\*danh sách lk:

typedef struct \_listnode{

int num;

struct \_listnode \*next;

} ListNode;

void printList(ListNode \*head){

ListNode \*a=head;

if (a == NULL )

printf("khong co phan tu nao trong danh sach.\n");

while(a!=NULL){

printf("%d\n", a -> num);

a=a->next;

}

}

int main(){

ListNode \*node0, \*node1, \*head;

node0 = malloc(sizeof(ListNode));

node1 = malloc(sizeof(ListNode));

node0->num = 20;

node1->num = 30;

node0->next =node1;

node1->next =NULL;

head =node0;

printList(head);

}

void xuat(ListNode \*head,int n){

int i;

ListNode \*a=head;

for(i=0;i<n;i++){

printf("%d\n", a -> num);

a=a->next;

}

}

Nhập danh sách có n phần tử:

scanf("%d", &n);

if(n>0){

a=malloc(sizeof(ListNode));

head=a;

printf("cac phan tu nhap vao la: ");

for (i=1; i<n; i++){

scanf("%d", &a->num);

a->next=malloc(sizeof(ListNode));

a=a->next;

}

scanf("%d", &a->num);

a->next=NULL;

xuat(head,n);

}

ListNode \*findNode(ListNode \*head, int i) {

ListNode \*cur = head;

if ((head == NULL) || (i < 0)) {

printf("Danh sach lien ket rong hoac phan tu tim kiem khong ton tai");

return NULL;

}

while (i > 0 ) {

cur = cur->next ;

if (cur == NULL) {

printf("Phan tu tim kiem khong ton tai");

return NULL;

}

i--;

}

return cur;

}

void insertNode(ListNode \*\*pHead, int index, int value){

ListNode \*cur, \*newNode;

if (\*pHead == NULL || index == 0 ){

newNode = malloc(sizeof(ListNode));

newNode->num = value;

newNode->next = \*pHead ;

\*pHead = newNode ;

}

else if ((cur = findNode( \*pHead, index - 1 )) != NULL){

newNode = malloc(sizeof(ListNode));

newNode->num = value;

newNode->next = cur -> next ;

cur->next = newNode ;

} else printf("can not insert the new item at index %d!\n", index);

}

void removeNode(ListNode \*\*ptrHead, int index){

ListNode \*cur, \*pre;

if(index == 0){

cur = \*ptrHead;

\*ptrHead = cur->next;

free(cur);

} else{

pre = findNode(\*ptrHead, index);

cur = pre->next;

pre->next = cur->next;

free(cur);

}

}

int isEmptyStack(Stack \*s){

if((s->II).size == 0) return 1;

return 0;

}

void push(Stack \*s, int item){

insertNode(&(s->II.head),0,item);

s->II.size++;

}

int pop(Stack \*s){

int item;

if(!isEmptyStack(s)){

item = ((s->II).head)->num;

removeNode(&(s->II),0);

(s->II).size--;

return item;

}

else

return NULL;

}

//queue

void enqueue(Queue \*q, int item){

insertNode(&(q->II.head),0,item);

q->II.size++;

}

int dequeue(Queue \*q){

int item;

if(q->II.head!= NULL){

item = (q->II.head)->num;

removeNode(&(q->II.head),0);

(q->II).size = (q->II).size -1 ;

return item;

}

else{

printf("hang doi rong");

return 0;

}

}

//cây

void TreeTraversal\_PreOrder(BTNode \*cur){

if(cur != NULL){

printf("%d ", cur->item); tiền

TreeTraversal\_PreOrder(cur->left);

//printf("%d ", cur->item); trung

TreeTraversal\_PreOrder(cur->right);

//printf("%d ", cur->item); hậu

}

}

//trả về giá trị nút có giá trị bằng c có nút gốc là cur.

void BSTT(BTNode2 \*cur, char c){

if(cur == NULL){

return NULL;

}

if(c==cur->item){

return cur;

}

if(c< cur->item){

return BSTT(cur->left,c);

}

else

return BSTT(cur->right,c);

}

//chèn 1 nút có giá trị c trên cây có nút gốc là cur

BTNode2\* BSTT2(BTNode2 \*cur, char c){

if(c==cur->item) return NULL;

if(c<cur->item){

if(cur->left == NULL) return cur;

else BSTT2(cur->left,c);

}

else {

if(cur->right == NULL){

return cur;

}

BSTT2(cur->right,c);

}

}

//xóa nút trên cây

void removeNode(BTNode2 \*\*t, int num){

if((\*t)->item > num){

removeNode(&((\*t)->left),num);

}

else if((\*t)->item < num){

removeNode(&((\*t)->right),num);

}

else{

if((\*t)->left == NULL){

(\*t)=(\*t)->right;

}

else if((\*t)->right == NULL){

(\*t)=(\*t)->left;

}

else {

BTNode2 \*p = lastLeftChild(((\*t)->right));

(\*t)->item = p->item;

removeNode(&((\*t)->right), p->item);

}

}

}

//tìm chiều cao của cây

int chieucao(BTNode \*cur){

int l,r,x;

if(cur == NULL){

return 0;

}

l =chieucao(cur->left);

r =chieucao(cur->right);

x=max(l,r)+1;

return x;

}

//selection sort

void selection\_sort(int k[], int n){

int i,j,m,x;

for(i=1;i<n-1;i++){

m=i;

for(j=i+1;j<n;j++){

if(k[j]<k[m]){

m=j;

}

}

if(m!=i){

x=k[i];

k[i]=k[m];

k[m]=x;

}

}

}

//insert sort

void insert\_sort(int k[], int n){

int i,j,x;

for(i=1;i<n;i++){

x=k[i];

j=i-1;

while(x < k[i] ){

k[j+1] = k[j];

j=j-1;

}

k[j+1] = x;

}

}

//bubble sort

void bubble\_sort(int k[], int n){

int i,j,x;

for (i=1;i<n;i++){

for(j=n-1;j>i;j--){

if(k[j]<k[j-1]){

x = k[j];

k[j]=k[j-1];

k[j-1]=x;

}

}

}

}

//quick sort

int part(int k[],int LB,int UB){

int i,m;

i=LB+1;

int j=UB;

while(i<j){

while(k[i]<k[LB]) i=i+1;

while(k[j]>k[LB]) j=j-1;

if(i<j){

m = k[i];

k[i] = k[j];

k[j] = m;

i = i+1;

j=j-1;

}

}

if(k[LB] > k[j]){

m = k[LB];

k[LB] = k[j];

k[j] = m;

}

return j;

}

void QuickSort(int k[], int LB,int UB){

int j;

if(LB < UB){

j = part(k,LB,UB);

QuickSort(k,LB,j-1);

QuickSort(k,j+1,UB);

}

}

//heap sort

void ADJUST(int k[], int i, int n){

int key = k[i];

int j = 2\*i;

int a = floor(j/2);

while(j<=n){

if(j<n && (k[j] < k[j+1])) j = j+1;

if(key > k[j]){

k[j/2] = key;

return;

}

k[j/2] = k[j];

j = j \* 2;

ADJUST(k,j,n);

}

k[ (j/2) ] = key;

}

void HeapSort(int k[],int n){

int i;

int tmp;

for(i=floor(n/2);i>=1;i--){

ADJUST(k,i,n);

}

for(i=n-1;i>=1;i--){

tmp = k[1];

k[1]= k[i+1];

k[i+1] = tmp;

ADJUST(k,1,i);

}

}

//merge sort

void Merge(int b[], int h,int t, int k){

int c[t-h+1];

int i=0, u= h, v=t+1;

for(i=0;i<=t-h+1;i++){

c[i] = b[h+i];

}

i=0;

while( i<=t-h){

if(v<=k && b[v] < c[i]){

b[u] = b[v];

u++;

v++;

}

else{

b[u] = c[i];

u++;

i++;

}

}

while(v <= t){

b[u] = b[v];

u++;

v++;

}

}

void MergeSort(int b[], int h, int k){

if(k-h<1) return;

int t=(h+k)/2;

MergeSort(b,h,t);

MergeSort(b,t+1,k);

Merge(b,h,t,k);

}

