2023.09.11

extern "C" bildirimi

```
// extern "C"  bildirimi

/*
    C'de derlenmiş fonksiyonların bildiriminde kullanılır

    extern "C" int foo(int);

    extern "C"
    {
        int foo(int;
        int bar(int,int);
        int baz(double);
    }
*/

/* predefined symbolic constants ( C )

    __LINE__
    __FILE__
    __DATE__
    __TIME__

    __STDC__
    __cplusplus__
*/
```

```
#ifdef _cplusplus
extern "C"
{
#endif
    int foo(int;
    int bar(int,int);
    int baz(double);

#ifdef _cplusplus
}
#endif
```

Composition (has a relation-ship)

```
class Engine
{
    public:
        void start();
        void stop();
    private:
        int cyl_volume;
}

class Car
{
    public:
          void start()
          {
              /*
              eng.cyl_volume private elemanlara erişemeyiz ancak
              friend bildirimyle erişelebilriz
              */
          }
    private:
        Engine eng;
}

int main()
{
    Car mycar;

    // mycar.start() böyle bir şey yok
    /*
        Engine interfacesini default olarak Car'ın
        interfacesine katamayız
    */
}
```

```
class Myclass
{
    public:
    /*
        1) Tanımlama sırasına göre A, B, C hayata gelecek bu yüzden
        ctor'da bizde aynı şekilde atama yapmalıyız

        2) ax, bx ve cx nesnelerin ctor'ları delete edilmiş veya private
        bölümündeyse  Myclass classın default ctor'u derleyici
        tarafından delete edilir
    */
    private:
        A ax;
        B bx;
        C cx;

}
```

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

class B
{
    public:
        B (int x) : ax {x}
        {

        } // Uygun kullanım yöntemi
        A ax { 12 }; // Hata olmaz derleyici A(int) kullanır
        A ax; // Hata olur derleyici B'nın ctor'u delete eder
};

int main ()
{
    B bx;
}
```

Example:

```
class Person
{
    public:
        Person(const std::string & name, const std::string sname, int age) :
        m_name{ name }, m_surname { surname }, m_age {age}
        {
            m_name = name;
            /*
                yukarıdaki gibi tanımlarsak eğer m_name önce default ctor ile
                hayata gelir daha sonra copy assignment çağrılır
                bu da maliyetli olur
            */
        }
    private:
        /*
            string sınıfın default ctor çağrılır eğer Person
            classında default ctor kullanılırsa
        */
        std::string m_name;
        std::string m_surname;
        int age; // indetermined value ile başlarlar
};

int main()
{
    Person per {"Emre", "Bahtiyar", 26};
}
```

```
class Myclass
{
    public:
        Myclass(int x, int y, int z) : x( x ), y( y ), z( z )
        int x, y, z;
        //Doğru bir kod değil ama isim aramadan dolayı hata olmaz
        // Yani  ( x ) zaten publicteki x olamaz
}

int main()
{
    Myclass myclass{1, 2, 3};
}
```

Copy Ctor

```
class Myclass
{
    public:
        Myclass(const Myclass &other) : ax(other.ax), bx(other.bx)
        {
            /*
                Eğer cx init etmezsem cx'ın default ctor'u çağrılır
            */
        }
    private:
        A ax;
        B bx;
        C cx;
}
```

Example

```
class Member
{
    public:
        Member()
        {
            std::cout << "Member default ctor\n";
        }
        Member(const Member&)
        {
            std::cout << "Member copy ctor\n";
        }
};

class Owner
{
    public:
        Owner();
        Owner(const Owner &other) : x(other.x) {}
    private:
        Member mx;
        int x;
}

int main()
{
    Owner x; // Member default ctor çağrılır
    /*
        Member default ctor çağrılır çünkü Owner class'nın copy
        ctor'u da mx'i init etmedik
    */
    Owner y(x);
}
```

Move ctor && Copy Assignment && Move Assignment

```
class Myclass
{
    public:
        Myclass(const Myclass &other) : ax(other.ax), bx(other.bx) {}
        Myclass(Myclass &&other) : ax(std::move(other.ax)), bx(std::move(bx)),
cx(std::move(cx))
        {
            /*
                1) Elemanlardan birini unutsak unutulan eleman default init edilir
                2) ax(std::move(other.ax)) yerine ax(other.ax) yazarsak
                ax copy ctor ile init edilir
            */
        }
        /*
            assign etmeyi unuttursak unuttuğumuz elemanlar
            eski değerleriyle kalacak
        */
        Myclass operator=(const Myclass &other)
        {
            ax = other.ax;
            bx = other.bx;
            cx = other.cx;

            return *this;
        }

        Myclass operator=(Myclass &&other)
        {
            ax = std::move(other.ax);
            bx = std::moev(other.bx);
            cx = std::move(other.cx);

            return *this;
        }

    private:
        A ax;
        B bx;
        C cx;
}
```

Example:

```
class Member
{
    public:
        Member& operator=(const Member&)
        {
            std::cout << "Member::operator=(const Member&)\n";
        }
};

class Owner
{
    public:
        Owner& operator=(const Owner& other)
        {
            //..
            return *this;
        }
    private:
        Member mx;
}

int main()
{
    Owner x;

    x = y;
}
```

Reference Qualifiers

Referans Niteleyecisi

Modern Cpp ile dile eklendi ancak C++17 ve 20 std ile

ilave özellikler kazandı.

Bir sınıf nesnesi için:

sınıf nesnesinin value category'sinin ne olduğuna ve / veya

sınıf nesnesinin const olup olmadığına bağlı olarak:

a) bir üye fonksiyon çağrısının legalitesini belirliyor

b) function overloading olması durumunda hangi fonksiyon seçilmesi

gerektiğini belirliyor

```
class Myclass
{
    public:
        void set(int);
};

int main()
{
    Myclass{}.set(12); // legal ama mantıklı değil

    Myclass m;
    // legal
    m = Myclass{};
    Myclass{} = m;
    Myclass{} = Myclass{};
}
```

Example

```
class Myclass
{
    public:
        // foo sadece L value kategorisindeki sınıf nesneleri için çağrılabilir
        void  foo(int)&; // L value ref qualifiers
        // bar sadece R value kategorisindeki sınıf nesneleri için çağrılabilir
        void  bar(int)&&; // R value ref qualifiers
        void  baz(int) cont&; // L value ref qualifiers
        void  func(int) cont&&; // L value ref qualifiers

        // function overloading var
        void foo()&; // L value ise bu
        void foo()&&; // R value ise bu
        void foo()const&;
        void foo()const&&;

        /*
            void foo()&,
            void foo();
            böyle bir overloading yapamayız hatalıdır
        */
};

int main()
{
    Myclass m;
    m.foo(12); // foo() fonksiyonu L value sınıf nesneleri için çağrılabilir

    Myclass{}.foo(12); // L value olmadığı için hatalı

    m.bar(12) // hatalı
    Myclass{}.bar(12) // hatalı değil

    Myclass{}.baz(12); // hatasız
    m.baz(12) // hatasız

    Myclass{}.func(12) // hatasız
    m.func(12) // hatalı

}
```

```
/*

Myclass operator=(const Myclass &other)& = default;
L valeu reference qualifiers böyle default edebiliriz böylece

Myclass m;
Myclass{} = m // sytnax hatası olur

*/
```