### 上机实验2：栈和递归

#### 班级：2022211305 学号：2022211119 姓名：赵宇鹏

#### **Email:zhaoyupeng@bupt.edu.cn**

1. 实验内容：完成了括弧匹配检验问题以及完全背包问题，无选做部分。
2. 算法与源程序结构：

括弧匹配问题：有is\_empty, is\_full, init, push, pop以及main函数，分别用来判断栈空，栈满，初始化栈以及判断是否配对，思路为仅将字符串中的”(”和”[”存入创建的堆栈中，若读取到”)”和”]”则出栈与其进行配对，如果配对成功则进行下一个字符的配对，直到栈空并且所有字符串读取完毕，若在开始时就输入”]”和”)”，则用另外的条件判断语句进行判断，若有不配对情况，则直接返回并输出不配对。

完全背包问题：声明一全局数组bestItems[N]用来储存最佳方案并不受递归多次而重置状态的影响，函数knap用来递归出最大价值和最佳方案，思路如下：使用knap函数对储存物品信息的结构体items进行递归，从第一个物品开始递归，到第N件物品开始返回上一层knap，期间对每一件物品的数目由0到能装下的最大值进行遍历递归，由1到N，由0到最大从而对所有情况进行统计和比对，若总价值大于目前所得最大值，则使用bestItems进行储存。

3、实验总结：完成此次作业大概花费三小时时间。在完成括弧匹配问题时，刚开始忽略了字符串开头就是”]”或者”)”的情况，由此产生了输入为”)”或者”]”时，由于这两个符号不入栈，所以栈仍为空，输出配对的bug，后加入一个条件判断语句进行了修改。

主要时间花费在完全背包问题上：由于代码开始时时仿写ppt中代码，只实现了输出最大价值，却无法输出最佳方案，在研究过程中，曾使用增加在结构体中增加num变量以储存每个物品被拿走的次数，但是递归加遍历使我难以找到合适的重置状态的位置，num中的数据为物品在递归中被拿走的次数的总和而非一个方案的次数，由此耗费了大量时间，后对knap函数进行了重构，用num来储存物品数目，总价值在最后一次递归中算出，之后再重置num以解决该问题。

## 括弧匹配问题

#include<stdio.h>

typedef struct Stack{

    char data[100];

    int top;

}Stack;

void init(Stack\* stack){

    stack->top = -1;

}

int is\_empty(Stack\* stack){

    if(stack->top == -1)

    return 1;//空栈返回1

    else return 0;//非空返回0

}

int is\_full(Stack\* stack){

    if(stack->top == 99)

    return 1;//满栈返回1

    else return 0;//未满返回0

}

void push(Stack\* stack, char data){

    if(is\_full(stack) == 0){

        stack->top++;

        stack->data[stack->top] = data;

    }else{

        printf("栈满\n");

    }

}

char pop(Stack\* stack){

    if(is\_empty(stack) == 0){

        char data = stack->data[stack->top];

        stack->top--;

        return data;

    }else{

        printf("空栈");

        return '\0';

    }

}

int main()

{

    char item[100];

    char temp;

    int i = 0;

    while((temp = getchar()) != '\n'){

        item[i] = temp;

        i++;

    }

    Stack\* stack;

    init(stack);

    for(i = 0;item[i] != '\0';i++){

        if(item[i] == '(' || item[i] == '['){

            if(is\_full(stack) == 0)

            push(stack,item[i]);

        }

        else if(item[i] == ')' || item[i] == ']'){

            if(is\_empty(stack) == 0){

                char temp = pop(stack);

                if((temp == '(' && item[i] == ']') || (temp == '[' && item[i] == ')')){

                    printf("不配对\n");

                    return 0;

                }

            }else{

                printf("不配对\n");

                return 0;

            }

        }

    }

    if(is\_empty(stack) == 1){

        printf("配对\n");

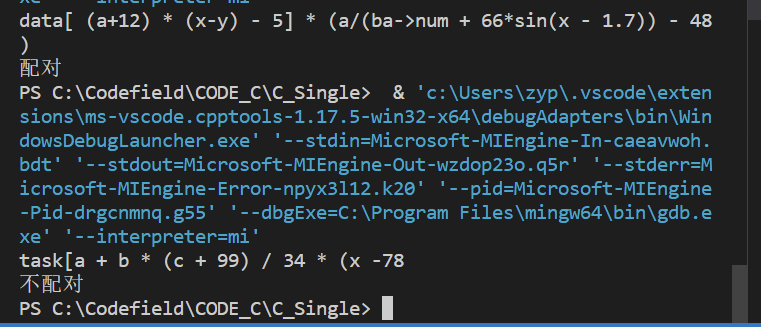
    }else{

        printf("不配对\n");

    }

    return 0;

}



## 完全背包问题

//完全背包问题，背包空间小于1000，物品种类小于100

#include <stdio.h>

#include <stdlib.h>

#define N 5

struct ITEM {

    int size;

    int val;

    int num;

};

struct ITEM bestItems[N];//储存最佳方案

int max = 0;

void knap(int cap, struct ITEM \*items, int n) {

    if (n == N) { //n为遍历到的第n件物品

        int total = 0;

        for (int i = 0; i < N; i++) {

            total += items[i].val \* items[i].num; //把每件物品的价值和数目相乘后累加起来

        }

        if (total > max) {

            max = total;

            for (int i = 0; i < N; i++) {

                bestItems[i] = items[i]; //找到一个更好的方案就进行更换

            }

        }

        return;

    }

    for (int i = 0; i <= cap / items[n].size; i++) {

        items[n].num = i;  //对所有的物品的数目都进行遍历

        knap(cap - i \* items[n].size, items, n + 1);

        items[n].num = 0; // 重置状态

    }

}

int main() {

    struct ITEM items[N];

    for (int i = 0; i < N; i++) {

        scanf("%d",&items[i].size);

        scanf("%d",&items[i].val);

        items[i].num = 0;

    }

    knap(17, items, 0);

    for (int i = 0; i < N; i++) {

        printf("物品[%d]: %d\n", i, bestItems[i].num);

    }

    printf("最大价值：%d\n", max);

    return 0;

}

