

# Programozás Alapjai

## GCC figyelmeztetések és hibák

Dr. Gergely Tamás  
Dr. Jász Judit

Szegedi Tudományegyetem  
Informatikai Intézet  
Szoftverfejlesztés Tanszék

2023

(v0914)



1

GCC hibaüzenetek

- Bevezető
- Figyelmeztetések

- Fordítási hibák
- Szerkesztési hibák



1

GCC hibaüzenetek

- Bevezető
- Figyelmeztetések

- Fordítási hibák
- Szerkesztési hibák



- 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 11.2.0–19ubuntu1) 11.2.0 fordítóval készültek.



1

GCC hibaüzenetek

● Bevezető  
● **Figyelmeztetések**

● Fordítási hibák  
● Szerkesztési hibák



# 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 engedékenységből 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
  - Nem használt változó
  - Előjeles és előjeltelen érték összehasonlítása
  - Utasítás, aminek nincs hatása a program futására
  - Használjunk zárójelezést az egyértelműségért
  - Megtévesztő indentálás

# Hiányzó visszatérési érték

-Wall, -Wreturn-type

- `noret.c`:

```
1#include <stdio.h>
2
3int pozitiv_e(int i) {
4    if (0 < i)
5        return 1;
6}
7
8int main() {
9    int i;
10   scanf ("%d", &i);
11   printf ("%d_%spositiv\n", i, pozitiv_e(i) ? "" : "nem_");
12   return 0;
13}
```

- *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

-Wall, -Wimplicit-function-declaration

- `noinc.c`:

```
1 int main() {
2     printf("Helló Világ!\n");
3     return 0;
4 }
```

- *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:1:1: note: include '<stdio.h>' or provide a declaration of 'printf'
+++ |+#include <stdio.h>
   1 | int main() {
noinc.c:2:5: warning: incompatible implicit declaration of built-in function 'pr
>intf' [-Wbuiltin-declaration-mismatch]
   2 |     printf("Helló Világ!\n");
     |     ~~~~~
noinc.c:2:5: note: include '<stdio.h>' or provide a declaration of 'printf'
```

# Hiányzó include

-Wall, -Wimplicit-function-declaration

- *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 argumentumokkal használjuk a standard függvényeket, akkor a hibaüzeneteken túl nincs következménye.
- *Javítás*: include-oljuk be a megfelelő header fájlt.

```
#include <stdio.h> // javítva

int main() {
    printf("Helló_Világ!\n");
    return 0;
}
```

# Implicit függvénydeklaráció - warning

-Wall, -Wimplicit-function-declaration

- `impldecl.c`:

```
1#include <stdio.h>
2
3int main() {
4    int n = 5;
5    printf("A Fibonacci-sorozat %d. eleme:\t%d", n, fib(n));
6    return 0;
7}
8
9int fib(int n) {
10    if (n == 1 || n == 2)
11        return 1;
12    else
13        return fib(n - 1) + fib(n - 2);
14}
```

- *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.

```
#include <stdio.h>

int fib(int n); // függvénydeklaráció

int main() {
    // ...
}

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

# Változó deklaráció típus nélkül

-Wall, -Wimplicit-int

- `impltype.c`:

```
1#include <stdio.h>
2
3i;
4
5int main() {
6    scanf("%d", &i);
7    printf("A beolvasott szám: %d\n", i);
8    return 0;
9}
```

- *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' [-Wimplicit-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 expliciten írjuk ki a típust!
  - Még ha `int`-et szeretnénk használni akkor is!

```
#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`:

```
1#include <stdio.h>
2
3max(int a, int b) {
4    return a > b ? a : b;
5}
6
7int main() {
8    int i, j;
9    scanf("%d%d", &i, &j);
10   printf("max(%d,%d)=%d\n", i, j, max(i,j));
11   return 0;
12}
```

- *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!

```
#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`:

```
1#include <stdio.h>
2
3int main() {
4    double d = 0;
5    scanf("%lf", d);
6    printf("A beolvasott szám: %lf\n");
7    return 0;
8}
```

- *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:6:34: 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 kiírása 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!

```
#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:

```
1#include <stdio.h>
2
3int main() {
4    int t;
5    printf("%d\n", t);
6    return 0;
7}
```

- *Hibaüzenet:*

```
uninitialized.c: In function 'main':
uninitialized.c:5:5: warning: 't' is used uninitialized [-Wuninitialized]
  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.

```
#include <stdio.h>

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



# Nem használt változó

-Wall, -Wunused-variable

- `unused.c`:

```
1#include <stdio.h>
2
3int main() {
4    double a, b, c, s;
5    scanf("%lf%lf%lf", &a, &b, &c);
6    printf("T2=%lf\n", (a+b+c) * (-a+b+c) * (a-b+c) * (a+b-c) / 16.0);
7    return 0;
8}
```

- *Hibaüzenet:*

```
unused.c: In function 'main':
unused.c:4:21: warning: unused variable 's' [-Wunused-variable]
 4 |     double a, b, c, s;
  |                       ^
```

# Nem használt változó

-Wall, -Wunused-variable

- *Hiba*: Nem használunk egy deklarált változót.
- *Következmény*: Attól függ, hogyan került oda a deklaráció?
  - Ha csak ottfelejtettük de már nem kell, akkor „csak” a megértést rontja.
  - Ha okkal deklaráltuk, akkor valami hiányzik a programból.
- *Javítás*: Deklaráció törlésével vagy a változó használatával.

```
#include <stdio.h>

int main() {
    double a, b, c; // s törölve lett
    scanf("%lf%lf%lf", &a, &b, &c);
    printf("T^2_□=□%lf\n", (a+b+c) * (-a+b+c) * (a-b+c) * (a+b-c) / 16.0);
    return 0;
}
```

# Előjeles és előjeltelen érték összehasonlítása

-Wextra, -Wsign-compare

- `signcompare.c`:

```
1#include <stdio.h>
2
3int main() {
4    signed int s;
5    unsigned int u;
6    scanf("%d%u", &s, &u);
7    if (s > u) {
8        printf("%d>%u\n", s, u);
9    }
10}
```

- *Hibaüzenet:*

```
signcompare.c: In function 'main':
signcompare.c:7:11: warning: comparison of integer expressions of different sign»
»edness: 'int' and 'unsigned int' [-Wsign-compare]
   7 |         if (s > u) {
     |             ^
```



# Előjeles és előjeltelen érték összehasonlítása

-Wextra, -Wsign-compare

- *Hiba*: Előjeles és előjeltelen típusú értékeket hasonlítottunk össze.
- *Következmény*: Furcsa eredmény, pl.  $-1 > 1$ 
  - Ilyenkor az előjeles érték előjeltelenné konvertálódik, ami az alulcsordulás miatt nagyobb pozitív érték lehet, mint amihez hasonlítottuk.
- *Javítás*: Használjunk azonosan előjeles vagy előjeltelen típusokat, vagy összehasonlítás előtt konvertáljuk mindkét értéket „bővebb” típusúra.

```
#include <stdio.h>

int main() {
    signed int s;
    unsigned int u;
    scanf("%d%u", &s, &u);
    if ((long long int)s > (long long int)u) { // bővebb típusra konvertáltunk
        printf("%d>%u\n", s, u);
    }
}
```

# Utasítás, aminek nincs hatása a program futására

-Wall, -Wunused-value

- `noeffect.c`:

```
1 int main() {
2     int t[100], i;
3     for (i = 0; i < 100; i++)
4         t[i] == i;
5     return 0;
6 }
```

- *Hibaüzenet:*

```
noeffect.c: In function 'main':
noeffect.c:4:14: warning: statement with no effect [-Wunused-value]
  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 elgépettünk 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 értékadás egyértelműsítésére

-Wall, -Wparentheses

- useparenth.c:

```
1#include <stdio.h>
2
3int main() {
4    int i, j;
5    scanf("%d %d", &i, &j);
6    if (i = j)
7        printf("Egyenlők\n");
8    return 0;
9}
```

- *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 melleleg `i` értéke megváltozik.
- *Javítás*: Értelemszerűen javítsuk, vagy ha szándékosan alkalmazzuk, akkor tegyük zárójelbe!

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

# Megtévesztő indentálás

-Wall, -Wmisleading-indentation

- indentation.c:

```
1#include <stdio.h>
2
3int main() {
4    int i, sum = 0;
5    for (i = 0; i < 20; ++i)
6        sum += i;
7        printf("%d: %d\n", i, sum);
8    return 0;
9}
```

- *Hibaüzenet:*

```
indentation.c: In function 'main':
indentation.c:5:5: warning: this 'for' clause does not guard... [-Wmisleading-indentation]
   5 |     for (i = 0; i < 20; ++i)
     |     ~~~
indentation.c:7:9: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the 'for'
   7 |         printf("%d: %d\n", i, sum);
     |         ~~~~~
```

# Megtévesztő indentálás

-Wall, -Wmisleading-indentation

- *Hiba*: A rossz indentálás miatt úgy látszik, hogy valami egy blokkban van, pedig nincs.
- *Következmény*: A hiba jellegétől függ.
  - Önmagában a rossz indentálás kevésbé megérthetővé teszi a programot egy ember számára, de a fordítót ez nem zavarja.
  - Ugyanakkor sokkal valószínűbb, elfelejtettük kirakni a { és } jeleket a ciklusmag / feltételes blokk köré.
- *Javítás*: Megfelelő indentálással vagy a blokkhatárolók kihelyezésével.

```
#include <stdio.h>

int main() {
    int i, sum = 0;
    for (i = 0; i < 20; ++i) { // kiraktuk a blokkezdő {-t
        sum += i;
        printf("%d:_%d\n", i, sum);
    } // kiraktuk a blokkzáró }-t
    return 0;
}
```

1

GCC hibaüzenetek

- Bevezető
- Figyelmeztetések

- **Fordítási hibák**
- Szerkesztési hibák





# 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:

```
1#include <stdio.h>
2
3int main() {
4    scanf("%d", &i);
5    printf("A beolvasott szám: %d\n", i);
6    return 0;
7}
```

- 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 each
>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 hagyjuk 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`:

```
1#include <stdio.h>
2#include <math.h>
3
4int main() {
5    double d;
6    scanf("%lf", &d);
7    double d = sin(d);
8    printf("A szám sinus értéke: %lf\n", d);
9    return 0;
10}
```

- *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' with type 'double'
 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); // javítás: d helyett r-t használunk  
    printf("A szám sinus értéke: %lf\n", r);  
    return 0;  
}
```

# (Implicit) függvény újradeklarálása „máshogyan”

## Fordítási hiba

- `impldeclredecl.c`:

```
1 int main() {
2     double d = my_sin(10);
3     d /= 2.0;
4     return 0;
5 }
6
7 double my_sin(double d) {
8     return (d < -1.0 || 1.0 < d) ? 1.0 / d : d;
9 }
```

- *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'; have 'double(double
>>)'
    7 | double my_sin(double d) {
      |           ~~~~~
impldeclredecl.c:2:16: note: previous implicit declaration of 'my_sin' with type
>> 'int()'
    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;
}
```

- missing-brace.c:

```
1#include <stdio.h>
2
3void kiir(int i) {
4    printf("A szám: %d\n", i);
5
6int main() {
7    int i;
8    scanf("%d", &i);
9    kiir(i);
10   return 0;
11}
```

- 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() {
     |     ^~~~
```



- *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.

```
#include <stdio.h>

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

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

- missing-semicolon.c:

```
1#include <stdio.h>
2
3int main() {
4    int i;
5    scanf("%d", &i)
6    printf("A beolvasott szám: %d\n", i);
7    return 0;
8}
```

- 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.

```
#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`:

```
1#include <stdio.h>
2
3int main() {
4    int i;
5    scanf("%d", &i);
6    printf("A beolvasott szám: %d\n", i);
7    return 0;
8}}
```

- *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.

```
#include <stdio.h>

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



1

GCC hibaüzenetek

- Bevezető
- Figyelmeztetések

- Fordítási hibák
- Szerkesztési hibák**



# 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`:

```
1#include <stdio.h>
2
3int Main() {
4    int i;
5    scanf("%d", &i);
6    printf("A beolvasott szám: %d\n", i);
7    return 0;
8}
```

- *Hibaüzenet:*

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/11/../../../../x86_64-linux-gnu/Scrt1.o:»
» in function '_start':
(.text+0x1b): 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.

```
#include <stdio.h>

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

- `undefref.c`:

```
1#include <stdio.h>
2
3int Fibb(int);
4
5int main() {
6    int i;
7    scanf("%d", &i);
8    printf("Az %d-edik Fibonacci szám: %d\n", i, Fibb(i));
9    return 0;
10}
```

- *Hibaüzenet:*

```
/usr/bin/ld: /tmp/cceDczDG.o: in function 'main':
undefref.c:(.text+0x3c): 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`:

```
1 int main() {  
2     return 0;  
3 }
```

- *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.