HW3 결과보고서

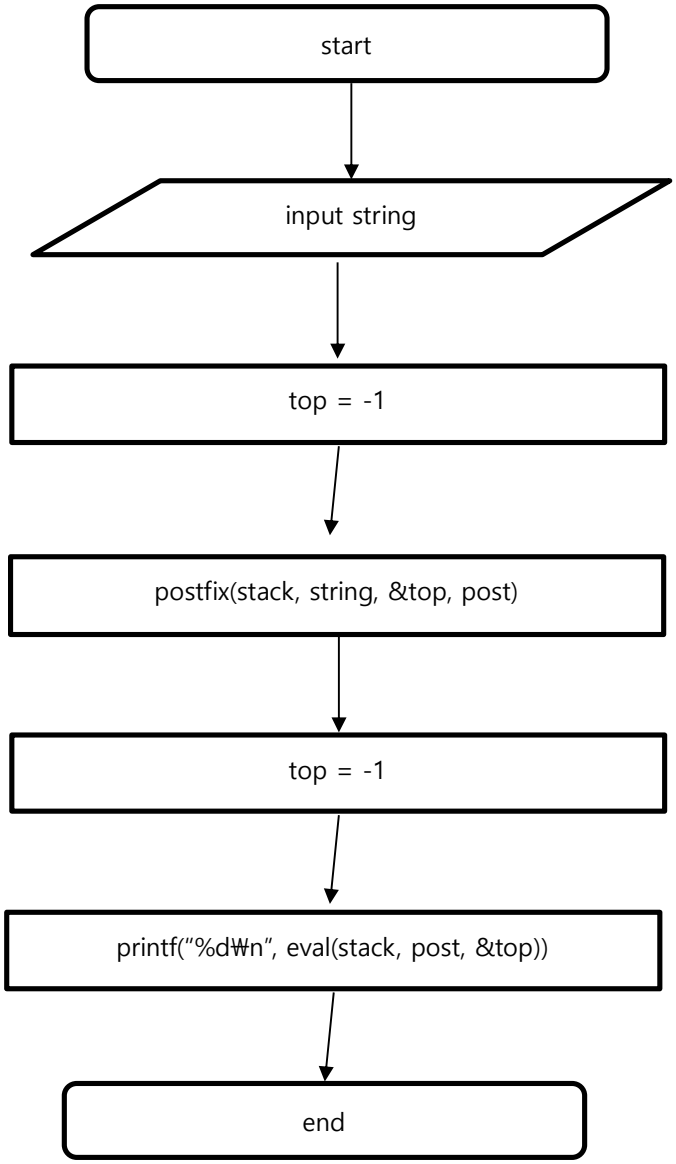
20211522

김정환

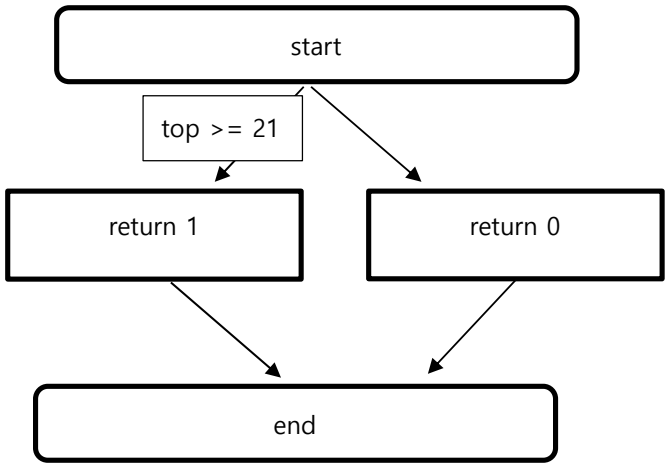
1번 문제

<flow chart>

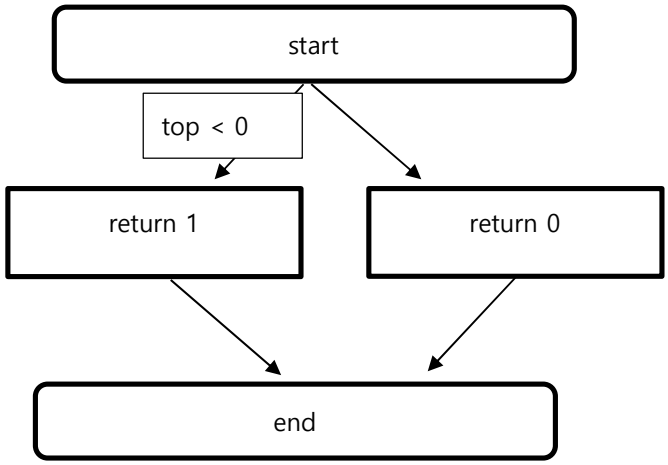
-int main()



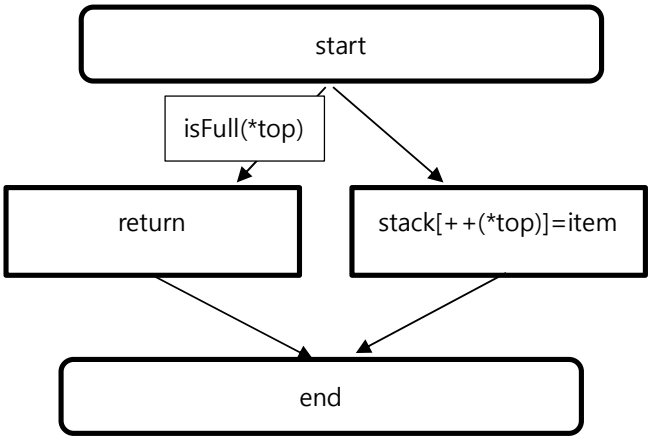
-int isFull(int top)



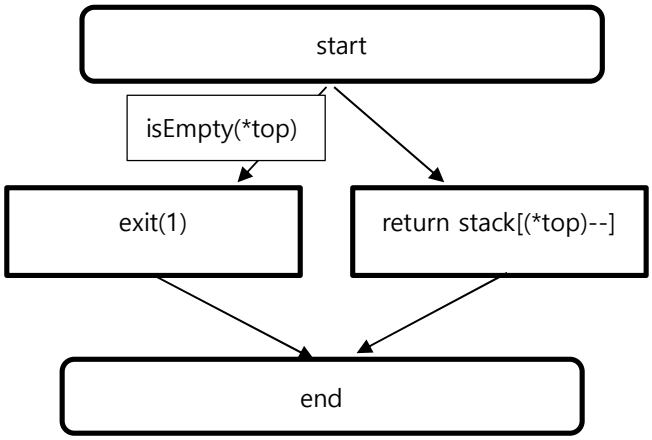
-int isEmpty(int top)



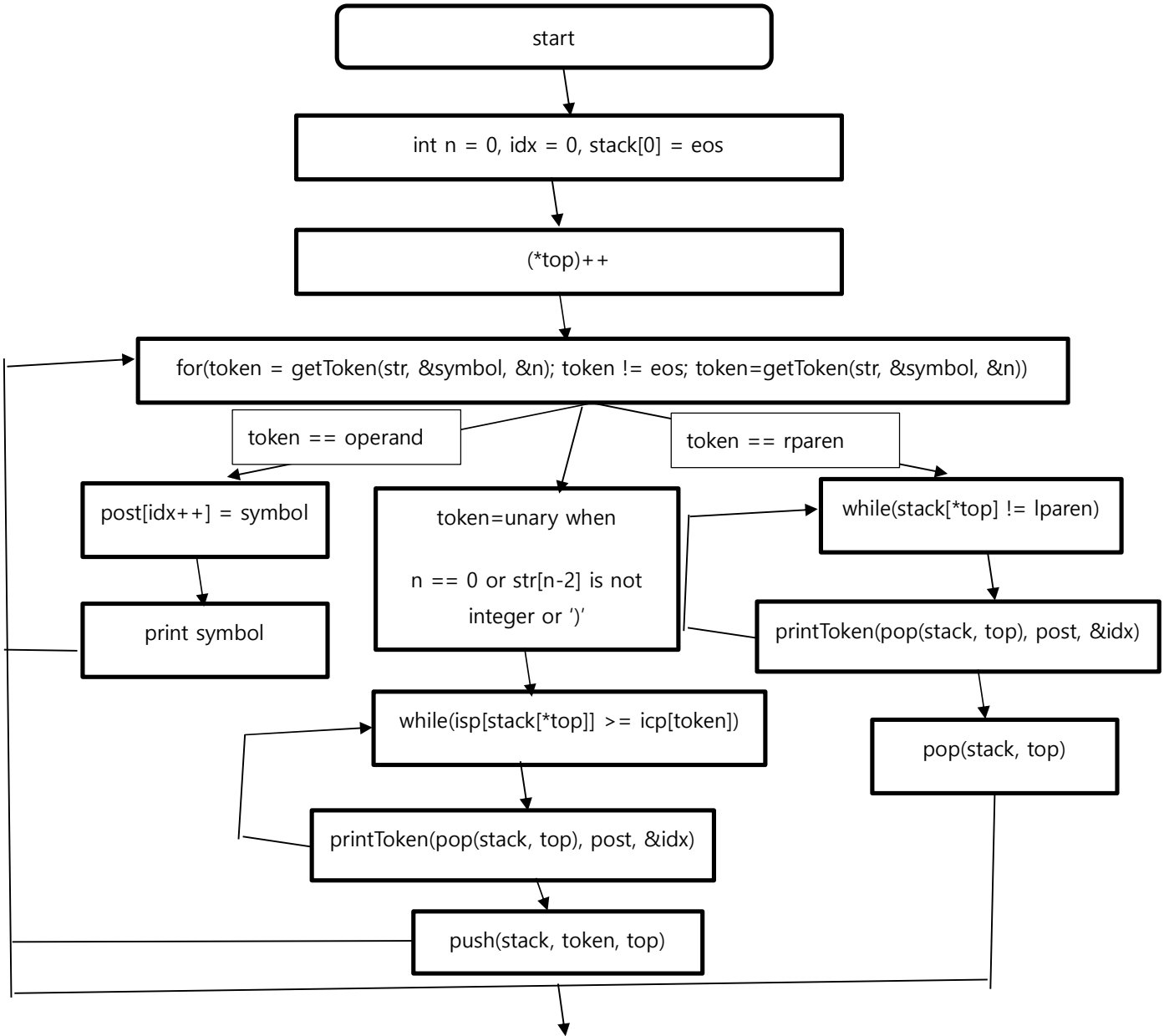
-void push(int* stack, int item, int* top)

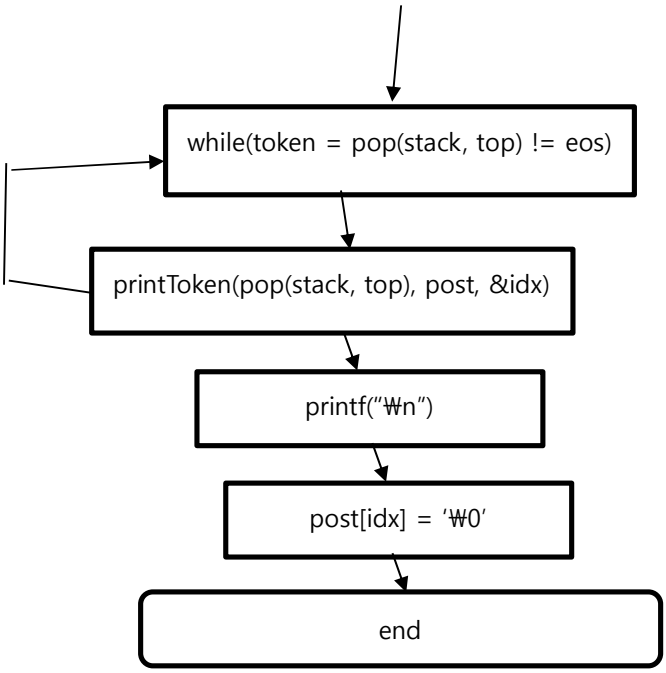


-int pop(int* stack, int* top)

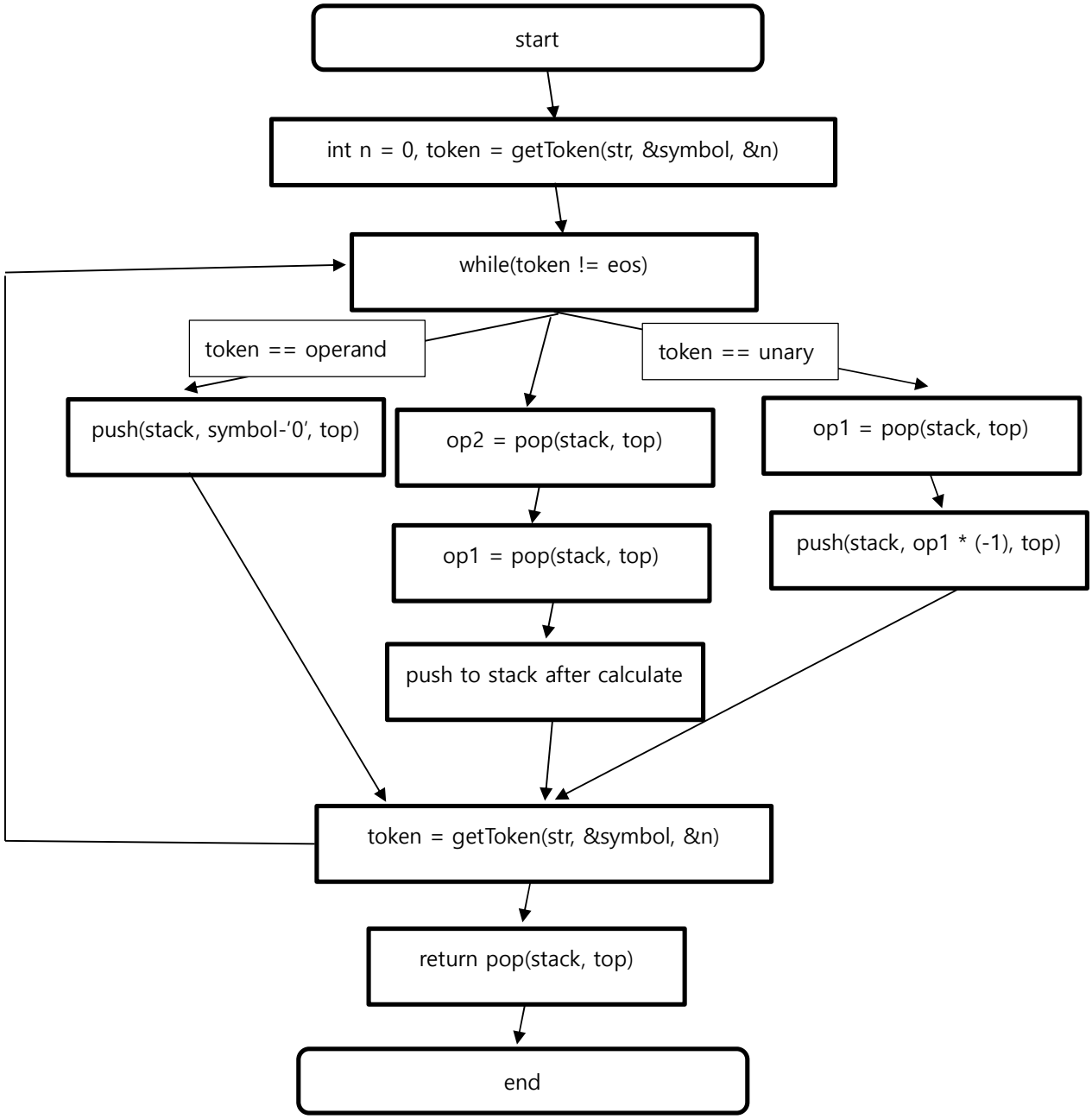


-void postfix(int* stack, char* str, int* top, char* post)

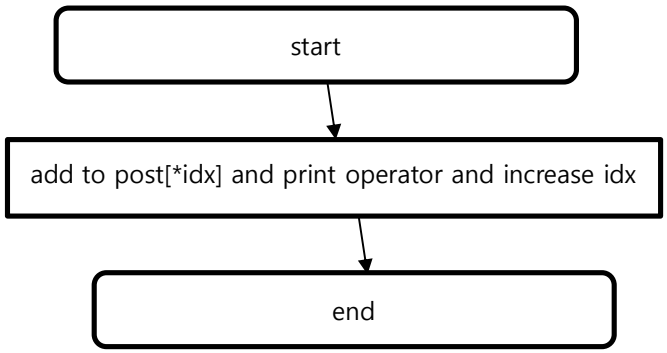




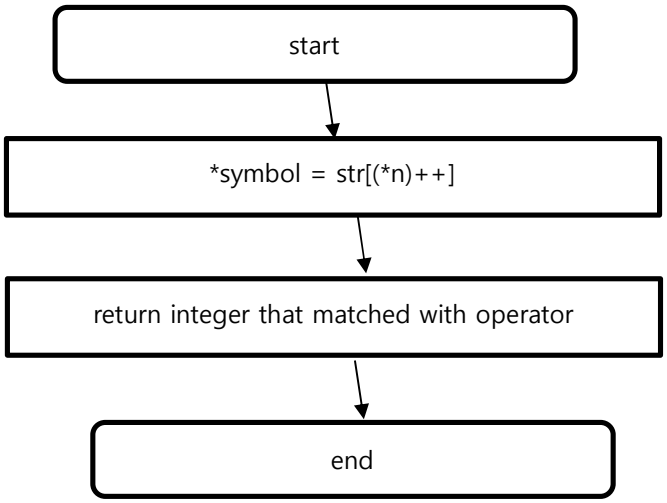
-int eval(int* stack, char* str, int* top)



-void printToken(precedence symbol, char* post, int* idx)



-precedence getToken(char* str, char* symbol, int* n)



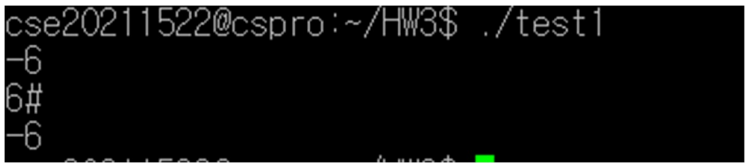
<구현 설명>

우선 infix expression을 postfix로 바꾸는 function과 postfix expression으로 계산하는 function은 강의자료를 참고하여 구현하였다. 우선 stack을 이용하기 위해서 isFull(), isEmpty(), push(), pop()을 정의했다. int isFull(int top)은 top integer value를 argument로 받아서 stack의 max size를 넘는지 판단한다. 문제에서 주어진 string의 최대 길이가 20글자이므로 stack의 max size를 21로 정의했고, 따라서 isFull function은 top이 21 이상인지 판단하여 21 이상이면 1, 아니면 0을 return한다. int isEmpty(int top)은 top이 음수, 즉 stack에 아무 것도 없을 때 true이므로 1을 return하고, 아닐 때는 0을 return 한다. void push(int* stack, int item, int* top)은 문제의 제한 조건에서 global variable의 사용이 제한되어, stack의 array와 top을 pointer로 argument를 넣어주고 넣어줄 value를 item으로 하여 argument를 받았다. 그리고 stack이 다 찼을 때는 추가하면 안 되므로 isFull이 true일

때는 return으로 종료하도록 하고, 아닐 때는 stack에 top을 1 증가시켜 item을 넣어준다. int pop(int* stack, int* top)은 stack이 비어있을 때는 return할 값이 없으므로 exit(1)로 종료되도록 하였고, 그 외에는 stack[*top]의 값을 return하면서 top의 값이 1 감소하도록 구현하였다. 다음으로 operator에 대한 구분을 위해서 precedence type을 정의하였다. lparen, rparen, .. 등의 operator를 순서대로 나타낼 수 있도록 enum을 사용하였다. precedence getToken(char* str, char* symbol, int* n)은 infix string으로부터 character를 얻었을 때 operator와 operand를 구분하도록 하는 function이다. switch문으로 symbol에 저장한 character를 경우에 따라 나누고 precedence type을 정의한 대로 경우에 맞는 value를 return할 수 있도록 구현하였다. 강의자료에 있는 부분에서 unary를 추가했기에 해당 부분도 역시 추가했다. void printToken(precedence symbol, char* post, int* idx)는 token으로 구분된 operator를 post array에 저장하고, operator를 출력하는 function이다. idx argument를 통해서 post에 이어서 저장할 수 있도록 하였고, switch문으로 각각의 operator들을 구분하였다. void postfix(int* stack, char* str, int* top, char* post) function은 infix expression을 postfix로 바꿔주는 function이다. local variable로 declare한 variable들은 char type의 symbol, int type의 n, idx가 있고, precedence type의 token이 있다. function의 argument들은 stack은 stack 이용을 위한 int type의 array이고, str은 infix expression이 저장되어 있는 string, top은 stack의 top을 저장한 integer value를 가리키는 pointer이고, char* post는 postfix expression을 저장할 string이다. stack의 첫 item은 eos를 저장하고, top이 0이 되도록 1 증가시킨다. 그 후 for loop를 통해서 token에 str로부터 getToken() function을 이용하여 eos가 아닌 동안 value를 받아온다. token이 operand이면 post에 symbol을 그대로 저장하고, symbol을 print한다. post에 저장할 때는 token이 rparen일 때는 lparen이 나올 때까지 while loop를 통해서 stack에서 pop한 값들을 printToken을 통해서 print 및 post에 저장한다. 그 후 lparen을 pop으로 stack에서 제거한다. 그 외의 경우에는 function 내에서 declare해둔 isp와 icp array에 저장된 priority에 따라 operator를 stack에 저장하거나 pop한다. 그 전에 minus 기호에 대해서는 unary와 구분을 해둬야 하는데, unary인 경우는 -가 가장 처음에 있거나 str 내에서 – 기호 바로 전이 integer가 아니거나)가 아니면 unary이므로 해당 경우에는 token을 unary로 바꿔준다. 그 후 stack 내에 있는 operator와 priority를 비교하여 stack 내에서 더 높은 operator들은 printToken으로 print 및 저장하면서 pop으로 제거한다. 그 후 stack에 token을 push해준다. for loop가 끝난 후에는 남은 operator들을 모두 printToken으로 print 및 저장해주고, output 쪽에서는 ₩n을 print하여 줄바꿈을 해주고 post에는 마지막에 ₩0을 추가하여 string의 끝을 표시할 수 있도록 한다. postfix function을 call한 이후로 같은 stack array를 eval function에서도 사용하므로 top을 -1로 다시 initialize해준다. int eval(int* stack, char* str, int* top)은 str에 받은 postfix expression을 계산한 결과를 return하는 function이다. token을 받는 것은 postfix function과 유사하고, 차이점으로는 op1, op2 local variable이 있는 것이다. token이 operand인 경우에는 '0'을 빼줘 integer value로 만들고 stack에 push한다. token이 unary인 경우에는 1개의 operand만 pop해주고 -1을 곱해 push한다. 그 외에는 op1, op2에 pop한 value를 저

장하고 token에 따라 맞는 operator에 따른 계산결과를 push해준다. token이 eos가 될 때까지 반복하면 while loop가 종료되고 마지막으로 남은 value를 pop해주면서 return해준 value가 계산 결과이다. main function에서는 infix expression을 char string[21]에 input 받고, int stack[21]과 int top = -1을 declare한다. 또한 postfix expression을 저장할 char post[21]을 declare하고, postfix function을 call한 이후, top을 -1로 다시 initialize해주고, eval function의 value를 출력해준다.

다음으로는 구현한 code에 대한 결과 확인이다.



처음에는 unary의 확인을 위해 -6을 input으로 주었다. postfix로 6#으로 잘 변환된 것을 확인할 수 있고, 결과도 -6으로 잘 나온 것을 확인할 수 있다.

다음으로는 괄호와 unary, - 등이 섞인 expression을 input으로 주었고 minus와 unary를 잘 구분하여 변환하는 것을 확인할 수 있고, 결과도 맞게 계산된 것을 확인할 수 있다.

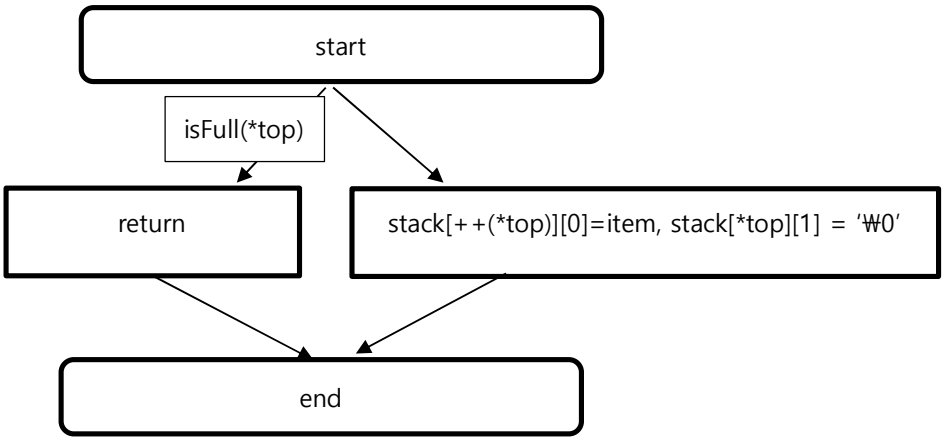
다음으로는 다른 operator들도 잘 작동하는지 확인하였고, *나 + 역시 문제없이 돌아가고, 계산결과도 올바르게 계산된 것을 확인 가능하다.

2번 문제

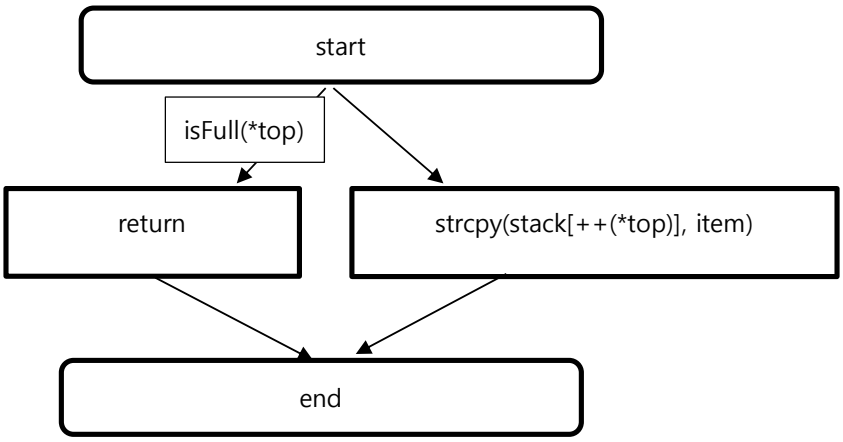
<flow chart>

-int isFull(int top), int isEmpty(int top), void push(int* stack, int item, int* top), int pop(int* stack, int* top), precedence getToken(char* str, char* symbol, int* n)은 1번 문제와 같은 구현이므로 생략하였다.

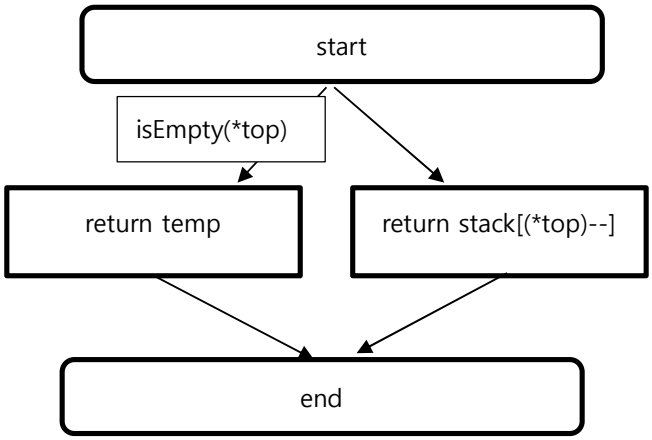
-void char_push(char stack[][21], char item, int* top)



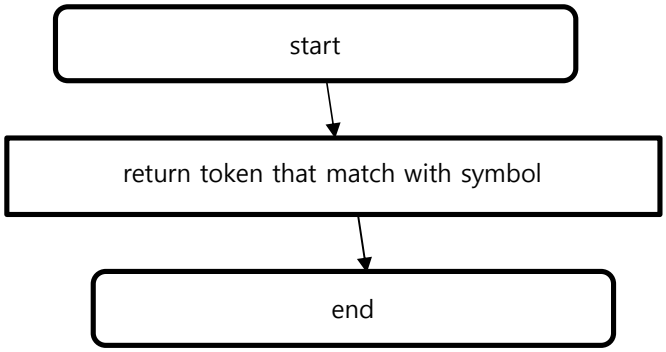
-void str_push(char stack[][21], char* item, int* top)



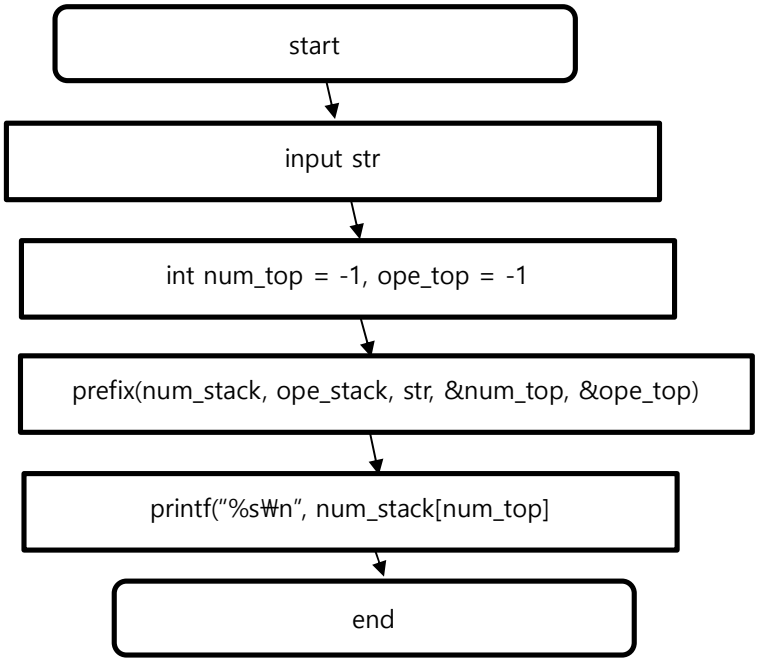
-char* str_pop(char stack[][21], int* top)



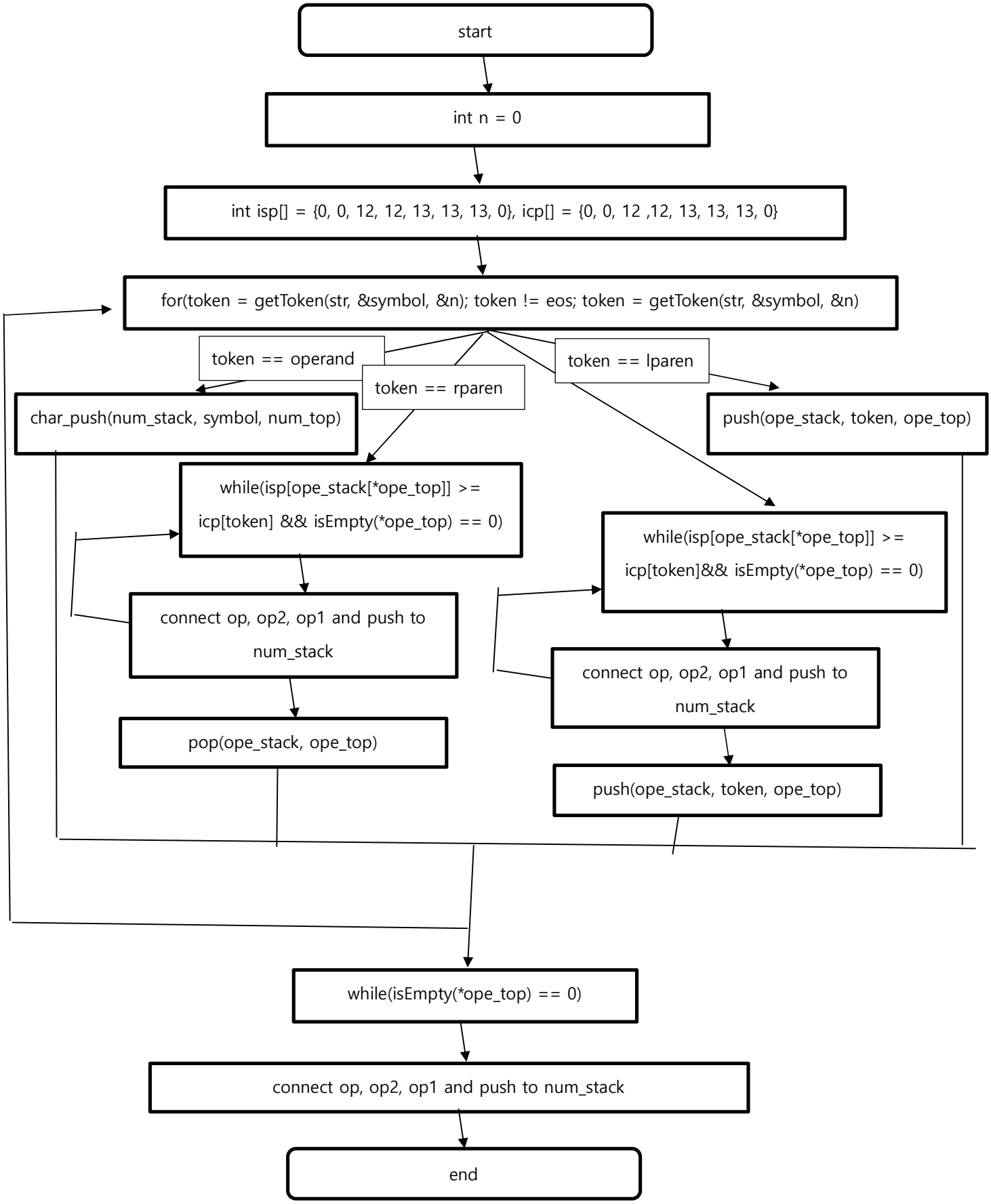
-char printToken(precedence symbol)



-int main()



-void prefix(char num_stack[][21], int* ope_stack, char* str, int* num_top, int* ope_top)



<구현설명>

prefix를 postfix를 사용하지 않고 나타내기 위해서 stack을 2개 이용하였다. operand를 저장할 stack은 string을 저장할 수 있는 stack으로 구현하였고, operator를 저장할 stack은 integer를 저장할 수 있는 stack으로 구현했다. operand를 저장하는 stack은 character 하나만 item에 들어오는 경우와 string이 들어오는 경우 두 가지로 나누어서 push를 구현했다. void char_push(char stack[][21], char item, int* top)은 character 하나만 들어오는 경우이다. isFull이 true인 경우는 똑같이 종료하지만, item을 넣는 과정에서 차이가 있다. stack[++(*top)][0]에 주어진 character인 item을 넣고 string으로 인식하도록 stack[*top][1]에 '₩0'을 넣었다. void str_push(char stack[][21], char* item, int* top)은 string이 들어온 경우이다. 이 경우에도 isFull이 true인 경우에는 똑같이 종료하고, item의 string을 넣어줄 때는 strcpy를 이용하여 stack[++(*top)]에 복사해준다. pop의 경우에는 character 하나만 들어왔을 때도 string으로 인식할 수 있도록 하였으므로 둘 다 공통으로 구현했다. char* str_pop(char stack[][21], int* top)은 isEmpty가 true일 때는 temp에 저장해둔 '₩0'을 return 하고, 아닌 경우에는 stack[(*top)--]를 return 한다. 이외에 stack에서 공통적으로 사용되는 부분은 1번 문제와 같은 방법으로 구현했고, printToken만 사용을 다르게 하기 위하여 다르게 구현하였다. char printToken(precedence symbol)은 switch문으로 구분하고, 각각의 token value에 따라 맞는 operator의 character를 return하도록 구현했다. void prefix(char num_stack[][21], int* ope_stack, char* str, int* num_top, int* ope_top)은 str argument로 받아온 infix expression을 prefix로 바꿔주는 function이다. 각각의 argument는 num_stack은 operand를 넣어줄 stack을 저장하는 array, ope_stack은 operator를 저장할 stack의 array, num_top과 ope_top은 각각 stack의 top을 가리킬 variable이다. int isp, icp array로 operator의 priority를 나타내고, searching을 위한 int n = 0을 declare했다. 1번 문제와 같이 for loop로 str의 끝까지 searching하면서 변환한다. 여기서 token이 operand인 경우에 char_push를 통해서 symbol을 num_stack에 넣어준다. token이 lparen인 경우에는 push로 ope_stack에 넣어준다. token이 rparen인 경우에는 while loop를 통해서 lparen이 나올 때까지 ope_stack에서 pop해주는데, 하나하나 pop할 때마다 operand와의 계산을 나타내야 한다. 여기서 string을 이어붙이는 방식으로 구현했다. op1과 op2를 str_pop을 통해서 나온 string을 strcpy로 복사받아온다. op에는 printToken으로 pop된 operator를 저장하고, 이들을 char temp[21]에 op, op2, op1의 순서로 연결하고, 이를 str_push로 num_stack에 넣어주는 방식을 구현했다. 여기서 temp[0] = op;temp[1] = '₩0'으로 strcat을 사용 시에 string으로 인식될 수 있도록 op를 넣을 때 사용하였다. while loop가 끝나면 pop으로 lparen을 제거한다. 이외의 경우에는 while loop를 icp[token]<=isp[ope_stack[*ope_stack]]일 동안 돌리고 위와 같이 string을 연결하는 방식으로 num_stack에 push해준다. 그리고 while loop가 끝난 후 push로 token을 ope_stack에 넣어준다. for loop가 끝난 후에 남은 ope_stack의 operator들을 모두 위처럼 string 연결 방식으로 넣어주면 function이 종료된다. 이렇게 모든 loop가 끝났을 때 num_stack의 top에 있는 string이 prefix expression이 된다. main function에서는 char str[21]을 declare하고 여기에 infix expression을 input

받는다. char num_stack[21][21]과 int ope_stack[21], int num_top = -1, int ope_top = -1을 declare 해주었는데 모두 array의 size가 21인 이유는 input으로 들어올 infix expression의 max size가 20이기 때문이다. prefix function에 argument들을 type에 맞게 전달하고 call한 후에 num_stack[num_top]에 저장된 string을 print해주면 prefix expression으로 된 식을 print할 수 있다.

다음으로는 제대로 동작하는지에 대한 test이다.

```
cse20211522@cspro:~/HW3$ ./test2
3*8+7/1
+*38/71
```

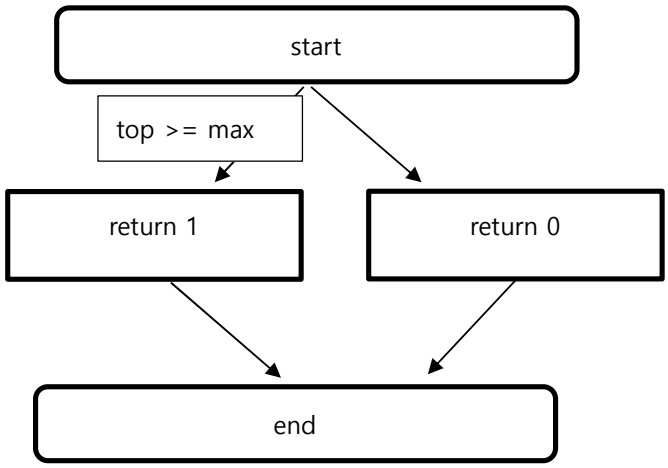
```
cse20211522@cspro:~/HW3$ ./test2
(4-9/5)*(4/1-2)
*-4/95-/412
```

pdf에 주어진 예시들을 test해본 결과 prefix로 잘 변환된 결과를 확인할 수 있다.

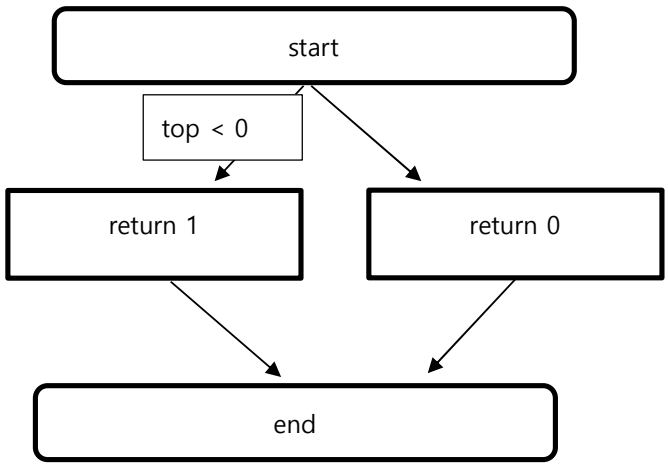
3번 문제

<flow chart>

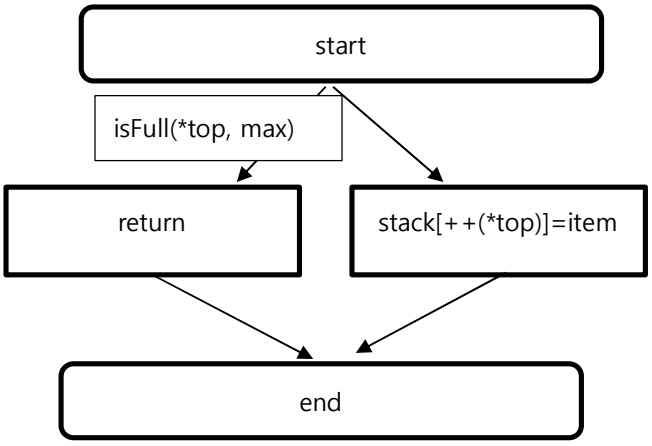
-int isFull(int top, int max)



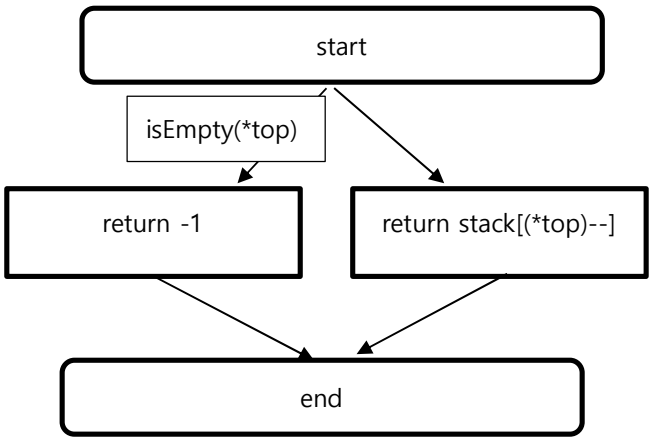
-int isEmpty(int top)



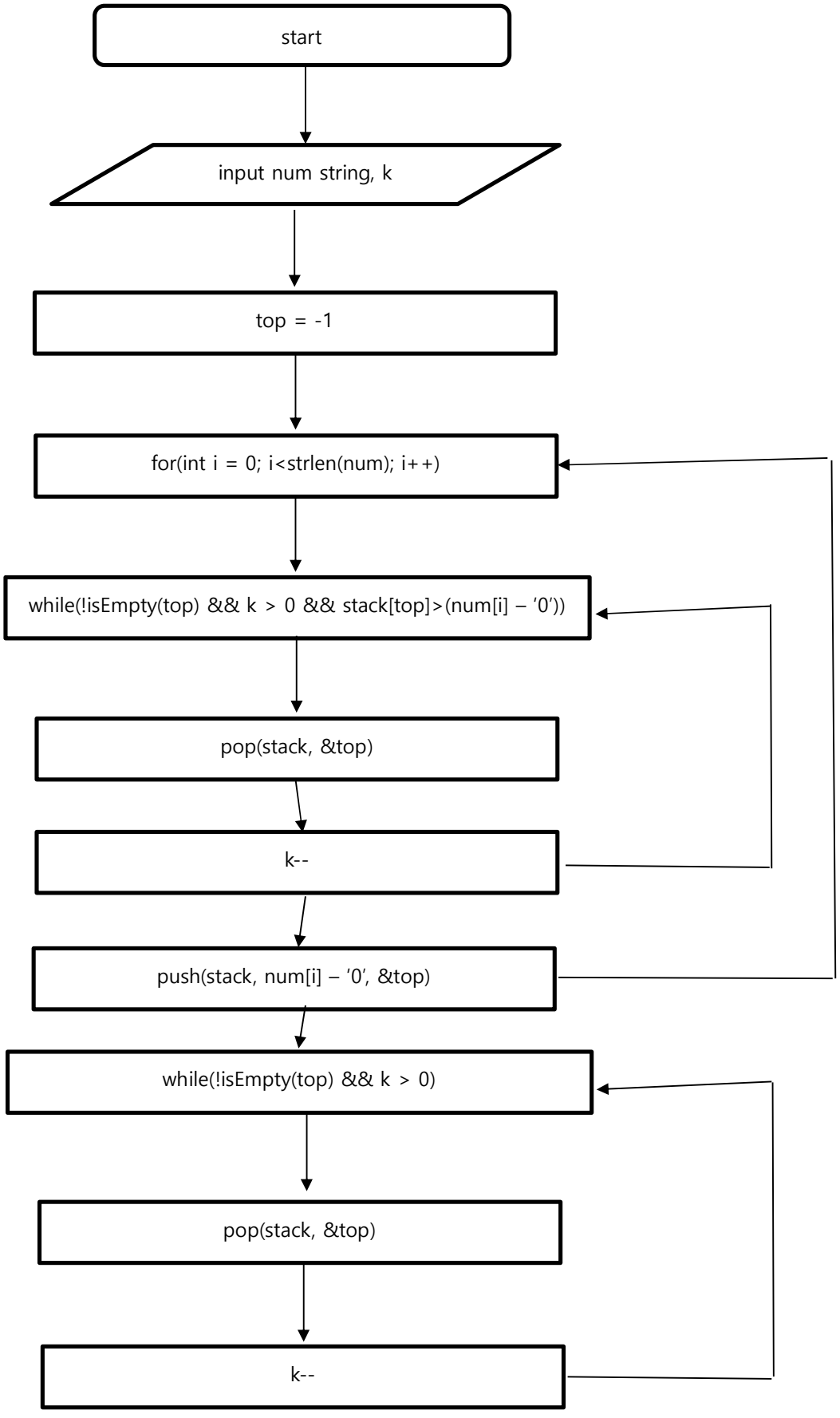
-void push(int* stack, int item, int* top, int max)

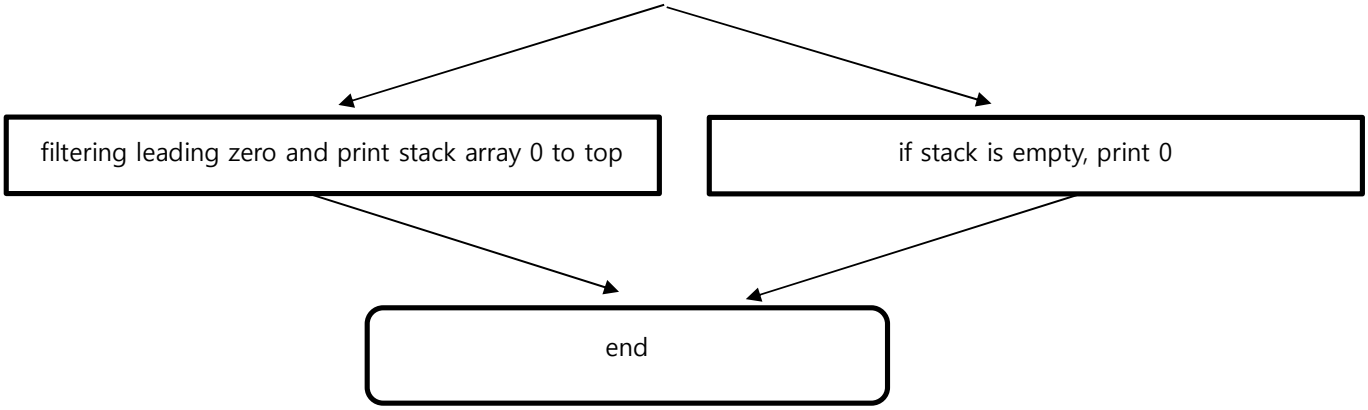


-int pop(int* stack, int* top)



-int main()


start
input num string, k
top = -1
for(int i = 0; i<strlen(num); i++)
while(!isEmpty(top) && k > 0 && stack[top]>(num[i] – '0'))
pop(stack, &top)
k--
push(stack, num[i] – '0', &top)
while(!isEmpty(top) && k > 0)
pop(stack, &top)
k--



<구현 설명>

Stack의 구현에 있어서는 int형의 item을 넣는 stack의 형태로 구현했다. stack의 구현에서는 int isFull(int top, int max)에는 int max argument를 추가했다. 주어진 num에 따라 stack의 max_size가 달라지므로 argument로 전달받아 비교하도록 구현했다. int isEmpty(int top)은 1, 2번 문제와 차이 없이 top이 음수인 경우 1을 return하도록 구현했다. void push(int* stack, int item, int* top, int max)는 isFull function이 max를 전달받아 비교하도록 바뀌었으므로 argument에 max_size를 나타내는 int max argument를 추가하여 isFull을 call할 때 전달하도록 구현했다. int pop(int* stack, int* top)은 1, 2번 문제와 차이 없이 구현하였는데 이번에는 stack에 음수가 들어가지 않으므로 isEmpty가 true이면 -1을 return하도록 구현했다. 이 외의 경우에는 stack[*top]을 return하면서 *top을 1 줄인다. 문제를 보면 num이 최대 몇 자리를 가진 수가 input될 지 주어져 있지 않다. 따라서 main function에서 input 구현시에 새로 dynamic allocating을 할 수 있도록 구현하였다. num을 input받을 때는 string으로 받을 수 있도록 num variable은 char* type으로 declare하였다. 또한 int type으로 max_len을 나타내서 비교할 max_len을 저장해놓도록 했다. 또한 int type으로 idx를 declare하여 한 character씩 input받을 때 input 받은 개수를 표시하도록 하고, '₩n'이 들어올 때까지 while loop를 통해 input받는다. 이때 '₩n'은 '₩0'으로 바꿔줘 string의 끝을 표시하고 break를 통해 while loop를 빠져나간다. idx + 1이 max_len 이상일 때는 array size가 더 필요한 것이므로 temp로 dynamic allocating을 통해 allocating한 array에 이미 받은 input을 저장해놓고, num에는 max_len이 하나 늘어난 size로 새로 allocating을 해준 후, num에 다시 data를 옮긴다. 그래도 새로 allocating하는 case는 오래 걸리므로 되도록 실행되지 않게 max_len은 1000001로 initialize하였다. 그 후 scanf를 통해서 int type의 k를 input 받고, stack array는 int type array가 되도록 num의 size + 1의 size로 alllocating한다. 문제를 해결하기 위한 부분에 있어서는 num array에 저장된 숫자들을 자릿수가 높은 순서대로 stack에 push한다. 이 때 stack에 push하기 전에 이미 stack에 들어있는 숫자들 중 top부터 이번에 넣을 숫자보다 큰 경우에 대해서 pop해준다. 여기서 k개의 숫자만을 제거하므로 pop은 k번만 실행되도록 한다. 이 부분에 대한 구현은 for loop로 num array의 0부터 strlen(num)-1까지 search하도록 했고, push를 실행하기 전, while loop로 pop을 condition에 따라 실행되도록 구현했다. while loop는 stack이 비어있지 않고(!isEmpty(top)), k가 0보다 크고,

stack[top] > num[i] – '0'일 동안 반복된다. pop을 실행한 이후에는 k의 value를 1 감소시켜 pop은 k번 실행되도록 했다. while loop 이후에는 push를 통해서 stack에 num[i] – '0'을 넣어준다. for loop가 끝난 후에도 k가 0이 되지 않았을 수 있으므로 while loop를 isEmpty가 false이고, k > 0인 동안 pop을 call하며 pop이 call될 때마다 k를 1씩 감소하여 남은 개수만큼 자릿수를 제거할 수 있도록 하였다. 그 후에 stack에 남아있는 숫자가 정답이므로 이를 print하도록 한다. stack에는 높은 자리수부터 넣었으므로 stack array에서 0부터 top까지 index의 순서대로 출력해야 한다. 그 전에 stack이 비어 있는 case에는 0이 print되어야 하므로 isEmpty가 true이면 0을 print 후 program을 종료하도록 하였다. 이외에 출력하는 과정에서 맨 앞자리가 0인 경우에 0을 앞 자리에 붙여서 print하는 문제가 있어 해당 문제를 해결하기 위해 int type의 fir variable을 이용하였다. fir의 value를 0으로 initialize해두고, for loop를 통해서 stack array를 순서대로 print할 때 fir이 0이고 stack[i]가 0, 그리고 top이 0이 아닌 case에는 continue로 넘어가도록 하였다. top이 0일 때는 결과가 0인 것이므로 print해야 하고, 나머지의 경우에는 첫 번째가 0이 되면 안 되기 때문이다. 또한 이외의 case에는 fir를 1로 바꿔 첫 번째 숫자 print 후에는 나머지가 print되도록 하였다. print 후 allocating해준 num과 stack을 free로 해제하였다.

다음은 구현한 program에 대한 test이다. 우선 pdf에 있는 예시들에 대해서 test해보았다.

위의 3가지 case에서 첫 번째 자리가 0이 되는 case에 대해서도 0을 제외하고 잘 출력하는 것을 확인할 수 있고, 2개를 다 제거하자 0으로 잘 print하는 것을 확인할 수 있다. 또한 case 1에서도 예시와 같은 숫자로 답이 나왔다. 이 외에 추가적인 test로는 자릿수보다 k가 큰 case에 대해서 test해보았다.

위와 같이 더 많은 자릿수를 뺐을 때는 0으로 결과가 잘 print되는 것을 볼 수 있다.

숫자의 자릿수가 증가하는 순서로만 있을 때는 뒤부터 잘 pop하는 것을 확인했다.