Sınıfların Özel Üye Fonksiyonları(Special Member Functions)

Bildiğimiz gibi sınıfların fonksiyonlarına üye fonksiyon denilmektedir. Her sınıfa ait özel üye fonksiyonlar bulunmaktadır. Bu fonksiyonların özel olmasının sebebi aslında çeşitli koşullar sağlandığı takdirde bu fonksiyonların derleyici tarafından yazılıyor olmasından kaynaklıdır. Bu dokümanımızda bu fonksiyonların hangi durumlarda derleyici tarafından yazıldığını ve hangi durumlar delete edildiği gibi pek çok değişik durumu inceleyeceğiz.

Bunlardan en popüleri Java’da da bulunan default Constructor’dır. Bu elemanın ne önemli özelliği herhangi bir parametre almamasıdır. Diğer bir önemli tarafları ise bu fonksiyonun non-static olması ve geri dönğş değeri kavramına sahip olmamasıdır. Geri dönüş değeri kavramına sahip olmamakla geri dönüş değeri olmamak aynı şey demek değildir. (Geri dönüş değeri olmayan fonksiyonlarda void anahtar sözcüğü kullanılırken geri dönüş değeri kavramı olmayan fonksiyonlarda void yazılacak kısım boş bırakılır.) Constructorların hepsi bu şekilde iplemende edilir. Bu özel üye fonksiyonun içerisine çoğunlukla bir şey yazılmaz. Sınıfın veri elemanlarına default olarak ilk değer verilmesi isteniyorsa bu durumda ilerde ele alıcağımız constructor initilazer list kullanılır.

Default constructor aşağıdaki örnekte gösterildiği gibi tanımlanabilir :

#include <iostream>

class Myclass {

public:

Myclass()

{

std::cout << "Myclass Default ctor" << '\n';

}

};

main içerisinde eğer bir nesne oluşturuluyorsa bu aslında bir constructor çağrısıdır. Programcı constructorları doğrudan çağıramaz. Bunun nedeni bu fonksiyonların non-static olmasından kaynaklanmaktadır.

Aşağıdaki örnekte bir sınıf nesnesi oluşturulduğu için bu durumda default constructor çağrılacaktır:

#include <iostream>

class Myclass {

public:

Myclass()

{

std::cout << "Myclass Default ctor" << '\n';

}

};

int main()

{

Myclass m;

}

Yani son durumda ekrana default constructorın çağrıldığına dair bir yazı ekrana basılacaktır.

Anahtar Notlar: Programcı eğer herhangi bir constructorı kendi yazarsa bu durumda derleyici default contructorı yazmaz. Aşağıdaki örnekte sentaks hatası olmasının nedeni budur:

#include <iostream>

class Myclass {

public:

Myclass(int)

{

std::cout << "Myclass (int) ctor" << '\n';

}

};

int main()

{

Myclass m; //error

}

Yukarıdaki sentaks hatası aşağıdaki gibi giderilebilir:

#include <iostream>

class Myclass {

public:

Myclass(int)

{

std::cout << "Myclass (int) ctor" << '\n';

}

};

int main()

{

Myclass m{10}; //legal

}

Çünkü burada çağrılan default constructor değil int parametreli constructor olacaktır. Fonksiyonlarda sıkça kullanılan iki tane anahtar sözcük bulunmaktadır. Bunlardan biri default anahtar sözcüğüdür ve kullanımı şu şekildedir:

class Myclass {

public:

Myclass() = default;

};

Burada kısaca default anahtar sözcüğünün yaptığı iş “emrediyorum derleyici default constructorı yaz.” Anlamına gelmektedir. Bu şekilde tanımlamaya default declared denilmektedir. Derleyici tarafından yazılmasına ise implicitly declared denilmektedir.

Sık kullanılan diğer bir anahtar sözcük ise delete anahtar sözcüğüdür. Bu anahtar sözcük adından da anlaşılabileceği gibi fonksiyonun silinmesi sağlar. Burada dikkat edilmesi gereken taraf delete edilmiş olan fonksiyonların çağrılmasıyla bir fonksiyonun name look-up ile aranıp bulunamaması arasından fark vardır. Örneğin aşağıdaki örnekte fonksiyon bulunamadı değil, fonksiyon bulundu ancak delete edilmiş hatası alınacaktır:

void func(char);

void func(int);

void func(double) = delete;

void func(long double);

int main()

{

func(2.5);

}

Bazı koşullar sağlandığında özel üye fonksiyonların bazıları da delete edilmektedir. Örneğin eğer özel üye fonksiyonun implicilty declared olması durumunda bir sentaks hatası oluşması durumuna karşın özel üye fonksiyonlar derleyici tarafından delete edilmektedir. Örneğin aşağıdaki örnekte sentaks hatasının nedeni const bir değişkene ilk değer verilmemesinden değil, Myclass sınıfının constructorının delete edilmesinden kaynaklanmaktadır. Yani bu örnekte m nesnesi hiç tanımlanmasa bir sentaks hatası oluşmayacaktı:

#include <iostream>

class Myclass {

public:

private:

const int mx;

};

int main()

{

Myclass m; //error

}

Aşağıdaki örnekte B sınıfının default ctoru her ne kadar default olarak bildiriliyormuş gibi görünse de burada delete edilmiştir. Bu yüzden B bx ifadesi delete edilmiş bir fonksiyona çağrı yapılma girişiminden kaynaklı olarak sentaks hatasına neden olur. Yani burada main içerisinde B türünden bir nesne tanımlanmasaydı, bir sentaks hatası oluşmazdı. Bunun nedeni derleyicinin kendini sentaks hatasından koruma girişimidir.

class A{

private:

A(int);

};

class B{

public:

B()=default;

private:

A ax;

};

int main()

{

B bx; //error

}

C++ dilinde bir sınıf nesnesi hayata aynı türden bir başka sınıf nesnesinin değerini alarak getirilmek istendiğinde sınıfın özel üye fonksiyonu olan copy constructor çağrılır. Örneğin aşağıdaki örnekte bir copy constructor çağrısı söz konusudur.

class Myclass{

}

int main()

{

Myclass m1;

Myclass m2 = m1;//m2 için copy ctor çağrılacak.

}

Copy constructor özel üye fonksiyonunun tek bir parametresi vardır. Bu parametre de const sınıf türünden referanstır. Copy constructor özel üye fonksiyonunun tanımlanması aşağıdaki gibi yapılabilir:

#include <iostream>

class Myclass

{

public:

Myclass() = default;

Myclass(const Myclass&)

{

std::cout << "Myclass copy ctor" << '\n';

}

};

int main()

{

Myclass m1;

Myclass m2 = m1;

}

Dikkat edilmesi gereken bir kısım eğer herhangi bir constructor tanımlanırsa bu durumda default constructor derleyici tarafından yazılmaz. Bu duruma copy constructor da dahildir. Örneğin yukarıdaki örnekte default constructor tanımlanmasaydı m1 nesnesi tanımlamasından bir sentaks hatası olacaktır.

Destructor özel üye fonksiyonu ise sınıfın sonlandırıcı işlemi ya da diğer bir değişle sınıfın azraili denilebilir. Bu fonksiyonun sınıf nesnesinin ömrü bittiği zaman mutlaka çağrılması gerekmektedir. Aksi durumda bir tanımsız davranış oluşur. Elbette sınıfın destructorının ne zaman çağrılacağıyla ilgili durumlar değişiklik göstermektedir. Örneğin aşağıdaki gibi bir nesnede destructor main devam ediyor yazısından sonra basılacaktır.

#include <iostream>

class Myclass

{

public:

Myclass()

{

std::cout << "Myclass ctor" << '\n';

}

~Myclass()

{

std::cout << "Myclass dtor" << '\n';

}

};

int main()

{

Myclass m1;

std::cout << "Main devam ediyor\n";

}

Ancak bu nesne bir blok içerisinde tanımlansaydı o blok içerisinde hayata başlayacağından dolayı main devam ediyor yazısı bu kez destructor çağrısından sonra ekrana basılacaktır.

#include <iostream>

class Myclass

{

public:

Myclass()

{

std::cout << "Myclass ctor" << '\n';

}

~Myclass()

{

std::cout << "Myclass dtor" << '\n';

}

};

int main()

{

{

Myclass m1;

}

std::cout << "Main devam ediyor\n";

}

Ancak bu değişken eğer static anahtar sözcüğü ile tanımlansaydı bu durumda static ömürlü olduğu için main devam ediyor yazısından sonra destructor çağrısı gerçekleştirilecektir.

#include <iostream>

class Myclass

{

public:

Myclass()

{

std::cout << "Myclass ctor" << '\n';

}

~Myclass()

{

std::cout << "Myclass dtor" << '\n';

}

};

int main()

{

{

static Myclass m1;

}

std::cout << "Main devam ediyor\n";

}

Anahtar Notlar: Sınıfların başka bir sınıf türünden de olabilir bu durumda yaratılan sınıf nesnesinin constructorı söz konusu içerisinde tanımlanan sınıf nesnesinin ctorundan sonra çağrılır. Unutmamak gereken bir kısım da contructorı önce çağrılanın her daim destructorı sonra çağrılacaktır.

#include <iostream>

class A{

public:

A()

{

std::cout << "A default ctor" << '\n';

}

~A()

{

std::cout << "A destructor" << '\n';

}

};

class B

{

public:

B()

{

std::cout << "B default ctor" << '\n';

}

~B()

{

std::cout << "B destructor" << '\n';

}

private:

A mx;

};

int main()

{

B bx;

}

Yukarıdaki örnekte bx nesnesi tanımlandığında öncelikle A sınıfının ctoru çağrılacaktır. Çünkü burada da görüldüğü gibi B sınıfının A sınıfı türünden bir veri elemanı vardır. A sınıfının ctorundan sonra ise B sınıfının ctoru çağrılacaktır. Sonrasında ise B sınıfının destructorı ve en son da A sınıfının destructorı çağrılacaktır.

Copy Assignment fonksiyonu ise bir sınıf türünden bir nesnesin başka bir nesneye atanması ile çağrılır. Ancak bu durumun bir initialization olmaması gerekir. Çünkü bu durumda copy assignment özel üye fonksiyonu değil copy constructor özel üye fonksiyonu çağrılırdı. Örneğin:

#include <iostream>

class Myclass

{

public:

Myclass& operator=(const Myclass&)

{

std::cout << "Myclass copy assignment" << '\n';

return \*this;

}

};

int main()

{

Myclass m1;

Myclass m2;

m2 = m1; //copy assignment çağrılacak.

}

Anahtar Notlar: copy constructor veya copy assignment fonksiyonları eğer kullanıcı tarafından yazılmış ise bu durumda move constructor ve move assignment fonksiyonları derleyici tarafından yazılmaz.

#include <iostream>

class Myclass

{

public:

Myclass& operator=(const Myclass&)

{

std::cout << "Myclass copy assignment" << '\n';

return \*this;

}

};

int main()

{

Myclass m1;

Myclass m2;

m2 = m1;

Myclass m3;

m3 = std::move(m2);//\*\*

}

Örneğin normalde \*\* ile belirtilmiş ifadede move assignment çağrılması gerekmektedir ancak move assignment fonksiyonu not declared olduğu için burada copy assignment çağrıldı. Kısaca eğer copy elemanlarından herhangi biri kullanıcı tarafından yazılsaydı, bu durumda move elemanları not declared olacaktır.

Sınıfın move özel üye fonksiyonları iki tanedir. Bunlar sırasıyla move constructor ve move assignment fonksiyonlarıdır. Bu fonksiyonlar std::move standart fonksiyonu kullanımı olduğu zaman çağrılmaktadır. Örneğin move constructor özel üye fonksiyonu aşağıdaki örnekte gösterildiği gibi çağrılır:

#include <iostream>

class Myclass

{

public:

Myclass() = default;

Myclass(Myclass&&)

{

std::cout << "Myclass move ctor" << '\n';

}

};

int main()

{

Myclass m1;

Myclass m2 = std::move(m1);

}

Burada da görüldüğü gibi eğer initialization işlemi olduğu takdirde bu durumda move ctor çağrılacaktır. Ancak eğer aşağıdaki örnekte gösterildiği gibi bir initialization yapılmasaydı bu durumda sınıfın move assignment özel üye fonksiyonu çağrılacaktı:

#include <iostream>

class Myclass

{

public:

Myclass() = default;

Myclass&& operator=(Myclass&&)

{

std::cout << "Myclass move assignment" << '\n';

return std::move(\*this);

}

};

int main()

{

Myclass m1;

Myclass m2;

m2 = std::move(m1);

}

Daha önceden de hatırlayabileceğimiz gibi sınıfın herhangi bir ctoru eğer programcı tarafından yazılmışsa bu durumda sınıfın default ctoru not declared olucaktır. Örneğin aşağıdaki kodda bu sebepten bir sentaks hatası oluşur:

#include <iostream>

class Myclass

{

public:

Myclass(Myclass&&)

{

std::cout << "Myclass move ctor" << '\n';

}

};

int main()

{

Myclass m1;//error

}

Sentaks hatasının nedeni default ctorun derleyici tarafından yazılmamış olmasından kaynaklanmaktadır.

Copy elemanlarından herhangi birinin bildirilmesi durumunda sınıfın move elemanları not declared olmaktaydı. Diğer taraftan eğer sınıfın herhangi bir move elemanı programcı tarafından tanımlanırsa bu durumda sınıfın copy elemanları delete edilir. Örneğin:

#include <iostream>

class Myclass

{

public:

Myclass() = default;

Myclass&& operator=(Myclass&&)

{

std::cout << "Myclass move assignment" << '\n';

return std::move(\*this);

}

};

int main()

{

Myclass m1;

Myclass m2;

m2 = m1;

}

Buradaki sentaks hatasının nedeni silinmiş bir fonksiyona çağrı yapma girişiminden kaynaklanmaktadır.

Kısa bir özel geçmek gerekirse sınıfın özel üye fonksiyonları bazı koşullar sağlandığı zaman derleyici tarafından yazılan üye fonksiyonlardır. Eğer programcı herhangi bir ctor yazarsa bu durumda default ctor derleyici tarafından yazılmaz. Unutulmaması gereken bir kısım ise ne olursa olsun destructor özel üye fonksiyonunun ne olursa olsun derleyici tarafından yazılacağıdır.

Move ve copy özel üye fonksiyonlarını özetlemek gerekirse programcı eğer copy özel üye fonksiyonlarından birini kendi yazarsa bu durumda move elemanları not declared olur. Diğer taraftan eğer sınıfın move elemanlarından herhangi biri programcı tarafından yazılırsa bu durumda ise sınıfın copy özel üye fonksiyonları delete edilecektir.

Not: Bu kısa özet dokümanın sonundaki tabloda da gösterilmiştir.

Son olarak özel fonksiyonları sıralamak gerekirse:

1-) Default Constructor

2-) Destructor

3-) Copy Constructor

4-) Copy Assignment

5-) Move Constructor

6-) Move Assignment

Özel üye fonksiyonların bildirilmesi, bildirilmemesine veya delete edilip edilmemesine bağlı olarak delete edilmesinden ve bildirilmemesinden bahseden tablo aşağıdaki gibidir:

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu tablo aslında tüm konunun bir özeti olarak da görülebilir.