx.h. Contoh: jika anda memerlukan ADT JAM, DATE dan mesin KATA maka ada 3 buah file header yaitu Jam.h, DATE.h dan KATA.h
2. File yang berisi BODY dari File header yang bersangkutan: xxx.c. File ini disebut sebagai file realisasi dari prototype yang didefinisikan pada xxx.h. Akan ada sebuah xxx.c untuk setiap xxx.h. Untuk contoh di atas, akan ada JAM.c, DATE.c dan KATA.c.
3. File yang berisi main program (dan prosedur/fungsi lain yang hanya dibutuhkan oleh main), misalnya dengan nama main.c

Jadi sebuah program utuh akan terdiri dari sebuha main.c,sebuah xxx.h dan xxx.c
Skema penulisan program utuh untuk setiap jenis file diberikan sebagai berikut

1. File header :

```
/* File xxx.h                                                  */
/* Deskripsi : keterangan isi file header                      */
/* Isi : deklarasi type dan variabel                           */
/*       deklarasi TYPE dan PROTOTYPE                          */
/* File header tidak boleh mengandung deklarasi variabel!!!    */
/*                                                             */
#ifndef xxx_h
#define xxx_h
/* Bagian I : berisi deklarasi konstanta                      */



/* Bagian II : berisi deklarasi TYPE                          */






/* Bagian III : berisi deklarasi prototype prosedur dan fungsi*/
/* yang merupakan primitif TYPE tsb  */
/* Kelompokkan fungsi dan prosedur sesuai dengan standard di kelas*/
/* Misalnya : konstruktor, selektor, predikat, operator relasional*/
/* operator aritmatika, operator lain dsb */



#endif
```

2. File BODY :

```
/* File xxx.c                                                  */
/* Deskripsi : keterangan isi file body                        */
/* Isi : realisasi/kode program dari semua prototype           */
/*       yang didefinisikan pada xxx.h                         */
/* Untuk sebuah MESIN, akan mengandung deklarasi state variabel */
/*   dari mesin ybs                                            */
```

```
#include "xxx.h"
/* Realisasi kode program, sesuai urutan pada xxx.h         */
/* Copy dari xxx.h, kemudian edit */
```

3. File main program

```
/* File : main.c                                                    */
/* Deskripsi : program utama dan semua nama lokal thd persoalan*/

#include "xxx.h"
/* include file lain yang perlu */

/* Bagian I : berisi kamus GLOBAL dan Prototypr             */
/* deklarasi semua nama dan prosedur/fungsi global*/

/* Bagian II : PROGRAM UTAMA                                */
int main () {
/* Kamus lokal terhadap main */

/* Algoritma */


   return 0;

}

/* Bagian III : berisi realisasi kode program yang merupakan  */
/* BODY dari semua prototype yang didefinisikan pada file ini */
/* yaitu pada bagian I, dengan urut-urutan yang sama          */
/* Copy prototype, kemudian edit!! */
```

Untuk memproses, lakukan kompilasi terpisah untuk xxx.c dan main.c, kemudian link untuk membentuk sebuah executabel file.
Pada kuliah ini, untuk setiap xxx.h dan xxx.c, dibuat main program yang sengaja dibuat untuk mentest setiap prosedur dan fungsi yang dibuat, yang disebut sebagai "driver", atau "test stub". xxx.h, xx.c dan mxxx.c disimpan dalam sebuah direktori dan hasil test juga disimpan dalam direktori yang sama dan terpisah. Dengan cara semacam ini, sebuah modul yang sudah ditest dengan baik dapat digunakan ulang secara efisien, dan terdokumentsi dengan rapi.

PROGRAM SANGAT BESAR DENGAN VARIABEL GLOBAL

Untuk program besar, sangat perlu untuk menjaga konsistensi deklarasi/definisi objek. Karena itu, semua variabel global dalam program didefinisikan/dideklarasikan dalam suatu file header. File header ini lalu dimasukkan ke dalam semua unit translasi yang ada. Perubahan pada suatu variabel cukup dilakukan dengan mengubah file header tersebut. Contoh berikut diadaptasikan dari [Tanenbaum-91], dan merupakan pengecualian, dimana kita boleh meletakkan variabel dalam file header berekstensi `.h`. Karena file berisi variabel dengan maka harus dijamin bahwa tidak terjadi lebih dari satu kali inclusion!

```
      /* File : global.h   */
            #ifdef UTAMA
            #define EXTERN
            #else
            #define EXTERN extern
            #endif
            /* definisi & deklarasi variabel-variabel global */
            EXTERN int x;
            EXTERN char pola = {'i', 'f', '2', '2', '3'};
             :
```

```
/* File : file1.c */
/* File1.c adalah salah satu modul yang akan dipakai oleh main
program */
/* jika ada lebih dari 1 file, maka akan ada file2.c, file3.c dst */
/* Mungkin saja setiap filex.c mempunyai filex.h yang sesuai */
```

```
#include "global.h"


/* Kode-kode program di dalam file1.c */



/* File : main.c */
/* UTAMA didefinisikan agar semua variabel di global.h terdefinisi */
# define UTAMA
# include "global.h"

/* fungsi di file MAIN */

/**** Main program: di halaman berikut ******/

int main ()
{ /* Kamus "lokal" thd main */

  /* Algoritma main program */

  return 0;
}
```

INPUT/OUTPUT

Perintah input/output bukan merupakan bagian dari definisi bahasa C yang standard. Namun demikian, semua program memerlukan input/output untuk berkomunikasi dengan dunia luar.
Tidak mengherankan bahwa ada banyak sekali variasi perintah untuk input/output
Pada kuliah ini, Input/ouput standard yang digunakan adalah input/output berformat sebagai berikut
Untuk membaca dari papan kunci :

```
scanf ("<format>", <list-vatiabel);
```

Untuk menuliskan ke layar

```
printf ("<format>", <list-vatiabel);
```

Perhatikanlah bahwa penulisan dengan format yang benar sangat penting dalam sebuah program C. Suatu nilai dapat dibaca/tulis dengan format apapun (tidak menimbulkan kesalahan kompilasi), namun akan memberikan hasil yang "aneh-aneh" karena format sebenarnya adalah memetakan representasi internal dengan penampakan di layar.
Ada beragam format dan kombinasinya (baca pada referensi). Secara ringkas, format yang dipakai sesuai dengan type variabel yang dibaca/ditulis adalah :

Format	Untuk type
%d	int
%f	float
%c	char
%s	string

Lihat contoh program kecil untuk pemakaian format yang sangat sederhana.

Format Output

Bentuk:

```
+-----------------------------------------------+
¦  % [flag] [lebar][.presisi] [F|N|h|l|L] type ¦
+-----------------------------------------------+
```

Keterangan:

```
flag   : Justifikasi, tanda numerik, titik desimal,
         ekor nol (trailing zero), prefiks oktal/hex.
lebar  : Jumlah minimum karakter yang dicetak,
         dipenuhi (padded) dengan blank atau nol.
```

```
 presisi: Jumlah maksimum karakter dicetak;
          untuk integer, jumlah minimum digit untuk dicetak.
 size   : Override ukuran argumen default:
                  N = near pointer
                  F = far pointer
                  h = short int (d, i, o, u, x, X)
                  l = long int (d, i, o, u, x, X);
                       double (e, E, f, g, G)
                  L = long double (e, E, f, g, G)
  tipe  :   Tipe argumen yang dicetak.
```

FLAG:

Karak.	Arti
-	Rata kirikan hasil, isi bagian kanan dengan spasi. Jika tidak, hasil rata kanan,bagian kiri diisi spasi.
+	Hasil konversi bertanda selalu diawali dengan + atau -.
blank	Jika nilai = 0, output diawali dengan blank. Jika < 0, output diawali dengan -.
#	Argumen dikonversi dengan format alternatif: Karakter konversi — Efek # terhadap argumen c,s,d,i,u — tak ada o — 0 ditambahkan di awal argumen <> 0 x,X — 0x (0X) ditambahkan di awal argumen e,E,f — titik desimal selalu ada g,G — sama dengan e,E,f, dengan tambahan ekor nol tak dihilangkan

LEBAR:

Spek. Lebar	Efek terhadap lebar output
n	Minimum n karakter tercetak. Jika output kurang dari n, ditambahkan blank (sesuai flag).
0n	Minimum n karakter tercetak. Jika output kurang dari n, ditambahkan nol/'0' (sesuai flag).
*	Daftar argumen mengandung spesifikasi lebar, yang mendahului argumen yang diformat.

PRESISI:

Spek. prec.	Efek terhadap output
	(tak ada), presisi diset default: 1 untuk d, i, o, u, x, X 6 untuk e, E, f sebanyak angka penting untuk g, G sepanjang string untuk tipe s
.0	Untuk d, i, o, u, x presisi default; untuk e, E, f titik desimal tak dicetak.
.n	n karakter atau n posisi desimal dicetak. Jika harga output > n karakter, output dipotong atau dibulatkan.
*	Daftar argumen mengandung spesifikasi presisi, yang mendahului argumen yang diformat.

TIPE:

Karakter	Tipe argumen	Dikonversikan ke
d,i	int	bilangan desimal bertanda
o	int	bilangan oktal takbertanda
x	int	bilangan hex takbertanda, dengan karakter abcdef
X	int	bilangan hex takbertanda, dengan karakter ABCDEF
u	int	bilangan desimal takbertanda
c	int	karakter (setelah dikonversi ke karakter takbertanda)
s	char*	karakter-karakter dari string s dicetak hingga ditemui '\0' atau presisi tercapai
f	double	notasi desimal dengan bentuk [-]mmm.ddd, jumlah d ditentukan oleh presisi (default 6), m adalah mantissa angka tsb.
e,E	double	notasi desimal dengan bentuk [-]m.ddddddE±xx, jumlah d ditentukan oleh presisi (default 6).
g,G	double	%e atau %E dipakai jika eksponen <-4 atau = presisi; jika tidak, dipakai format %f. Ekor nol atau titik desimal tak tercetak.
p	void*	dicetak sebagai pointer (tergantung implementasi)
n	int*	jumlah karakter yang telah tertulis pada pemanggilan printf akan dicetak, tak ada argumen terkonversi
%		mencetak %, tak ada konversi argumen

Format Input

Bentuk:

```
+----------------------------------+
¦ % [*] [lebar] [F|N] [h|l|L] type ¦
+----------------------------------+
```

Keterangan:

```
  *      :   Input field berikut tak diberi assignment.
  lebar  :   Jumlah maksimum karakter yang dibaca.
  ukuran:    Override ukuran default argumen:
                   N = near pointer
                   F = far pointer
  Tipe Arg.:  Override tipe default alamat argumen:
                   h = short int (d, i, n, o, u, x)
                   l = long int (d, i, n, o, u, x)
                          double (e, f, g)
                   L = long double (e, f, g)
```

TIPE:

Karakter	Tipe argumen	Input yang diharapkan
d,i	int*	integer desimal
o	int*	integer oktal
u	unsigned int *	integer desimal takbertanda
x	int*	integer hex
c	char*	karakter input berikut disimpan ke array c, hingga sebanyak yang dispesifikasikan lebar. Skip pada karakter "blank" tak dilakukan. '\0' tak ditambahkan.
s	char*	string dari karakter bukan white space, '\0' ditambahkan.
e,f,g	float*	bilangan floating point
p	void*	pointer ke suatu objek
n	int*	menyimpan jumlah karakter yang telah terbaca pada pemanggilan scanf, tak ada input yang dibaca

Karakter	Tipe argumen	Input yang diharapkan
[...]	char*	mencari string tak kosong terpanjang dari input yang berasal dari himpunan karakter dalam kurung, '\0' ditambahkan.
[^...]	char*	mencari string tak kosong terpanjang yang tidak mengandung karakter dalam kurung. '\0' ditambahkan.
%		literal %

FILE EKSTERNAL

Operasi Primitif terhadap file yang penting adalah :

- deklarasi nama file dalam program
- mengasosiasikan nama internal program dengan nama fisik file
- membuka file (dapat dilakukan sekaligus dengan asosisasi), dan menentukan untuk dibaca, ditulis, atau baca/tulis
- mengakses rekaman dalam file
- menentukan EOF (End Of File), supaya pembacaan dapat dihentikan
- menutup file ketika file sudah tidak dibutuhkan.

Ada banyak sekali variasi perintah untuk melakukan primitif operasi di atas (dapat dibaca secara lengkap dalam referensi kompilator ybs).
Standard yang dipakai di kuliah ini adalah perintah standard sebagai berikut untuk file sekuensial yang dibaca per karakter.

Operasi	Perintah	Contoh

Deklarasi nama lojik dalam program	`FILE * <namafile>`	`FILE *fp;`
		`int retval; /* kode return value*/`
		`char CC;`

Buka	`fopen(char *name,char*mode)`	`fp-fopen("mydat", "r")`
Baca sebuah rekaman	`retval=fscanf(fp,<format>,var>)`	`retval=fscanf(fp,"%c",CC);`
End of file (konstanta)	`EOF`	`retval = EOF`
End Of File (fungsi)	`fEOF(<namafile>)`	`fEOF(fp)`

Contoh skema pembacaan dan penulisan file sekuensial dapat dilihat pada Contoh program kecil.

Mode pada fopen adalah

Mode	Arti
`"r"`	Baca saja
`"w"`	tulis, buang isi yang lama
`"a"`	append, tulis di akhir
`"r+"`	update (baca dan tulis)
`"w+"`	update, buang isi yang lama
`"a+"`	update, di akhir file
`"b"`	file biner

PUSTAKA BAHASA C YANG STANDARD

Bahasa C menyediakan pustaka standard (standard library) yang siap dipakai dengan file inclusion. berikut ini diberikan daftar pustaka standard yang tersedia. isinya dapat dilihat pada buku referensi atau lewat fasilitas Help. Pakailah fungsi yang ada sebelum anda menuliskannya sendiri, dan include-lah dengan benar file di mana fungsi tsb berada.

Nama file	Isi
stdio.h	Berisi fungsi, type dan makro yang terdefinisi untuk operasi Input/Output, yaitu: operasi file format output format input fungsi input dan output untuk karakter fungsi untuk “direct input and output” Menentukan posisi pada file fungsi-fungsi sehubungan dengan error
ctype.h	Character Class type, yaitu fungsi-fungsi untuk melakukan test terhadap karakter
math.h	fungsi matematik
stdlib.h	berisi deklarasi fungsi untuk konversi, alokasi memori dan sejenisnya
string.h	Fungsi-fungsi untuk manipulasi string, yang terdiri dari dua kelompok: kelompok yang dimulai dengan str, untuk manipulasi string (misalnya copy, membandingkan, dsb) kelompok yang dimulai dengan mem, untuk manipulasi objek bertype caharacter array
assert.h	untuk diagnostik program
stdarg.h	fasilitas untuk memanfaatkan argumen fungsi
setjmp.h	untuk menghindari pemanggilan/pengembalian nilai fungsi yang normal. Berguna misalnya untuk keluar langsung dari nested function call yang rumit
signal.h	untuk menangani exception yang terjadi selama eksekusi, misalnya interrupt dari piranti eksternal atau error pada saat eksekusi
time.h	berisi type dan fungsi untuk manipulasi waktu (date, time)
limit.h	Tergantung implementasi kompilator. Berisi nilai konstanta untuk type integral (misalnya nilai integer minimum, maksimum dsb)

Prototype dari semua fungsi yang tersedia dapat dibaca pada setiap header file tersebut, dan dapat dilihat dengan fasilitas on-line Help dari kompilator yang anda pakai. Jika anda belum familiar, disarankan agar anda melakukan penyalinan tekstual, dan mengganti parameter sesuai dengan kebutuhan.

TERJEMAHAN DARI NOTASI ALGORITMIK KE C :

ALGORITMA	C
1. ASSIGNMENT : <nama> <- harga	 <nama> = harga;
2. KONDISIONAL : If <kondisi> then Aksi-A	 if (kondisi) Aksi-A;

if <kondisi>
then
 	
Aksi-A
 	
else
 	
Aksi-B
	if (kondisi)
	
	Aksi-A;
	
	else
	
	Aksi-B;

depend on <nama> <kondisi-1> : Aksi-1 <kondisi-2> : Aksi-2 <kondisi-3> : Aksi-3 . . . . <kondisi-n> : Aksi-n	if (kondisi-1) Aksi-1; else if (kondisi-2) Aksi-2; else if (kondisi-3) Aksi-3; else if (kondisi-4) . . . else if (kondisi-n) Aksi-n;

Catatan:

Untuk depend-on, jika <kondisi-1>..<kondisi-n> berbentuk
<nama-var> = <ekspresi konstan>
dapat dipakai statement switch:

```
switch (NamaVar) {
      case exp-konstan-1: Aksi-1;
      case exp-konstan-2: Aksi-2;
      case exp-konstan-3: Aksi-3;
    :
      case exp-konstan-1: Aksi-1;
}
```

ALGORITMA	C
3. PENGULANGAN :	
while <kondisi-ULANG> do Aksi	while (kondisi-ULANG) Aksi;
repeat Aksi until<kondisi-STOP>	do Aksi while !(kondisi-STOP);

iterasi
 	
Aksi-A
 	
stop <kondisi-STOP>
 	
Aksi-B
	for(;;) {
	
	Aksi-A;
	
	if (kondisi-STOP)
	exit;
	else
	AKSI-B;
	}

i traversal[Awal..Akhir]
 	
Aksi
	/* Jika Awal <= Akhir */
	for(i=Awal, i<=Akhir, i++)
	
	Aksi;
	
	/* Jika Awal >= Akhir */
	for(i=Awal, i>=Akhir, i--)
	
	Aksi;

4. INPUT/OUTPUT	
read <nama>	fscanf(stream, format, &nama) ; scanf(format,nama);
write <nama>	fprintf(stream, format, nama); printf (format, nama);