1. 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define NUMBER  256
#define PLUS    257
#define STAR    258
#define LPAREN  259
#define RPAREN  260
#define END     261
#define EXPRESSION 0
#define TERM    1
#define FACTOR  2
#define ACC     999

typedef enum {INT_VAL, ADD, MUL} NODE_NAME;
typedef struct node {
   NODE_NAME name;
   int val;
   struct node *llink, *rlink;
} NODE;
NODE *value[1000];
NODE *makenode(NODE_NAME name, int v, NODE *p, NODE *q);

int action[12][6] = {
   {5,0,0,4,0,0}, {0,6,0,0,0,ACC}, {0,-2,7,0,-2,-2},
   {0,-4,-4,0,-4,-4}, {5,0,0,4,0,0}, {0,-6,-6,0,-6,-6},
   {5,0,0,4,0,0}, {5,0,0,4,0,0}, {0,6,0,0,11,0},
   {0,-1,7,0,-1,-1}, {0,-3,-3,0,-3,-3}, {0,-5,-5,0,-5,-5}
};

int go_to[12][3] = {
   {1,2,3}, {0,0,0}, {0,0,0}, {0,0,0}, {8,2,3}, {0,0,0},
   {0,9,3}, {0,0,10}, {0,0,0}, {0,0,0}, {0,0,0}, {0,0,0}
};

int prod_left[7] = {0, EXPRESSION, EXPRESSION, TERM, TERM, FACTOR, FACTOR};
int prod_length[7] = {0,3,1,3,1,3,1};

int stack[1000]; int top = -1; int sym;
char yytext[32];
int yylval;

int yyparse();
```

```
void push(int i);
void shift(int i);
void reduce(int i);
void yyerror();
int yylex();
void lex_error();

void printTab(int i) {
   for(int x = 0; x < i; x++) {
       printf("\t");
   }
}

void printNode(NODE* node, int depth) {
   printTab(depth*2);
   switch(node->name) {
   case 0:
       printf("Node name:  INT_VAL\n");
       printTab(depth*2);
       printf("Node value: %d\n", node->val);
       break;
   case 1:
       printf("Node name:  ADD\n");
       printTab(depth*2);
       printf("Node value: -\n");
       break;
   case 2:
       printf("Node name:  MUL\n");
       printTab(depth*2);
       printf("Node value: -\n");
       break;
   default:
       break;
   }
   printf("\n");
}

void printTree(NODE* node, int depth) {
   if(node != NULL) {
       printTree(node->llink, depth+1);
       printNode(node, depth);
       printTree(node->rlink, depth+1);
   }
}
```

```
void main() {
   yyparse();
   printTree(value[top], 0);
}

NODE *makenode(NODE_NAME name, int v, NODE *p, NODE *q) {
   NODE *n;
   n = (NODE *)malloc(sizeof(NODE) * 4);
   n->name = name;
   n->val = v;
   n->llink = p;
   n->rlink = q;

   return n;
}

int yyparse() {
   int i = 0;

   stack[++top] = 0;
   sym = yylex();

   do {
       i = action[stack[top]][sym-256];

       if(i == ACC) ;
       else if (i > 0) shift(i);
       else if (i < 0) reduce(-i);
       else yyerror();
   } while(i != ACC);

}

void push(int i) {
   stack[++top] = i;
}

void shift(int i) {
   push(i);
   value[top] = makenode(INT_VAL, yylval, NULL, NULL);
   sym = yylex();
}

void reduce(int i) {
   int old_top;
```

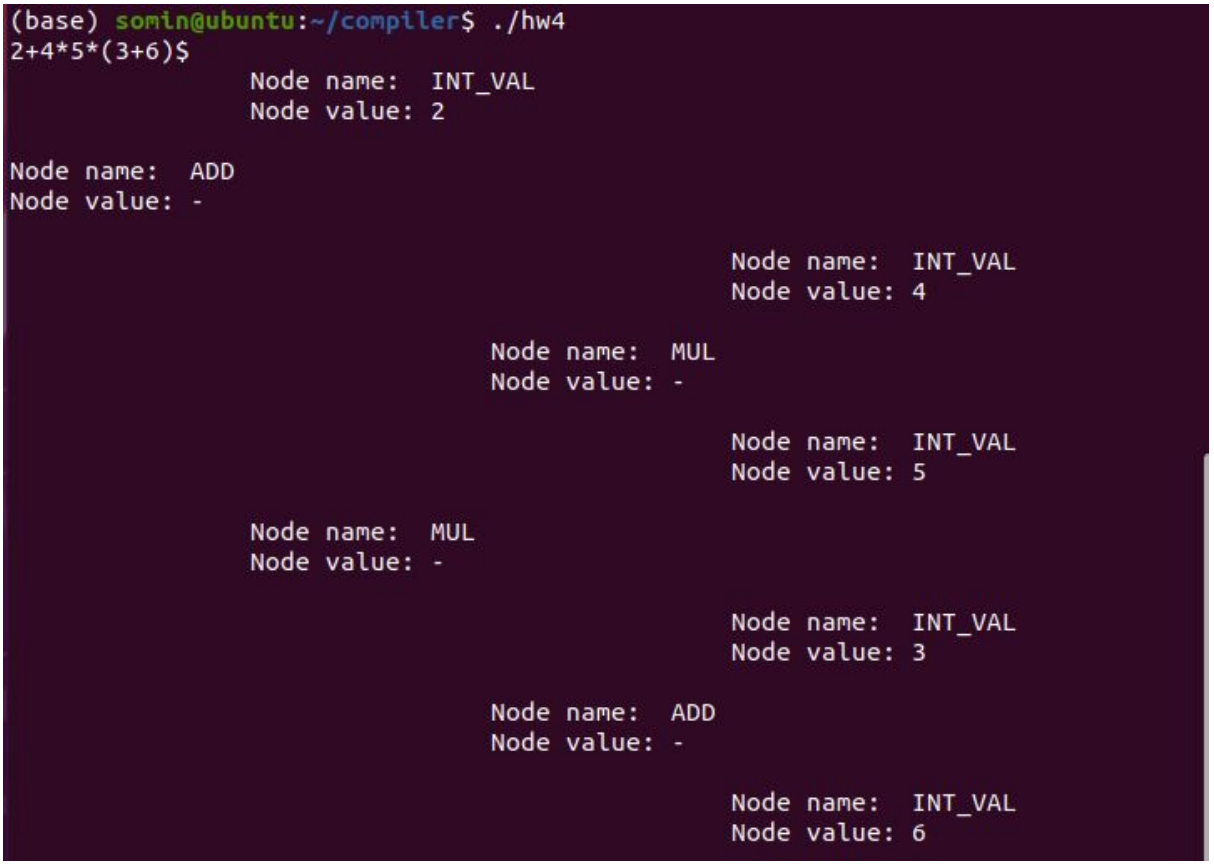
```
    top -= prod_length[i];
    old_top = top;
    push(go_to[stack[old_top]][prod_left[i]]);

    switch(i) {
        case 1:
            value[top] = makenode(ADD, 0, value[old_top+1], value[old_top+3]);
            break;
        case 2:
            value[top] = value[old_top+1];
            break;
        case 3:
            value[top] = makenode(MUL, 0, value[old_top+1], value[old_top+3]);
            break;
        case 4:
            value[top] = value[old_top+1];
            break;
        case 5:
            value[top] = value[old_top+2];
            break;
        case 6:
            value[top] = makenode(INT_VAL, value[old_top+1]->val, NULL, NULL);
            break;
        default:
            yyerror("parsing table err");
            break;
    }

}

void yyerror() {
    printf("Syntax error\n");
    exit(1);
}

int yylex() {
    static char ch = ' ' ;
    int i = 0;

    while(ch == ' ' || ch == '\t' || ch == '\n') ch = getchar();
    if(isdigit(ch)) {
        do {
            yytext[i++] = ch;
            ch = getchar();
        } while(isdigit(ch));
```

```
        yytext[i] = 0;
        yylval = atoi(yytext);
        return (NUMBER);
    }
    else if (ch == '+') {ch = getchar(); return (PLUS);}
    else if (ch == '*') {ch = getchar(); return (STAR);}
    else if (ch == '(') {ch = getchar(); return (LPAREN);}
    else if (ch == ')') {ch = getchar(); return (RPAREN);}
    else if (ch == '$') return (END);
    else lex_error();
}

void lex_error() {
    printf("illegal token\n");
    exit(1);
}
```

2. 수행 결과

```
(base) somin@ubuntu:~/compiler$ ./hw4
(1+2)*3$
                              Node name:  INT_VAL
                              Node value: 1

                Node name:  ADD
                Node value: -

                              Node name:  INT_VAL
                              Node value: 2

Node name:  MUL
Node value: -

                Node name:  INT_VAL
                Node value: 3
```

```
(base) somin@ubuntu:~/compiler$ ./hw4
3+4$
                Node name:  INT_VAL
                Node value: 3

Node name:  ADD
Node value: -

                Node name:  INT_VAL
                Node value: 4
```



3. 문제 해결 방법

트리를 위아래로 출력하는 것은 구현이 어려워서 옆으로 출력하는 방식을 선택했습니다.
inorder 방식을 선택해서 왼쪽에 있는 것(llist)을 모두 출력하고, 자신(node)을 출력하고, 그 다음에 오른쪽(rlist)에 있는 것을 모두 출력했습니다.
더 왼쪽에 있을 수록 먼저 출력되기 때문에 상단에 출력되고 트리를 눕혀둔 것과 같은 형태로 출력이 됩니다.
main함수에서 yyparse()가 끝나면 트리 출력 함수를 호출합니다. value[top]에서 시작해서 llist가 null일 때까지 트리 출력 함수를 재귀호출 합니다. 그래서 왼쪽에 있는 모든 것을 출력하게 되면 본인을 출력하고 rlist가 null일 때까지, 즉 더 이상 rvalue가 없을 때까지 또 트리 출력 함수를 재귀호출하는 방식을 사용했습니다.
또한 depth에 비례해서 tab을 출력해서 같은 depth의 node들은 동일 선상에 표현했습니다.

수식 계산을 할 것이 아니라서 shift를 할 때에도 노드를 만들어야 하기 때문에 2번 과제 shift 함수에서의 value[top] = yylval; 부분을 value[top] = makenode(INT_VAL, yylval, NULL, NULL);로 수정하였습니다.