자료구조론

Infix/Postfix conversion/evaluation

u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW8$gcc hw8_2016025041.c

u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW8$./a.out

postfix_to_infix expression: 23+2*4-

eval: 6

infix_to_postfix expression:(3+5)*4)

eval: 32

u@hataeseong-ui-MacBook-Pro:~/Desktop/2017_CSE2010_2016025041/HW8$

- 실행 결과 일치

```
int prec(char op) {
   switch(op){
              case '(': case ')': return 0;
              case '+': case '-': return 1;
              case '*': case '/': return 2;
              }
              return -1;
}
```

- 연산자 우선순위 지정

```
void infix_to_postfix(char infix[],char postfix[])
{
   StackType s;
   char x,token;
   int i,j;   //i-index of infix, j-index of postfix
   init(&s);
   j=0;

   for(i=0;infix[i]!='\0';i++)
   {
```

```
        token=infix[i];
        if(isalnum(token))
           postfix[j++]=token;
        else
           if(token=='(')
             push(&s,'(');
        else
           if(token==')')
              while((x=pop(&s))!='('
                   postfix[j++]=x;
              else
              {
                 while(!is_empty(&s) && (prec(token) <= prec(peek(&s))))
                 {
                    x=pop(&s);
                    postfix[j++]=x;
                 }
                 push(&s,token);
              }
     }

     while(!is_empty(&s))
     {
        x=pop(&s);
        postfix[j++]=x;
     }

     postfix[j]='\0';
              printf("postfix_to_infix expression: %s", postfix);
              printf("\n");
  }
```

- 숫자는 바로 출력(postfix에 넣음), 연산자는 스택에 넣고 다른 연산자가 들어오면 우선순위를 비교해 스택의 탑에 있는 연산자가 높을경우 탑에 있는 연산자를 pop해서 출력, 입력된 연산자가 탑보다 높을경우 스택에 push. 단, 왼쪽 괄호는 무조건 push, 오른쪽 괄호가 나오면 왼쪽괄호가 나올때까지 모두 pop.

```
  char* postfix_to_infix(char expression[])
  {
     printf("\ninfix_to_postfix expression:");
     int count, length;
     char element, operator;
     StackType s;
```

```
	length = strlen(expression);

	for(count = 0; count < MAX_STACK_SIZE; count++){
		s.stack[count] = 0;
	}

	printf("(%c", expression[0]);

	for(count = 1; count < length; count++){
		if(expression[count] == '+' || expression[count] == '-' || expression[count] == '*' ||
expression[count] == '/'){
			element = pop(&s);
			operator = expression[count];
			printf("%c%c)", operator, element);
		}
		else
			push(&s, expression[count]);
	}
	printf("\n");

	return expression;
}
```

- 첫번째 숫자는 바로 출력하고 나머지 숫자는 전부 스택에 push. 연산자를 만날 경우 연산자를 출력하고 스택에서 pop 실행.

```
int postfixEval(char exp[])
{
  /* fill in the blank */
  StackType s;
  init(&s);
  int tmp;

  for(int i = 0; exp[i] != '\0'; i++){
		if(isdigit(exp[i])){
			push(&s, exp[i] - '0');
		}
		else{
			switch (exp[i]) {
				case '+':
					tmp = s.stack[s.top - 1] + s.stack[s.top];
					pop(&s);
```

```
                    pop(&s);
                    push(&s, tmp);
                    break;
                case '-':
                    tmp = s.stack[s.top - 1] - s.stack[s.top];
                    pop(&s);
                    pop(&s);
                    push(&s, tmp);
                    break;
                case '*':
                    tmp = s.stack[s.top - 1] * s.stack[s.top];
                    pop(&s);
                    pop(&s);
                    push(&s, tmp);
                    break;
                case '/':
                    tmp = s.stack[s.top - 1] + s.stack[s.top];
                    pop(&s);
                    pop(&s);
                    push(&s, tmp);
                    break;
                default:
                    break;
            }
        }
    }
    return peek(&s);
}
```

- 숫자가 나올경우 모두 스택에 push. 연산자를 만날경우 stack에서 top과 top - 1에 있는 숫자를 pop 해서 연산자로 계산한후 그 결과를 다시 stack에 push. 위와 같은 과정 반복 후 마지막에 스택에 남은 결과 출력