**İŞLEV YÜKLEMESİ**

**(FUNCTION OVERLOADING)**

Nesne yönelimli programlamada iki çeşit Polimorfizm(Çok biçimlilik) vardır. Bunlar:

1-) Runtime Polimorfzim(Çalışma Zamanı Çok Biçimliliği)

2-) Compile-Time Polimorfzim(Derleme Zamanı Çok Biçimliliği)

Çalışma zamanı çok biçimliliği function override ile mümkün olabilir. Çünkü hangi fonksiyonun çağrılacağı bu durumda çalışma zamanında belli olur. Function Overriding terimini ilerde ele alacağınız. Derleme zamanı çok biçimliliği ise function overloading(işlev yüklemesi) ile mümkün olabilir. Function Overloading terimi çoğu kitapta fonksiyonların aşırı yüklenmesi olarak geçmektedir ve bu durumdan olumsuz bir anlam çıkmaktadır oysa bu durum Nesne yönelimli programlamanın en temel konseptlerinden biridir. Bu yüzden terimin karşılığı olarak işlev yüklemesi terimi daha uygundur.

**Anahtar Notlar:** Function overloading'in programın çalışma zamanına herhangi bir maliyeti yoktur. Overload resolution'a göre aday metotlara bakılıp uygun fonksiyon çağrılır. Eğer fonksiyonların ikisi de aynı uygunluktaysa bu durumda ambiguity(iki anlamlılık) oluşur.

İşlev yüklemesi ile ilgili dikkat edilmesi gereken en önemli durumlardan biri işlev yüklemesi var mı sorusunun önce sorulmasındadır çünkü bu soruya cevap vermeden x şekilde çağrı yaparsam bu fonksiyon çağrılır demek oldukça mantıksızdır. Peki bir kodda işlev yüklemesinden söz edilebilmesi için neler gerekir?

1-) Fonksiyonların aynı isme sahip olması gerekmektedir.

2-) Fonksiyonların aynı faaliyet alanı içerisinde olması gerekmektedir.

3-) Fonksiyonların imzaları birbirinden farklı olmalıdır. Fonksiyonların imzalarını fonksiyonların geri dönüş değeri dışındaki parametrik yapısı etkiler.

Aşağıdaki örnekte işlev yüklemesinden bahsedilemez çünkü foo fonksiyonların faaliyet alanları farklıdır.

class Myclass

{

public:

static void foo(int)

{

}

};

void foo(double)

{

}

int main() {

}

Aşağıdaki örnekte bir işlev yüklemesi söz konusu değildir. Bu durum doğrudan syntax hatasıdır. Fonksiyonun geri dönüş değerinin değişmesi imzasının değişmesine sebep olmaz.

void foo();

double foo();

Aşağıdaki örnekte gene bir işlev yüklemesinden bahsedilemez. Bu bir işlev yüklemesi değil bir yeniden bildirimdir(redeclaration).

typedef int Word;

void foo(Word);

void foo(int);

**Anahtar Notlar:** iki tane bildirimin işlev yüklemesi olup olmadığını anlamanın en kolay yolu iki fonksiyona da gövde yazmaktır. Eğer iki fonksiyona da gövde yazılırsa ve bir syntax hatasına neden olmazsa bu durumda ancak overloading’den bahsedilebilir.

Aşağıdaki örnekte 3 tane overload vardır.

void foo(char);

void foo(unsigned char);

void foo(signed char);

**Anahtar Notlar:** Fonksiyonların varsayılan argüman alması imzasının değişmesine engel değildir. Bu sebeple aşağıdaki örnekte işlev yüklemesinden bahsedilebilir.

void func(int x,int y=5);

void func(int x);

Aşağıdaki örnekte işlev yüklemesi değil bir redeclaration söz konusudur. İleride ele alıcağımız const overloading bu durumdan farklıdır.

void bar(int x);

void bar(const int x);

Aşağıdaki örnekte ise bir işlev yüklemesi söz konusudur. Buna const overloading denilmektedir ve oldukça yaygın şekilde kullanılmaktadır.

void bar(int\* x);

void bar(const int\* x);

Aşağıdaki örnekte 2 adet overloading vardır. 1 nolu zar fonksiyonu 3 nolu zar fonksiyonunun overloadıdır. Ancak 3 nolu fonksiyon 2 nolu fonksiyonun redeclationıdır.

void zar(const int\* ptr);//1

void zar(int\* const ptr); //2

void zar(int\* ptr); //3

Aynı Pointerlarda olduğu gibi referanslarda da const overloading yapılabilir. Aşağıdaki buna bir örnektir. Bu yüzden aşağıdaki örnek bir işlev yüklemesidir.

void tar(int& r);

void tar(const int& r);

Aşağıdaki örneğin işlev yüklemesi olup olmaması derleyiciye bağlıdır. Eğer 64 bitlik bir derleyici kullanılıyorsa bu bir redeclationdır ancak 32 bitlik bir derleyici için bu örnek bir işlev yüklemesi olabilir.

void func(int32\_t);

void func(int);

Aşağıdaki örnekte ise derleyici bağlı bir durumdan bahsedilemez çünkü biri 32 bitlik int türü diğeri ise 64 bitlik int türüdür bu yüzden bu örnek doğrudan bir işlev yüklemesidir.

void foo(int32\_t);

void foo(int64\_t);

int main(){

int a=10;

foo(a);

return 0;

}

Function Overload Resolution kuralları:

1-) Overload resolution kurallarına göre öncelikle aday fonksiyonlar(candicate) belirlenir. Bir fonksiyonun aday fonksiyon olması için uygun isme sahip olması yeterlidir.

2-) Aday fonksiyonlar arasından overload resolution kurallarına göre çağrıda bulunan argüman sayısının aday fonksiyonlarda herhangi birine uygun olup olmadığı test edilir. Eğer uygun fonksiyon bulunamazsa bu duruma no match adı verilir.

3-) Birden çok uygun metot olması durumunda uygunluk durumları bazı kurallara göre değerlendirilir eğer ikisi de eşit uygunluktaysa bu durumda ambiguity oluşur.

Function Overload Resolutionda Fonksiyonların Uygunluk Sıralaması:

1-)Exact Match: Bu durum tam uyum olarak adlandırılmaktadır ve uygunluk olarak en üstün olan durumdur. Bu durumda fonksiyona gönderilen argümanların türleriyle fonksiyonun parametre değişkenlerinin türleri tam bir uyum içerisindedir.

2-)Promotion : Bu durum terfih olarak adlandırılır ve uygunluk olarak exact matchten sonra seçilebilirlik anlamında en üstün durum budur. Bu durumun oluşabilmesi için iki farklı durum söz konusudur. Bunlardan biri integral promotiondır. Bunda int altı türler olan bool short gibi türlerin int yükseltilmesi durumudur. Diğer promotion durumu ise float türünden double türüne yükseltme durumdur.

3-)Standart Conversion : Bu durumda fonksiyonun çağrıldığı argümanlarla fonksiyonun parametre değişkenleri birbirine dönüşümü geçerli olan(Promotion olmayan) türler için bu durum geçerlidir.

4-)Variadic Parametreli Fonksiyonlar: Variadic parametreli fonksiyonlar diğerlerine göre uygunluk bakımından en kötü olan çağrılardır. Bildiğimiz gibi bu fonksiyonlar sayısız parametre ile çağrılabilir.