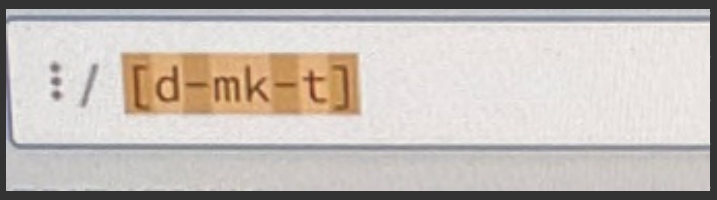
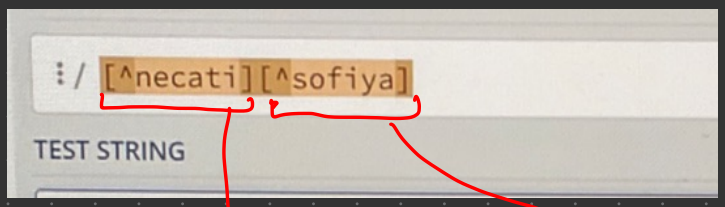
• metacharacter'lar (. , !) gibi search etmek için escape character kullanılmalı (\.) gibi
(\ ) ters bölü



→ d-m aralığı + k-t aralığı



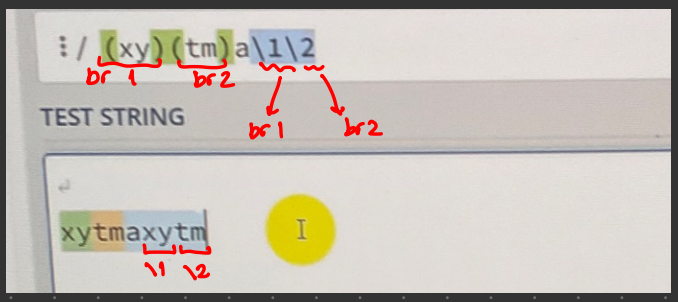
1. karakter necati'nın harfleri olamaz.

2. karakter sofiya'nın karakterleri olamaz.

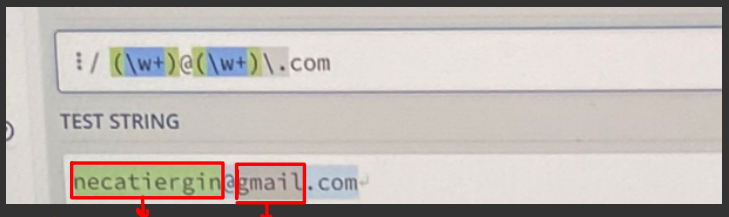
* = zero or more
\+ = one or more
\? = kendinden önce üşe ya olacak ya da olmayacak
\d = digit
\D = non-digit
\w = underscore ve alphanumeric values
\W = non-alphanumeric
\s = whitespace
\S = non-whitespace

→ hepsinin küçük boolean true
hepsinin büyük boolean false

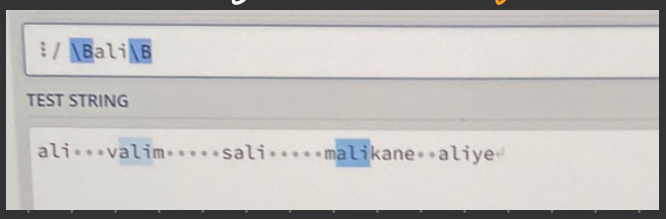
( ) = öncelik belirtiliyor
= back reference oluşturur. (\1 back reference dir) → Yani parenteze içindeki ilk grubun aynısı onlemine gelir.



=capture group oluşturur.



\b = word boundry → kelime en başında ya da sonunda



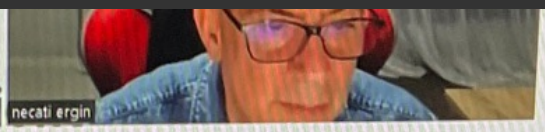
→ Başında ya da sonunda olmayanlar alır \B ile

Escaping (outside character classes)

There are several characters that need to be escaped to be taken literally (at least outside char classes):

- Brackets: `[]`
- Parentheses: `()`
- Curly braces: `{}`
- Operators: `*`, `+`, `?`
- Anchors: `^`, `$`
- Others: `.`, `\`
- In order to use a literal `^` at the start or a literal `$` at the end of a regex, the character must be escaped.
- Some flavors only use `^` and `$` as metacharacters when they are at the start or end of the regex respectively. In those flavors, no additional escaping is necessary. It's usually just best to escape them anyway.

Parenthesis-Related Regular Expressi



- `(ABC)` — This will group multiple tokens together and remember the substring matched by them for later use. This is called a capturing group.
- `(?:ABC)` — This will also group multiple tokens together but won't remember the match. It is a non-capturing group.
- `\d+(?=ABC)` — This will match the token(s) preceding the `(?=ABC)` part only if it is followed by `ABC`. The part `ABC` will not be included in the match. The `\d` part is just an example. It could be any other regular expression string.
- `\d+(?!ABC)` — This will match the token(s) preceding the `(?!ABC)` part only if it is *not* followed by `ABC`. The part `ABC` will not be included in the match. The `\d` part is just an example. It could be any other regular expression string.

- **Positive lookahead:** `(?=pattern)`
- **Negative lookahead:** `(?!pattern)`
- **Positive lookbehind:** `(?<=pattern)`
- **Negative lookbehind:** `(?<!pattern)`

* Regex in Cpp: • include <regex>

• Regex olmadan bir tür eş ismi

• Constructor'a const char* veya string / cstring yollamalı

```
int main()
{
    using namespace std;

    regex rgx1{ "([a-z]{4})\\d{4}\\1" };
                    dada 1234 dada ise valid !


}
```

```
int main()
{
    using namespace std;

    string rstr{ "([a-z]{4})\\d{4}\\1" };

    regex rgx{rstr};

    cout << "regex string : " << rstr << '\n';
```

* Regex_match:

```
int main()
{
    using namespace std;

    string rstr{ "([a-z]{4})\\d{4}\\1" };

    regex rgx{rstr};

    cout << "regex string : " << rstr << '\n';

    for (int i = 0; i < 10; ++i) {

    string s;
    std::cout << "bir yazi girin: ";
    getline(cin, s);
    cout << "[" << s << "]\n";

    if (regex_match(s, rgx))
        std::cout << "uygun\n";
    else
        std::cout << "uygun degil\n";
    }


}
```

```cpp
#include <regex>
#include "nutility.h"


int main()
{
    using namespace std;

    auto vec = file_to_strvec("words.txt");

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }
    string rstr{ "([a-z]{3,})\\1" };

    regex rgx{rstr};

    ofs << "regex string : " << rstr << '\n';

    for (const auto& s : vec) {
        if (regex_match(s, rgx))
            ofs << s << '\n';
    }


}
```

* mark_count: →capture grup

```cpp
#include <regex>

int main()
{
    using namespace std;

    regex r7{ "n(?:eca)ti" };
    cout << "r7 icin alt ifade sayisi : " << r7.mark_count() << "\n";
}
```

* regex_search: string içerisinde kriterlere uyan en az 1 yeri var mı?
→ returns bool

```cpp
#include <regex>

int main()
{
    using namespace std;

    regex rgx{ "[artk]{3,}x\\d+" };

    string str{};
    std::cout << "bir yazi girin: ";
    getline(cin, str);
    cout << "[" << str << "]\n";

    cout << (regex_search(str, rgx) ? "valid" : "invalid") << "\n";


}
```

Smatch Container: → submatchler container'i

```cpp
int main()
{
    using namespace std;

    regex rgx{ "([a-f]{3,})x(\\d+)" };

    string str{"mustdefax9876sofia"};
    //              ---------

    cout << "[" << str << "]\n";

    smatch sm;
    //smatch submatch'lerin bir container'i


    if (regex_search(str, sm, rgx)) {
        cout << "sm.size() = " << sm.size() << "\n";
        for (size_t i{}; i < sm.size(); ++i) {
            //cout << "sm[" << i << "] = " << sm[i].str() << "\n";
            cout << "sm[" << i << "] = " << sm.str(i) << "\n";
        }

    }


}
```

3

→ tam anlamadım

* Sregex Iterator:

```cpp
int main()

{
    using namespace std;

    auto str = get_str_from_file("emails.txt");
    std::cout << "str.size() = " << str.size() << "\n";
    (void)getchar();
    cout << "\n";
    std::cout << str << "\n";

    regex rgx{ "([a-z0-9]+)@([a-z0-9]+)\\.com" };

    for (sregex_iterator iter{ str.begin(), str.end(), rgx }, end; iter != end; ++iter) {
        cout << iter->str(0) << "\n";
    }

}
```

→ 0 olunca tam substring yani emailin hepsini

→ 1 olunca @ isretine kadar

→ 2 olunca gmail / hotmail vs... @ 'den sonrası