A feladatok Visual Studio 2008-cal lettek lefordítva. Mindenki tesztelheti a saját gépére felinstallált fordítót, de ehhez nézze meg, hogyan kell optimalizált és optimalizálatlan verziót készíteni. Az optimalizálást érdemes a legagresszívebbre állítani, főleg a futási időt optimalizálja a fordító.

MSVC 2008-nál a `/O2` fordítási opciót kell használni.

1. Feladat

Fordítsa le a `ciklus.c` forrást optimalizálás nélkül és optimalizálva. Nézze meg, hogy a futási idők hogyan alakultak. Lehetséges output:

```
E:\Reversing.Tex\ELOADAS\ea2\code>cl  ciklus.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation.  All rights reserved.
ciklus.c
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation.  All rights reserved.
/out:ciklus.exe
ciklus.obj

E:\Reversing.Tex\ELOADAS\ea2\code>ciklus
Eredmeny: 100000001.745418
Futasi ido: 4.324000

E:\Reversing.Tex\ELOADAS\ea2\code>cl /O2 ciklus.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation.  All rights reserved.
ciklus.c
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation.  All rights reserved.
/out:ciklus.exe
ciklus.obj

E:\Reversing.Tex\ELOADAS\ea2\code>ciklus
Eredmeny: 100000001.745418
Futasi ido: 1.419000
```

Nálam az optimalizált verzió kb. 3-szor gyorsabban fut.

Írassa ki a `dumpbin` program segítségével a `ciklus.obj` fájlban lévő kódot. `dumpbin /disasm ciklus.obj |more`

Nem kell megérteni a pogramot, de lehet látni, hogy az optimalizált változatban a cikluson belül ismétlődő lebegőpontos összeadások vannak. A ciklust a fordító kibontotta (loop unroll), mert gyorsítótár használata esetén a kevesebb ugrást végrehajtó programok gyorsabban futnak.

```
  ....
  00000026: B8 00 E1 F5 05      mov         eax,5F5E100h
  0000002B: 83 E8 01            sub         eax,1
  0000002E: D8 C1               fadd        st,st(1)
  00000030: D8 C1               fadd        st,st(1)
  00000032: D8 C1               fadd        st,st(1)
  00000034: D8 C1               fadd        st,st(1)
  00000036: D8 C1               fadd        st,st(1)
  00000038: D8 C1               fadd        st,st(1)
  0000003A: D8 C1               fadd        st,st(1)
  0000003C: D8 C1               fadd        st,st(1)
  0000003E: D8 C1               fadd        st,st(1)
  00000040: D8 C1               fadd        st,st(1)
  00000042: 75 E7               jne         0000002B
  00000044: DD D9               fstp        st(1)
  ....
```

2. Feladat

Fordítsuk le az `aliasing.c` programot, úgy hogy a `*px =0;` utasítás ki legyen kommentezve. Tegyünk be a ciklusba felesleges utasításokat, pl. ismételjük meg sokszor az `y = x + 0.1;` utasítást. Fordítsuk le a programot optimalizálva és optimalizálás nélkül. Így négy futási eredményt kapunk.

Mit tapasztaltunk?

Optimalizálás nélkül nő a futás ideje. Optimalizáláskor a fordító kihagyja a felesleges utasításokat (ami nincs hatással az eredményre), így a futási idő változatlan maradt.

3. Feladat

Tegyük be a kódba (vegyük ki a kommentből) a `*px =0;` utasítást. A `px` mutasson az `x` változóra.

A `for` ciklusban csak ez legye (a többit kommentezzük ki):

```
for( i=0; i<1000000000; i++)
{
    y = x + 0.1;
    *px =0;
    y = x + 0.1;
    x = y;
}
```

Futtassuk le megint az optimalizált és a nem optimalizált változatot. Mit tapasztalunk? Az optimalizált pillanatok alatt lefut. Mivel a fordító rájött, hogy az `x` változó értéke független a ciklustól, ezért a teljes ciklust kioptimalizálta. Győződjünk meg erről. Ha a `dumpbin /disasm aliasing.obj |more` parancs segítségével a tárgykódot kiíratjuk, nem találunk benne `jne` vagy egyéb ugró utasításokat.

4. Feladat

Az `aliasing.c` fájlban a `px`-et állítsuk át, hogy az `y` változóra mutasson `px = &y;` és fordítsuk le optimalizálva. Mit tapasztaltunk? A fordító felismerte, hogy a `px` pointer az `y` változóra mutat, ami az `y` értékét kinullázza, de rögtön utána az `y`-ba kerül a számított érték, tehát az utasítás elhagyható. Ebben az esetben hasonló kód keletkezett, mintha a `px` pointer ott sem lett volna.

5. Feladat

Valóban olyan egyszerű a fordítónak felismerni, hogy a `px` pointer hova mutat, mint ahogy ezt a 3-4. feladat alapján gondolnánk? Az `aliasing2.c` fájlban a `px` pointer a `px = &y;` utasítás hatására az `y` változóra mutat, de a programban a `--px;` utasítás után az `x` változóra fog mutatni. Vigyázz, fordítófüggő, hogy a stack-en milyen sorrendben vannak a lokális adatok. A függvényparamétereknél más a helyzet a változó számú paraméterek átadása miatt. Ugyanakkor szó (vagy egyéb) határra való igazítás is szerepet játszhat. Az MSVC esetén ez a trükk működött. Fordítsuk le a programot optimalizálva és optimalizálás nélkül. Mi a két futás között az eltérés?

A különbözőképpen lefordított kód eltérő eredményt ad, ami fordítási hibára utal. A fordító nem tudta követni (aliasing probléma), hogy a pointer futás időben hova mutat és az optimalizálás hibát okozott.

Ha valaki más fordítóval teszteli, kérem írja meg a tapasztalatait. Akkor is, ha nem tudja reprodukálni az itt leírtakat.