* Maximal Munch Kuralı: En uzun atomik birimi oluşturma.
(tokeni)

```
int main() {
    int x = 10
    int y = 20
    int z = x+++y
}
```

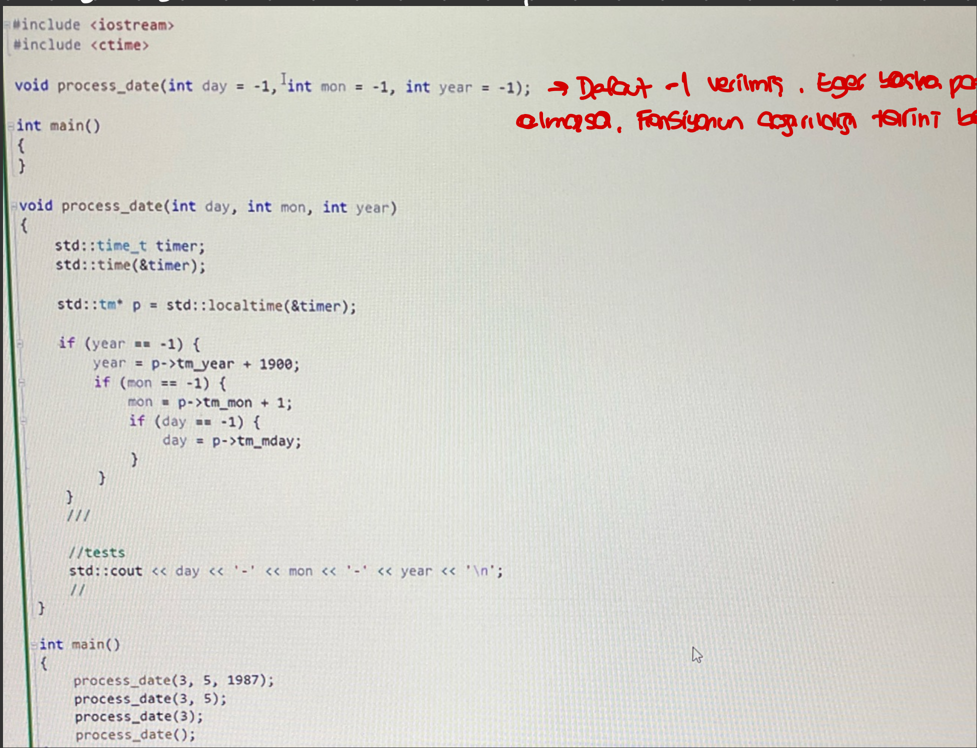
↳ en uzun token ++ ve toplama işlemi

→ Benzeri bir olayı "default value" atamasında yaşayabiliriz.

→ func (const char * = "mert")

↳ ayrı yazmazsak, çarpma işlemi olur.

→ Varsayılan argüman kullanım alanlarından biri de şu örnekte incelenebilir.



```
#include <iostream>
#include <ctime>

void process_date(int day = -1, int mon = -1, int year = -1);

int main()
{
}

void process_date(int day, int mon, int year)
{
    std::time_t timer;
    std::time(&timer);

    std::tm* p = std::localtime(&timer);

    if (year == -1) {
        year = p->tm_year + 1900;
        if (mon == -1) {
            mon = p->tm_mon + 1;
            if (day == -1) {
                day = p->tm_mday;
            }
        }
    }
    ///

    //tests
    std::cout << day << '-' << mon << '-' << year << '\n';
    //
}

int main()
{
    process_date(3, 5, 1987);
    process_date(3, 5);
    process_date(3);
    process_date();
```

→ Default -1 verilmiş. Eğer başka parametre olmazsa, fonksiyonun çağrıldığı tarihi basıyor.

* Referanslar ve Referans Semantiği:

→ C dilinde genelde "pointer semantiği" kullanılmaktadır. C++'da ise pointer semantiğine ek "reference semantiği" eklenmiştir.
→ Fakat Assembly düzeyde pointer ve referans arasında fark oluşmuyor.
→ C++'da en başta "Operator overloading" yapmak için referans semantiği kullanıldı.

→ C dilindeki pointerlara C++ alternatifi olarak (C dilindeki pointerlara artık naked / raw pointer diyoruz)
↳ 1. referans semantiği
↳ 2. smart pointer

→ Modern C++ (C++11)'de 3 adet referans semantiği vardır.
↳ L value reference ⟶ modern C++ öncesinde referans buydu.
↳ R value reference
↳ Forwarding (Universal) value reference

Örnek:

```
int main() {
    int x {10};
    int * ptr = &x;
}
```

→ ptr'de x'in adresi var.

Bu durumda `(*ptr) = x`

↓ de-referencing

→ Bunu referans semantiği ile yazsak;

→ Yani `r = x` yapmak için

```
int x {10};
int &r = x
```

→ r, x'e referans

(int) referans türü

(&r) ampersan kullanarak referans oluşturur

/ referans değer aramadan init edilmez.

*** Önemli:** Bir referans hangi nesneye bağlanırsa, scope'u sonuna kadar yalnızca ona referans eder

```
int main() {
    int x {10};
    int y = 56;
    int &r = x ]→ r, x'e bir referans

    r = y; ]→ x=y olur. r, y'nin referansı olmaz. X'in değeri 56 olur.
}
```

→ Referansları initialize etmek zorundayız! Bu tip Referanslar L value referanstır. Yalnızca L value expression ile initialize edilir.

* C Tekrar:

1. Expression (İfade): Sabitlerin isimlerle ve operatörlerle yaptığı birleşim.

   ⇒ `x+10` bir expression
   `x+10;` bir statement

2. Data Type → int, double, float, int*... / C++'da aynı

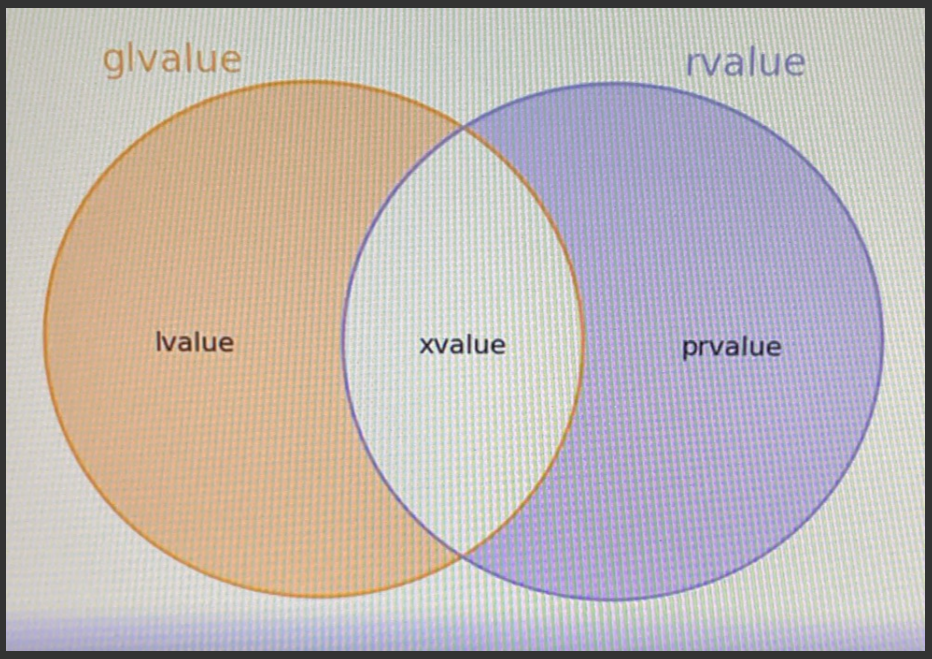
   Value Category → L value expression
   R value expression

   → C dili için: Bir nesnenin adres operatörünün operandı yapsak ve syntax hatası olursa R
   olmazsa L value
   ↙ adresi alınabiliyorsa

Modern C++'da 3 adet Primary Value Category vardır.

- PR value (expression) → pure R value
- L value (expression)
- X value → eXpiring value

• Bu Primaryler kullanarak Combined Value oluşturulur.



→ L value ∪ X value = gl value

→ PR value ∪ X value = r value

→ C. Dizi pointer örneği (Pointer to Array):

```
int a[5] = {1,2,3,4,5}
```

⇒ C++ referans mantığıyla ⇒

```
int a[5] = {1,2,3,4,5}
int (&r)[] = a   → Diziye referans
auto &r = a;     → Auto ile type deduction yaparak da diziye referans yapabilirdik.
```

```
int (*p) = &a  → *p, a dizisinin kendisi
int *ptr = a   → ptr, a dizisinin ilk elemanı
```

* Call By Value / Call By Reference:

→ Referans semantiğinin en sık kullandığı yerlerden biri, fonksiyonlarda call by reference

→ C dilinde default olarak → call by value

```
→ void func(int)  → call by value'dur
                    x'in değerini DEĞİŞTİREMEZ!
int main(){
    int x=10;
    func(x);
}
```

```
→ void func(int *a)
{
    *a = 999;  /→ C++'da referans ile → r = 999;
}

int main(){
    int x = 10;
    func(&x);   → Call by reference olduğu için, x'in değeri 999 olur.
}
```

* C dilinde fonksiyonun içeriğini görmeden, call by value diyebiliriz. Fakat C++'da bu cümlemizde bulunmamaz.

↳ C++'da: func(int) → CBV
func(int&) → CBR

→ void func(int *p) : Adresini verdiğimiz nesneyi değiştirebilir. ⇒ Bunlara Mutator Function Denir!
void foo(const int *p) : Bu fonksiyona, nesnenin adresini gönderirsek, o nesne Değişmez!!
↳ salt okunur hale getirdi.

Not/: Diyelim ki yalnızca okuma yapacak bir fonksiyon yazıcaz. CONST CORRECT olmalı, void print_arr(const int *p, size_t size) gibi

Not/:
```
int main() {
    int x = 10;
    int * const p = &x;
```
Buna üstteki const denir. ⇒ p'nin değişmeyeceğini söylüyor.

```
    int const * p = &x;
```
Buna pointer to const denir. / const int * yazmakla AYNI!

→ Benzeri durum referanslar için de geçerli
→ int const &r = x;

* C dilinde ADRES DÖNDÜREN Fonksiyonlar:

```
adres döndüren bir fonksiyon C'de

a) statik ömürlü bir nesne adresi döndürebilri
    a) global değişken adresi
    b) static yerel değişken adresi
    c) strinng literal

b) dinamik ömürlü nesne adresi
```

c) Dışarıdan Alınan Adres

⇒ Referansta da bu durum geçerli.

```
int g = 10;

int& func()
{
    //

    return g;
}

int main()
{
    func() = 999;
}
```
→ g = 999 olur.

• Bir fonksiyon L value TF dönerse, diğer kategorisi L value expr olur.

* Pointer ile Referans Arasındaki Farklar:

⇒ Assembly Düzeyinde fark yok, semantik olarak var. Bunlar;

1. Pointerlara ilk değer vermek zorunda değiliz. Verilmemesi legal. Fakat referanslarda bu mümkün değil.

```
int* ptr ✓        int &r; ✗
                    ↳ syntax hatası
```

2. Pointer değişkeni, kendisi const olmadığı sürece, o değişken, farklı değişkenleri point edebilir. Fakat referans sadece tek nesneyi refer eder. Re-bindable değil. Pointerlar re-bindable yani.

↳ Aynı etkiyi top level const ile yaratabiliriz (`int* const`)

3. Pointerların dizisi olabilir. Fakat referans dizisi yok. Referanslar dizide tutulamaz.

Örn:
```
int g1 = 10
int g2 = 20
int g3 = 30

int main(){
  int* p0[3] = { &g1, &g2, &g3 }
}
```
↳ Elemanları pointer olan pointer array.

4. Pointer to pointer var ama reference to reference yok.

5. Null pointer kavramı. Null reference yok.

⇒ Null pointer = hiç bir adresi göstermez ama geçerli bir pointerdır.
- → Adres döndüren fonksiyonlarda, TSF başarısızda nullptr döner
- → Arama fonksiyonları tipik olarak adres döndürür. Aranan bulunamazsa null ptr.