2023 07 28

Referans Semantiği

- L value reference
- R value reference
- Universal reference (forwarding reference)

L value expression

-Eğer bir expression bir nesneye(storoge var, yeri ve kullanılıabilir) karşılık geliyorsa ve bir nesne gösteriyorsa bu ifade L value expressiondır.

R value expression

-Bir nesneye karşılık gelmez ama bir değer ifade ediyorsa yani bir nesneye değer olarak verebiliyorsak R value expression denir.

Bir ifadenin başına adress operator (&) verirsek ve hata vermiyorsa L value expressiondır veriyorsa R value expressiondır diyebiliriz.

```
/*
    ++x, --x ifadesi C'de R value expression C++'da L value expression
    x, y ifadesi C'de R value expression C++'da L value expression
    x > 10  ? x : y ifadesi C'de R value expression C++'da L value expression
    x = 5 ifadesi C'de R value expression C++'da L value expression
*/
```

Value Category

Primary value categories

- **L value**
- **PR Value (pure R value)**
- **X Value (eXpiring value)**

combined / unified value category

PR value || X value --> R value

L value || X value --> GL value

```
int main()
{
    // x'in value category'si olmaz
    // x'in  türü int
    int x = 0;
    // value category'si var -- L value category
    x
    // NOT: isimlerin oluşturduğu ifadeler her zaman L value categorydedir.
}
```

L value expression yapan operatorlar:

- ++x
- --x
- *ptr
- ptr[35]
- x, y
- a ? b : c

yukarıdaki dışındaki operatorlar PR value expressiondır

PR value expression yapan operatorlar:

- x++
- a + b
- +x
- -c
- !a
- a > b
- a == b
- &a

```cpp
// Bir ifadenin value category bulan kod
template <typename T>
struct ValCat {
    static constexpr const char* p = "PR Value";
};

template <typename T>
struct ValCat <T&> {
    static constexpr const char* p = "L value";
};

template <typename T>
struct ValCat <T&&> {
    static constexpr const char* p = "X value";
};

#include <iostream>

#define pvcat(expr) std::cout << "value category of expr '" #expr "' is <<
ValCat<decltype((expr))>::p << "\n"


int foo();
int main()
{
    pvcat(foo); // L value
    pvcat(foo()); // PR value

    int x {0};
    int *p  = &x;
    int **ptr = &p;

    pvcat(&p) // PR value

    // Hepsi L value
    pvcat(x);
    pvcat(p);
    pvcat(*p);
    pvcat(ptr);
    pvcat(*ptr);
    pvcat(**ptr);

    int a[2]{};
    pvcat(a); // L value
    pvcata(a[0]); // L value
    pvcat(&a[0]); // PR value
    pvcat(a + 1); // PR value
}
```

Reference Semantiği

```
int main()
{
    int x = 10;
    int& r = x;

    r = 45; // x = 45
    ++r; // ++x

    r = y  // x = y
}
```

```
int main()
{
    int x = 10;
    int *ptr = &x; // bildirimdeki *ptr'deki * decralator'tur
    *ptr = 10; // burdaki (expression) * operator'dur
}
```

```
int main()
{
    int x; // default init
    const int y; // syntax hatası

    // referans değişkenler defaılt init edilemez
    int &r; // syntax hatası
}
```

NOT: R value expression'na R value ref, L value expression'a L value ref verebiliriz

```
int main()
{
    // L value ref'a R value expressionla ilk değer vermeyiz.
    int &r = 10; // syntax hatası

    int x = 10;
    int &r = x; // r referansı  x'e bind edilmiş
}
```

```
int foo();
int main{}
{
    int x = 345;

    int &r1 = x; // geçerli
    int &r2 = +x; // geçersiz
    int &r3 = x++; // geçersiz
    int &r4 = ++x; // geçerli
    int &r5 = &x; // geçersiz

    int &r6 = foo(); // geçersiz
}
```

```cpp
// references are not rebindable
int main()
{
    int x{};
    int &r = x;

    int y{};
    &r = y; // syntax hatası
}
```

```cpp
// const L value reference
int main()
{
    const int x{};

    // ikiside aynı anlamda
    const int& r1 = x;
    int const& r2= x;


    int y{};
    const int& r3 = y;

    y = 32;
    r3 = 32; // geçersiz

    const int z{};
    int &r4 = z; // geçersiz
}
```

```cpp
int g = 24;
void foo(int *& r)
{
    r = &g;
}
int main()
{
    int * p = nullptr;
    foo(p); // p'ye adres verdik
}
```

```cpp
int g = 35;
int& foo(void)
{
    return g;
}
int main()
{
    foo(); // L value expression
    // geçerli
    foo() == 99;
    ++foo();

    int *ptr = &foo();
    ++* ptr;
}
```

```
int &foo()
{
    // tanımsız davranış otomatik ömürlü nesnesin adresi döndürüyor
    int x = 45;
    return x;
}
```