C ve C++ arasındaki Farklılıklar

Bildirimlere `(Declarations)` İlişkin Farklılıklar

C89'da fonksiyonların örtülü *(implicit)* bildirimi geçerli. C99 standartları ile bu durum geçersiz hale getirildi. Ancak C derleyicilerinin hemen hepsi geçmişe doğru uyumluluğu korumak amacıyla bu duruma izin veriyor. C++'da örtülü işlev bildirimi geçerli değil.

```
void func()
{
 foo();
 //C89'da geçerli implicit olarak
 //int foo();
 //bildirimi yapılmış kabul ediliyor
 //C99'da geçersiz C++'da geçersiz
}
```

Klasik C döneminden gelen "eski stil fonksiyon tanımlamaları" *(old style function definitions)* C'de geçerliliğini koruyor. C++ dilinde geçerli değil.

```
sum(a, b, c)
double a, b, c;
{
 return a + b + c;
}
```

Yukarıdaki fonksiyon tanımı C'de geçerli, C++ dilinde geçersiz.

C89'da *implicit int* (gizli int - örtülü int - kapalı int) kuralı geçerli.

C99 standartları ile bu durum geçersiz kılındı. C++ dilinde "*implicit int*" geçerli değil. C99/11 derleyicileri, geçmişe doğru uyumluluğu korumak için bir bulgu iletisi *(diagnostic)* vererek kodu derleyebilir. Derleyicinizin *switch*'lerini inceleyiniz.

```
foo(); //C++'da geçersiz

func()  //C++'da geçersiz
{
 const x = 1;  //C++'da geçersiz
 static y = 2; //C++'da geçersiz
 ///
 return 1;
}
```

C89 standartlarına göre yukarıdaki kodda + bildirilen *foo* işlevinin gerei dönüş değeri türü *int* + tanımlanan *func* işlevinin geri dönüş değeri türü *int* + *func* işlevi içinde tanımlanan *x* ve *y* değişkenlerinin türü *int*

C dilinde aşağıdaki iki bildirim arasında önemli bir fark var:

```
  void f1();      //f1 işlevinin parametre değişkenleri hakkında bir bilgi verilmiyor
  void f2(void);  //f2 işlevinin parametre değişkeni yok
```

C'de bu iki bildirimin arasındaki farklılık geçmişe doğru uyumluluğu korumayı amaçlıyor. C++ dilinde bu iki bildirim eşdeğer. (İki bildirimin arasında bir anlam farkı yok) Aşağıdaki kodu hem C dilinde hem de C++ dilinde derleyerek derleyicinizin verdiği bulgu iletilerini *(diagnostic messages)* inceleyin:

```
void func(void);
void foo();

int main()
{
 func(1, 2, 4);
 foo(1, 2, 4);
}
```

C89'da for döngülerinin parantezi içindeki birinci kısımda bildirim yapılamıyor. C99 standartları ile böyle bildirimlere olanak sağlandı. C++ dilinde ise for döngülerinin birinci kısmında bildirim yapılabiliyor. Ancak burada bildirilen isimlerin kapsamlarına *(scope)* ilişkin C ile C++ dilleri arasında önemli bir kural farklılığı var. Aşağıdaki kodu inceleyin:

```
#include <stdio.h>

int main()
{
 for (int i = 0; i < 10; ++i) {
  int i = 876; //c++'da geçersiz
  printf("%d ", i); //876
 }

 return 0;
}
```

Aşağıdaki bildirim C'de geçerli ve *str* dizisinde tutulan yazının sonunda *null* karakter yok:

```
char str[5] = "Ahmet";
```

Bu bildirim C++ dilinde geçersiz.

***const* nesne bildirimlerine ilişkin farklılıklar:**

- C'de *const* nesnelere ilk değer vermek zorunlu değil. C++ dilinde zorunlu.

- C'de sabit ifadeleri ile ilk değerini almış *const* nesneler sabit ifadesi gereken yerlerde kullanılamıyorlar. C++'da kullanılabiliyorlar.
- C'de global *const* nesneler varsayılan biçimde dış bağlantıya *(external linkage*) aitler. C++ dilinde ise global *const* nesneler varsayılan biçimde iç bağlantıya *(internal linkage)* aitler. C++ dilinde global *const* nesneleri dış bağlantıya sokmak için bunların tanımında "*extern*" anahtar sözcüğünü kullanmak gerekiyor.

C'de bir *enum*, *union* ve *struct* anahtar sözcükleri ile oluşturulan türler söz konusu olduğunda bildirimde kullanılan etiket *(tag)* değişken bildirimlerinde doğrudan kullanılamıyor:

```
struct Point {
 int x, y;
};

union Data {
 char c1;
 char c2;
 int x;
}

enum Color {Red, Blue, Black};

Point p = {1, 3}; // C'de geçersiz C++'da geçerli
Color c = Black; // C'de geçersiz C++'da geçerli
Data data = {'A'}; // C'de geçersiz C++'da geçerli
```

const ve *static* anahtar sözcüklerinin pointer parametre değişkenleriyle kullanımı C'de geçerli C++'ta geçersiz. Aşağıdaki bildirimler C'de geçerli C++'da geçersiz

```
int foo(int p[const]);     // int foo(int *const p)
int bar(char s[static 5]); // uzunluğu en az 5 olan bir yazı adresi bekleniyor.
```

C'de kullanımdan düşmüş olan *auto* anahtar sözcüğünün C++ dilindeki anlamı farklı. Aşağıdaki kod C'de geçerli ancak C++'da geçersiz:

```
void func()
{
 auto int x; //C'de geçerli C++'da geçersiz
 auto int y = 10;  //C'de geçerli C++'da geçersiz
 //...
}
```

auto C++ dilinin en sık kullanılan anahtar sözcüklerinden biri. auto anahtar sözcüğü "Tür çıkarımı" *(type deduction)* denilen araç seti ile ilgili. Tür çıkarımı C++ dilinin en önemli araçlarından biri.

C'de dosya kapsamında *(file scope)* bir nesne birden fazla kez tanımlanabilir. Bu durumda tanımlanan aynı nesnelerdir. (İsimleri aynı olan farklı nesneler değil.) C'de bu duruma "*tentative definition*" denmektedir. C++'da bu durum geçerli değildir. Aşağıdaki kod C'de geçerli C++ dilinde geçersizdir:

```
int a, a, a;
int a;
int a;
```

C'de tür eş isimlerinin ve yapı/birlik/enum etiket *(tag)* isimlerinin aynı olması durumu geçerlidir. C++'da bu durum geçersizdir. Aşağıdaki kod C'de geçerli C++'da geçersizdir:

```
struct S {
 int x;
};

typedef struct U {
 int y;
} S;
```

C'de işlev bildirimlerinde, tanımlamalarında, *sizeof* operatörünün parantezi içinde, tür dönüştürme operatörünün içinde yeni tür bildirimleri yapılabilir. Böyle bildirimlerin hepsi C++ dilinde geçersizdir. Aşağıdaki kod *(Andrey Tarasevich)* C'de geçerli C++'da geçersizdir:

```
enum { G, H, I } foo()
{
  int a = sizeof(enum E { A, B, C }) + (enum X { D, E, F }) 0;
  enum E e = a + B + F;
  return e == 0 ? G : H;
}
```

C'de aynı çeviri biriminde *(translation unit)* tamamlanmamış bir türden *(incomplete type)* bir nesnenin önce tanımlanması daha sonra bu türün bildiriminin yapılması şartıyla geçerlidir.

```
struct S s;          // struct S türü henüz "tamamlanmamis"
struct S { int i; };
```