HW5 결과보고서

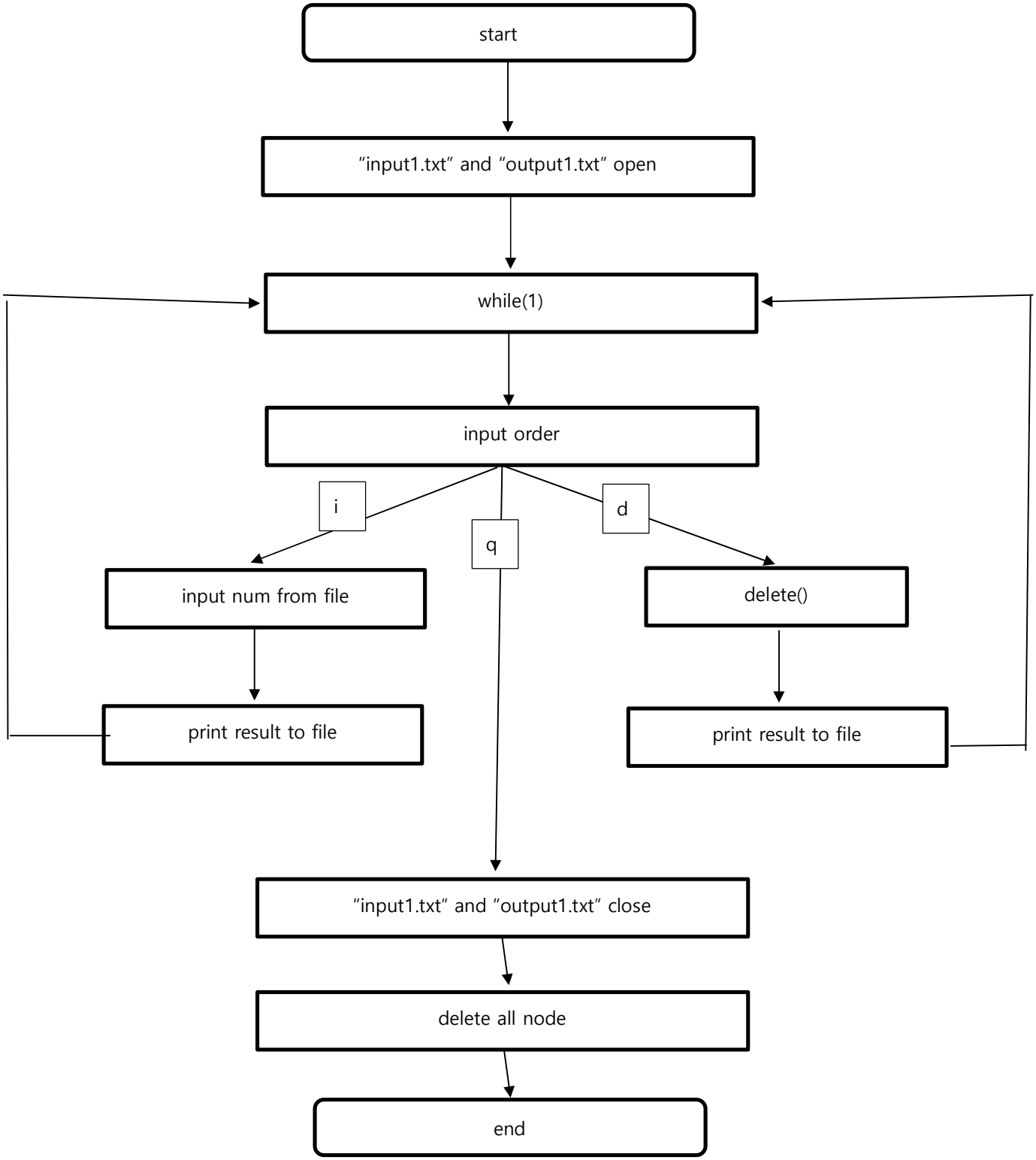
20211522

김정환

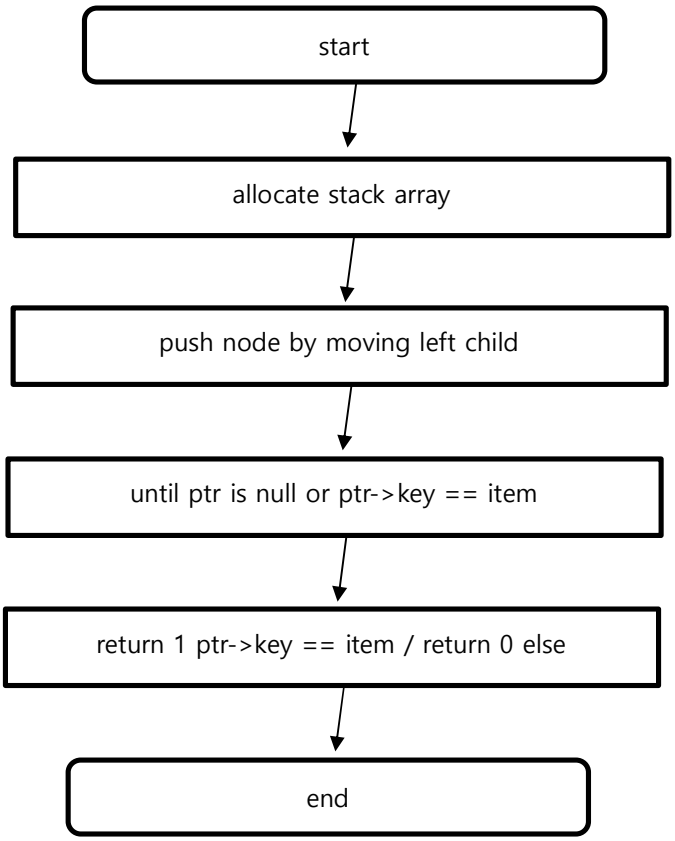
1번 문제

<flow chart>

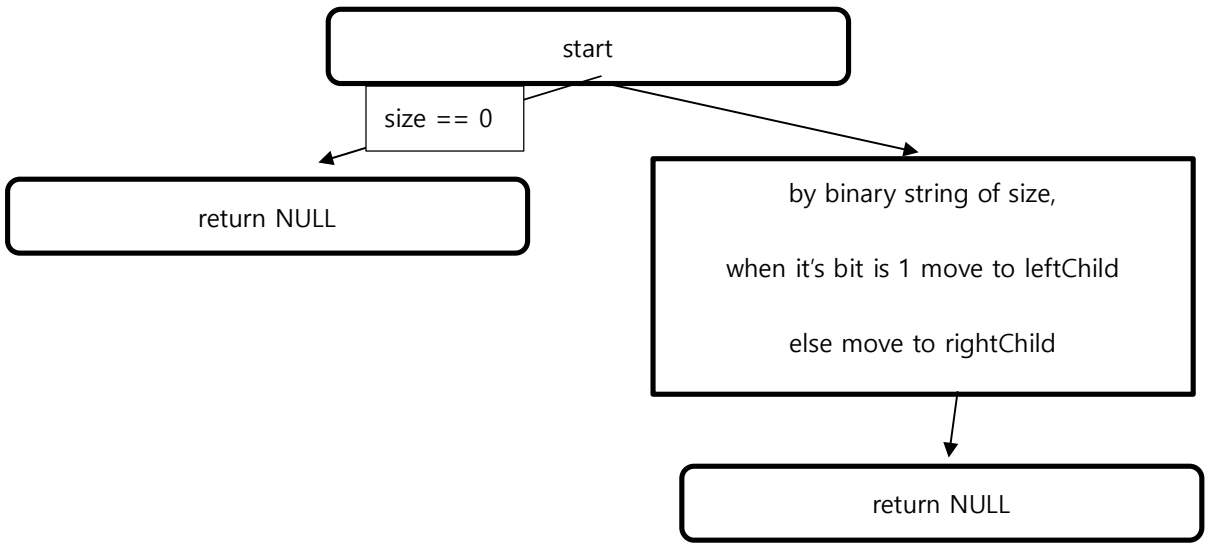
-int main()



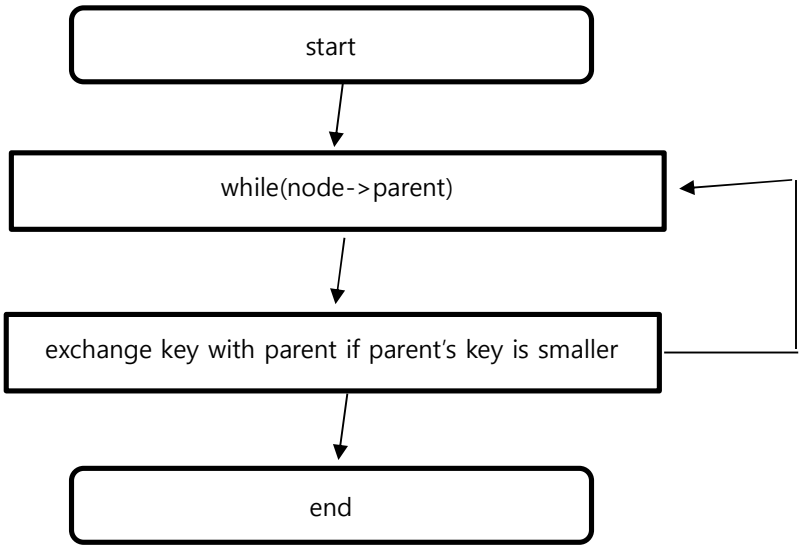
-int search(int item)



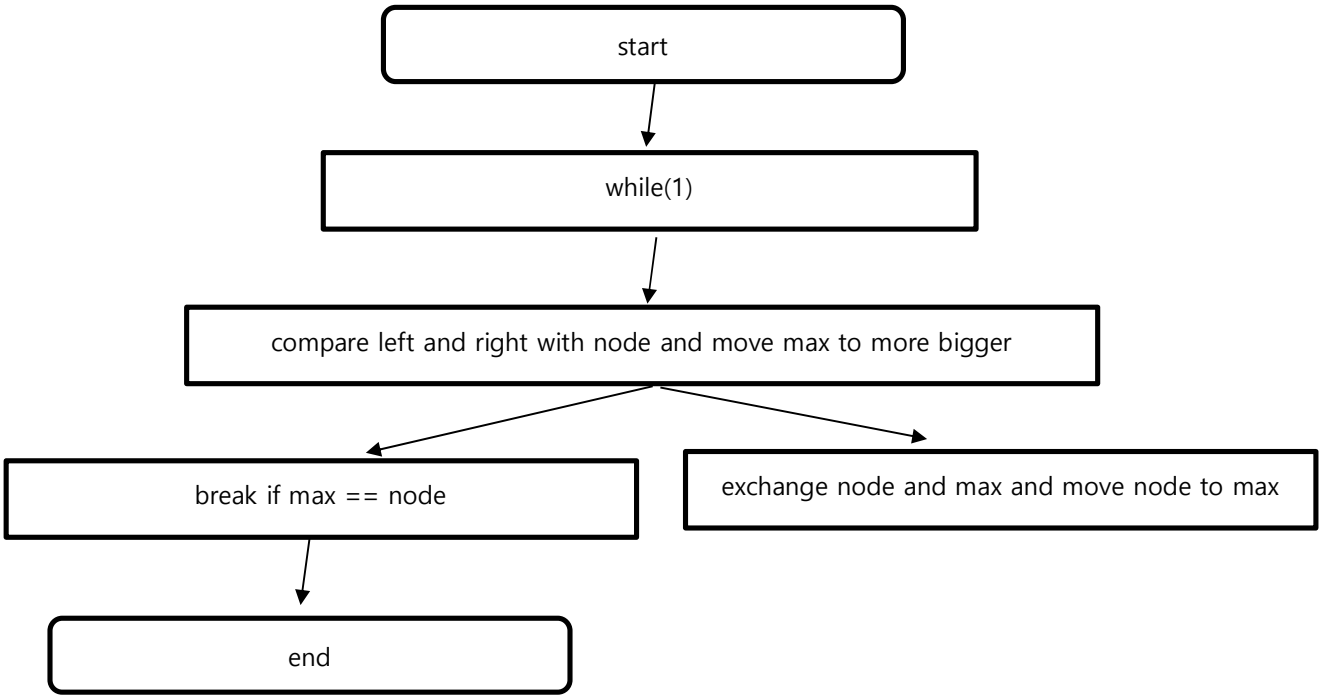
-treePointer insert_loca()



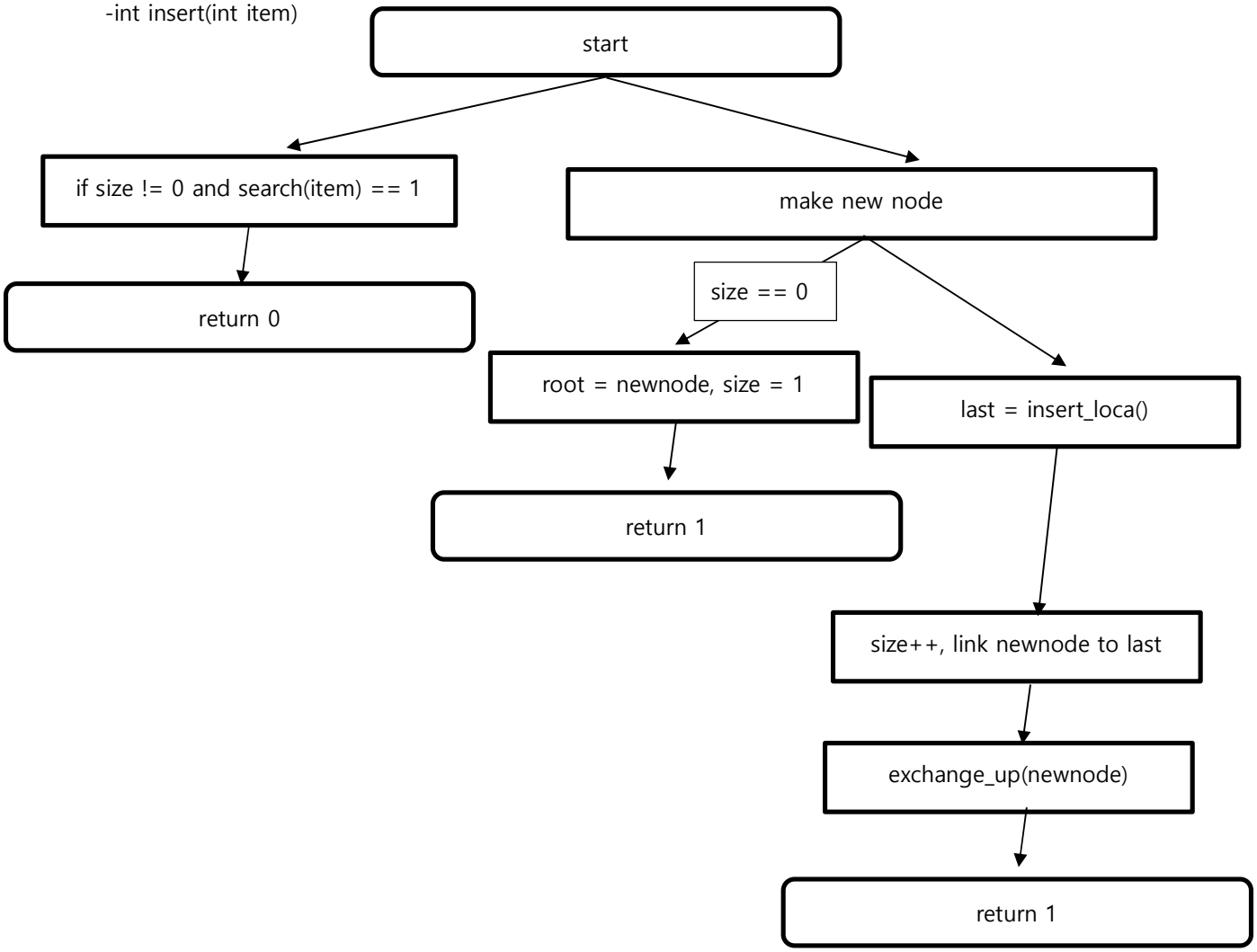
-void exchange_up(treePointer node)



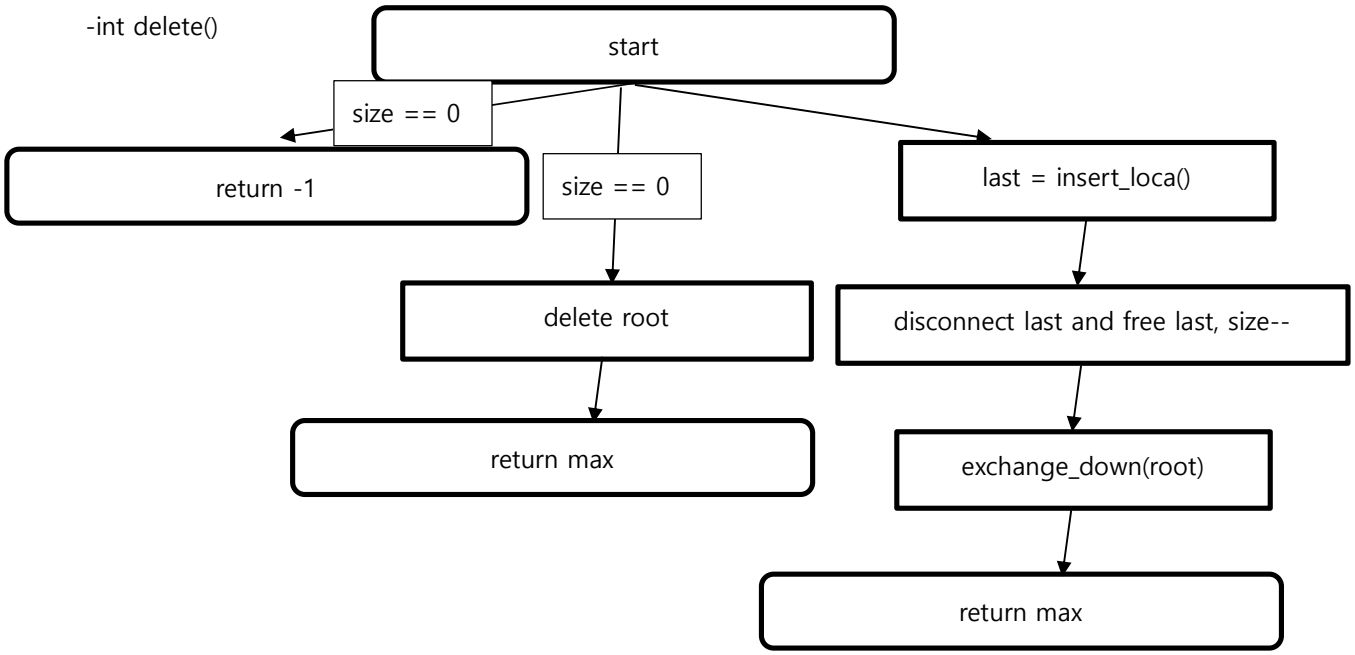
-void exchange_down(treePointer node)



-int insert(int item)



-int delete()



<구현 설명>

tree의 struct는 int type의 key value와 treePointer로 alias를 declare한 node* type의 parent, leftChild, rightChild로 구현했다. main function에서는 input1.txt와 output1.txt 파일을 열어서 input1.txt로부터 input을 받는다. output1.txt는 앞으로 print할 내용이 들어가는 file이다. condition을 1로 한 while loop를 통해서 input을 받는데 char type의 order variable에 fscanf로 input 받아 switch 문으로 구분하였다. 'i'인 case에서는 int type의 num에 input을 받아 node에 넣은 integer를 input 받는다. 그 후 insert(num)을 ins variable에 저장하고, ins가 0인 경우에 이미 존재하는 integer인 경우이므로 문제에 주어진 문장을 print하도록 하고, 아닌 경우에는 Insert (num)을 print하도록 하였다. order가 'd'인 경우에는 del variable에 delete()의 return value를 저장한다. del이 -1인 경우는 heap이 비어있는 경우이므로 문제에 주어진 양식대로 print하고 아닌 경우에는 "Delete (del)"을 print한다. order가 'q'인 경우에는 기존에 0이던 flag를 1로 바꾼다. 이외의 모든 default value에는 continue를 실행한다. 그 후 flag가 1인 경우에 while loop를 break로 escape한다. 그 후 input1.txt과 output1.txt를 닫고, del이 -1이 될 때까지 실행하여 모든 memory를 free 처리한다. insert function에서 쓰이는 다른 function들로는 search, inte_to_string, insert_loca, exchange_up이 있다. insert에 대한 설명 이전 먼저 설명하자면 int search(int item)은 item과 같은 value가 있는지 확인하는 function이다. search function은 stack을 이용하여 search를 진행한다. 우선 조건을 달지 않은 for loop를 통해 계속 반복하도록 하고, tree 내에서 계속 leftChild로 이동하며 stack의 array에 넣는다. tree 내에서 마지막 leftChild까지 도달하면 top이 0 이상일 때 stack에서 한 node를 pop한다. 모든 node가 pop되면 break로 for loop를 종료하고 그 전에는 만약 item가 같은 key를 가진 node가 있으면 stack array를 free 처리하고 1을 return하여 true임을 나타낸다. 그 후 ptr은 rightChild로 이동시켜 모든 node를 조사할 수 있도록 한다. 모든 node 조사 후에도 같은 value가 없으면 stack array를 free 처리하고 0을 return한다. void inte_to_string(char* arr, int num)은 char type array인 arr에 num을 binary로 나타낸 string으로 변환하여 저장하는 function이다. 이는 binary number로 변환하는 부분에서 '0'을 더해 char type으로 올바르게 나타내는 부분만 추가되어 있다. treePointer insert_loca()는 마지막 node의 pointer value를 return하는 function이다. size가 0인 경우에는 NULL을 return하고, 그 외에는 binary number string을 이용한다. 이 경우에는 끝까지 search하면서 각 character가 '1'인 경우 leftChild로, '0'인 경우 rightChild로 이동하는 방식이다. 이동 후 ptr을 return하며 종료된다. void exchange_up(treePointer node)는 마지막에 추가된 node의 key value가 올바른 위치로 갈 수 있도록 하는 function이다. 이는 while loop를 통해서 node->parent가 NULL이 아니고 node->parent->key가 node->key보다 작은 동안 서로의 key value를 swap하고 node를 node->parent로 이동하는 것이 반복된다. 이들을 이용하여 int insert(int item)을 구현하였다. 우선 size가 0이 아닌 경우, search(item)이 1을 return하면 이미 item이 들어있는 것이므로 0을 return한다. 그 외에는 새로운 node로 newnode를 allocate하고, size가 0인 경우는 root가 필요하므로 root를 newnode로 한다. 그 외에는 last variable에

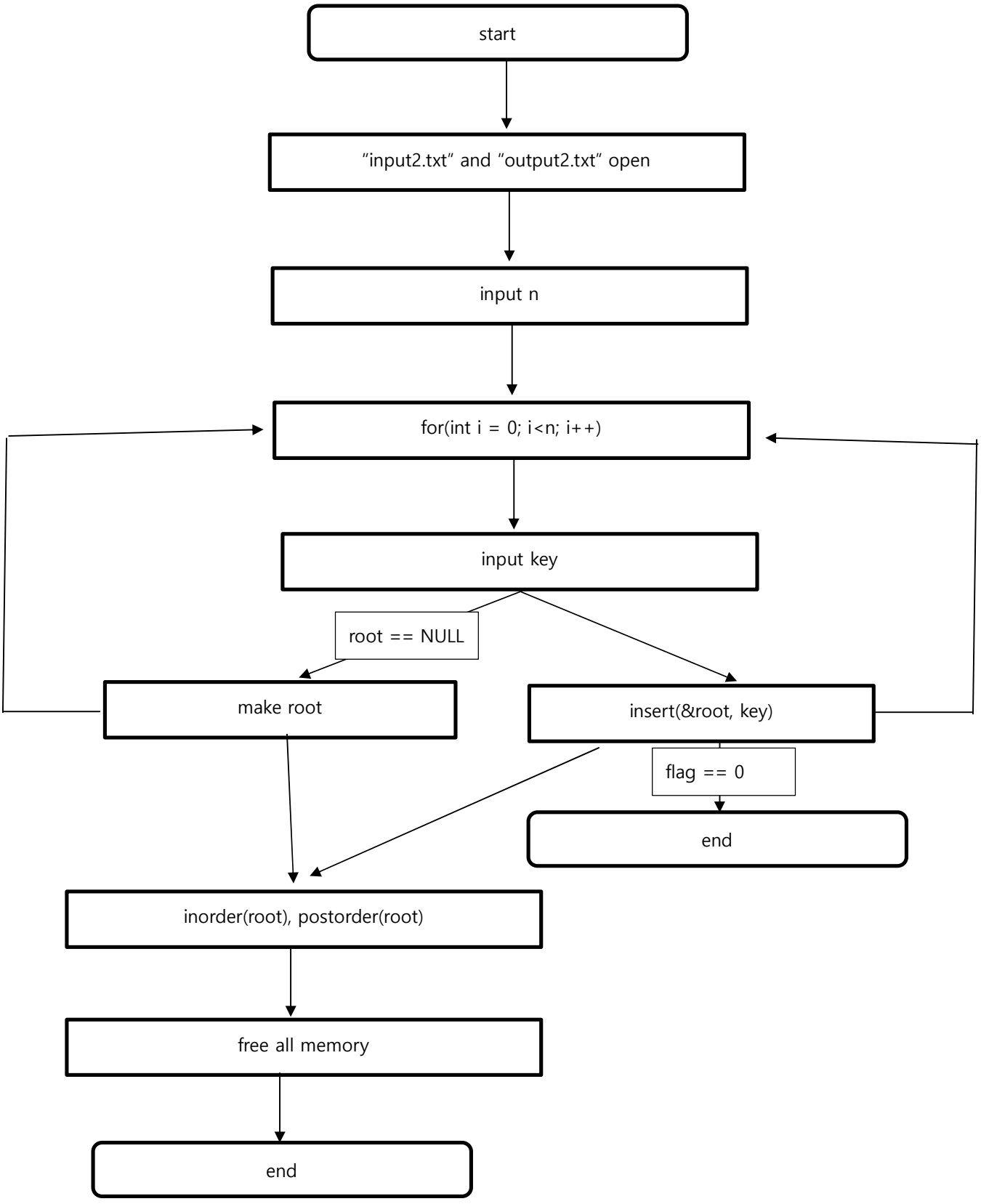
insert_loca()의 return value를 저장하고 newnode의 parent를 last로 한다. 그 후 last의 leftChild가 NULL인 경우 leftChild에, 그 외에는 rightChild에 newnode를 대입한 후 exchange_up()으로 sorting 해준다. 새로 node가 추가된 경우 size도 1 증가시키고, 1을 return한다. delete의 경우에는 exchange_up과 반대로 exchange_down이다. void exchange_down(treePointer node)는 max를 node로 저장해놓고, left와 right child와 비교하여 큰 것을 max로 한다. 그 후 max와 node의 key value를 swap하고 node를 max로 하는 과정을 max와 node가 같아질 때까지 반복한다. int delete()는 우선 size가 0인 경우에는 -1을 return하여 비어있음을 나타낸다. 아닌 경우에는 우선 max에 root->key를 저장해두고, 각각 size가 1인 경우와 나머지를 구분한다. size가 1일 때는 root node를 제거하므로 root를 free 해주고, size를 1 감소, max를 return한다. 그 외에는 last에 insert_loca()를 통해 마지막 node를 저장하고, root->key에 last->key를 넣는다. 그 후 last가 parent의 left, right 중 연결된 쪽을 찾아 NULL로 바꾸고 last를 free 처리한 후 size 1 감소, 그 후에 exchange_down(root)로 sort한 후, max를 return하며 종료한다.

1번 문제에 대한 code의 test 결과는 위와 같이 확인할 수 있다. input 예시에서 기능이 잘 작동함을 잘 확인할 수 있어 다른 case에 대한 test는 필요하지 않을 것으로 확인했다. 구현에 있어 아쉬운 점으로는 마지막 node를 찾는 부분에 대한 구현에 있어서 binary를 이용한 방법 외의 방법을 찾지 못한 점에 대한 아쉬운 점이 있었다.

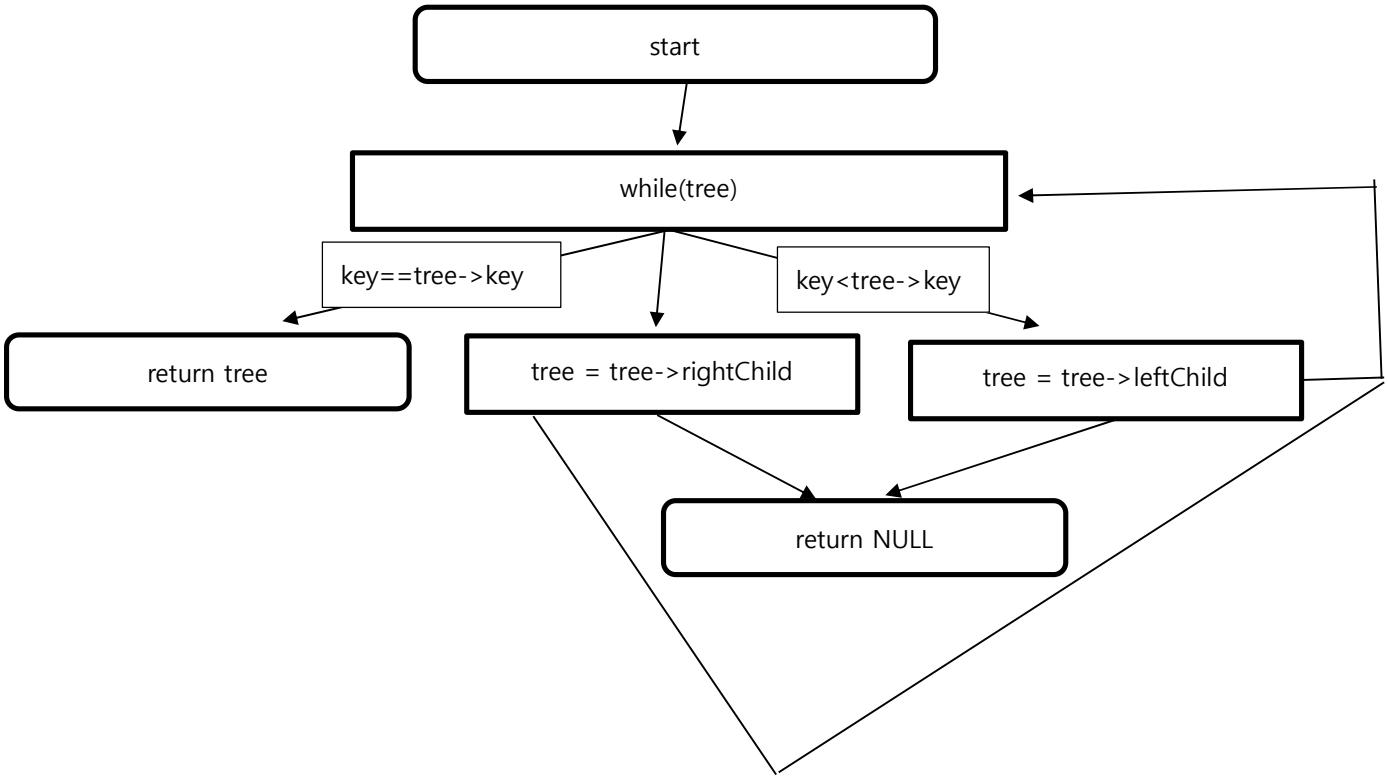
2번 문제

<flow chart>

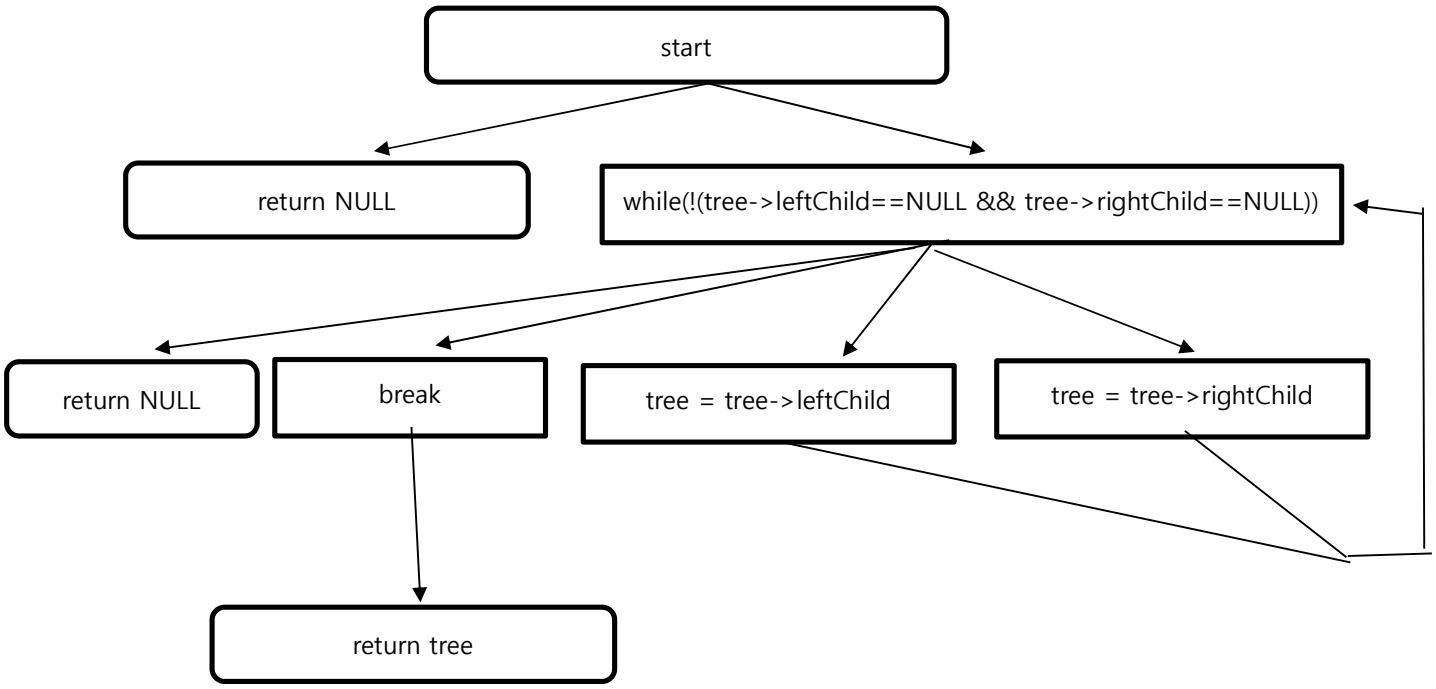
-int main()



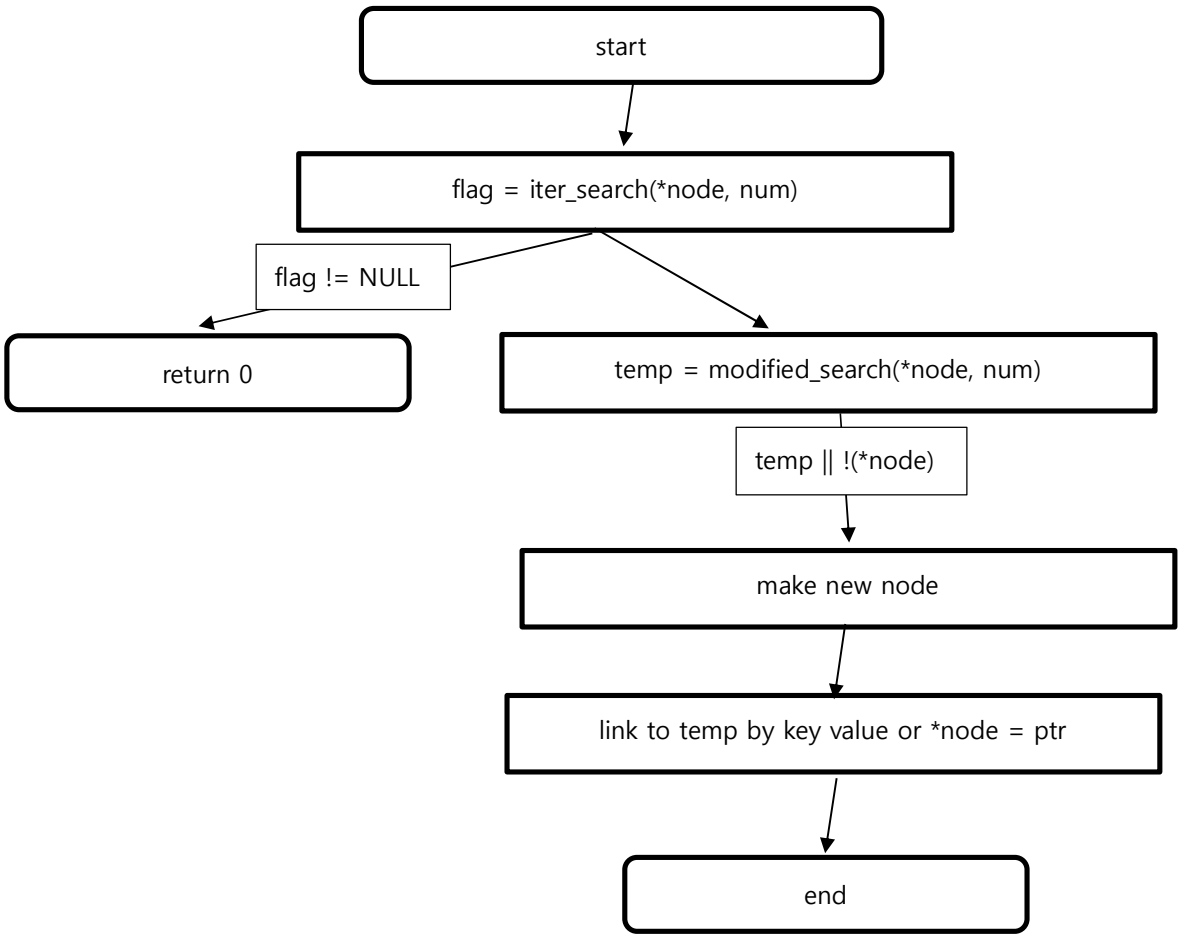
-treePointer iter_search(treePointer tree, int key)



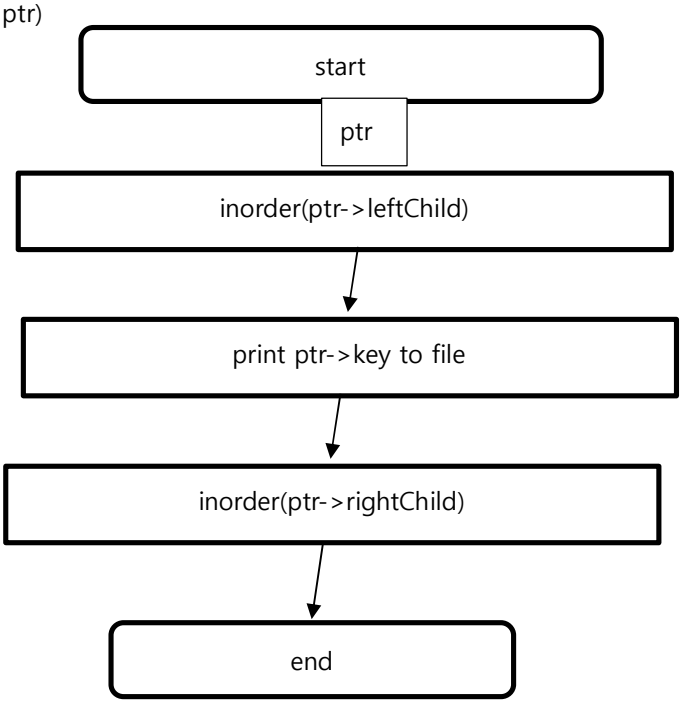
-treePointer modified_search(treePointer tree, int key)



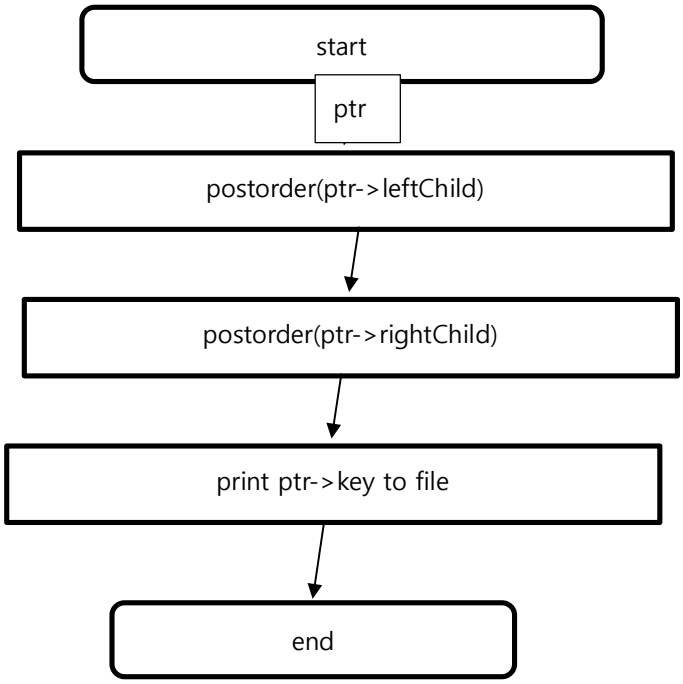
-int insert(treePointer* node, int num)



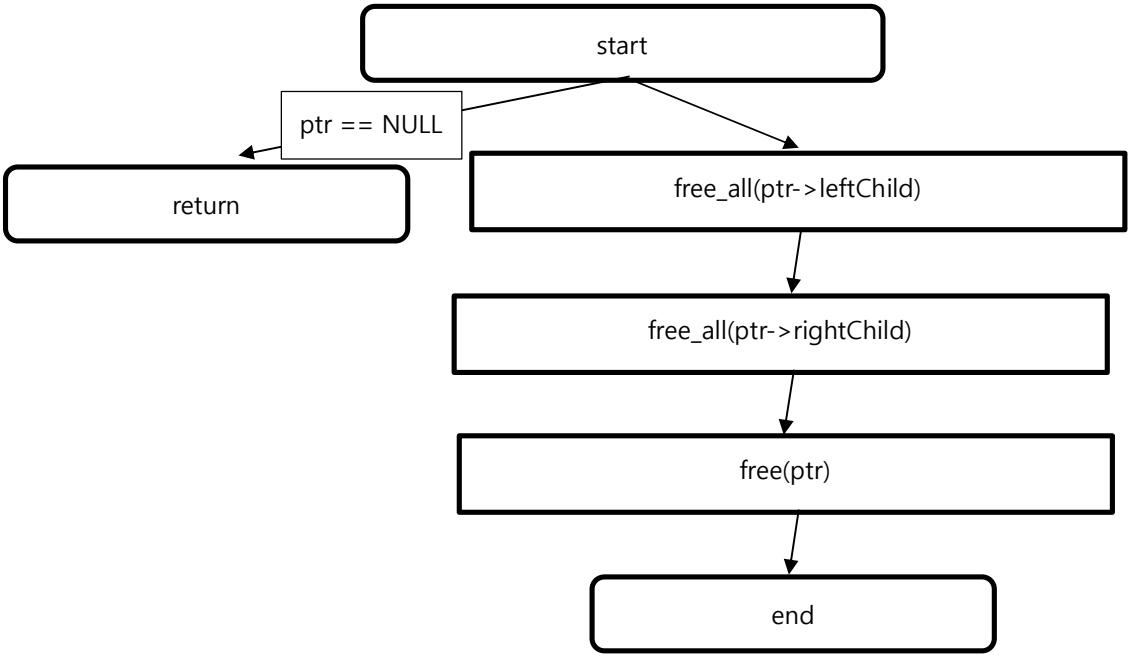
-void inorder(treePointer ptr)



-void postorder(treePointer ptr)



-void free_all(treePointer ptr)



<구현 설명>

tree의 struct는 1번 문제와 유사하지만 parent variable만 넣지 않고 구현하였다. main function에서는 global variable로 선언된 fp1, fp2에 각각 input2.txt, output2.txt를 open한다. 그 후 fp1으로부터 int type의 n을 input받는다. 그 후 for loop로 n번 반복하면서 int type의 key를 input받는데, 이 때 global variable인 root가 NULL인 경우 root node를 allocate해주고, root의 key에 input받은 key를 넣는다. 아닌 경우에는 int type의 flag에 insert(&root, key)의 return value를 넣어주며 tree에 node를 추가한다. flag가 0인, 즉 같은 value가 이미 tree에 있는 경우에는 문제에 주어진 양식대로 file에 print하고 function을 종료한다. 그 후 주어진 양식에 맞도록 file에 print해주며 inorder(root)와 postorder(root)를 call하여 file에 result가 저장되도록 하고 fp1, fp2를 닫은 후에 free_all(root)로 모든 allocate된 memory를 해제한다. treePointer iter_search(treePointer tree, int key)는 key와 같은 value를 가진 node가 있는지 search하는 function이다. while loop를 통해서 tree가 NULL이 아닌 동안 tree->key가 key와 같으면 tree를 return하고 큰 경우는 leftChild로, 아닌 경우는 rightChild로 이동하며 search한다. while loop 탈출 시 NULL을 return한다. treePointer modified_search(treePointer tree, int key)는 insert될 node를 return하는 function이다. tree가 NULL인 경우에는 NULL을 return한다. 아닌 경우에 tree의 leftChild와 rightChild가 동시에 NULL이 아닌 동안 반복하면서 tree가 NULL일 때는 멈추고, 그 동안은 key와의 compare를 통해 leftChild 또는 rightChild로 이동한다. while loop 종료 후 tree를 return한다. int insert(treePointer* node, int num)은 node를 insert하는 function이다. 우선 flag에 iter_search의 return value를 저장하고 flag가 NULL이 아닌 경우, 즉 같은 value가 있을 때 0을 return한다. 그 후 temp에는 modified_search의 return value를 저장하고, temp가 NULL이 아니거나 *node가 NULL인 경우, 새로운 node를 allocate한다. 이를 *node가 NULL이 아닌 경우에는 temp에 link하고, 아닌 경우는 *node를 새로운 node로 한다. void inorder(treePointer ptr)과 void postorder(treePointer ptr)은 강의자료와 똑같이 구현하였고, file에 print를 위해 fprintf로 fp2에 print하도록 변경하였다. 마지막으로 void free_all(treePointer ptr)은 모든 memory 해제를 위햔 function이다. ptr이 NULL인 경우 return으로 function을 종료하고 recursive call을 통해 leftChild와 rightChild로 이동한다. 그 후 free(ptr)로 memory를 해제한다.

다음으로는 testcase에 대해서 test를 진행했다. 우선 중복 value가 주어지지 않은 case에 대한 test이다.

```
cse20211522@cspro:~/HW5$ cat input2.txt
6
30 5 2 40 35 80


cse20211522@cspro:~/HW5$ cat output2.txt
Inorder: 2 5 30 35 40 80
Postorder: 2 5 35 80 40 30
cse20211522@cspro:~/HW5$
```

위와 같이 문제없이 result가 나오는 것을 확인할 수 있다. 다음으로는 중복 value에 대한 test 이다.

```
cse20211522@cspro:~/HW5$ ./test2
cse20211522@cspro:~/HW5$ cat input2.txt
6
30 5 5 40 35 80
cse20211522@cspro:~/HW5$ cat output2.txt
cannot construct BST
cse20211522@cspro:~/HW5$
```

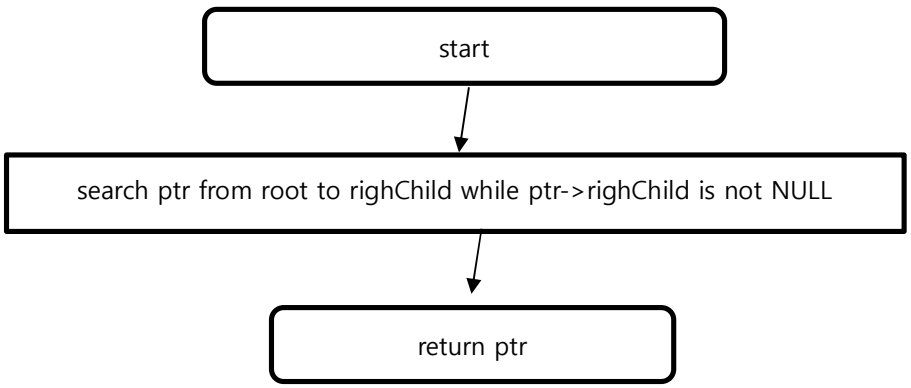
양식에 맞게 잘 print되는 것을 확인할 수 있다. 구현하면서 아쉬운 점으로는 동일한 value를 filtering하는 과정에서 modified_search 하나로 처리하는 것이 아닌 추가로 다른 search를 구현하여 사용한 점에서 아쉬운 점이 있다. 한 function으로 동일 value filtering과 insert할 pointer return을 모두 구현했다면 좀 더 나은 code가 되었을 것이라 생각한다.

3번 문제

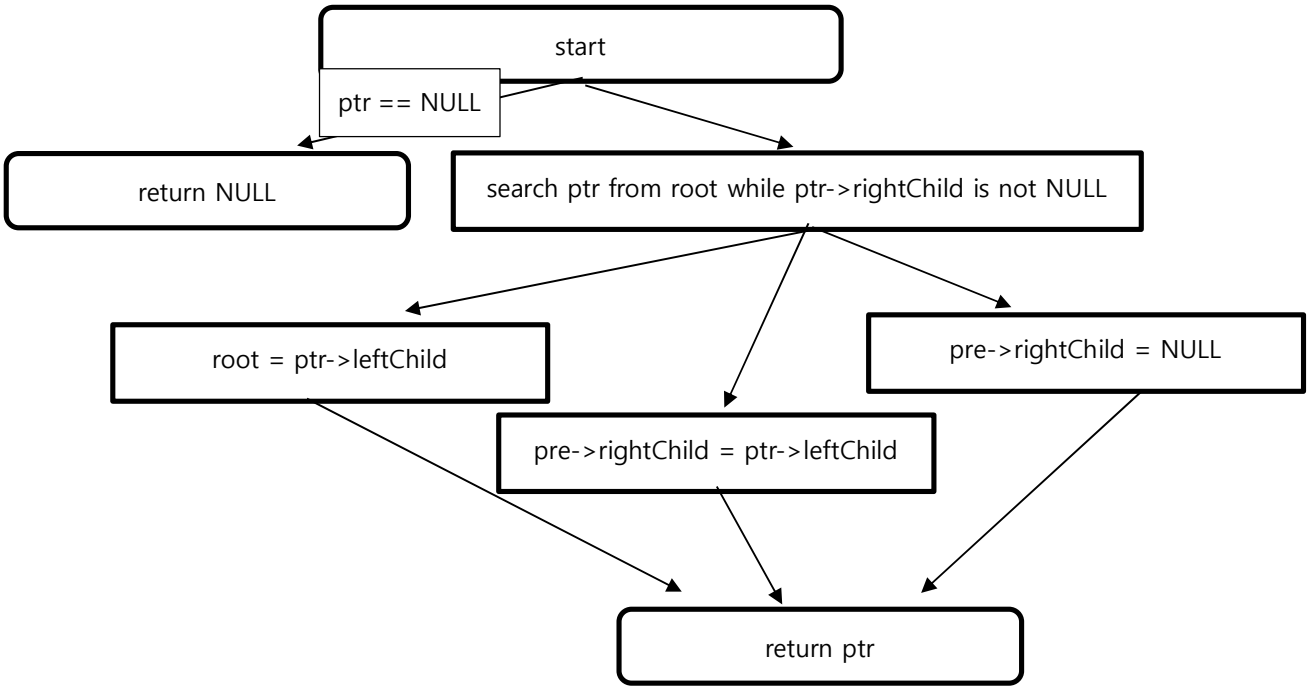
<flow chart>

iter_search, modified_search, insert, free_all은 2번 문제와 동일하다.

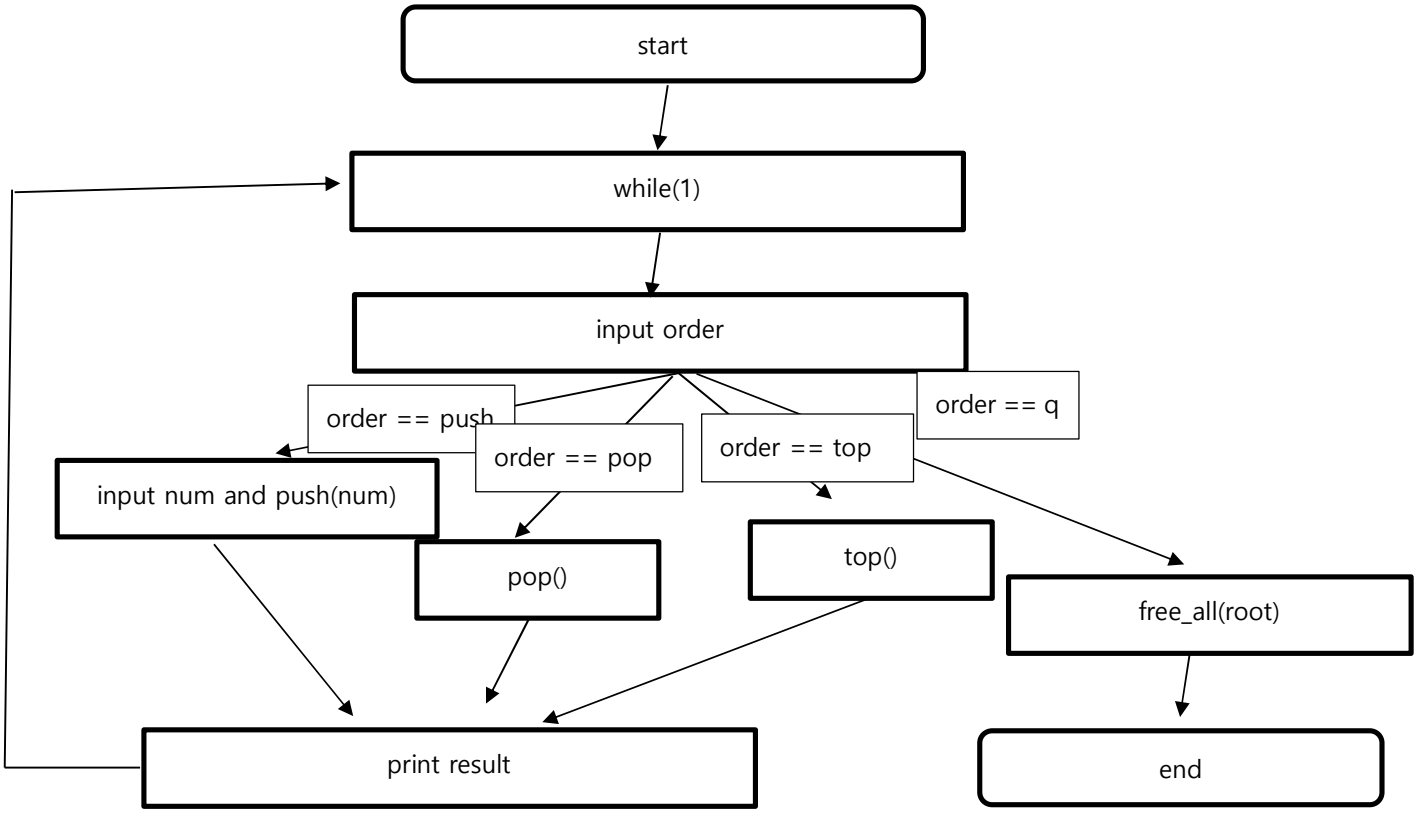
-treePointer top()



-treePointer pop()



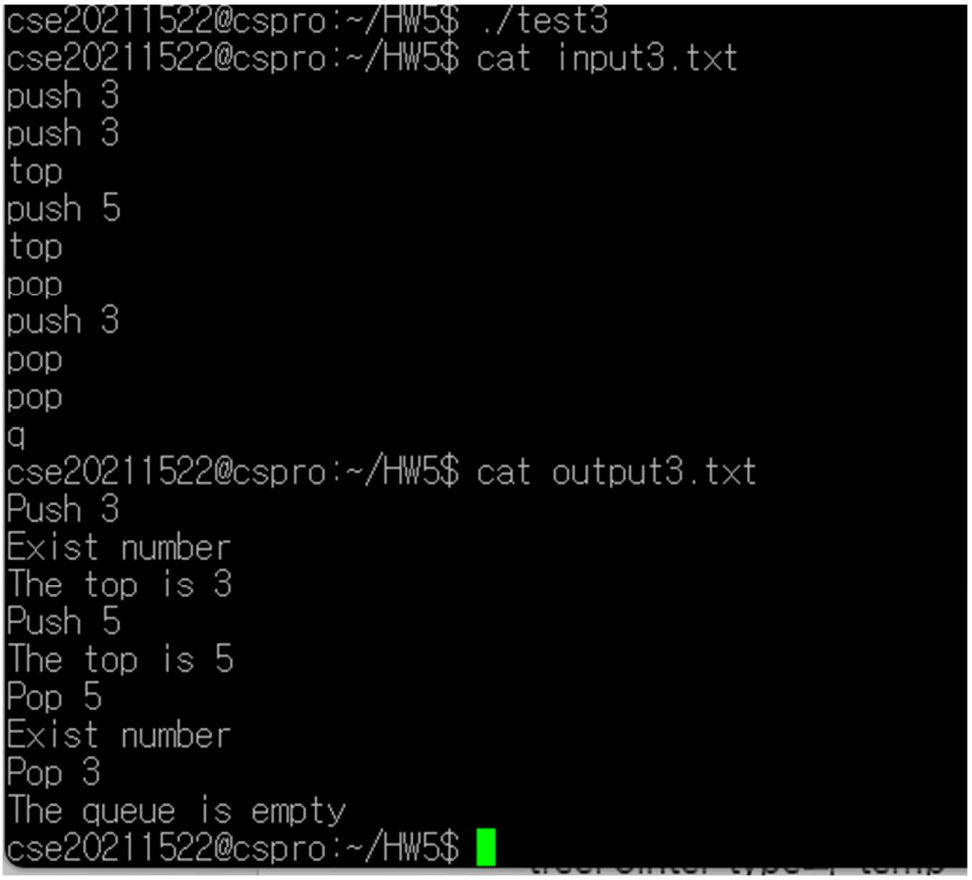
-int main()



<구현 설명>

2번 문제와 동일하게 binary search tree를 사용하므로 node의 struct나 insert function을 구현하는 방식, memory를 모두 해제하는 free_all function의 구현은 동일하게 진행했다. 2번 문제와 다른 function은 top과 pop function이 있다. treePointer top()은 binary search tree로 구현된 priority queue에서 가장 큰 node를 return한다. binary search tree에서 가장 큰 node는 rightChild 방향으로 끝까지 이동했을 때의 node이다. 따라서 while loop를 통해서 ptr->rightChild가 NULL이 아닌 동안 rightChild로 이동한 ptr을 return하여 top을 return한다. treePointer pop()은 priority queue에서 가장 큰 node를 return하며 제거하는 function이다. 우선 variable ptr에 root를 넣어 search를 위한 variable을 declare했다. ptr이 NULL인 경우에는 빈 tree이므로 NULL을 return하며 종료한다. 그 뒤에는 이전 parent node를 저장할 pre variable을 declare하고, top function과 똑같이 rightChild로의 끝 node로 이동한다. 우선 이동 후에도 ptr이 root인 경우는 root가 가장 큰 node일 때이다. 이 경우에는 root를 leftChild로 이동시킨 후 ptr을 return한다. 그 외에는 이동 후 ptr->leftChild가 NULL이 아닌 경우이다. 이 때는 pre의 key value보다는 ptr->leftChild의 value가 더 크기 때문에 pre->righChild에 ptr->leftChild를 link하고 ptr을 return한다. 이 두 경우를 제외하면 pre->rightChild를 NULL로 바꾸어 link를 끊어 주고 ptr을 return하며 function을 종료한다. main function에서는 input3.txt와 output3.txt를 fopen으로 열어 각각 fp1, fp2에 저장한다. 그 후 while loop로 char order[5]에 계속 input받으면서 strcmp로 compare하여 case를 나누었다. order가 "push"인 경우, int type의 num을 추가로 input 받는다. res variable에 insert(&root, num)을 call하며 return value를 저장한다. res가 0인 경우는 이미 num이 존재했을 때이므로 양식에 맞게 print하고, 아닌 경우는 "Push (num)"을 print한다. 다음으로는 order가 "pop"일 때이다. 이 경우에는 treePointer type의 temp에 pop()을 call하며 return value를 저장한다. temp가 NULL일 때는 tree가 비어있는 것이므로 양식에 맞게 print하고, 아닌 경우는 "Pop (temp->key)"를 print한 후 temp를 free 처리한다. order가 "top"일 경우, temp에 top()의 return value를 저장하고 temp가 NULL일 때는 pop과 같이 처리한다. 아닌 경우에는 "The top is (temp->key)"를 print한다. 그리고 order가 "q"일 때는 flag를 1로 바꿔줘 while loop를 escape할 수 있도록 한다. while loop 종료 후에는 fclose로 file을 닫고, free_all(root)로 모든 memory를 free 처리한다.

다음으로는 testcase에 대해서 test를 해보았다.



위와 같이 testcase에 대해서 print가 잘 나오는 것을 확인할 수 있었다. 이 test에 대해서도 기능에 대한 확인이 되었다고 판단하여 다른 test에 대해서는 크게 확인할 필요가 없다고 판단하였다. 구현에 있어 아쉬운 점으로는 2번 문제와 유사한 binary search tree 구현을 하였으므로 아쉬운 점은 공유된다. 그 외에 있어서는 문제가 없는 것으로 판단하였다.