자료구조 프로그래밍 과제 2

제출일: 2022-10-27
학번/이름: 2016121150 / 윤준영

문제 1. 연결리스트를 사용하여 희소행렬을 저장하고, 이에 대한 다음 연산을 수행하는 프로그램을 작성하시오. 입력을 위한 희소 행렬은 파일에 저장되어 있으며, 읽어들인 희소 행렬에 추가적인 행렬 덧셈 연산을 수행한다. 유지되는 희소 행렬은 덧셈의 결과가 유지된다. 수행 시 처음 요청 작업은 "입력"으로 시작한다. 정수형만 사용한다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

typedef struct _node {
    /* define node struct */
    int row , col , val ;
    struct _node *next ;
} node ;

node * list_init (void ){
    /* initialize list
       return : list head node
    */
    node *head = (node *)malloc (sizeof (node ));
    node *tail = (node *)malloc (sizeof (node ));
    if (head ==NULL || tail ==NULL ) return NULL ;
    // if memory allocation fail, return NULL
    head ->next = tail ;
    tail ->next = tail ;
    return head ;
}

void list_print (node * h ){
    /* print list head to tail*/
    node * n ;
    n = h ;
    while (n ->next != n ){ // if not tail
       printf ("%d %d %d ₩n ", n ->row , n ->col , n ->val );
       n = n ->next ;
    }
}
```

1) node : 먼저 행, 열, 값, 다음 노드포인터를 포함하는 노드를 정의한다.
2) list_init : 리스트를 생성하고 초기화하는 함수이다. head 노드와 tail 노드를 생성하고, head의 다음 노드는 tail, tail의 다음 노드는 tail로 설정한 후, head 노드를 반환한다.

3) list_print : 리스트의 head 노드를 입력받아 리스트의 요소(row, column, value)를 출력하는 함수이다. 노드 n을 head 노드로 설정하고, head 노드부터 tail 이전 노드까지 다음 노드로 넘어가며 리스트의 요소를 출력한다. head 노드부터 출력하는 이유는 이후에 희소행렬을 리스트로 표현할 때, 희소행렬의 첫 열은 그 행렬의 크기를 나타내며 정렬되지 않으므로 구분하기 위하여 head 노드에 희소행렬의 크기 정보를 저장하기 때문이다.

```
void list_append (node * h , int row , int col , int val , int mode ){
    /* append new value in list, ascending
      input : head node, row, column, value, mode
      mode = 0(file load), 1(add sparse matrix)
    */
    node *s = h ;
    node *p = h ->next ;
    node *new = (node *)malloc (sizeof (node ));
    // order : s >new >p
    new ->row = row ;
    new ->col = col ;
    new ->val = val ;
    // fill new node infos
    while (p ->next != p ){
      // until p = tail
      if ((p ->row <row ) || ((p ->row == row ) &&(p ->col <col ))){
        s = p ;
        p = p ->next ;
      } // if p's row, column is smaller check next node
      else {
        break ;
      } // else break
    }
    if (mode &&((p ->col == col ) &&((p ->row == row )))){
      p ->val += val ;
      free (new );
      // in adding mode, if new row, column is same with p
      // add values and free new node
      if (p ->val == 0 ){
        s ->next = p ->next ;
        free (p );
        h ->val -= 1 ;
      } // if added value is 0, delete node
    }
    else {
      new ->next = p ;
      s ->next = new ;
      if (mode ) h ->val += 1 ;
    } // insert new node between s and p
     // if adding mode, num of value(head->val) += 1
}
```

4) list_append : 리스트의 head 노드와 새로운 행의 정보를 입력받아 오름차순으로 리스트에 추가하는 함수이다. 오름차순으로 정렬하기 위하여 new 노드를 할당하고, 탐색하려는 노드 p와 그 뒤 노드인 s 노드를 지정하여 하나씩 비교하여 다음 노드로 넘어간다. 만약 p의 row가 새로운 row보다 크거나, row는 같지만 column이 새로운 column보다 크면 s노드와 p노드 사이에 new노드를 추가한다. 덧셈연산에서도 사용하기 위하여, 추가하려는 row, column의 값이 이미 리스트에 존재하면, 있는 value에 새로운 value를 더하여 저장한다. 또한 mode 변수를 지정하여 mode가 1이면, 새로운 row, column이 추가되어 새로운 노드가 생성되면 head 노드의 value를 +1하고, 노드의 value가 0이 되면, 노드를 삭제시키고 head 노드의 value를 –1한다.

```
node * smat_load (char * str , node * h ){
   /* load sparse matrix file to list
     input : filename, list head node
     return : list head node
   */
   FILE * fp ;
   char line [MAX ];
   int last = 0 ;
   int row , col , val ;
   if ((fopen_s (&fp , str , "rb")) != NULL ){
     printf ("파일이 없습니다 ₩n ");
     return h ;
   }
   /* first line -> head node */
   fgets (line , MAX , fp );
   if (line [strlen (line )-1 ] != 10 ) last = 1 ; // lastline == {CONTENTS, NULL}
    else line [strlen (line )-2 ] = 0 ; // not lastline == {CONTENTS,CR,LF,NULL} make
{CONTENTS,NULL}
   h ->row = atoi (strtok (line , " ")); // strtok separates string by " "
   h ->col = atoi (strtok (NULL , " ")); // head node contains matrix size
   h ->val = atoi (strtok (NULL , " ")); // and number of non 0 value
   /* second to last line -> append(sorted) */
   while (!last ){
     fgets (line , MAX , fp );
     if (line [strlen (line )-1 ] != 10 ) last = 1 ; // lastline == {CONTENTS, NULL}
      else line [strlen (line )-2 ] = 0 ; // not lastline == {CONTENTS,CR,LF,NULL} make
{CONTENTS,NULL}
     row = atoi (strtok (line , " ")); // strtok separates string by " "
     col = atoi (strtok (NULL , " "));
     val = atoi (strtok (NULL , " "));
     list_append (h , row , col , val , 0 ); // append to list (mode 0)
   }
   fclose (fp );
   return h ;
}
```

5) smat_load : 파일을 통해 희소행렬을 입력받아 리스트로 생성하는 함수이다. 입력은 파일의 이름과 리스트의 head 노드를 받아 리스트 생성 후 head 노드를 반환한다. 파일의 첫 줄은 희소행렬의 크기를 나타내므로, 리스트에 정렬하지 않고 head 노드에 입력한다. 그 후 한줄씩 읽어 strtok을 이용하여 문자열을 " "로 구분한 뒤, list_append함수를 이용하여 오름차순으로 리스트에 삽입한다.

```
void smat_save (char *str , node * h ){
    /* load sparse matrix file to list
      input : filename, list head node
    */
    FILE * fp ;
    node * n = h ;
    char temp [MAX ];
    if ((fopen_s (&fp , str , "wb")) != NULL ){
      printf ("파일 저장 실패 ₩n ");
      return ;
    }
    /* first to last-1 line */
    while (n ->next ->next != n ->next ){
      // make string with row column value and new_line and write to file
      sprintf (temp , "%d %d %d ₩n ", n ->row , n ->col , n ->val );
      fputs (temp , fp );
      n = n ->next ;
    }
    /* last line */
    // make string with row column value and write to file
    sprintf (temp , "%d %d %d ", n ->row , n ->col , n ->val );
    fputs (temp , fp );
    fclose (fp );
}

void smat_print (node *h ){
    /* print sparse matrix list with matrix form */
    node *n = h ->next ;
    int row = h ->row ;
    int col = h ->col ;
    int *mat = (int *)calloc (row *col , sizeof (int ));
    while (n ->next != n ){
      mat [col *n ->row + n ->col ] = n ->val ;
      n = n ->next ;
    } // make matrix with list
    for (int i = 0 ; i <row ; i ++){
      for (int j = 0 ; j <col ; j ++) printf ("%d ", mat [i *col +j ]);
      printf ("₩n ");
    } // print matrix[i*col+j] i=0~row-1, j=0~col-1
}
```

6) smat_save : 리스트에 저장되어 있는 희소행렬을 파일로 저장하는 함수이다. 입력은 파일의 이름과 리스트의 head 노드를 받는다. head 노드부터 tail-2 노드까지 sprintf함수를 이용하여 temp 문자열에 row column value "₩n"을 저장한 후 파일에 입력하고, tail-1 노드의 경우 마지막 "₩n"을 생략하고 저장한다.

7) smat_print : 리스트에 저장되어 있는 희소행렬을 행렬의 형태로 출력하는 함수이다. head 노드를 입력받아 head의 row와 column의 크기로 배열을 생성한다. 그 후 head 다음 노드를 n 노드로 지정한 후, 순차적으로 node를 이동하며 tail의 이전 노드까지 배열의 n->row*h->col + n->col의 요소를

n->val로 입력한다. 그 후 행렬을 이중 반복문을 이용하여 출력한다.

```
node * smat_add (node *h1 , node *h2 ){
    /* add 2 sparse matrix with list form
       input : list1 head node, list2 head node
       return : list1 head node(summed)
    */
    if (!(h1 ->row == h2 ->row ) &&(h1 ->col == h2 ->col )){
       printf ("행렬의 크기가 다릅니다!₩n ");
       return h1 ;
    } // if matrix size does not match
    node * n ;
    n = h2 ->next ;
    while (n ->next != n ){
       list_append (h1 , n ->row , n ->col , n ->val , 1 );
       n = n ->next ;
    } // append head to tail (except head and tail) (add mode)
    return h1 ;
}

void input (char * s ){
    /* get string and delete new_line */
    fgets (s , MAX , stdin );
    if (s [strlen (s )-1 ] == '₩n ') s [strlen (s )-1 ] = '₩0 ';
}

int option (void ){
    /* get integer using input function */
    char s [5 ];
    input (s );
    return atoi (s );
}
```

8) smat_add : 희소행렬이 저장된 두 리스트를 입력받아 두 희소행렬을 더하는 함수이다. head1 노드와 head2 노드를 입력받아 계산 결과를 head1의 리스트에 저장한다. 만약 두 행렬의 크기가 다를 경우 경고문과 함께 h1 노드를 반환한다. 노드 n을 h2->next로 지정한 후 tail 이전 노드까지 순차적으로 list_append (mode 1) 함수를 이용하여 head1의 리스트에 더한다. list_append는 같은 row column을 가진 요소가 존재할 경우 각 요소를 더하며, 존재하지 않을 경우 새로운 노드를 삽입한다.

9) input, option : input 함수는 문자열을 fgets로 입력받아 마지막 "₩n"을 "₩0"(NULL)로 바꿔주는 함수이며, option 함수는 input함수로 입력받은 문자열을 정수로 반환해주는 함수이다.

```
int main (void ){
    node *h1 = list_init ();
    node *h2 = list_init ();
    int c ;
    char file [MAX ];
    do {
       printf ("> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): ");
       c = option ();
```

```
      switch (c ){
        case 1 :
          printf ("파일 이름을 입력하세요: ");
          input (file );
          h1 = smat_load (file , h1 );
          printf ("희소 행렬을 생성했습니다.₩n ");
          list_print (h1 );
          break ;
        case 2 :
          printf ("파일 이름을 입력하세요: ");
          input (file );
          h2 = smat_load (file , h2 );
          printf ("더할 희소 행렬입니다.₩n ");
          list_print (h2 );
          h1 = smat_add (h1 , h2 );
          printf ("더한 결과입니다.₩n ");
          list_print (h1 );
          break ;
        case 3 :
          printf ("희소 행렬을 행렬 형태로 출력합니다.₩n ");
          smat_print (h1 );
          break ;
        case 4 :
          printf ("파일 이름을 입력하세요: ");
          input (file );
          smat_save (file , h1 );
          printf ("희소 행렬을 파일에 저장했습니다.₩n ");
          break ;
        default :
          break ;
      }
    } while (c ); // if input is 0 or NULL break
    printf ("Bye~₩n ");
    return 0 ;
}
```

10) main : main 함수는 위와 같다. 먼저 두 리스트를 생성하여 각 head 노드를 h1, h2로 지정한다. 명령을 선택할 정수 c를 선언하고 파일 이름을 저장할 문자열 file을 선언한다. option 함수를 이용하여 명령을 입력받고, c에 따라 해당 명령을 수행한다.

c = 1 : 파일 이름을 입력받고, smat_load함수를 이용하여 해당하는 파일을 읽어 h1에 희소행렬을 표현한 리스트로 저장한다. 저장 후 list_print 함수를 이용하여 리스트를 출력한다.

c = 2 : 더할 파일 이름을 입력받고, smat_load함수로 h2에 희소행렬을 표현한 리스트로 저장한다. smat_add함수를 이용하여 h1과 h2 리스트를 더한 후 h1에 저장한다. list_print함수를 이용하여 h1을 출력한다.

c = 3 : smat_print함수를 이용하여 h1함수를 행렬 형태로 출력한다.

c = 4 : 파일 이름을 입력받고 smat_save함수를 이용하여 리스트를 파일로 저장한다.

c = 0 or NULL : 프로그램을 종료한다.

11) 실행 결과 : 함수의 실행 결과는 다음과 같다.

```
> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): 1
파일 이름을 입력하세요: array1.txt
희소 행렬을 생성했습니다.
4 6 5
0 4 2
1 1 3
2 2 -3
2 5 8
3 1 4
> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): 2
파일 이름을 입력하세요: array2.txt
더할 희소 행렬입니다.
4 6 4
1 2 4
1 4 3
2 2 3
2 5 3
더한 결과입니다.
4 6 6
0 4 2
1 1 3
1 2 4
1 4 3
2 5 11
3 1 4
> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): 3
희소 행렬을 행렬 형태로 출력합니다.
0 0 0 0 2 0
0 3 4 0 3 0
0 0 0 0 0 11
0 4 0 0 0 0
> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): 4
파일 이름을 입력하세요: array3.txt
희소 행렬을 파일에 저장했습니다.
> 원하는 작업을 선택하세요 (1:입력, 2:행렬 더하기, 3: 출력, 4:저장, 0:종료): 0
Bye~
```

터미널이 작업에서 다시 사용됩니다. 닿으려면 아무 키나 누르세요.

잘 실행되는 것을 확인할 수 있다.

array3.txt는 아래와 같이 잘 저장된 것을 확인할 수 있다.

array3.txt

```
1   4 6 6
2   0 4 2
3   1 1 3
4   1 2 4
5   1 4 3
6   2 5 11
7   3 1 4
```

문제 2. x에 대한 함수를 후위 표기법으로 입력받아 수식나무에 저장하고, 이에 대하여 함수를 주위 표기법으로 출력하거나, 값을 받아 결과를 출력하는 프로그램을 작성하시오. x의 값과 계산 결과는 모두 정수형으로 가정한다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

typedef struct _node *nodeptr ;
typedef struct _node {
    char val ;
    nodeptr left ;
    nodeptr right ;
} node ;
typedef struct _hnode {
    nodeptr stack [MAX ];
    int top ;
    nodeptr tree ;
} hnode ;

hnode * tree_init (void ){
    /* initialize list
       return : list head node
    */
    hnode *head = (hnode *)malloc (sizeof (hnode ));
    node *tail = (node *)malloc (sizeof (node ));
    if (head ==NULL || tail ==NULL ) return NULL ;
    // if memory allocation fail, return NULL
    head ->top = -1 ;
    head ->tree = tail ;
    tail ->left = tail ;
    tail ->right = tail ;
    return head ;
}

void input (char * s ){
    /* get string and delete new_line */
    fgets (s , MAX , stdin );
    if (s [strlen (s )-1 ] == '₩n ') s [strlen (s )-1 ] = '₩0 ';
}

int str2int (void ){
    /* get integer using input function */
    char s [5 ];
    input (s );
    return atoi (s );
}
```

```
int is_operator (int i ){
    return (i == '+' || i == '-' || i == '*' || i == '/');
} /* true if i is an operator, else 0 */

int num (int i ){
    if (i >= 48 &&i <= 57 ) return i -48 ;
    else return i ;
} /* make str number to int */

nodeptr pop (hnode *h ){
    if (h ->top == -1 ){
      printf ("Stack underflow!₩n ");
      return NULL ;
    }
    return h ->stack [h ->top --];
} /* pop top node, top-- */

void push (node *a , hnode *h ){
    if (h ->top == 99 ){
      printf ("Stack overflow!₩n ");
      return ;
    }
    h ->stack [++h ->top ] = a ;
} /* top ++, push node to top*/

void parse_tree (char *s , hnode *h ){
    node *n ;
    node *tail = h ->tree ;
    h ->top = -1 ;
    while (*s ){
      while (*s == ' ') s ++; // skip ' '
      n = (node *)malloc (sizeof (node ));
      n ->val = *s ;
      n ->left = tail ;
      n ->right = tail ;
      if (is_operator (*s )){
        n ->right = pop (h );
        n ->left = pop (h );
      }
      push (n , h );
      s ++;
    }
    h ->tree = pop (h );
}
```

1) hnode : head 노드는 수식나무를 생성하기 위한 node 포인터의 스택인 stack, 스택의 가장 위를 나타내는 top, 수식나무의 시작노드를 나타내는 node 포인터 tree로 이루어진다.

2) node : node의 값을 나타내는 문자 val, left와 right child를 나타내는 node 포인터 left, right로 이루

어진다.

3) tree_init : 리스트를 생성하고 초기화하는 함수이다. head 노드(hnode)와 tail 노드를 생성하고, head의 tree는 tail, tail의 left, right 노드는 tail로 설정한 후, head 노드(hnode)를 반환한다.

4) input, str2int : 문제 1에서의 input, option과 같다.

5) is_operator : 입력받은 문자가 연산자일 경우 1을 반환하고, 아니면 0을 반환한다.

6) num : 입력받은 문자가 숫자일 경우, 정수 형태로 숫자를 반환한다.

7) pop, push : head노드를 입력받아 head에 포함된 stack을 pop 또는 push하는 함수이다. pop의 경우 stack[top]의 노드를 빼낸 후, top을 1 감소시킨다. 빼낸 노드를 반환한다. push의 경우 top을 1 증가시킨 후, stack[pop]에 입력받은 노드를 저장한다.

8) parse_tree : 입력받은 문자열을 수식나무 리스트에 저장하는 함수이다. 문자열과 리스트의 head 노드를 입력받으면, tail 노드를 head의 next로 지정한 후, 한 문자씩 수식나무에 저장한다. 만약 문자가 공백일 경우 다음 문자로 넘어간다. 노드 n을 할당한 후, 문자를 val에 저장한다. 만약 문자가 연산자가 아닐 경우 스택에 push한다. 연산자일 경우 스택에 저장되어 있는 문자를 두 번 pop하여 각각 right와 left노드로 저장한다. 문자열이 끝날 경우 while문을 빠져나오고, 이 때 스택에 가장 상위의 subtree가 저장되어 있다. 이 tree를 head의 tree로 저장한다.

```
int operate (node *n , int l , int r ){
    l = l -48 ;
    r = r -48 ;
    if (n ->val == '+') return l + r + 48 ;
    if (n ->val == '-') return l - r + 48 ;
    if (n ->val == '*') return l * r + 48 ;
    if (n ->val == '/') return l / r + 48 ;
}

int parse_tree_calc (int x , node *t ){
    /* 1. if t is not leaf (if t->left and t-> right is not same(tail)),
        then it is operator: calculate chilren first
       2. if t is a leaf:
        2-1. if t->val is x, return x
        2-2. if t->val is number, return number
    */
    if (t ->left != t ->right ){
      int l = parse_tree_calc (x , t ->left );
      int r = parse_tree_calc (x , t ->right );
      return operate (t , l , r );
    }
    if (t ->val == 'x') return x +48 ;
    else {
      return t ->val ;
    }
}
```

9) operate : 노드의 val를 연산자로 l과 r값을 연산하는 함수이다. l과 r은 문자로 저장되어 있으므로 각각 –48을 한 후, 결과값을 +48하여 문자로 반환한다.

10) parse_tree_calc : 수식나무를 계산하는 함수이다. x와 tree를 입력받아 저장되어 있는 x를 대입하여 계산한다. 후위순회를 이용하여 계산한다. 재귀함수를 이용하여, 만약 가장 하위 노드가 아닐 경우, left subree를 먼저 parse_tree_calc를 이용하여 계산한 후, right subtree를 계산한다. 만약 가장 하위 노드일

경우, 노드의 값이 x일 경우 x에 해당하는 값을 반환하고, 아닐 경우 현재의 val을 반환한다.

```
int order (int s ){
    /* return order of operator
       order : num >>* >/ >>+ >-
    */
    if (is_operator (s )){
      if (s == '*') return 5 ;
      if (s == '/') return 4 ;
      if (s == '+') return 2 ;
      if (s == '-') return 1 ;
    }
    return 9 ;
}

void parse_tree_inorder (node *t , int b ){
    if (!((t ->left == t ) &&(t ->right == t ))){
      if (b ) printf ("( ");
      parse_tree_inorder (t ->left , order (t ->val ) >(order (t ->left ->val )+1 ));
      printf ("%c ", t ->val );
      parse_tree_inorder (t ->right , order (t ->val ) >= order (t ->right ->val ));
      if (b ) printf (") ");
    }
}
```

11) order : 중위표기법으로 출력할 때, 괄호를 입력하기 위하여 연산의 우선순위를 출력한다. num >> * > / >> + > - 순서로 우선순위를 지정하였다.

12) parse_tree_inorder : 중위순회를 이용하여 중위표기법을 출력한다. 재귀함수를 이용하여 left subtree를 중위표기법으로 출력하고, 현재 노드를 출력하고 right subtree를 중위표기법으로 출력한다. 정수 b가 1이면 괄호를 출력하고, b가 0이면 출력하지 않는다. left와 right subtree를 출력할 때, 현재 노드(n)와 좌측 노드(l), 우측 노드(r)의 order을 비교하여 order(n)>(order(l)+1)의 결과가 true이면 left subtree에 괄호를 입력하고, order(n)>=order(r)의 결과가 true이면 right subtree에 괄호를 출력한다.

```
int main (void ){
    hnode *h1 = tree_init ();
    int c ;
    char fx [MAX ];
    int x ;
    int result ;
    do {
      printf ("> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): ");
      c = str2int ();
      switch (c ){
        case 1 :
          printf ("함수 f(x)를 후위식으로 입력하세요: ");
          input (fx );
          parse_tree (fx , h1 );
          break ;
```

```
        case 2 :
          printf ("x값을 을 입력하세요: ");
          x = str2int ();
          result = parse_tree_calc (x , h1 ->tree );
          printf ("f(%d ) = %d ₩n ", x , result -48 );
          break ;
        case 3 :
          printf ("f(x) = ");
          parse_tree_inorder (h1 ->tree , 0 );
          printf ("₩n ");
          break ;
        default :
          break ;
      }
    } while (c != 4 ); // if input is 4
    printf ("Bye~₩n ");
    return 0 ;
}
```

13) main : 먼저 트리를 초기화하여 head node를 h1으로 지정한다. 명령 입력을 위한 정수 c와 f(x)를 입력할 문자열 fx, f(x)를 계산할 x값 x, 결과를 출력할 result를 초기화한다. str2int 함수를 이용하여, 명령을 입력받고, switch를 이용하여 입력에 해당하는 명령을 수행한다.

c = 1 : 함수 f(x) 문자열을 입력받고, parse_tree함수로 fx 문자열에 대한 수식 나무를 생성한다.

c = 2 : x값을 입력받아 parse_tree_calc함수로 수식나무에 저장되어 있는 x에 대입하여 계산한다.

c = 3 : parse_tree_inorder함수를 이용하여 수식나무에 저장되어 있는 수식을 중위식으로 출력한다.

c = 4 : 프로그램을 종료한다.

14) 실행 결과 : 프로그램의 실행 결과는 다음과 같다.

```
> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): 1
함수 f(x)를 후위식으로 입력하세요: x x * x 2 - * 3 x * + x 1 - -
> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): 2
x값을 을 입력하세요: 3
f(3) = 16
> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): 2
x값을 을 입력하세요: 4
f(4) = 41
> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): 3
f(x) = x * x * ( x - 2 ) + 3 * x - ( x - 1 )
> 원하는 연산을 선택하세요 (1:f(x)입력, 2:함수값 계산, 3:중위식 출력, 4:종료): 4
Bye~
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

입력한 수식에 대해 함수값 계산과 중위식 출력이 잘 되는 것을 확인할 수 있다.