ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 3

KIM HYUN UK

21800201@handong.edu

1. Introduction

I have to find the recursive pattern to represent the palindrome bit string up to length 6. Recursively, It generates bit string of palindrome every time. If I make the logical basis set and more easily computation recursive code, It can be solved very easily.

2. Approach

Binary palindrome bit string has some attributes to all expected result. If we set the middle of bit string, then there are two cases exist. “1” E “1” and “0” E “0”. This is the recursive case of all binary palindrome bit string.

```
1   S :== E
2   E :== "0"
3   E :== "1"
4   E :== "00"
5   E :== "11"
6   E :== "0" E "0"
7   E :== "1" E "1"
```

I set basis as “0” “1” “00” “11”. Because it is the really base case of recursive function. And then, I read the grammar text and save recursive part in recur[][] array and basis part in base[][] array.

After I made each array, call the recursive function. In function using 2 nested for-loop to calculate all possible cases. For example, In first case I generates 0[0]0 / 0[1]0 / 0[00]0 / 0[11]0 / 1[0]1 / 1[1]1 / 1[00]1 / 1[11]1. I write input value in [ ]. And this case is the basis for recursively calling function. Therefore, I generate length of binary palindrome bit string like as 3,4 / 5,6 / 6,7 … so on.

```
for(int i = 0 ; i < base_count ; i++) {
    for(int k = 0 ; k < 2 ; k++) {
        char recur_one[10];
        strcpy(recur_one, recur[k]);
        int len = strlen(recur_one);
```

```
while(strstr(recur_one, "E") != NULL) {

    char* ptr = strstr(recur_one, "E");
    recur_one[len-1 + strlen(base[i]) - 1] = recur_one[len-1];
    strncpy(ptr, base[i], strlen(base[i]));
    if(strlen(recur_one) > max_length) return;

    if(strstr(recur_one, "E") == NULL) {
        strcpy(new_base[new_base_count], recur_one);
        printf("[ %s ]", new_base[new_base_count++]);
    }
}
```

But in the question I have to represent up to length 6. Therefore, In recursive function I made the condition of return.

```
if(strlen(recur_one) > max_length) return;
```

3. Evaluation

Max_length = 4

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer1.txt 4
[ ANSWER ]

[ 0 ][ 1 ][ 00 ][ 11 ][ 000 ][ 101 ][ 010 ][ 111 ][ 0000 ][ 1001 ][ 0110 ][ 1111 ]
```

Max_length = 6

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer1.txt 6
[ ANSWER ]

[ 0 ][ 1 ][ 00 ][ 11 ][ 000 ][ 101 ][ 010 ][ 111 ][ 0000 ][ 1001 ][ 0110 ][ 1111 ][ 00
 100001 ][ 010010 ][ 110011 ][ 001100 ][ 101101 ][ 011110 ][ 111111 ]
```

```
00 ][ 10001 ][ 01010 ][ 11011 ][ 00100 ][ 10101 ][ 01110 ][ 11111 ][ 000000 ][
```

Max_length = 8

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer1.txt 8
[ ANSWER ]

[ 0 ][ 1 ][ 00 ][ 11 ][ 000 ][ 101 ][ 010 ][ 111 ][ 0000 ][ 1001 ][ 0110 ][ 1111 ][
 100001 ][ 010010 ][ 110011 ][ 001100 ][ 101101 ][ 011110 ][ 111111 ][ 0000000 ][ 10
0 ][ 1001001 ][ 0101010 ][ 1101011 ][ 0011100 ][ 1011101 ][ 0111110 ][ 1111111 ][ 00
 11100111 ][ 00011000 ][ 10011001 ][ 01011010 ][ 11011011 ][ 00111100 ][ 10111101 ][
```

```
[ 00000 ][ 10001 ][ 01010 ][ 11011 ][ 00100 ][ 10101 ][ 01110 ][ 11111 ][ 000000 ][
1000001 ][ 0100010 ][ 1100011 ][ 0010100 ][ 1010101 ][ 0110110 ][ 1110111 ][ 000100
00000000 ][ 10000001 ][ 01000010 ][ 11000011 ][ 00100100 ][ 10100101 ][ 01100110 ][
][ 01111110 ][ 11111111 ]
```

4. Discussion

gcc hw3_1.c -o langdump

./langdump grammer1.txt 10

5. Conclusion

It’s different to calculate the math problem. Even it is great to understand the code for human, It’s calculation is difficult for computer. Therefore, I have to make the grammar more easily computable.

ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 3

KIM HYUN UK

21800201@handong.edu

1. Introduction

I have to find the recursive pattern to represent the properly nested one or more parenthesis, curly brackets, square bracket. I have to write the grammar in basis and recursive part. When I make the new generated basis set, If the length of them is same, It'll be more easy to setting the condition in recursive function.

2. Approach

Properly nested brackets and parenthesis have the same recursive pattern. E "(" ")" / "(" ")" E / "(" E ")". It's the case of recursive pattern. Consecutive case and nested case. Using nested for-loop, Input the basis set in the place of E and D, R also too.

```
1   S :== E
2   S :== D
3   S :== R
4   E :== "(" ")"
5   D :== "{" "}"
6   R :== "[" "]"
7   E :== E "(" ")"
8   E :== "(" ")" E
9   E :== "(" E ")"
10  D :== D "{" "}"
11  D :== "{" "}" D
12  D :== "{" D "}"
13  R :== R "[" "]"
14  R :== "[" "]" R
15  R :== "[" R "]"
```

Then It will make like this.

```
() {} []
()()  (())
{}{}  {{}}  (){}  {}()  {()}
[][]  [[]]  ()[]  []()  [()]  {}[]  []{}  [{}]
```

Second line is for E basis, third line if for D basis and fourth line is for R basis. E basis can input to E,D,R in recursive basis. And D basis can input to D, R in recursive basis. Finally, R only can input to R recursive. Therefore, I can represent every properly compound brackets and parenthesis.

```
for(int j = 0; j < 3 ; j++){
    for(int i = 0 ; i < e_base_count ; i++){
        char* recur_one = malloc(sizeof(char) * 1000);
        strcpy(recur_one, recur[j]);
        int len = strlen(recur_one);

        while(strstr(recur_one, "E") != NULL) {
            char* ptr = strstr(recur_one, "E");
            if(strcmp(ptr, "E()") == 0)
            {
                strncpy(ptr, E_base[i], strlen(E_base[i]));
                recur_one[strlen(E_base[i])] = '(';
                recur_one[strlen(E_base[i])+1] = ')';
                strcpy(new_E_base[new_e_base_count++], recur_one);
```

This code is for understanding for replacing the E part.

```
for(int i = 0 ; i < new_r_base_count ; i++) {

    int count = 0;
    for(int j = i+1 ; j < new_r_base_count ; j++) {
        if(strcmp(new_R_base[i], new_R_base[j]) == 0) {
            count++;
        }
    }
    if(count == 0) {
        if(strlen(new_R_base[i]) <= max_length)
            printf(" %s " , new_R_base[i]);
        else
            rcount++;
    }
}
printf("\n");

if(rcount > 0) return;
```

In nested for-statement, using Bubble sort logic, comparing (strcmp) for all generated string and if it is same as the pivot, not print as result. (In my case all duplicated case occurred in consecutively, therefore I could apply this logic)

3. Evaluation

Max_length = 6

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer2.txt 6
 () {} []
 ()()  (())
 {}{}  {{}}  (){}  {}()  {()}
 [][]  [[]]  ()[]  []()  [()]  {}[]  []{}  [{}]
 (())()  ()()()  ()(())  (()())  ((()))
 {{}}{}  ()[]{}  {()}{}  {}{}{}  {}{()}  {}()[]  {}[]()  {}[{}]  {{}}()  {{()}}  {()
 [[]][]  ()[][]  [()][]  {}[][]  [{}][]  [][][]  [][[]]  []()[]  []{}()  []{()}  []{
]]  ()()[]  (())[]  []()()  []{}()  []{()}  {}{}[]  {{}}[]  ()[]{}  {}()[]
{()}]
```

```
(){}  {{}()}  {{()}}  ()()[]  (())[]  {}()()  {}(())  {()()}  {(())}
]{}[]  [][]{}  []{}[]  [{}][]  [[]][]  [()[]]  [[]()]  [{}()]  [[]{}]  [[{}
  {()}[]  []{}{}  []{{}}  []()[]  []{}()  []{()}  [{}{}]  [{{}}]  [()[]]  [
```

In the max_length = 6, all result is properly represented through 3 recursive function call. Therefore, I can prove that max_length = 10 present the result properly.

Max_length = 10

4. Discussion

gcc hw3_2.c -o langdump

./langdump grammer2.txt 10

5. Conclusion

Every logic is simple if I give the grammar properly to computer. However finding the duplicated result is may be difficult if I make the grammar wrong.

ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 3

KIM HYUN UK

21800201@handong.edu

1. Introduction

Postfix representation is the rule to calculate for computer. And its logic is quite different unlikely we use as usual. Every postfix representation is odd number because the number of operator must be -1 compare to the number of operand.

2. Approach

I think that It has really simply logic E E D. E for operand and D for operator. And recursively it make all case of up to length 10. But there is restriction for duplication and not efficient in using data memory. Therefore I edit the grammar like this.

```
S :== E
D :== "*"
D :== "+"
D :== "-"
D :== "/"
E :== "1"
E :== "2"
E :== "3"
E :== "4"
```

```
E :== E 1 D
E :== E 2 D
E :== E 3 D
E :== E 4 D
E :== 1 E D
E :== 2 E D
E :== 3 E D
E :== 4 E D
```

I made that every recursive part have one E and D for implement the code for replacing E and D as basis set. Using several nested for-loop I could get the result.

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer3.txt 3
GOOD JOB!
[ 11* ][ 11+ ][ 11- ][ 11/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 41* ][ 4
/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 42* ][ 42+ ][ 42- ][ 42/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 23* ]
 43/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 24* ][ 24+ ][ 24- ][ 24/ ][ 34* ][ 34+ ][ 34- ][ 34/ ][ 44
 ][ 12/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 2
24- ][ 24/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 33* ][ 33+ ][ 33- ][ 33/ ]
][ 42- ][ 42/ ][ 43* ][ 43+ ][ 43- ][ 43/ ][ 44* ][ 44+ ][ 44- ][ 44/ ]rcount = 128
```

I set basis “1” “2” “3” “4” as E basis and “*” “+” “-” “/” as D basis. And in the recursive function, using nested for-statement get the all result from replacing recursive part E and D as basis for them. And recursively make the new E and D basis, In the same logic I could get the correct result.

```
if(strcmp(ptr, "E1D") == 0)
{
    strncpy(ptr, E_base[i], strlen(E_base[i]));
    recur_one[strlen(E_base[i])] = '1';
while(strstr(recur_one, "D") != NULL) {
    // Symbol(D)의 Index 위치를 파악
    char* ptr = strstr(recur_one, "D");
    //printf("ptr = %s", recur_one);
    strncpy(ptr, D_base[j], strlen(D_base[j]));
    strcpy(new_D_base[new_d_base_count++], recur_one);
}
```

Using strstr to find the E and D position and strncpy to replace the E and D to each basis set.

```
int rcount = 0;
for(int i = 0 ; i < new_d_base_count ; i++) {
    if(strlen(new_D_base[i]) == max_length) rcount++;
        printf("[ %s ]", new_D_base[i]);
}
if(rcount >0) {
    printf("rcount = %d", rcount);
    return;
}
```

If the length of new_D_base is same with max_length(argv[2]) increase the rcount variable, then after the print the all result and return the function.

3. Evaluation

Max_length = 3

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer3.txt 3
[ 11* ][ 11+ ][ 11- ][ 11/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 41* ][ 41+
/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 42* ][ 42+ ][ 42- ][ 42/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 23* ][
 43/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 24* ][ 24+ ][ 24- ][ 24/ ][ 34* ][ 34+ ][ 34- ][ 34/ ][ 44*
 ][ 12/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 2
24- ][ 24/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 33* ][ 33+ ][ 33- ][ 33/ ]
][ 42- ][ 42/ ][ 43* ][ 43+ ][ 43- ][ 43/ ][ 44* ][ 44+ ][ 44- ][ 44/ ]
```

Max_length = 5

```
PS C:\Users\vldrj\OneDrive\바탕 화면\이산 수학\HW3> ./langdump grammer3.txt 5
[ 11* ][ 11+ ][ 11- ][ 11/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 41* ][ 41+ ][
/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 42* ][ 42+ ][ 42- ][ 42/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 23* ][ 23+
 43/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 24* ][ 24+ ][ 24- ][ 24/ ][ 34* ][ 34+ ][ 34- ][ 34/ ][ 44* ][
 ][ 12/ ][ 13* ][ 13+ ][ 13- ][ 13/ ][ 14* ][ 14+ ][ 14- ][ 14/ ][ 21* ][ 21+ ][ 21- ][ 21/ ][ 22*
24- ][ 24/ ][ 31* ][ 31+ ][ 31- ][ 31/ ][ 32* ][ 32+ ][ 32- ][ 32/ ][ 33* ][ 33+ ][ 33- ][ 33/ ][ 3
][ 42- ][ 42/ ][ 43* ][ 43+ ][ 43- ][ 43/ ][ 44* ][ 44+ ][ 44- ][ 44/ ]1[ 11*1* ][ 11*1+ ][ 11*1- ]
1-1- ][ 11-1/ ][ 11/1* ][ 11/1+ ][ 11/1- ][ 11/1/ ][ 21*1* ][ 21*1+ ][ 21*1- ][ 21*1/ ][ 21+1* ][ 2
* ][ 21/1+ ][ 21/1- ][ 21/1/ ][ 31*1* ][ 31*1+ ][ 31*1- ][ 31*1/ ][ 31+1* ][ 31+1+ ][ 31+1- ][ 31+1
[ 31/1/ ][ 41*1* ][ 41*1+ ][ 41*1- ][ 41*1/ ][ 41+1* ][ 41+1+ ][ 41+1- ][ 41+1/ ][ 41-1* ][ 41-1+ ]
2*1+ ][ 12*1- ][ 12*1/ ][ 12+1* ][ 12+1+ ][ 12+1- ][ 12+1/ ][ 12-1* ][ 12-1+ ][ 12-1- ][ 12-1/ ][ 1
/ ][ 22+1* ][ 22+1+ ][ 22+1- ][ 22+1/ ][ 22-1* ][ 22-1+ ][ 22-1- ][ 22-1/ ][ 22/1* ][ 22/1+ ][ 22/1
[ 32+1- ][ 32+1/ ][ 32-1* ][ 32-1+ ][ 32-1- ][ 32-1/ ][ 32/1* ][ 32/1+ ][ 32/1- ][ 32/1/ ][ 42*1* ]
2-1* ][ 42-1+ ][ 42-1- ][ 42-1/ ][ 42/1* ][ 42/1+ ][ 42/1- ][ 42/1/ ][ 13*1* ][ 13*1+ ][ 13*1- ][ 1
```

Number of all case for length 5 is 4024 and next length might be bigger than 20000. I couldn’t make the data memory for cover that, but the logic is not change.

4. Discussion

gcc hw3_3.c -o langdump

./langdump grammer3.txt 10

5. Conclusion

I wanted to declare two-dimension array as malloc but there is a parameter error I couldn’t solve. But the result up to length 5 is correct. The logic is right but the computer memory is not easy to solve.