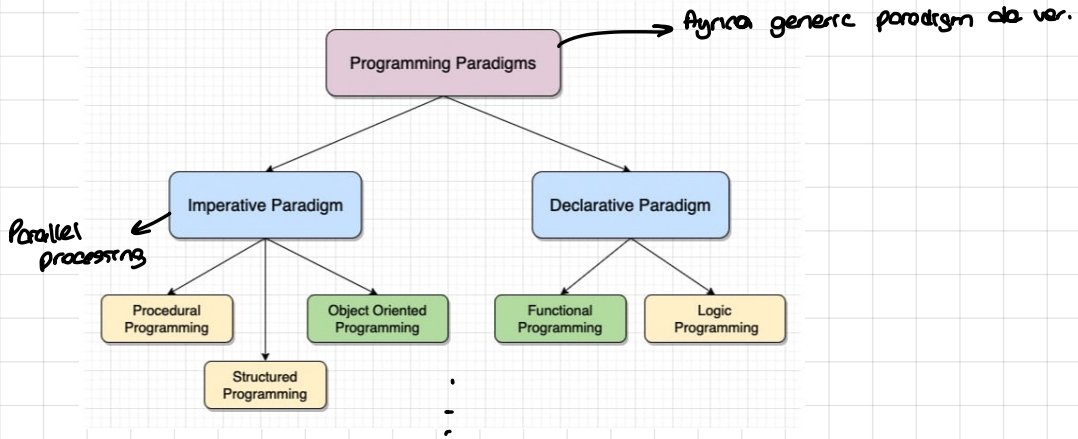


• C dir → prosedural programlama dır.

→ Bu Programming Paradigm olarak geçer



→ Ancak C++ multi-paradigm bir dır / OOP, genelle, Procedural

→ C++ verimlilik (efficiency) gereken yerlerde kullanılır.
 → Android/video
 → Fin tech
 → Sistem programlama
 vs.... / Java C# kullanılır çünkü
 C++ daha güvenli genelle

→ C'de Olup C++'da Olmayan Özellikler:

- Compound Literal:
 - Variable Length Array
 - Designated Initializers
- kullanılamaz baki.

* Konu 1: C++ ve C Arasındaki İmpatirite Durumlar:

• Implicit Int: Güçlü int → C++'da geçerli değır. C'de geçerli
 Kapsalı int
 Örtülü int

• Normalde type ki belirlenir uygulanı gereken yerlerde, C'dir default olarak signed int varsayar.

func(void) C'de uygulanır, C++ uygulamaz
 → return değeri signed int kılal etti

• Fonksiyon Parametre tanımları → C'dirinde:
 - int foo()
 - int func(void)
 → farklıdır.
 foo(1,2,3) .. gibi
 çağırılır. YANMA!!

→ C++'da:
 - int foo()
 - int func(void)
 → Aynıdır.

• Block Scopes: → C'dirinde:
 • Scope tanımlı statement
 sonra declaration uygulanır. (C 99'da gitti)

→ C++'da:
 • Bu uygulanır.

• Old-style Function Definition: → C'dirinde:

```

double sum(d1, d2, d3)
double d1, d2, d3
{
    return d1+d2+d3;
}
    
```

 → Fonksiyonun
 Ana başı olmadan
 önce deklarasyon tanımlı
 sonra fonksiyon (C 99'a kadar)

→ C++'da:
 • Bu yapı.

→ Loose Type Control: C dilinde derleme zamanında type control çok zayıfken
C++ dilinde oldukça katı.

→ Aritmetik Terlerden Pointer'a Implicit Dönüşüm: C dilinde uygulanır / $\text{int } x = 10$
C++ dilinde syntax error / $\text{int } * \text{ptr} = x$

→ Typecast Geçitleri: C++'ın kendine özgü castingleri var

- Static cast
- Const cast
- reinterpret cast
- dynamic cast

C'de terler arasında typecast var.

→ Karakter Sabitleri: C'de 'A', '\n', '\101' gibi değerlerin teri **int**
(character literal) C++'da teri **char**.

→ String Sabitleri: "kırım" → C dilinde **char[6]** (char arraydır)
(string literal) kullanırken ancak, array decay olur. **char*** olarak işlem görür. (Array to char*)

→ C++'da **const char[6]**
Array **const char***'a decay olur.

- Array Decay / Array to Pointer Conversion:

$\text{int } a[10] = \{0\};$ / $a[0]$ → Dizi'nin ismi
dizi'nin ilk elemanına atıf

→ Const Keyword: C dilinde const değrken int edilmey zorunda değil. Değer atanmay zorunda değil.

C++ dilinde zorunlu !!!

→ Linkage: $\text{int } x = 10;$ → external linkage
aynı tem modelde
link time ve run time da
aynı değeri ifade eder

→ eğer global olarak, static diye
tanıtılıyorsa internal linkage

• C++'da global const nesneler $\left\{ \begin{array}{l} \text{C dilinde external} \\ \text{C++ dilinde internal} \end{array} \right.$

→ Initializing Different Const Qualifiers:
 * const int x=10 } C'de legal
 int * p = &x } C'de illegal

→ Boolean: C 99'da _Bool eklenmiştir. Aynı bir tür. (stdbool kütüphane).

bool flag = true
_Bool flag = true 'dir olmuyor, C++ dilinde bool ayrı bir tür (distinct type)

→ true ve false bir macro'dur
enum olarak da tanımlanabilir.

* Karşılaştırma Operatörleri 10 > 5 } C'de signed int olarak işlem görür. → C'de 0
C++'da bool → C++ bool

* C++'da bool'dan aritmetik işlemlere dönüşüm var → int x=0 bool b=x
int x=1 bool b=x
↓ false ↓ true

* C++ 17'ye kadar bool flag = false diye yazılabiliyordu. C++ 17'de kaldırıldı. Modern C++'da syntax error.
++flag

* C++'da pointer to bool da var. Null pointer false.
Diğer tüm pointerlar true.
(tam tersi her zaman)
syntax error

→ NULL: C dilinde NULL bir macro'dur. Güçlü compiler bunu null pointer olarak kullanır. C++'da ve nullptr conversion NULL kullanılmıyordu.

C++ en başında int * p = 0 gibi kullanımı vardı → Null pointer yerine 0

Modern C++'da nullptr geldi. (nullptr-t type'inde). Artık NULL'da yok. 0'a eşittir de yok.
↓
true / false gibi bir sabit.

• int * p = nullptr; ✓
int x = nullptr; ✗
• bool x = nullptr; ✗

• C dilinde int * p = 0 'da null pointer'dır.
int * p; diye yazmak C dilinde de global olarak tanımlandıysa null pointer yoksa 0'ın değeri

