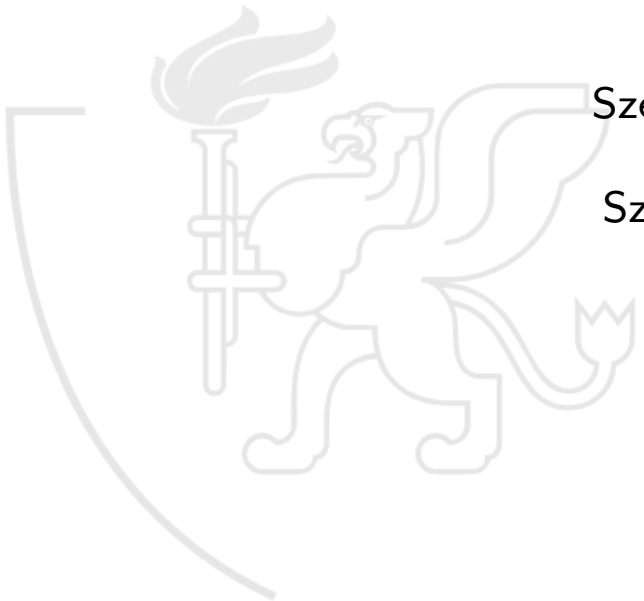
Programozás Alapjai

GCC figyelmeztetések és hibák

Dr. Gergely Tamás
Dr. Jász Judit

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék



2021

(v0901)

Tartalom

Tartalom

gcc hibaüzenetek

warning, error

- A kurzus egyik kimondott célja, hogy a hallgatóknak *ne jelentsen gondot az alapvető C fordítási hibaüzenetek értelmezése, és ez alapján a hiba megtalálása, javítása*. Ennek eléréséhez rengeteg gyakorlás, és sok elrontott program szükséges.
- A gcc „hibaüzeneteinek" fajtái:
  - Figyelmeztetések (warnings): nem akadályozzák meg a program lefordítását. Olyan pontokra hívják fel a figyelmet, amik miatt esetleg „hibásan", az elvárttól eltérően működhet a program.
  - Fordítási hibák (compile errors): megakadályozzák a program lefordítását. Olyan szintaktikailag vagy szemantikailag hibás pontok, amelyekről a fordító nem tudja eldönteni, hogy a programozó mit is szeretne pontosan.
  - Szerkesztési hibák (linker errors): megakadályozzák a program összerakását. A C forrás lefordul (az object fájl előáll), de nincs meg minden információ ahhoz, hogy ebből össze lehessen rakni a futtatható programot.

gcc hibaüzenetek

warning, error

- A gcc és a legtöbb C fordító manapság egyre bővebb és pontosabb hibaüzeneteket ad, néha még a javítás (valószínű) módját is közli. Éppen ezért hasznos lehet időt szánni a fordító hibaüzeneteinek átolvasására, megismerésére, megértésére.
- A következő példák a `gcc (Ubuntu 9.3.0-17ubuntu1 20.04) 9.3.0` fordítóval készültek.

Tartalom

Figyelmeztetések (warning)

- A figyelmeztetések nem akadályozzák meg a program lefordítását. Olyan pontokra hívják fel a figyelmet, amik miatt esetleg „hibásan”, az elvárttól eltérően működhet a program. Ezek lehetnek a C nyelv engedékenységéből adódó esetek, de van elgépelésből adódó gyakori hibát jelző figyelmeztetés is.
- Példák:
  - Hiányzó visszatérési érték
  - Hiányzó include
  - Implicit függvénydeklaráció
  - Változó deklaráció típus nélkül
  - Függvény visszatérési típus nélkül
  - Formátum sztring hiba
  - Inicializálatlan változó használata
  - Utasítás, aminek nincs hatása a program futására
  - Használjunk zárójelezést az egyértelműségért

Hiányzó visszatérési érték

-Wall, -Wreturn-type

- `noret.c`:

```c
#include <stdio.h>

int pozitiv_e(int i) {
    if (0 < i)
        return 1;
}

int main() {
    int i;
    scanf ("%d", &i);
    printf("%d␣%spozitív\n", i, pozitiv_e(i) ? "" : "nem␣");
    return 0;
}
```

- *Hibaüzenet*:

```
noret.c: In function ‘pozitiv_e’:
noret.c:6:1: warning: control reaches end of non-void function [-Wreturn-type]
    6 | }
      | ^
```

Hiányzó visszatérési érték

-Wall, -Wreturn-type

- *Hiba:* nem void visszatérésű függvényben van olyan végrehajtási útvonal, amelyiken nem adjuk meg a visszatérési értéket (azaz nincs `return` utasítás).
  - A hibajelzés a függvény végére szól.
- *Következmény:* a hívás helyén egy nem definiált értéket használunk fel.
  - `pozitiv_e(i)` nem pozitív számok esetében nem definiált.
- *Javítás:* minden útvonalon adjunk meg visszatérési értéket.
  - Vagy a függvény végén egy alapértelmezettet.

```
int pozitiv_e(int i) {
    if (0 < i)
        return 1;
    return 0; // ha nem pozitív, akkor 0-val tér vissza
}
```

Hiányzó include

- `noinc.c`:

```c
int main() {
    printf("Helló Világ!\n");
    return 0;
}
```

- *Hibaüzenet*:

```
noinc.c: In function ‘main’:
noinc.c:2:5: warning: implicit declaration of function ‘printf’ [-Wimplicit-func»
»tion-declaration]
    2 |     printf("Helló Világ!\n");
      |     ^~~~~~
noinc.c:2:5: warning: incompatible implicit declaration of built-in function ‘pr»
»intf’
noinc.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | int main() {
```

Hiányzó include

- *Hiba:* Olyan gyakori, általános függvényt használunk, amihez szükség lenne valamelyik standard header include-olására.
  - A gcc felismer bizonyos „beépített” (built-in) függvényeket, így ezek deklarációjának hiánya nem hibát, csak figyelmeztetést okoz.
- *Következmény:* Ha helyes argumantumokkal használjuk a standard függvényeket, akkor a hibaüzeneteken túl nincs következnénye.
- *Javítás:* include-oljuk be a megfelelő header fájlt.

```c
#include <stdio.h> // javítva

int main() {
    printf("Helló␣Világ!\n");
    return 0;
}
```

Implicit függvénydeklaráció - warning

-Wall, -Wimplicit-function-declaration

- `impldecl.c`:

```c
#include <stdio.h>

int main() {
    int n = 5;
    printf("A Fibonacci-sorozat %d. eleme:\t%d", n, fib(n));
    return 0;
}

int fib(int n) {
    if (n == 1 || n == 2)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

- *Hibaüzenet*:

```
impldecl.c: In function ‘main’:
impldecl.c:5:53: warning: implicit declaration of function ‘fib’ [-Wimplicit-fun»
»ction-declaration]
    5 |     printf("A Fibonacci-sorozat %d. eleme:\t%d", n, fib(n));
      |                                                     ^~~
```

Implicit függvénydeklaráció - warning

-Wall, -Wimplicit-function-declaration

- *Hiba:* Olyan függvényt használunk, amit csak később (vagy később sem) deklarálunk.
  - Ha a definíció is elmarad, az már szerkesztési hibát (is) okoz.
- *Következmény:* A fordító ilyenkor a konkrét hívás alapján „megelőlegezi” a függvény deklarációját.
  - Ezek után a függvényt tudjuk használni.
  - Ha a későbbi deklaráció nem egyezik meg a megelőlegezettel, akkor már fordítási hibát (is) kapunk.
- *Javítás:* Deklaráljuk a függvényt.
  - Ezt akár egy megfelelő header fájl include-olásával is megtehetjük.

```c
#include <stdio.h>

int fib(int n); // függvénydeklaráció

int main() {
    // ...
}

int fib(int n) {
    // ...
}
```

Változó deklaráció típus nélkül

-Wall, -Wimplicit-int

- `impltype.c`:

```c
#include <stdio.h>

i;

int main() {
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

- *Hibaüzenet*:

```
impltype.c:3:1: warning: data definition has no type or storage class
    3 | i;
      | ^
impltype.c:3:1: warning: type defaults to ‘int’ in declaration of ‘i’ [-Wimplici»
»t-int]
```

Változó deklaráció típus nélkül

-Wall, -Wimplicit-int

- *Hiba:* Globális változót típus megadása nélkül deklarálok.
- *Következmény:* A fordító ilyenkor `int` típussal hozza létre a változót.
- *Javítás:* Változódeklarációnál mindig explicite írjuk ki a típust!
  - Még ha `int`-et szeretnénk használni akkor is!

```c
#include <stdio.h>

int i; // javítva, megadtuk az 'i' típusát

int main() {
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

Függvény visszatérési típus nélkül

-Wall, -Wreturn-type

- `implret.c`:

```c
#include <stdio.h>

max(int a, int b) {
    return a > b ? a : b;
}

int main() {
    int i, j;
    scanf("%d␣%d", &i, &j);
    printf("max(%d,␣%d)␣=␣%d\n", i, j, max(i,j));
    return 0;
}
```

- *Hibaüzenet*:

```
implret.c:3:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
    3 | max(int a, int b) {
      | ^~~
```

Függvény visszatérési típus nélkül

-Wall, -Wreturn-type

- *Hiba:* A függvénynek nem adtuk meg a visszatérési típusát.
- *Következmény:* A fordító ilyenkor `int` típusúnak tekinti.
  - Ha másik típussal akarunk visszatérni, akkor gond lehet, például a függvényben visszaadott `long long` érték `int` méretűre lesz konvertálva.
- *Javítás:* Mindig írjuk ki a függvény visszatérési típusát!
  - Még ha `int`-et szeretnénk használni akkor is!

```c
#include <stdio.h>

int max(int a, int b) { // javítva, megadtuk a visszatérés típusát
    return a > b ? a : b;
}

int main() {
    int i, j;
    scanf("%d %d", &i, &j);
    printf("max(%d, %d) = %d\n", i, j, max(i,j));
    return 0;
}
```

Formátum sztring hiba

-Wall, -Wformat, -Wformat-extra-args

- `format.c`:

```c
#include <stdio.h>

int main() {
    double d = 0;
    scanf("%lf", d);
    printf("A beolvasott szám: %lf\n");
    return 0;
}
```

- *Hibaüzenet*:

```
format.c: In function ‘main’:
format.c:5:14: warning: format ‘%lf’ expects argument of type ‘double *’, but ar»
»gument 2 has type ‘double’ [-Wformat=]
    5 |     scanf("%lf", d);
      |            ~~^   ~
      |              |   |
      |              |   double
      |              double *
format.c:6:35: warning: format ‘%lf’ expects a matching ‘double’ argument [-Wfor»
»mat=]
    6 |     printf("A beolvasott szám: %lf\n");
      |                                 ~~^
      |                                   |
      |                                   double
```

Formátum sztring hiba

-Wall, -Wformat, -Wformat-extra-args

- *Hiba:* `scanf`/`printf`/`fscanf`/`fprintf` paraméterezése nem felel meg a formátum sztringben megadott paraméterezésnek.
- *Következmény:* A konkrét hibától függően sokféle lehet, például:
  - Az adott érték helyett valami értelmetlen jelenik meg (pl. `printf` esetén valós érték kiíratása egészként, vagy az argumentum elmaradása).
  - Szegmentálási hibával megáll a program futása (pl. `scanf` esetén a `&` operátor vagy a teljes argumentum elmaradása).
- *Javítás:* Megfelelő számú és típusú paramétert adjunk meg!

```c
#include <stdio.h>

int main() {
    double d;
    scanf("\%lf", &d); // & jel oda lett írva
    printf("A␣beolvasott␣szám:␣%lf\n", d); // megadtuk a 'd'-t
    return 0;
}
```

Inicializálatlan változó használata

-Wall, -Wuninitialized

- `uninitialized.c`:

```c
#include <stdio.h>

int main() {
    int t;
    printf("%d\n", t);
    return 0;
}
```

- *Hibaüzenet*:

```
uninitialized.c: In function ‘main’:
uninitialized.c:5:5: warning: ‘t’ is used uninitialized in this function [-Wunin»
»itialized]
    5 |     printf("%d\n", t);
      |     ^~~~~~~~~~~~~~~~~
```

Inicializálatlan változó használata

-Wall, -Wuninitialized

- *Hiba:* Olyan változót használunk, aminek előtte nem adtunk értéket.
- *Következmény:* A változó értéke definiálatlan, ami gyakorlatilag egy véletlenszerű értéket jelent.
  - Lehetnek olyan helyzetek, amikor a program így is jól működik, de ez a véletlen műve.
- *Javítás:* Inicializálással vagy értékadással.

```c
#include <stdio.h>

int main() {
    int t = 0;
    printf("%d\n", t);
    return 0;
}
```

Utasítás, aminek nincs hatása a program futására

-Wall, -Wunused-value

- `noeffect.c`:

```c
int main() {
    int t[100], i;
    for (i = 0; i < 100; i++)
        t[i] == i;
    return 0;
}
```

- *Hibaüzenet*:

```
noeffect.c: In function ‘main’:
noeffect.c:4:14: warning: statement with no effect [-Wunused-value]
    4 |         t[i] == i;
      |         ~~~~~^~~~
noeffect.c:4:10: warning: ‘t[i]’ may be used uninitialized in this function [-Wm»
»aybe-uninitialized]
    4 |         t[i] == i;
      |         ~^~~
```

Utasítás, aminek nincs hatása a program futására

-Wall, -Wunused-value

- *Hiba:* Olyan utasítást írtunk, aminek az eredménye nincs hatással a program futására.
  - Pl. `a t[i] == i` kiértékelődik, de az érték nincs felhasználva.
- *Következmény:* A hiba jellegétől függ.
  - Önmagában a plusz utasítás kevésbé megérthetővé teszi a programot, de hatása nincs.
  - Ugyanakkor sokkal valószínűbb, hogy nem egy haszontalan utasítást akartunk leírni, hanem elgépeltünk egy egyébként hasznosat (pl. `a = 5;` helyett `a == 5;`), így a program nem az elvárásainknak megfelelően fog működni.
- *Javítás:* Értelemszerűen (nincs általános szabály).

```
int main() {
    int t[100], i;
    for (i = 0; i < 100; i++)
        t[i] = i; // == helyett = kell
    return 0;
}
```

Használjunk zárójelezést az értékedás egyértelműsítésére

-Wall, -Wparentheses

- `useparenth.c`:

```c
#include <stdio.h>

int main() {
    int i, j;
    scanf("%d␣%d", &i, &j);
    if (i = j)
        printf("Egyenlők\n");
    return 0;
}
```

- *Hibaüzenet*:

```
useparenth.c: In function ‘main’:
useparenth.c:6:9: warning: suggest parentheses around assignment used as truth v»
»alue [-Wparentheses]
    6 |     if (i = j)
      |         ^
```

Használjunk zárójelezést az értékadás egyértelműsítésére

-Wall, -Wparentheses

- *Hiba:* Nem feltétlenül hiba.
  - Pl. az `if (f=fopen("be.txt", "r")) { /* beolvas */ }` kódrészlet teljes mértékben helyes.
  - De egy ilyen konstrukció sok esetben elírás eredménye, ami hibához vezet, pl. `if (i == j)` helyett `if (i = j)` vagy `if (a < b || c < d)` helyett `if (a < b | c < d)` esetén.
- *Következmény:* Változatos, de ha elírás, akkor szinte biztosan hibához vezet.
  - Pl. `i == j` akkor igaz, ha `i` és `j` egyenlő míg `i = j` akkor igaz, ha `j` nem nulla, és mellesleg `i` értéke megváltozik.
- *Javítás:* Értelemszerűen javítsuk, vagy ha szándékosan alkalmazzuk, akkor tegyünk zárójelbe!

```
if (i == j) // javítás: = helyett ==
    printf("Egyenlők\n");
```

Tartalom

Fordítási hibák (compile error)

- A fordítási hibák megakadályozzák a program lefordítását. Olyan szintaktikailag vagy szemantikailag hibás pontok, amelyekről a fordító nem tudja eldönteni, hogy a programozó mit is szeretne pontosan. Ez adódhat szimpla elgépelésből, programrészek kihagyásából (vagy éppen ismétléséből), de helytelen műveletek megadásából is.
- Példák:
  - Nem deklarált változó
  - Változó újradeklarálása
  - Lokális változó típus nélkül
  - (Implicit) függvény újradeklarálása „máshogyan”
  - Hiányzó }
  - Hiányzó pontosvessző
  - Túl sok bezáró }

Nem deklarált változó

Fordítási hiba

- `undeclvar.c`:

```c
#include <stdio.h>

int main() {
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

- *Hibaüzenet*:

```
undeclvar.c: In function ‘main’:
undeclvar.c:4:18: error: ‘i’ undeclared (first use in this function)
    4 |     scanf("%d", &i);
      |                  ^
undeclvar.c:4:18: note: each undeclared identifier is reported only once for eac»
»h function it appears in
```

Nem deklarált változó

Fordítási hiba

- *Hiba:* Olyan változót használunk, amit nem deklaráltunk korábban.
  - A fordító pontosan megmondja, hogy melyik változóról van szó.
  - Ha lokális változó deklarációjánál lehagyjuk a típust, akkor is ezt a hibaüzenetet kapjuk

```
int main() {
    i; // nincs kiírva a típus, ez nem deklaráció!!!
    scanf("%d", &i);
```

- *Következmény:* Fordítási hiba.
- *Javítás:* Deklaráljuk a változót!

```
#include <stdio.h>

int main() {
    int i; // javítva
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

Változó újradeklarálása

Fordítási hiba

- `redeclvar.c`:

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

int main() {
    double d;
    scanf("%lf", &d);
    double d = sin(d);
    printf("A szám sinus értéke: %lf\n", d);
    return 0;
}
```

- *Hibaüzenet*:

```
redeclvar.c: In function ‘main’:
redeclvar.c:7:12: error: redeclaration of ‘d’ with no linkage
    7 |     double d = sin(d);
      |            ^
redeclvar.c:5:12: note: previous declaration of ‘d’ was here
    5 |     double d;
      |            ^
```

Változó újradeklarálása

Fordítási hiba

- *Hiba:* Egy már deklarált változót újradeklarálunk ugyanabban a blokkban.
- *Következmény:* Fordítási hiba.
- *Javítás:*
  - Ha ugyanarról a változóról van szó, akkor töröljük a második deklarációt.
  - Ha új változót akartunk bevezetni, akkor nevezzük át bevezetett változót a deklarációnál és a használatoknál.

```
int main() {
    double d;
    scanf("%lf", d);
    double r = sin(d); // javitas: d helyett r-t használunk
    printf("A szám sinus értéke: %lf\n", r);
    return 0;
}
```

(Implicit) függvény újradeklarálása „máshogyan”

Fordítási hiba

- `impldeclredecl.c`:

```c
int main() {
    double d = my_sin(10);
    d /= 2.0;
    return 0;
}

double my_sin(double d) {
    return (d < -1.0 || 1.0 < d) ? 1.0 / d : d;
}
```

- *Hibaüzenet*:

```
impldeclredecl.c: In function ‘main’:
impldeclredecl.c:2:16: warning: implicit declaration of function ‘my_sin’ [-Wimp»
»licit-function-declaration]
    2 |     double d = my_sin(10);
      |                ^~~~~~
impldeclredecl.c: At top level:
impldeclredecl.c:7:8: error: conflicting types for ‘my_sin’
    7 | double my_sin(double d) {
      |        ^~~~~~
impldeclredecl.c:2:16: note: previous implicit declaration of ‘my_sin’ was here
    2 |     double d = my_sin(10);
      |                ^~~~~~
```

(Implicit) függvény újradeklarálása „máshogyan”

Fordítási hiba

- *Hiba:* Ugyanazon függvényhez többféle (implicit vagy explicit) deklaráció tartozik.
  - Egy függvényt már deklaráltunk valamilyen paraméterekkel és visszatérési értékkel, majd ugyanezt a függvény máshogyan akarjuk ismét „bevezetni”.
    - A korábbi deklaráció lehet explicit (azaz kiírjuk) vagy implicit (a fordító „kitalálta”) is.
- *Következmény:* Fordítási hiba.
- *Javítás:* Egy függvényt csak egyféle paraméterezéssel és visszatérési értékkel deklaráljunk.

```
double my_sin(double d); // deklarálom az első használat előtt

int main() {
    double d = my_sin(10);
    return 0;
}

double my_sin(double d) {
    return (d < -1.0 || 1.0 < d) ? 1.0 / d : d;
}
```

Hiányzó }

Fordítási hiba

- `missing-brace.c`:

```c
#include <stdio.h>

void kiir(int i) {
    printf("A szám: %d\n", i);

int main() {
    int i;
    scanf("%d", &i);
    kiir(i);
    return 0;
}
```

- *Hibaüzenet*:

```
missing-brace.c: In function ‘kiir’:
missing-brace.c:6:5: warning: ‘main’ is normally a non-static function [-Wmain]
    6 | int main() {
      |     ^~~~
missing-brace.c:11:1: error: expected declaration or statement at end of input
   11 | }
      | ^
At top level:
missing-brace.c:6:5: warning: ‘main’ defined but not used [-Wunused-function]
    6 | int main() {
      |     ^~~~
```

Hiányzó }

Fordítási hiba

- *Hiba:* Nem zártunk le egy blokkot.
  - Több nyitó { van, mint záró }.
- *Következmény:* Fordítási hiba.
  - A hibaüzenet nem feltétlenül annak a függvénynek vagy blokknak a végére mutat, amelyik nem lett lezárva, hanem a fordítóprogram a C szintaxisa szerint tovább értelmezi a programot, és csak ahol elakad, ott jelez hibát.
- *Javítás:* Keressük meg a nem lezárt függvényt/blokkot, és zárjuk le.

```c
#include <stdio.h>

void kiir(int i) {
    printf("A␣szám:␣%d\n", i);
} // javítva,

int main() {
    ...
}
```

Hiányzó pontosvessző

Fordítási hiba

- `missing-semicolon.c`:

```c
#include <stdio.h>

int main() {
    int i;
    scanf("%d", &i)
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

- *Hibaüzenet*:

```
missing-semicolon.c: In function ‘main’:
missing-semicolon.c:5:20: error: expected ‘;’ before ‘printf’
    5 |     scanf("%d", &i)
      |                    ^
      |                    ;
    6 |     printf("A beolvasott szám: %d\n", i);
      |     ~~~~~~
```

Hiányzó pontosvessző

Fordítási hiba

- *Hiba:* Hiányzik a pontosvessző egy utasítás vagy deklaráció végéről.
  - A fordító elég pontosan megmondja a hiba helyét, bár inkább a „Mi előtt hiányzik?” mint a „Mi után hiányzik?” kérdésre válaszolva.
- *Következmény:* Fordítási hiba.
- *Javítás:* Pótoljuk a hiányzó pontosvesszőt.

```c
#include <stdio.h>

int main() {
    int i;
    scanf("%d", &i); // javítva
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

Túl sok bezáró }

Fordítási hiba

- `extra-brace.c`:

```c
#include <stdio.h>

int main() {
    int i;
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}}
```

- *Hibaüzenet*:

```
extra-brace.c:8:2: error: expected identifier or ‘(’ before ‘}’ token
    8 | }}
      |  ^
```

Túl sok bezáró }

Fordítási hiba

- *Hiba:* Több záró } van, mint nyitó {.
- *Következmény:* Fordítási hiba
  - A fordítóprogram a szerinte felesleges } helyét mutatja.
- *Javítás:* Töröljük a felesleges } jelet, vagy pótoljuk a hiányzó {-t.

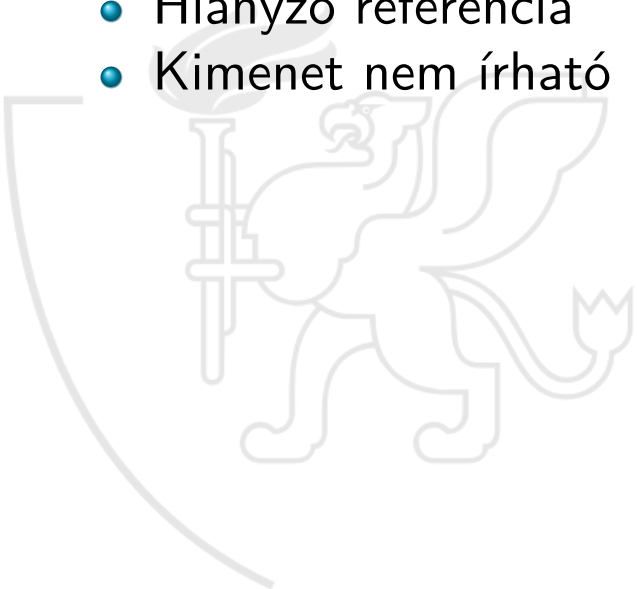
```c
#include <stdio.h>

int main() {
    int i;
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
} // javítva, a felesleges { törölve
```

Tartalom

Szerkesztési hibák (linker error)

- A szerkesztési hibák nem akadályozzák a fordítást, de a program összerakását igen. Akkor kapjuk őket a linkertől (`ld`), amikor nincs meg minden információ ahhoz, hogy össze lehessen állítani a futtatható programot. Jellemzően függvények megvalósítása és `extern`-ként deklarált változóknak való helyfoglalás szokott hiányozni.
- Példák:
  - Hiányzik a `main` függvény
  - Hiányzó referencia
  - Kimenet nem írható



Hiányzik a `main` függvény

Szerkesztési hiba

- `nomain.c`:

```c
#include <stdio.h>

int Main() {
    int i;
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

- *Hibaüzenet*:

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o: »
»in function ‘_start’:
(.text+0x24): undefined reference to ‘main’
collect2: error: ld returned 1 exit status
```

Hiányzik a `main` függvény

Szerkesztési hiba

- *Hiba:* Nincs megadva a program belépési pontja, a `main` függvény.
- *Következmény:* Szerkesztési hiba, nem lehet futtatható programot készíteni.
- *Javítás:*
  - Ha futtatható programról van szó, akkor adjunk meg `main` függvényt (case sensitive)!
  - Ha csak egy modult készítünk, akkor adjuk meg a gcc-nek a `-c` kapcsolót, hogy csak fordítson, ne szerkesszen.

```c
#include <stdio.h>

int main() { // javítva, Main helyett main
    int i;
    scanf("%d", &i);
    printf("A beolvasott szám: %d\n", i);
    return 0;
}
```

Hiányzó referencia

Szerkesztési hiba

- `undefref.c`:

```c
#include <stdio.h>

int Fibb(int);

int main() {
    int i;
    scanf("%d", &i);
    printf("Az %d-edik Fibonacci szám: %d\n", i, Fibb(i));
    return 0;
}
```

- *Hibaüzenet*:

```
/usr/bin/ld: /tmp/ccsMci4F.o: in function ‘main’:
undefref.c:(.text+0x39): undefined reference to ‘Fibb’
collect2: error: ld returned 1 exit status
```

Hiányzó referencia

Szerkesztési hiba

- *Hiba:* A szerkesztő egy olyan hivatkozást talál, amihez nincs meg a definíció.
  - Általában egy függvény megvalósítása hiányzik, de lehet pl. `extern` változó is.
- *Következmény:* Szerkesztési hiba, a program nem készül el.
- *Javítás:* A hiba okától függően többféle lehet:
  - Adjuk meg azt a forrás vagy object fájlt is, amiben a függvény meg van valósítva vagy a változónak hely van foglalva!
  - Mondjuk meg a linkernek, hogy milyen függvénykönyvtárban keresse a megvalósítást (pl. matematikai függvények esetén a `libm.a` átnézésére a `-lm` kapcsolóval utasíthatjuk)!
  - Ha máshol nincs megvalósítva a függvény, akkor adjuk meg a definícióját!

```
int Fibb(int);

int main() { ... }

// javítás: megvalósítjuk a Fibb függvényt
int Fibb(int i) { ... }
```

Kimenet nem írható

Általános hiba

- `feladat.c`:

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

- *Hibaüzenet*:

```
/usr/bin/ld: cannot open output file feladat: Is a directory
collect2: error: ld returned 1 exit status
```

Kimenet nem írható

Általános hiba

- *Hiba:* A fordító vagy szerkesztő nem tudta elmenteni a kimeneti fájlt.
  - Nem létező útvonalat, létező könyvtárat vagy létező de általunk nem módosítható fájlt adtunk meg.
  - Egylépéses fordítás esetén a hiba a végső kimenetet előállító szerkesztőnél (linkernél) jelentkezik.
- *Következmény:* Nem jön létre a kimeneti fájl.
- *Javítás:* Olyan kimenetet adjunk meg, amire van fájlként írási/létrehozási jogunk.
  - Ellenőrizzük az útvonalat, a jogosultságainkat.
- Megjegyzés: ZH-k alatt a biztonságos home könyvtárban lehet egy `feladat` könyvtár, ami a `root` tulajdonában van, ezt a használt user nem fogja tudni törölni/felülírni, így a `gcc feladat.c -o feladat` parancs ezzel a hibával fog visszatérni, ha a biztonságos home könyvtárban adjuk ki.