컴퓨터 프로그래밍 2
-hw12-

학번 : 201602038
제출일 : 2017.6.5.
이름 : 이 미 진

1. 함수 설명

1-1. 수식 계산

1. main

1) AppIO* AppIO_new(): 객체 생성

2) void AppIO_delete(AppIO* _this): 객체 소멸

3) Boolean AppIO_in_postfixExpression(AppIO* _this,char* anExpression): 수식을 입력받음 , $를 입력받으면 종료

4) void AppIO_out_errorInExpression (AppIO* _this): 수식에 오류가 있을 때 오류 메시지 출력

5) void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue): 계산값 출력

6) void AppIO_out_startingMessage(AppIO* _this): 시작 메시지 출력

7) void AppIO_out_endingMessage(AppIO* _this): 종료 메시지 출력

8) void AppIO_outLine(AppIO* _this,char* aMessage): 수식에 오류가 있을 때 오류 메시지 출력

9) void AppIO_printTop(AppIO* _this): <Top> 출력

10) void AppIO_out_newLine(AppIO* _this) : 개행

11) void AppIO_out_bottomOfStack(AppIO* _this): <Bottom> 출력

2. AppIO

1) AppIO* AppIO_new(): 객체 생성

2) void AppIO_delete(AppIO* _this): 객체 소멸

3) Boolean AppIO_in_postfixExpression(AppIO* _this,char* anExpression): 수식을 입력받음 , $를 입력받으면 종료

4) void AppIO_out_errorInExpression (AppIO* _this): 수식에 오류가 있을 때 오류 메시지 출력

5) void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue): 계산값 출력

6) void AppIO_out_startingMessage(AppIO* _this): 시작 메시지 출력

7) void AppIO_out_endingMessage(AppIO* _this): 종료 메시지 출력

8) void AppIO_outLine(AppIO* _this,char* aMessage): 수식에 오류가 있을 때 오류 메시지 출력

9) void AppIO_printTop(AppIO* _this): <Top> 출력

3. AppController

1) AppController* AppController_new(): 객체 생성

2) void AppController_delete(AppController* _this): 객체 소멸

3) void AppController_run(AppController* _this): 프로그램 실행

4) void AppController_out_postfixEvaluationErrorMessage(AppController* _this,PostfixError aPostfixError) : 에러 메시지 출력

4. Stack

1) Stack* Stack_new(): stack 객체 생성

2) void Stack_delete(Stack* _this): stack 객체 소멸

3) Boolean Stack_isEmpty(Stack* _this): 스택이 empty 이면 TRUE를, 아니면, FALSE를 얻는다.

4) Boolean Stack_isFull(Stack* _this): 스택이 full 이면 TRUE 를, 아니면, FALSE 를 얻는다.

5) Element Stack_push(Stack* _this, Element anElement): 스택에 anElement 를 삽입

6) Element Stack_pop(Stack* _this): 스택의 top에서 원소를 삭제하고 그 값을 얻는다.

7) void Stack_reset(Stack* _this) : 스택 초기화

8) int Stack_size(Stack* _this) : 스택 사이즈 리턴

9) Element Stack_elementAt(Stack* _this, int aPosition) : 스택 리스트의 aPosition 번째 원소를 얻는다.

10) void Stack_deleteLinkedNodes(Stack* _this) : 연결 노드 삭제

5. Node

1) Node* Node_new(): 노드 객체 생성

2) void Node_delete(): 노드 객체 소멸

3) void Node_setElement(Node* _this, Element newElement): 노드 element의 설정자

4) Element Node_element(Node* _this): 노드의 원소 리턴

5) void Node_setNext(Node* _this, Node* newNext): 노드 next의 설정자

6) Node* Node_next(Node* _this): 노드의 next 리턴

6. Postfix

1) Postfix* Postfix_new(int givenMaxNumberOfTokens): 객체 생성

2) void Postfix_delete(Postfix* _this): 객체 소멸

3) void Postfix_setExpression(Postfix* _this, char* anExpression): 계산할 postfix 수식인 expression[] 을 postfix 객체에 전달

4) void Postfix_showTokenAndStack(Postfix* _this, char currentToken): 스택과 토큰을 보여줌

5) PostfixError Postfix_evaluate(Postfix* _this): 현재 객체가 가지고 있는 postfix 수식을 계산하도록 지시

6) int Postfix_evaluatedValue(Postfix* _this): 계산된 결과 값을 Postifx 객체로부터 얻는다.

1-2. 추가 과제

1. 수식 계산 함수에서 추가된 함수

1) Element Stack_peekElement(Stack* _this): top에 있는 원소를 리턴

2) Node* Stack_top(Stack* _this): top 리턴

3) void Postfix_setInfix(Postfix* _this, char* newInfix): Postfix 객체에게 infix 수식을 전달

4) Boolean Postfix_infixToPostfix(Postfix* _this): Postfix 객체가 가지고 있는 infix 수식을 postfix 수식으로 변환할 것을 Postfix 객체에게 시킨다

5) char* Postfix_postfix(Postfix* _this): 변환된 postfix 수식을 얻는다.

2. 전체 코드

2-1. 수식 계산

1) main.c

```
//
//  main.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include <stdio.h>
#include "AppIO.h"
#include "AppController.h"
#include "Common.h"

int main() {
    AppController* appController =  AppController_new();    // 객체
생성
    AppController_run(appController);   // 프로그램 실행
    AppController_delete(appController);    // 객체 소멸
}
```

2) AppIO.c

```
//
//  AppIO.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppIO.h"

void AppIO_out_newLine(AppIO* _this);
void AppIO_out_bottomOfStack(AppIO* _this);

struct _AppIO {

};

AppIO* AppIO_new() {    // 객체 생성
    AppIO* _this = NewObject(AppIO);

    return _this;
}

void AppIO_delete(AppIO* _this) {   // 객체 소멸
    free(_this);
}
```

```
Boolean AppIO_in_postfixExpression(AppIO* _this,char*
anExpression) {   // 수식을 입력받음 , $를 입력받으면 종료
    printf(">Postfix 수식을 입력하시오 :");
    scanf("%s",anExpression);
    if( anExpression[0] == '$' ) {
        return FALSE;
    }
    return TRUE;
}
void AppIO_out_errorInExpression (AppIO* _this) {   // 수식에 오류가
있을 때 오류 메시지 출력
    printf(">수식에 오류가 있습니다\n");
}

void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue)
{  // 계산값 출력
    printf("계산값 : %d\n",anEvaluatedValue);
}

void AppIO_out_newLine(AppIO* _this) {  // 개행
    printf("\n");
}

void AppIO_out_bottomOfStack(AppIO* _this) {    //  <Bottom> 출력
    printf("<Bottom>");
}

void AppIO_out_startingMessage(AppIO* _this) {  // 시작 메시지 출력
    printf("<Postfix 수식을 계산합니다>\n");
}

void AppIO_out_endingMessage(AppIO* _this) { // 종료 메시지 출력
    printf("<계산을 종료합니다>\n");
}

void AppIO_outLine(AppIO* _this,char* aMessage){  // 수식에 오류가 있을
때 오류 메시지 출력
    printf("%s\n",aMessage);
}

void AppIO_printTop(AppIO* _this) { //<Top> 출력
    printf("<Top>\n");
}
```

3) AppIO.h

```
//
//  AppIO.h
```

```
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppIO_h
#define AppIO_h

#include <stdio.h>
#include "Common.h"
#include "Message.h"
#include "Postfix.h"

typedef struct _AppIO AppIO;

AppIO* AppIO_new();     // 객체 생성
void AppIO_delete(AppIO* _this);     // 객체 소멸
Boolean AppIO_in_postfixExpression(AppIO* _this,char*
anExpression);   // 수식을 입력받음 , $를 입력받으면 종료
void AppIO_out_errorInExpression (AppIO* _this);   // 수식에 오류가 있
을 때 오류 메시지 출력
void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue);
// 계산값 출력
void AppIO_out_startingMessage(AppIO* _this);  // 시작 메시지 출력
void AppIO_out_endingMessage(AppIO* _this); // 종료 메시지 출력
void AppIO_outLine(AppIO* _this,char* aMessage);     // 수식에 오류가
있을 때 오류 메시지 출력
void AppIO_printTop(AppIO* _this);  // <Top> 출력


#endif /* AppIO_h */
```

4) AppController.c

```
//
//  AppController.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppController.h"

struct _AppController {
    AppIO* _appIO;
    Stack* _stack;
    char _expression[MAX_NUMBER_OF_TOKENS];
```

```
    Postfix* _postfix;
};

void
AppController_out_postfixEvaluationErrorMessage(AppController*
_this,PostfixError aPostfixError);

AppController* AppController_new(){ // 객체 생성
    AppController* _this;
    _this = NewObject(AppController);
    _this->_stack = Stack_new();
    _this->_appIO = AppIO_new();

    return _this;
}

void AppController_delete(AppController* _this){    // 객체 소멸
    AppIO_delete(_this->_appIO);
    Stack_delete(_this->_stack);
    free(_this);
}

void
AppController_out_postfixEvaluationErrorMessage(AppController*
_this,PostfixError aPostfixError) {
    if( aPostfixError == PostfixError_ExpressionTooLong ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_ExpressionTooLong );
    }
    else if( aPostfixError == PostfixError_OperandsTooMany ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_OperandsTooMany);
    }
    else if( aPostfixError == PostfixError_OperandsTooFew ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_OperandsTooFew );
    }
    else if( aPostfixError == PostfixError_UnknownOperator ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_UndefinedOperator);
    }
    else if( aPostfixError == PostfixError_DivideByZero ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_DivideByZero );
    }
}

void AppController_run(AppController* _this) {   // 프로그램 실행
    Boolean nextPostfixExpressionIsAvailable;
    PostfixError evaluationError;

    AppIO_out_startingMessage(_this->_appIO);   // 시작 메시지 출력
    _this->_postfix = Postfix_new(MAX_NUMBER_OF_TOKENS);
    nextPostfixExpressionIsAvailable =
AppIO_in_postfixExpression(_this->_appIO, _this->_expression);
```

```
    while( nextPostfixExpressionIsAvailable ) {
        Postfix_setExpression(_this->_postfix, _this-
>_expression);
        evaluationError = Postfix_evaluate(_this->_postfix);

        if( evaluationError == PostfixError_None ) {
            AppIO_out_evaluatedValue(_this->_appIO,
Postfix_evaluatedValue(_this->_postfix));
        }
        else {
            AppController_out_postfixEvaluationErrorMessage(_this,
evaluationError);
        }
        nextPostfixExpressionIsAvailable =
AppIO_in_postfixExpression(_this->_appIO, _this->_expression);
    }
    Postfix_delete(_this->_postfix);
    AppIO_out_endingMessage(_this->_appIO);

}
```

5) AppController.h

```
//
//  AppController.h
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppController_h
#define AppController_h

#include <stdio.h>
#include "Common.h"
#include "Postfix.h"
#include "Message.h"
#include "AppIO.h"
#include "Stack.h"

#define MAX_NUMBER_OF_TOKENS    200

typedef struct _AppController AppController;

AppController* AppController_new(); // 객체 생성
void AppController_delete(AppController* _this);    // 객체 소멸
void AppController_run(AppController* _this);   // 프로그램 실행

#endif /* AppController_h */
```

6) Stack.c

```
//
//  Stack.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Stack.h"
#include "Node.h"

#define MAX_STACK_SIZE 5

void Stack_deleteLinkedNodes(Stack* _this);

struct _Stack {
    int _size;
    Node* _top;
    Element* _elements;
};

Stack* Stack_new() {    // stack 객체 생성
    Stack* _this;
    _this = NewObject(Stack);
    _this->_top = NULL;
    _this->_size = 0;

    return _this;
}

void Stack_delete(Stack* _this){    // stack 객체 소멸
    Stack_deleteLinkedNodes(_this);
    free(_this);
}

void Stack_reset(Stack* _this) {    // 스택 초기화
    _this->_top = NULL;
}

Boolean Stack_isEmpty(Stack* _this){    // 스택이 empty 이면 TRUE를,
아니면, FALSE를 얻는다.
    return ((_this->_size) == 0);
}
Boolean Stack_isFull(Stack* _this){ // 스택이 full 이면 TRUE 를, 아니
면, FALSE 를 얻는다.
    return FALSE;
}

int Stack_size(Stack* _this) {  // 스택 사이즈 리턴
```

```
    return _this->_size;
}

Element Stack_push(Stack* _this, Element anElement){    // 스택에
anElement 를 삽입
    Node* addedNode = Node_new();

    if( Stack_isFull(_this)) {
        return FALSE;
    }
    Node_setElement(addedNode,anElement);
    Node_setNext(addedNode,_this->_top);
    _this->_top = addedNode;
    _this->_size++;

    return TRUE;
}

Element Stack_pop(Stack* _this){    // 스택의 top에서 원소를 삭제하고 그
값을 얻는다.
    Element removedElement;
    Node* removedNode = _this->_top;

    if( Stack_isEmpty(_this)) {
        return FALSE;
    }
    removedElement = Node_element(removedNode);
    if( Node_next(removedNode) != NULL ) {
        _this->_top = Node_next(removedNode);
    }
    else {
        _this->_top = NULL;
    }
    Node_delete(removedNode);
    _this->_size--;

    return removedElement;
}

void Stack_deleteLinkedNodes(Stack* _this) {
    Node* currentNode = _this->_top;
    Node* nodeToBeDeleted = NULL;

    while( currentNode != NULL ) {
        nodeToBeDeleted = currentNode;
        currentNode = Node_next(currentNode);
        Node_delete(nodeToBeDeleted);
    }
}
```

```
Element Stack_elementAt(Stack* _this, int aPosition) {   //스택 리스트의 aPosition 번째 원소를 얻는다.
    Element top;
    Node* positionNode = _this->_top;

    for( int Position=0; Position < aPosition; Position++ ) {
        positionNode = Node_next(positionNode);
    }

    top = Node_element(positionNode);

    return top;
}
```

7) Stack.h

```
//
//  Stack.h
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Stack_h
#define Stack_h

#include <stdio.h>
#include "Common.h"

typedef struct _Stack Stack;
typedef int Element;

Stack* Stack_new(); // stack 객체 생성
void Stack_delete(Stack* _this);    // stack 객체 소멸
Boolean Stack_isEmpty(Stack* _this);    // 스택이 empty 이면 TRUE를, 아니면, FALSE를 얻는다.
Boolean Stack_isFull(Stack* _this); // 스택이 full 이면 TRUE 를, 아니면, FALSE 를 얻는다.
Element Stack_push(Stack* _this, Element anElement);    // 스택에 anElement 를 삽입
Element Stack_pop(Stack* _this);    // 스택의 top에서 원소를 삭제하고 그 값을 얻는다.
void Stack_reset(Stack* _this); // 스택 초기화
int Stack_size(Stack* _this);   // 스택의 사이즈 리턴
Element Stack_elementAt(Stack* _this, int aPosition);   //스택 리스트의 aPosition 번째 원소를 얻는다.
```

```
#endif /* Stack_h */
```

8) Node.c

```
//
//  Node.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Node.h"

Node* Node_new() {  // 노드 객체 생성
    Node* _this = NewObject(Node);
    _this->_element = 0;
    _this->_next = NULL;

    return _this;
}

void Node_delete(Node* _this) { // 노드 객체 소멸
    free(_this);
}

void Node_setElement(Node* _this, Element newElement) {  // 노드
element의 설정자
    _this->_element = newElement;
}

Element Node_element(Node* _this){  // 노드의 원소 리턴
    return _this->_element;
}

void Node_setNext(Node* _this, Node* newNext) {  // 노드 next의 설정자
    _this->_next = newNext;
}

Node* Node_next(Node* _this) {  // 노드의 next 리턴
    return _this->_next;
}
```

9) Node.h

```
//
//  Node.h
//  CP2_WEEK12
```

```
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Node_h
#define Node_h

#include <stdio.h>
#include "Common.h"
#include "Stack.h"

typedef struct _Node Node;
struct _Node {
    Element _element;
    Node* _next;
};

Node* Node_new();   // 노드 객체 생성
void Node_delete(); // 노드 객체 소멸
void Node_setElement(Node* _this, Element newElement);  // 노드
element의 설정자
Element Node_element(Node* _this);  // 노드의 원소 리턴
void Node_setNext(Node* _this, Node* newNext);  // 노드 next의 설정자
Node* Node_next(Node* _this);   // 노드의  next 리턴


#endif /* Node_h */
```

10) Common.h

```
//
//  Common.h
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Common_h
#define Common_h

#include <stdlib.h>

#define NewVector(TYPE,SIZE) (TYPE*)malloc(sizeof(TYPE)*SIZE)
#define NewObject(TYPE) (TYPE*)malloc(sizeof(TYPE))
typedef enum {FALSE,TRUE} Boolean;  // FALSE와 TRUE 값을 갖는 Boolean
선언
```

```
typedef enum {
    PostfixError_None,
    PostfixError_ExpressionTooLong,
    PostfixError_OperandsTooMany,
    PostfixError_OperandsTooFew,
    PostfixError_UnknownOperator,
    PostfixError_DivideByZero
} PostfixError;

#endif /* Common_h */
```

11) Postfix.c

```
//
//  Postfix.c
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Postfix.h"
#include "Stack.h"
#include "AppIO.h"
#include <string.h>


struct _Postfix {
    int _maxNumberOfTokens;
    char* _expression;
    int _evaluatedValue;
    Stack* _operandStack;
    AppIO* _appIO;
};

Postfix* Postfix_new(int givenMaxNumberOfTokens) {  // 객체 생성
    Postfix* _this = NewObject(Postfix);
    _this->_maxNumberOfTokens = givenMaxNumberOfTokens;
    _this->_expression = NewVector(char, givenMaxNumberOfTokens);
    _this->_operandStack = Stack_new(givenMaxNumberOfTokens);

    return _this;
}

void Postfix_delete(Postfix* _this) {    // 객체 소멸
    Stack_delete(_this->_operandStack);
    free(_this->_expression);
    free(_this);
}
```

```
void Postfix_setExpression(Postfix* _this, char* anExpression) {
    //계산할 postfix 수식인 expression[] 을 postfix 객체에 전달
    strcpy(_this->_expression,anExpression);
}

void Postfix_showTokenAndStack(Postfix* _this, char currentToken){
// 스택과 토큰을 보여줌
    printf("%c : Stack <Bottom> ", currentToken);
    for( int i = Stack_size(_this->_operandStack)-1; i>=0; i-- ){
        int stackElement = Stack_elementAt(_this->_operandStack,
i);
        printf("%d ", stackElement);
    }
    AppIO_printTop(_this->_appIO);
}

PostfixError Postfix_evaluate(Postfix* _this){   //현재 객체가 가지고
있는 postfix 수식을 계산하도록 지시
    int operand, operand1, operand2, calculated;
    char currentToken;
    int i = 0;
    Stack_reset(_this->_operandStack);

    while ( _this->_expression[i] != '\0' ) {
        currentToken = _this->_expression[i];
        if( currentToken >= '0' && currentToken <= '9' ){
            operand = ( currentToken - '0' );
            if( Stack_isFull(_this->_operandStack) ) {
                return PostfixError_ExpressionTooLong;
            }
            else {
                Stack_push(_this->_operandStack, operand);
            }
        }

        else {
            if( currentToken == '+') {  // 덧셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);
                    calculated = operand1 + operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else if ( currentToken == '-') {    // 뺄셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
```

```
                operand1 = Stack_pop(_this->_operandStack);
                calculated = operand1 - operand2;

                Stack_push(_this->_operandStack, calculated);
            }
            else {
                return PostfixError_OperandsTooFew;
            }
        }
        else if ( currentToken == '*') {    // 곱셈
            if(Stack_size(_this->_operandStack) >= 2 ) {
                operand2 = Stack_pop(_this->_operandStack);
                operand1 = Stack_pop(_this->_operandStack);
                calculated = operand1 * operand2;
                Stack_push(_this->_operandStack, calculated);
            }
            else {
                return PostfixError_OperandsTooFew;
            }
        }
        else if( currentToken == '/') { // 나눗셈
            if(Stack_size(_this->_operandStack) >= 2 ) {
                operand2 = Stack_pop(_this->_operandStack);
                operand1 = Stack_pop(_this->_operandStack);
                if( operand2 == 0 ) {
                    return PostfixError_DivideByZero;
                }
                calculated = operand1 / operand2;
                Stack_push(_this->_operandStack, calculated);
            }
            else {
                return PostfixError_OperandsTooFew;
            }
        }
        else if ( currentToken == '%' ) {   // 나머지 연산
            if(Stack_size(_this->_operandStack) >= 2 ) {
                operand2 = Stack_pop(_this->_operandStack);
                operand1 = Stack_pop(_this->_operandStack);

                calculated = operand1 % operand2;
                Stack_push(_this->_operandStack, calculated);
            }
            else {
                return PostfixError_OperandsTooFew;
            }
        }
        else {
            return PostfixError_UnknownOperator;
        }
    }
    Postfix_showTokenAndStack(_this,currentToken);
```

```
        i++;
    }// end of while
    if( Stack_size(_this->_operandStack) > 1 ) {
        return PostfixError_OperandsTooMany;
    }
    else if( Stack_size(_this->_operandStack) == 1 ) {
        _this->_evaluatedValue = Stack_pop(_this->_operandStack);
    }
    return PostfixError_None;
    }

int Postfix_evaluatedValue(Postfix* _this){ // 계산된 결과 값을
Postifx 객체로부터 얻는다.
    return (_this->_evaluatedValue);
}
```

12) Postfix.h

```
//
//  Postfix.h
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Postfix_h
#define Postfix_h

#include <stdio.h>
#include "Common.h"
typedef struct _Postfix Postfix;

Postfix* Postfix_new(int givenMaxNumberOfTokens);  // 객체 생성
void Postfix_delete(Postfix* _this);    // 객체 소멸
void Postfix_setExpression(Postfix* _this, char* anExpression); //
계산할 postfix 수식인 expression[] 을 postfix 객체에 전달
void Postfix_showTokenAndStack(Postfix* _this, char currentToken);
// 스택과 토큰을 보여줌
PostfixError Postfix_evaluate(Postfix* _this);   //현재 객체가 가지고
있는 postfix 수식을 계산하도록 지시
int Postfix_evaluatedValue(Postfix* _this); // 계산된 결과 값을
Postifx 객체로부터 얻는다.

#endif /* Postfix_h */
```

13) Message.h

```
//
//  Message.h
//  CP2_WEEK12
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Message_h
#define Message_h

#define ErrorMsg_ExpressionTooLong  "[오류] 수식이 너무 길어 처리가 불가능합니다.\n"
#define ErrorMsg_OperandsTooMany    "[오류] 연산자에 비해 연산값의 수가 적습니다.\n"
#define ErrorMsg_OperandsTooFew "[오류] 연산자에 비해 연산값의 수가 많습니다.\n"
#define ErrorMsg_UndefinedOperator  "[오류]수식에 알 수 없는 연산자가 있습니다.\n"
#define ErrorMsg_DivideByZero   "[오류] 나눗셈의 분모가 0 입니다.\n"

#endif /* Message_h */
```

2-2. 추가 과제

1) main.c

```
//
//  main.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include <stdio.h>
#include "AppIO.h"
#include "AppController.h"
#include "Common.h"

int main() {
    AppController* appController =  AppController_new();    // 객체생성
    AppController_run(appController);   // 프로그램 실행
    AppController_delete(appController);    // 객체 소멸
}
```

2) AppIO.c

```
//
//  AppIO.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppIO.h"


void AppIO_out_bottomOfStack(AppIO* _this);

struct _AppIO {

};

AppIO* AppIO_new() {    // 객체 생성
    AppIO* _this = NewObject(AppIO);

    return _this;
}

void AppIO_delete(AppIO* _this) {   // 객체 소멸
    free(_this);
}

Boolean AppIO_in_postfixExpression(AppIO* _this,char*
anExpression) {   // 수식을 입력받음 , $를 입력받으면 종료
    printf(">infix 수식을 입력하시오 :");
    scanf("%s",anExpression);
    if( anExpression[0] == '$' ) {
        return FALSE;
    }
    return TRUE;
}
void AppIO_out_errorInExpression (AppIO* _this) {   // 수식에 오류가
있을 때 오류 메시지 출력
    printf(">수식에 오류가 있습니다\n");
}

void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue)
{  // 계산값 출력
    printf("계산값 :d\n",anEvaluatedValue);
}

void AppIO_out_newLine(AppIO* _this) {  // 개행
    printf("\n");
```

```
}

void AppIO_out_bottomOfStack(AppIO* _this) {    // <Bottom> 출력
    printf("<Bottom>");
}


void AppIO_out_startingMessage(AppIO* _this) {  // 시작 메시지 출력
    printf("<Postfix 수식을 계산합니다>\n");
}

void AppIO_out_endingMessage(AppIO* _this) { // 종료 메시지 출력
    printf("<계산을 종료합니다>\n");
}

void AppIO_printTop(AppIO* _this) { // <Top> 출력
    printf("<Top>\n");
}

void AppIO_outLine(AppIO* _this,char* aMessage){    // 오류 메시지 출
력
    printf("%s\n",aMessage);
}
```

3) AppIO.h

```
//
//  AppIO.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppIO_h
#define AppIO_h

#include <stdio.h>
#include "Common.h"
#include "Message.h"
#include "Postfix.h"

typedef struct _AppIO AppIO;

AppIO* AppIO_new();    // 객체 생성
void AppIO_delete(AppIO* _this);    // 객체 소멸
Boolean AppIO_in_postfixExpression(AppIO* _this,char*
anExpression);   // 수식을 입력받음 , $를 입력받으면 종료
```

```
void AppIO_out_errorInExpression (AppIO* _this);   // 수식에 오류가 있
을 때 오류 메시지 출력
void AppIO_out_evaluatedValue(AppIO* _this, int anEvaluatedValue);
// 계산값 출력
void AppIO_out_startingMessage(AppIO* _this);  // 시작 메시지 출력
void AppIO_out_endingMessage(AppIO* _this); // 종료 메시지 출력
void AppIO_outLine(AppIO* _this,char* aMessage);    // 오류 메시지 출
력
void AppIO_printTop(AppIO* _this);  // <Top> 출력
void AppIO_out_newLine(AppIO* _this);   // 개행
#endif /* AppIO_h */
```

4) AppController.c

```
//
//  AppController.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "AppController.h"

struct _AppController {
    AppIO* _appIO;
    Stack* _stack;
    char _expression[MAX_NUMBER_OF_TOKENS];
    Postfix* _postfix;
};

void
AppController_out_postfixEvaluationErrorMessage(AppController*
_this,PostfixError aPostfixError);

AppController* AppController_new(){ // 객체 생성
    AppController* _this;
    _this = NewObject(AppController);
    _this->_stack = Stack_new();
    _this->_appIO = AppIO_new();

    return _this;
}

void AppController_delete(AppController* _this){    // 객체 소멸
    AppIO_delete(_this->_appIO);
    Stack_delete(_this->_stack);
    free(_this);
}
```

```
void
AppController_out_postfixEvaluationErrorMessage(AppController*
_this,PostfixError aPostfixError) {
    if( aPostfixError == PostfixError_ExpressionTooLong ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_ExpressionTooLong );
    }
    else if( aPostfixError == PostfixError_OperandsTooMany ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_OperandsTooMany);
    }
    else if( aPostfixError == PostfixError_OperandsTooFew ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_OperandsTooFew );
    }
    else if( aPostfixError == PostfixError_UnknownOperator ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_UndefinedOperator);
    }
    else if( aPostfixError == PostfixError_DivideByZero ) {
        AppIO_outLine(_this->_appIO, ErrorMsg_DivideByZero);
    }
}

void AppController_run(AppController* _this) {  //  프로그램 실행
    Boolean nextPostfixExpressionIsAvailable;
    PostfixError evaluationError;

    AppIO_out_startingMessage(_this->_appIO);
    _this->_postfix = Postfix_new(MAX_NUMBER_OF_TOKENS);
    nextPostfixExpressionIsAvailable =
AppIO_in_postfixExpression(_this->_appIO, _this->_expression);

    while( nextPostfixExpressionIsAvailable ) {
        Postfix_setExpression(_this->_postfix, _this-
>_expression);
        evaluationError = Postfix_evaluate(_this->_postfix);

        if( evaluationError == PostfixError_None ) {
            AppIO_out_evaluatedValue(_this->_appIO,
Postfix_evaluatedValue(_this->_postfix));
        }
        else {
            AppController_out_postfixEvaluationErrorMessage(_this,
evaluationError);
        }
        nextPostfixExpressionIsAvailable=
AppIO_in_postfixExpression(_this->_appIO, _this->_expression);
       Postfix_delete(_this->_postfix);
    }

    AppIO_out_endingMessage(_this->_appIO);

}
```

5) AppController.h

```
//
//  AppController.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef AppController_h
#define AppController_h

#include <stdio.h>
#include "Common.h"
#include "Postfix.h"
#include "Message.h"
#include "AppIO.h"
#include "Stack.h"

#define MAX_NUMBER_OF_TOKENS    200

typedef struct _AppController AppController;

AppController* AppController_new(); // 객체 생성
void AppController_delete(AppController* _this);    // 객체 소멸
void AppController_run(AppController* _this);   // 프로그램 실행


#endif /* AppController_h */
```

6) Stack.c

```
//
//  Stack.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Stack.h"

struct _Stack {
    int _size;
    Node* _top;
};

void Stack_deleteLinkedNodes(Stack* _this);
```

```
Stack* Stack_new() {    // stack 객체 생성
    Stack* _this;
    _this = NewObject(Stack);
    _this->_top = NULL;
    _this->_size = 0;

    return _this;
}

void Stack_delete(Stack* _this){    // stack 객체 소멸
    Stack_deleteLinkedNodes(_this);
    free(_this);
}

void Stack_reset(Stack* _this) {    // 스택 초기화
    Stack_deleteLinkedNodes(_this);
    _this->_size = 0;
}

Boolean Stack_isEmpty(Stack* _this){    // 스택이 empty 이면 TRUE를,
아니면, FALSE를 얻는다.
    return ((_this->_size) == 0 );
}
Boolean Stack_isFull(Stack* _this){ // 스택이 full 이면 TRUE 를, 아니
면, FALSE 를 얻는다.
    return FALSE;
}

int Stack_size(Stack* _this) {  // 스택의 크기 리턴
    return _this->_size;
}

Element Stack_push(Stack* _this, Element anElement){    // 스택에
anElement 를 삽입
    Node* addedNode = Node_new();

    if( Stack_isFull(_this)) {
        return FALSE;
    }
    Node_setElement(addedNode,anElement);
    Node_setNext(addedNode,_this->_top);
    _this->_top = addedNode;
    _this->_size++;

    return TRUE;
}

Element Stack_pop(Stack* _this){    // 스택의 top에서 원소를 삭제하고 그
값을 얻는다.
```

```
    Element removedElement;
    Node* removedNode = _this->_top;

    if( Stack_isEmpty(_this)) {

    }
    removedElement = Node_element(removedNode);
    _this->_top = Node_next(removedNode);
    Node_delete(removedNode);
    _this->_size--;

    return removedElement;
}

Element Stack_peekElement(Stack* _this) {   // top에 있는 원소를 리턴
    return Node_element(_this->_top);
}

Element Stack_elementAt(Stack* _this, int aPosition) {   //스택 리스
트의 aPosition 번째 원소를 얻는다.
    Element top;
    Node* positionNode = _this->_top;

    for( int Position=0; Position < aPosition; Position++ ) {
        positionNode = Node_next(positionNode);
    }

    top = Node_element(positionNode);

    return top;
}

void Stack_deleteLinkedNodes(Stack* _this) {
    Node* currentNode = _this->_top;
    Node* nodeToBeDeleted = NULL;

    while( currentNode != NULL ) {
        nodeToBeDeleted = currentNode;
        currentNode = Node_next(currentNode);
        Node_delete(nodeToBeDeleted);
    }
}

Node* Stack_top(Stack* _this) { // top 리턴
    return _this->_top;
}
```

7) Stack.h

```
//
```

```
//  Stack.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 30..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Stack_h
#define Stack_h
#include <stdio.h>
#include "Common.h"
#include "Node.h"

typedef struct _Stack Stack;


Stack* Stack_new(); // stack 객체 생성
void Stack_delete(Stack* _this);    // stack 객체 소멸
Boolean Stack_isEmpty(Stack* _this);    // 스택이 empty 이면 TRUE를,
아니면, FALSE를 얻는다.
Boolean Stack_isFull(Stack* _this); // 스택이 full 이면 TRUE 를, 아니
면, FALSE 를 얻는다.
Element Stack_push(Stack* _this, Element anElement);    // 스택에
anElement 를 삽입
Element Stack_pop(Stack* _this);    // 스택의 top에서 원소를 삭제하고 그
값을 얻는다.
void Stack_reset(Stack* _this);    // 스택 초기화
int Stack_size(Stack* _this);   // 스택의 크기 리턴
Element Stack_elementAt(Stack* _this, int aPosition);  //스택 리스트의
aPosition 번째 원소를 얻는다.
Element Stack_peekElement(Stack* _this);   // top에 있는 원소를 리턴
Node* Stack_top(Stack* _this); // top 리턴

#endif /* Stack_h */
```

8) Postfix.c

```
//
//  Postfix.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Postfix.h"
#include "Stack.h"
#include "AppIO.h"
```

```c
#include <string.h>

int Postfix_inComingPrecedence(Postfix* _this, char aToken);
int Postfix_inStackPrecedence(Postfix* _this, char aToken);
void Postfix_showOStackAll(Stack* _this);   // OStack을 보여줌
void Postfix_showTokenAndStack(Postfix* _this, char currentToken);

struct _Postfix {
    int _maxNumberOfTokens;
    char* _expression;
    char* _infixExpression;
    int _evaluatedValue;
    Stack* _operandStack;
    AppIO* _appIO;
};

Postfix* Postfix_new(int givenMaxNumberOfTokens) {  // 객체 생성
    Postfix* _this = NewObject(Postfix);
    _this->_maxNumberOfTokens = givenMaxNumberOfTokens;
    _this->_expression = NewVector(char, givenMaxNumberOfTokens);
    _this->_infixExpression = NewVector(char,
givenMaxNumberOfTokens);
    _this->_operandStack = Stack_new(givenMaxNumberOfTokens);

    return _this;
}

void Postfix_delete(Postfix* _this) {    // 객체 소멸
    Stack_delete(_this->_operandStack);
    free(_this->_expression);
    free(_this);
}

void Postfix_setExpression(Postfix* _this, char* anExpression) {
    //계산할 postfix 수식인 expression[] 을 postfix 객체에 전달
    strcpy(_this->_expression,anExpression);
}

void Postfix_showTokenAndStack(Postfix* _this, char currentToken)
{

    printf("%c : Stack <Bottom> ",currentToken);
    for( int i=Stack_size(_this->_operandStack)-1; i>=0; i++ ) {
        float Stack = Stack_elementAt(_this->_operandStack, i);
        printf("%.1f ",Stack);
    }
    AppIO_printTop(_this->_appIO);
}

void Postfix_showOStackAll(Stack* _this) {  // OStack을 보여줌
    printf("<OStack> ");
```

```
    for( int i= Stack_size(_this)-1; i>=0; i-- ) {
        int Stack = Stack_elementAt(_this, i);
        printf("%c ",Stack);
    }
    printf("\n");
}

PostfixError Postfix_evaluate(Postfix* _this){   //현재 객체가 가지고
있는 postfix 수식을 계산하도록 지시
    int operand, operand1, operand2, calculated;
    char currentToken;
    int i = 0;
    Stack_reset(_this->_operandStack);

    while ( _this->_expression[i] != '\0' ) {
        currentToken = _this->_expression[i];
        if( currentToken >= '0' && currentToken <= '9' ){
            operand = ( currentToken - '0' );
            if( Stack_isFull(_this->_operandStack) ) {
                return PostfixError_ExpressionTooLong;
            }
            else {
                Stack_push(_this->_operandStack, operand);
            }
        }

        else {
            if( currentToken == '+') {  // 덧셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);
                    calculated = operand1 + operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else if ( currentToken == '-') {    // 뺄셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);
                    calculated = operand1 - operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else if ( currentToken == '*') {    // 곱셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
```

```
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);
                    calculated = operand1 * operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else if( currentToken == '/') { // 나눗셈
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);
                    if( operand2 == 0 ) {
                        return PostfixError_DivideByZero;
                    }
                    calculated = operand1 / operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else if ( currentToken == '%' ) {   // 나머지 연산
                if(Stack_size(_this->_operandStack) >= 2 ) {
                    operand2 = Stack_pop(_this->_operandStack);
                    operand1 = Stack_pop(_this->_operandStack);

                    calculated = operand1 % operand2;
                    Stack_push(_this->_operandStack, calculated);
                }
                else {
                    return PostfixError_OperandsTooFew;
                }
            }
            else {
                return PostfixError_UnknownOperator;
            }
        }
        Postfix_showTokenAndStack(_this,currentToken);
        i++;
    }// end of while
    if( Stack_size(_this->_operandStack) > 1 ) {
        return PostfixError_OperandsTooMany;
    }
    else if( Stack_size(_this->_operandStack) == 1 ) {
        _this->_evaluatedValue = Stack_pop(_this->_operandStack);
    }
    return PostfixError_None;
}
```

```
int Postfix_evaluatedValue(Postfix* _this){ // 계산된 결과 값을
Postifx 객체로부터 얻는다.
    return (_this->_evaluatedValue);
}

void Postfix_setInfix(Postfix* _this, char* newInfix){  // infix 설
정자
    _this->_infixExpression = newInfix;
}

Boolean Postfix_infixToPostfix(Postfix* _this){
    // infix -> postfix
    Stack* OStack = Stack_new();
    int i = 0;  // infix
    int p = 0;  // postfix
    char currentToken,poppedToken;

    if( _this->_infixExpression != NULL ) {
        while( _this->_infixExpression[i] != '\0' ) {
            currentToken = _this->_infixExpression[i++];

            if( isDigit(currentToken)) {
                _this->_expression[p++] = currentToken;
            }
            else {
                if( currentToken == ')' ) {
                    if( !(Stack_isEmpty(OStack))) {
                        poppedToken = Stack_pop(OStack);

                        while( poppedToken != '(' ) {
                            _this->_expression[p++] = poppedToken;
                            poppedToken = Stack_pop(OStack);
                        }
                    }
                    else {
                        return FALSE;
                    }
                }
                else {
                    int postfix_inComing =
Postfix_inComingPrecedence(_this, currentToken);

                    if( Stack_top(OStack) != NULL ) {
                        if( Postfix_inStackPrecedence(_this,
Stack_peekElement(OStack)) >= postfix_inComing ) {
                            _this->_expression[p++] =
Stack_pop(OStack);
                            Stack_push(OStack, currentToken);
                        }
                        else {
                            Stack_push(OStack, currentToken);
```

```
                        }
                    }
                    else {
                        Stack_push(OStack, currentToken);
                    }
                }
                Postfix_showOStackAll(OStack);
            }
        }
    }
    else {
        return FALSE;
    }
    while( Stack_top(OStack) != NULL ) {
        _this->_expression[p++] = Stack_pop(OStack);
    }
    return TRUE;
}

char* Postfix_postfix(Postfix* _this){
    return _this->_expression;
}

int Postfix_inComingPrecedence(Postfix* _this, char aToken) {
    // 각 연산자의 입력 토큰 상태의 우선 순위를 돌려준다.
    if( aToken == '(' ) {
        return 20;
    }
    else if( aToken == ')' ) {
        return 19;
    }
    else if( aToken == '^' ) {
        return 17;
    }
    else if( aToken == '*' ) {
        return 13;
    }
    else if( aToken == '/' ) {
        return 13;
    }
    else if( aToken == '%' ) {
        return 13;
    }
    else if( aToken == '+' ) {
        return 12;
    }
    else if( aToken == '-' ) {
        return 12;
    }
    else if( aToken == '$' ) {
        return 0;
```

```
    }
    else {
        return -1;
    }
}

int Postfix_inStackPrecedence(Postfix* _this, char aToken) {
    // 각 연산자의 스택 안에서의 우선 순위를 돌려준다.
    if( aToken == '^' ) {
        return 16;
    }
    else if( aToken == '*' ) {
        return 13;
    }
    else if( aToken == '/' ) {
        return 13;
    }
    else if( aToken == '%' ) {
        return 13;
    }
    else if( aToken == '+' ) {
        return 12;
    }
    else if( aToken == '-' ) {
        return 12;
    }
    else if( aToken == '$' ) {
        return 0;
    }
    else {
        return -1;
    }
}
```

9) Postfix.h

```
//
//  Postfix.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Postfix_h
#define Postfix_h

#include <stdio.h>
```

```
#include "Common.h"
typedef struct _Postfix Postfix;

Postfix* Postfix_new(int givenMaxNumberOfTokens);  // 객체 생성
void Postfix_delete(Postfix* _this);    // 객체 소멸
void Postfix_setExpression(Postfix* _this, char* anExpression); //
계산할 postfix 수식인 expression[] 을 postfix 객체에 전달
void Postfix_showTokenAndStack(Postfix* _this, char currentToken);
PostfixError Postfix_evaluate(Postfix* _this);   //현재 객체가 가지고
있는 postfix 수식을 계산하도록 지시
int Postfix_evaluatedValue(Postfix* _this); // 계산된 결과 값을
Postifx 객체로부터 얻는다.
void Postfix_setInfix(Postfix* _this, char* newInfix); // Postfix
객체에게 infix 수식을 전달
Boolean Postfix_infixToPostfix(Postfix* _this); // Postfix 객체가 가
지고 있는 infix 수식을 postfix 수식으로 변환할 것을 Postfix 객체에게 시킨다
char* Postfix_postfix(Postfix* _this);  // 변환된 postfix 수식을 얻는
다.

#endif /* Postfix_h */
```

10) Common.h

```
//
//  Common.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Common_h
#define Common_h

#include <stdlib.h>

#define NewVector(TYPE,SIZE) (TYPE*)malloc(sizeof(TYPE)*SIZE)
#define NewObject(TYPE) (TYPE*)malloc(sizeof(TYPE))
#define isDigit(CHAR) (('0' <= CHAR) && (CHAR <= '9'))
typedef enum {FALSE,TRUE} Boolean;  // FALSE와 TRUE 값을 갖는 Boolean
선언
typedef int Element;

typedef enum {
    PostfixError_None,
    PostfixError_ExpressionTooLong,
    PostfixError_OperandsTooMany,
    PostfixError_OperandsTooFew,
    PostfixError_UnknownOperator,
```

```
    PostfixError_DivideByZero
} PostfixError;


#endif /* Common_h */
```

11) Node.c

```
//
//  Node.c
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#include "Node.h"

Node* Node_new() {
    Node* _this = NewObject(Node);
    return _this;
}

void Node_delete(Node* _this) {
    free(_this);
}

void Node_setElement(Node* _this, Element newElement) {
    _this->_element = newElement;
}

Element Node_element(Node* _this){
    return _this->_element;
}

void Node_setNext(Node* _this, Node* newNext) {
    _this->_next = newNext;
}

Node* Node_next(Node* _this) {
    return _this->_next;
}
```

12) Node.h

```
//
//  Node.h
//  CP2_WEEK12_2
```

```
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Node_h
#define Node_h

#include <stdio.h>
#include "Common.h"

typedef struct _Node Node;

struct _Node {
    Element _element;
    Node* _next;
};

Node* Node_new();
void Node_delete();
void Node_setElement(Node* _this, Element newElement);
Element Node_element(Node* _this);
void Node_setNext(Node* _this, Node* newNext);
Node* Node_next(Node* _this);


#endif /* Node_h */
```

13) Message.h

```
//
//  Message.h
//  CP2_WEEK12_2
//
//  Created by stu2017s10 on 2017. 5. 31..
//  Copyright © 2017년 stu2017s10. All rights reserved.
//

#ifndef Message_h
#define Message_h

#define ErrorMsg_ExpressionTooLong  "[오류] 수식이 너무 길어 처리가 불가능합니다.\n"
#define ErrorMsg_OperandsTooMany    "[오류] 연산자에 비해 연산값의 수가 많습니다.\n"
#define ErrorMsg_OperandsTooFew "[오류] 연산자에 비해 연산값의 수가 적습니다.\n"
#define ErrorMsg_UndefinedOperator  "[오류]수식에 알 수 없는 연산자가 있습니다.\n"
```

```
#define ErrorMsg_DivideByZero   "[오류] 나눗셈의 분모가 0 입니다.\n"

#endif /* Message_h */
```

3. 전체 설명

3-1. 수식 계산

1) main() 에서 AppController_new() 함수를 통해 appController 객체를 생성한다.

2) AppController_run(appController) 함수로 프로그램을 실행시킨다.

3) AppController_run() 에서는 AppIO_out_startingMessage(_this->_appIO) 함수가 시작 메시지를 출력하고, Postfix_new(MAX_NUMBER_OF_TOKENS) 함수를 통해 postfix 객체를 생성한다.

4) AppIO_in_postfixExpression() 함수를 통해 수식을 입력받고, 입력받은 수식을 nextPostfixExpressionIsAvailable에 저장한다.

5) while문을 통해 Postfix_evaluate() 함수가 계산한 수식의 오류를 검사하고, 오류가 있으면 AppController_out_postfixEvaluationErrorMessage() 함수로 오류 메시지를 출력한다. 그리고 $이 입력되지 않을 때까지 반복한다.

6) 반복이 종료되면 Postfix_delete() 객체가 postfix 객체를 소멸시킨다.

7) AppIO_out_endingMessage() 함수가 종료 메시지를 출력한다.