プログラミング言語実験・C言語第一回実験レポート

1410149 吉田健太朗

2016年4月25日

1　目的

逆ポーランド記法，スタック，二分木の実装について確認し考察する．

2　方法

実験は以下の方法で行った．

1. 課題3
   (a) 中置記法の式の文字列を用意し，式中のトークンについて優先順位を数値でつけた．
   (b) 各トークンをスタックを通じて優先順位の高いものから取り出す．
   (c) 出力結果が逆ポーランド記法になることを確認した．
2. 課題4
   (a) ポインタで二分木を作成し，数値データを格納する．
   (b) 格納は絶対値が小さい値を左の子に，大きい値を右の子に格納する．
   (c) 格納した数値を絶対値昇順ソートで並んだ順で和を取る．

データセットは講義サイトに用意してあったものを使用した．また，トークンと優先順位の対応は以下の通りとした．

表1: トークンと優先順位

トークン	優先順位
非演算子	5
(	4
*,/	3
+,-	2
)	1
=	0

3　ソースコード

実験で用いたソースコード，assignment3.c，assignment4.c を以下に示す．それぞれが課題 3，課題 4 に対応している．

ソースコード 1: assignment3.c

```
1
2  /*
3  *中置記法の式を逆ポーランド記法に変換し出力する関数を作成
4  *また,例を取りその実行結果を示す
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 typedef struct _Node{
12   char token;
13   int order;
14   struct _Node* next;
15 } Node;
16
17 typedef struct _Stack{
18   Node* head;
19   Node* crnt;
20 } Stack;
21
22 int main(void){
23
24   void init_stack(Stack* stack);
25   char pop(Stack* stack);
26   void push(Stack* stack, char token);
27   int show_order(char token);
28   void input_token(Stack* stack, char* formulae);
29   void reverse_polish(char* formulae);
30   void print_stack(Stack* stack);
31   void print_array(char* array);
32
33   char* infix1 = "A=(B-C)/D+E*F";
34   char* infix2 = "A=B(C/D+E)*F";
35   char* infix3 = "A=B-C/(D+E*F)";
36
37   Stack stack1;
38   Stack stack2;
39   Stack stack3;
40
41   init_stack(&stack1);
42   init_stack(&stack2);
43   init_stack(&stack3);
44
45   input_token(&stack1, infix1);
46   input_token(&stack2, infix2);
47   input_token(&stack3, infix3);
48
49   printf("読み込む中置記法の式\n");
50   printf("式 1:␣%s\n", infix1);
51   printf("式 2:␣%s\n", infix2);
52   printf("式 3:␣%s\n", infix3);
53   printf("------------------------------\n");
54
55   printf("逆ポーランド記法へ変換し，出力\n");
56   printf("式 1:␣"); reverse_polish(infix1);
57   printf("式 2:␣"); reverse_polish(infix2);
58   printf("式 3:␣"); reverse_polish(infix3);
59
60
```

```
61    return 0;
62  }
63
64  void init_stack(Stack* stack){
65    stack->head = NULL;
66    stack->crnt = NULL;
67  }
68
69  char pop(Stack* stack){
70    char token = '\0';
71    Node* tmp = NULL;
72    if(stack->head != NULL){
73      token = stack->head->token;
74      tmp = stack->head;
75      stack->head = stack->head->next;
76      if(tmp->next == NULL) stack->head = NULL;
77      free(tmp);
78    }
79    return token;
80  }
81
82  void push(Stack* stack, char token){
83    Node* new_node = (Node*)malloc(sizeof(Node));
84    new_node->token = token;
85    if(stack->head == NULL) new_node->next = NULL;
86    if(stack->head != NULL) new_node->next =stack->head;
87    stack->head = new_node;
88  }
89
90  int show_order(char token){
91    int order;
92    switch(token){
93      case '=':
94        order = 0;
95        break;
96      case ')':
97        order = 1;
98        break;
99      case '+':
100     case '-':
101       order = 2;
102       break;
103     case '*':
104     case '/':
105       order = 3;
106       break;
107     case '(':
108       order = 4;
109       break;
110     default:
111       order = 5;
112       break;
113   }
114   return order;
115 }
116
117 void input_token(Stack* stack, char* formulae){
118   int formula_length = strlen(formulae);
119   for(int i = 0; i < formula_length; i++){
120     push(stack, formulae[i]);
121   }
122 }
123
124 void reverse_polish(char* formulae){
125   int formulae_size = strlen(formulae);
126   Stack stack;
127   init_stack(&stack);
128   for(int i = 0; i < formulae_size; i++){
129     char new_token = formulae[i];
```

```
130     while(stack.head != NULL &&
131           stack.head->token != '(' &&
132           show_order(new_token) <= show_order(stack.head->token)){
133       printf("%c", pop(&stack));
134     }
135     if(new_token != ')') push(&stack, new_token);
136     else pop(&stack);
137   }
138   while(stack.head != NULL){
139     printf("%c", pop(&stack));
140   }
141   printf("\n");
142 }
143
144 void print_stack(Stack* stack){
145   Node* node = stack->head;
146   printf("スタックの内容:␣");
147   while(node != NULL){
148     printf("%c", node->token);
149     node = node->next;
150   }
151   printf("\n");
152   node = stack->head;
153 }
```

ソースコード 2: assignment4.c

```
1
2  /*
3   *ポインタで二分木を作り,課題 1について
4   *同様に絶対値昇順ソートし,総和をとる
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10
11 typedef struct _Node{
12   double dnum;
13   struct _Node* left;
14   struct _Node* right;
15 } Node;
16
17 int main(void){
18
19   void insert(Node** root, double dnum);
20   void abs_sort_insert(Node** new_root, Node* old_root);
21   void make_tree(Node* root, double dnum);
22   double sum_tree(Node* root);
23   void print_tree(Node* root);
24
25   FILE *fp;
26   char *fname = "num.dat";
27   char s[100];
28
29   Node* root = NULL;
30
31   fp = fopen(fname, "r");
32   //ファイルオープン失敗
33   if(fp == NULL){
34     printf("%s の読み込みに失敗しました\n", fname);
35     return -1;
36   }
37
38    //成功->データ取り込み
39   printf("読み込んだデータ\n");
40   int i = 0;
41   while(fgets(s, 100, fp) != NULL){
```

```
42      insert(&root, atof(s));
43      printf("Data[%d]:␣%s", i++, s);
44    }
45    printf("\n");
46    printf("----------------------------------------\n");
47    fclose(fp);
48
49    //木に絶対値昇順ソートして挿入
50    Node* root2 = NULL;
51    abs_sort_insert(&root2, root);
52
53    printf("ソート後二分木内のデータを昇順に取り出す\n");
54    print_tree(root2);
55    printf("----------------------------------------\n");
56
57    // printf("DEBUG:: %f\n",root_sorted->dnum);
58
59    printf("昇順に総和をとった結果:␣%f\n", sum_tree(root));
60
61
62
63    return 0;
64
65  }
66
67  void insert(Node** root, double dnum){
68    Node* cursor = *root;
69    Node* new_node = (Node*)malloc(sizeof(Node));
70    new_node->dnum = dnum;
71    new_node->left = NULL;
72    new_node->right = NULL;
73
74    if(cursor != NULL){
75      double abs_new_dnum = fabs(new_node->dnum);
76      double abs_crnt_dnum = fabs(cursor->dnum);
77
78      while(1){
79        abs_crnt_dnum = fabs(cursor->dnum);
80        if(abs_new_dnum <= abs_crnt_dnum){
81          if(cursor->left == NULL) break;
82          cursor = cursor->left;
83        }
84        if(abs_new_dnum > abs_crnt_dnum){
85          if(cursor->right == NULL) break;
86          cursor = cursor->right;
87        }
88      }
89      if(abs_new_dnum <= abs_crnt_dnum) cursor->left = new_node;
90      if(abs_new_dnum > abs_crnt_dnum) cursor->right = new_node;
91    } else {
92      *root = new_node;
93    }
94  }
95
96  void abs_sort_insert(Node** new_root, Node* old_root){
97    if(old_root != NULL){
98      Node* cursor = old_root;
99      Node* left = cursor->left;
100     Node* right = cursor->right;
101
102     abs_sort_insert(new_root, left);
103     insert(new_root, cursor->dnum);
104     abs_sort_insert(new_root, right);
105   }
106 }
107
108 void print_tree(Node* root){
109   if(root != NULL){
110   Node* cursor = root;
```

```
111     Node* left = root->left;
112     Node* right = root->right;
113
114     print_tree(left);
115     printf("data: %f\n", cursor->dnum);
116     print_tree(right);
117     }
118 }
119
120 double sum_tree(Node* root){
121     double result = 0;
122     if(root != NULL){
123         Node* cursor = root;
124         Node* left = root->left;
125         Node* right = root->right;
126
127         result += sum_tree(left);
128         result += cursor->dnum;
129         result += sum_tree(right);
130     }
131     return result;
132 }
```

以上のプログラムにより，次のデータファイルを読み込む．

ソースコード 3: num.dat

```
1  1.0e16
2  −1.0e2
3  23
4  −6.4
5  3.6e2
6  −0.01
7  8.0
8  −70
9  5.0e3
10 1.2e−2
11 −3.0e3
12 46
13 −1.7e3
14 10
15 −5.0e2
16 7.0
17 −2.0e−3
18 0.3
19 −30
20 3.1
21 −1.0e16
```

4 結果

上記のソースコードを実行した結果は以下のようになった．ログファイル名の付番は実行ファイルの付番と対応している．

ソースコード 4: result3.txt

```
読み込む中置記法の式
:
式 1: A=(B−C)/D+E∗F
式 2: A=B(C/D+E)∗F
式 3: A=B−C/(D+E∗F)
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
逆ポーランド記法へ変換し,出力:
式 1: ABC−D/EF∗+=
式 2: ABCD/E+F∗=
式 3: ABCDEF∗+/−=
```

ソースコード 5: result4.txt

```
読み込んだデータ

Data[0]: 1.0e16
Data[1]: −1.0e2
Data[2]: 23
Data[3]: −6.4
Data[4]: 3.6e2
Data[5]: −0.01
Data[6]: 8.0
Data[7]: −70
Data[8]: 5.0e3
Data[9]: 1.2e−2
Data[10]: −3.0e3
Data[11]: 46
Data[12]: −1.7e3
Data[13]: 10
Data[14]: −5.0e2
Data[15]: 7.0
Data[16]: −2.0e−3
Data[17]: 0.3
Data[18]: −30
Data[19]: 3.1
Data[20]: −1.0e16
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
ソート後二分木内のデータを昇順に取り出す
data: −0.002000
data: −0.010000
data: 0.012000
data: 0.300000
data: 3.100000
data: −6.400000
data: 7.000000
data: 8.000000
data: 10.000000
data: 23.000000
data: −30.000000
data: 46.000000
data: −70.000000
data: −100.000000
data: 360.000000
data: −500.000000
data: −1700.000000
data: −3000.000000
data: 5000.000000
data: 10000000000000000.000000
data: −10000000000000000.000000
```

```
47 ----------------------------------------
48 昇順に総和をとった結果: 52.000000
```

5 考察

結果について，以下の考察を示す．

5.1 課題 3 について

逆ポーランド記法について，目的の結果を得られたことを確認した．今回はスタックにポップとプッシュを繰り返すことで，目的の記法に変換したが，トークンに与えられた優先度の値でデータを取り出していたことを考えるとこれは課題 4 の問題とほぼ等価なのではないかと考えられる．なぜなら，課題 3 も課題 4 も数値が小さい順番でデータにアクセスしている点が等しいからだ．よって優先度にしたがってヒープを作成すれば，二分木でも逆ポーランド記法変換の実装が可能であると思われる．

5.2 課題 4 について

今回はデータを絶対値を基準に左右の子にに挿入したものと，それを深さ優先で取り出してデータを根から順に格納した木が結果的にできた．

今回の結果から，実際に二分木を直接昇順ソートで挿入することは避けたほうが良いと思われた．なぜなら，絶対値昇順ソートでの挿入は根を最小の値とし右の枝にしか新たなノードが生まない一直線の木になり，二分木としての探索効率が最悪になってしまうためである．加えて，前述のようにソートのために一度データを絶対値を基準に左右の子に格納した木を作成しており，それを深さ優先で全データにアクセスすることで絶対値昇順ソートされた木を作成している，この時点で絶対値の昇順にデータの総和を取ることは可能であり，わざわざ 2 つ目の木を作成する必要がないのである．

データの取り出しの際にかかる計算量は，一直線の木で $o(\frac{n}{2})$ となり，そうでない木で $o(\log_2 2n)$ 以上 $o(\frac{n}{2})$ 以下である．同じ目的が達成できるならば計算量が少なくまたコード量が少ない手段を取るべきである．