//#include <stdio.h>

//#include <math.h>

//

////作业4.1 对称数判断

//

//void symnum\_tell(int num);

//void symnum\_tell2(int num);

//

//int main()

//{

// int num=0;

// scanf("%d", &num);

//

// symnum\_tell2(num);

//

// return 0;

//}

//

//void symnum\_tell(int num)

//{

//

//// int n=0;

//// while(num/=10 != 0) //死循环，赋值优先级低于关系

//// {

////// num/=10;

//// n++;

//// }

//// printf("%d", n);

//

// int left=0, right=0;

// int n=0, num\_bk=0;

// int flag=1;

// while(num>10)

// {

// num\_bk=num;

// n=0;

// while(num != 0)

// {

// num/=10;

// n++;

// }

//

// right=num\_bk%10;

// left=num\_bk/pow(10,n-1); //double pow(double x, double y)

// num=num\_bk%(int)pow(10,n-1)/10;

//

// if(left != right)

// {

// flag=0;

// break;

// }

//

// }

// flag && printf("yes\n");

// flag || printf("no\n");

//}

//

////通过逆置比较，判断对称数

//void symnum\_tell2(int num)

//{

// int a=num;

// int b=0;

// while(num)

// {

// b=b\*10+num%10;

// num/=10;

// }

// if(a==b)

// {

// printf("yes\n");

// }else{

// printf("no\n");

// }

//}

//#include <stdio.h>

//

////作业4.2 n阶层的计算

//int main(){

// int n=0, fac=1;

// scanf("%d", &n);

//

// 倒序计算n的阶层

// while(n>1)

// {

// fac\*=n;

// n--;

// }

// printf("%d", fac);

//

// return 0;

//}

//#include <stdio.h>

////作业4.3

////某人想将手中的一张面值100元的人民币换成10元、5元、2元和1元面值的票子。

////要求换正好40张，且每种票子至少一张。问：有几种换法？

//

//int main(){

// //列举所有情况，筛出100元、40张的

// int n\_rmb10=0, n\_rmb5=0, n\_rmb2=0, n\_rmb1=0;//a,b,c,d

// int sum\_rmb=0, sum\_n=0, num=0;

// for(n\_rmb10=1; n\_rmb10<=100/10; n\_rmb10++)

// {

// for(n\_rmb5=1; n\_rmb5<=100/5; n\_rmb5++)

// {

// for(n\_rmb2=1; n\_rmb2<=100/2; n\_rmb2++)//n\_rmb2<=37

// {

// for(n\_rmb1=1; n\_rmb1<=100/1; n\_rmb1++)//n\_rmb1<=37

// {

// sum\_rmb=n\_rmb10\*10+n\_rmb5\*5+n\_rmb2\*2+n\_rmb1\*1;

// sum\_n=n\_rmb10+n\_rmb5+n\_rmb2+n\_rmb1;

// if(100==sum\_rmb && 40==sum\_n)

// {

// num++;

// }

// }

// }

// }

// }

// printf("%d", num);

//

// return 0;

//}

////作业10，顺序表操作-插入、删除、打印

////输入6'\n'1'\n' 输出1 6 2 3'\n'6 2 3

////11/17

//

//#include <stdio.h>

//#include <stdlib.h>

//

//#define SqlMaxSize 50

//typedef int SqlElemType;

//

//typedef struct SqList{

// SqlElemType data[SqlMaxSize];

// int length;

//}SqList;

//

////顺序表打印

//void sqlist\_print(SqList S){

// if(0==S.length){

// printf("NULL");

// }

// for(int i=0;i<S.length;i++){

// printf("%3d", S.data[i]);

// }

// printf("\n");

//}

////顺序表初始化

//void sqlist\_init(SqList &S){

// S.length=0;

//}

////顺序表插入

//bool sqlist\_insert(SqList &S, int pos, SqlElemType in){

// if(SqlMaxSize==S.length){

// return false;

// }

// if(pos<1 || pos>S.length+1){

// return false;

// }

// for(int i=S.length;i>=pos;i--){

// S.data[i]=S.data[i-1];

// }

// S.data[pos-1]=in;

// S.length++;

// return true;

//}

////顺序表删除

//bool sqlist\_delete(SqList &S, int pos, SqlElemType &out){

// if(0==S.length){

// return false;

// }

// if(pos<1 || pos>S.length){

// return false;

// }

// out=S.data[pos-1];

// for(int i=pos;i<S.length;i++){

// S.data[i-1]=S.data[i];

// }

// S.length--;

// return true;

//}

//

//int main(){

// SqList S;

// sqlist\_init(S);

// S.data[0]=1;

// S.data[1]=2;

// S.data[2]=3;

// S.length=3;

// sqlist\_print(S);

//

// SqlElemType input=0;

// scanf("%d", &input);

// if(sqlist\_insert(S, 2, input)){

// sqlist\_print(S);

// }else{

// printf("false\n");

// }

//

// int pos=0;

// SqlElemType del=0;

// scanf("%d", &pos);

// if(sqlist\_delete(S, pos, del)){

// sqlist\_print(S);

// }else{

// printf("false\n");

// }

//

// return 0;

//}

////作业11，链表操作-头插法、尾插法、输出

////11/17

//

//#include <stdio.h>

//#include <stdlib.h>

//

//typedef int LilElemType;

//typedef struct LinkNode{

// LilElemType data;

// struct LinkNode \*next;

//}LinkNode, \*LinkList;

//

////链表打印

//void linklist\_print2(LinkList L){

// L=L->next;

// while(L){

// printf("%d", L->data);

// L=L->next;

// if(L){

// printf(" ");

// }

// }

// printf("\n");

//}

////链表头插法建立

//void linklist\_headcreate(LinkList &L){

// L=(LinkList)malloc(sizeof(LinkNode));

// L->next=NULL;

// LinkList l;

// LilElemType input=0;

//

// scanf("%d", &input);

// while(input!=9999){ //判断语句不能 while(scanf()!=999)

// l=(LinkList)malloc(sizeof(LinkNode));

// l->data=input;

// l->next=L->next;

// L->next=l;

// scanf("%d", &input);

// }

////or

//// while(scanf("%d", &input);){

//// if(9999==input){

//// break;

//// }

//// l=(LinkList)malloc(sizeof(LinkNode));

//// l->data=input;

//// l->next=L->next;

//// L->next=l;

//// }

//}

////链表尾插法建立

//void linklist\_tailcreate(LinkList &L){

// L=(LinkList)malloc(sizeof(LinkNode));

//// L->next=NULL;

// LinkList l,end=L;

// LilElemType input=0;

// scanf("%d", &input);

//

// while(input!=9999){

// l=(LinkList)malloc(sizeof(LinkNode));

// l->data=input;

// end->next=l;

// end=l;

// scanf("%d", &input);

// }

// end->next=NULL;

//}

//

//int main(){

// LinkList L;

// linklist\_tailcreate(L);

// linklist\_print2(L);

//

// return 0;

//}

////作业12，链表操作-查找、插入、删除

////11/17

//

//#include <stdio.h>

//#include <stdlib.h>

//

//typedef int LinkElemType;

//typedef struct LinkNode{

// LinkElemType data;

// LinkNode \*next;

//}LinkNode, \*LinkList;

//

////链表打印

//void linklist\_print(LinkList L){

// L=L->next;

// while(L){

// printf("%3d", L->data);

// L=L->next;

// }

// printf("\n");

//}

////尾插法建表

//void linklist\_tailcreate(LinkList &L){

// L=(LinkList)malloc(sizeof(LinkNode));

// LinkList l, end=L;

// LinkElemType input=0;

// scanf("%d", &input);

// while(input!=9999){

// l=(LinkList)malloc(sizeof(LinkNode));

// l->data=input;

// end->next=l;

// end=l;

// scanf("%d", &input);

// }

// end->next=NULL;

//}

////按位查找

//LinkList linklist\_elemget(LinkList L, int pos){

// if(pos<0){

// return NULL;

// }

// int i=0;

// while(i<pos && L){

// L=L->next;

// i++;

// }

// return L;

//}

////插入元素

//bool linklist\_insert(LinkList L, int pos, LinkElemType in){

// LinkList pre=linklist\_elemget(L, pos-1);

// if(pre){

// LinkList l;

// l=(LinkList)malloc(sizeof(LinkNode));

// l->data=in;

// l->next=pre->next;

// pre->next=l;

// return true;

// }

// return false;

//}

////删除元素

//bool linklist\_delete(LinkList L, int pos, LinkElemType &out){

// LinkList pre=linklist\_elemget(L, pos-1);

// if(pre){

// LinkList l=pre->next;

// out=l->data;

// pre->next=l->next;

// free(l);

// l=NULL;

// return true;

// }

// return false;

//}

//

//

//int main(){

// LinkList L;

// linklist\_tailcreate(L);

// linklist\_print(L);

//

// LinkList get=NULL;

// get=linklist\_elemget(L, 2);

// if(get){

// printf("%d\n", get->data);

// }else{

// printf("NULL\n");

// }

//

// bool ret;

// ret=linklist\_insert(L, 2, 99);

// if(ret){

// linklist\_print(L);

// }else{

// printf("insert failed\n");

// }

//

// LinkElemType del;

// ret=linklist\_delete(L, 4, del);

// if(ret){

// linklist\_print(L);

// }else{

// printf("delete failed\n");

// }

//

//

// return 0;

//}

////作业13，栈与队列-入栈、出栈、入队、出队

////11/23

//#include <stdio.h>

//#include <stdlib.h>

//

////栈部分

//#define SqsMaxSize 10

//typedef int SqsElemType;

//typedef struct SqStack{

// SqsElemType data[SqsMaxSize];

// int top;

//}SqStack;

//void sqstack\_init(SqStack &SS){

// SS.top=-1;

//}

//bool sqstack\_push(SqStack &SS, SqsElemType in){

// if(SqsMaxSize-1==SS.top){

// return false;

// }

// SS.data[++SS.top]=in;

// return true;

//}

//bool sqstack\_pop(SqStack &SS, SqsElemType &out){

// if(-1==SS.top){

// return false;

// }

// out=SS.data[SS.top--];

// return true;

//}

//

////循环队列部分

//#define SqqMaxSize 5

//typedef int SqqElemType;

//typedef struct SqQueue{

// SqqElemType data[SqqMaxSize];

// int front, rear;

//}SqQueue;

//void sqqueue\_init(SqQueue &SQ){

// SQ.front=SQ.rear=0;

//}

//bool sqqueue\_en(SqQueue &SQ, SqqElemType in){

// if((SQ.rear+1)%SqqMaxSize==SQ.front){

// return false;

// }

// SQ.data[SQ.rear]=in;

// SQ.rear=(SQ.rear+1)%SqqMaxSize;

// return true;

//}

//bool sqqueue\_de(SqQueue &SQ, SqqElemType &out){

// if(SQ.front==SQ.rear){

// return false;

// }

// out=SQ.data[SQ.front];

// SQ.front=(SQ.front+1)%SqqMaxSize;

// return true;

//}

//

//int main(){

// //栈部分

// SqStack SS;

// sqstack\_init(SS);

//

// SqsElemType input;

// for(int i=0; i<3; i++){

// scanf("%d", &input);

// sqstack\_push(SS, input);

// }

// SqsElemType output;

// for(int i=0; i<3; i++){

// sqstack\_pop(SS, output);

// printf("%2d", output);

// }

// printf("\n");

//

// //循环队列部分

// SqQueue SQ;

// sqqueue\_init(SQ);

//

// SqqElemType input2;

// bool ret;

// for(int i=0; i<5; i++){

// scanf("%d", &input2);

// ret=sqqueue\_en(SQ, input2);

// if(!ret){

// printf("false\n");

// }

// }

// SqqElemType output2;

// for(int i=0; i<4; i++){

// sqqueue\_de(SQ, output2);

// printf("%2d", output2);

// }

// printf("\n");

//

// return 0;

//}

////作业14，二叉树

////1.层次建树读取abcdefghij，然后前序遍历

////2.层次建树，然后中序遍历，后序遍历，层序遍历

////11/29

//

//#include <stdio.h>

//#include <stdlib.h>

//

//typedef char BitElemType;

//typedef struct BiNode{

// BitElemType data;

// BiNode \*lchild;

// BiNode \*rchild;

//}BiNode, \*BiTree;

//

////辅助链队

//typedef BiTree LiqElemType;

//typedef struct LinkNode{

// LiqElemType data;

// struct LinkNode \*next;

//}LinkNode, \*LinkList;

//typedef struct LinkQueue{

// LinkList front, rear;

//}LinkQueue;

//void linkqueue\_init(LinkQueue &LQ){

// LQ.front=LQ.rear=(LinkList)malloc(sizeof(LinkNode));

// LQ.rear->next=NULL;

//}

//bool linkqueue\_isempty(LinkQueue LQ){

// return LQ.front==LQ.rear;

//}

//void linkqueue\_en(LinkQueue &LQ, LiqElemType in){

//// LinkList lq=(LinkList)malloc(sizeof(LinkNode));

// LinkList lq=(LinkList)calloc(1,sizeof(LinkNode));

// lq->data=in;

// lq->next=LQ.rear->next;

// LQ.rear->next=lq;

// LQ.rear=lq;

//}

//bool linkqueue\_de(LinkQueue &LQ, LiqElemType &out){

// if(linkqueue\_isempty(LQ)){

// return false;

// }

//// LQ.front=LQ.front->next; //空间无法释放

//// out=LQ.front->data;

// LinkList lq=(LinkList)malloc(sizeof(LinkNode));

// lq=LQ.front->next;

// out=lq->data;

// LQ.front->next=lq->next;

// if(LQ.rear==lq){

// LQ.rear=LQ.front;

// }

// free(lq);

// return true;

//}

//

////二叉树操作

//void bitree\_init(BiTree &BT){

// BT=NULL;

//}

//bool bitree\_isempty(BiTree BT){

// return NULL==BT;

//}

//void bitree\_create(BiTree &BT){

// BiTree bt;

// bitree\_init(bt);

// LinkQueue lq;

// linkqueue\_init(lq);

// LiqElemType pcur; //存放当前父子树结点

// BitElemType input;

//

// while(scanf("%c", &input)){

// if('\n'==input){

// break;

// }

// bt=(BiTree)calloc(1,sizeof(BiNode));

// bt->data=input;

// if(bitree\_isempty(BT)){

// BT=bt;

// linkqueue\_en(lq, bt);

// linkqueue\_de(lq, pcur);

// }else{

// if(NULL==pcur->lchild){

// pcur->lchild=bt;

// linkqueue\_en(lq, bt);

// }else if(NULL==pcur->rchild){

// pcur->rchild=bt;

// linkqueue\_en(lq, bt);

// linkqueue\_de(lq, pcur);

// }

// }

// }

//}

//void bitree\_preorder(BiTree BT){

// if(!bitree\_isempty(BT)){

// printf("%c", BT->data);

// bitree\_preorder(BT->lchild);

// bitree\_preorder(BT->rchild);

// }

//}

//void bitree\_inorder(BiTree BT){

// if(!bitree\_isempty(BT)){

// bitree\_inorder(BT->lchild);

// printf("%c", BT->data);

// bitree\_inorder(BT->rchild);

// }

//}

//void bitree\_postorder(BiTree BT){

// if(!bitree\_isempty(BT)){

// bitree\_postorder(BT->lchild);

// bitree\_postorder(BT->rchild);

// printf("%c", BT->data);

// }

//}

//void bitree\_leverorder(BiTree BT){

// LinkQueue lq;

// linkqueue\_init(lq);

// LiqElemType pcur;

// linkqueue\_en(lq, BT);

// while(!linkqueue\_isempty(lq)){

// linkqueue\_de(lq, pcur);

// printf("%c", pcur->data);

// if(NULL!=pcur->lchild){

// linkqueue\_en(lq, pcur->lchild);

// }

// if(NULL!=pcur->rchild){

// linkqueue\_en(lq, pcur->rchild);

// }

// }

//// printf("n\");

//}

//

//int main(){

//// //链队列代码测试

//// LinkQueue LQ;

//// linkqueue\_init(LQ);

//// linkqueue\_en(LQ, 8);

//// linkqueue\_en(LQ, 4);

//// bool ret;

//// LiqElemType out;

//// ret=linkqueue\_de(LQ, out);

//// printf("%d\n", out);

//// ret=linkqueue\_de(LQ, out);

//// printf("%d\n", out);

//// ret=linkqueue\_de(LQ, out);

//// if(ret){

//// printf("%d\n", out);

//// }else{

//// printf("false\n");

//// }

//

// BiTree BT;

// bitree\_init(BT);

// bitree\_create(BT); //层次建树

// bitree\_preorder(BT); //前序遍历

// printf("\n");

// bitree\_inorder(BT); //中序遍历

// printf("\n");

// bitree\_postorder(BT); //后序遍历

// printf("\n");

// bitree\_leverorder(BT); //层序遍历

// printf("\n");

//

// return 0;

//}

////作业19，C语言语法

////1.读取一个有符号数，左移输出、右移输出

////2.输入5个数，例如 8 5 3 5 8 ，输出只出现一次的数

////11/30

//

//#include <stdio.h>

//

//int main(){

// int input=0;

// scanf("%d", &input);

// printf("%2d\n", input<<1);

// printf("%2d\n", input>>1);

//

// int in[5];

// for(int i=0;i<5;i++){

// scanf("%d", &in[i]);

// }

// int result=0;

// for(int i=0;i<5;i++){

// result^=in[i];

// }

// printf("%d\n", result);

//

// return 0;

//}

////作业20，数据的机器级表示

////1.通过单步调试查看内存，计算1.456指数部分和小数部分

////例：4.5 输出 2 1，即4.5=2^2 \* 1.001B

////12/2

//

//#include <stdio.h>

//

//int main(){

// float i;

// i=1.456;

// //内存小端 35 5e ba 3f

// //00111111 10111010 01011110 00110101

// //0 01111111 01110100101111000110101

// //1.456=2^(2^7-1-127) \* 1.01110100101111000110101

// printf("%3d%3d\n", 0, 13);

// return 0;

//}

////作业20，汇编入门

////1.汇编语言 Intel 操作码：数据传送指令

////12/2

//

//#include <stdio.h>

//

//int main(){

// printf("mov\npush\npop\n");

//

// return 0;

//}