实验一

|  |  |  |
| --- | --- | --- |
| 姓名： | | 学号： |
| 班级： | | 专业：人工智能 |
| 实验名称： | 线性表的基本操作 | |
| 实验目的： | 1、掌握线性表的定义；  2、掌握线性表的基本操作，如建立、查找、插入和删除等。 | |
| 实验要求： | | |
| (1) 程序要添加适当的注释，程序的书写要采用缩进格式。  (2) 程序要具在一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如插入删除时指定的位置不对等等。  (3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。  （4）每项任务给出代码及运行截图 | | |
| 实验内容： | | |
| 分别定义顺序表和链表，存储内容自行设计，使其具有如下功能：  (1) 根据指定元素个数，逐个输入元素信息；  (2) 显示线性表中所有元素信息；  (3) 根据元素值进行查找，返回此元素的位置；  (4) 根据指定的位置可返回相应的元素信息； （只操作链表）  (5) 给定一个元素信息，插入到表中指定的位置；  (6) 删除指定位置的元素；  (7) 统计表中元素个数（只操作链表）。  (8) 归并两个有序链表（选做） | | |
| 实验代码及截图 | | |
| //线性表（顺序表）  #define \_CRT\_SECURE\_NO\_WARNINGS//Vscode的scanf报错  #include<stdio.h>  #include<stdlib.h>  #include<string>  #include<iostream>  using namespace std;  #define TRUE 1  #define False 0  #define OK 1  #define ERROR 0  #define overflow -2  #define LIST\_INIT\_SIZE 100 //线性表储存空间  #define LISTINCREMENT 10 //线性表增长空间  typedef int Status;  typedef int ElemType;  typedef struct //线性表的储存结构  {  ElemType\* elem; //一维数组，采用动态分配  int length; //线性表的长度  int listsize; //当前分配的数组大小  }SqList;  Status InitList\_Sq(SqList &L) { //顺序表的初始化  L.elem = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType)); //通过Elemtype\*为struct分配大小为INIT\_SIZE的储存空间  if (!L.elem)return(overflow);  L.length = 0;  L.listsize = LIST\_INIT\_SIZE;  return OK;  }//InitList\_Sq  Status Clearlist\_Sq(SqList &L) { //顺序表的制空  L.length = 0;  return OK;  }//ClearList\_Sq;  Status ListInsert\_Sq(SqList& L,int i,ElemType e) { //在线性表L的第i个元素前插入元素e，其中i的取值为1~sizeof(L+1)  ElemType\* p, \* q;  if (i<1 || i>L.length + 1)return ERROR; //插入位置不合法  q = &(L.elem[i - 1]); //q指示插入位置  for (p = &(L.elem[L.length - 1]); p >= q; --p)  \*(p + 1) = \*p;//插入位置及之后的元素右移  \*q = e;  ++L.length;  return OK;  }//ListInsert\_sq  Status ListDelete\_Sq(SqList& L, int i, ElemType& e) { //删除线性表L的第i个元素，并用参数e返回被删元素的值，其中i的取值为1~L.length  ElemType\* p, \* q;  if ((i < 1) || (i > L.length))return ERROR;  p = &(L.elem[i - 1]); //p为被删元素的位置  e = \*p; //被删元素赋值给e  q = L.elem + L.length - 1;//表示元素的位置  for (++p; p <= q; ++p)\*(p - 1) = \*p;  --L.length;  return OK;  }//ListInsert\_Sq  Status ListShow\_Sq(SqList& L) {//输出线性表的每一个元素  printf("-------------------------\n");  printf("The sorted list is:\n");  for (int i = 0; i < L.length; i++) {  printf("%d\t", L.elem[i]);  printf("\n");  }  printf("-------------------------\n");  return OK;  }//ListShow\_Sq  Status ListSort\_Sq(SqList& L) {  int i, j, k; // 三个中间变量i和k是位置索引  ElemType t; // 临时变量，用于交换元素值  for (i = 0; i < L.length - 1; i++) {  k = i; // 初始化 k 为当前轮次的最小元素索引  // 在未排序部分中找到最小元素的索引  for (j = i + 1; j < L.length; j++) {  if (L.elem[k] > L.elem[j]) {  k = j; // 更新最小元素位置  }  }  // 如果最小元素不在当前位置（找到了最小值），则进行交换  if (k != i) {  t = L.elem[i];  L.elem[i] = L.elem[k];  L.elem[k] = t;  }  }  return OK;  }  void MergeList(SqList& La, SqList& Lb, SqList& Lc) {  int i = 0, j = 0, k = 0; // 初始化索引 i、j、k 分别为 0  while (i < La.length && j < Lb.length) { // 循环直到 La 或 Lb 中的元素全部取完  if (La.elem[i] <= Lb.elem[j]) { // 如果 La 中的元素较小或相等  ListInsert\_Sq(Lc, ++k, La.elem[i]); // 将 La 中的元素插入到 Lc 中  ++i; // 更新 La 索引  }  else { // 如果 Lb 中的元素较小  ListInsert\_Sq(Lc, ++k, Lb.elem[j]); // 将 Lb 中的元素插入到 Lc 中  ++j; // 更新 Lb 索引  }  }  while (i < La.length) { // 处理剩余的 La 元素  ListInsert\_Sq(Lc, ++k, La.elem[i]);  ++i;  }  while (j < Lb.length) { // 处理剩余的 Lb 元素  ListInsert\_Sq(Lc, ++k, Lb.elem[j]);  ++j;  }  }  Status DestroyList\_Sq(SqList& L) { //顺序表的销毁  if (L.elem)  free(L.elem);  L.elem = NULL;  return OK;  }//DestroyList\_Sq  int main() {  SqList La, Lb, Lc;  int i;  InitList\_Sq(La);  InitList\_Sq(Lb);  InitList\_Sq(Lc);  // 向 La 中插入元素  int n1,ele1;  cout << "-------------------------" << endl;  cout << "Input the number of the elements in the la" << endl;  cin >> n1;  cout << "-------------------------" << endl;  cout << "Input the elements of the la" << endl;  for (i = 1; i <= n1; i++) {  scanf\_s("%d", &ele1);  ListInsert\_Sq(La, i, ele1);  }  cout << "-------------------------" << endl;  ListShow\_Sq(La); // 输出 La 的元素  cout << "-------------------------" << endl;  // 向 Lb 中插入元素  int n2,ele2;  cout << "-------------------------" << endl;  cout << "Input the number of the elements in thr lb" << endl;  cin >> n2;  cout << "-------------------------" << endl;  cout << "Input the elements of the lb" << endl;  for (i = 1; i <= n2; i++) {  scanf\_s("%d", &ele2);  ListInsert\_Sq(Lb, i, ele2);  }  cout << "-------------------------" << endl;  printf("Lb: ");  ListShow\_Sq(Lb); // 输出 Lb 的元素  cout << "-------------------------" << endl;  // 合并 La 和 Lb，并将结果存入 Lc  cout << "--------------------------" << endl;  cout << "The merged list lc:\n" << endl;  MergeList(La, Lb, Lc);  ListShow\_Sq(Lc); // 输出 Lc 的元素  // 对 Lc 进行排序  cout << "-------------------------" << endl;  ListSort\_Sq(Lc);  printf("Lc (sorted): ");  ListShow\_Sq(Lc); // 输出排序后的 Lc 的元素  cout << "-------------------------" << endl;  // 删除 Lc 中的第一个元素  ElemType e;  ListDelete\_Sq(Lc, 1, e);  printf("Deleted element from Lc: %d\n", e);  printf("Lc: ");  ListShow\_Sq(Lc); // 输出删除元素后的 Lc 的元素  cout << "-------------------------" << endl;  DestroyList\_Sq(La);  DestroyList\_Sq(Lb);  DestroyList\_Sq(Lc);  return 0;  } | | |
| 线性表（链表）：  #define \_CRT\_SECURE\_NO\_WARNINGS//Vscode的scanf报错  #include <cstdio>  #include <iostream>  #include <cstdlib>  #include <stdio.h>  #define OK 1  #define ERROR 0  #define overflow -2  typedef int Elemtype;  typedef int Status;  typedef struct Node { //对链节进行定义  Elemtype data; //数据域  struct Node\* next; //指针域(注意这里应该是Node本身的指针域)  } Node;  /\*typedef struct Node\* Linklist; //Linklist是指向首元节点的首指针  typedef struct DuNode { //对双向链表进行定义  Elemtype data; //数据域  struct DuNode\* prior; //前向指针  struct DuNode\* next; //后向指针  } DuNode;  typedef DuNode\* DuLinklist;\*/ //这里只是记录双向链表的初始化，未使用  Status InitList(Linklist& L) { //构造一个空的单链表L（初始化首元节点）  L = new Node; //头指针指向新生成的Node  L->next = NULL; //我们这时让在Node指针域的指针指向NULL避免野指针  return OK;  }  Status GetElem(Linklist L, int i, Elemtype& e) {//在带头节点的单链表L中根据i获取元素的值，用e返回第i个元素的值  Node\* p = L->next; //因为是中间节点所以指向next而不是NULL  Elemtype j = 1;  while (p != NULL && j < i) {  p = p->next;  ++j;  }  if (!p || j > i) return ERROR;  e = p->data;  return OK;  }  Node\* locateElem(Linklist L, Elemtype e) { //返回的是节点的地址所以这里使用Node起手  Node\* p = L->next; //都需要创造首元节点  while (p != NULL && p->data != e) {  p = p->next;  }  return p;  }  Status ListInsert(Linklist& L, int i, Elemtype e) {//指定位置插入  Node\* p = L;  Elemtype j = 0;  while (p != NULL && j < i - 1) {  p = p->next;  ++j;  }  if (!p || j > i - 1) return ERROR;  Node\* s = new Node;  s->data = e;  s->next = p->next; //隔空链接  p->next = s;  return OK;  }  Status ListDelete(Linklist& L, int i) {//删除指定位置节点  Node\* p = L;  Elemtype j = 0;  while (p->next != NULL && j < i - 1) {  p = p->next;  ++j;  }  if (!(p->next) || (j > i - 1)) return ERROR;  Node\* q = p->next;  p->next = q->next;  delete q;  return OK;  }  void CreateList\_Q(Linklist& L, int n) {//插入元素：  L = new Node;  L->next = NULL;  for (int i = 0; i < n; ++i) {  Node\* p = new Node;  std::cin >> p->data;  p->next = L->next;  L->next = p;  }  }  void CreatList\_R(Linklist& L, int n) {//后插法插入  L = new Node;  L->next = NULL;  Node\* r = L;  for (int i = 0; i < n; ++i) {  Node\* p = new Node;  std::cin >> p->data;  p->next = NULL;  r->next = p;  r = p;  }  }  void MergeList\_L(Linklist& LA, Linklist& LB, Linklist& LC) {  Node\* pa = LA->next;  Node\* pb = LB->next;  Node\* pc = LC;  while (pa && pb) {  if (pa->data <= pb->data) {  pc->next = pa;  pc = pa;  pa = pa->next;  }  else {  pc->next = pb;  pc = pb;  pb = pb->next;  }  }  pc->next = pa ? pa : pb;  }  int main() {  Linklist L1,L2;  InitList(L1);  InitList(L2);  printf("-------------------------\n");  printf("Inserting elements at specific positions:Insert the quantity of the inserted number\n");  int n;  scanf("%d", &n);  printf("-------------------------\n");  printf("Insert the elements of the inserted number\n");  for (int i = 1; i <= n; i++) {  int ele1;  scanf("%d", &ele1);  ListInsert(L1, i, ele1);  }  printf("-------------------------\n");  printf("The elements of the l1 is: \n ");  Node\* p = L1->next;  while (p != NULL) {  printf("%d\n ", p->data);  p = p->next;  }  printf("------------------------\n");  printf("Getting an element at a specific position:Insert the position of the inserted number\n");  printf("------------------------\n");  int pos; scanf("%d",&pos);  Elemtype e;  if (GetElem(L1, pos, e) == OK) {  printf("Element at the fixed position : %d\n", e);  printf("------------------------\n");  }  else {  printf("Failed to get element at position .\n");  printf("------------------------\n");  }  printf("Finding the location of an element,Insert the position of the located number\n");  printf("------------------------\n");  int pos2; scanf("%d", &pos2);  Node\* location = locateElem(L1, pos2);  if (location != NULL) {  printf("Location of element : %p\n", location);  printf("------------------------\n");  }  else {  printf("Element 20 not found.\n");  printf("------------------------\n");  }  printf("Deleting an element at a specific position,Insert the position of the specific deleting number\n");  printf("------------------------\n");  int pos3; scanf("%d", &pos3);  if (ListDelete(L1, pos3) == OK) {  printf("Element at noticedposition deleted successfully.\n");  printf("------------------------\n");  }  else {  printf("Failed to delete element at position 3.\n");  printf("------------------------\n");  }  printf("Displaying the updated list,The list which edited before: \n");  p = L1->next;  while (p != NULL) {  printf("%d \n", p->data);  p = p->next;  }  printf("------------------------\n");    return 0;  } | | |
| 实验结论： | | |
| 复习了结构体，链表的应用。  对宏定义，类型定义有了深入的了解。  掌握了C++语言线性表的写改删查。 | | |