HW3

一. 选择题

1. C;

二. 算法设计题

1. (1) AD;

   (2)

```
bool isHandlerAllowed(const char handler[], const int n)
{
    // n 输入之前记得不要计算字符串最后的"\0"
    if (n < 0 || n % 2 == 1) return false;

    int stack_state = 0;
    for (int i = 0; i < n; i++)
    {
        if (handler[i] == 'I') stack_state++;
        else if (handler[i] == 'O') stack_state--;

        if (stack_state < 0) return false;
    }

    if (stack_state != 0) return false;
    return true;
}
```

2. (1)

```
int ack(int m, int n)
{
    if (m == 0) return n + 1;
    if (n == 0) return ack(m - 1, 1);
    return ack(m - 1, ack(m, n - 1));
}
```

```
Ack(2,1) = Ack(1, Ack(2, 0))
         = Ack(1, Ack(1, 1))
         = Ack(1, Ack(0, Ack(1, 0)))
         = Ack(1, Ack(0, Ack(0, 1)))
         = Ack(1, Ack(0, 2))
         = Ack(1, 3)
         = Ack(0, Ack(1, 2))
         = Ack(0, Ack(0, Ack(1, 1)))
         = Ack(0, Ack(0, Ack(0, Ack(1, 0))))
         = Ack(0, Ack(0, Ack(0, Ack(0, 1))))
```

```
          = Ack(0, Ack(0, Ack(0, 2)))
          = Ack(0, Ack(0, 3))
          = Ack(0, 4)
          = 5
```

(2)

```h
/*
这里是代码的.h文件。
*/

#include <stdbool.h>

typedef enum
{
    START = 0,
    WAITING,
    END
} StackState;

typedef struct StackFrame // 栈帧（节点信息）
{
    int m;
    int n;
    int result; // 初始值为-1
    StackState state;
    struct StackFrame* next;
} StackFrameHandler;

typedef struct
{
    StackFrameHandler* top;
    int size;
} Stack;

// 栈操作函数
void initStack(Stack* stack);
bool isStackEmpty(const Stack* stack);
void push(Stack* stack, int m, int n);
StackFrameHandler* pop(Stack* stack);
void freeStack(Stack* stack);

// ack(m, n)
int ack(const int m, const int n);
int ackWithStack(const int m, const int n);
```

```c
/*
这里是函数具体实现的.c文件。
*/

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "ackermann.h"
```

```
9
10  void initStack(Stack* stack)
11  {
12      if (stack == NULL) return;
13      stack->top = NULL;
14      stack->size = 0;
15  }
16
17  bool isStackEmpty(const Stack* stack)
18  {
19      if (stack == NULL) return true;
20      return stack->size == 0;
21  }
22
23  void push(Stack* stack, int m, int n)
24  {
25      StackFrameHandler* node = malloc(sizeof(StackFrameHandler));
26      if (node == NULL)
27      {
28          printf("内存满了，开不动了。");
29          exit(1);
30      }
31      node->m = m;
32      node->n = n;
33      node->result = -1;
34      node->state = START;
35      node->next = stack->top;
36      stack->top = node;
37      stack->size++;
38  }
39
40  StackFrameHandler* pop(Stack* stack)
41  {
42      if (isStackEmpty(stack)) return NULL;
43      StackFrameHandler* top_frame = stack->top;
44      stack->top = stack->top->next;
45      stack->size--;
46      return top_frame;
47  }
48
49  void freeStack(Stack* stack)
50  {
51      while (!isStackEmpty(stack))
52      {
53          StackFrameHandler* temp = pop(stack);
54          free(temp);
55      }
56  }
57
58  int ack(const int m, const int n)
59  {
60      if (m == 0) return n + 1;
61      if (n == 0) return ack(m - 1, 1);
62      return ack(m - 1, ack(m, n - 1));
63  }
64
65  int ackWithStack(const int m, const int n)
66  {
```

```
    // 初始化阶段
    Stack* stack = malloc(sizeof(Stack));
    int result = -1;
    initStack(stack);

    // 正式处理阶段
    push(stack, m, n);
    while (!isStackEmpty(stack))
    {
        StackFrameHandler* current = stack->top;
        switch (current->state)
        {
            case START:
                if (current->m == 0)
                {
                    current->result = current->n + 1;
                    current->state = END;
                }
                else if (current->n == 0)
                {
                    current->m -= 1;
                    current->n = 1;
                }
                else
                {
                    push(stack, current->m, current->n - 1);
                    current->state = WAITING;
                }
                break;

            case WAITING: // 进WAITING说明 (m != 0 && n != 0)
                assert(result != -1); // 保证结果如我预期

                current->m -= 1;
                current->n = result;
                current->state = START;
                result = -1;
                break;

            case END:
                result = current->result;
                pop(stack);
                break;
        }
    }

    // 结束收尾阶段
    freeStack(stack);
    free(stack);
    return result;
}
```

3.

```
/*
这是在.h文件中的内容。
*/
typedef struct Node
{
```

```c
    int data;
    struct Node* next;
} Node;

typedef struct
{
    int count;
    int sum;
} Analysis;

int findMaxData(const Node* head);
int countNodesNum(const Node* head);
double averageData(const Node* head);

/*
这是在.c文件中的内容。
*/
#include  <stdio.h>
#include <stdlib.h>
#include "linked_list.h"

static int findMax(const Node* node)
{
    if (node->next == NULL) return node->data;
    int max = findMax(node->next);
    return (node->data > max) ? node->data : max;
}

static int countNodes(const Node* node)
{
    if (node == NULL) return 0;
    return countNodes(node->next) + 1;
}

static Analysis averageAnalysis(const Node* node)
{
    Analysis temp = {0, 0};
    if (node == NULL) return temp;

    temp = averageAnalysis(node->next);
    temp.count++;
    temp.sum += node->data;

    return temp;
}

int findMaxData(const Node* head)
{
    Node* current = head->next; // 跳过头节点
    if (current == NULL)
    {
        printf("无法找到空链表的最大元！");
        exit(1);
    }

    int max = findMax(current);
    return max;
}
```

```
64
65  int countNodesNum(const Node *head)
66  {
67      Node* current = head->next;
68      return countNodes(current);
69  }
70
71  double averageData(const Node *head)
72  {
73      Node* current = head->next;
74      Analysis analysis = averageAnalysis(current);
75
76      if (analysis.count == 0) return 0.0;
77
78      double result = (double) analysis.sum / analysis.count;
79      return result;
80  }
```

附. 算法设计补充题

```
1   /*
2   由于.h文件中仅有最后一个函数的函数声明，故略。
3   */
4
5   #include <stdio.h>
6   #include <ctype.h>
7   #include <math.h>
8   #include "fun.h"
9
10  static char signal[3] = {' ', '+', '-'};
11
12  static void print_array(const char* str)
13  {
14      const char* temp = str;
15      while (*temp != '\0')
16      {
17          if (*temp == ' ');
18          else printf("%c", *temp);
19
20          temp++;
21      }
22
23      printf("\n");
24  }
25
26  static int calculator(const char* str)
27  {
28      const char* temp = str;
29      int num = 0;
30      int sum = 0;
31      int sign = 1; // 1表示正；-1表示负
32
33      while (*temp != '\0')
34      {
35          if (*temp == ' ');
36          else if (isdigit(*temp)) num = num * 10 + (*temp - '0');
```

```
        else if (*temp == '+' || *temp == '-')
        {
            sum += sign * num;
            num = 0;
            sign = (*temp == '+') ? 1 : -1;
        }

        temp++;
    }

    sum += sign * num;
    return sum;
}

static void generate(char* str, int key)
{
    // key 是从 0 到 3**8 - 1
    char* current = str + 1;
    int index = 0;
    while (*current != '\0')
    {
        index = key % 3;
        *current = signal[index];
        current += 2;
        key /= 3;
    }
}

void enumerate_possiblity()
{
    char original[] = "1 2 3 4 5 6 7 8 9";
    for (int i = 0; i < pow(3, 8); i++)
    {
        generate(original, i);
        if (calculator(original) == 110) print_array(original);
    }
}
```