プログラミング言語実験・C 言語　第 2 回課題レポート

I 類 メディア情報学
氏名：LEORA DAVID
学籍番号：2210745

2024 年 04 月 24 日

課題３（逆ポーランド記法）

被演算数、演算子、および括弧のトークンで構成された中置記法の算術式を読み込み、スタックを用いて逆ポーランド記法の式を出力する C プログラムを作成した。それから、次の（1）～（3）の中置記法の算術式に対する実行結果を求めた。

(1) A = (B - C / D + E) * F
(2) A = B - (C / D + E) * F
(3) A = B - C / (D + E * F)

また、プログラム中で使用する演算子は、二項演算しの加算「+」、減算「-」、乗算「*」、除算「/」、および代入「=」だけに限定することにした。最後に、単項演算子の加減算を追加した。

ソースコード 1　reversePolishNotation.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4
5   #define SIZE 100
6   #define TRUE 1
7   #define FALSE 0
8   #define SUCCESS 1
9   #define FAILURE 0
10
11  typedef char data_type;
12  typedef struct node_tag {
13      data_type data;
14      struct node_tag *next;
15  } node_type;
16
17  // スタックを初期化します
18  void initialize(node_type **pp){
19      *pp = NULL; // スタックの先頭をNULL に設定します（空の状態にします）
20  }
21
22  // スタックが空かどうかを確認します
23  int is_empty(node_type *p){
24      return (p == NULL) ? TRUE : FALSE;
```

```
25      }
26
27      // スタックの先頭にあるデータを取得します
28      data_type top(node_type *p){
29          if (is_empty(p)){ // スタックが空の場合、文字を返します NULL
30              printfスタックは空です ("\n");
31              return '\0';
32          }
33          return p->data; // 空出ない場合、先頭のデータを返します
34      }
35
36      // 新しいノードを作成し、スタックに追加します
37      node_type *new_node(data_type x, node_type *next) {
38          node_type *temp = (node_type *)malloc(sizeof(node_type));
39          if (temp != NULL) { // メモリの割り当てが成功した場合、データを設定し、次のノードを
                設定します
40              temp->data = x;
41              temp->next = next;
42          }
43          return temp;
44      }
45
46      // スタックにデータを追加します
47      int push(data_type x, node_type **pp){
48          node_type *temp = new_node(x, *pp); // 新しいノードを作成します
49          if (temp == NULL) return FAILURE;
50          *pp = temp; // スタックの先頭を新しいノードに設定します
51          return SUCCESS;
52      }
53
54      // スタックからデータをポップします
55      int pop(node_type **pp)
56      {
57          node_type *temp;
58
59          if (!is_empty(*pp)){ // スタックが空でない場合、先頭のノードを削除します
60              temp = (*pp)->next; // 次のノードを一時的に保存します
61              free(*pp); // 現在の先頭のノードを削除します
62              *pp = temp; // スタックの先頭を次のノードに設定します
63              return SUCCESS;
64          }
65          return FAILURE;
66      }
67
68      // 与えられた演算子の優先順位を返します
69      int checkPriority(char a){
```

```
70          int priority;
71          switch (a) {
72              case '=':
73                  return 0;
74              case ')' :
75                  return 1;
76              case '+' :
77                  return 2;
78              case '-' :
79                  return 2;
80              case '*' :
81                  return 3;
82              case '/' :
83                  return 3;
84              case '(' :
85                  return 4;
86              default:
87                  return 5;
88              }
89      }
90
91      // 現在の文字が単項演算子かどうかを確認します
92      int isOperator(char previous, char current){
93          // 前の文字が演算子または開き括弧であり、現在の文字が単項演算子の場合、TRUE を返します
94          if ((previous == '=' || previous == '(' || previous == '+' || previous == '-'
                || previous == '*' || previous == '/') && (current == '-')) {
95              return TRUE;
96          } else return FALSE;
97      }
98
99      int main() {
100         node_type *stack; // スタックを宣言します
101         initialize(&stack); // スタックを初期化します
102         char input[SIZE], previousChar = 0; // 入力文字列と前の文字を格納する変数を宣言し
                ます
103         int track = 0; // 入力文字列の位置を追跡する変数を宣言します
104
105         printf式を入力してください：("\n");
106         fgets(input, SIZE, stdin); // 入力文字列を取得します
107         int len = strlen(input);
108         if(input[len-1] == '\n') // 改行文字を削除します
109             input[len-1] = '\0'; // 文字に置き換えます NULL
110
111         while (input[track] != '\0') { // 入力文字列の終わりに達するまで繰り返します
112             if(input[track] == ' '){ // 空白文字をスキップします
113                 track++;
```

```
114                 continue;
115             }
116
117             if(isOperator(previousChar, input[track])){ // 単項演算子の判定を行います
118                 track++;
119                 push('~', &stack);
120                 push(input[track], &stack);
121                 previousChar = input[track]; // 前の文字を更新します
122                 track++;
123                 continue;
124             }
125
126             // スタックの先頭にある演算子の優先順位が現在の文字の優先順位よりも高い場合、スタックから演算子を取り出し、出力します
127             while(!is_empty(stack) && top(stack) != '(' && checkPriority(input[track])
                    <= checkPriority(top(stack))) {
128                 printf("%c", top(stack));
129                 pop(&stack);
130             }
131
132             // 現在の文字が閉じ括弧の場合、開き括弧が見つかるまでスタックから演算子を取り出し、出力します
133             if (input[track] != ')') {
134                 push(input[track], &stack);
135             }
136             else { // 閉じ括弧が見つかった場合、開き括弧が見つかるまでスタックから演算子を取り出し、出力します
137                 while (!is_empty(stack) && top(stack) != '('){
138                     printf("%c", top(stack));
139                     pop(&stack);
140                 }
141                 if (!is_empty(stack) && top(stack) == '(') {
142                     pop(&stack);
143                 }
144             }
145             previousChar = input[track]; // 前の文字を更新します
146             track++;
147         }
148
149         while(!is_empty(stack)){ // スタックに残っている演算子を取り出し、出力します
150             if(top(stack) != '\0') {
151                 printf("%c", top(stack));
152             }
153             pop(&stack);
154         }
155         printf("\n");
156         return 0;
```

```
157     }
```

課題３の実行結果

1. 中置記法の算術式：A=(B-C/D+E)*F、実行結果：ABCD/-E+F*=
2. 中置記法の算術式：A=B-(C/D+E)*F、実行結果：ABCD/E+F*-=
3. 中置記法の算術式：A=B-C/(D+E*F)、実行結果：ABCDEF*+/-=
4. 中置記法の算術式：A=((-B)-(-C)/(-D)+(-E))*(-F)、実行結果：AB-C-D-/-E-+F-*=
5. 中置記法の算術式：A=B-((-C)/(-D)+(-E))*(-F)、実行結果：ABC-D-/E-+F-*-=
6. 中置記法の算術式：A=B-(-C)/((-D)+(-E)*(-F))、実行結果：ABC-D-E-F-*+/-=

課題４（二分探索木）

入力として整数（int 型）のデータが与えられ、これらのデータには同じ整数が重複して出現してもよいものとする。このとき、ポインタによる二分探索木を実現して、整数データを二分探索木に挿入した後、この二分探索木に挿入されたデータを取り出して、データを照準（厳密には非減少順）に出力する C のプログラムを作成した。ただし、同じ値の整数データが複数存在する場合には、入力時と同じ個数分出力することにした。

ここで、ポインタによる二分探索木の実現に際して、各ノードは、整数データを保持する int 型変数 data と、その整数が出現した頻度（度数）を保持する int 型変数 frequency をもつものとする。最後に、データの削除も行えるように改良した。

ソースコード 2　binarySearchTree.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define TRUE 1
5  #define FALSE 0
6  #define SUCCESS 1
7  #define FAILURE 0
8
9  typedef int data_type;
10 typedef int freq_type;
11 typedef struct node_tag {
12     data_type data;
13     freq_type freq;
14     struct node_tag *left;
15     struct node_tag *right;
16 } node_type;
17
18 void initialize(node_type **pp) {
19     *pp = NULL;
20 }
21
22 int is_member(data_type x, node_type *p) {
23     if (p == NULL) return FALSE;
24     else {
25         if(x == p->data) return TRUE;
26         else {
27             if (x < p->data) return is_member(x, p->left);
28             else return is_member(x, p->right);
29         }
30     }
31 }
```

```
32
33 data_type min(node_type *p) {
34     while (p->left != NULL) {
35         p = p->left;
36     }
37     return p->data;
38 }
39
40 node_type *new_node(data_type x) {
41     node_type *temp = (node_type *)malloc(sizeof(node_type));
42     if (temp == NULL) return NULL;
43     else {
44         temp -> data = x;
45         temp -> freq = 1;
46         temp -> left = NULL;
47         temp -> right = NULL;
48         return temp;
49     }
50 }
51
52 int insert (data_type x, node_type **pp) {
53     if (*pp == NULL) {
54         node_type *temp = new_node(x);
55         if (temp == NULL) return FAILURE;
56         *pp = temp;
57         return SUCCESS;
58     } else {
59         if (x < (*pp)->data) return insert (x, &((*pp)->left));
60         else if (x > (*pp)->data) return insert(x, &((*pp)->right));
61         else {
62             (*pp)->freq++;
63             return SUCCESS;
64         }
65     }
66 }
67
68 /** 未実装：データの削除（予定）
69 int delete (data_type x, node_type **pp) {
70
71 }
72 **/
73
74 void print_in_order(node_type *p) {
75     if(p != NULL) {
76         print_in_order(p->left);
77         for (int i = 0; i < p->freq; i++){
```

```
78              printf("%d ", p->data);
79          }
80          print_in_order(p->right);
81      }
82  }
83
84  // プログラム終了時に、動的に割り当てたメモリを開放する処理
85  void free_tree(node_type *p) {
86      if (p != NULL) {
87          free_tree(p->left);
88          free_tree(p->right);
89          free(p);
90      }
91  }
92
93  int main(){
94      node_type *root;
95      initialize(&root);
96      int choice, x;
97
98      while(TRUE) {
99          printf("\数値を入力してください（プログラム修了：－１）：n");
100         scanf("%d", &x);
101         if (x == -1) break;
102         if (insert(x, &root) == SUCCESS) {
103             printf挿入が成功しました("\n");
104             printf木の中身：("");
105             print_in_order(root);
106             printf("\n");
107         } else {
108             printf挿入が失敗しました("\n");
109         }
110     }
111     free_tree(root);
112     printfプログラムを終了します。("\n");
113     return 0;
114 }
```

課題４の実行結果

参考文献

[1] 第 2 回　動的データ構造と再起処理（スタック、二分木）,
URL : https://www.ied.inf.uec.ac.jp/text/laboratory/C/second_week/index02.html