

Karma C/C++ Kodlama

/*****

* **Bâkır EMRE**

* **emre ~ enderunix.org**

* EnderUNIX Yazılım Geliştirme Takımı

* <http://www.enderunix.org>

*

* Sürüm : 1.0

* Tarih : 26.06.2006

* Makalenin en yeni versiyonunu

* www.enderunix.org/docs/ccplusplusMixedCode.pdf

* adresinden elde edilebilir.

*****/

<u>Giriş</u>	<u>3</u>
<u>C Koduna C++ Kaynak Dosyasından Erişim</u>	<u>3</u>
<u>Bağlantı Belirtiminin Tanımlaması</u>	<u>3</u>
<u>C Başlık Dosyalarının C++ kodu içerisine dahil edilmesi</u>	<u>4</u>
<u>Karma Başlık Dosyası Oluşturma</u>	<u>5</u>
<u>C++ koduna C kaynak dosyasından erişim</u>	<u>6</u>
<u>Referanslar</u>	<u>11</u>

Giriş

C++ programlama dili “linkage specification” diye adlandırılan farklı derleyiciler ile derlenen ve farklı dillerde yazılan kodu aynı program içerisinde yazmaya imkan tanıyan bir mekanizmaya sahiptir. Linkage specification farklı dillerde yazılan fonksiyonları veya procedure leri bağlamaya yarayan bir protokol olarak düşünülebilir. *Linkage* (bağlantı) terimi C++ standartlarında nesnelerin aynı ya da farklı dosyalardan erişilmesi anlamına gelir. üç tip olarak karşımıza çıkar

- No linkage
- Internal linkage
- External linkage

Internal linkage (dahili bağlantı) de fonksiyonun parameterlerine ve değişkenlerine bağlı olarak kod içerisinde erişilebilirken , No Linkage de sadece fonksiyon içerisinden erişilebilir.

External Linkage (Harici bağlantıda) aşağıda örneklerini görebileceğimiz şekilde gerçekleşir. mesela C koduna C++ kullanarak erişmek istersek :

C Koduna C++ Kaynak Dosyasından Erişim

Yukarıda C++ dilinin bağlantı tanımlaması “linkage specification” ile desteklenen bir dil ile oluşturulacak program içerisine bağlandığı bir fonksiyon veya nesne tanımlamasını yapmayı sağladığını söylemiştik. Ayrıca derleyicilerin de bu özelliği desteklemesi gerektiğinde göz önünde bulundurulmalıdır. Tüm C++ derleyicileri C bağlantısını desteklemektedir.

- GNU Compiler Collection
- HP aC++
- Sun Studio C++ compiler
- IBM XL C/C++ Compiler

bunlardan bir kaçıdır ve bazı C derleyicileride bu özelliği desteklemektedir.

C bağlantısı ile derlenmiş bir fonksiyona erişmek istediğiniz zaman C bağlantılı fonksiyonu tanımlamasını yapın. Birçok c++ derleyicisinin C ve C++ nesnelere karşı farklı bağlantılarının olmamasına karşın siz C++ kodu içerisinde C bağlantısı yapacağınız verileri tanımlayın.

Bağlantı Belirtiminin Tanımlaması

Aşağıdaki yapılardan birini kullanarak nesne ya da fonksiyonun dil bağlantısını gerçekleştirebiliriz.

```
extern "Kullanılacak_dilin_adi" bildirim ;
extern "Kullanılacak_dilin_adi" { bildirim ; bildirim ; ... }
```

İlk gösterim tanım ya da bildirimini Kullanılacak dilin adına göre tanımlarken ikincisinde parantez içerisindeki her şeyin kullanılacak dile bağlanacağını gösterir. Burada dikkat etmeniz gereken nokta parantezden sonra noktalı virgülün kullanılmayışı.

Bağlantı tanımlamaları iç-içe kullanabilirsiniz. Fakat bu yapı burada scope oluşturmamaktadır.

```
extern "C" {
    void f(); // C bağlantısı
    extern "C++" {
        void g(); // C++ bağlantısı
        extern "C" void h(); // C bağlantısı
        void g2(); // C++ bağlantısı
    }
    extern "C++" void k(); // C++ bağlantısı
    void m(); // C bağlantısı
}
```

Yukarıdaki fonksiyonların hepsi iç içe kullanılmasına karşın aynı global scope a sahiptir.

C Başlık Dosyalarının C++ kodu içerisine dahil edilmesi

Eğer C derleyicileri için tasarlanmış kendi hazırladığınız başlıkları C kütüphanelerini ile beraber kullanmak isterseniz. Başlığınızı extern "C" parantezleri içerisinde yazabilirsiniz.

```
extern "C" {
    #include "baslik.h"
}
```

Dikkat-

Bu tekniği sistem başlık dosyaları ile kullanmayınız!.

Karma Başlık Dosyası Oluşturma

C ve C++ derleyicileri için uygun bir başlık dosyası hazırlamak isterseniz bütün belirteçleri extern "C" içerisine yazınız. Fakat bütün C derleyicileri Bu sözdizimini tanımayabilir. Bütün C++ derleyicileri __cplusplus makrosunu önceden tanımlanmıştır bu makroyu kullanarak c++ sözdizimini koruyabilirsiniz.

```
#ifndef __cplusplus
extern "C" {
#endif
... /* başlık kısmı */
#ifdef __cplusplus
} /* extern "C" kapatılması */
#endif
```

C++ Sınıflarına C den erişim

C++ sınıflarına C kodundan nasıl erişebiliriz. Şöyle yaparsak C ++ sınıfını C struct yapısında tanımlarsanız ve bunu ve bir şekilde bunu üye fonksiyonu çağırabiliriz

Şu şekilde bir sınıfımızın olduğunu ele alalım

```
class HerHangi {
public:
    virtual int digeri(int);
    // ...
private:
    int i, j;
};
```

Bu HerHangi Sınıfını C kodu içerisinde tanımlayamazsınız. En iyi yapacağınız şey Herhangi Sınıfının nesnelere işaretçilerle ulaşmak. Şimdi C++ kodunda extern "C" yazarak içersine HerHangi sınıfına ve nesnelere ulaşabileceğiniz bir fonksiyon yazarsak. Diğer üye fonksiyonuna erişebileceğimize. Kod aşağıdaki gibidir.

```
extern "C" int fonk_HerHangi_digeri(HerHangi* h, int i) {

    return h->digeri(i);

}
```

Aşağıda ise HerHangi sınıfını kullanan bir C kodu bulunmaktadır

```
struct HerHangi;
int fonk_HerHangi_digeri(struct HerHangi*, int);
int f(struct HerHangi* p, int j)
{ return fonk_HerHangi_digeri(p, j); }
```

C++ koduna C kaynak dosyasından erişim

Eğer C++ fonksiyonunu C bağlantısı tanımlarsanız C derleyicisi tarafından derlenmiş fonksiyon ile çağırabilirsiniz. C bağlantısı ile tanımlanmış bu fonksiyon C++ ın bütün özelliklerini taşır, fakat C dosyası içersinden çağırarak isterseniz. parametreler ve dönüş tipleri C tarafından ulaşılabilir olmalıdır mesela

Burada C++ fonksiyonun C bağlantısı ile görebilirsiniz.

```
#include <iostream>
extern "C" int print(int i, double d)
{
    std::cout << "i = " << i << ", d = " << d;
}
}
```

print fonksiyonunun başlık dosyasını C ve C++ kodu tarafında paylaşacak şekilde aşağıdaki gibi yazabilirsiniz

```
#ifndef __cplusplus
extern "C"
#endif
int print(int i, double d);
```

Simdi bir kaç örnek üzerinde konuyu anlatırsak daha anlaşılır olacak

Örnekler

Önce Merhaba Dünya yazan bir C++ fonksiyonu yazalım

```
// MerhabaYaz.cpp
```

```
#include <iostream>
extern "C" void MerhabaDunyaYaz();

void MerhabaDunyaYaz()
{
    std::out << " Merhaba Dünya! " ;
}
```

Daha sonra bu fonksiyonu kullanan bir başka C fonksiyonu

```
// MerhabaGoster.c
```

```
void MerhabaDunyaYaz();

void MerhabaDunyaGoster()
{
    MerhabaDunyaYaz();
}
```

Ve en sonunda ise bu C bağlantılı C++ fonksiyonu çalıştıran bir ana programı yazarsak

```
// anaProgram.cpp
```

```
extern "C" void MerhabaDunyaGoster();

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

Şimdi ise geriye bu yazılan C ve C++ dosyalarını derlemeye geldi

```
g++ -c oMerhabaYaz.o MerhabaYaz.cpp
gcc -c oMerhabaGoster.o MerhabaGoster.c
g++ -c oanaProgram.o anaProgram.cpp

g++ anaProgram.o MerhabaGoster.o MerhabaYaz.o
```

şimdi burada yapılan olay sırasınca C++ ve C dosyalarını kendi derleyicileri ile derlemek ve derleme sonucu oluşan object dosyalarını kullanarak ana programı çalıştırılabilir dosya haline getirme .Buradaki parametrelere gelince

- c parametresi kaynak dosyayı derler fakat link (bağlama işlemini yapmaz)
- o parametresi ise derleme sonucu ortaya çıkacak dosyanın adı için kullanılır

Şimdi C++ kodu içerisinde bir nesne oluşturup buna ilkdeğerini atayalım

//cppdosyasi.cpp

```
#include <iostream.h>

typedef class nesne* nesne_isaretcisi;

extern "C" void nesneyi_ilklendir (nesne_isaretcisi& p);
extern "C" void nesneyi_sil(nesne_isaretcisi p);
extern "C" void nesneyi_yaz(nesne_isaretcisi p);

struct nesne {
private:
    int x;
public:
    nesne() {x = 12;}
    friend void nesneyi_yaz(nesne_isaretcisi p);
};

void nesneyi_ilklendir(nesne_isaretcisi& p) {
    p = new nesne;
}

void nesneyi_sil(nesne_isaretcisi p) {
    delete p;
}

void nesneyi_yaz(nesne_isaretcisi p) {
    cout << "nesnenin degeri->x " << p->x << "\n";
}
}
```

Şimdide C dosyasını yazalım. Bunda ise bu ilgili fonksiyonları çağırıp nesneyi oluşturup, ekrana yazıp ve silelim.

//cdosyasi.c

```
typedef struct nesne* nesne_isaretcisi;

int main () {

    nesne_isaretcisi f;

    nesneyi_ilklendir(&f);

    nesneyi_yaz(f);

    nesneyi_sil(f);

}
```

kodu derlersek

```
cc -c cdosyasi.c
cc -c cplusplusdosyasi.cpp
g++ -o calistirilabilir cplusplusdosyasi.o
cdosyasi.o
```

ve çıktısı

```
nesnenin degeri->x 12
```

Bir örnek daha gösterelim önce c dosyasını yazalım

```
int ikiyeKatla(int i)
{
    return 2*i;
}
```

Fonksiyona gelen degerin 2 katını alan bi fonksiyon

```
extern "C" f(int i);

class BenimSinifim
{
public :

int cFonksiyonunuTetikle(int i)
{
    return ikiyeKatla(i);
}
};

int main()
{
    BenimSinifim oylesine;

    std::cout << "fonksiyonu dogrudan cagirinca degeri: " <<
ikiyeKatla(5) << " BenimSinifim sınıfı icerisinden
cagirinca degeri: " << oylesine.cFonksiyonunuTetikle(5)
<< std::endl;

}
```

Son örnekte ise yazılan fonksiyonu paylaşımlı kütüphane yapıp kodu derlerken bu kütüphaneyi kullanarak çalıştırılm.

Baslık dosyası

```
//baslik.h  
extern char *MerhabaDunya();
```

C kodu

```
//MerhabaDunyaYaz.c  
  
#include "baslik.h"  
  
char *MerhabaDunya() {  
    return ((char *) "Merhaba Dunya");  
}
```

Ve C++ kodu

```
//anaProgram.cpp  
  
#include <iostream.h>  
#include "baslik.h"  
  
int main() {  
    char *giris = MerhabaDunya();  
    cout << giris << "\n";  
    return (0);  
}
```

Yine bu dosyaları derlersek

```
cc -shared -o libmerhaba.so MerhabaYaz.c  
%CC -lmerhaba anaProgram.cpp  
%./a.out  
Merhaba Dunya
```

Burada yapılan olay kısaca MerhabaYaz.c dosyasından bi adet shared object oluşturmak (libmerhaba.so) ve anaProgram.cpp derlerken bu kütüphaneyi (merhaba kütüphanesini) kullanarak derleme.

Eğer derleme sırasında

/usr/libexec/elf/ld: cannot find -lmerhaba gibi bi hata alıyorsanız ya -l parametersinden sonra oluşan paylaşımlı kütüphane dosyasının tam yolunu yazın yada bu kütüphane dosyasını kütüphane dizinine kopyalayın

```
g++ -l/home/emre/mixedcode/merhaba anaProgram.cpp
```

yada

```
cp libmerhaba.so /usr/local/lib/
```

sonra derleyin

```
g++ -lmerhaba anaProgram.cpp
```

Referanslar

<http://msdn2.microsoft.com/en-us/library/0603949d.aspx>

<http://developers.sun.com/prodtech/cc/articles/mixing.html>

http://developers.sun.com/solaris/articles/external_linkage.html

<http://docs.hp.com/en/2159/otherlangs.htm>

<http://publib.boulder.ibm.com/infocenter/lnxpcomp/v8v101/index.jsp?topic=/com.ibm.xlcpp8l.doc/language/ref/cplr020.htm>