

과제 2

문자열 구현하기
계산기
scanf 만들기

20232907 정현승

목차

1. 문제 1: 문자열 구현하기	3
2. 문제 1 해결 방안	3
3. 문제 1의 결과	4
4. 문제 2: 계산기	5
5. 문제 2 해결 방안	5
6. 문제 2의 결과	7
7. 문제 3: scanf 만들기	10
8. 문제 3 해결 방안	11
9. 문제 3의 결과	12
10. 소스코드	13

1. 문제 1: 문자열 구현하기

C 언어를 이용하여, 기존 문자열 저장 방식의 `strlen()` 함수가 오래 걸린다는 문제를 해결하는 새로운 문자열 자료형을 생성한다.

2. 문제 1 해결 방안

이 문제는 단순히 **문자열의 길이를 문자와 같이 저장**하는 단순한 방식으로 해결할 수 있다. 따라서 문자열의 길이를 문자열 포인터와 같이 구조체에 저장하는 것으로 문제를 해결했다. 다만 문자열을 동적 할당받는 것을 선택한바, 동적으로 할당받은 길이까지 구조체에 같이 저장하였다.

저번 프로그래밍 과제 3에 썼던 코드를 또다시 가져왔는데, 문제 3과 달리 이 문제에서는 `calloc`, `realloc`, `free` 기능만을 활용하기 위해 가져왔다. (그 코드의 핵심인 `sgets` 함수는 이 과제에서 사용하지 않는다.)

우선 문자열을 입력받는 `new_gets` 함수는, 저번 프로그래밍 과제 3에 썼던 문자열 입력 함수를 가져와 조금만 수정해 사용했다. 매번 할당 또는 재할당할 때마다 구조체에 저장하는, 할당받은 길이와 현재 문자열의 길이를 변경하였다. 단 한 번 입력을 받고 문자열의 현재 길이를 업데이트 할 때는, 어차피 문자를 하나하나 검사해야 하는 특성상, 기존의 `strlen` 함수를 사용했다.

`create_str` 함수는 기존 저장된 문자열(기존 방식으로 저장된 문자열)을 새로운 방식의 문자열로 변경하는 함수로, 이때도 역시 기존의 `strlen` 함수를 이용했다. 문자열을 포인터만 복사하면 나중에 `free` 하거나 `realloc` 할 때 문제가 생길 수 있으므로 문자 내용을 하나하나 복사하는 방법을 선택했다. 다만 중복 코드 방지를 위해 복사 자체는 후술할 `new_strcpy` 함수에서 진행했다. 이번 코드에서 유일하게 사용한 `strlen` 함수는 이 두 개로 끝이다. (include 한 코드에서 `strlen` 함수가 더 사용되나 해당 코드가 포함된 함수를 사용하지 않았으므로 문제는 없다.)

이후는 전후 새로운 문자열을 다루는 함수이다.

`new_strlen` 함수는 새로운 방식으로 저장된 문자열의 길이를 반환하는 것이다. 앞에서 언급했듯 구조체에 문자열의 길이도 전부 저장되어 있으니 그 값만 반환해 주면 되며, 실제 코드도 한 줄 짜리 코드로 다른 기능을 수행하지 않는다. (단순히 Java의 `getter` 수준이다.) 따라서 **$O(1)$ 을 만족**한다.

`new_strcpy` 함수는 문자열을 복사하는 함수로, 우선 기존 문자열은 더 이상 사용하지 않음으로 할당 해제 등 초기화를 진행하고, 이후 저장할 문자열에 저장할 공간을 동적으로 할당받은 뒤 기존 문자열을 `memcpy`로 복사했다. 정확한 길이를 별도로 저장해 놓기 때문에 굳이 `strcpy` 함수를 사용하지 않아도 된다. 복사된 문자열의 길이와 할당받은 크기도 같이 구조체에 저장하면 문자열

복사는 끝난다. 복사한 내용이 들어갈 구조체는 call by value가 아니라 call by reference로 진행하였으므로, return 값은 필요 없다.

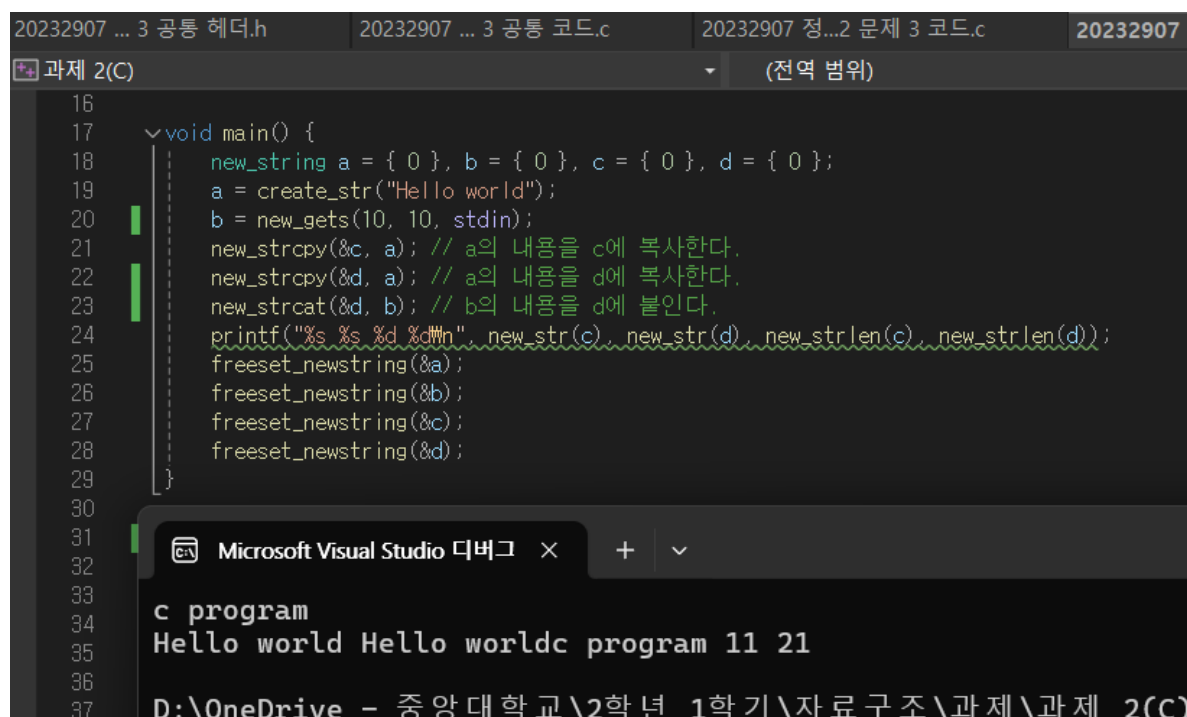
new_strcat 함수는 한 문자열을 다른 문자열 뒤에 붙이는 함수로, 기존 문자열이 NULL이면 new_strcpy를 시행만 하고 끝냈다. 기존 문자열에 뭔가 있지만 새로운 문자열을 붙이기에는 부족하면 realloc을 한 뒤 문자열을 붙였다. 기존 문자열이 저장된 공간을 버리고 새로운 공간에 기존 문자열과 새로운 문자열 모두를 붙일 수도 있었지만, realloc 함수는 기존에 저장된 정보를 복사하는 기능도 가지고 있기에, 굳이 코드를 복잡하게 만들 필요성을 느끼지 못하여 realloc 함수를 사용했다. 이 외는 new_strcpy 함수와 다른 점은, 복사의 시작점을 붙여 넣을 문자열의 시작 부분이 아닌, 붙여 넣을 문자열에 기존 문자열의 길이를 더한 곳으로 바꾼 것 말고는 없다.

new_str 함수는 위 new_strlen 함수와 비슷하게 새로운 방식으로 저장된 문자열을 기존 방식으로 바꾸는 함수이며, 콘솔 출력 등 기존 방식으로 처리해야 하는 게 필요할 때 사용한다.

마지막으로 freeset_newstring 함수는 새로운 방식으로 저장된 문자열을 초기화하는 함수이다. 특히 동적 해제를 진행하기 위해 사용하는 함수이다.

위와 같이 구조체를 이용하여 문제 1을 해결하였다. 그 결과는 다음과 같다.

3. 문제 1의 결과



```
20232907 ... 3 공통 헤더.h | 20232907 ... 3 공통 코드.c | 20232907 정...2 문제 3 코드.c | 20232907
과제 2(C) | (전역 범위)
16
17 void main() {
18     new_string a = { 0 }, b = { 0 }, c = { 0 }, d = { 0 };
19     a = create_str("Hello world");
20     b = new_gets(10, 10, stdin);
21     new_strcpy(&c, a); // a의 내용을 c에 복사한다.
22     new_strcpy(&d, a); // a의 내용을 d에 복사한다.
23     new_strcat(&d, b); // b의 내용을 d에 붙인다.
24     printf("%s %s %d %d\n", new_str(c), new_str(d), new_strlen(c), new_strlen(d));
25     freeset_newstring(&a);
26     freeset_newstring(&b);
27     freeset_newstring(&c);
28     freeset_newstring(&d);
29 }
30
31 Microsoft Visual Studio 디버그
32
33 c program
34 Hello world Hello worldc program 11 21
35
36 D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 2(C)
```

그림 3.1

결과의 첫째 줄은 입력이고 둘째 줄은 출력이다. b의 입력을, 콘솔을 통해 받았다는 점을 빼면 과제의 예시와 완전히 같으며, 이를 통해 정상 작동한다는 점을 알 수 있다.

4. 문제 2: 계산기

Python을 이용하여, 문자열로 입력받은 수식을 직접 계산하는 프로그램을 제작한다. 사용되는 연산자는 `+`, `-`, `*`, `/`, `(`, `)`, `^^`(제곱)이다. 입력되는 숫자는 실수형이며 수식이 잘못 입력되었으면 틀린 위치를 알려준다.

5. 문제 2 해결 방안

우선 문자열을 입력받은 뒤, 문자열에 저장된 문자를 하나하나 확인하며 연산을 진행했다. 이때 문자열에서 현재 문자의 index 값이 필요하기 때문에 `for i in str` 대신 `for i in range`를 사용했다.

반복문을 들어가기 전 각 변수에 관해 설명하자면, `instr`은 입력받은 문자열, `codelist`는 문자와 기호를 저장할 스택, `pointflag`는 현재 숫자에 소수점이 있었는지를 저장하는 변수(0과 1만 사용한다. 이를 선언할 당시 python에 `boolean`을 별도의 `import` 없이 사용할 수 있다는 사실을 기억하지 못했다.), `oustr`은 결과값 문자열을 저장할 공간(에러메시지 또는 최종 연산 결과, 아직 저장되기 전에는 빈 문자열이다.), `startpt`는 `instr`에서의 숫자의 시작지점(index), `endpt`는 `instr`에서의 숫자의 종료지점(index) 값이다. `startpt`와 `endpt`는 지정되지 않았으면 -1이 저장된다. 이때 `startpt`는 숫자의 시작지점 외에도 일부 정보를 더 저장하는데, 이 변수의 값이 -2면 이전에 '`^`' 문자가 입력된 것('은 단독으로 입력될 수 없고 꼭 두 개가 같이 입력되어야 한다.), -3이면 이전에 ')' 문자가 입력된 것이다('문자는 다른 문자와 달리 뒤에 숫자가 아닌 연산자가 와야 한다).

이 문자 하나당 실행하는 과정은, 이전 반복문에서 에러 메시지가 저장되었으면 더 이상 계산하는 의미가 없기에 반복문을 종료하는 것부터 시작한다(이후 반복문 내에서 `break` 문 사용을 중복해서 하지 않고 한 번에 처리하기 위함이다). `startpt`가 -2인 상태에서 이번 문자가 '`^`'이 아니면 오류로 판단하고 에러 메시지를 지정한다. (앞서 언급했듯 '`^`'은 단독으로 쓰일 수 없다.) 이번 문자열이 공백 문자이면, 숫자가 시작한 상태에서 숫자의 끝이 지정되지 않았으면 숫자의 끝을 지정해 주는 것 정도만 시행한다. 이번 문자열이 숫자이면, 숫자가 시작하지 않았으면 숫자 시작 지점을 지정하고, `startpt`가 -3이면(숫자가 아닌 연산자가 와야 한다) 에러 메시지 지정, `endpt`가 이미 지정되었으면 공백문자로 분리된 숫자 2개가 있는 것이므로 역시 에러 메시지를 지정하였다. 이번 문자열이 소수점이면, `startpt`가 지정되지 않았으면 에러 메시지를 넣고(단 `startpt`가 -2인 건 앞에서 처리했으니, 별도의 처리가 필요 없다) `pointflag`가 꺼져 있으면 키고, 이미 켜져 있다면 에러 메시지를 넣었다. 이외는 연산자 또는 연산할 수 없는 문자이다. 우선 이번에 입력된 문자가 '('인지 또는 `startpt`가 -1인지 (`startpt`가 -2, -3이거나, '('가 오는 경우는 연산자가 두 번 이상 연속해야 하는 경우이다) 확인해, 그렇지 않다면 에러 메시지를 넣는다. 이후 연산자가 연속해야 하는 경우가 아니면 지정된 `startpt`와 `endpt`를 따라 숫자를 분리해 `float` 형으로 변경하여 `codelist`에 `append` 한다. 이때 `endpt`만 지정되어 있지 않다면 현재의 index를 `endpt`로 지정한다. 이후 `endpt`와 `startpt`를 -1, `pointflag`를 0으로 리셋한다. 단 `startpt`는 직전에 '`^`'가 들어왔다는 의미인,

startpt에 -2가 저장되어 있으면 여기서 리셋하지 않는다.

이후 이번의 문자(연산자)별로 처리가 다른데, +, -, *, /이나 ')'가 들어오면 codelist에 저장된 각종 연산을 수행한다. 이 수행 방법은 codelist에서 element 3개를 pop하고, 두 번째로 pop 된 연산자에 따라 알맞은 연산을 수행한다. 이 수행은 연산자 부분에 '('가 나오거나, codelist에 남은 element가 하나밖에 없거나(이 말은 지금까지의 모든 입력에 대한 연산을 수행하였다는 말이 된다), *이나 /가 이번 연산자(instr[i])이면서 이번에 수행해야 하는 연산(code)이 우선도가 낮은 +나 -가 나오면 수행을 종료한다. 단 이번 연산자(instr[i])가 ')'이면 codelist에 남은 element 수는 상관하지 않고 계산하며, 예외가 발생하면 '('가 없는 문제로 보고 에러 메시지를 지정했다. 즉 지금까지 입력된 연산 중 뒤에 나오는 연산자보다 우선도가 높아 **나중에 한 번에 연산한다 해도 먼저 연산하게 되는 건 미리미리 연산하는 것이다**. 앞에서의 codelist에 저장되는 문자의 순서도 숫자와 연산자가 번갈아 오도록 배치했기 때문에 (단 '(' 제외, ')'는 어차피 codelist에 추가되지 않는다) 이 때문에 문제가 생길 일도 없다. 또한 괄호가 여러 개 나오는 상황에도 ')'가 나올 때마다 자동으로 가장 최근에 입력된 '('까지 연산하게 되니 괄호가 여러 번 나오는 상황도 대비할 수 있다. 이렇게 수행이 모두 끝났으면 이번 연산자(instr[i])가 ')'가 아닌 경우 이번에 입력된 연산자를 codelist에 추가하는 것만 진행하고, ')'인 경우 쌍이 되는 '('를 codelist에서 제거해 준 후 이후에 연산자가 연속해야 하므로 startpt 값을 -3으로 변경한다.

이번의 문자(연산자, instr[i])로 '^'가 들어오면, '^'는 2개가 연속으로 들어와야 의미가 있으므로, 첫 번째 '^'이면 startpt를 -2로 지정해 다음 '^' 확인 시에 처리할 수 있도록 하고, 두 번째 '^'이면 codelist에 연산자를 추가하고 startpt를 -1로 리셋했다. '^'가 첫 번째인지 두 번째인지는 startpt 값으로 구분했다(이는 앞에서 startpt가 -2이면 리셋을 하지 않는 이유이기도 하다). 위에 언급했던 각종 연산은 이때에는 수행하지 않는다. (다음 codelist 연산 시에 한 번에 시행한다.) 따라서 테스트는 하지 않았지만, '^'^ 연산자가 2번 이상 들어가게 되면, 뒤에서부터 연산하게 된다. 제공의 의미를 보면 제공 연산자가 연속으로 올 때는 이런 식으로 하는 게 맞다고 생각한다. 즉, 단순히 연산을 건너뛰었을 뿐인데, '^'^ 연산자가 2회 이상 들어올 때 뒤에서부터 연산하는 걸 구현하게 된 것이다. 다만 이는 요구사항이 아니기 때문에 테스트는 진행하지 않았다.

이번의 문자(연산자, instr[i])로 '('가 들어오면, 연산할 게 없으므로 이것을 codelist에 넣어주는 것으로 끝낸다.

이 이외의 문자이면 오류로 판단하고 에러 메시지를 지정했다.

위의 과정이 문자 하나 당 실행하는 과정이며, 입력된 문장의 끝까지 이를 반복한다.

이후 에러 메시지가 지정되어 위의 반복문이 종료된 게 아니면 마지막으로 codelist에 저장되어 있는 각종 연산을 수행한다. 수행 전 먼저 입력된 숫자가 아직 codelist에 없을 수 있으므로 startpt가 지정되었으면 codelist에 넣어준다. 이후 codelist에 남은 element가 하나일 때까지 위

연산을 수행하며, 연산하는 숫자가 잘못 지정되는 등 각종 예외 상황이 발생하면 예외 처리를 진행해 오류메시지를 지정했다(다만 이 단계에서 예외가 되는 상황은 입력된 수식이 연산자로 끝났거나, 마지막 ')'가 없는 경우뿐이다). 참고로 닫는 괄호가 나오면 쌍이 되는 여는 괄호도 같이 처리하므로 정상적인 입력이면 괄호는 남아 있을 수 없다.

마지막으로 연산 결과를 담는데, 실행을 해 보니 **소수점 13자리 이하의 값이 정확하지 않은 문제를 발견**하였다. 이를 코드의 문제가 아니라 **float형의 유효자리 문제로** 판단하고 14자리부터는 표시되지 않도록 round를 진행하였다. 또 결과가 정수가 되면 소수점 한자리가 출력되는 문제도 발생하여, round로 만든 정수의 형태(다만 자료형은 float이다)가 기존 결과와 같으면, 최종 결과가 정수가 되도록 중간에 int 형 형 변환도 시행하였다.

마지막으로 결과를 출력하는 방법으로 문제 2를 해결하였다.

6. 문제 2의 결과

아래 그림 6.1~그림 6.7은 그림 6.6을 제외하고 요구사항의 입출력 예시를 그대로 따라 작성한 것이다. 그림 6.6은 그림 6.5의 수식을 정상으로 만든 것으로, 괄호와 '^' 연산을 요구사항의 예시만으로는 정상 작동하는지 파악할 수 없어 추가하였다. 모두 정상 작동한다는 점을 알 수 있다.

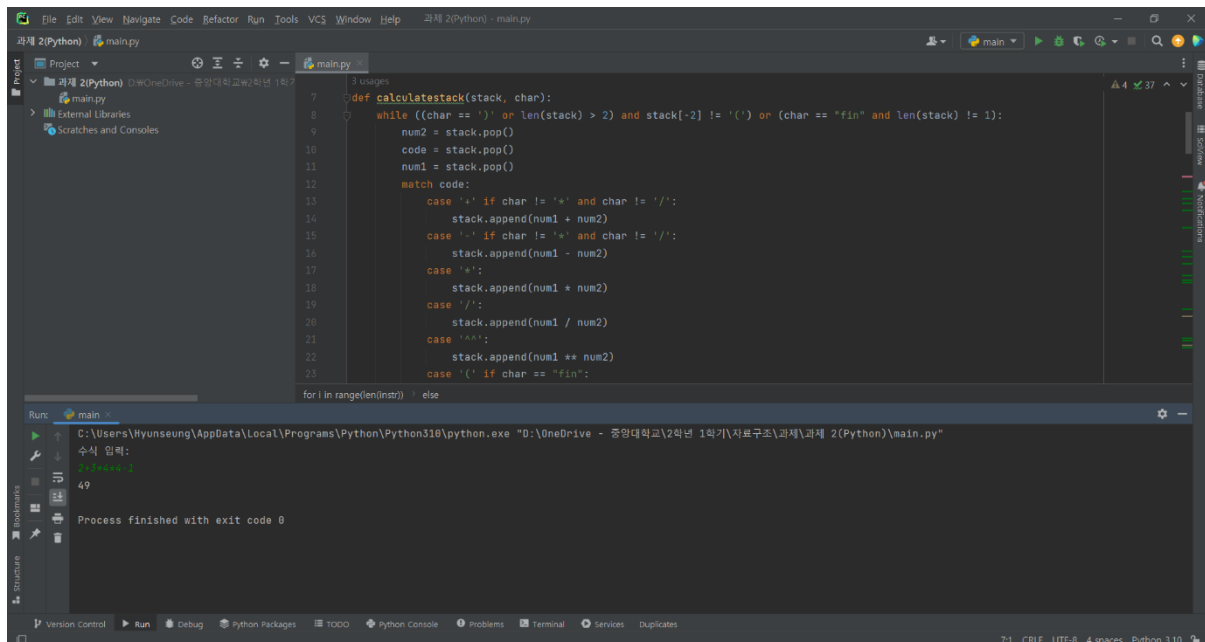


그림 6.1

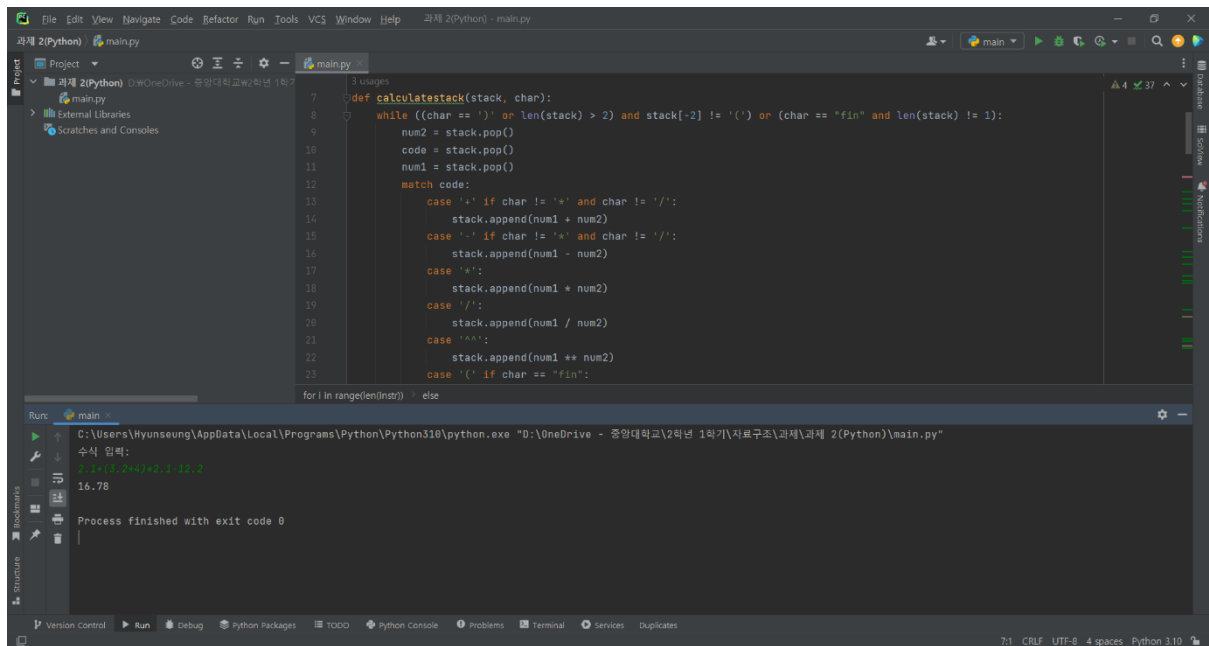


그림 6.2

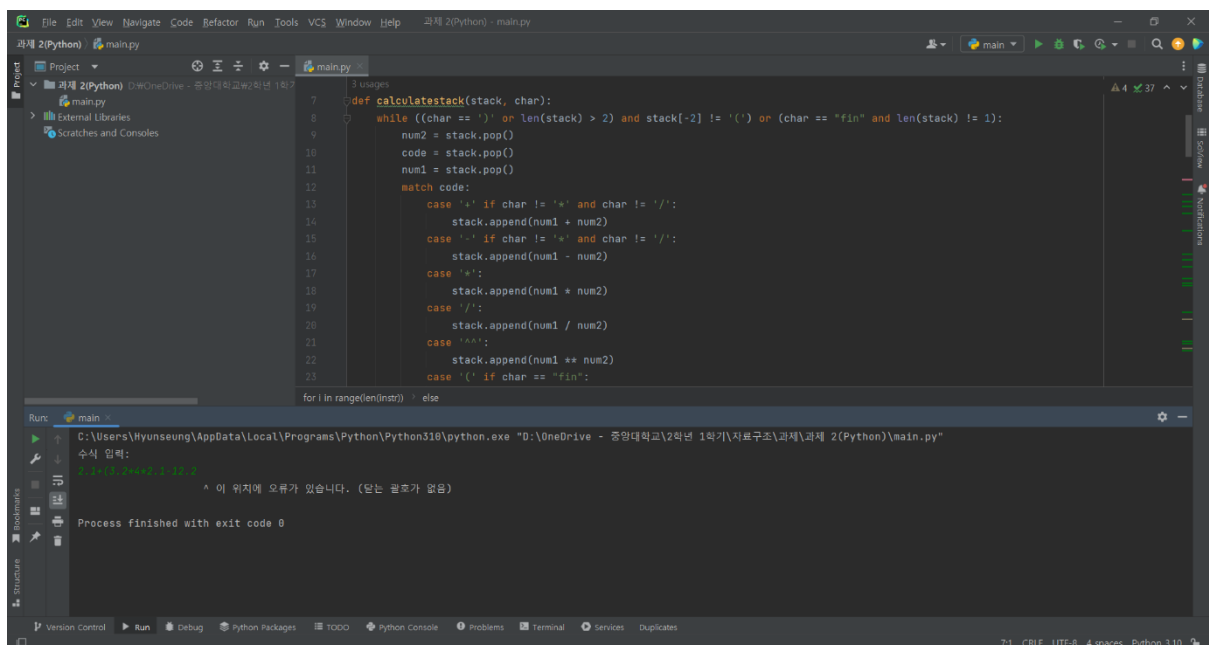


그림 6.3

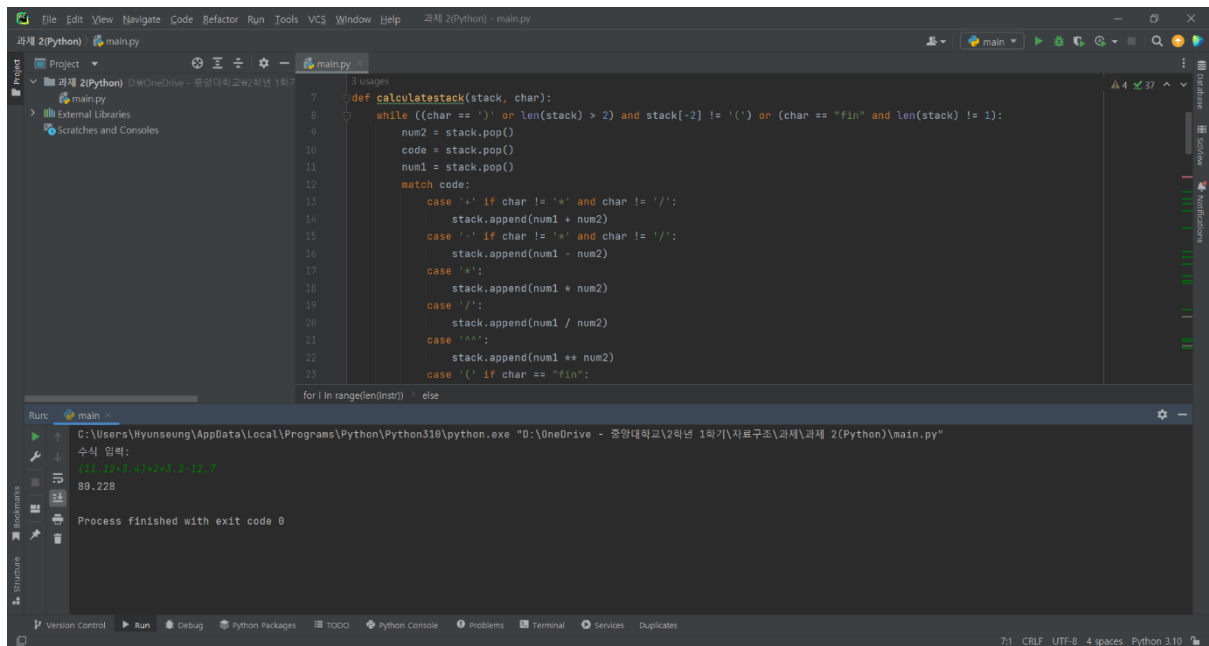


그림 6.4

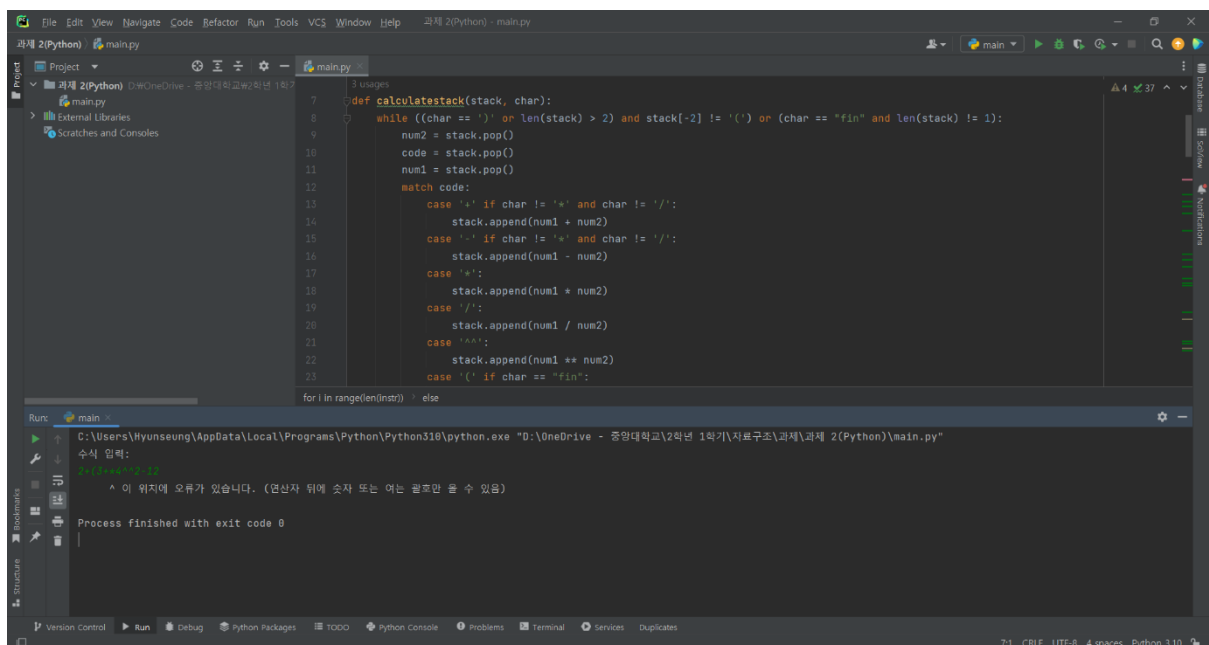


그림 6.5

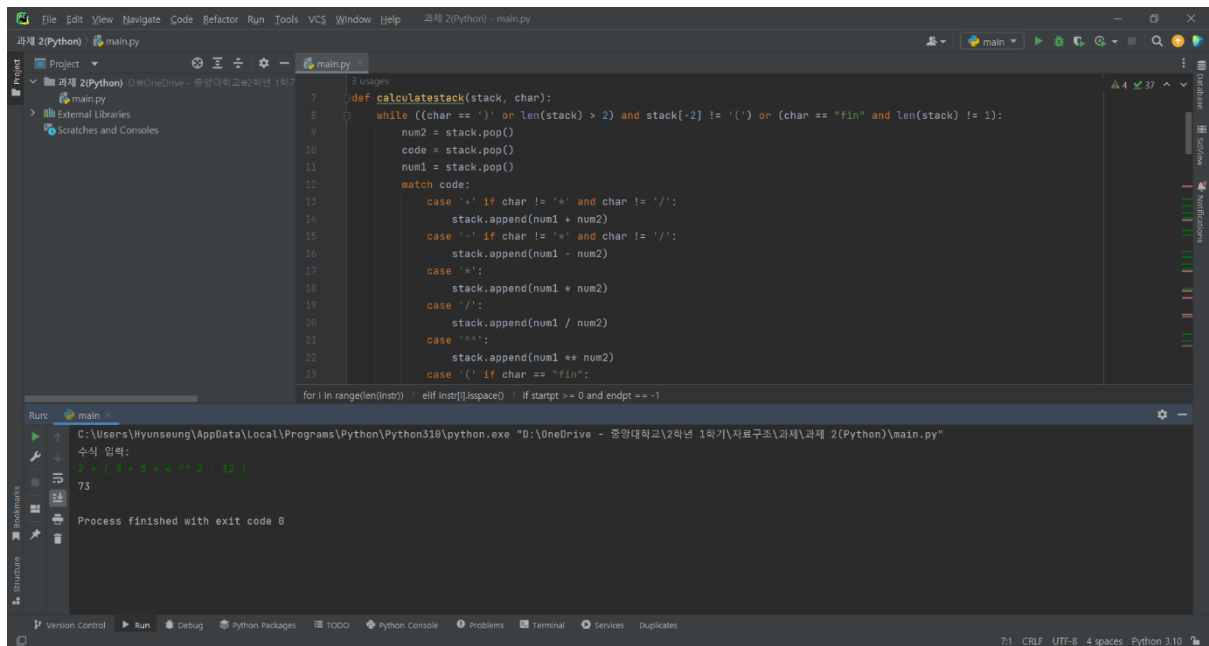


그림 6.6

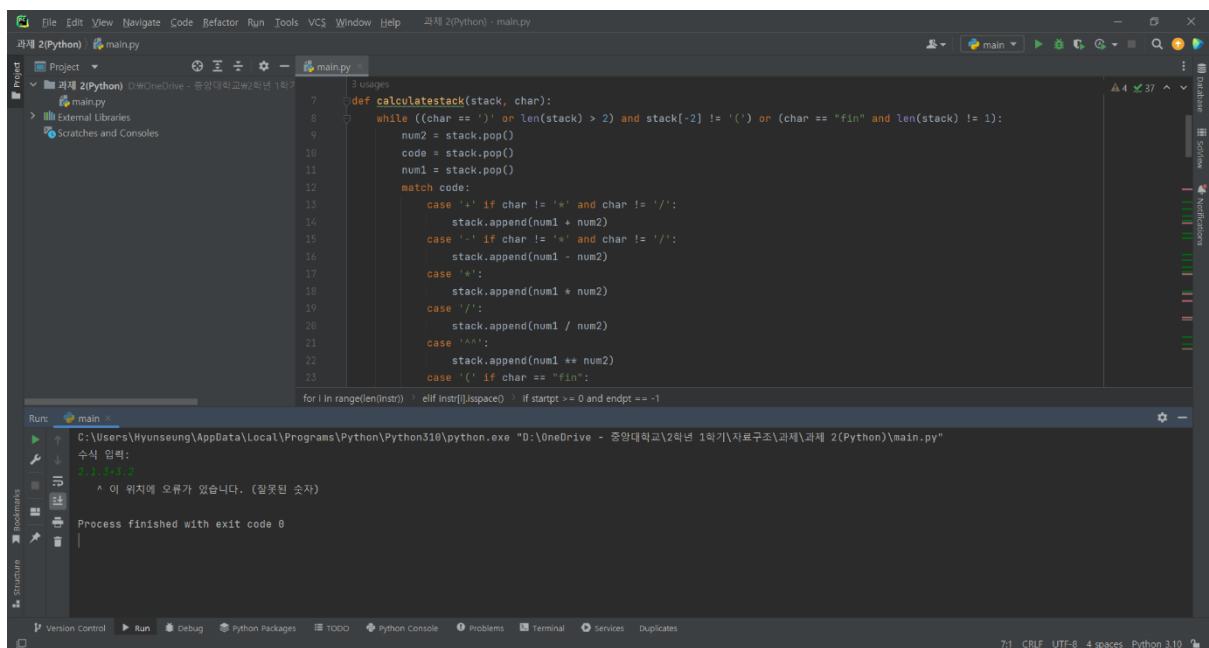


그림 6.7

7. 문제 3: scanf 만들기

C 언어의 함수 중 scanf 함수를 제작한다. 우선 정수(%d)와 실수(%f)만 구현하고, 이때 scanf 계열의 함수를 사용할 수 없다.

8. 문제 3 해결 방안

우선 구현 범위를 정하러 요구받았는데, 필수 요구사항인 **%d와 %f만 구현**하기로 했다. 이 외에도 전부 구현하려 하면 코드가 너무 복잡해질 수 있기 때문이다. %s는 막상 구현하면 난이도가 치솟을 것을 예상했으며, %f는 % 뒤에 문자가 2개 들어가는 관계로 switch 문을 어느 정도 개조해야 하는 문제가 있다. 무엇보다 가변 인자같이 처음 써 보는 기능도 있는 이 문제에 요구사항 이상으로 구현하면 시간이 매우 오래 소요될 것으로 예상해 일부러 필수 요구사항 2개에만 초점을 두었다. (다만 입력받은 문자열에 strtok 함수를 사용하지 않는 등 다른 기능이 추가될 것을 고려한 설계는, 기존 설계에 큰 영향이 없는 한 어느 정도 해 두긴 했다.)

my_scanf 함수의 변수부터 설명하면 input은 콘솔 입력받은 문자열을 저장하는 포인터(동적 할당으로 처리하기에, 한 번 할당받았으면 free 하기 전까지 움직이지 않는다), inputpos는 움직이지 못하는 input 대신 input에 저장된 문자열을 하나하나 읽는 포인터(즉, 커서), res는 기존 scanf 함수의 반환 값이자 읽어야 하는 변수의 개수, pos는 움직이지 못하는 args 대신 args에 저장된 문자열을 하나하나 읽는 포인터(즉, 커서)임과 동시에 숫자를 int 또는 float 형으로 바꿀 때 문자열에서 숫자의 시작 부분을 가리키는 포인터, end는 그 숫자의 끝부분을 가리키는 포인터, tempstr은 숫자로 바꿀 문자열을 임시로 저장하는 곳(atoi, atof 함수 사용을 위해서는 정확히 숫자만 문자열 내에 있어야 한다. 따라서 숫자만 따로 복사하는 것이다.)이면서 res 값을 구할 때 사용하는 임시 변수, toput은 숫자 값을 넣어야 하는, 가변 인자로 들어온 int 형 또는 float 형 포인터, 마지막으로 ap는 가변인자에 활용하는 변수이다.

우선 입력받아야 하는 변수의 개수를 구한다. scanf에는 %, %.3f같이 %가 연속으로 오는 일도, 중간에 숫자나 문자가 들어가는 일도 없다고 가정하고, '%'의 개수로 입력받아야 하는 변수의 수를 구했다. 이후 이 변수를 이용해 가변 인자를 시작했다(va_start). 이후 입력받는 과정을 그 개수만큼 반복했다. 아래의 입력받는 과정은 args의 커서 pos와 input의 커서 inputpos를 동시에 움직이므로, 둘을 헷갈리지 않게 주의가 필요하며, 헷갈리지 않기 위한 장치로 커서를 언급할 때마다 그게 어떤 문자열의 커서인지도 같이 언급한다.

입력받는 과정은 먼저 이번에 입력받아야 하는 자료형을 찾는 것부터 시작한다. 커서인 pos가 args에서 '%'를 가리킬 때까지 이동한다. 이후 pos와 d/f 글자 사이 존재할 수 있는 공백마저 지워주기 위해 공백이 존재하면 그만큼 또 pos(args) 커서를 옮겨준다. 이후에는 정수 입력과 실수 입력의 처리가 다르지만, %s로의 확장을 대비한 것일 뿐 비슷한 부분이 매우 많으니 같이 서술한다. 만약 이 함수가 처음 실행되어 input이 NULL이면, 콘솔 입력을 받고 inputpos를 설정한다. 이때 콘솔 입력을 받는 건 저번 프로그래밍 과제 3의 코드를 또 가져왔다. 이후 inputpos(input)의 커서를 숫자, +, -가 나올 때까지 이동했으며 input 문자열의 끝에 도달했으면 input에 저장된 공간을 해제하고 콘솔 입력을 다시 받았다. 숫자 또는 부호의 시작점에 도달했으니, inputpos(input) 점은 가만히 놔두고 end에 inputpos 커서 위치를 넣는다. 이후 end 커서로 inputpos의 숫자 끝부

분을 찾는데, 첫 문자는 숫자뿐만 아니라 부호가 와도 숫자의 일부로 보았으며, 그 이후 숫자가 나오지 않을 때까지 end 커서를 이동했다. 여기에 자료형이 float이면, 숫자가 아닌 소수점이 들어올 때 첫 번째에만 숫자의 일부로 보았다. 문제 2와는 달리 '도 소수점으로 보았다(실제로 서양에서는 소수점을 '로 쓰는 것으로 알고 있다). 소수점이 첫 번째로 등장하는지 두 번째 등장인지는, 다른 게 아니라 코드를 복사해서, 즉 소수점 이전과 이후에 숫자 확인하는 코드를 소수점 확인 코드 직전과 직후에 따로 두어 해결하였다. 이제 int와 float 형 공통으로, tempstr에 문자열로 된 숫자를 복사하고(복사할 공간을 동적 할당받는 것 포함), va_arg로 각 포인터를 받은 뒤 숫자로 변경해 **가변 인자로 받은 포인터에 저장했다**. 이후 tempstr에 할당받은 공간을 해제하고 inputpos(input)를 end가 가리키는 커서로 옮기면 입력은 끝이다. 이를 res번 반복하면 모든 입력을 처리할 수 있다.

위의 과정을 모두 끝냈으면 가변 인자를 종료해 주고(va_end), scanf 함수처럼 res 값을 반환했다. 위의 과정으로 문제 3을 해결했으며, 그 결과는 다음과 같다.

9. 문제 3의 결과

The screenshot shows a C program in Visual Studio. The code defines a `my_scanf` function that reads input into `int` and `float` variables. The output window shows the results of running the program with the input `-2.3 +4.5 -6`.

```

20232907 ... 3 공통 헤더.h | 20232907 ... 3 공통 코드.c | 20232907 정... 문제 3 코드.c
+ 과제 2(C) (전역 범위)
7 void main() {
8     int a, b;
9     float c, d;
10    my_scanf("%d%f", &a, &c);
11    my_scanf("% f % d", &d, &b);
12    printf("%d\\n%d\\n%f\\n%f", a, b, c, d);
13 }
14
15 int my_scanf(const char* const args, ...) {
16
17
18
19 1 -2.3
20 +4.5 -6
21 1
22 -6
23 -2.300000
24 4.500000
25
26 D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제
  
```

그림 9.1

위 그림 9.1은 int 형과 float 형, 부호 값, args에서의 띄어쓰기까지 모든 것을 테스트한 결과이다. 위 그림 9.1과 같이 모든 상황에서 정상 동작하는 것을 알 수 있다.

10. 소스코드

첨부파일 참조

문제별 코드 실행 시 다음 파일이 모두 필요합니다.

문제 1 코드 실행 시: 프로그래밍 과제 3 공통 코드, 프로그래밍 과제 3 공통 헤더, 문제 1 코드 필요

문제 2 코드 실행 시: 문제 2 코드 필요(다른 코드 필요 없음)

문제 3 코드 실행 시: 프로그래밍 과제 3 공통 코드, 프로그래밍 과제 3 공통 헤더, 문제 3 코드 필요