Pengolahan Citra
Pada
Mobil Robot

Tabratas Tharom

tharom@yahoo.com

LAMPIRAN A

PEMROGRAMAN BAHASA C/C++

Bahasa C populer dengan kekuatannya, keefisienannya, dan juga kerumitannya. Bahasa ini dianggap dan diakui sebagai bahasa tingkat tinggi yang bisa seefisien *assembly* karena konstruksi bahasanya sangat dekat dengan keadaan sebenarnya di bahasa mesin. Sebagai bahasa sistem operasi, C/C++ digunakan luas : dari jajaran **UNIX** (yang membuat populer bahasa ini), **Microsoft Windows** (**NT/98/2000**), **Macintosh OS** (sejak versi **PowerPC-**nya, bahasa resmi **Apple** ialah C/C++), dan **BeOS.** Hampir semua OS yang ada di dunia mempromosikan penggunaan bahasa C.

Kita mulai dengan *data structure*. Tipe data standar dalam C antara lain adalah:

- `int`: bilangan bulat, tersedia dalam bentuk `short int` atau `long int` (`int`-nya boleh dibiarkan ada atau dihilangkan). `short` berarti 16 bit dan

`long` berarti 32 bit. Tanpa jenis `short/long` berarti memakai ukuran *natural* bagi mikroprosesor yang digunakan (kalau 8086 : `int` = 16 bit, kalau 80386 : `int` = 32 bit)

- `char` : karakter, 8-bit.
- `float` : bilangan real dalam bentuk *floating point*, 32-bit
- `double` : bilangan real dalam bentuk *floating point* 64-bit

Deklarasi dalam C harus diletakkan sebelum instruksi pertama dalam suatu blok program.

Contoh:

```
.....
{
        int i;
.....
```

Untuk menyatukan data bertipe sama digunakan *array;* sintaksnya ialah setelah nama variabel yang dideklarasikan kurung siku buka, angka yang menyatakan banyaknya elemen, lalu kurung siku tutup,

```
....
{
        int i[100];
....
```

Untuk menyatukan data bertipe berbeda, tapi harus diasosiasikan sebagai satu objek, digunakan `struct`.

Sintaksnya:

```
struct nama_tag {
        ... (deklarasi-deklarasi) ...
} nama_variabel;
```

Baik `nama_tag` ataupun `nama_variabel` hanya wajib ada satu (kalaupun keduanya ada, boleh juga). Deklarasi ialah variabel yang akan ada di dalam struktur berformat `nama_tag. Nama_variabel` adalah nama variabel yang dideklarasikan bertipe struktur berformat nama_tag. Untuk

menyatakan bahwa suatu variabel mungkin memiliki beberapa tipe, digunakan `union`.

```
union nama_tag {
        ... (deklarasi-deklarasi) ...
} nama_variabel;
```

Maka *memory* yang disiapkan untuk nama_variabel bisa diakses sebagai beberapa tipe data. Merupakan kewajiban pemrogram untuk menyadari bahwa tiap deklarasi dalam satu `union` akan mengacu ke blok *memory* yang sama saja, meskipun berbeda nama.

Dalam pemrograman, metode yang paling sering dipakai untuk memecahkan masalah ialah `divide` and `conquer`. Maka dalam bahasa C sangat mengakar yang disebut sebagai fungsi (atau *subprogram*) yang merupakan rutin kecil yang membangun program. Masing-masing rutin kecil ini bisa dipanggil berulang kali oleh bagian program utama yang disebut fungsi `'main'`.

```
...
tipe_fungsi nama_fungsi(..daftar parameter..)
{
        ..deklarasi..
        ..badan fungsi..
}
```

Tipe data fundamental

- tipe `integer` (bilangan bulat)
- `char` 8-bit(1 byte) -128..127
- `short` 16-bit(2 byte) -32768..32767
- `int` 16-bit atau 32-bit (tergantung pada *compiler*-nya, dibuat untuk prosesor 16-bit atau 32-bit), *range* untuk yang 32-bit ialah –2 147 483 648..2 147 483 647
- `long` 32-bit –2 147 483 648..2 147 483 647

- Jika kata '`unsigned`' ditambah di depan tipe integral ini, maka *range*-nya berubah menjadi dari 0..2*max+1.
- Tipe real
- `float` 8-bit *excess*-127 *binary exponent*, 23-bit *mantissa*, 1-bit sign (total 32 bit = 4 bytes) sesuai dengan standar IEEE untuk *floating point* 32-bit
- `double` 11-bit *excess*-1023 *binary exponent*, 52-bit mantissa, 1-bit sign (total 64 bit = 6 bytes, sesuai dengan standar IEEE untuk *floating point* 64-bit)
- `long double`, sesuai dengan standar IEEE untuk *floating point* 80-bit (10 bytes).

DEFINISI

- *Source Program*: kumpulan deklarasi, definisi, variabel, fungsi, dan *statement*, terdiri atas satu atau lebih *source file.*
- *Source File*: sebuah file teks berisi semua atau sebagian dari C *source program* tadi. (Misalnya, hanya berisi beberapa fungsi yang dibutuhkan program). Ketika sebuah program di-*compile* (diterjemahkan dari teks menjadi bahasa mesin), tiap-tiap *source* file harus *di-compile,* untuk kemudian di-*link* (digabung) satu sama lain dan juga dengan *library* yang digunakan, menjadi sebuah file *executable* (bisa dieksekusi oleh *operating system*).

Tiap program memiliki fungsi utama yang disebut '`main`', merupakan bagian dari program yang pasti dieksekusi waktu file hasil *link* tadi dijalankan. Sebagai fungsi utama sebenarnya hanya '`main`' yang dieksekusi. Di dalam fungsi '`main`' ini terjadi pemanggilan fungsi lainnya. Pengecualian hanyalah pada pemrograman Windows; fungsi yang dieksekusi pada awal program ialah fungsi '`WinMain`'. Hal ini tidak berarti bahwa jika Anda menggunakan Windows Anda harus membuat fungsi

`WinMain`. Jika Anda membuat project berjenis *Win32 console application,* maka otomatis fungsi `WinMain` akan disertakan pada salah satu *file library*, yang di dalamnya memanggil fungsi `'main'`. Pada *operating system* selain Windows, hal ini tidak terjadi (fungsi utama selalu bernama `'main'`).

- *Identifier:* nama pengenal suatu variabel/fungsi.
- *Object*: bagian tertentu dari memori yang bisa memuat suatu nilai. Tiap objek memiliki nama (*identifier*) dan tipe (*data type*).
- *Object file:* file berisi kode mesin beserta informasi simbolik fungsi dan variabel.
- *Scope*: jangkauan suatu *identifier* ialah bagian program di mana sebuah *identifier* bisa digunakan untuk mengakses objeknya.

JENIS *SCOPE*

- ***block (lokal)***. *Scope*-nya dimulai dari deklarasi sampai akhir blok.
- ***Function.*** *Scope* dimulai dari awal fungsi sampai akhir fungsi, hanya label untuk `goto` yang memiliki *function scope.*
- ***Function prototype***.
- ***File (global).*** *Scope*-nya dari titik deklarasi sampai akhir file.

- *Deklarasi*: 'pengumuman' mengenai variabel dan fungsi yang akan digunakan program. Sebuah variabel atau fungsi tidak boleh dipakai sebelum dideklarasikan.

Pada C (bukan C++), ada pengecualian untuk fungsi; jika belum dideklarasikan, fungsi dianggap bertipe `int` dan semua parameternya juga bertipe `int`.

Sintaks:

```
[[ sc-specifier ]] [[ type-specifier ]] declarator [[ = initializer ]] [[, declarator = [[initializer]] ]]…
```

Catatan: '[[' dan ']]' berarti bahwa suku di antara kedua tanda ini boleh ada atau boleh tidak ada.

- *sc-specifier* ialah *storage-class specifier.* Ada empat yang didefinisikan C : `auto, extern, register,` dan `static`. Deklarasi berskop lokal/blok *default*-nya `auto`, deklarasi berskop global (variabel global) *default*-nya `static. 'auto'` berarti variabel itu bersifat otomatis, dialokasikan jika fungsi itu dimulai, dan didealokasikan jika fungsi itu berakhir. Jadi, antara akhir pemanggilan fungsi dengan awal pemanggilan berikutnya tidak ada jaminan bahwa nilainya tetap. Kalau `'static'` variabel itu dialokasikan sekali untuk seumur program itu, jadi antara pemanggilan fungsi dijamin tidak akan berubah. `'register'` : variabel itu sedapat mungkin diletakkan di memori berkecepatan paling tinggi, yaitu register. `'extern' :`
- *type-specifier* ialah jenis tipe variabel/fungsi yang dideklarasikan. Jika tidak diberikan, *default*-nya adalah `signed int` (bilangan bulat bertanda)
- *declarator* ialah nama yang bisa ditambahkan;kurung siku [] untuk *array, asterisk (*)* untuk *pointer,* atau tanda kurung '( )' untuk fungsi.
- *initializer* ialah nilai mula bagi variabel yang dideklarasikan, merupakan ekspresi bertipe sama dengan *type-specifier*. Untuk objek bersifat `'static'` harus berupa ekspresi konstan. Pada C++ boleh merupakan ekspresi yang menggunakan fungsi dan variabel yang telah dideklarasikan sebelumnya. Untuk objek bersifat `'auto'` (*automatic*), *initializer* boleh berupa ekspresi apa pun yang legal.
- *Array:* sederetan data bertipe sama yang diakses dengan indeks.
- *Fungsi :* sekumpulan deklarasi dan *statement*.

Definisi fungsi menentukan nama fungsi, tipe fungsi, nama dan tipe parameter-parameter fungsi itu, dan deklarasi dan statement yang menentukan apa yang dilakukan fungsi itu.

Sintaks:

```
[[sc-specifier]] [[type-specifier]] declarator ([[parameter-list]])
```

- *function-body*: Deklarasi fungsi/*prototype* menentukan nama, tipe fungsi, dan tipe parameter fungsi itu saja supaya fungsi itu bisa dipakai meskipun definisinya belum ada. Sintaksnya sama dengan definisi, tanpa *function-body*, dan bisa tanpa nama parameter.
- *Statement:* terdiri atas kata kunci *(keyword),* ekspresi, dan bisa juga statement lainnya, mengendalikan jalannya program.

JENIS STATEMENT

- `if, while, for, break, return`
- `do:` sintaksnya: `do` *statement* `while` (*expr*); untuk melakukan *statement* berulang-ulang sampai *expr* bernilai *false*.
- `continue:` dipakai untuk melanjutkan suatu *loop* `for` ke nilai berikutnya
- `switch:`
  sintaksnya:

```
switch (expr) {
case const-expr1 : statement1;
case const-expr2 : statement2;
default:
default-statement;
}
```

Maka berdasarkan nilai *expr*, dicari nilai *const-expr* yang sama dengan *expr*, dan *statement* setelah *case* dengan *const-expr* yang itu dieksekusi. Kalau tidak ada yang sama, *default-statement* yang akan

dieksekusi. Biasanya tiap *statement* diakhiri dengan `break` untuk mencegah eksekusi *statement* untuk *case* berikutnya.

- *Compound (blok):* dengan mengawali sekumpulan *statement* dengan '{' dan mengakhirinya dengan '}' maka sekumpulan *statement* itu menjadi satu blok *statement*.
- *Expression:* jadi satu ekspresi bisa dianggap sebagai *statement*, dengan catatan nilai hasil ekspresi itu tidak diacuhkan.
- *Sintaks*: aturan penulisan.
- *Operand*: konstanta atau nilai variabel yang akan dihitung/dioperasikan
- *Operator*: menunjukkan dengan cara apa *operand* dihitung/dimanipulasi untuk menghasilkan suatu nilai.
- *Presedensi operator (operator precedence)*: urutan prioritas operator dalam pengasosiasian operator dengan *operand*-nya.

Semua *operand* adalah ekspresi, karena semua *operand* mempunyai nilai. Semua ekspresi bisa dijadikan *operand* kalau dia dianggap sebagai satu kesatuan untuk dioperasikan dengan *operand* lain.

Jenis *operand*/ekspresi:

- **konstanta,** suatu bilangan yang jelas ditulis angkanya.
- **Identifier** (nama suatu variabel/fungsi), nama fungsi akan mewakili *pointer* yang berisi alamat fungsi tersebut, begitu pula nama suatu *array* akan mewakili alamat elemen pertama *array* tersebut.
- **string** (karakter dibatasi kutip ganda ["]), dalam C string adalah deretan karakter yang akhirnya ditandai oleh karakter nul (0) yang otomatis ditambahkan ke setiap konstanta *string*.
- **function call** / pemanggilan fungsi, bersintaks: *expr* ([*expr-list*])
- **expr** bernilai suatu alamat fungsi yang akan dipanggil (ingat bahwa nama fungsi mewakili alamat fungsi tersebut jika dipakai sebagai *operand*)

- **expr-list** berisi suatu daftar ekspresi yang dipisahkan dengan koma, yang nilai-nilainya akan diberikan kepada fungsi yang dipanggil sebagai parameternya. Untuk C, hanya nilai yang dilewatkan;artinya, jika salah satu parameter fungsi diisikan ekspresi berupa variabel, dan fungsi itu mengubah parameternya tersebut, maka variabel tadi, yang berada di luar fungsi, tidak akan ikut berubah, meskipun dalam perhitungan di fungsi itu parameternya sudah terubah oleh fungsi itu sendiri.
- **Ekspresi subskrip unidimensional,** bersintaks: *expr1 [expr2],* ditandai oleh kurung siku yang khas untuk ekspresi jenis ini. Ekspresi ini mewakili nilai elemen ke-*expr2* dari *array* yang dimulai pada alamat *expr1.* Secara *pointer,* ekspresi ini ekivalen dengan **(expr1+expr2).* Catatan: *expr1* harus *pointer* ke suatu tipe data (`int, char`, dll.) dan *expr2* haruslah integral/bilangan bulat. Lihat aturan penjumlahan *pointer*.
- **Ekspresi subskrip multidimensional [optional]**: sintaks: *expr1 [expr2] [expr3] [expr4] …* Ekspresi jenis ini diasosiasikan dari kiri ke kanan. Pertama, subskrip terkiri dihitung yaitu *‘ expr1 [expr2]’* dengan aturan yang sama dengan *subskrip unidimensional*. *Expr1* dan *expr2* dijumlahkan menjadi suatu ekspresi *pointer*, lalu didereferensi. Kemudian, nilai yang didapat dijumlahkan dengan *expr3,* didereferensi, dst sampai habis. Lihat aturan penjumlahan *pointer*.
- **Ekspresi seleksi-anggota:** Ekspresi ini mengekspos anggota dari suatu struktur supaya bisa dioperasikan. Sintaks: *expr1.identifier atau expr2->identifier*

- **expr1** ialah struktur yang memiliki anggota bernama *identifier*
- **expr2** ialah *pointer* ke struktur yang beranggota bernama *identifier*

Maka, anggota struktur bisa dioperasikan sebagai suatu *operand*.

- **Ekspresi dengan operator**: (lebih detail di bagian operator)
- **Operator unary**, bersintaks: *unop operand*
- **Operator binary**, bersintaks: *operand1 binop operand2*
- **Operator ternary**, bersintaks: *cond_operand* ? *operand1* : *operand2*
- **Ekspresi dengan tanda kurung**: untuk memastikan asosiasi *operand* dengan operator. Sekumpulan *operand* dan operator dalam sepasang tanda kurung dianggap sebagai satu operand bagi operator diluar kurung.
- **Ekspresi type-cast**: merupakan cara eksplisit untuk mengubah tipe *operand*, sintaks: (*tipe-name*) *operand*
- **Ekspresi konstan**: ekspresi yang dihitung dari konstanta sehingga hasilnya pun konstan pula.

JENIS OPERATOR

UNARY

- **- ~ !** (minus, tilde, dan tanda seru) berarti (negasi, inversi bit, dan inversi boolean) [sesuai urutannya dlm koma]
  inversi boolean : *true* dibuat jadi *false,* dan *false* dibuat jadi *true.* Tapi ingat bahwa *true* itu semua yang bukan nol, dan *false* itu nol.
- ***** (indireksi / dereferensi) dan & (operator alamat-dari / *address-of)* kedua operator ini saling berlawanan kerjanya.
- `'sizeof'`
- **+ (unary plus)**

BINARY

- *** (perkalian) / (pembagian) %** (modulo/sisa bagi)
- **+ -**
- **<<** (pergeseran bit ke kiri) **>>** (pergeseran bit ke kanan)
- **< > <= >= = = !=** (operator perbandingan)
- **& (bitwise AND) | (bitwise OR) ^ (bitwise XOR)**

- **&& (boolean AND) || (boolean OR)**

  Operator ini bekerja dengan *operand* bertipe *bool (*atau dikonversikan ke *bool* jika belum) dan menghasilkan nilai *bool*.

- **,** (koma : operator *evaluasi sekuensial*)

  contoh:

  `if ((b=3,c=f(b),a)= =2) break ;` akan mengisi b dengan 3, memanggil fungsi f dengan b sebagai parameter, hasilnya diletakkan dalam c, dan mengambil nilai a sebagai nilai ekspresi ini, memeriksa apakah nilai a itu sama dengan dua, jika benar melakukan `break`.

Operator binary asosiasinya dari kiri ke kanan, artinya 6/2/3, yaitu 1 dan bukannya 4.5

TERNARY

Hanya ada satu, yaitu *conditional operator* **'? :'**

Asosiasinya dari kanan ke kiri.

Sintaks:

cond_expr ? *expr1* : *expr2*

Di sini *cond_expr* adalah ekspresi kondisional bertipe *bool* (kependekan dari *boolean,* dari *logika boolean*). Ingat bahwa semua angka bukan nol akan dikonversikan menjadi *true* dan angka nol menjadi *false*. Jika *cond_expr* bernilai benar / *true,* maka nilai hasil operasi adalah *expr1*, jika tidak maka *expr2* yang menjadi hasil ekspresi ini.

Contoh:

```
a =  (b>3) ?  999 : -999;
```

berarti memeriksa apakah b>3; jika benar, maka a akan diisi dengan 999, dan jika salah, maka a diisi dengan –999.

ASSIGNMENT OPERATOR

Bersintaks *lvalue assignment-op expr*

- *lvalue* (dari *left-value*) adalah ekspresi yang memiliki suatu lokasi di memori, dan isi *lvalue* ini harus bisa diubah (jadi bukan bertipe `const`,

yang tidak boleh diubah). Paling sederhana *lvalue* berupa nama variabel.

- *assignment-op* adalah *operator assignment*, untuk *simple assigment* berupa **'='**, untuk *compound assigment* : **++** (*unary increment*), **--** (*unary decrement*), ***=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=.** Pada *compound assignment*, efeknya akan sama dengan sintaks *lvalue* = *lvalue op expr,* dengan op adalah operator yang diasosiasikan dengan *assignment-op* tadi, jika ***=** maka op ialah *****, pada */=* op adalah */*, dst. Untuk **++** : hampir ekivalen dengan *lvalue* = *lvalue* **+ 1** dan untuk **--** hampir ekivalen dengan *lvalue* = *lvalue* **- 1.**
- *expr* adalah suatu nilai; untuk *simple assigment,* nilai ini akan diisikan pada *lvalue*, sedangkan untuk *compound assignment,* efeknya bervariasi untuk tiap jenis operator.
- Nilai ekspresi dengan *assigment operator* ialah nilai *lvalue* setelah operasi dilakukan.
- *Asosiasi dari kanan ke kiri*
- Kasus khusus pada *unary increment/decrement*: `lvalue++` disebut *post-increment.* Hasil ekspresi ini ialah lvalue sebelum ditambah dengan 1, sedangkan `++lvalue` disebut *pre-increment*, hasil ekspresi ini ialah *lvalue* setelah ditambah dengan 1.

Contoh penggunaan *assignment operator*:

seperti `'a = 2',`

berarti memasukkan angka '2' ke dalam 'a'. Nilai ekspresi seperti ini ialah nilai yang dimasukkan ke variabel tersebut, yaitu 2. Implikasi fakta ini ialah bisa ada *statement* `if` seperti: `'if ((x=a)= =2) break;'` yang berarti memasukkan a ke dalam x, lalu memeriksa apakah nilai a itu sama dengan 2, dan melakukan `break` jika benar.

Contoh lain:

`'a=b=0;'`

akan mengisi a dan b dengan nilai nol.

Tabel ***escape sequence***

`\n`	Baris baru
`\t`	tab horizontal
`\v`	tab vertikal
`\b`	backspace
`\r`	carriage return
`\f`	form feed (ganti halaman untuk printer)
`\a`	Bell/alert
`\'`	kutip tunggal
`\"`	kutip ganda
`\\`	backslash (\)
`\`*`ddd`*	karakter ASCII dlm notasi oktal
`\x`*`ddd`*	karakter ASCII dlm notasi heksadesimal

STATEMENT

- `if (expr)` statement1;
- `if (expr)` statement1; `else` statement2;
- `do`-statement:
  `do loop`-statement; `while (expr)`
- `while`-statement:
  `while (expr) loop`-statement;
- compound-statement:
  `{ declarations`; statement1; statement2; ...; `}`
- `switch`-statement:
  `switch (int-expr)`
  `{`
  `case int-const-expr1 :`
  statement1;
  statement2;

```
...
[break;]
case int-const-expr1:
statement3;
statement4;
...
[break;]
[default:
statement5;
...
]
}
```

- `goto` label;

DEKLARASI FUNGSI

```
type funcname(type, type, ... );
```

DEFINISI

```
type funcname(type param1, type param2, ...)
```

CONTOH PROGRAM

(Semua program menggunakan ANSI C harus berekstensi .c, sementara C++ berekstensi .cpp)

```
#include <stdio.h>
int main()
{
        printf("Hello,world!\nHow are you?");
        return 0;
}
```

'`\n`': *escape sequence* untuk *newline* (ganti baris)

'`main`' : fungsi utama

`'stdio.h'` : tempat adanya prototype untuk `printf dan scanf`

`'return 0'`: karena main adalah fungsi bertipe integer (`int` = bilangan bulat) maka standar C++ mengharuskan ada nilai kembali

`'printf'` : mengeluarkan output ke layar

'{' : awal blok

'}' : akhir blok

Program tadi jika dibuat menggunakan ***Standard C++ library:***

```
#include <iostream.h>
int main()
{
        cin << "Hello,world!" << endl << "How are you?";
}
```

'`endl`' : *end-of-line* / baris baru

```
#include <stdio.h>
int main()
{
        int a;
        a=5*5-3;
        printf("a = %i\n",a);
        return 0;
}
```

`‘%i’` : kode format untuk tipe int

`'5*5-3'` : suatu ekspresi = sesuatu yang bisa menghasilkan suatu nilai / angka

Presedensi (urutan) operasi : Jika didapati operasi yang tidak jelas menunjukkan mana yang harus dilakukan, ada suatu aturan yang membuat jelas urutan operasinya:

```
()
*/
+-
```

(makin atas makin didahulukan)

Jadi perkalian didahulukan atas penjumlahan

Dengan C++:

```
#include <iostream.h>
int main()
{
        int a;
        a=5*5-3;
        cout << "a = "<<a<<endl;
        return 0;
}

#include <stdio.h>
int main()
{
        int a;
        a=5*5-3;
        printf("a = %i\n",a);
        return 0;
}


#include <stdio.h>
float f(float x)
{
        float n=x*x-3;
        printf("menghitung f(x)…\n");
        return n;
}
int main()
{
        float a;
        a=f(5);
        printf("a = %i\n",a);
        return 0;
}

#include <stdio.h>
```

```
float f(float x)
{
        float n=x*x-3;
        printf("menghitung f(x)…\n");
        return n;
}
int main()
{
        float a;
        a=5.4;
        printf("f(a) = %f\n",f(a));
        return 0;
}
```

'%f' : kode format untuk tipe `float`

```
#include <stdio.h>
float f(float x)
{
        float n=x*x-3;
        printf("menghitung f(%f)…\n",x);
        return n;
}
int main()
{
        float a;
        printf("a=?");
        scanf("%f",&a);
        printf("f(%f)=%f\n",a,f(a));
        return 0;
}
```

```
#include <stdio.h>
int main()
{       int a;
        for (a=1; a<6; a++)
                printf("\t%i",a);
        printf("\n");
```

```
        return 0;
}

#include <stdio.h>
int main()
{       int a=3;
        while (a<9)
        {
                printf("\t%i",a);
                a++;
        }
        printf("\n");
        return 0;
}
```

'\t' : *escape sequence* untuk tabulasi (8 spasi)

```
#include <stdio.h>
int main()
{       int a;
        printf("masukkan nilai a:");
        scanf("%i",&a);
        if (a>30)
                printf("\na lebih besar dari 30\n");
        else
                printf("\na lebih kecil atau sama dengan 30\n");
        return 0;
}
```

'\t' : *escape sequence* untuk tabulasi (8 spasi)

'&' : *address-of operator* (alamat dari)

```
#include <stdio.h>
int main()
{
        double a,b,tmp;
        printf("Program mencari nilai faktor persekutuan terbesar\n");
        printf("Masukkan dua angka, a=?");
        scanf("%lf",&a);
        printf("\tb=?"); scanf("%lf",&b);
```

```
        while (1)
        {
                if (b>a)
                {       tmp=a; a=b; b=tmp; }
                if (b= =0)
                        break;
                a=a-b;
        }
        return 0;
}
```

‘`break`’ : menghentikan *loop* yang terdalam (karena bisa ada *loop* di dalam *loop*, yang dihentikan oleh `break` hanyalah *loop* terdalam/terakhir ditulis)

`'(1)'` : dalam C, ekspresi kondisional bertipe *bool*. Jika bukan tipe *bool*, akan dilakukan konversi. Semua bilangan bukan nol akan dikonversikan menjadi *true* (benar) dan bilangan nol akan dikonversikan menjadi *false* (salah). Karena di sini ekspresi-kondisi yang dipakai ialah '1', yang *notabene* dianggap logika benar, *loop* ini akan berulang terus-menerus (tentu saja dihentikan dengan `break`).