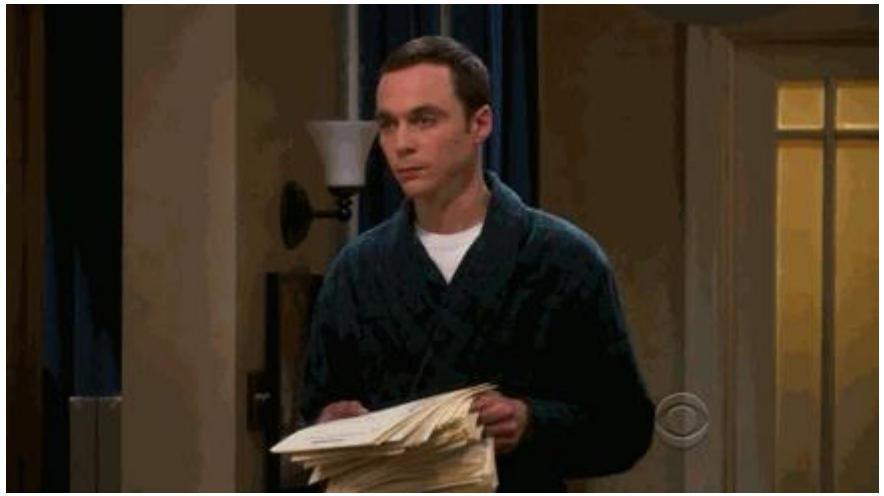
A Programozás Alapjai 2
Objektumorientált szoftverfejlesztés

Dr. Forstner Bertalan
forstner.bertalan@aut.bme.hu

Tudnivalók a tárgyról



Gyors ismétlés: a fordítás folyamata

- A **preprocesszor** behelyettesítést végez
  - pl. #define, #include, stb.
  - DE: tokenizál
    - a „*#define a A*” nem fog minden *char*t kicserélni *chA*rra.
- a **fordító** (compiler) végzi az “oroszlánrészt”,
- a **linker** összefűzi a különböző fordítókimeneteket egy állományba.

Kvíz: mit fog kiírni?

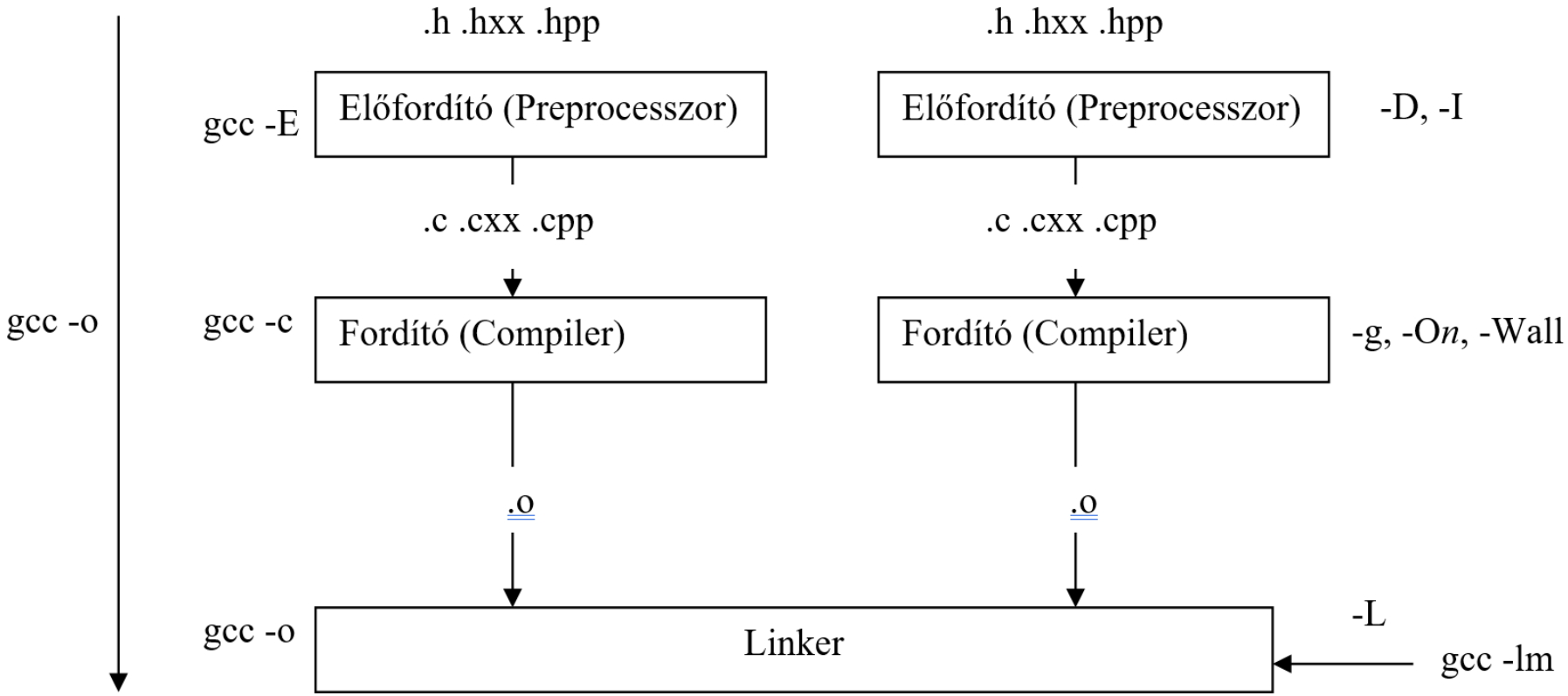
```
#include <stdio.h>
#define PONT 17;

int main(int argc, char* argv[]) {
     int i;
     i = PONT + 3;
     printf("%d", i);
     getchar();
     return 1;
}
```

Kvíz: mit fog kiírni?

```
#include <stdio.h>
#define PONT 17;

int main(int argc, char* argv[]) {
     int i;
     i = PONT + 3;
     printf("%d", i);
     getchar();
     return 1;
}
```

A: „PONT”
B: „17”
C: „20”
D: „17+3”

Gyors ismétlés: a fordítás folyamata



Általános CISC architektúra felépítése

- ALU („Számológép)
- Regiszterek („D tárolók”):
  > EAX: aritmetikai műveletek
  > EIP: utasítás mutató
  > ESP: veremmutató
  > EBP: bázismutató
- Memória
- Vezérlőegység („Karmester”): lehívja az utasításokat a memóriából, majd végrehajttatja azokat.

Srácok, feltaláltam a Neumann elvet!

Utasítástípusok

- Mozgató utasítások
  > memóriacímek és regiszterek között
- Aritmetikai-logikai
  > összeadás, kivonás, összehasonlítás
- Vezérlésátadó utasítások
  > ugrások, feltételes ugrások, függvényhívás

Példa

- While ciklus assembly kódja

```
int i = 0;
while (i != 4)
  { i++; }
```

```
  int i = 0;
010D16EE  mov         dword ptr [i],0
  while (i != 4) {
010D16F5  cmp         dword ptr [i],4
010D16F9  je          main+36h (010D1706h)
  i++;
010D16FB  mov         eax,dword ptr [i]
010D16FE  add         eax,1
010D1701  mov         dword ptr [i],eax
```

Példa

- A függvényhívás folyamata és paraméter-átadás

A verem a függvényben:

Elmentett további regiszterek (Ide mutat a függvényen belüli ESP)
Lokális változók
Régi EBP (ide mutat a régi ESP)
Régi EIP
c Függvényparaméterek b a

```
int main(int argc, char* argv[])
{
  fv(1, 2, 3);
  …
}
```

```
  fv(1, 2, 3);
00D61B0E  push        3
00D61B10  push        2
00D61B12  push        1
00D61B14  call        fv
(0D6134Dh)
00D61B19  add         esp,0Ch
```

```
void fv(int a, int b, int c) {
  int s = a + b + c;
  return;
}
```

```
void fv(int a, int b, int c) {
00D616D0  push        ebp
00D616D1  mov         ebp,esp
00D616D3  sub         esp,0CCh
00D616D9  push        ebx
00D616DA  push        esi
00D616DB  push        edi
00D616DC  lea         edi,[ebp-0CCh]
00D616E2  mov         ecx,33h
00D616E7  mov         eax,0CCCCCCCCh
00D616EC  rep stos    dword ptr
es:[edi]
int s = a + b + c;
00D616EE  mov         eax,dword ptr [a]
int s = a + b + c;
00D616F1  add         eax,dword ptr [b]
00D616F4  add         eax,dword ptr [c]
00D616F7  mov         dword ptr [s],eax
```

```
return;
}
00D616FA  pop         edi
00D616FB  pop         esi
00D616FC  pop         ebx
00D616FD  mov         esp,ebp
00D616FF  pop         ebp
00D61700  ret
```

Tanulságok

- Paraméterek jobbról balra a stack-be másolódnak. **Kivétel: tömb!**
- A lokális változók a vermen foglalódnak le, és a függvényből való kilépés után eltűnnek
- Érték szerint átadott paraméterek a függvényben megváltoztva sem változnak kívül
- A hívó takarít, mert lehetnek változó paraméterek is
- A verem elfogyhat: pl.: rekurzió

A linker feladata: címfeloldás

- Az object fájl még tartalmazhat ismeretlen hivatkozást
  > Ld. pl. prototípus + megvalósítás C függvényeknél
- Extern: változó vagy függvény előtt
- Példa

A C++ atyja

Történeti áttekintés

Referencia

- Példa

Referencia

- Péld

C- rossz

```
#include<stdio.h>

void f(int i)
{
	i=i+2;
}

int main(void)
{
	int i=0;
	f(i);
	printf("%d\n",i);
}
/* A kimenet 0 */
```

C-jó

```
#include<stdio.h>

void f(int* pi)
{
	(*pi)=(*pi)+2;
}

int main(void)
{
	int i=0;
	f(&i);
	printf("%d\n",i);
}
/* A kimenet 2 */
```

C++

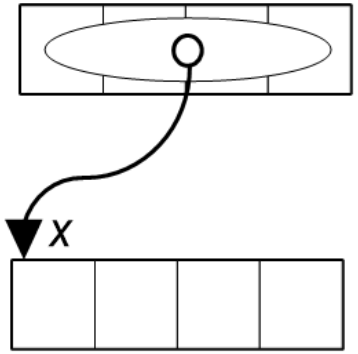
```
#include<stdio.h>

void f(int& i)
{
	i=i+2;
}

int main(void)
{
	int i=0;
	f(i);
	printf("%d\n",i);
}
/* A kimenet 2 */
```

Referencia

```
int x=10;
int* p=&x;
int& r=x;
```

- r egy int referencia
- Alias rá, ugyanazt jelenti, mint x
- **Nem keverendő** a veszem-a-címét operátorral!

r jelenti annak a rekesznek a tartalmát, ahova ez a pointer mutat



- **Memóriakép** szempontjából *r* egy pointer lesz az *x* változóra
- **Mire jó?** Ha nagyobb méretű változó stacken utazik…

…ééééés most:

Bemutatom a srácot, aki most azonnal megajánlott ötöst kap!!!

Kvíz – mit ír ki?

```
int main(int argc, char* argv[]) {

    int i = 0;
    int& ri = i;

    int* pi = &i;
    int* pri = &ri;
    if (pi == pri)
        printf("IGAZ");
    else
        printf("HAMIS");

    getchar();
    return 1;
}
```

Kvíz – mit ír ki?

```
int main(int argc, char* argv[]) {

    int i = 0;
    int& ri = i;

    int* pi = &i;
    int* pri = &ri;
    if (pi == pri)
        printf("IGAZ");
    else
        printf("HAMIS");

    getchar();
    return 1;
}
```

Nem lehet a referencia mögötti pointerhez hozzáférni!

Visszatérés referenciával

- Példa
- Ökölszabály:

Sose szolgáltassuk ki lokális változó címét a függvényen kívülre!

(Sem pointer, sem referencia által).

Visszatérés referenciával

```
int& fv(int x, int&v)
{
        int a = 10;
        return a; //HIBA! lokális!
        //V2 //
        x++;
        return x; //HIBA! Továbbra is lokális: a stackre másolódott
        //V3 //
        v++;
        return v; //Ez OK, referenciát kaptunk külső változóra.

}
```

Referencia paraméter

```
int r, p = 1, q = 0;
r = fv(1,2); //HIBA! referenciának változót kell adni!
r = fv(p, q);//OK
r = fv(1, q);//Ugyanaz, mint fenn, csak ideiglenes változóval
```

```
int& fv(int x, int&v)
{
   …
}
```

Összefoglalás

- Ismétlés: a fordítás folyamata
- A C++ története
- A referencia
  - > Inicializálása
  - > Visszatérés referenciával