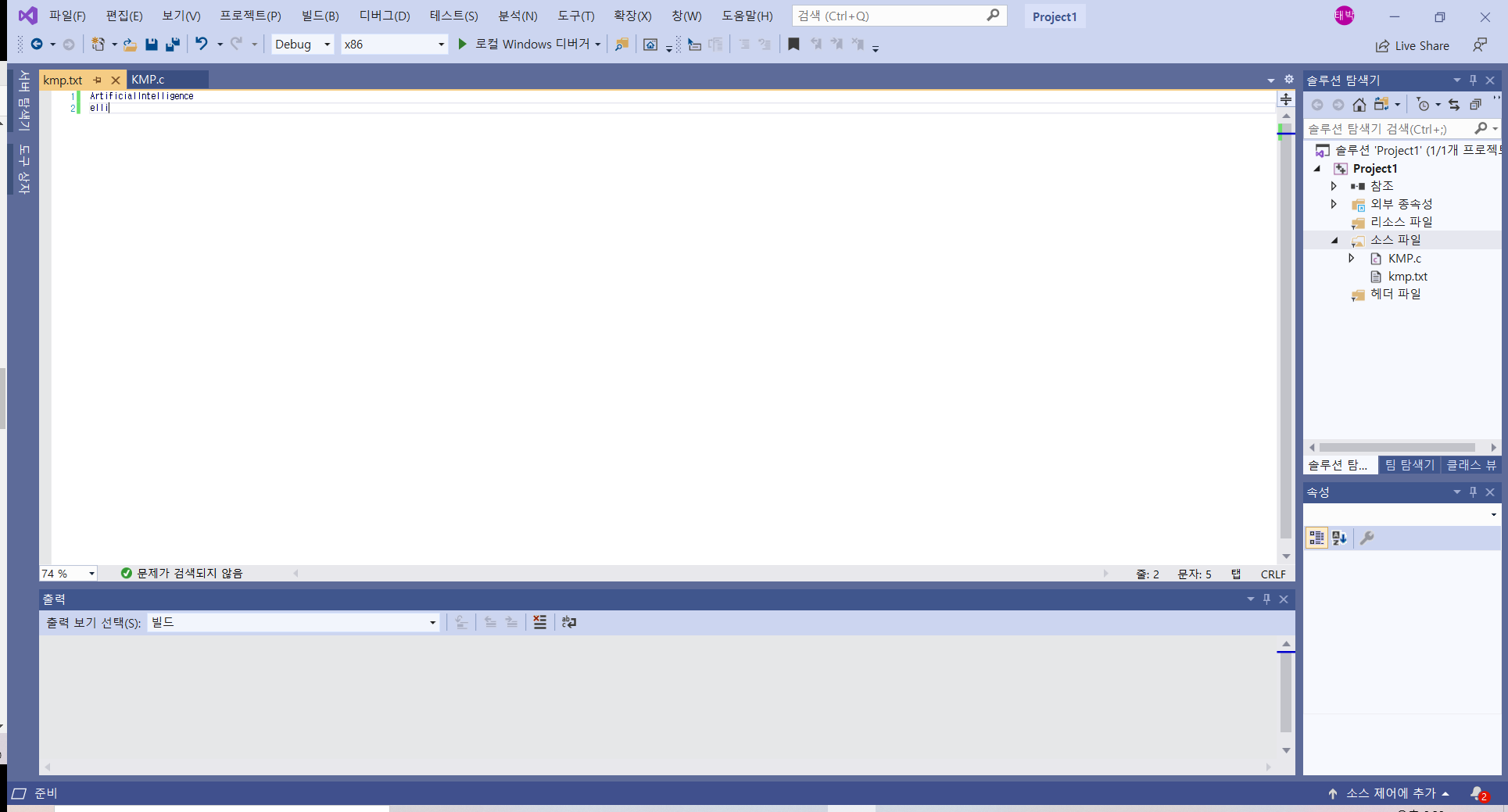
자료구조 오픈랩3

컴퓨터공학과 20171646 벅태윤



텍스트이(가) 표시된 사진

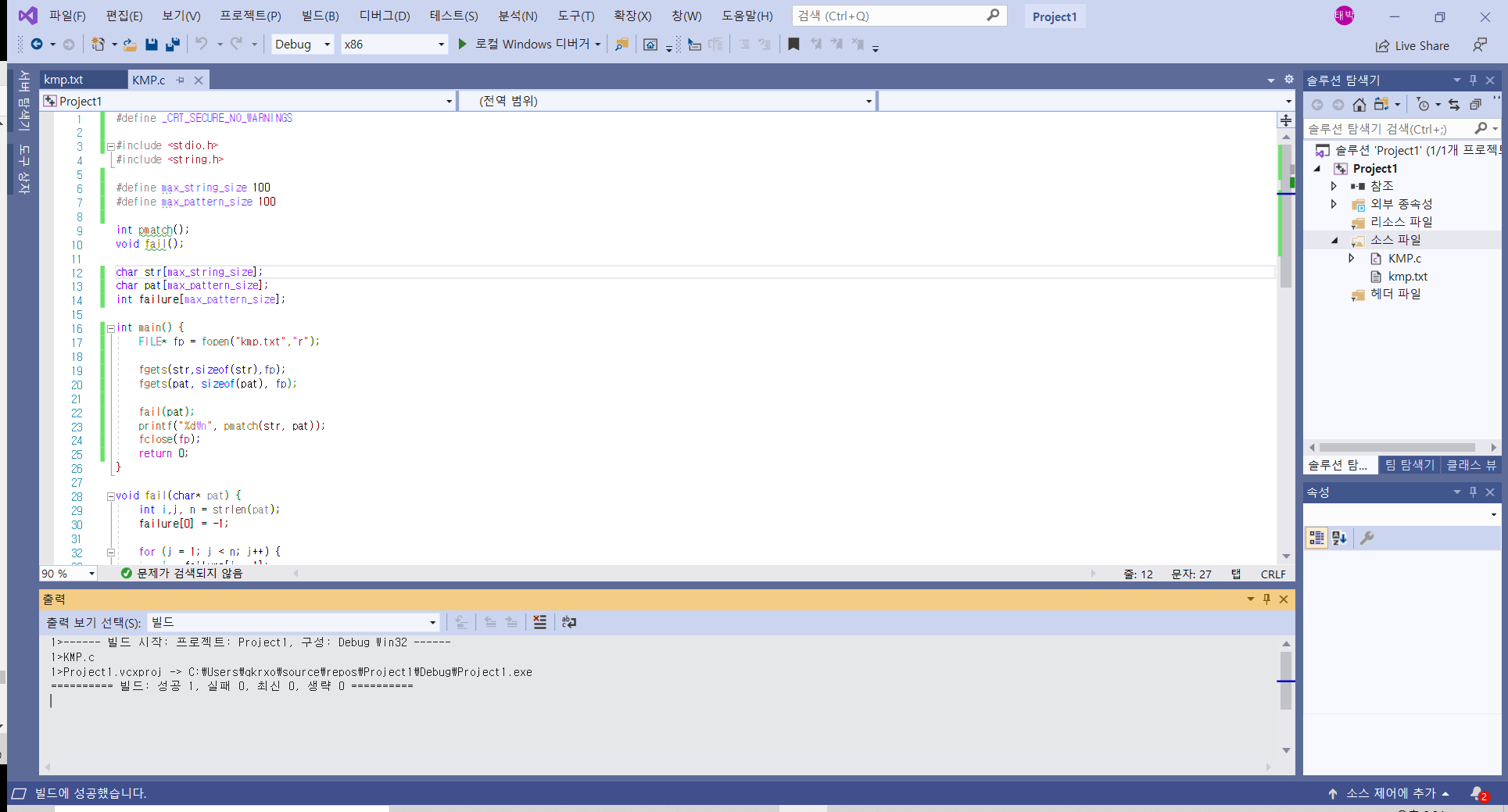
자동 생성된 설명

문자열과 패턴을 나타내는 “kmp.txt”라는 텍스트 파일과 텍스트 파일을 통해 입력을 받아 코드를 실행시킨 결과이다.

(코드 / 알고리즘 설명)

KMP알고리즘이란 문자열 str에서 문자열 pat를 찾아주는 알고리즘이다. 일반적으로 처음부터(index : 0) pat이랑 비교를 하다가 다른 부분이 나타나면 str의 그 다음 인덱스(index : 1)에서부터 다시 pat랑 비교를 하면서 찾아가는 방식을 생각할 수 있지만, 이는 효율성이 좋지 못하기 때문에 KMP알고리즘을 사용한다. 이 알고리즘을 사용하면 앞에서 본 경우처럼 처음부터 끝까지 하나하나 비교를 해 나가는 것이 아닌 str과 pat을 비교하다가 다른 부분이 나타나면 pat의 전체가 아닌 pat의 일부분만을 비교해서 탐색을 할 수 있다.

**- main**



다음 코드와 같이 int형 함수 pmatch와 void형 함수 fail을 선언하였고 전역변수로 char type배열 str과 pat, int type 배열 failure를 선언하였다. 각 배열의 size를 정의하였으며 failure배열은 pat(패턴)배열에 대응하는 것이기 때문에 같은 사이즈로 선언하였다. “kmp.txt”파일을 fopen을 통해 열었고 텍스트 파일을 보면 문자열과 패턴이 각각 첫번째 줄과 두번째 줄에 쓰여져 있기 때문에 ‘\n’까지 입력을 받는 fgets함수를 사용하여 str과 pat에 각각 문자열과 패턴을 넣어주었다. 이후 fail함수를 통해 failure배열에 값을 넣어주었고 pmatch함수의 return값을 출력하였다. 이 때, pmatch는 문자열이 패턴을 포함하고 있으면 문자열에서 패턴이 시작하는 index를, 포함하지 않고 있으면 -1을 반환한다.

**- fail**

스크린샷, 노트북, 컴퓨터, 모니터이(가) 표시된 사진

자동 생성된 설명

fail함수는 패턴에 따라 failure배열에 값을 넣어주는 역할을 한다. 처음에 failure[0] = -1으로 정해준다. 이후 반복문을 (패턴의길이 – 1)만큼 수행을 해주는데 이는 0을 제외한 나머지 index(1,2,3…)에 값을 넣어주기 위함이다. 반복문 처음에 i변수에 값을 지정해주는데 이는 failure[j-1]이고 이는 이전 인덱스의 failure값을 의미한다. 이후 while반복문이 나오고 이 반복문의 조건은 (pat[j] != pat[i+1]) && (i>=0)이다. 이후 조건문을 보는데 pat[j]랑 pat[i+1]이 같으면 failure[j]에 i+1값을, 그렇지 않으면 -1을 넣는다. 이 과정을 j < n일 때 까지 반복을 한다. 이 함수에서 중요한 부분은 while의 조건과 if문의 조건이다. if문의 조건을 먼저 살펴보면 pat[j] == pat[i+1]인데, 이것은 패턴에서 현재 j인덱스에 해당하는 값과 pat[failure[j-1]+1]이 같은지를 판단하는 조건이다. 이것이 필요한 이유는 다음과 같은 예시로 설명할 수 있다. ababa라는 패턴이 있다고 가정하면

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |
| pat | a | b | a | b | a |
| failure | -1(i) |  |  |  |  |

처음 failure는 무조건 -1이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |
| pat | a | b(j) | a | b | a |
| failure | -1(i) | -1 |  |  |  |

pat[i+1]랑 b랑 다른 값이므로 b에 해당하는 failure에도 -1을 넣는다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |
| pat | a | b | a(j) | b | a |
| failure | -1 | -1(i) | 0 |  |  |

pat[i+1]이랑 a랑 같은 값이므로 a에 해당하는 failure에 0(i+1, if문 실행)을 넣는다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |
| pat | a | b | a | b(j) | a |
| failure | -1 | -1 | 0(i) | 1 |  |

i = failure[j-1]이기 때문에 이번에는 0이다. 그러므로 pat[i+1]이랑 b랑 같은 값이므로 i+1인 1을 넣는다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |
| pat | a | b | a | b | a(j) |
| failure | -1 | -1 | 0 | 1(i) | 2 |

위와 같은 이유로 failure[4]는 2가 된다.

조건문은 위와 같은 방식으로 failure배열에 값을 넣어준다. 즉, 패턴을 읽다가 중간에 pat[0]부터 반복되는 부분이 있는지 없는지를 살펴보기 위해 있는 것이 failure배열인 것이다. 이어서 while문을 보면 다음과 같다.

while ((pat[j] != pat[i + 1]) && (i >= 0))

i = failure[i];

while문은 패턴 문자열 안에서 같은 부분이 반복이 되어 failure의 값을 증가시키다가 어느 일정 부분에서 반복되는 부분이랑 달라지는 부분이 생기면 i값을 뒤로 돌리기 위해 있는 반복문이다. 앞에서 본 예시 뒤에 c를 추가한 ababac라는 패턴을 보면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |  |
| pat | a | b | a | b | a | c(j) |
| failure | -1 | -1 | 0 | 1 | 2(i) |  |

pat[5]는 반복되는 부분이 아니다. while문의 조건인 pat[j] != pat[i+1] && i >= 0을 만족시키기 때문에 반복문 안의 i = failure[i];문장을 실행한다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |  |
| pat | a | b | a | b | a | c(j) |
| failure | -1 | -1 | 0(i) | 1 | 2 |  |

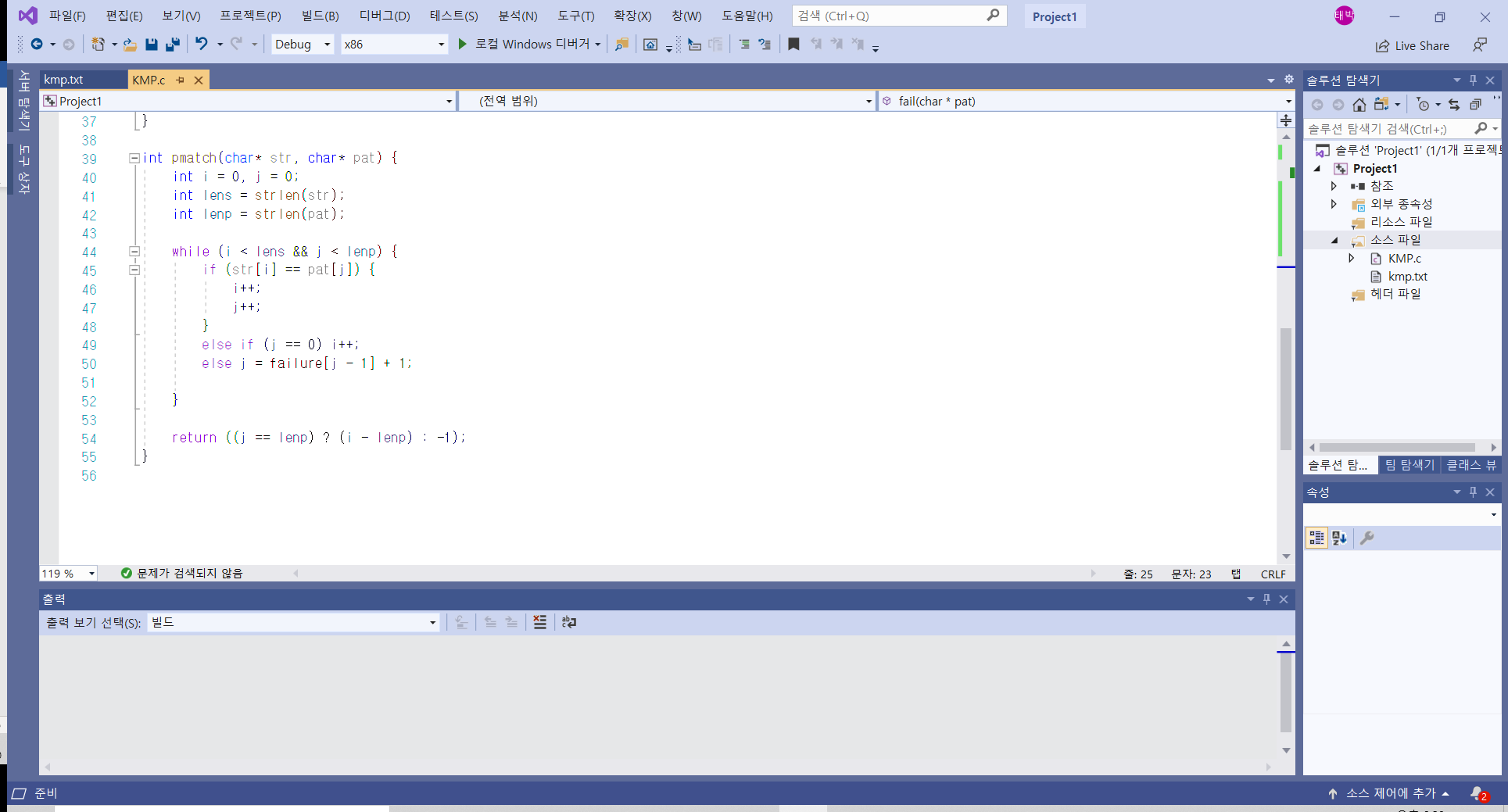
위의 i = failure[i]문장을 실행한 결과이다. 여전히 while문의 조건에 부합하므로 i=failure[i]를 한 번 더 실행한다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pat[i+1] |  |  |  |  |  |  |
| pat | a | b | a | b | a | c(j) |
| failure | -1(i) | -1 | 0 | 1 | 2 | -1 |

최종적으로 i=0으로 되돌아가고, while문의 i>=0 조건을 만족시키지 못하므로 반복문을 탈출한다. 이후 (if~else)문에 의해 failure[5] = -1이라는 값을 가지게 된다.

이러한 방식을 통해 KMP알고리즘을 제대로 수행할 수 있도록 failure배열에 pat안에서 반복되는 부분이 있는지 없는지를 판단할 수 있도록 값을 넣어준다.

**-pmatch**



str안에 pat와 같은 부분이 존재하는지 아닌지를 판단해주고 존재하면 존재하는 부분의 시작 index를 반환해주는 함수이다. str에 대응되는 변수 i, pat에 대응되는 변수 j를 선언하여 i 또는 j가 각각 str과 pat의 길이를 넘어설 때까지 while문을 반복한다. 반복문은 str[i] == pat[j]인 부분이 나타나면 i와 j 둘 다 1씩 증가시키고, 그렇지 않을 때 만일 j가 0, 즉 패턴의 시작 부분부터 비교 중 이였다면 str의 다음 인덱스부터 비교를 하기 위해 i를 1 증가, 그렇지 않고 패턴의 중간 부분이랑 str을 비교하다가 다른 부분이 나타나면 j = failure[j-1] + 1을 수행한다. 가장 중요한 부분은 else문이다. 예시를 들면 다음과 같이 동작함을 이해할 수 있다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| str | a | b | a | b | a | b | c |
| pat | a | b | a | b | c |  |  |
| failure | -1 | -1 | 0 | 1 | 2 |  |  |

표시된 부분처럼 인덱스 0부터 3까지, 총 4부분이 같이 때문에 i=4,j=4값이 들어간 이후 4번째 인덱스를 본다. 이 때 str[4] = a, pat[4] = c로 다르기 때문에 else문을 수행한다. j = failure[j-1] + 1에 의해서 j = 2가 되고 다음 비교는

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| str | a | b | a | b | a | b | c |
| pat |  |  | a | b | a | b | c |
| failure |  |  | -1 | -1 | 0 | 1 | 2 |

인덱스 1에서부터가 아닌 2에서부터 비교를 하게 된다.

정리를 하면 KMP알고리즘은 처음부터 끝까지 하나하나 비교해 나가는 방식이 아닌 failure배열의 값을 통해 pat안에서 반복되는 부분을 표시하여 효율적으로 비교를 하고 str에 pat가 존재하는지 아닌지, 존재한다면 몇 번째 인덱스에 위치해 있는지를 판단해주는 알고리즘이다.