Cplusplus_Ders_Notlari / **ozel_uye_islevlerin_derleyici_tarafindan_tanimlanmasi.md**

`C++` dilinin en karmaşık ve en fazla öğrenme zorluğu içeren araçlarından biri sınıfların özel işlevleri `(special members)`. `C++11` standartlarına göre sınıfların `6` özel işlevi var. Peki sınıfların bu işlevlerini özel yapan ne? Bu işlevlerin yazılması derleyiciye bırakılabiliyor. Yani yeterli koşullar oluştuğunda derleyici bizim adımıza sınıfın özel işlevlerini yazabiliyor. Önce sınıfların bu özel işlevlerini listeleyelim:

```
class A {
public:
 A(); //varsayılan kurucu işlev (default constructor)
 ~A(); //sonlandırıcı işlev (destructor)
 A(const A &); //kopyalayan kurucu işlev (copy constructor)
 A &operator=(const A &); //kopyalayan atama işlevi (copy assignment operator function)
 A(A &&); //taşıyan kurucu işlev (move constructor)
 A &operator=(A &&); //taşıyan atama işlevi (move assignment operator function)
};
```

Yukarıdaki özel işlevlerden herhangi biri aşağıdaki durumlardan birinde olabilir.

1. Hiç bildirilmeyebilir `(not declared)`. Bu durumda söz konusu işlev yoktur.
2. Derleyici tarafından örtülü olarak bildirilebilir `(implicitly declared)`. Bu durum söz konusu işlevin derleyici tarafından `default` edilmesi ya da `delete` edilmesi ile mümkündür.
3. Programcı tarafından bildirilebilir `(user declared)`. Bu durum söz konusu işlevin programcı tarafından bildirilmesi, tanımlanması, `default` edilmesi ya da `delete` edilmesi ile mümkündür.

Aşağıda `A` isimli bir sınıfın varsayılan kurucu işlevinin `(default constructor)` programcı tarafından bildirilme senaryoları gösteriliyor:

```
//a.h
class A {
public:
 A(); //1
 A(){}; //2
 A() = default; //3
 A() = delete; //4
};
```

- 1 Sınıfın varsayılan kurucu işlevi programcı tarafından sınıf tanımı içinde bildirilmiş ancak işlevin tanımı kod dosyasına bırakılmış.
- 2 Sınıfın varsayılan kurucu işlevi programcı tarafından sınıf tanımı içinde inline olarak tanımlanmış.
- 3 Sınıfın varsayılan kurucu işlevi programcı tarafından default edilmiş.
- 4 Sınıfın varsayılan kurucu işlevi programcı tarafından delete edilmiş.

Bir özel işlevin hiç bildirilmemesi `(not declared)` ile `delete` edilmesi `(deleted)` aynı şey değildir. Bildirilmeyen bir işlev olmayan bir işlevdir, yoktur. Bildirilmeyen işlevler işlev yükleme çözümlemesine `(function overload resolution)` dahil edilmezler. `delete` edilen işlevler ise bildirilmişlerdir; işlev yükleme çözümlemesine dahil edilirler. Ancak `delete` edilmiş bir işleve bağlanmış bir çağrı sentaks hatası oluşturur. Aradaki farkı gösteren bir örnek verelim:

```
class A{
public:
 template <class ...Args>
 A(Args&& ...args);
};
```

Yukarıda `A` isimli bir sınıf için varsayılan kurucu işlev bildirilmemiş. Sınıfta değişken sayıda parametreli `(variadic)` bir kurucu işlev şablonu `(template)` bildirildiğini görüyorsunuz. Varsayılan kurucu işlevin çağrılması gereken bir durumda, derleyici bu şablondan hareketle varsayılan kurucu işlevin kodunu yazar. Şimdi de aşağıdaki tanıma bakalım:

```
class A{
public:
 template <class ...Args>
 A(Args&& ...args);
 A() = delete;
};
```

Burada ise `A` sınıfının varsayılan kurucu işlevi `delete` edilmiş. Bu işlev var ve işlev yüklenmesi çözümlenmesine dahil edilir. Ancak bu işleve yapılan bir çağrı sentaks hatası olarak kabul edilir. Bu durumda `A` sınıfı türünden bir nesnenin varsayılan şekilde hayata getirilmesi artık mümkün değildir.

Peki hangi durumlarda özel işlevler derleyici tarafından örtülü `(implicit)` olarak bildirilirler? Bir sınıf için ne bir özel işlev ne de bir (özel olmayan) kurucu işlev bildirilmiş ise sınıfın tüm özel işlevleri `default` edilmiş olur. Yani

```
class A{
public:
};
```

gibi sınıf tanımı ile

```
class A {
public:
 A() = default;
 ~A() = default;
 A(const A &) = default;
 A &operator=(const A &) = default;
 A(A &&) = default;
 A &operator=(A &&) = default;
};
```

Yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir. Bir başka deyişle bu durumda sınıfın tüm özel işlevleri derleyici tarafından örtülü olarak bildirilir.

Eğer programcı tarafından sınıf için özel olmayan bir kurucu işlev bildirilirse bu durumda derleyici, varsayılan kurucu işlev dışında tüm işlevleri `default` eder.

```
class A{
public:
    A(int);
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 A(int);
 ~A() = default;
 A(const A &) = default;
 A &operator=(const A &) = default;
 A(A &&) = default;
 A &operator=(A &&) = default;
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir.

Programcı tarafından bir varsayılan kurucu işlevin bildirilmesi durumunda yine diğer tüm özel işlevler derleyici tarafından `default` edilir.

```
class A{
public:
    A();
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 A();
 ~A() = default;
 A(const A &) = default;
 A &operator=(const A &) = default;
 A(A &&) = default;
 A &operator=(A &&) = default;
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir.

Eğer programcı tarafından sınıfın sonlandırıcı işlevi bildirilirse derleyici sınıfın taşıma işlevleri `(move members)` dışındaki tüm özel işlevlerini `default` eder.

```
class A{
public:
    ~A();
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 A() = default;
 ~A();
 A(const A &) = default;
 A &operator=(const A &) = default;
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir. Bu durumda sınıfın taşıma işlevleri bildirilmemiş durumdadır. Aslına bakılırsa sonlandırıcı işlevin bildirilmesi durumunda derleyicinin sınıfın kopyalayan işlevlerini `default` etmesi doğru sayılmamalıdır. Eğer programcı sınıfın sonlandırıcı işlevini bildirmişse muhtemelen sınıfın kopyalayan işlevlerinin de programcı tarafından yazılması için bir gereklilik söz konusudur. Bu yüzden bu durum `C++11` standartlları tarafından eskimiş `(deprecated)` olarak değerlendirilmektedir. Gelecekteki standartlarda bu kuralın değiştirilmesi gündemdedir. Yani programcı tarafından bir sonlandırıcı işlevin bildirilmesi durumunda gelecekteki standartlara göre, derleyici ileride bu durumda sınıfın kopyalayan işlevlerini de `default` etmeyecektir.

Eğer programcı tarafından sınıfın kopyalayan kurucu işlevi bildirirse derleyici sınıfın kopyalayan atama işlevini ve sonlandırıcı işlevlerini `default` eder. Bu durumda sınıfın varsayılan kurucu işlevi ve taşıyan işlevleri bildirilmemiş durumdadır.

```
class A{
public:
    A(const A &);
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 ~A() = default;
 A(const A &);
 A &operator=(const A &) = default;
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir. Eğer programcı tarafından sınıfın kopyalayan atama işlevi bildirilirse derleyici sınıfın varsayılan kurucu işlevini, kopyalayan kurucu işlevini ve sınıfın sonlandırıcı işlevini `default` eder. Bu durumda sınıfın ve taşıyan işlevleri yine bildirilmemiş durumdadır.

```
class A{
public:
    A &operator=(const A &);
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 A() = default;
 ~A() = default;
 A(const A&) = default;
 A& operator=(const A&);
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir.

Eğer programcı tarafından sınıfın taşıyan kurucu işlevi bildirilirse derleyici yalnızca sınıfın sonlandırıcı işlevini

`default` eder. Bu durumda sınıfın kopyalayan işlevleri `(copy members)` derleyici tarafından `delete` edilir. Sınıfın kurucu işlevi ve sınıfın taşıyan atama işlevi ise bildirilmemiş durumdadır.

```
class A {
public:
 A(A &&);
};
```

gibi bir sınıf tanımı ile

```
class A {
public:
 ~A() = default;
 A(const A &) = delete;
 A &operator=(const A &) = delete;
 A(A &&);
};
```

yukarıdaki gibi bir sınıf tanımı birbirine eşdeğerdir. Eğer programcı tarafından sınıfın taşıyan atama işlevi bildirilirse derleyici sınıfın varsayılan kurucu işlevini ve sınıfın sonlandırıcı işlevini `default` eder. Bu durumda yine sınıfın kopyalayan işlevleri derleyici tarafından `delete` edilir. Sınıfın taşıyan kurucu işlevi ise bildirilmemiş durumdadır.

```
class A {
public:
 A &operator=(A &&);
};
```

gibi bir sınıf tanımıyla

```
class A {
public:
 A() = default;
 ~A() = default;
 A(const A &) = delete;
 A &operator=(const A &) = delete;
 A &operator=(A &&);
};
```

yukarıdaki gibi bir sınıf tanımı eşdeğerdir. Nesneleri taşınabilir fakat kopyalanamaz bir sınıf oluşturmak için sınıfın taşıyan işlevlerini bildirmemiz yeterlidir. Bu durumda derleyici sınıfın kopyalama işlevlerini `delete` edeceğinden sınıf nesnelerinin kopyalanmasına neden olan durumlarda sentaks hatası oluşur.

Buraya kadar derleyicinin hangi koşullarda sınıfın özel işlevlerini içsel olarak oluşturduğunu inceledik. Peki ya derleyicinin bu özel işlevleri oluşturmasını engelleyen bir durum söz konusu ise ne olacak? Bu durumda derleyici yazması gereken işlevi `delete` eder. Yani `default` edilmiş bir özel işlev derleyicinin bu işlevi yazmaması durumunda derleyicinin `delete` ettiği bir işleve dönüşür. Derleyicinin özel bir işlevi yazamaması farklı nedenlerle söz konusu olabilir:

Derleyicinin sınıfın varsayılan kurucu işlevini `delete` etmesi

- Derleyicinin varsayılan kurucu işlevi yazması sürecinde, sınıfın başka bir sınıf türünden öğesinin ya da taban sınıf alt nesnesinin varsayılan kurucu işlevine yaptığı çağrı, bu işlevin `delete` edilmiş olması ya da bu işlevin ilgili sınıfın `private` bölümünde olması nedeniyle erişilmez durumda olması ya da bu işlevin var olmaması nedeniyle geçersiz durumuna düşüyorsa derleyici yazması gereken varsayılan kurucu işlevi `delete` eder.
- Sınıfın başka sınıf türünden `static` olmayan bir veri öğesinin ya da sınıfın taban sınıfının sonlandırıcı işlevi `delete` edilmiş ise ya da erişilemez durumda ise derleyici yazması gereken varsayılan kurucu işlevi `delete`

eder.

- Yine sınıfın `static` olmayan bir veri öğesinin `const` ya da referans olması durumunda, bu öğeye sınıf içinde ilk değer verilmemiş ise derleyici yazması gereken varsayılan kurucu işlevi `delete` eder.

Derleyicinin sınıfın sonlandırıcı işlevini `delete` etmesi

- Derleyicinin bir sınıf için sonlandırıcı işlevi yazması sürecinde, sınıfın başka sınıf türünden static olmayan bir veri öğesinin ya da sınıfın taban sınıfının sonlandırıcı işlevine yaptığı çağrı, söz konusu işlevin `delete` edilmiş olması ya da erişilemez durumda olması yüzünden sentaks hatası durumuna düşüyor ise, derleyici yazması gereken sonlandırıcı işlevi `delete` eder.

Derleyicinin sınıfın kopyalayan kurucu işlevini `delete` etmesi

- Derleyicinin kopyalayan kurucu işlevi yazması sürecinde, sınıfın başka bir sınıf türünden static olmayan bir veri öğesinin ya da taban sınıf alt nesnesinin kopyalayan kurucu işlevine yaptığı çağrı, bu işlevin `delete` edilmiş olması ya da bu işlevin ilgili sınıfın `private` bölümünde olması nedeniyle erişilmez durumda olması nedeniyle geçersiz durumuna düşüyorsa derleyici yazması gereken kopyalayan kurucu işlevi `delete` eder.
- Sınıfın başka sınıf türünden static olmayan bir veri öğesinin ya da sınıfın taban sınıfının sonlandırıcı işlevi delete edilmiş ise ya da erişilemez durumda ise derleyici yazması gereken kopyalayan kurucu işlevi `delete` eder.

Derleyicinin sınıfın kopyalayan atama işlevini `delete` etmesi

- Derleyicinin kopyalayan atama işlevini yazması sürecinde, sınıfın başka bir sınıf türünden öğesinin ya da taban sınıf alt nesnesinin kopyalayan atama işlevine yaptığı çağrı, bu işlevin `delete` edilmiş olması ya da bu işlevin ilgili sınıfın `private` öğesi olması yüzünden geçersiz durumuna düşüyorsa derleyici yazması gereken kopyalayan atama işlevini delete eder.
- Sınıfın static olmayan bir veri öğesinin `const` ya da referans olması durumunda, derleyici yazması gereken kopyalayan atama işlevini delete eder.

Derleyicinin sınıfın taşıyan kurucu işlevini `delete` etmesi

- Derleyicinin sınıfın taşıyan kurucu işlevini yazması sürecinde, sınıfın başka bir sınıf türünden static olmayan bir veri öğesinin ya da taban sınıf alt nesnesinin taşıyan kurucu işlevine yaptığı çağrı, söz konusu işlevin `delete` edilmiş olması ya da söz konusu işlevin ilgili sınıfın private bölümünde olması nedeniyle erişilmez durumda olması ya da bu işlevin var olmaması nedeniyle geçersiz durumuna düşüyorsa, derleyici yazması gereken taşıyan kurucu işlevi `delete` eder. Derleyici tarafından `delete` edilmiş sınıfın taşıyan kurucu işlevi yüklenmiş işlev çözümlenmesi sürecine katılmaz, yani bildirilmemiş olarak ele alınır.

Derleyicinin sınıfın taşıyan atama işlevini `delete` etmesi

- Derleyicinin sınıfın taşıyan atama işlevini yazması sürecinde, sınıfın başka bir sınıf türünden öğesinin ya da taban sınıf alt nesnesinin taşıyan atama işlevine yaptığı çağrı, söz konusu işlevin delete edilmiş olması ya da söz konusu işlevin ilgili sınıfın `private` öğesi olması yüzünden geçersiz durumuna düşüyorsa derleyici yazması gereken taşıyan atama işlevini `delete` eder.
- Sınıfın static olmayan bir veri öğesinin const ya da referans olması durumunda, derleyici yazması gereken taşıyan atama işlevini `delete` eder. Derleyici tarafından `delete` edilmiş sınıfın taşıyan atama işlevi yüklenmiş işlev çözümlenmesi sürecine katılmaz, yani bildirilmemiş olarak ele alınır.

Derleyicinin özel işlevleri `delete` etmesine örnekler verelim:

```
class Member {
 Member();
};


class A {
 Member m;
};


int main()
{
 A a;
}
```

Yukarıdaki kodda `A` sınıfının `Member` sınıfı türünden bir öğeye sahip olduğunu görüyorsunuz. `Member` sınıfının varsayılan kurucu işlevi `Member` sınıfının `private` bölümünde bildirilmiş. `main` işlevi içinde `A` sınıfı türünden bir nesne tanımlanıyor. Derleyicinin bu durumda `A` sınıfının varsayılan kurucu işlevini içsel olarak tanımlaması gerekiyor. Bu işlevin tanımında derleyicinin m veri öğesi için `Member` sınıfının `private` varsayılan kurucu işlevine çağrı yapması gerekiyor. Bu durum geçersiz olduğu için derleyici `A` sınıfının varsayılan kurucu işlevini `delete` eder. Yukarıdaki kod için kullandığım derleyicinin verdiği hata iletisi şu oldu:

```
A::A()' is implicitly deleted because the default definition would be ill-formed:
```

Şimdi de aşağıdaki koda bakalım:

```
class A {
 const int mcx = 0;
 //
};

int main()
{
 A a1, a2;
 a1 = a2;
}
```

`A` sınıfının `mcx` isimli veri öğesinin `const` olarak bildirildiğini görüyorsunuz. main işlevi içinde iki `A` nesnesi birbirine atanıyor. Derleyicinin yazması gereken kopyalayan atama operatör işlevi `const` bir veri öğesine atama yapmak geçerli olmadığından derleyici tarafından `delete` edilir. Yukarıdaki kod için kullandığım derleyicinin hata iletisi şöyleydi:

```
error: use of deleted function 'A& A::operator=(const A&)'
```

Sınıfın taşıyan kurucu işlevi ya da taşıyan atama işlevi derleyici tarafından yazılamadığı için `delete` edilirse hiç bildirilmemiş olarak ele alınır. Aşağıdaki koda bakalım:

```
#include <iostream>

using namespace std;

class Member {
 Member(Member &&){}
public:
 Member(){}
 Member(const Member &){cout << "Member Copy Ctor";}
};

class A {
 Member m;
};

int main()
{
 A a1;
 A a2(move(a1));
}
```

`main` işlevi içinde tanımlanan `A` sınıfı türünden `a2` isimli nesneye bir sağ tarafı değeri ifadesi ile ilk değer veriliyor. Bu durumda `a2` nesnesi için sınıfın taşıyan kurucu işlevinin çağrılması gerekiyor. Derleyici tarafından yazılacak olan `A` sınıfının taşıyan kurucu işlevi `A` sınıfının veri öğesi olan Member türünden `m`'nin taşınamaması yüzünden `delete` ediliyor. Bu durumda derleyici tarafından `delete` edilmiş `A` sınıfının taşıyan kurucu işlevi bildirilmemiş olarak ele alınıyor ve `a2` sınıf nesnesi için sınıfın kopyalayan kurucu işlevi işlevi çağrılıyor. Özel işlevler hakkında bir noktayı daha vurgulayalım: Sınıfın taşıyan işlevleri olmayabilir ama bir sınıfın kopyalayan işlevleri her zaman vardır. Bir sınıfın kopyalayan işlevleri ya programcı tarafından bildirilir ya derleyici tarafından `default` edilir ya da derleyici tarafından `delete` edilir.

Kaynak: Everything You Ever Wanted to Know About Move Semantics, Howard Hinnant, Accu 2014