

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; // 空でない場合、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 == '-'
95         || previous == '*' || previous == '/') && (current == '-')) {
96         return TRUE;
97     } else return FALSE;
98 }
99
100 int main() {
101     node_type *stack; // スタックを宣言します
102     initialize(&stack); // スタックを初期化します
103     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
104     int track = 0; // 入力文字列の位置を追跡する変数を宣言します
105
106     printf式を入力してください: ("\n");
107     fgets(input, SIZE, stdin); // 入力文字列を取得します
108     int len = strlen(input);
109     if(input[len-1] == '\n') // 改行文字を削除します
110         input[len-1] = '\0'; // 文字に置き換えます NULL
111
112     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
113         if(input[track] == ' '){ // 空白文字をスキップします
            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
127     // ックから演算子を取り出し、出力します
128     while(!is_empty(stack) && top(stack) != '(' && checkPriority(input[track])
129           <= checkPriority(top(stack))) {
130         printf("%c", top(stack));
131         pop(&stack);
132     }
133
134     // 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
135     // し、出力します
136     if (input[track] != ')') {
137         push(input[track], &stack);
138     }
139     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
140     // り出し、出力します
141         while (!is_empty(stack) && top(stack) != '('){
142             printf("%c", top(stack));
143             pop(&stack);
144         }
145         if (!is_empty(stack) && top(stack) == '(') {
146             pop(&stack);
147         }
148     }
149     previousChar = input[track]; // 前の文字を更新します
150     track++;
151 }
152
153 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
154     if(top(stack) != '\0') {
155         printf("%c", top(stack));
156     }
157     pop(&stack);
158 }
159
160 printf("\n");
161 return 0;

```

課題 3 の実行結果

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. 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： $ABCDEF*+/-$

単項演算子の加減算を追加した場合、次の (4)～(6) の中置記法の算術式に対する実行結果を求めた。

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
12 // ノードの構造を定義するための構造体です。
13 typedef struct node_tag {
14     data_type data;
15     freq_type freq;
16     struct node_tag *left;
17     struct node_tag *right;
18 } node_type;
19
20 // 引数として与えられたノードのポインタをに初期化します NULL
21 void initialize(node_type **pp) {
22     *pp = NULL;
23 }
24
25 // 指定された値が二分探索木に存在するかどうかを判定します
26 int is_member(data_type x, node_type *p) {
27     if (p == NULL) return FALSE;
28     else {
29         if(x == p->data) return TRUE;
30         else {
31             if (x < p->data) return is_member(x, p->left);
```

```

32         else return is_member(x, p->right);
33     }
34 }
35 }
36
37 // 与えられたノードの中で最小値を探します
38 data_type min(node_type *p) {
39     while (p->left != NULL) {
40         p = p->left;
41     }
42     return p->data;
43 }
44
45 // 新しいノードを作成します
46 node_type *new_node(data_type x) {
47     node_type *temp = (node_type *)malloc(sizeof(node_type));
48     if (temp == NULL) return NULL; // メモリが割り当てられなかった場合
49     else {
50         temp->data = x; // データを設定
51         temp->freq = 1; // データの出現回数を設定
52         temp->left = NULL; // 左の子ノードをに設定 NULL
53         temp->right = NULL; // 右の子ノードをに設定 NULL
54         return temp; // 新しいノードを返す
55     }
56 }
57
58 // 二分探索木にデータを挿入します
59 int insert (data_type x, node_type **pp) {
60     if (*pp == NULL) { // ノードが空の場合
61         node_type *temp = new_node(x);
62         if (temp == NULL) return FAILURE;
63         *pp = temp;
64         return SUCCESS;
65     } else { // ノードが空でない場合
66         if (x < (*pp)->data) return insert (x, &((*pp)->left)); // 左の子ノードに
        挿入
67     else if (x > (*pp)->data) return insert(x, &((*pp)->right)); // 右の子ノード
        に挿入
68     else {
69         (*pp)->freq++; // 既に存在するデータの出現回数を増やす
70         return SUCCESS;
71     }
72 }
73 }
74
75 // 二分探索木からデータを削除します

```

```

76 int delete (data_type x, node_type **pp) {
77     if (*pp == NULL) return FAILURE; // ノードが空の場合
78     else if (x < (*pp)->data) { // 左の子ノードを探索
79         return delete(x, &((*pp)->left));
80     } else if (x > (*pp)->data) { // 右の子ノードを探索
81         return delete(x, &((*pp)->right));
82     } else { // ノードが見つかった場合
83         if ((*pp)->left == NULL && (*pp)->right == NULL) { // 子ノードがない場合
84             free(*pp);
85             *pp = NULL;
86             return SUCCESS;
87         } else if ((*pp)->left == NULL) { // 左の子ノードがない場合
88             node_type *temp = *pp;
89             *pp = (*pp)->right;
90             free(temp);
91             return SUCCESS;
92         } else if ((*pp)->right == NULL) { // 右の子ノードがない場合
93             node_type *temp = *pp;
94             *pp = (*pp)->left;
95             free(temp);
96             return SUCCESS;
97         } else { // 両方の子ノードがある場合
98             (*pp)->data = min((*pp)->right); // 右の子ノードの最小値を取得
99             return delete((*pp)->data, &((*pp)->right)); // 右の子ノードから最小値を
              削除
100         }
101     }
102 }
103
104 // 二分探索木を中順で出力します
105 void print_in_order(node_type *p) {
106     if(p != NULL) { // ノードが空でない場合
107         print_in_order(p->left);
108         for (int i = 0; i < p->freq; i++){ // 出現回数分だけデータを出力
109             printf("%d ", p->data); // データを出力
110         }
111         print_in_order(p->right); // 右の子ノードを探索
112     }
113 }
114
115 // 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
116 void free_tree(node_type *p) {
117     if (p != NULL) {
118         free_tree(p->left); // 左の子ノードを開放
119         free_tree(p->right); // 右の子ノードを開放
120         free(p); // ノードを開放

```



```

121     }
122 }
123
124 int main(){
125     node_type *root; // 二分探索木のルートノード
126     initialize(&root); // ルートノードを初期化
127     int choice, x; // ユーザーの選択とデータを格納する変数
128
129     while(TRUE) { // ユーザーが終了するまで繰り返す
130         printf("\n1. Insert\n2. Delete\n3. Exit\nEnter your choice: ");
131         scanf("%d", &choice);
132
133         switch(choice) {
134             case 1: // 挿入
135                 printf("Enter the number to insert: ");
136                 scanf("%d", &x);
137                 if (insert(x, &root) == SUCCESS) {
138                     printf("=====\nInsertion was successful\n");
139                     printf("Tree contents: ");
140                     print_in_order(root);
141                     printf("\n=====\n");
142                 } else {
143                     printf("Insertion failed\n");
144                 }
145                 break;
146             case 2: // 削除
147                 printf("Enter the number to delete: ");
148                 scanf("%d", &x);
149                 int result = delete(x, &root);
150                 if (result == SUCCESS) {
151                     printf("=====\nDeletion was successful\n");
152                     printf("Tree contents: ");
153                     print_in_order(root);
154                     printf("\n=====\n");
155                 } else if (result == FAILURE) {
156                     printf("NUMBER NOT FOUND\n");
157                 } else {
158                     printf("Deletion failed\n");
159                 }
160                 break;
161             case 3: // 終了
162                 printf("=====\nExiting the program.\n\n");
163                 free_tree(root);
164                 return 0;
165             default: // 1, 2, 3 以外の場合

```

```

166             printf("Invalid choice. Please enter 1, 2, or 3.\n");
167         }
168     }
169 }

```

課題 4 の実行結果

プログラムの実行結果にはいくつかの例を載せる。

例 1. 挿入 23, 挿入 50, 挿入 34, 挿入 56, 挿入 98, 挿入 1, 挿入 15, 挿入 29, 挿入 33, 挿入 40

実行結果：

```

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 23
=====
Insertion was successful Tree contents: 23
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 50
=====
Insertion was successful Tree contents: 23 50
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 34
=====
Insertion was successful Tree contents: 23 34 50
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 56
=====
Insertion was successful Tree contents: 23 34 50 56
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 98
=====
Insertion was successful Tree contents: 23 34 50 56 98
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 1
=====
Insertion was successful Tree contents: 1 23 34 50 56 98
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 15
=====
Insertion was successful Tree contents: 1 15 23 34 50 56 98
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 29
=====

```

Insertion was successful Tree contents: 1 15 23 29 34 50 56 98

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 33

=====

Insertion was successful Tree contents: 1 15 23 29 33 34 50 56 98

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 40

=====

Insertion was successful Tree contents: 1 15 23 29 33 34 40 50 56 98

=====

**例2. 挿入1、挿入2、挿入3、挿入4、挿入5、挿入4、挿入3、挿入2、挿入1、挿入4（度数を保持する）
実行結果：**

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 1

=====

Insertion was successful Tree contents: 1

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 2

=====

Insertion was successful Tree contents: 1 2

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 3

=====

Insertion was successful Tree contents: 1 2 3

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 4

=====

Insertion was successful Tree contents: 1 2 3 4

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 5

=====

Insertion was successful Tree contents: 1 2 3 4 5

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 4

=====

Insertion was successful Tree contents: 1 2 3 4 4 5

=====

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 3

```

=====
Insertion was successful Tree contents: 1 2 3 3 4 4 5
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 2
=====
Insertion was successful Tree contents: 1 2 2 3 3 4 4 5
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 1
=====
Insertion was successful Tree contents: 1 1 2 2 3 3 4 4 5
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 4
=====
Insertion was successful Tree contents: 1 1 2 2 3 3 4 4 4 5
=====

```

例３．挿入１０、挿入２０、挿入３０、削除１０、挿入４０、挿入５０、挿入６０、削除２０（データの削除）
実行結果：

```

1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 10
=====
Insertion was successful Tree contents: 10
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 20
=====
Insertion was successful Tree contents: 10 20
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 30
=====
Insertion was successful Tree contents: 10 20 30
=====
1. Insert 2. Delete 3. Exit Enter your choice: 2 Enter the number to delete: 10
=====
Deletion was successful Tree contents: 20 30
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 40
=====
Insertion was successful Tree contents: 20 30 40
=====

```

```
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 50
=====
Insertion was successful Tree contents: 20 30 40 50
=====
1. Insert 2. Delete 3. Exit Enter your choice: 1 Enter the number to insert: 60
=====
Insertion was successful Tree contents: 20 30 40 50 60
=====
1. Insert 2. Delete 3. Exit Enter your choice: 2 Enter the number to delete: 20
=====
Deletion was successful Tree contents: 30 40 50 60
=====
```

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