```cpp
if constexpr (exp) {
    //code_a
}
else if constexpr() {
    //code_b
}
else
```

else if için de constexpr kullanılır

```cpp
template <typename T>
std::string as_string(T x)
{
    if constexpr (std::is_same_v<T, std::string>) {
        return x;
    }
    else if constexpr (std::is_arithmetic_v<T>) {
        return std::to_string(x);
    }
    else {
        return std::string(x);
    }
}

#include <iostream>

int main()
{
    std::cout << as_string(x:42) << '\n';
    std::cout << as_string(x:4.2) << '\n';
    std::cout << as_string(x:std::string("hello")) << '\n';
    std::cout << as_string(x:"hello") << '\n';
}
```

```cpp
template <typename T>
void func(T tx)
{
    if constexpr (std::is_integral_v<T>) {
        if (tx != 0) {
            func(tx--);
            //...
        }
    }
    else {
        undeclared_f(); //syntax error (name non dependant on template parameter)
        undeclared(tx);   //error if else part instantiated
        //static_assert(false, "not integral type"); //syntax error
    }
}
```

eğer undeclared_f(tx) olsaydı, syntax hatası kalkardı → compile hatası olurdu

```cpp
//function return type may be int or double

constexpr auto func()
{
    if constexpr (sizeof(int) > 4u) {
        return 1;
    }
    else {
        return 0.5;
    }
}


//function return type may be int or void

auto foo()
{
    if constexpr (sizeof(int) > 4) {
        return 1;
    }
}

int main()
{
    constexpr auto val = func();
}
```

→ Auto return type

*Tag Dispatch: → Compile-time da kod / ter seçimi,

```cpp
struct construct_this_way {};
struct construct_that_way {};


class Myclass {
public:
    Myclass(construct_this_way);
    Myclass(construct_that_way);
};


int main()
{
    Myclass m{ construct_this_way{} };
    Myclass m{ construct_that_way{} };
}
```

Bu şekilde empty classlar kullanılarak farklı türden constructorlar oluşturulabilir.

Daha iyi kullanımı şu şekilde

```cpp
class Myclass {
public:
    struct construct_this_way_t {}construct_this_way;
    struct construct_that_way_t {}construct_that_way;

    Myclass(construct_this_way_t);
    Myclass(construct_that_way_t);
};


int main()
{
    Myclass m{ Myclass::construct_that_way_t };
}
```

```cpp
#include <iostream>


template <typename T>
void func(T x, std::true_type)
{
    std::cout << "tam sayi turleri icin imlementasyon\n";
}


template <typename T>
void func(T x, std::false_type)
{
    std::cout << "tam sayi olmayan turler icin imlementasyon\n";
}

template <typename T>
void func(T x)
{
    func(x, std::is_integral<T>{});
}

int main()
{
    func(x: 12);
    func(x: 5.6);
}
```

IF CONSTEXPR. kullanımı!

```cpp
#include <iostream>


template <typename T>
void func(T x)
{
    if constexpr (std::is_integral_v<T>) {
        std::cout << "tam sayi turleri icin kod\n";
    }
    else {
        std::cout << "tam sayi olmayan turler icin kod\n";
    }
}

int main()
{
    func(x: 12);
    func(x: 5.6);
}
```

* Advance Örneği:

```cpp
#include <iterator>

namespace details {
    template <typename Raniter, typename Distance>
    void advance(Raniter& it, Distance n, std::random_access_iterator_tag)
    {
        it += n;
    }

    template <typename Biditer, typename Distance>
    void advance(Biditer& it, Distance n, std::bidirectional_iterator_tag)
    {
        if (n > 0) {
            while (n--) ++it;
        }
        else {
            while (n++) --it;
        }
    }

    template <typename Initer, typename Distance>
    void advance(Initer& it, Distance n, std::input_iterator_tag) {
        while (n--) {
            ++it;
        }
    }
}

template <typename Iter, typename Distance>
void advance(Iter& it, Distance n)
{
    details::advance(it, n,
        typename std::iterator_traits<Iter>::iterator_category{});
}
```

* Variadic Function Template:

```cpp
template <typename T, typename ...Ts>
void print(const T& x, const Ts& ...args)
{
    if constexpr (sizeof...(args) == 0)
        std::cout << x << '\n';
    else
        std::cout << x << ", ";

    if constexpr (sizeof...(args) != 0)
        print(args...);
}

int main()
{
    print(12, 3.4, "ali");
}
```

<T> Provide sample template arguments for...

→ Buradaki sizeof, variadic argüman paketinde kaç eleman kaldığını söyler.

print(args...) → pack expansion yapar. → Compile time recursion
↳ Constexpr olduğu için if

Compile time recursion'a ayrıyetten bir "base case" yazmaya gerek kalmadı.

compile time recursion şu şekilde açılırdı.

```cpp
void print(const int& x, const double& p1, const char*& p2)
{
    std::cout << x << ", ";
    print(p1, p2);
}

void print(const double& x, const char*& p1)
{
    std::cout << x << ", ";
    print(p1);
}
```

→ sürekli yeni print fonksiyonları bu şekilde yazılıyor olacaktı compiler tarafından

*Copy Array örneği:

```cpp
template <typename T, std::size_t N>
void copy_array(T(&dest)[N], const T(&source)[N])
{
    if constexpr (std::is_trivially_copyable_v<T>)
    {
        std::memcpy(dest, source, N * sizeof(T));
    }
    else
    {
        std::copy(source, std::end(source), dest);
    }
}

#include <iostream>

int main()
{
    int a[5] = { 3, 6, 7, 9, 1 };
    int b[5];

    copy_array(b, a);

    for (auto i : b)
        std::cout << i;
}
```

→ Safely copyable with memcpy
↳ safely copyable types:
- arithmetic types
- enum types
- pointer types + nullptr
- pointer to members

→ bunların const/volatile hali

→ Örneğin String bu şekilde kopyalanır.

* Array size Döndüren Template: → C'deki asize.

```cpp
#include <type_traits>
#include <algorithm>


template <typename T, std::size_t n>   <T> Provide sample template arguments f
constexpr std::size_t asize(const T(&r)[n])
{
    return n;
}


int main()
{
    double a[5]{};

    constexpr auto const unsigned int size = asize(r: a);
}
```

compile time sabiti oldu.

* SFINAE: Substitution Failure Is NOT AN ERROR

8.4 SFINAE (Substitution Failure Is Not An Error)
In C++ it is pretty common to overload functions to account for various argument types.
When a compiler sees a call to an overloaded function, it must therefore consider each candidate separately, evaluating the arguments of the call and picking the candidate that matches best (see also Appendix C for some details about this process

In cases where the set of candidates for a call includes function templates, the compiler first has to determine what template arguments should be used for that candidate, then substitute those arguments in the function parameter list and in its return type, and then evaluate how well it matches (just like an
However, the substitution process could run into problems:
It could produce constructs that make no sense. Rather than deciding that such meaningless substitutions lead to errors, the language rules instead say that candidates with such substitution problems are simply ignored.

We call this principle **SFINAE (pronounced like sfee-nay)**, which stands for _"substitution failure is not an error."_
Note that the substitution process described here is distinct from the on-demand instantiation process (see Section 2.2 on page 27):
The substitution may be done even for potential instantiations that are not needed (so the compiler can evaluate whether indeed they are un
It is a substitution of the constructs appearing directly in the declaration of the function (but not its body).

```cpp
template <typename T>
T func(T x);

double func(double);
void func(int);

int main()
{
    func(1.2);
}
```

* T'nin double olduğunun anlaşılması ⇒ substitution!!

```cpp
template <typename T>
typename T::value_type func(T x);

void func(int);

int main()
{
    func(1.23);
}
```

Double'ın value type'ı yok!!

* Burada substitution hatası VAR!!

Fakat bu olmasına rağmen error vermeyecek Candidate overload setten çıkartılır.

int olan çağrılır.

#SFINAE ile kod seçimi / eliminasyonu:

→ 02.03.23'teki örnekleri tekrar et

```cpp
#include <iostream>

1) template<typename T, unsigned N>
std::size_t len(T(&)[N])
{
    return N;
}

2) template<typename T>
typename T::size_type len(T const& t)
{
    return t.size();
}

int main()
{
    int a[10]{};
    std::cout << len(a);

    //std::cout << len("tmp");
    //int* p{};
    ////std::cout << len(p); //gecersiz

    //std::allocator<int> x;
    //std::cout << len(x); //gecersiz
}
```

- 2) → SFINAE'd at
- int a[10]{}; std::cout << len(a); → substitution fail ve 1 seçilir
- //std::cout << len("tmp"); → 1 seçilir.
- //std::allocator<int> x; //std::cout << len(x); → syntax hatası olacak,

*Enable_If: → Type traits kütüphanesindeki bir template

→ maksat bu araçları kullanarak, istemediğimiz fonksiyonları SFINAE'e düşürmek

```cpp
template<bool B, class T = void>
struct enable_if {};

template<class T>
struct enable_if<true, T> {
    typedef T type;
};

template<bool B, class T = void >
using enable_if_t = typename enable_if<B, T>::type;

int main()
{
    enable_if<true>::type
}
```

- enable_if<true, T> → partial specialization
- enable_if_t → Alias template
- true → eğer false olsaydı ::type kullanımı syntax hatası olacaktı

```cpp
#include <iostream>

template <typename T>
void func(T x, std::enable_if_t<std::is_integral_v<T>>* = nullptr)
{
    std::cout << "integer types\n";
}

template <typename T>
void func(T x, std::enable_if_t<!std::is_integral_v<T>>* = nullptr)
{
    std::cout << "non integer types\n";
}

struct A {};

int main()
{
    func(x:1); → 2. fonksiyon SFINAE'd out
    //func(1u);
    //func('A');
    //func(true);
    func(x:1.1); → 1. fonksiyon SFINAE'd out
    //func(A{});
}
```

* SFINAE örneği:

```cpp
#include <iostream>

template<int I>
void func(char(*)[I % 2 == 0] = nullptr)
{
    std::cout << "even numbers\n";
}

template<int I>
void func(char(*)[I % 2 == 1] = nullptr)
{
    std::cout << "odd numbers\n";
}

int main()
{
    func<6>(); → Çift sayılarda 2. template SFINAE'd out
    func<13>(); → Tek " 1. template SFINAE'd out
}
```

NON-TYPE PARAMETRE → 1 bir sabit.

pointer to array. → Çift sayıda eşit elemanlı array, true = 1

```cpp
#include <type_traits>
#include <iostream>

template <class T>
typename std::enable_if_t<std::is_arithmetic_v<T>, T>
func(T t)
{
    std::cout << "func<arithmetic T>\n";
    return t;
}
```

*** Kendime Not:**

- Ders 52 ve 53'ü tekrar izle
- Anlayamama nedenim template eksiği