25 Nisan 2022

inline expansion: bir optimizasyon tekniğidir
inline bildirilmesine rağmen inline expand edilebilir veya edilmeyebilir. Garanti yok .

eğer bir başlık dosyasındaki tüm fonksiyonlar, o başlık dosyasında inline olarak tanımlanmışlar ise bu durumda bu başlık dosyasına eşlik eden bir cpp dosyası olmadan kütüphanelere header-only library denir. Fakat gizlimiz saklımız kalmıyor.

inline sözcüğü olmasa da üye fonksiyon implicitly inline'dır.

inline variables: birden fazla kaynak dosyada aynı varlığa işaret eder. Tek bir değişken kabul edilir. ODR ihlal edilmez.

C++ kaynak dosyalarından C'de derlenmiş fonksiyonları çağırmak:
extern "C" int sum_square(int, int);

macrolar:
__cplusplus : derleyici hangi standartları desteklediğini gösterir

condition compiling:
```
#ifdef __cplusplus
    extern "C" {
#endif
```

bunları özel yapan bu fonksiyonları biz tanımlamasak da derleyici belirli koşullar oluşunca derleyici bizim yerimize bu fonksiyonların kodunu yazabilmesi

bir nesnenin hayata gelmiş kabul edilebilmesi için programın akışının contructor'ın içine girmesi ve constructor kodunun sonuna kadar gelmesi gerekiyor.

global sınıf nesneleri hayata main fonksiyonu çağırılmadan başlarlar.
main'in çalışması sonlandıktan sonra hayatları biter.

---

global değişkenler hiç kullanılmayacak diye birşey yok.
tamam dezavantajları var.
inline global değişkenler de aynı dezavantajlara sahip

std::cout => global değişken
std::cin => global değişken

global değişkenleri daha efektif kullanma şekilleri var. İleride anlatılacak.

Murat.h'yı bir sınıfın tanımı aynı projede birden fazla cpp'de include edilse, ODR ihlal edilmiş olmaz
class Murat {

};

Dikkat !
global fonksiyonlarda olduğu gibi sınıfın üye fonksiyonları da inline olarak başlık dosyasında tanımlanabilir ve yine bu durum ODR'ı ihlal etmez.
inline ifadesi en az bir yerde kullanılmalı !! Ya fonksiyon tanımında
veya
fonksiyon tanımında

```
// murat.h
class Murat {  // bildirim,declaration
    public:
        void set(int x);
    private:
        int mx;
};

inline void Murat::set(int x)   // tanım, definition
{
    mx = x;
}
```

Daha sık kullanılan yöntem aşağıda:

```
// murat.h
class Murat {  // bildirim
    public:
        void set(int x)   // inline sözcüğü olmasa da üye fonksiyon implicitly
inline'dır.
        {
            mx = x;
        }
    private:
        int mx;
};
```

inline fonksiyona aday olanlar : özellikle sık çağırılan fakat maaliyeti düşük, küçük kodlar

global fonksiyonlarda olduğu gibi sınıfın üye fonksiyonları da inline olarak başlık dosyasında tanımlanabilir ve yine bu durum ODR'ı ihlal etmez. inline ifadesi en az bir yerde kullanılmalı !! Ya fonksiyon tanımında veya fonksiyon tanımında

```
// murat.h
class Murat {  // bildirim,declaration
    public:
        void set(int x);
    private:
        int mx;
};

inline void Murat::set(int x)   // tanım, definition
{
    mx = x;
}
```

Daha sık kullanılan yöntem aşağıda:

```
// murat.h
class Murat {  // bildirim
    public:
        void set(int x)  // inline sözcüğü olmasa da üye fonksiyon implicitly inline'dır.
        {
            mx = x;
        }
    private:
        int mx;
};
```

üretimde böyle kullanılmayabilir.

```
class Myclass{
public:
    void func(int = 0, int = 0);
    void func(double);
};

int main()
{
    Myclass mx;

    mx.func();  // mx.func(0,0);
    mx.func(10);  // mx.func(10, 0);
    mx.func(10, 20);  // mx.func(10,20);
    mx.func(3.4);  // overloading
}
```

C'de derlenmiş programları C++'da çağıracaksanız :

```
// ali.h

extern "C" void f1();
extern "C" void f2();
extern "C" void f3();
extern "C" void f4();

veya

extern "C" {
    void f1();
    void f2();
    void f3();
    void f4();

}
```

bir nesnenin hayata gelmiş kabul edilebilmesi için programın akışının contructor'ın içine girmesi ve constructor kodunun sonuna kadar gelmesi gerekiyor.

1. constructor ismi sınıfın ismi ile aynı olmalı
2. constructor'lar ve destructor'lar geri dönüş değeri kavramı yoktur.
3. constructor overload edilebilir.
4. default constructor : bir sınıfın parametresi olmayan, ya da tüm parametreleri default argüman alacak, yani argüman gönderilmeden çağıralabilecek constructor'ıdır.
5. constructor const anahtar sözcüğü ile tanımlanamaz.
5. constructor private olabilir ama private ile nesne oluşması engellenir.
6. constructor inline olarak tanımlanabilir.

```
class Myclass{
public:
   Myclass();  // default constructor
   Myclass(int );  // constructor
};
```

destructor :

1. ismi sınıfla aynı , fakat ~Myclass
2. sınıfın non-static üye fonksiyonu olmak zorunda
   global olamaz static üye fonksiyon olamaz
3. const üye fonksiyon olamaz
4. private- protected olabilir
   overload edilemez (parametresi olmamalı)

geri dönüş değeri türü kavramı yok !!!

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

global sınıf nesneleri hayata main fonksiyonu çağırılmadan başlarlar. main'in çalışması sonlandıktan sonra hayatları biter.

```
class Nec{
public:
    Nec()
    {
        std::cout << "Nec default constructor  this = " << this << "\n";
    }
    ~Nec()
    {
        std::cout << "Nec default destructor  this = " << this << "\n";
    }
};

Nec g;

int main()
{
        std::cout << "main başladı \n";
        std::cout << "&g = "  <<  ;

        std::cout << "main sona eriyor \n";
}
```

```
class Myclass{
public:
    Myclass();  // default constructor  // special member function
    Myclass(int );  // special member function değil
};
```

Sınıfın veri elemanlarının initialize edilmesi (en zor bölümlerden biri):
Bir sınıfın contructor ve destructor fonksiyonları global fonksiyon olamaz !!!, statik olamaz. non-static olmak zorundalar.

special member functions (özel üye fonksiyonlar) :

default constructor
destructor
copy constructor
move constructor
copy assignment
move assignment

bunları özel yapan bu fonksiyonları biz tanımlamasak da derleyici belirli koşullar oluşunca derleyici bizim yerimize bu fonksiyonların kodunu yazabilmesi

derleyicinin fonksiyonu default etmesi demek : fonksiyonun kodunu yazabilmesi demek

bir nesnenin hayata gelmiş kabul edilebilmesi için programın akışının contructor'ın içine girmesi ve constructor kodunun sonuna kadar gelmesi gerekiyor.

constructor için de this pointer'ı kullanılabilir.
constructor non-static üye fonksiyon olduğundan, tüm non-static üye fonksiyonlarda olduğu gibi this pointer'ı kullanılabilir.

aynı kaynak dosyada birden fazla global nesne olsaydı, isim sırasına göre constructor'ları çağırılır.

```
Nec nx;  // önce
Nec ex;

int main()
{
        std::cout << "main başladı \n";

        std::cout << "main sona eriyor \n";
}
```

destructorlarda hayata daha önce gelen sonra öldürülür. (constructor'un tam tersi)

static initialization fiasco (ilerde anlatılacak)

farklı kaynak dosyalardaki global nesnelerinin (değişken) hangisinin constructor'ının önce çağırılacağı belirlenmiş değildir.

bir global değişken diğerini kullanıyor, fakat hayata gelmediğinden problem olur.

static yerel değişkenler (static local values)

programın akışı o fonksiyona ilk kez girdiğinden hayata geliyor, (constructor çağrılıyor)
destructor main fonksiyonunun çalışması bittikten sonra çağırılır.

```
        void foo()    // static'lerin hayata gelmesi için ilk çağrı yeterli,
        constructor diğer çağrılarda çağrılmaz, bir kaç kez çağrılarda
        static'lerin hayatı devam eder
        {
            static int cnt{};  // static yerel değişken
            static Nec nx;  // nesne ilk foo çağrısında hayata geldi,
            std::cout << "foo fonksiyonuna yapılan " << ++cnt << " .
        cagri\n";
        }
```

otomatik ömürlü nesneler

static anahtar sözcüğü olmadan kullanılan nesneler
otomatik ömürlü

hayata getiren constructor, hayatını sonlandıran
destructor // scope içinde çağırılır ve scope içinde
sonlanır

```
int main()
{
    Nec nx;  // contructor çağırılır

    Nec& r = nx;  // contructor çağırılmaz, referans nesne değil
}
```

```
int main()
{
    Nec ar[5];   // her  nesne elemanı için constructor çağırılır
    // destructor'ı tam tersi şekilde çağırılır.

}
```

default initialize

demek ki bir sınıfın default constructor'ı olmayabilir

```
class Nec {
public :
    Nec(int x)
    {

    }
};
```

```
int main()
{
    Nec nx; // uygun bir default constructor yok !!
}
```

```
int main()
{
    for (int i = 0; i < 10; ++i){  // 10 kez constructor 10 kez destructor çağırılacak
        Nec x;
        (void) getchar();
    } // şurada destructor çağırılır.  // scope'u sonlandıran closing brace'de çağrılır

    std::cout << "main sonra erdi \n";
}
```

```
class Nec {
public :
    Nec()
    {
        std::cout << "Nec::Nec()\n";
    }
};

int main()
{
    Nec nx{} ; //  default constructor çağırılması için value initialization
    Nec nx();  // geri dönüş değeri Nec türü olan bir fonksiyon bildirimi , function decleration - not instantiation  // hata değil !!!  ?????
}
```

copy initialization

```
class Nec {
public :
    Nec(int x)
    {
        std::cout << "Nec::Nec(int x)  x =" << x << '\n';
    }
};

int main()
{
    Nec n1 = 10;  // copy initialization, 10 değeri constructor'ın x'ine gönderilir.
    Nec n2 (20);  // direct initialization
    Nec n3 {30};  // brace- uniform initialization    // daraltıcı, ifade double olsa diğerleri geçerli bu hata verir
}
```