<자료구조 5차 프로그래밍 과제 - Tree>

20172076 소프트웨어학과 이균

1. Data Structure

본 프로그램은 prefix형식으로 작성된 expression을 입력 받아 그 계산 결과를 출력하는 프로그램이다. 이때 수식을 binary tree로 구현한 후, 그에 따라 tree traversal algorithm을 사용하여 식을 계산한다.
따라서 tree를 구현하기위해 linked list형태로 tree를 구현하였다.

```
typedef struct node *treePointer;
typedef struct node{
   char data;
   int value;
   treePointer leftChild, rightChild;
}Node;
```

Node에는 입력되는 string을 저장할 data와 수식의 계산결과를 저장하기 위한, 즉 자신의 자식 tree에서의 계산 결과를 저장하게 될 value가 포함된다.

다음은 입력되는 string에 따라 tree를 만들기 위한 코드이다.

```
treePointer createTree(char** expr)
{
   char oneCharacter[2] = {0, };
   exptr = *expr;
   treePointer ptr = createNode(*exptr);
   if(isOperator(*exptr))
   {
      exptr += 1;
      ptr->leftChild = createTree(&exptr);
      exptr += 1;
      ptr->rightChild = createTree(&exptr);
   }
   else if(48 < *exptr && *exptr < 57){ // Not Operator -> expression is a
digit
      oneCharacter[0] = *exptr;
      ptr->value = atoi(oneCharacter); // change character to integer
   }
   else //
   {
      error = ERROR_OCCURE;
      return NULL;
   }
   return ptr;
}
```

포인터의 증가시키며 다음 노드를 만들어야 하기 때문에 포인터의 증가를 위해서 전역변수 char* exptr;을 선언하였다. createTree에서는 우선 createNode를 통해서 노드를 만들고, 이후 string을 차례로 읽으면서 만약 operator일 경우 포인터를 증가해서 다음 character에 대하여 다시 tree를 만든다.
만약 character가 숫자일 경우에는 그 character을 integer로 바꿔주어야 추후에 수식의 계산을 저장하는 value값으로 사용할 수 있기 때문에 atoi함수를 이용해서 character을

integer로 바꿔주고 있다. 이때 atoi함수의 특성상 한 문자만 다루기 위해서 oneCharacter 배열을 선언하여 하나의 character만 변환할 수 있게 하였다.
(char oneCharacter[2] = {0, };)

마지막으로 operater에 +,-,/,*가 아닌 다른 문자가 올 경우 에러를 표시하기 위해서 int error = 0; 라는 전역변수를 선언하였고, else case일 경우 이를 -1(ERROR_OCCURE)로 바꿔줘서 에러를 체크하도록 하였다.

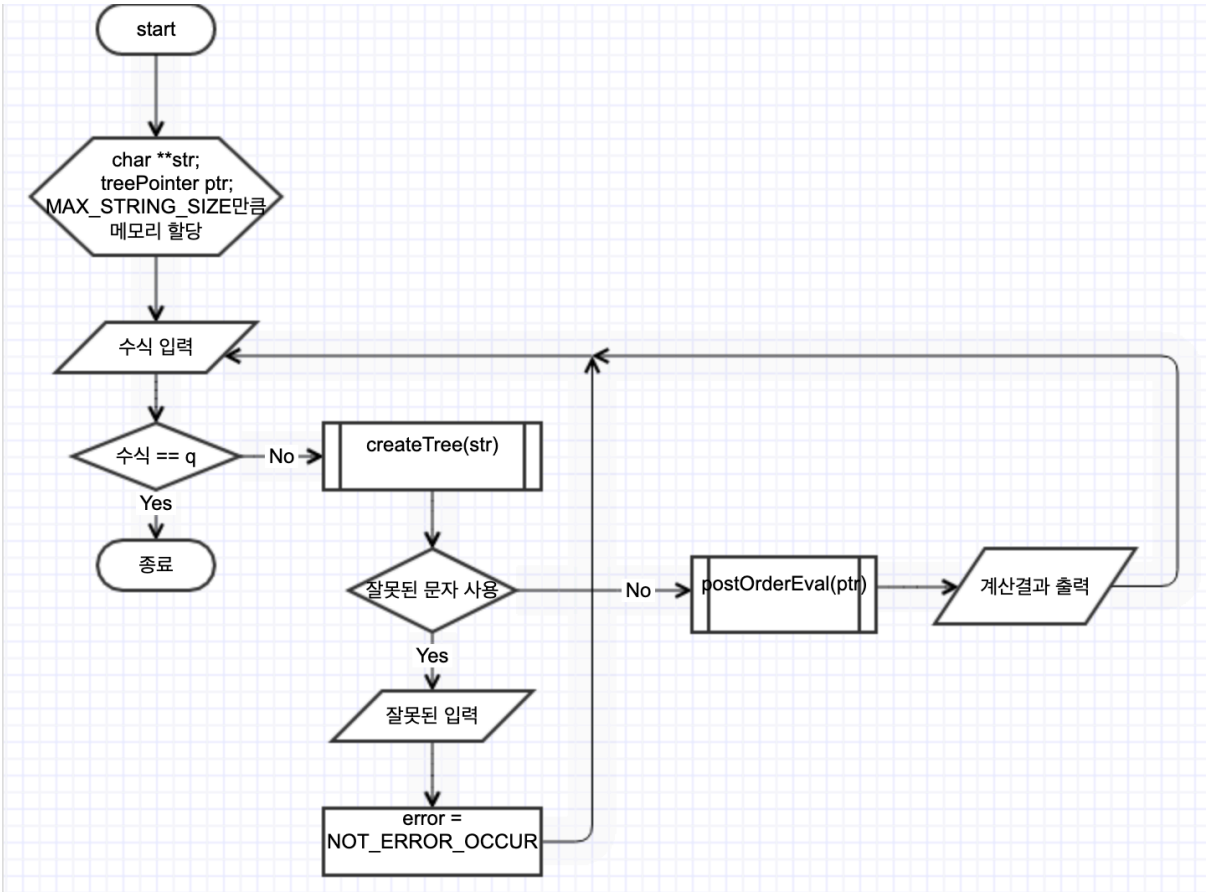
참고로 createNode함수에서는 Node만큼 메모리를 할당해준 뒤에 데이터를 집어넣고, 포인터와 value를 초기화하는 작업을 하고 있다.

다음은 tree로 표현된 prefix expression을 evaluation하기 위한 코드이다.

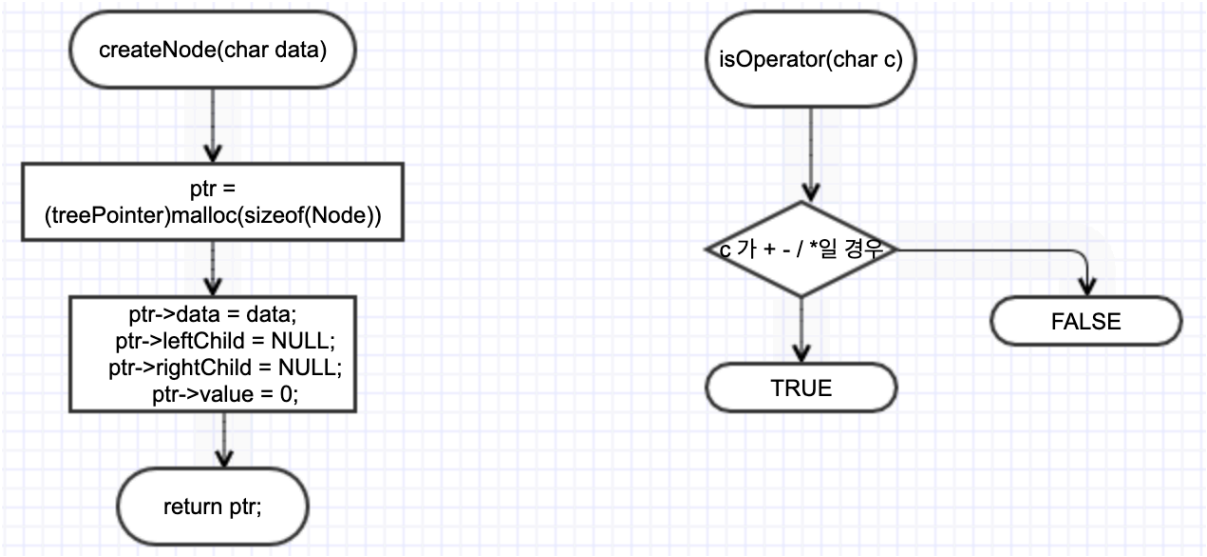
```
void postOrderEval(treePointer node)
{
   if(node)
   {
      postOrderEval(node->leftChild);
      postOrderEval(node->rightChild);
      switch (node->data) {
         case '+':
            node->value = (node->leftChild->value) + (node->rightChild-
>value);
            break;
         case '-':
            node->value = (node->leftChild->value) - (node->rightChild-
>value);
            break;
         case '*':
            node->value = (node->leftChild->value) * (node->rightChild-
>value);
            break;
         case '/':
            node->value = (node->leftChild->value) / (node->rightChild-
>value);
            break;
         default:
            break;
      }
   }
}
```

계산을 위해서 앞서 언급한 value라는 값을 이용해야 한다. 수식 계산을 할 때에는 postfix traversal algorithm을 사용하는데 이는 식을 계산 하려면 operand와 operand를 파악한 뒤 operator을 적용해야 하기 때문에 이러한 논리의 흐름에 알맞게 하기위해서는 postfix traversal algorithm을 사용해야 한다.
따라서 node가 있을 때 postOrderEval을 왼쪽과 오른쪽 자식에게 적용한 뒤(재귀적으로) 자식 tree의 value값을 노드의 operator에 따라서 계산해주는 코드가 바로 위의 코드이다. Operator가 +일 경우에는 왼쪽자식의 vlaue와 오른쪽 자식의 value를 더해주면 되고, 나머지 연산자들도 같은 논리이다.

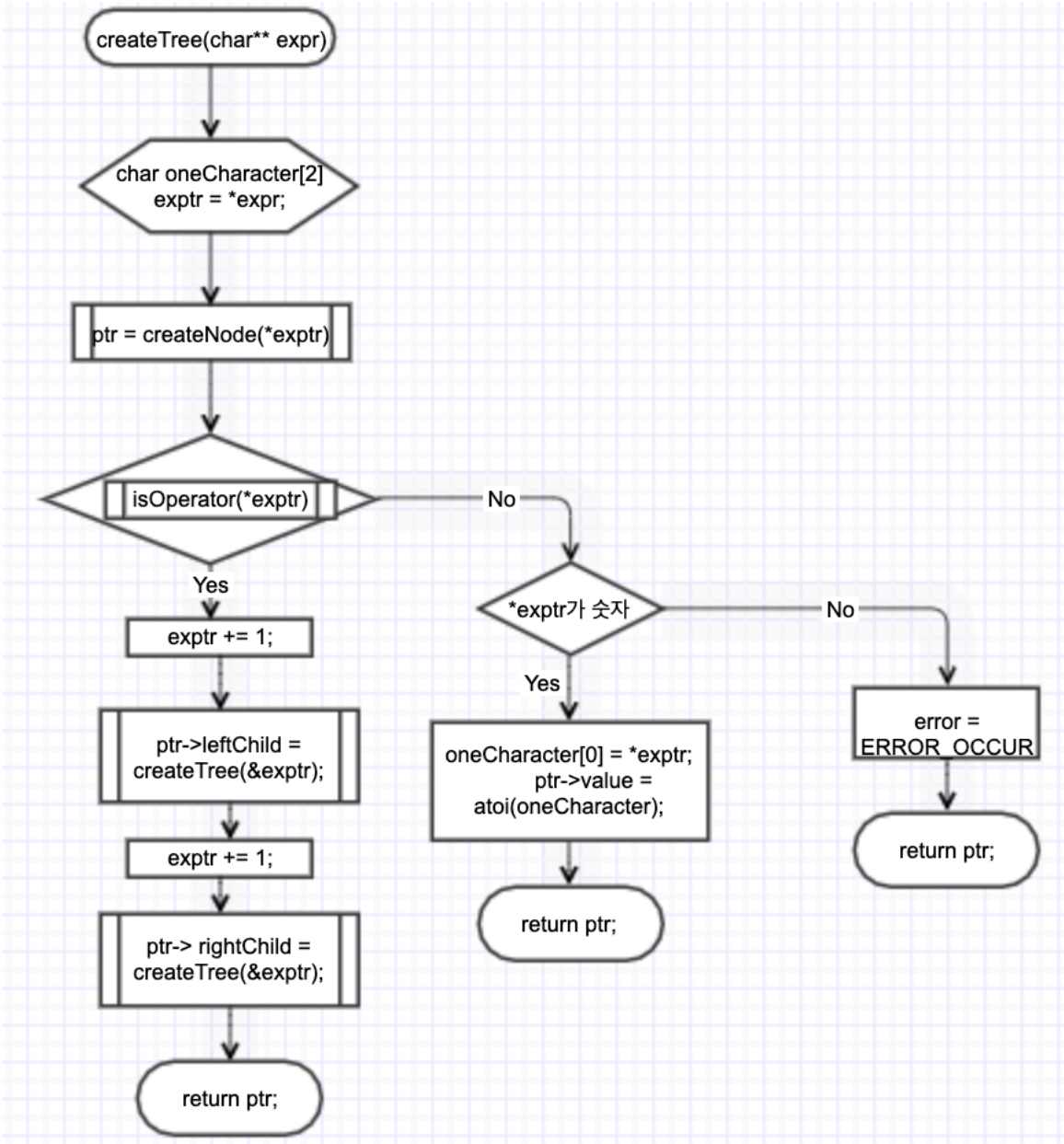
2.Flow Chart



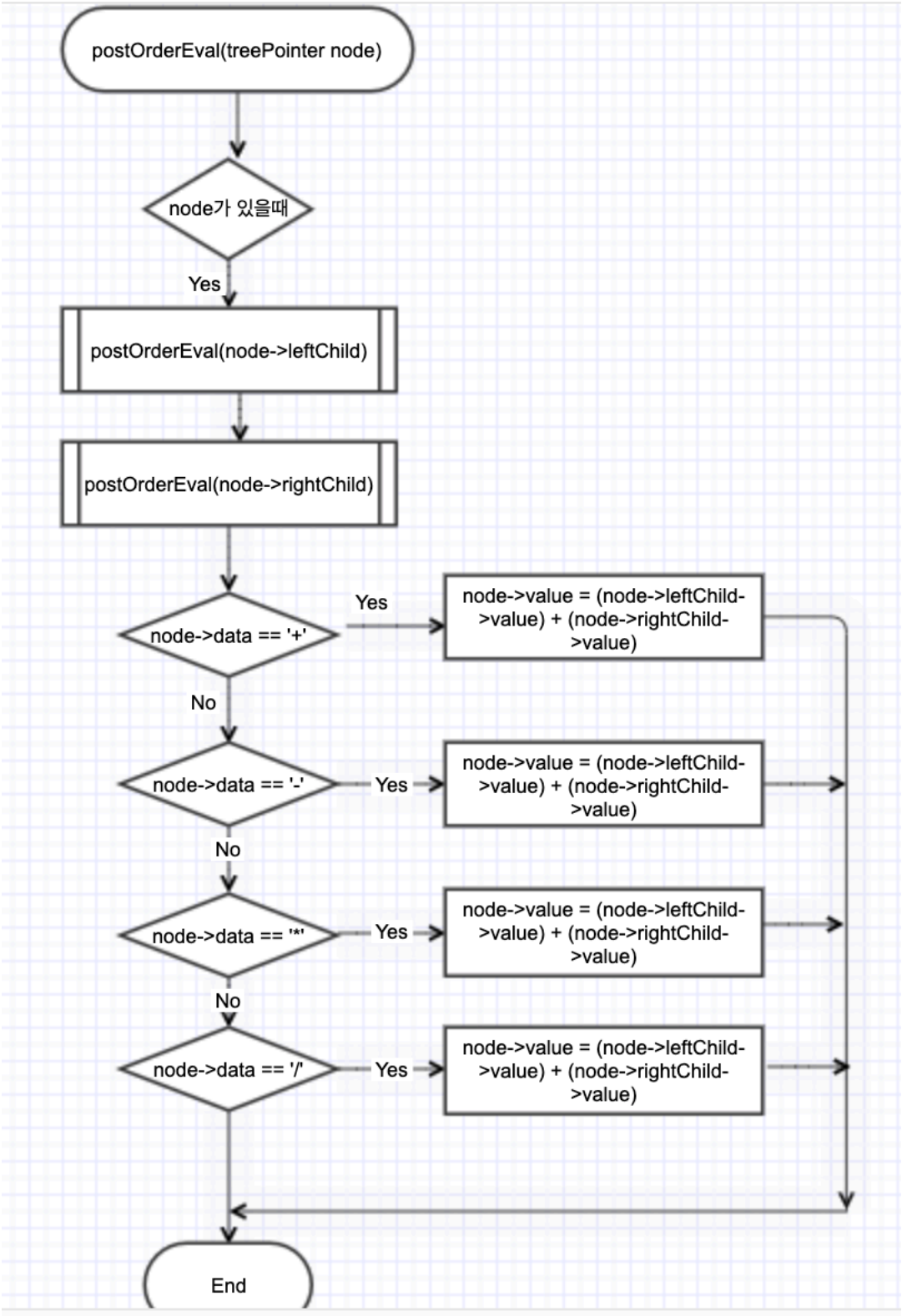
Main 함수



createNode함수, isOperator함수



createTree함수



postOrderEval함수