HW2 결과보고서

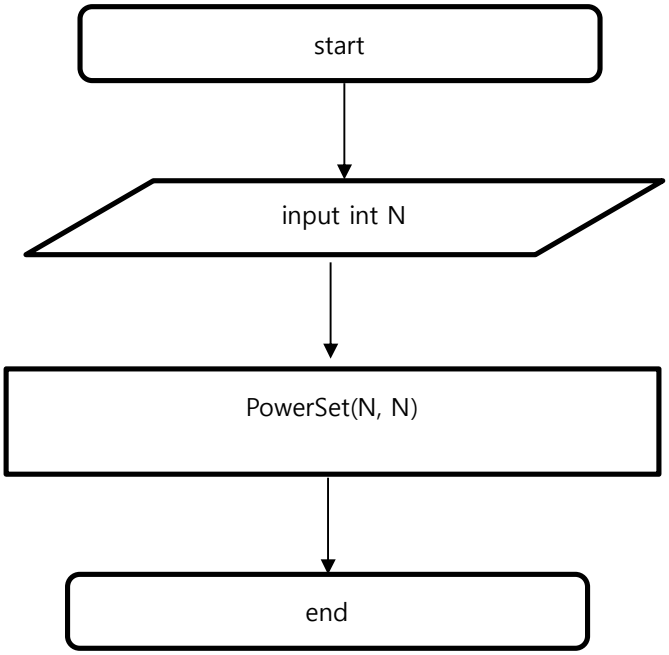
20211522

김정환

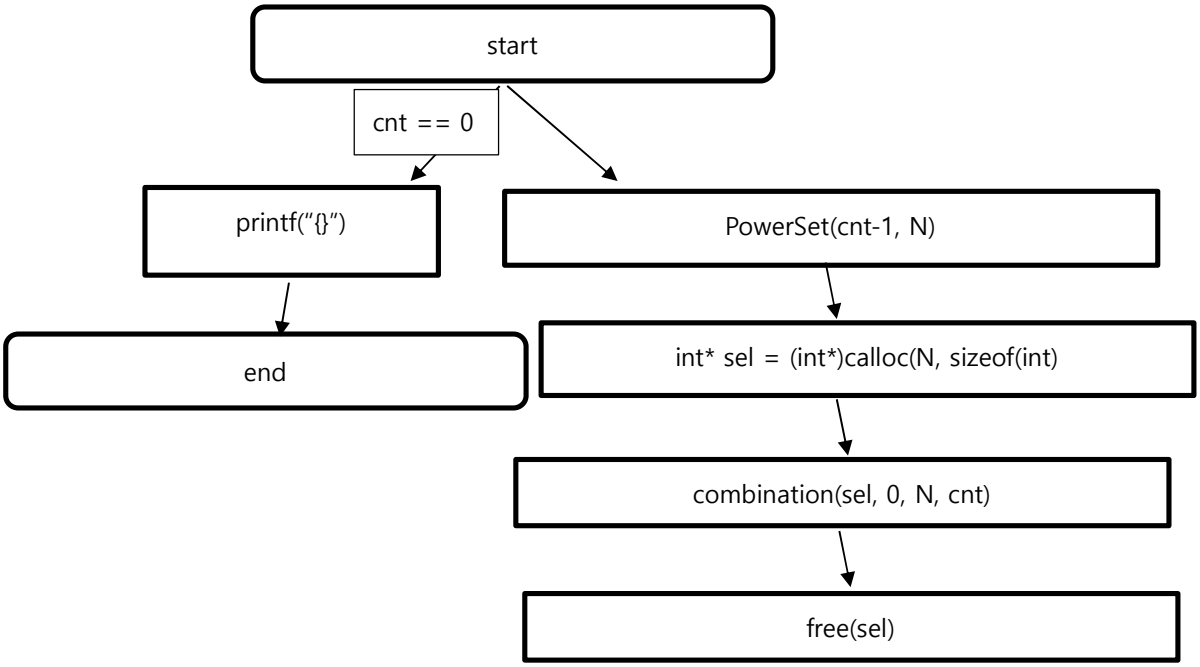
1번 문제

<flow chart>

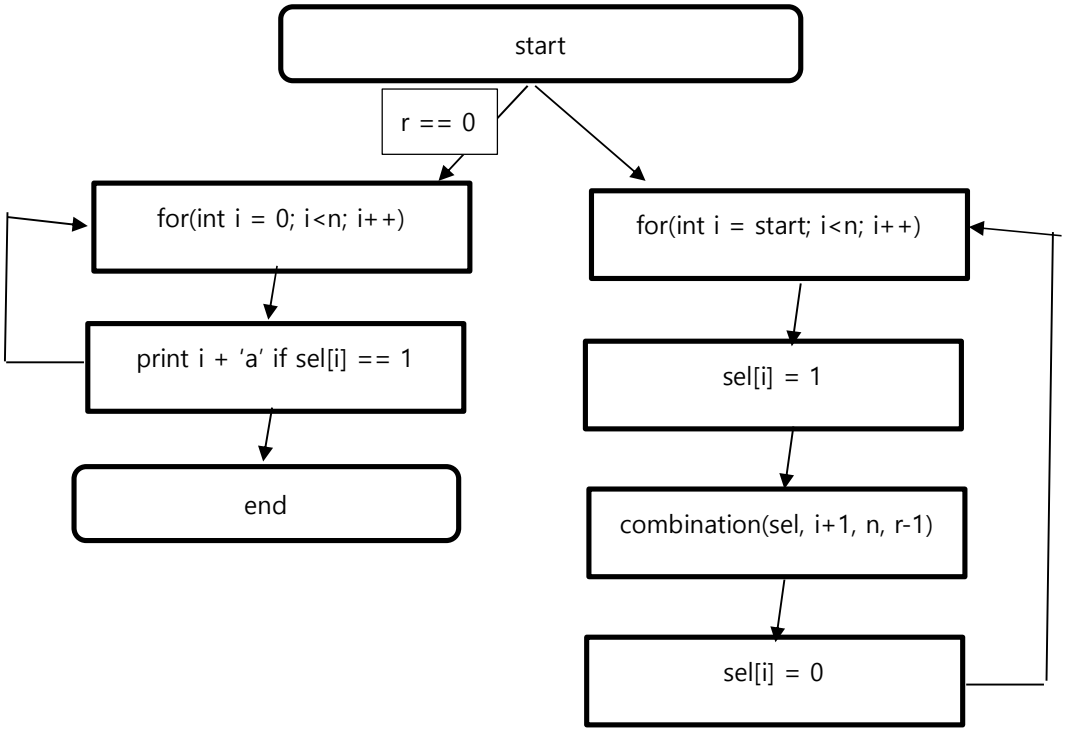
-int main()



-void PowerSet(int cnt, int N)



-void combination(int* sel, int start, int n, int r)



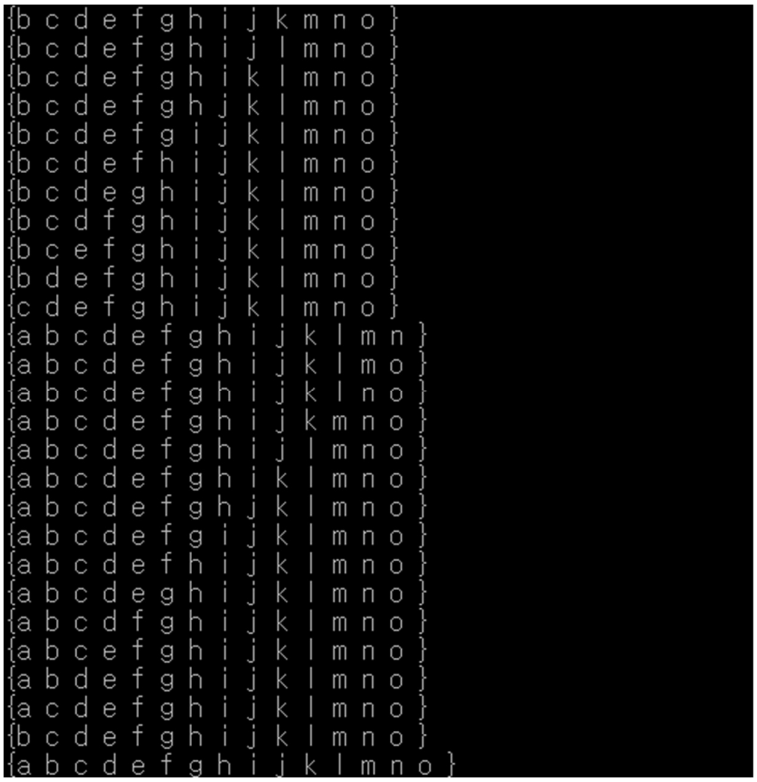
<구현 설명>

input으로 들어오는 interger 값 N은 N>0의 조건으로 input된다고 가정되어 있으므로 N < 0의 경우에 대한 처리는 생략할 수 있었다. void PowerSet(int cnt, int N) function의 경우에는 cnt의 value가 0이 되었을 때 공집합을 print하고 return하도록 하였다. 그리고 0이 되기 전까지는 PowerSet()을 recursive하게 call하면서 function argument에 cnt-1, N을 전달한다. recursive하게 call 한 이후에는 sel array를 dynamic allocation을 이용하여 memory를 allocating을 해주는데, calloc을 사용하여 0으로 value가 initiating될 수 있도록 하였다. 그리고 array의 size는 N으로 하였다. 이는 combination의 경우를 print 해줄 combination function에 전달하는데 쓰이는데, allocating 이후, combination(sel, 0, N, cnt)로 function을 call한뒤 free(sel)로 memory를 해제한다. void combination(int* sel, int start, int n, int r)의 경우에는 argument로 전달받은 sel은 combination에 포함된 alphabet을 나타낼 array, start는 각각의 call에서 for loop의 시작지점, n과 r은 combination에 해당하는 n과 r을 나타낸다. r이 0일 때 출력을 하는데, 이는 for loop를 돌면서 sel array에 1로 표기된 경우들만 i + 'a'를 통해 알파벳으로 print되도록 한다. 그 외의 case에는 start 부터 n-1까지 for loop를 돌면서 sel[i]를 1로 바꾸고 recursive call을 통해 combination(sel, i+1, n,

r-1)을 call하고 sel[i]를 0으로 다시 돌려놓으면서 수행한다.

구현한 code를 test해보았다. 우선 주어진 pdf 파일에 있는 예시인 3이 input으로 주어졌을 때를 test해보았다. 해당 경우에

위와 같이 결과를 확인할 수 있다. 주어진 조건에서 N>0인 경우만 input된다고 했기에 test해볼 특수 case는 없고, 상대적으로 N이 큰 경우만 test해보았다. N이 15로 input되었을 때를 test해보았을 때

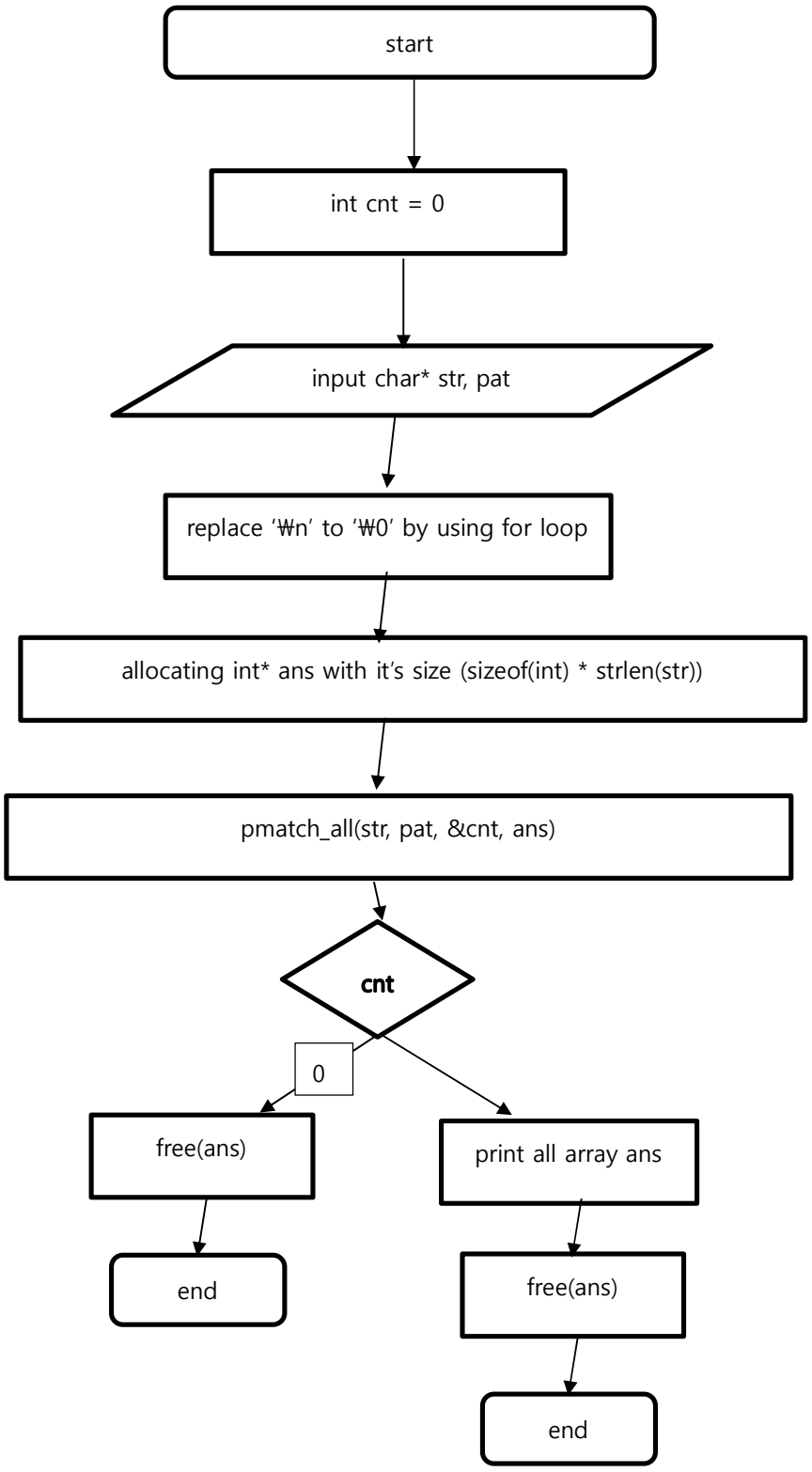


위와 같이 print되는 줄 수가 많아서 일부만 캡처하였으나 element의 개수가 14인 set과 15인 set의 case를 확인해보았을 때 잘 print된 것을 확인할 수 있다.

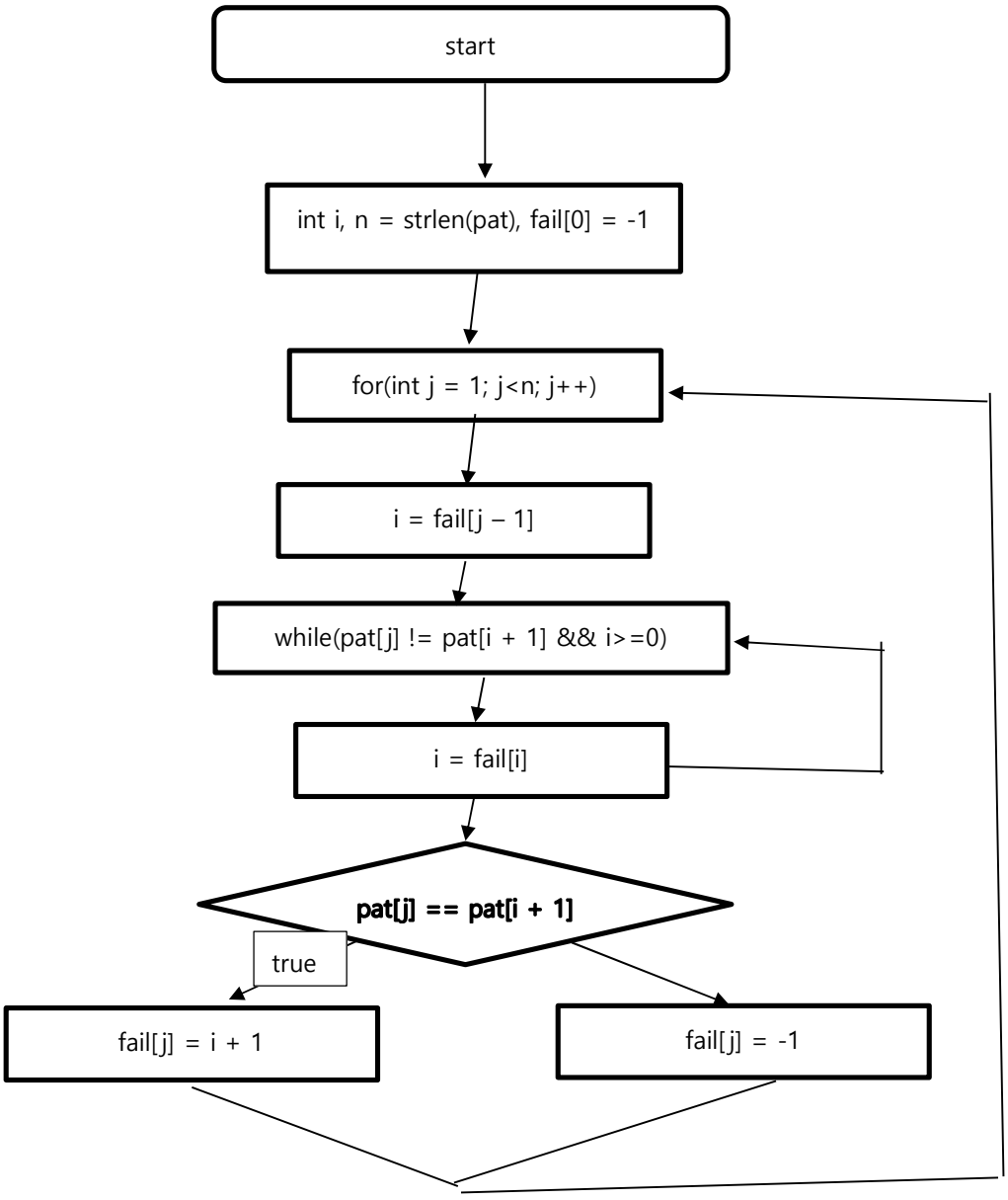
2번 문제

<flow chart>

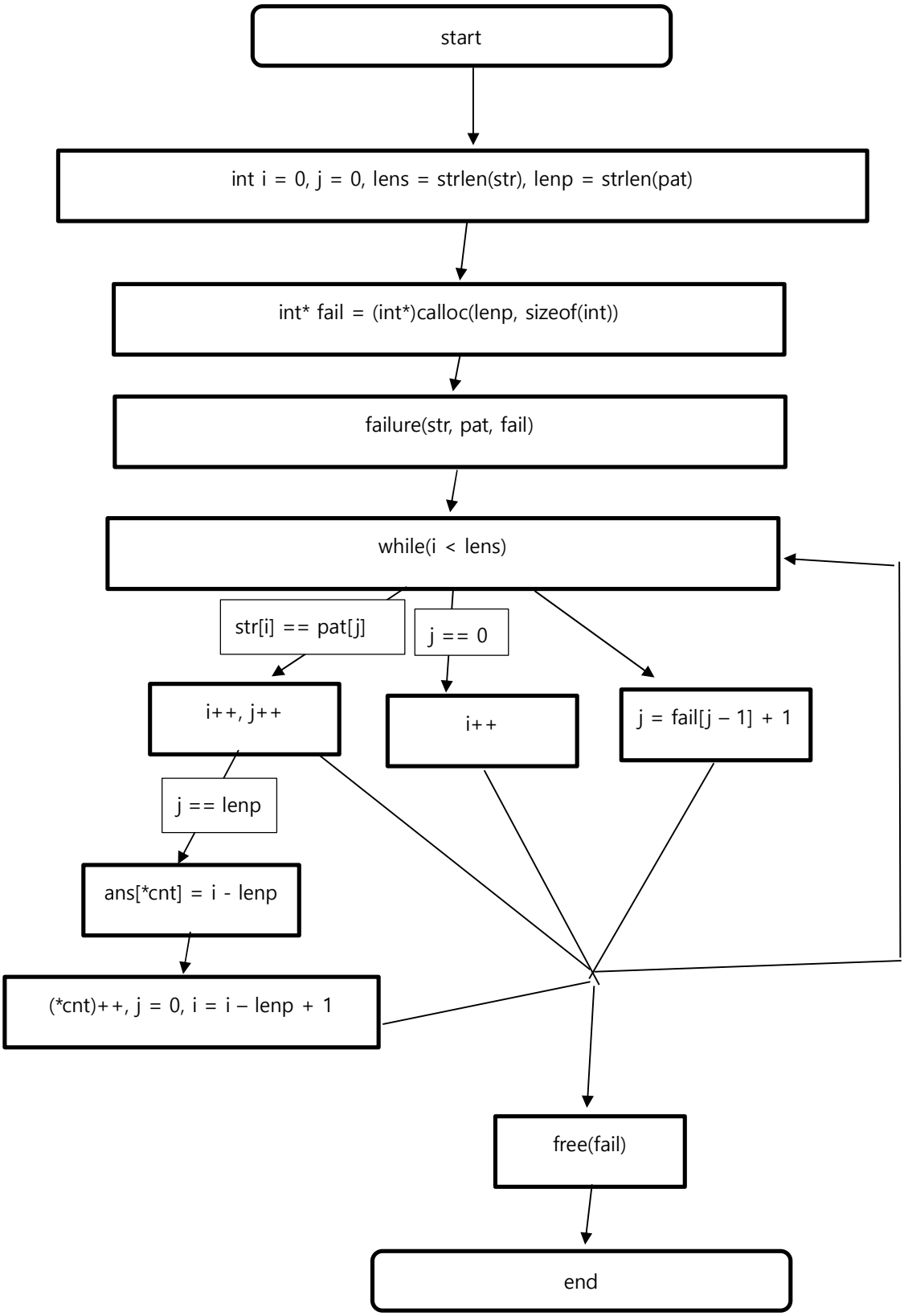
-int main()



-void failure(char* arr, char* pat, int* fail)



-int pmatch_all(char* str, char* pat, int* cnt, int* ans)



<구현 설명>

초반 구현은 강의 자료의 pmatch를 참고하여 구현하였다. 해당 code는 처음 pattern을 발견했을 때 해당 index를 return하고 종료하는 function이다. 여기서 추가로 모든 index를 return할 수 있도록 변형하였다. 해당 방식을 구현하기 위해 추가된 index의 개수를 저장할 cnt variable과 index들을 저장할 ans 배열을 추가로 pmatch_all function에 argument로 전달하였다. 여기서 if 문에서 str[i] == pat[j] condition이 만족되는 경우 중에서 j == lenp인 condition을 추가로 구분하였다. 해당 case가 pattern을 하나 찾은 것이므로 해당 case에서 ans[*cnt]에 i – lenp를 넣어주고, *cnt는 1 증가시킨다. 그리고 j는 0으로 initialize해주고, i = i – lenp + 1로 바꿔주어 바로 뒷 부분부터 search하도록 구현하였다. 이 방식으로 pmatch_all function은 ans array에 찾은 index들을 저장한다. main function은 우선 각각 search할 string을 저장할 char str[31]과 pattern을 저장할 char pat[31]을 declare하였다. 각각 최대 31글자로 문제의 조건에 주어져 있어 각각의 size를 31로 하였다. 또한 pmatch_all function에 전달할 cnt도 0으로 initialize하면서 declare하였다. cspro 상에서 redirection을 통해서 file이 입력되므로 stdin으로 input을 구현하면 input되기에 scanf를 이용하여 str과 pat의 입력을 받고, '₩n'을 제거하기 위해 for loop를 통해 '₩n'을 '₩0'으로 replace해주었다. index를 저장할 ans array는 최대 str의 size만큼 나오기 때문에 dynamic allocation으로 strlen(str) 만큼의 integer array를 allocating하였다. 그 뒤 pmatch_all을 call하여 str, pat, &cnt, ans를 argument로 전달하였다. 그 후 cnt가 0, 즉 아무 index도 없을 때는 ans를 free로 memory를 해제하고, 아닌 경우에는 ans array를 모두 print한 후 ans의 memory를 해제한다.

이제 해당 program의 Perfomance와 제대로 작동하는지 확인해보았다. 첫 번째로 pdf에 주어진 아무 index가 나오지 않는 경우이다.

다음과 같이 answer.txt에 넣어주었을 때 아무 출력이 안 나오는 경우를 확인할 수 있다. 다음으로 pdf 주어진 경우에서 결과가 print되는 경우를 확인하였다.

위와 같이 원하는 index 값이 잘 print되는 것을 확인할 수 있다. 다음으로는 string보다 pattern의 size가 더 큰 case에 대해서 확인했다.

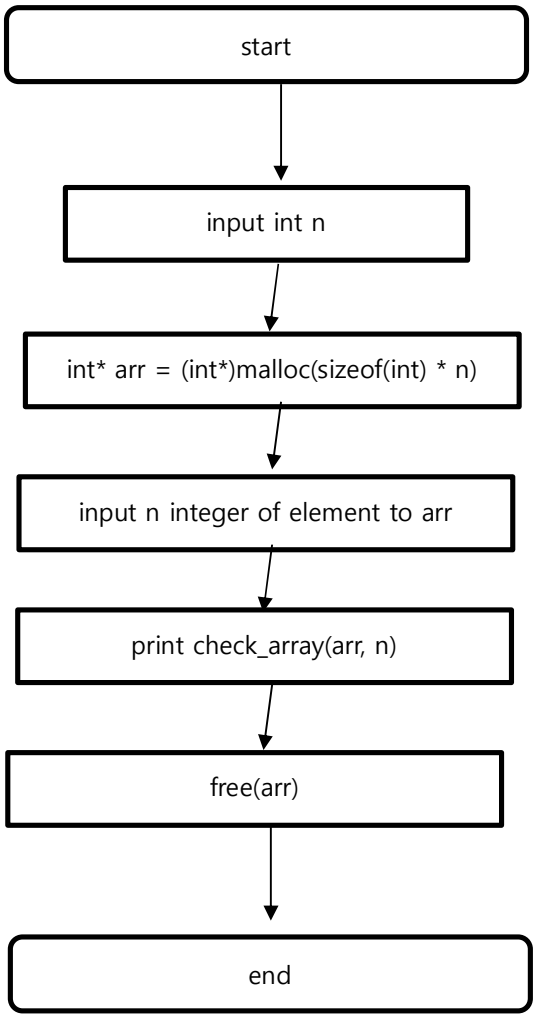
```
cse20211522@cspro:~/HW2$ cat answer.txt
bb
bbbcse20211522@cspro:~/HW2$ ./test2 < answer.txt
cse20211522@cspro:~/HW2$
```

위와 같이 아무런 결과도 나오지 않는 것으로 정상적으로 작동함을 알 수 있다.

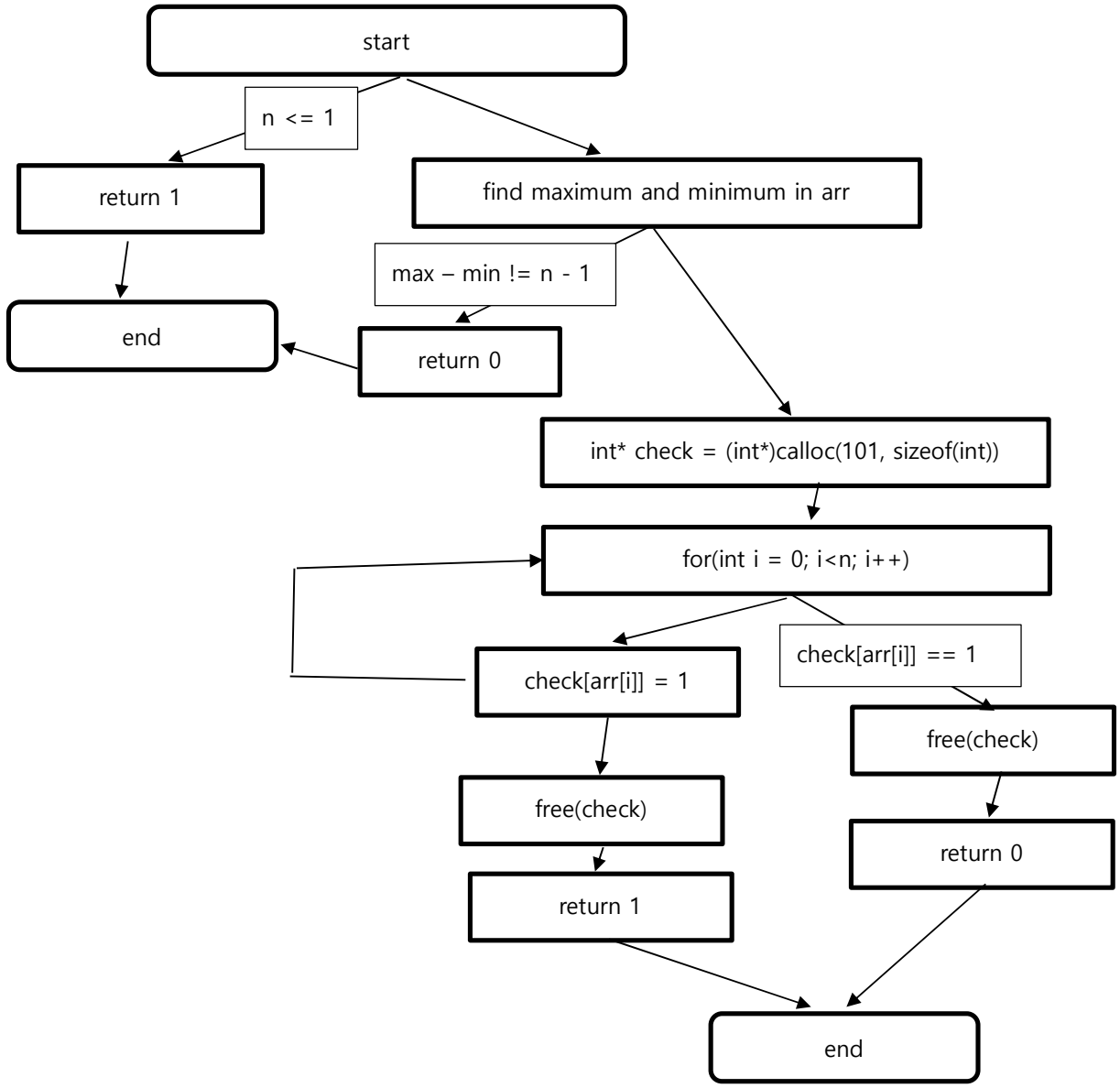
3번 문제

<flow chart>

-int main()



-int check_array(int* arr, int n)



<구현 설명>

main function에서는 file을 redirection으로 받기에 stdin 입력으로 구현하였다. array의 element 수를 넣어줄 integer type의 n을 declare하고, input된 n만큼 int* arr에 memory를 sizeof(int) * n의 size로 memory를 allocating한다. 그 후 for loop를 통해 arr에 element를 input받는다. 그 후 printf 내에서 check_array(arr, n)을 call하여 value를 받아 출력 후 arr의 memory를 해제하고 종료

된다. int check_array(int* arr, int n) function의 구현은 우선 n이 1 이하인 case에 대해서는 1을 return하여 true임을 나타낸다. 그 외에는 우선 array를 brute force로 search하여 maximum과 minimum을 찾아서 각각 max, min variable에 저장한다. 이 때, max와 min의 차이가 주어진 array의 size인 n보다 1 작은 값인 n – 1과 같지 않으면 pigeonhole principle과 유사하게 겹치거나 연속하지 않는 숫자가 존재하는 것이다. 따라서 이 경우에는 0을 return하면서 false임을 나타낸다. 이 과정까지 filtering한 후에는 pointer to integer type인 check variable을 declare하여 문제에서 주어진 조건을 봤을 때 arr에 들어있게 될 element의 maxkmum size가 100으로 주어졌으므로 check에는 sizeof(int)만큼의 size를 101개 가진 array를 dynamic allocation 해주었다. 그 후 arr array를 for loop로 index가 0부터 n-1까지 전부 search하면서 arr[i]가 이미 1로 표시되어 있을 때 check의 memory를 free해주고, 0을 return하여 false임을 나타낸다. 처음 나오는 value일 경우 check[arr[i]]를 1로 바꿔준다. for loop를 끝까지 돌았을 경우는 모두 연속하다는 것이므로 check의 memory를 free해주고, 1을 return하여 true임을 나타낸다.

다음으로는 program이 원하는 대로 작동하는지와 Performance에 대해서 확인을 해보았다. 우선 pdf에 있는 case에 대해서 test해보았다.

```
cse20211522@cspro:~/HW2$ cat input.txt
6
10 14 12 15 11 9cse20211522@cspro:~/HW2$
cse20211522@cspro:~/HW2$ ./test3 < input.txt
0
```

위의 사진을 보면 연속되지 않은 경우에 대해서 결과가 잘 나오는 것을 확인할 수 있다. 다음으로 연속된 경우에 대한 test도 아래와 같다.

```
cse20211522@cspro:~/HW2$ cat input.txt
5
4 1 5 2 3
cse20211522@cspro:~/HW2$ ./test3 < input.txt
1
```

입력되는 array의 element가 1개인 경우도

```
cse20211522@cspro:~/HW2$ cat input.txt
1
1cse20211522@cspro:~/HW2$ ./test3 < input.txt
1
```

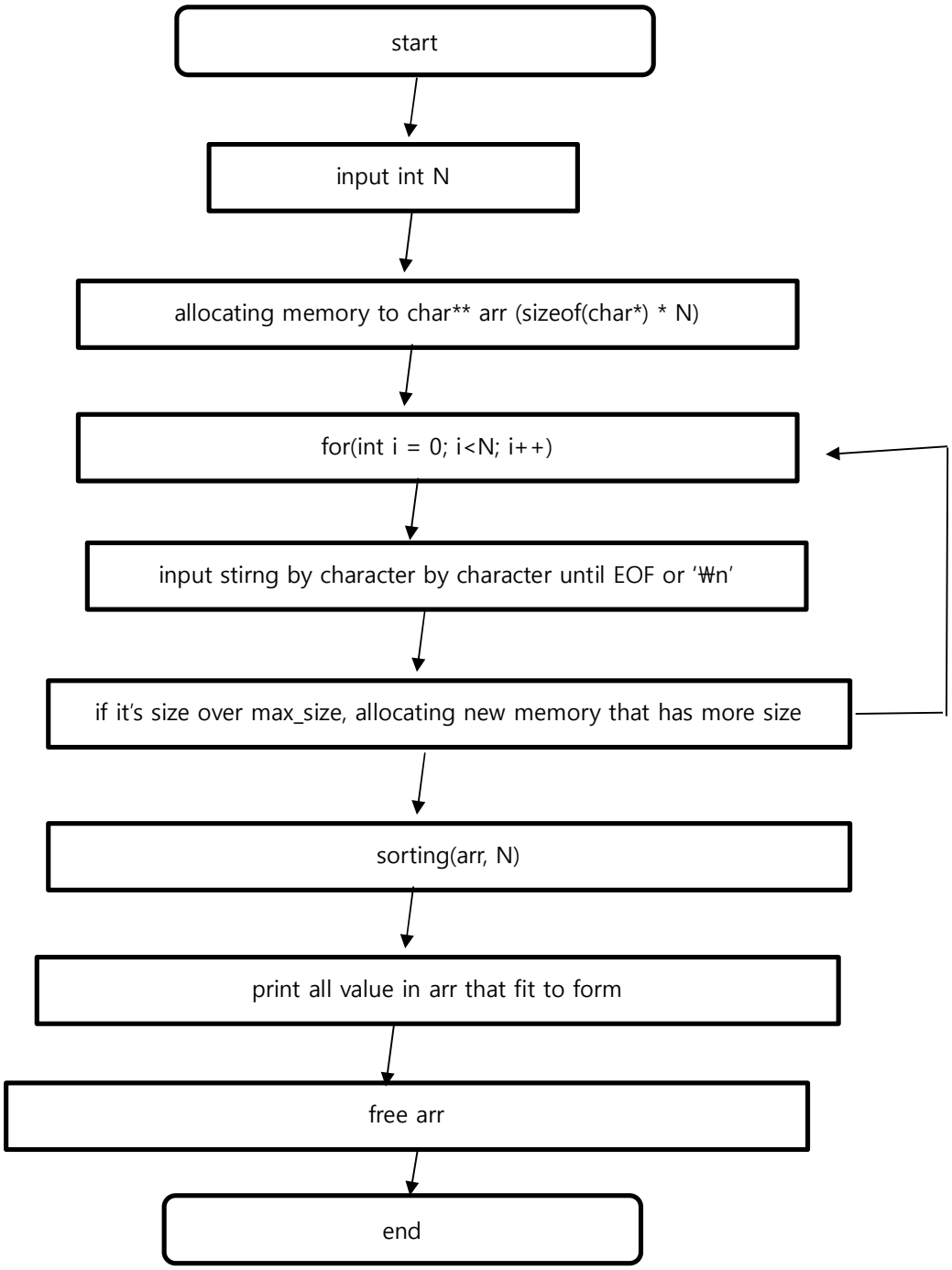
결과가 1로 잘 나온다. 주어지는 n이 1 이상 100 이하 이므로 크게 더 test해볼 case는 없는 것으로 판단하였다. 다음으로 이 문제에서는 check_array function의 time complexity가 O(n)이 되도

록 하는 조건이 있다. 구현한 function을 살펴보면 maximum과 minimum을 찾을 때 for loop 하나로 1 ~ n-1까지, 그리고 check array에 1로 표시하면서 겹치는 것을 확인할 때 0 ~ n-1까지 for loop 하나가 사용된다. 여기서 time complexity가 O(n)임을 확인할 수 있으므로 조건에 부합한다.

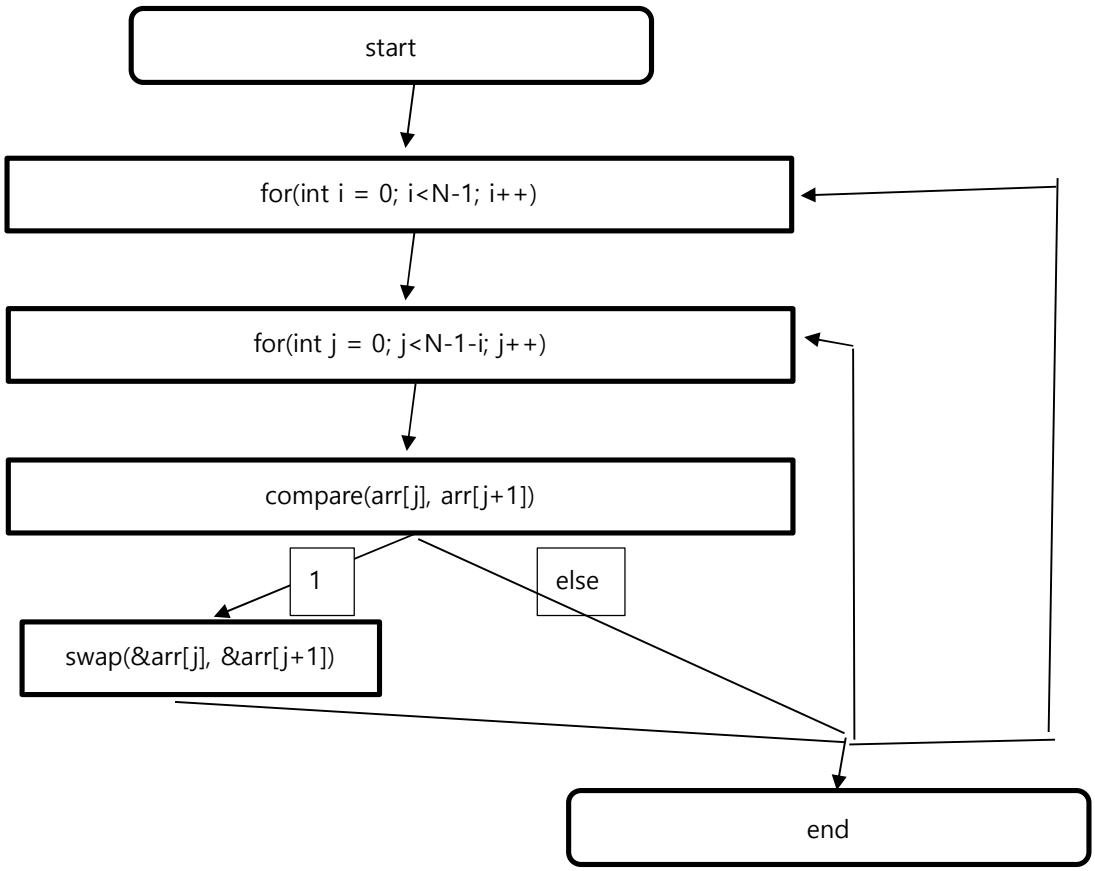
4번 문제

<flow chart>

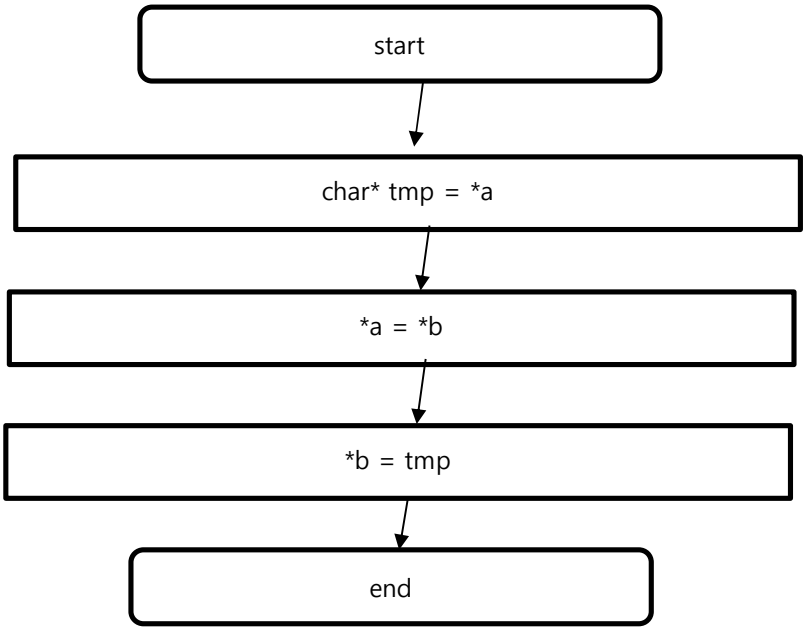
-int main()



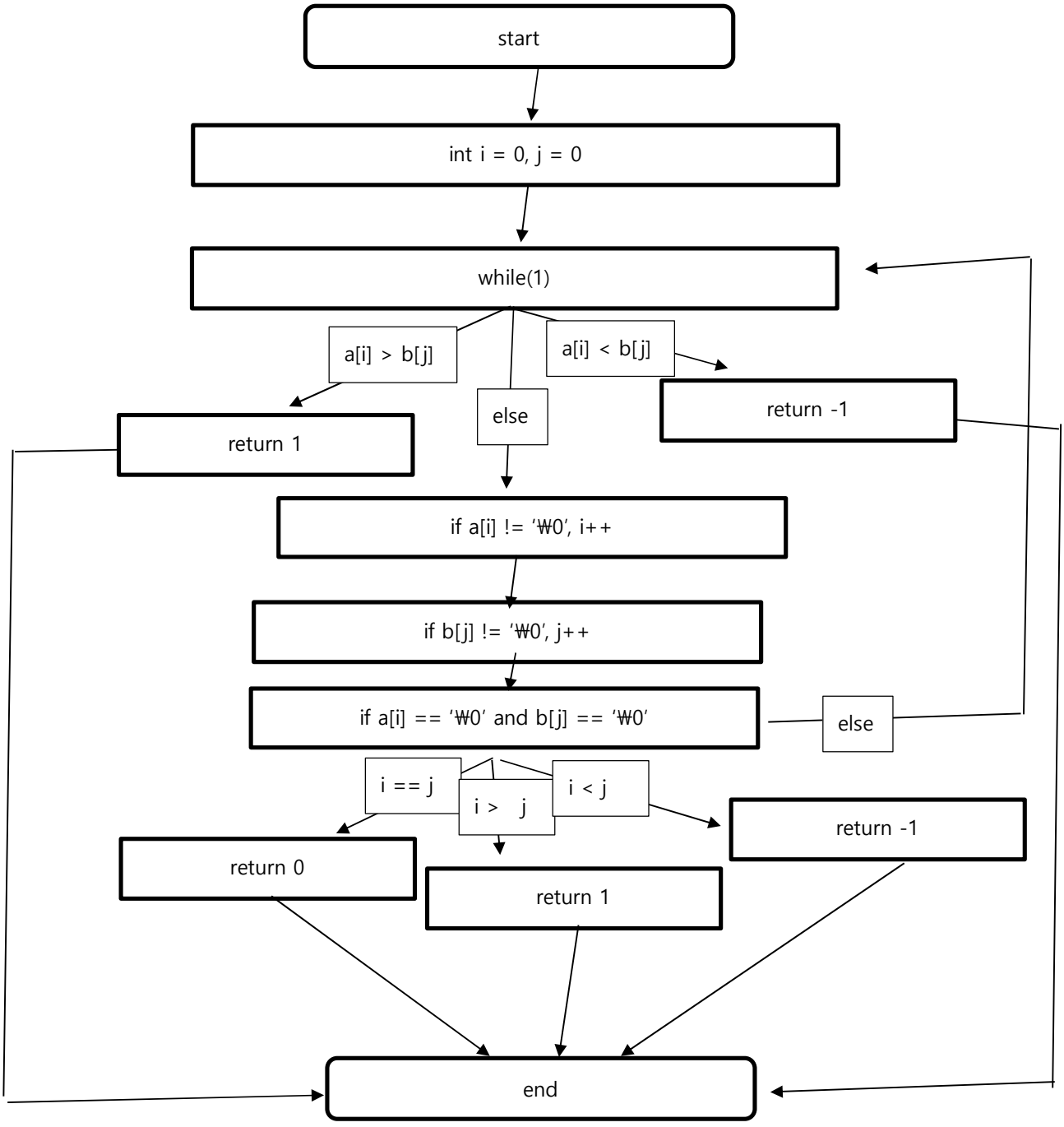
-void sorting(char** arr, int N)



-void swap(char** a, char** b)



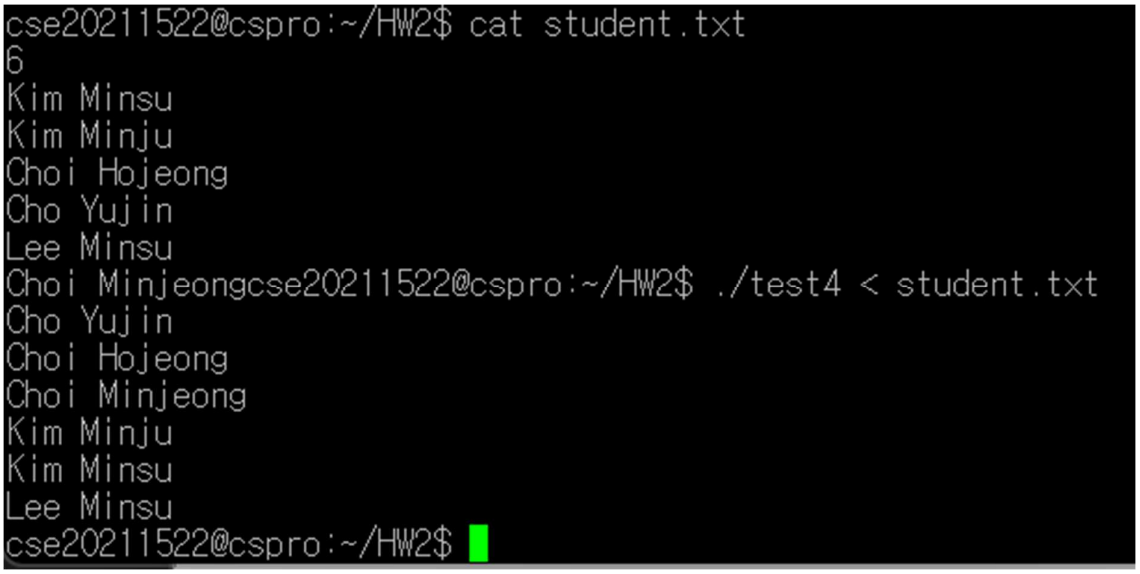
-int compare(char* a, char* b)



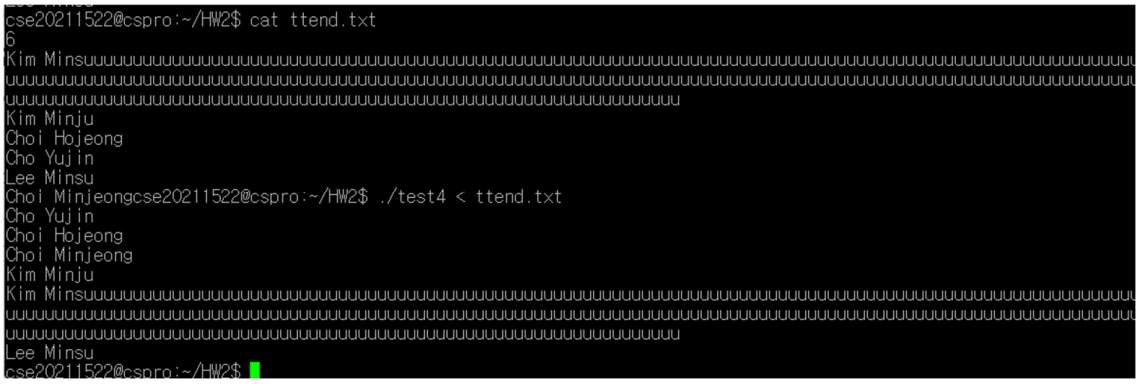
<구현 설명>

main function에서 학생 이름을 input받을 때, 우선 integer value이자 학생 수인 N을 input받는다. 이후 string으로 학생 이름을 input받을 때, '₩n'이나 '₩0'으로 한 line이 차지 않도록 getchar()로 먼저 input 받아 처리한 후, 학생 이름을 저장할 array arr을 declare하여 sizeof(char*) * N의 size로 dynamic allocation해준다. for loop를 통해 학생 이름을 받는데, 각각의 loop마다 우선 char type array를 101의 size로 생성하였다. 이는 대부분의 학생 이름이 100자를 넘지 않을 것이기에 적당한 수를 부여하였다. 그 후 input되는 글자수를 셀 integer type variable인 cnt, 최대 size를 저장하고 있을 max_cnt를 선언하여 각각 0, 101로 초기화하였다. 그 후 while loop로 one character 씩 txt 파일로부터 input받는다. 이를 cnt를 1씩 증가하면서 arr[i][cnt]에 저장하는데, 해당 값이 EOF거나 '₩n'일 때 해당 character를 '₩0'으로 바꿔준 후 while loop를 break로 멈춰 다음 이름을 input받도록 한다. 또한 cnt+1>=max_size 즉, 다음 character에 max_size에 도달할 때, char* temp를 선언하고 해당 pointer에 기존의 array를 저장해두고, arr[i]를 free 처리했다가 arr[i]에는 (max_size + 1) * sizeof(char) 만큼의 array를 새로 allocate해주어 여기에 기존 value들을 옮기고, temp를 free처리 해주어 size를 늘려준다. 이 방식으로 학생 이름을 input받고, sorting function을 arr, N을 argument로 전달해주면서 call한다. 그 후 sorting된 array를 순서대로 print한 후, arr의 memory를 free처리해주며 종료한다. void sorting(char** arr, int N) function에서 arr은 학생들의 이름이 저장된 array이고, N은 학생 수이다. sorting function은 bubble sort를 사용하여 구현하였다. for loop를 2개 사용하고, compare function을 call하여 해당 function이 1을 return하면 swap function을 통해 두 string의 index가 교환된다. int compare(char* a, char* b) function은 string 2개를 argument로 받아 비교한다. 우선 탐색할 index를 저장할 integer variable i와 j를 declare하고 string을 처음부터 비교할 것이므로 0으로 initialize해준다. 그 후 while loop의 조건에 1로 true value를 넣어주어 내부에서 break 하기 전에는 멈추지 않도록 하고, condition을 나눈다. 전달한 string 중 a가 sorting function에서 쓰일 때 더 index가 작은 string이다. 따라서 a의 order가 높은 경우에 swap되도록 해야 하고 1을 return할 때 swap되므로 a[i] > b[j]일 때 1을 return한다. 단 i와 j가 같을 때만 비교해야 하므로 i == j 조건도 &&로 연결했다. 앞에서부터 비교하기 있기 때문에 차이가 있을 때 그 지점에서 바로 return하여 종료해도 비교가 완료된다. 반대로 a[i] < b[j]이면 -1을 return하고, 그 외일 때, i와 j를 1씩 increase하는데, 각각 '₩0'이 아닐 때만 increase시킨다. 또한 a와 b 모두 '₩0'에 도달했을 때는 break로 loop에서 나온다. 두 string에서 비교가 차이 나지 않을 때는 끝까지 loop가 돌고 난 후 i와 j를 통해 길이를 비교한다. i == j인 경우, 0을 return하고, i>j일 때는 a의 order가 더 높으므로 1을 return 하고, 나머지 경우는 i<j이므로 -1을 return한다. void swap(char** a, char** b) function은 argument로 들어온 두 string a, b를 서로 바꿔주는 역할이다. char** type으로 argument를 받아 구현하였고 나머지 사항은 integer type을 swap할 때와 유사하게 구현하였다.

다음으로 program이 잘 correctness와 efficiency에 대한 부분이다. 우선 pdf 파일에 주어진 예제에 대해서 test해보았다.



위처럼 결과가 문제없이 잘 print되는 것을 볼 수 있다. 다음으로 test해보아야 하는 case는 이름이 매우 긴 사람이 포함된 경우이다. 위의 예제에서 Kim Minsu의 이름의 뒤에 u를 길게 연결하여 100자가 훨씬 넘도록 하였다. 해당 예제를 test한 결과는 다음과 같다.



긴 이름도 문제없이 포함되는 것을 확인할 수 있다. 다음으로는 string의 길이 차이에 대한 부분으로 sort가 잘 되는지 test해보았다.

위와 같이 모두 같은 character로 string의 length만 다르게 한 경우도 잘 sort되는 것을 확인할 수 있다. 해당 program의 efficiency에 대해서는 우선 sorting function에 bubble sort를 사용한 점이 time complexity에 있어서 효율적이지 않다. 해당 부분에 있어서는 더 나은 sorting algorithm을 사용하여 개선할 수 있을 것이다. 두 번째로 string을 input받는 과정에서 기존에 allocating한 array가 부족할 때 최대 size를 늘리는 과정이 전체를 search하는 for loop를 두 번이나 사용하여 효율적이지 않다. 이 두 부분을 개선한다면 더 efficient한 program이 될 것이다.