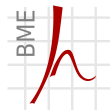


Linker és Loader

Kódvisszafejtés.



Híradástechnikai Tanszék

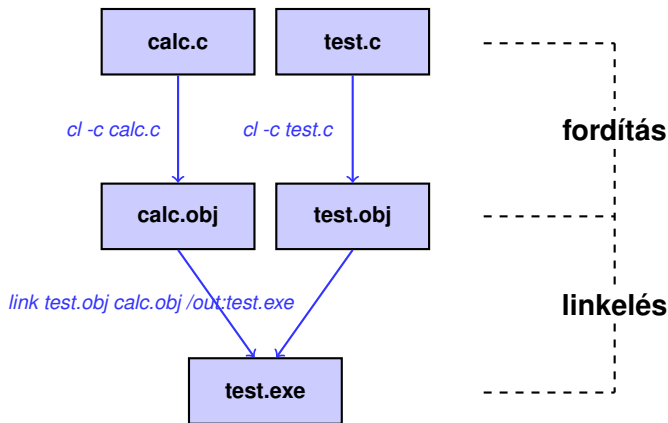
Izsó Tamás

2012. november 23.

Section 1

Linker és loader

Több modulból álló C program fordítása



Több modulból álló C program fordítás

```
1 // Inclusion guard
2 #ifndef _CALC_H_
3 #define _CALC_H_
4
5
6 int Add( int a, int b );
7 void Function( void );
8
9 // End the inclusion guard
10 #endif
```

calc.h header file

Több modulból álló C program fordítás

```
1 #include <stdio.h>
2 // Define 2 functions
3
4 // Add will return the sum of two numbers
5 int Add( int a, int b )
6 {
7     return a + b ;
8 }
9
10 // Function will print out a text string
11 void Function( void )
12 {
13     printf( "Function called! \n" );
14 }
```

calc.c rutinokat tartalmazó forrásfájl

Több modulból álló C program fordítás

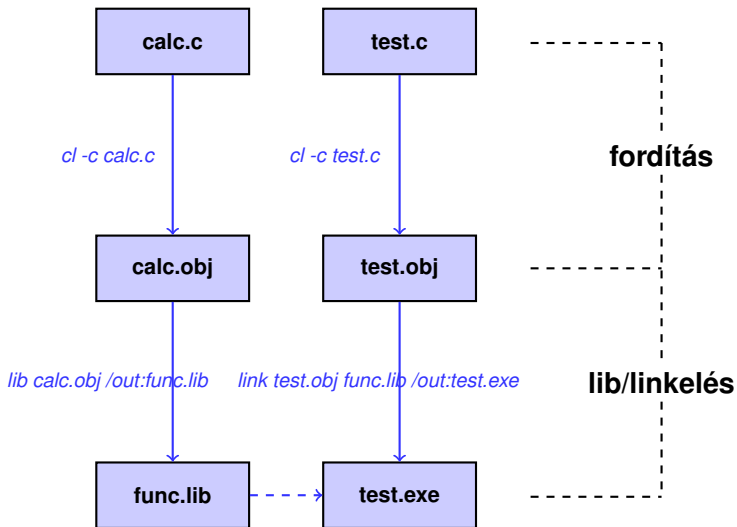
```
1 #include <stdio.h>
2 #include "calc.h"
3
4 int main()
5 {
6     printf("65+44=%d\n", Add(65, 44) );
7     Function();
8     return(1);
9 }
```

test.c főprogram forrásfájl

Statikus linkelés

- `*.lib` lefordított tárgykódok (obj) gyűjteménye (archive, library);
- linkelésnél a futtatható program részévé válik `.lib` egy része;
- egyszerű a programot telepíteni;
- sok memóriát használ, mivel minden egyes futtatható fájlban azonos kódrészek találhatóak;
- a javítás újralinkelést igényel (nehéz az sw. karbantartása).

Több modulból álló C program fordítás



Fordítás menete

Több modulból álló C program fordítás

```
cl -c calc.c  
lib calc.obj /OUT:func.lib  
cl -c test.c  
link test.obj func.lib /OUT:test.exe
```

Könyvtár megadása pragma direktívával

```
1 #include <stdio.h>
2 #include "calc.h"
3
4 #pragma comment(lib, "func.lib")
5
6 int main()
7 {
8     printf("65+44=%d\n", Add(65, 44) );
9     Function();
10    return(1);
11 }
```

Több modulból álló C program fordítás

```
cl -c calc.c  
lib calc.obj /OUT:func.lib  
cl -c test_pragma.c  
link test_pragma.obj /OUT:test.exe
```

Ilyenkor nem szükséges a func.lib megadása. Az standard C függvénykönyvtár is így van megadva. Ezt az információt a lefordított kód tárolja.

Dinamikus linkelés

- dinamikus (osztott) könyvtár a processzel egyidőben töltődik be, ha még nincs a memóriában;
- a dinamikus könyvtár tartalma külön egységet képez a háttértárolón és a memóriában is;
- a lefordított programba nem épül be;
- hatékony a memória felhasználás;
- könnyű a karbantartás;
- plugin koncepciót támogatja (interface kialakítás, későbbi funkciók létrehozása újralinkelés nélkül);

Dinamikus könyvtár – implicit link

```
1 // Inclusion guard
2 #ifndef _DLLTUT_DLL_H_
3 #define _DLLTUT_DLL_H_
4
5 __declspec(dllimport) int Add( int a, int b );
6 __declspec(dllimport) void Function( void );
7
8 // End the inclusion guard
9 #endif
```

calc.h dll-hez tartozó header fájl

Dinamikus könyvtár – implicit link

```
1 #include <stdio.h>
2 #include <windows.h>
3
4 // primitív összeadó függvény
5 __declspec(dllexport) int Add( int a, int b )
6 {
7     return( a + b );
8 }
9
10 // primitív kiíró függvény
11 __declspec(dllexport) void Function( void )
12 {
13     printf( "DLL Called! \n" );
14 }
```

Dinamikus könyvtár – implicit link

```
15
16 BOOL APIENTRY DllMain(HANDLE hModule,
17     DWORD ul_reason_for_call,
18     LPVOID lpReserved)
19 {
20     switch( ul_reason_for_call ) {
21     case DLL_PROCESS_ATTACH:
22         printf("DLL_PROCESS_ATTACH\n");
23         break;
24     case DLL_THREAD_ATTACH:
25         printf("DLL_THREAD_ATTACH\n");
26         break;
27     case DLL_THREAD_DETACH:
28         printf("DLL_THREAD_DETACH\n");
29         break;
30     case DLL_PROCESS_DETACH:
31         printf("DLL_PROCESS_DETACH\n");
32         break;
33     }
```

Dinamikus könyvtár – implicit link

```
cl /LD calc.c
```

calc.c DLL fordítás

```
__declspec( dllimport ) declarator  
__declspec( dllexport ) declarator
```

A `dllimport` és a `dllexport` Microsoft specifikus tárolási osztály attribútum. Meghatározza, hogy melyik adathoz vagy függvényhez férhetünk hozzá, vagy melyre van szükségünk egy másik DLL-ből. Fordító ezek alapján optimalizálja a DLL-ben lévő függvények hívását. A `.DEF` modul definíciós fájlban is meg lehet adni.

Dinamikus könyvtár – implicit link

```
1 // Loading DLLs The Easy Way
2 #include <stdio.h>
3
4 #include "../b/calc.h"
5
6 int main()
7 {
8     // call and print the result of Add() from DLL
9     printf("123+456=%d\n", Add(123, 456) );
10    Function();      // call Function() from DLL
11    return 1;
12 }
```

dll hívás egyszerű módon

Dinamikus könyvtár – implicit link

```
cl -c test_implicit.c  
link test_implicit.obj ../b/calc.lib
```

test_implicit program fordítása és linkelése

Dinamikus könyvtár – explicit link

```
1 // Dynamic Loading
2 #include <stdio.h>
3 #include <windows.h>
4
5 // Függvényre mutató pointer típusdefiníciója
6 typedef int (*AddFunc)(int, int);
7 typedef void (*PrintFunc)();
8
9 int main()
10 {
11     // Függvényre mutató pointer
12     int (*_AddFunc)(int, int);
13     void (*_PrintFunc)();
```

dll hívása bonyolult módon

Dinamikus könyvtár explicit linkelése

```
15  HINSTANCE hInstLib = LoadLibrary("calc.dll");
16
17  if (hInstLib) {
18      // pointerek beállítása
19      _AddFunc = (AddFunc)GetProcAddress(hInstLib, "Add");
20      _PrintFunc =
21          (PrintFunc)GetProcAddress(hInstLib, "Function");
22      // Hiba esetén NULL pointert kapunk vissza
23      if (_AddFunc) printf("123+456=%d\n", _AddFunc(123, 456) );
24      if (_PrintFunc) _PrintFunc();
25      // DLL könyvtár felszabadítása
26      FreeLibrary(hInstLib);
27  } else {
28      // Our DLL failed to load!
29      printf( "DLL betoltes hibas! \n");
30  }
31
32  return 0;
33 }
```

Dinamikus könyvtár explicit linkelése

```
cl -c test_explicit.c  
link test_explicit.obj
```

test_explicit program előállítás

Linkerek megjelenése

- Kezdetben a programozás gépi kódban történt;
- a program kezdőcíme ismert volt;
- az ugások címét az adatok és szubrutinok helyét a gépi utasítások hosszának ismeretében a programozó határozta meg;
- a rutinok összefűzése vagy bővítése esetén a címeket újra kellett számolni

1974-ben John Mauchly az ENIAC project vezetője olyan betöltő és linkelő programot írt amely képes volt a 0-ás címen kezdődő szubrutinokat összefűzni. Ekkor még nem volt fordítóprogram.

Szimbólumtábla

- A programozási nyelvekben az típusokra, adatokra, függvényekre, ugrási címekre szimbólikus nevekkel hivatkozunk.
- Vannak név nélküli objektumok, amelyekhez a fordító generál szimbólikus nevet.
- A kód és az adat különböző memóriaterületre kerül, ezért ezek a lefordított kódban is külön szegmensben vannak. A szegmensek nevekkel rendelkeznek, ami szintén megtalálható a szimbólumtáblában.
- A linker számára az a szimbólum érdekes, melyek a program kezdőcímétől függ. Például globális függvények, static változók, de lényegtelenek a típusok, lokális változók, függvényparaméterek.

Common Object File Format COFF

- Ezt a fájlformátumot először a Unix rendszerben a lefordított a.out program tárolására használták.
- Szimbólikus nevek csak 8 karakteresek lehetnek.
- Nem támogatja a DLL-ek hívását.
- A fordítók által generált összes debug információ tárolása nehézségekbe ütközik.
- Sok módosított változata létezik, XCOFF vagy a Microsoft létrehozta a PE/COFF, linuxban viszont áttértek az ELF fájl típusra.

PE/COFF fájl főbb részei

- fájl header – milyen gépen lehet futtatni a kódot, mikor keletkezett a fájl, szimbólumok hol találhatóak, stb;
- opcionális header – linker verzió, kód hossza, inicializált és nem inicializált adatok mérete, program első végrehajtandó utasításának a címe, stb;
- section header – egy bejegyzés tartalmazza a szegmens méretét és helyét, a hozzá tartozó relokációs információ helyét, line number table helyét, VM típus (read only, executable, stb);
- relokációs tábla – memóriahelytől függő információk;
- line number table – forrássorok összerendelése a kóddal (debug esetén) ;
- szimbólumtábla – szimbólum neve (8 bájtnál hosszabb nevek esetén a string táblára hivatkozik), szimbólumot tartalmazó szegmens sorszáma, szegmensben belüli hely;
- String tábla – 8 karakternél hosszabb neveket tartalmazza.

Példa szimbólumokra — a.c

```

int a, b; // globális, UNDEF section
int c = 5; // globális, .data section
const int d = 64; // globális, .rdata
char s1[]="alma"; // globális, .data
const char s2[]="korte"; // globális, .rdata
char* s3="dio"; // globális, .data
int t[100]; // globális, UNDEF section
int* p=t; // globális, .data
extern void func(const char *); // globális, UNDEF section

int main() {
    int e=5, f;
    static int g; // lokális .data
    char s4[]="banan";
    static char s5[]="ananasz"; // lokális .data
    switch( e ) {
        case 0 : func("case_0"); break;
        case 1 : func(s4); break;
        case 2 : func(s5); goto Lab5;
        case 3 : func(s1); break;
        case 4 : func(s2); break;
        Lab5: // lokális .text
        case 5 : func(s3); break;
    }
    return 0;
}

```

a.c program lefordított kódjá 1.

```

_main:
00000000 55                push  ebp
00000001 8B EC            mov   ebp, esp
00000003 83 EC 14        sub   esp, 14h
00000006 C7 45 F0 05000000 mov   dword ptr [ebp-10h], 5
0000000D A1 00000000     mov   eax, dword ptr [$_SG2485]
00000012 89 45 F4        mov   dword ptr [ebp-0Ch], eax
00000015 66 8B 0D 04000000 mov   cx, word ptr [$_SG2485+4]
0000001C 66 89 4D F8     mov   word ptr [ebp-8], cx
00000020 8B 55 F0        mov   edx, dword ptr [ebp-10h]
00000023 89 55 EC        mov   dword ptr [ebp-14h], edx
00000026 83 7D EC 05     cmp   dword ptr [ebp-14h], 5
0000002A 77 63          ja    0000008F
0000002C 8B 45 EC        mov   eax, dword ptr [ebp-14h]
0000002F FF 24 85 00000000 jmp   dword ptr $LN11[eax*4]
$LN6:
00000036 68 00000000     push  offset $_SG2493
0000003B E8 00000000     call  _func
00000040 83 C4 04        add   esp, 4
00000043 EB 4A          jmp   0000008F
$LN5:
00000045 8D 4D F4        lea  ecx, [ebp-0Ch]
00000048 51            push  ecx
00000049 E8 00000000     call  _func
0000004E 83 C4 04        add   esp, 4
00000051 EB 3C          jmp   0000008F
$LN4:
00000053 68 00000000     push  offset ?s5@?1??main@@@9@9
00000058 E8 00000000     call  _func

```

a.c program lefordított kódja 2.

```

0000005D 83 C4 04      add    esp,4
00000060 EB 1E         jmp    $Lab5$2496
$LN3:
00000062 68 00000000   push  offset _s1
00000067 E8 00000000   call  _func
0000006C 83 C4 04      add    esp,4
0000006F EB 1E         jmp    0000008F
$LN2:
00000071 68 00000000   push  offset _s2
00000076 E8 00000000   call  _func
0000007B 83 C4 04      add    esp,4
0000007E EB 0F         jmp    0000008F
$Lab5$2496:
00000080 8B 15 00000000 mov   edx,dword ptr [_s3]
00000086 52          push  edx
00000087 E8 00000000   call  _func
0000008C 83 C4 04      add    esp,4
0000008F 33 C0       xor   eax,eax
00000091 8B E5       mov   esp,ebp
00000093 5D          pop   ebp
00000094 C3          ret
00000095 8D 49 00    lea  ecx,[ecx]
$LN11:
00000098 00 00 00 00
0000009C 00 00 00 00
000000A0 00 00 00 00
000000A4 00 00 00 00
000000A8 00 00 00 00
000000AC 00 00 00 00

```

a.obj-ban lévő szimbólumtábla

```

000 00837809 ABS      notype      Static      | @comp.id
001 00000001 ABS      notype      Static      | @feat.00
002 00000000 SECT1   notype      Static      | .drectve
    Section length 2F, #relocs 0, #linenums 0, checksum 0
004 00000000 SECT2   notype      Static      | .debug$$
    Section length 7C, #relocs 0, #linenums 0, checksum 0
006 00000000 SECT3   notype      Static      | .data
    Section length 2F, #relocs 2, #linenums 0, checksum CFCD7EC4
008 00000000 SECT3   notype      Static      | $$SG2473
009 00000004 UNDEF   notype      External   | _a
00A 00000004 UNDEF   notype      External   | _b
00B 00000190 UNDEF   notype      External   | _t
00C 00000000 SECT3   notype      External   | _c
00D 00000000 SECT4   notype      Static      | .rdata
    Section length A, #relocs 0, #linenums 0, checksum 6DAF2FDC
00F 00000000 SECT4   notype      External   | _d
010 00000004 SECT3   notype      External   | _s1
011 00000004 SECT4   notype      External   | _s2
012 00000010 SECT3   notype      External   | _s3
013 00000014 SECT3   notype      External   | _p
014 00000018 SECT3   notype      Static      | $$SG2485
015 00000020 SECT3   notype      Static      | ?s5@?1??main@@@9@9 ('main'::'2'::s5)
016 00000028 SECT3   notype      Static      | $$SG2493
017 00000000 SECT5   notype      Static      | .text
    Section length B0, #relocs 14, #linenums 0, checksum 44157CC3
019 00000000 SECT5   notype      External   | _main
01A 00000080 SECT5   notype      Label     | $Lab5$2496
01B 00000071 SECT5   notype      Label     | $LN2
01C 00000062 SECT5   notype      Label     | $LN3
01D 00000053 SECT5   notype      Label     | $LN4
01E 00000045 SECT5   notype      Label     | $LN5
01F 00000000 UNDEF   notype      External   | _func
020 00000036 SECT5   notype      Label     | $LN6
021 00000098 SECT5   notype      Static      | $LN11

```

a.obj .text szegmenséhez tartozó relokációs tábla

Offset	Type	Applied To	Index	Name
0000000E	DIR32	00000000	14	\$SG2485
00000018	DIR32	00000004	14	\$SG2485
00000032	DIR32	00000000	21	\$LN11
00000037	DIR32	00000000	16	\$SG2493
0000003C	REL32	00000000	1F	_func
0000004A	REL32	00000000	1F	_func
00000054	DIR32	00000000	15	?s5@?1??main@@@9@9 ('main'::'2'::s5)
00000059	REL32	00000000	1F	_func
00000063	DIR32	00000000	10	_s1
00000068	REL32	00000000	1F	_func
00000072	DIR32	00000000	11	_s2
00000077	REL32	00000000	1F	_func
00000082	DIR32	00000000	12	_s3
00000088	REL32	00000000	1F	_func
00000098	DIR32	00000000	20	\$LN6
0000009C	DIR32	00000000	1E	\$LN5
000000A0	DIR32	00000000	1D	\$LN4
000000A4	DIR32	00000000	1C	\$LN3
000000A8	DIR32	00000000	1B	\$LN2
000000AC	DIR32	00000000	1A	\$Lab5\$2496

b.c fordítási egység

```
int b=2;
extern void func(const char *s) {
    int i;
    for(i=0; s[i]; i++ ) b+=*s;
}
```

b.obj .text szegmensének a szimbólumtáblája:

```
000 00837809 ABS      notype      Static      | @comp.id
001 00000001 ABS      notype      Static      | @feat.00
002 00000000 SECT1   notype      Static      | .drectve
      Section length 2F, #relocs 0, #linenums 0, chksum 0
004 00000000 SECT2   notype      Static      | .debug$$S
      Section length 7C, #relocs 0, #linenums 0, chksum 0
006 00000000 SECT3   notype      Static      | .data
      Section length 4, #relocs 0, #linenums 0, chksum AA09C88B
008 00000000 SECT3   notype      External   | _b
009 00000000 SECT4   notype      Static      | .text
      Section length 3B, #relocs 2, #linenums 0, chksum 14D5DB06
00B 00000000 SECT4   notype      External   | _func
```

.text szegmenshez tartozó relokációs tábla:

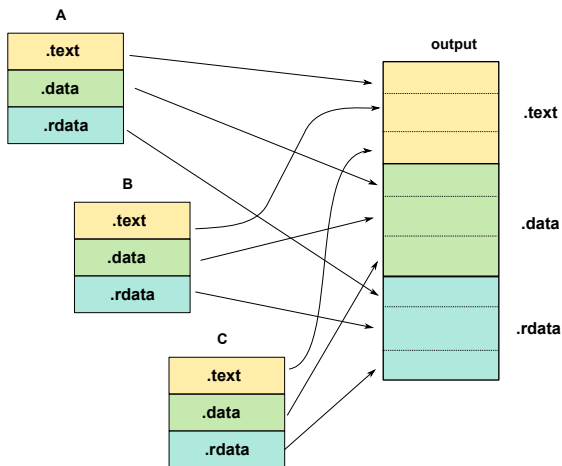
Offset	Type	Applied To	Index	Name
0000002B	DIR32	00000000	8	_b
00000031	DIR32	00000000	8	_b

Szegmensek

A szegmensek nevét a fordító határozza meg. A kódszegmens neve Visual C fordító esetén `.text` Borland fordító esetén `CODE`. Néhány szegmens elnevezése Visual Studió használatkor:

- `.drective` linker opció
- `.text` kódszegmens
- `.data` inicializált adatszegmens
- `.rdata` csak olvasható adatszegmens
- `.bbs` nem inicializált adatszegmens
- `.reloc` relokációs tábla
- `.rsrc` resource (ikon, cursor alak, stb.)
- `.edata` export tábla
- `.idata` import tábla

Objektumok linkelése



Adatok relokálása

- A linker vagy loader ismeri, hogy az m modulban lévő szegmens kezdete hova esik a betöltés vagy az össze-szerkesztés alatt.
- A szimbólumtábla tartalmazza, hogy a szimbólum melyik szegmensben van.
- A szimbólumtábla tartalmazza, hogy a szimbólum az öt befoglaló szegmens kezdetétől hol helyezkedik el.
- Minden szegmensnek külön relokációs táblája van.
- A linker és a loader a relokációs tábla alapján tudja a cím-kiigazítás helyét és a relokálandó adatok méretét. A kód értelmezésével nem foglalkozik.

Relokációs algoritmus

```

foreach section s {
  foreach relocation entry r {
    refptr = s + r.offset; /* ptr to reference to be relocated */

    /* Relocate a PC-relative reference */
    if (r.type == REL32) {
      /* ref's run-time address. */
      refaddr1 = ADDR(s) + r.offset + 4;
      *refptr = ADDR(r.symbol) + *refptr - refaddr;
    }

    /* Relocate an absolute reference */
    if (r.type == DIR32)
      *refptr = ADDR(r.symbol) + *refptr;
  }
}

```

¹Az IP a relokálható operandusú utasítás végrehajtása pillanatában már a következő utasításra mutat, és ehhez képest kell a relatív címet kiszámítani. A relokálás típusa egyértelműen megadja, hogy 4-et kell a relokálás helyéhez adni.

Linker által használt adatok a számításokhoz

- A linker a program kezdőcímének a 0x00400000 értéket használta. Ezen a helyen a PE/COFF fájlformátum header része található.
- A kód szegmens külön lapra kerül a virtuális memóriában. Egy lap mérete 4Kbyte, ezért az a.obj kódszegmense a 0x00401000 címre kerül.
- Az a.obj kódszegmense 0xB0 hosszú, ezért a b.obj kódszegmense közvetlenül az a.obj kódszegmense után, a 0x004010B0 címre kerül.
- A .data adatszegmens új lapra, azaz a 0x00402000 címre kerül.

Abszolút hivatkozás címkiszámítása

Relokálható adat az a.obj .text szegmensében:

```
00000037 DIR32 00000000 16 $SG2493
```

Relokáció helye a kódban:

```
00000036: 68 00000000      push      offset $SG2493
```

Szimbólumtábla bejegyzés:

```
006 00000000 SECT3 notype Static | .data
      length 2F, #relocs 2, #linenums 0, chksum CFCD7EC4
016 00000028 SECT3 notype Static | $SG2493
```

```
Addr( $SG2493 ) = Addr(.data) + Offset($SG2493) =
= 0x40A000 + 0x28 = 0x40A028
```

```
refptr = Addr(.text) + Addr(reloc offset)
= 0x401000 + 0x37 = 0x401037
```

```
*refptr = Addr(r.symbol) + *refptr
= 0x401028 + 0 = 0x401028
```

Eredmény megtekintése ollydbg alatt:

```
00401036 68 28A04000      PUSH OFFSET 0040A028
```

IP relatív hivatkozás címkiszámítása

Az a.obj .text szegmenséhez rendelt relokálandó adat:

```
00000059 REL32 00000000 1F _func
```

Relokálandó kód az a.obj moduban:

```
00000058: E8 00000000 call _func
```

A b.obj-ban lévő szimbólumtábla bejegyzés:

```
009 00000000 SECT4 notype Static | .text
    length 3B, #relocs 2, #linenums 0, chksum 14D5DB06
00B 00000000 SECT4 notype () External | _func
```

```
Addr(r.symbol) = Addr(_func)
    = Addr(b.obj .text)+Offset(_func) = 0x4010B0+0 = 0x4010B0
```

```
refaddr = Addr(a.obj .text) + r.offset + 4
    = 0x401000 + 0x59 + 4 = 0x40105D
```

```
refptr = Addr(.text) + r.offset = 0x401000 + 0x59 = 0x401059
```

```
*refptr = ADDR(r.symbol) + *refptr - refaddr
    = 0x4010B0 + 0 - 0x40105D = 0x53
```

Eredmény megtekintése ollydbg alatt:

```
00401058 E8 53000000 CALL 004010B0
```