

과제 3

단어 정렬

타자 게임

80자리 계산기

계산식

20232907 정현승

1. 문제 1

사용자가 end를 입력할 때까지 계속 문장을 입력받는다. 이후 화면에 입력된 문장 수, 단어 수, 중복을 제외한 단어의 수를 표시하고, 중복을 제외하고 입력된 단어를 정렬해서 출력한다.

2. 문제 1 해결 방안

우선 문장의 경우, 문장을 나누는 기준이 개행('\n')이 입력된다는 것 하나밖에 없으며, 글자 수 제한이 존재하지 않는다. 따라서 문장의 길이를 단정해서 문자열을 설정하기가 어렵다. 문자의 길이를 100으로 설정했는데 한 문장에 105자가 들어올 수도 있고, 200으로 설정했는데 210자가 들어올 수도 있고, 극단적으로는 100000으로 설정했는데 200000자가 들어올 수도 있는 것이다. 이는 각 단어의 위치를 저장하는 포인터 배열도 마찬가지이다. 따라서 배열의 길이를 프로그램 실행 도중에 바꿀 수 있는 동적할당을 사용하였다. 동적할당의 경우 할당이 거부당하는 경우가 존재할 수 있으나 이 문제보다는 문자열의 길이가 정해져 있다는 것이 더 문제라고 생각해 이처럼 구현하였다. 다만 할당이 거부당할 경우를 대비해서, 할당이 거부되면 10번 정도는 다시 시도해 보고, 그래도 안 된다면 예외 처리를 하여 프로그램을 끝내도록 했다. 할당이 계속 거부되어 프로그램을 종료할 때는 에러 메시지도 띄우도록 했다. 그리고 한 글자 입력받고 재할당을 받기에는 재할당을 너무 많이 받게 되어 프로그램이 느려질 수 있으므로, 기존 문자열의 공간이 부족해 재할당할 때 기존 공간에 201자의 공간만큼, 포인터 배열의 공간이 부족할 때는 10개의 포인터 변수를 저장할 수 있는 공간만큼 한꺼번에 추가하도록 했다. 또한 어떤 경우에서든 프로그램을 종료할 때는 할당받은 모든 공간을 free 하여 memory leak이 발생하지 않도록 했다.

또한 문장 구분은 개행 문자로 진행하였다. 앞서 설명했던 문장으로 몇 글자가 들어오지는 모르는 상태이다. 입력을 fgets 함수로 받기는 했지만, 이는 지정된 문자 수를 초과해 입력하면 문자열을 자른다. 그 이후의 문자들은 fgets를 다음에 호출할 때 입력을 받게 된다. fgets 실행할 때를 기준으로 문장을 자르면, 긴 문장을 입력하는 경우 문장 입력이 중간에 끊겨 같은 문장, 심지어는 하나의 단어임에도 입력 문자 수 제한으로 인해 다른 문장으로 판정되는 사례가 생길 수도 있다. 따라서 문장 숫자는 입력된 개행 문자의 수로 설정했다. 또한 문장 저장은 문장별로 나눠서 저장하지 않고 모든 문장을 한 개의 문자열에 저장하였다. 문장 입력의 경우 동적할당 공간을 저장하는 변수와 별개로 현재 입력 위치(커서)를 나타내는 포인터 변수를 선언하여, 입력 시마다 입력된 문장의 끝 지점 '\0'으로 이동시켰다. fgets의 매개변수로 이 커서 포인터와 커서로부터 남아 있는 할당받은 문자열 길이를 사용하여 입력받았다(입력 시 매개변수로 전체 문장의 시작점을 사용하면 기존 내용이 지워지므로, 전체 문장의 끝 지점부터 입력받았다. 또 할당받은 영역을 넘어가서는 안 되므로 현재 커서와 할당받은 영역의 끝 사이의 거리, 즉 이번에 최대를 받을 수 있는 문자열의 길이도 계산해 매개변수로 사용했다.) fgets 함수가 종료될 때마다 이번에 입력된 문자열이 입력 종료 조건에 해당하는 end인지 확인하고, 마지막으로 입력된 문자가 개행 문자인지(즉

문장 하나가 끝났는지), 또 할당된 공간 전부 사용까지 5개 이내의 문자가 들어갈 공간만 남아있는지 확인했으며, 각각에 해당하면 순서대로 입력 반복문을 탈출하고, 문장 개수를 세고, 공간을 재할당받았다. 공간을 재할당받을 때는 할당받은 공간의 주소가 변경될 수 있다는 점을 고려해, 커서도 변경된 주소에서 다음 입력을 받을 위치(새로운 공간 주소 + 지금까지 입력된 전체 문자열 길이 + 1)로 옮겼다. 꼭 다 채워지지 않아도 얼마 안 남았을 때, 할당된 공간 전부 사용까지 5개 이내의 문자가 들어갈 공간만 남은 상태에서 재할당받는 이유는, 다른 단어와 같이 한 문장으로 입력된 end는 종료 조건으로 보지 않는데, 이를 확인하는 기준이 저번 fgets 입력에서 문장 마지막이 개행 문자로 끝났는가이기 때문이다. ("end\n"에서 마지막 '\n'까지 일치해야 한다. "end"로 끝나면 입력이 안 끝났으므로 종료 조건으로 보지 않는다.) 이 상태에서 재할당받지 않고 종료 조건 end를 입력하면, 중간에서 입력이 잘려 종료 조건으로 인식되지 않는 문제가 있기도 하고, (예를 들어 e와 nd로 잘린 경우, 둘 다 종료 조건 end에 해당하지 않기에 종료 조건으로 인식하지 못한다.) 어차피 남은 공간으로 종료 조건 입력이 불가능해 더 많은 문자가 입력되는 것이 확정되므로 미리 재할당을 받는 것이다.


입력을 모두 받으면 문장들을 단어 단위로 나누고 그 수를 셸다. 문장을 단어로 나눌 때 더 이상 입력받은 문장들이 문장 형태로 더 이상 존재할 필요가 없으므로(이후로 입력받은 문장을 문장 형태로 활용하지 않는다) 이 문자열을 별도의 복사 없이 strtok함수를 이용해 공백이나 개행 문자 단위로 잘라 단어 단위로 분리하였다. (자르는 기준에는 이 둘 외에도 몇 가지를 더 넣어 놓았지만, 디버그 과정에서만 쓰일 뿐 문제 조건상 쓰지 않는다.) 우선 정렬이나 중복을 생각하지 않고 단어를 저장했으며, while 조건문을 활용하기 위해 strtok 함수를 반복문 마지막에, 임시 포인터 변수에 저장했고 이 포인터 변수가 NULL이 되었을 때 반복문을 빠져나왔다. 또 단어 수를 세면서 단어 각각의 위치를 저장할 공간이 부족하면, 10개의 포인터를 저장할 만큼의 공간을 재할당받았다.

단어들의 분리까지 끝났으면 qsort 함수를 이용해 문자열을 정렬했다. 이때 배열이 문자열의 배열이 아닌 문자열 포인터의 배열이므로, 각 문자열의 순서 비교를 하는 함수로 strcmp를 바로 사용할 수 없었다. (각각 '*' 1개씩으로 문자열에 대한 접근이 필요했다.) 이에 따라 strcmp 함수를 사용할 수 있도록 약간의 보정을 한 3줄짜리 함수를 만들어 이를 이용했다. (strcmp를 호출하되 매개변수로 *str1, *str2를, 즉 '*' 1개씩 붙여줘서 사용한다는 게 함수의 끝이다.) 정렬이 완료되면, 중복된 문자열의 경우 인접해 있다는 점을 이용해, 모든 단어를 뒤 단어와 비교해 중복된 단어가 있으면 그 포인터를 지우고, 그 뒤에 있는 포인터들은 중복된 단어의 수만큼 앞으로 당겨 저장했다. (정렬을 먼저 했으므로 모든 단어를 대상으로 단어가 일치하는지 볼 필요가 없다.) 이때 중복된 단어의 수를 셀 때는 임시 변수를 활용했다. 마지막으로 이렇게 저장된 문장 수, 단어 수, 중복을 제외한 단어 수를 출력하고, 단어의 위치를 저장한 포인터의 배열을 순서대로 출력하는 방향으로 문제를 해결했다. 출력을 끝낸 뒤 할당받은 문자열을 할당 해제하는 과정도 진행했다.

이를 적용하여 문제 1을 해결한 결과는 다음과 같다.

3. 문제 1의 결과

실행을 위해서 공통 코드, 공통 헤더, 문제 1 코드가 필요하다.



```

once when i was six years old i
saw a magnificent picture
end
2 12 11
a
i
magnificent
old
once
picture
saw
six
was
when
years

```

D:\OneDrive - 중앙대학교\1학년 2학기\프로그래밍\실습 문제\과제 3\64\Debug\과제 3.exe(프로세스 11096개)이(가) 종료되었습니다.(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

그림 1

```
Microsoft Visual Studio 디버그 콘솔
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
endend end
end
2 3 3
end
endend
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
D:\OneDrive - 중앙대학교\1학년 2학기\프로그래밍\실습 문제\과제 3\x64\Debug\과제 3.exe(프로세스 19760개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

그림 2

end가 단독으로 입력된 게 아니면 종료 조건에 해당하지 않고, 5개 문자 공간 이내로 공간이 남았을 때 end 입력 시 조건이 종료되는 것을 확인할 수 있다.

4. 문제 2

랜덤하게 생성된 5개의 문장을, 사용자가 입력하는 시간을 이용해 분당 입력 수를 구한다. 문장은 모두 공백 제외 40~50자이며, 하나의 단어는 영어 대문자만 4~10자이고, 영어 대문자와 공백만으로 구성된다. 입력 속도에 따른 점수를 얻으며, 속도를 구할 때 오타(공백 제외)가 있다면 이를 따로 표시하고 속도 계산에서 제외해야 한다.

5. 문제 2 해결 방안

문장의 경우 실행 예시를 보면 40~50자의 기준이 공백 제외인 점을 확인할 수 있다. 그리고 단어의 길이도 4~10자이고 꼭 뜻이 있는 단어를 완성할 필요는 없다는 점도 확인할 수 있다. 따라서 문장을 만드는 것은 단어의 길이를 정하고 단어의 길이만큼 무작위의 영어 대문자를 넣은 뒤, 단어 사이를 공백으로 구분하고 이를 (현재까지의 공백 제외 문장 길이 + 이번 단어의 길이)가 50을 넘게 될 때까지 반복하고, 마지막으로 아직 공백 문자로 남아있는 문장의 마지막 글자를 'W'로 바꿔주면 문장은 완성이다.

사용자 입력 시작 직전과 직후에 프로그램 실행 시간을 각각 저장하고, 틀린 부분을 출력한 뒤 틀린 부분을 제외하고 입력 속도를 구해 화면에 표시하고 점수에 합산한다. 이때 입력 속도는 float 형으로 저장해서 소수점 이하의 계산이 가능하게 하였다. 다만 이 때문에 화면에 표시된 속도의 합과 전체 점수가 다를 수는 있으나 이는 오히려 정확한 계산을 위해 소수점 이하를 살린 이상 불가피하다고 판단하였다. 입력은 문제 1에 썼던, 입력 길이에 제한이 없는 동적 할당 방법을 사용하였으며, 문제 1의 코드를 가져와 함수로 빼 사용했다.

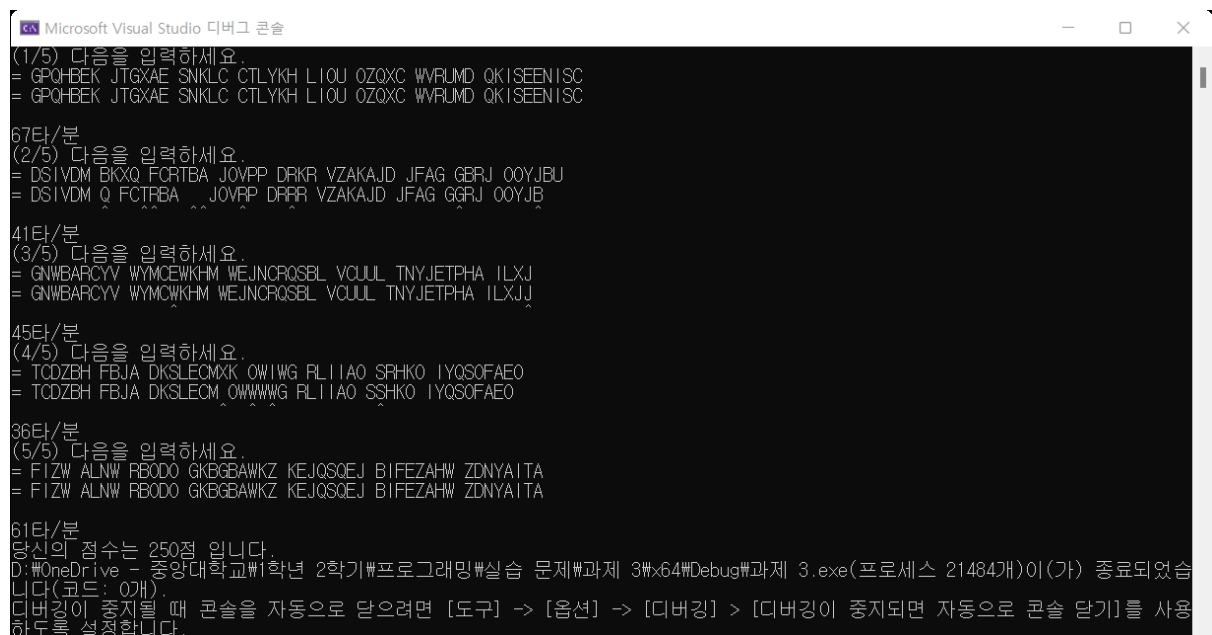
틀린 부분을 확인하는 함수는, 5자 이내에서 글자가 밀렸으면 이를 찾아 밀렸다는 체크를 하고 (이때 밀렸다는 것을 확인하는 방법은 확인하려는 위치부터 글자 3개를 이용하여, 저장된 문장과 입력받은 문장 중 하나는 확인하려는 위치로 고정하고 다른 하나는 그 지점으로부터 하나씩 최대 5번까지 밀면서 글자 3개가 같은지를 확인한다. 한 글자가 아닌 3글자로 보는 이유는, 같은 글자가 연속되어서 단순 오타인데도 글자가 밀린 것으로 확인되는 것을 막기 위함이다. 그렇다고 확인하는 글자 수를 계속 늘리면, 이후에 있는 단순 오타 때문에 밀렸다는 것을 잘 확인하지 못할 수 있다. 따라서 그냥 하나 잘 못 쓴 건데 그게 우연이 밀렸다고 체크될 수도, 확률은 매우 적지만 있기는 하며 이 경우 2번 밀린 것으로(한 번은 잘못 확인된 방향으로, 다른 한 번은 잘못된 방향 반대 방향으로) 확인될 수 있다. 또 글자 3개 중 오타가 하나 더 있으면 밀렸다고 체크되지 않고 모두 오타로 보는 것이다.) 그렇지 않으면 이 글자 하나만 틀린 것으로 처리하도록 하는 방향으로 문제를 해결했다. 글자가 밀린 경우, 저장된 문장과 입력받은 문장의 커서를 따로따로 두어 한쪽만 커서를 움직이게 하는 것으로 다음 비교를 진행해 나갔으며, 오타 발생 위치 저장은 입력받은 문장을 지우며 그 문장 자리에 저장했다. 점수 계산을 위한 오타 제외 글자 수 측정은, 단순 1개 글자 오타나 저장된 문장에 대해 입력된 문장이 뒤로 밀린 경우 또는 저장된 문장이 입력된

문장보다 먼저 끝날 때(저장된 문장을 입력하고 뒤에 추가로 다른 글자들을 집어넣은 경우)는 그 수만큼, 입력된 문장에 대해 저장된 문장이 뒤로 밀린 경우 1번 밀릴 때마다 1개씩, 입력된 글자 수에서 뺐으며, 입력된 문장이 저장된 문장보다 먼저 끝난 경우 뒤에 입력되지 않은 부분은 이미 입력된 글자 수에서 제외되었으므로 추가 계산하지 않았다. 입력된 문장에 대해 저장된 문장이 뒤로 밀린 경우 중간에 빠진 문자를 제외하고도 1개씩을 더 뺐는데, 이렇지 않으면 문제 이름이 "게임"이고 오타를 빼고 계산하는 것도 오타에 대한 불이익을 주기 위함인데, 이렇게 하지 않을 경우 저장된 문장에서 같은 문자가 2개 문자 이내로 반복될 경우, 그 문자 하나만 입력해 높은 점수를 가져가게 되기 때문이다. (간단하게 예를 들면 CSCYICR CTOC의 경우 C만 5번 누르면 5타 전부 인정되면서 빠른 속도로 분당 타수를 높일 수 있다. 이를 방지하기 위함이다.) 이 틀린 부분 확인은, 사람의 관점에 따라 바뀔 수 있으며, 이보다 더 좋은 알고리즘이 존재할 수 있다. 그러나 그렇다고 여기서 AI를 사용할 수는 없으므로, 여기까지만 구현하였다.

이를 적용하여 문제 2를 해결한 결과는 다음과 같다.

6. 문제 2의 결과

실행을 위해서 공통 코드, 공통 헤더, 문제 2 코드가 필요하다.



```

Microsoft Visual Studio 디버깅 콘솔
(1/5) 다음을 입력하세요.
= GPOHBEK JTGXAE SNKLC CTLYKH LIOU OZQXC WVRUMD QKISEENISC
= GPOHBEK JTGXAE SNKLC CTLYKH LIOU OZQXC WVRUMD QKISEENISC

67타/분
(2/5) 다음을 입력하세요.
= DSIVDM BKXQ FORTBA JOVPP DRKR VZAKAJD JFAG GBRJ OYJBU
= DSIVDM Q FCTRBA JOVPP DRKR VZAKAJD JFAG GBRJ OYJBU

41타/분
(3/5) 다음을 입력하세요.
= GNBARCYV WYMCWKHM WEJNCROSEL VOUL TNYJETPHA ILXJ
= GNBARCYV WYMCWKHM WEJNCROSEL VOUL TNYJETPHA ILXJ

45타/분
(4/5) 다음을 입력하세요.
= TCDZBH FBJA DKSLECMXK OWIWG FLIIAO SPHKO IYQSOF AEO
= TCDZBH FBJA DKSLECMXK OWIWG FLIIAO SPHKO IYQSOF AEO

36타/분
(5/5) 다음을 입력하세요.
= FIZW ALNW RBODG KKBGBAWKZ KEJQSQJEJ BIFEZAHW ZDNYAITA
= FIZW ALNW RBODG KKBGBAWKZ KEJQSQJEJ BIFEZAHW ZDNYAITA

61타/분
당신의 점수는 250점 입니다.
D:\OneDrive - 중앙대학교\1학년 2학기\프로그래밍\실습 문제\과제 3\64\Debug\과제 3.exe(프로세스 21484개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
  
```

그림 3

입력의 오타를 빼고 계산하는 과정도 Break point를 이용해 확인하긴 했지만, 이를 직접적으로 결과 화면에 나타내는 것은 문제 요구사항이 아니기 때문에 이를 보고서에 넣지 않았다.

7. 문제 3

long 형이나 long long 형으로도 처리할 수 없는 80자리 양의 정수 2개를 입력받고 각각의 덧셈과 뺄셈을 시행해 화면에 출력한다.

8. 문제 3 해결 방안

이 문제는 메모리 낭비를 고려하지 않는다면 매우 쉽게 만들 수 있다. 숫자를 한 자리씩 끊어 char의 배열로 숫자를 저장할 수도 있고, 9자리씩 끊어 int의 배열로 저장한 뒤 연산을 진행할 수 있다. 그러나 이렇게 되면 메모리 낭비가 너무 심해지게 된다. int의 배열로 저장하는 경우 1개의 int 공간에 0~999,999,999까지의 숫자가 저장되게 되는데, 1,999,999,999까지는 연산할 때 사용할 수 있겠지만 음수 부분이나 2,000,000,000 이후부터 int 형으로 저장할 수 있는 최대 숫자까지는 버려지는 메모리 공간이 된다. 이 하나로는 (단순 저장 시) 2비트 조금 넘는 크기가 낭비되는 수준에 그치지만 이것이 쌓이다 보면 메모리 낭비를 무시하기 힘들다. 따라서 unsigned int가 저장할 수 있는 최대 숫자인 4,294,967,295를 전부 사용할 수 있는 방식을 사용했다. 다만 이 경우 배열의 크기 처리와 숫자의 부호 처리가 힘들 것 같아 구조체를 사용하였다. 이 구조체는 만들 때 사용에 필요하다고 예상되는 만큼 동적 할당을 받아와 그 할당받은 공간에 숫자를 저장하고, 목적이 끝나면 할당을 해제하는 방식으로 활용하였다. 여기서 할당의 경우는 더할 때나 뺄 때 매번 다시 할당하였고, 이 이외 문자열을 숫자로 바꾸거나, 숫자를 곱하거나 하는 등의 경우에는 덧셈과 뺄셈을 이용해 연산하였기 때문에, 덧셈과 뺄셈 시 기존에 할당받은 공간은 해제하고 연산 결과를 저장한, 새로 할당받은 공간을 사용하는 방식으로 진행하였다. 숫자 저장 방식은 Visual Studio가 int 형 자료를 저장하는 방식처럼, 0번째 index에 가장 작은(낮은) 자리의 숫자가 저장되고, index가 증가할수록 점점 더 커지는 것이다. (꼭 Visual Studio만 이렇게 저장하는 것은 아니지만, 이런 방식을 사용하지 않는 경우도 존재하기 때문에 이를 예시로 들었다.) 할당받은 크기는 unsigned int를 사용한 만큼 4바이트 단위로 할당을 받았으며, 더하기의 경우 더하다 저장공간이 넘어가는 사례가 발생할 수 있기에 4바이트 더 여유롭게 할당했다. 4바이트를 할당받아 놓고 1비트만 사용하면 그것도 메모리 낭비이긴 하지만 앞에 설명한 방식처럼 낭비하는 메모리가 큰 것도 아니고, 또 그렇다고 자료형을 short나 char 기반으로 해 버리면 연산 시간이 길어지는 문제가 발생하여 해당하는 상황에 대한 별도의 처리는 진행하지 않았다. 이렇게 선언한 구조체의 이름이 lint이므로 이후 해당 방식으로 저장된 숫자를 lint로 표현하겠다.

먼저 문제 2에 사용했던 문자열 입력 받는 함수를 그대로 가져와 정수를 문자열로 입력받고, 이를 lint로 바꾸는 작업을 진행하였다. 작업 진행 방법은, atoi함수의 return 값이 int이고 lint가 unsigned int 기반임을 이용해 입력받은 문자열에서 9자리씩 끊어 숫자로 바꿨으며, 아직 그 뒤에 숫자가 남아 있다면(즉 이 9자리가 숫자의 끝이 아닌 경우) 기존 저장된 숫자를 1,000,000,000(10의 9제곱)만큼 곱하고 여기에 9자리씩 끊어 숫자로 바꾼 수를 더했다. (즉, 문제에서 80자리 수의

곱셈을 요구하지는 않았지만, 80자리의 곱셈을 구현해야 했다. 실제로 코드에는 80자리 수와 80자리 수를 곱하는 기능의 함수가 구현되어 있으며, 여기서도 이를 이용했다.) 다만 뒤에 남은 숫자의 수가 9를 넘지 못한다면 8자리 이하의 수만 남아있다는 말이므로, 10에 (남아있는 자릿수)를 제공한 수를 기존 저장된 수에 곱한 다음 남은 숫자만큼을 더했다.

여기서 사용되는 덧셈 연산 과정은 다음과 같다. 먼저 두 숫자의 부호가 다른 경우, 빼기 연산이므로 부호를 변경하여 빼기 연산을 진행하도록 한다. 이 확인이 끝나면, 먼저 연산 결과를 저장할 lint 변수에 공간을 할당한다. 할당할 공간은 더할 두 lint 숫자에 할당된 크기 중 큰 것에, 받아올림으로 인해 그 크기를 넘는 숫자가 저장될 것을 대비해 unsigned int 크기만큼(4바이트)의 공간을 더한 크기로 했다. 연산 과정은 둘 다 작은 자리에 저장된 숫자(index 0에 있는 숫자)부터 unsigned long long(8바이트)으로 선언된 숫자에 더하고, 그 결과를 unsigned int 형으로 저장한다. 자료형이 unsigned long long인 이유는 unsigned int로 저장할 수 없는 연산 결과를 저장하기 위함이다. 5바이트 정도만 필요하긴 하나 5~7바이트만 사용하는 정수형이 없어 8바이트를 사용하는 unsigned long long으로 선언하게 되었다. 이때 이 덧셈 연산 결과가 저장된 변수는 초기화하지 않고 shift 연산을 진행하는데, 이는 받아올림이 발생하면 다음 계산 시에 이를 반영해 주기 위함이다. 받아올림이 발생하면 그 수는 결과 저장 시 unsigned int 형으로 바꾸는 과정에서는 지워져 결과가 저장되지 않을 것이다. 이를 32비트(4바이트)만큼 오른쪽으로 밀면(>>연산), 결과로 저장된 숫자들은 전부 지워지고 저장되지 않은 숫자가 일의 자리부터 채우게 된다. 이를 다음 연산에 활용하면 이 받아올림 값을 저장할 수 있다. (따라서 덧셈 연산 결과를 저장하는 변수의 초기화는 연산 시작 전 딱 한 번만 진행한다.) 연산이 모두 끝나면 결과값을 저장한 변수가 동적할당 받은 공간을 다 썼는지 확인하고, unsigned int 크기만큼(4바이트) 공간의 여유분을 사용하지 않은 경우 구조체에 저장된 동적할당 받은 크기 값만 변경하였다. 원래는 할당까지 다시 받는 게 맞으나, 어차피 이 숫자로 많은 활용을 하지 않을 것이므로, 그 메모리 낭비를 잡으려고 재할당을 받는 행위는 하지 않았다. 저장된 크기 값만이라도 수정하는 이유는, 모든 동적 할당 시 할당받는 크기는 다른 lint 변수에 할당된 크기로 정해진다. 이 크기 값을 처리하지 않으면, 처음에는 낭비되는 메모리의 양이 상관없는 수준일지 몰라도 나중에는 그 크기가 매우 커지게 된다. 따라서 저장된 크기 값만이라도 수정하는 것이다.

$$\begin{array}{r}
 1 1 \\
 \times 0 1 \\
 \hline
 1 1 \\
 1 1 \\
 \hline
 0 0 \\
 0 1
 \end{array}$$

그림 4

곱셈의 경우 논리회로 시간에 배운 binary의 곱셈을 활용했다. 2번째 숫자를 비트 단위로 검사한 뒤, 특정 비트에 1이 저장되어 있다면, 기존 결과와 1번째 숫자를 그 1이 저장된 위치까지 왼쪽으로 shift 한 숫자를 더하며, 모든 비트를 확인할 때까지 이를 반복한다. 이해를 돕기 위해 왼쪽에 논리회로 강의자료를 그대로 가져왔다.ⁱ (인터넷에서 찾은 것은 아니다.) 이 곱셈 연산을 위해 덧셈 연산과 shift 연산(함수 이름은 push로 되어 있다.), 특정 비트값을 뽑아낼 수 있는 MUX(이 또한 논리회로 수업 시간에 학습한 내용이다.) 기능을 하는 함수를 전부 만들어 이용했다. 결과값의 부호는 두 수를 XOR 연산을 한 값으로 설정했다.

뺄셈의 경우 먼저 부호 차이로 인해 실제로는 덧셈을 진행해야 하는 경우, 부호를 바꿔서 덧셈

을 진행했고, 음수에서 음수를 빼는 경우 순서와 부호를 바꿔 양수에서 양수를 빼는 것으로, 작은 수에서 큰 수를 빼야 하는 경우 숫자의 순서를 바꿔서 뺄셈을 진행한 후 부호만을 변경했다. 여기서 부호나 순서를 바꿔 뺄셈을 진행해야 하는 경우, 기본적인 처리를 하고 본 연산으로 넘어가는 게 아니라 처리를 한 뒤 뺄셈 기능을 하는 함수를 다시 호출해 확인부터 다시 진행했다. (재귀 함수) 결과값을 할당받는 크기는 우선 두 수 중 큰 1번째 숫자의 크기만큼 할당받았다. 이후 덧셈 처럼 0번째 index부터 하나하나씩, 1번째 수에서는 더하고 2번째 수에서는 빼는데, 이번에는 그 연산 결과를 signed long long으로 선언된 숫자에 저장한다. 이는 비록 큰 수에서 작은 수를 빼는 과정이지만(반대의 과정이 들어오면 앞에서 검사를 통해 숫자의 순서를 바꿔 연산을 진행하므로) unsigned int 단위로 끊은 특성상 지금 당장은 저장된 연산 결과값이 음수가 될 수 있다. 연산이 계속 진행되면 뒤쪽(자릿수가 높은 부분)에서 자연스럽게 해결되겠지만 지금 당장은 음수가 저장되는 것이다. 이 경우 연산 결과로 잘못된 값이 저장될 수 있으니, 이 경우 다음 index에서 1만 가져와 연산 결과가 저장된 변수에 더해준다. 다음 index에서는 1이겠지만, 이게 이전 index에서 계산되므로 더하는 값은 $4,294,967,296(1 < 32)$ 이 된다. 이렇게 한 상태에서 결과를 저장하면 연산 결과가 양수가 되면서 숫자가 정상적으로 들어간다. 저장하고, shift 연산을 수행한 후 방금 다음 index에서 1을 가져온 경우, 이제 1을 빼준다. shift 연산을 이미 진행했기 때문에 1만 빼 주면 된다. 이를 모든 index를 확인할 때까지 반복하면 된다. 이후 덧셈 때처럼 할당된 크기 값을 알맞게 변경해 주면 뺄셈이 끝난다.

마지막은 이 숫자를 다시 십진수의 문자열로 바꿔주는 과정이다. 이 과정에서 인터넷을 참고했으며 참고한 코드를 토대로 재구성한 것이다. 먼저 결과를 저장할 문자열을 동적할당 받되, 쓰기는 이 할당받은 문자열의 마지막 직전 위치부터, 뒤에서부터 작성하였다. 또 저장된 수를 직접 건드려야 하니 동적 할당을 다시 받아 아예 lint 변수를 복사하고 시작하였다. 이번에는 나누기와 나머지 연산이 필요하다. 이는 lint와 정수 간의 연산이며, 별도의 함수로 빼지는 않았다. 여기서 활용되는 변수 left는 unsigned long long으로 선언된 숫자이다. 먼저 가장 높은 자릿수에 저장된 수를 left 변수에 저장한다. 이후 그 가장 높은 자릿수에 저장된 수를 left에 10을 나눈 수로 덮어씌운다. Left 변수는 다시 현재 저장하고 있는 값을 10으로 나눈 나머지를 32비트만큼 왼쪽으로 shift하고(즉 $4,294,967,296(1 < 32)$ 을 곱하는 것) 여기에 다음으로 높은 자릿수에 저장된 수를 더한다. 이후 다음으로 높은 자릿수에 저장된 수를 left에 10을 나눈 수로 덮어씌우고, 이를 마지막 자릿수까지 반복한다. 마지막 자릿수에 저장된 수를 left에 10을 나눈 수로 덮어씌운 후, left에 10으로 나눈 나머지를 저장한다. 이때 이 left가 저장하고 있는 수가 lint 숫자를 십진법으로 바꿨을 때 일의 자리에 들어가는 숫자이다. 이를 문자로 바꿔 저장해 준다. 커서를 한 칸 앞으로 옮기고, lint에 저장된 숫자가 0이 될 때까지 이를 반복한다.ⁱⁱ 중간에 문자열의 길이가 부족하면 10개의 문자가 들어갈 공간만큼을 다시 할당받고, 기존 문자열을 뒤에서부터 붙여 넣고 커서도 옮겨 재개한다. (문자를 뒤에서부터 쓰는 특성상 realloc를 사용하기 어려워 기존 문자열은 할당을 해제하고 새로운 문자열을 할당받는 것이다.) 이후 부호까지 붙여주고, 이렇게 작성된 문자열을 할당받은 위치 처음으로 옮겨주면(문자 하나하나 밀면) 완성이다. (할당받은 위치 처음으로 옮겨주는 이유는 그렇지 않으면 받는 쪽에서는 아무것도 없는 문자열로 인식하게 될 것이고, 문자열의 시작 위치

이 문제는 이렇게 여러 함수로 여러 기능을 만들어 해결하였다. 문제 요구사항은 아니었던 곱셈과 나눗셈, 나머지 연산도 필요해서 만들게 되었다.

[illegible]

왼쪽 그림은 python을 이용해 무작위 80자리 수를 뽑고, 각각의 덧셈과 뺄셈을 진행한 것이다. 다음 페이지에 있는 그림 6은, 여기서 나온 무작위 값을 프로그램에 넣었을 때의 출력 결과값이다. 단 마지막 결과값 출력 이후 다시 x 를 입력받는 부분은, 다른 입력값 및 출력값이 잘 보이도록 그림을 확대하는 과정에서 잘랐다. 그림 5나 그림 6 모두 잘 보이지 않을 것이고, 설명 잘 보이더라도 80자리 전체를 비교하기 힘들 것으로 보여 아래 페이지에 그 값을 전부 적어 놓았다. 단 한 자리 수 연산이나 문제 정의(과제 pdf 파일)에 있는 예시의 경우, 그림 5에는 없지만 그림 6 실행 결과값은 있으며, 한 자리 수의 경우에는 본인이 알아서 계산했고 문제 정의에 있는 예시는 pdf 파일에 있는 정답을 그대로 가져왔다. 마지막 예시의 경우는 오류 확인을 위해 임의로 넣은 숫자이므로, 연산 결과만 가져오고 숫자를 무작위로 뽑는 연산은 시행하지 않았다.

← 그림 5

그림 6

다음은 python과 프로그램 로그 순서대로 나열한 X(a), Y(b), 합, 차 값이다. 숫자가 크지 않으면 표로 정리하는 게 좋으나, 숫자가 너무 커서 표 없이 순서대로 나열하여 비교할 값이 이웃한 줄에 나열되도록 하는 게 더 보기 편하리라 생각해 그렇게 했다.

간단한 예시

X(a): 1

Y(b): 2

합(정답): 3

합(프로그램 출력값): 3

차(정답): -1

차(프로그램 출력값): -1

문제 정의 파일에 나와 있는 예시

X(a): 3289374289374289374238947

Y(b): 198428971238913470123978

합(정답): 3487803260613202844362925

합(프로그램 출력값): 3487803260613202844362925

차(정답): 3090945318135375904114969

차(프로그램 출력값): 3090945318135375904114969

양수 2개 80자리 예시 1

X 58303480404693901484285191211070534320824647503239463852166054921723149845783408

Y 3449304571788085325984455749237963813613729773498437455994706081750803051373482

합(python 계산값):

61752784976481986810269646960308498134438377276737901308160761003473952897156890

합(프로그램 출력값):

61752784976481986810269646960308498134438377276737901308160761003473952897156890

차(python 계산값):

54854175832905816158300735461832570507210917729741026396171348839972346794409926

차(프로그램 출력값):

54854175832905816158300735461832570507210917729741026396171348839972346794409926

양수 2개 80자리 예시 2

X 71647630378139893960654884817786831616802567437582627924870115561656063507026771

Y 863130564555085989781872207755885331469066304969020794578993052486453569127427

합(python 계산값):

80278936023694979950436757025542716948271633742551648719449108614142517076154198

합(프로그램 출력값):

80278936023694979950436757025542716948271633742551648719449108614142517076154198

차(python 계산값):

63016324732584807970873012610030946285333501132613607130291122509169609937899344

차(프로그램 출력값):

63016324732584807970873012610030946285333501132613607130291122509169609937899344

양수 2개 80자리 예시 3 (차가 음수인 경우)

X 13685997586028118481889239004205241180825616667053556219089041810302013856313135

Y 86323060054499528196402077773879813501193812913241651190142986608379327253144185

합(python 계산값):

100009057640527646678291316778085054682019429580295207409232028418681341109457320

합(프로그램 출력값):

100009057640527646678291316778085054682019429580295207409232028418681341109457320

차(python 계산값):

-72637062468471409714512838769674572320368196246188094971053944798077313396831050

차(프로그램 출력값):

-72637062468471409714512838769674572320368196246188094971053944798077313396831050

X가 양수 Y가 음수인 80자리 예시

X 97085224051971419646877056780534219315349283758423814207366834999056951930686239

Y-65734160356191329275433320738345301473743924451421713930811258049306977005923854

합(python 계산값):

31351063695780090371443736042188917841605359307002100276555576949749974924762385

합(프로그램 출력값):

31351063695780090371443736042188917841605359307002100276555576949749974924762385

차(python 계산값):

162819384408162748922310377518879520789093208209845528138178093048363928936610093

차(프로그램 출력값):

162819384408162748922310377518879520789093208209845528138178093048363928936610093

음수 2개 80자리 예시

X -8928799500443970076719821350055775415494207513175084705069905602630573075840359

Y-32572480313277280601676809291374427283559311034011845320000948536532192748242808

합(python 계산값):

-41501279813721250678396630641430202699053518547186930025070854139162765824083167

합(프로그램 출력값):

-41501279813721250678396630641430202699053518547186930025070854139162765824083167

차(python 계산값):

23643680812833310524956987941318651868065103520836760614931042933901619672402449

차(프로그램 출력값):

23643680812833310524956987941318651868065103520836760614931042933901619672402449

X가 음수 Y가 양수인 80자리 예시

X-40305157475456624490279689200545730363509071748261101011749980049328316034550632

Y 34432277635105089993031876326292882072227184942347133619966200837313153819940000

합(python 계산값):

-5872879840351534497247812874252848291281886805913967391783779212015162214610632

합(프로그램 출력값):

-5872879840351534497247812874252848291281886805913967391783779212015162214610632

차(python 계산값):

-74737435110561714483311565526838612435736256690608234631716180886641469854490632

차(프로그램 출력값):

-74737435110561714483311565526838612435736256690608234631716180886641469854490632

같은 양수 2개 80자리 예시

X 51016858116988078473995926511428763703256848032448953620828484139265633324034660

Y 51016858116988078473995926511428763703256848032448953620828484139265633324034660

합(python 계산값):

102033716233976156947991853022857527406513696064897907241656968278531266648069320

합(프로그램 출력값):

102033716233976156947991853022857527406513696064897907241656968278531266648069320

차(python 계산값): 0

차(프로그램 출력값): 0

같은 음수 2개 80자리 예시

X-26833194075829079387631625599501077054765089239500649680472359631524410513921250

Y-26833194075829079387631625599501077054765089239500649680472359631524410513921250

합(python 계산값):

-53666388151658158775263251199002154109530178479001299360944719263048821027842500

합(프로그램 출력값):

-53666388151658158775263251199002154109530178479001299360944719263048821027842500

차(python 계산값): 0

차(프로그램 출력값): 0

앞 50자리가 같은 80자리 예시

X 6206557895151192286498344191958603674381403409836227215426428842077236948382145

Y 6206557895151192286498344191958603674381403409836864987916775811995529865846785

합(python 계산값):

10. 문제 4

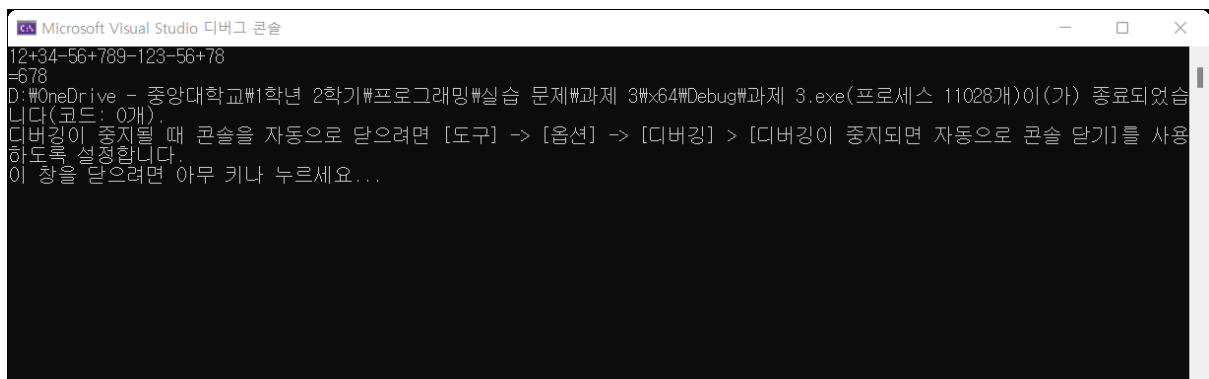
+, - 가 여러 개 포함된 계산식을 입력받아 계산 결과를 출력하는 프로그램을 작성한다.

11. 문제 4 해결 방안

우선 계산식을 문자열로 입력받는다. 입력받는 방법은 문제 2에서 만든 함수를 그대로 가져왔다. 여기서 strtok를 사용해서 숫자를 분리하는 경우, 숫자 사이의 연산 기호(+와 -)가 사라지기 때문에 입력받은 문자열을 문자 하나하나 검사했다. 현재 읽고 있는 문자의 위치를 나타낼 포인터와 마지막 연산자 직후의 숫자(즉 문자를 숫자로 바꿀 때 바꾸기를 시작하는 시점)를 저장할 포인터 변수를 이용하여, 문자를 앞에서부터 하나씩 확인했다. 그러다가 숫자가 아닌 문자가 등장하면 마지막 연산자 직후의 숫자부터 현재 읽고 있는 문자 직전까지의 문자(문자 형태로 입력된 숫자)를 숫자로 바꾸기 위해 다른 문자열로 옮기고, 현재 이전 마지막 연산자가 무엇이었는지에 따라 결과를 저장할 변수에 대해 덧셈과 뺄셈을 진행하였다. 여기서 결과를 저장할 변수는 int 형으로 선언하였는데, 들어오는 정수는 최대 4자리여도 연산 결과는 이를 넘을 수 있기 때문이다. 만약 연산자 2개가 연속으로 들어온다면, 뒤 숫자의 부호표시를 한 것이기 때문에, 2연속으로 -가 들어오면 음수를 뺀 것이므로 더하기 연산을, - 다음 +가 들어오면 양수를 뺀 것이므로 그대로 빼기 연산을 다음에 진행하도록 하였고, 이외에는 모두 나중에 입력된 연산자로 연산 되도록 했다. (++)이나 --가 들어오면 더하기, ++이나 +-이 들어오면 빼기 연산을 진행한다.) 입력된 문자열 중 첫 문자가 연산자인 경우도, 그 첫 문자 앞에 +가 있는 것처럼 처리를 진행했다. 연산 이후에는 마지막 연산자 직후의 숫자가 현재 읽고 있는 문자(연산자) 다음 위치를 가리키도록 업데이트해 주고 이 마지막 연산자 직후의 숫자가 '\0'이면 문장이 끝났다는 의미이므로 연산을 종료하였으며, 마지막으로 연산 결과를 출력하는 방향으로 문제를 해결하였다.

12. 문제 4 결과

실행을 위해서 공통 코드, 공통 헤더, 문제 4 코드가 필요하다.



```
Microsoft Visual Studio 디버그 콘솔
12+34-56+789-123-56+78
=678
D:\OneDrive - 중앙대학교\1학년 2학기\프로그래밍\실습 문제\과제 3\64\Debug\과제 3.exe(프로세스 11028개)이(가) 종료되었습니다(코드: 0x0).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

그림 9

그림 10

입력된 문장에서 첫 문자가 연산자인 경우와 연산자 2개가 연달아 오는 경우에 대한 결과이다. 식은 $(+1234)-(-5678)-(+8)+(-72)+(+869)-(-9999)$, 즉 $1234-(-5678)-8+(-72)+869-(-9999)$ 이며, 결과 값도 $1234+5678-8-72+869+9999=17700$ 으로 일치한다.

13. 소스코드

첨부파일 참조

문제별 코드 실행 시 다음 파일이 모두 필요합니다.

문제 1 코드 실행 시: 공통 코드, 공통 헤더, 문제 1 코드 필요

문제 2 코드 실행 시: 공통 코드, 공통 헤더, 문제 2 코드 필요

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

문제 4 코드 실행 시: 공통 코드, 공통 헤더, 문제 4 코드 필요

14. 참고문헌

ⁱ 이충현, Logic Circuits Chapter 4, 논리회로 02분반, 2023, p.25

ⁱⁱ <https://stackoverflow.com/questions/8023414/how-to-convert-a-128-bit-integer-to-a-decimal-ascii-string-in-c>

(left 변수를 사용하는 시점부터 참고했다. 문자열을 동적할당으로 받는 것은 참고한 내용이 아니다.)