学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 张蕊晗 | **学号** | 2020905198 |
| **学院** | 信息工程学院 | **专业** | 电子信息工程 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 复习了散列的相关知识 |
| 本周练习过的代码（例） |
| （本周练习写过的代码，如实填写，疑似抄袭会单独测试）  *代码1：*  *#include<stdio.h>*  *int main()*  *{*  *printf("HelloWorld\n");*  *return 0;*  *}*  *代码2：*  …….. |
| 本周练习过的代码 |
| #include<stdio.h>  #include<stdlib.h>  typedef struct NODE \*Node;  typedef Node \* List;  typedef struct HASHTABLE HashTable;  struct NODE{  int data;  Node next;  };  struct HASHTABLE{  int size;//表示哈希表的大小  List arr;  };  //初始化哈希表  void init(HashTable &H,int size){  H.size = size;//初始化哈希表的大小  H.arr = (List)malloc(sizeof(Node) \* size);//给哈希表中的指针数组分配空间  if(H.arr == NULL){  printf("指针数组分配空间失败!!!\n");  exit(0);  }  int i;  for(i = 0; i < size; i++){  H.arr[i] = (Node)malloc(sizeof(struct NODE));//给每一个节点分配空间  if(H.arr[i] == NULL){  printf("节点分配失败!!!\n");  exit(0);  }  //每一个下标对应一条链表，并且这条链表是一个带有假节点的链表  H.arr[i]->next = NULL;  }  }  int hash(HashTable &H,int key){  return key % H.size;//利用除留取余法，从而获取key在哈希表中的地址  }  /\*  在哈希表中查找关键字key:  1、首先需要利用除留取余法获得key在哈希表中所处的链表位置  2、找到所处的链表之后，遍历链表，判断是否能找到值尾key的节点，如果能找到，  就将这个节点返回，否则返回null  \*/  Node find(HashTable &H,int key){  int pos;  Node L,cur;  pos = hash(H,key);//获取key在哈希表中的位置  L = H.arr[pos];//获取key所在地址的链表  cur = L->next;//由于L是一个带有假节点的链表，那么L->next才是链表真正的头结点  while(cur != NULL && cur->data != key){  //如果当前的节点不为空，并且当前节点的值不是要找的关键字，那么继续遍历链表  cur = cur->next;  }  return cur;  }  /\*  在哈希表中插入关键字  判断关键字是否已经存在哈希表中了，如果存在了，那么不进行任何操作，否则就将其  插入到对应的链表的链表头处  \*/  void insert(HashTable &H,int key){  Node L,p;  p = find(H,key);  if(p == NULL){  //如果p为空，说明哈希表中并不存在这个这个关键字的节点，那么就将这个新节点插入到对应的链表头的位置  L = H.arr[key % H.size];//获取关键字所处的链表  p = (Node)malloc(sizeof(struct NODE));  p->next = L->next;  p->data = key;  L->next = p;  printf("插入成功!!!\n");  }else{  printf("关键字%d已经在哈希表中存在，所处的链表下标为%d\n",key,key % H.size);  }  }  void deleteElement(HashTable &H,int key){  Node L,p,tmp;  p = find(H,key);  if(p != NULL){  //如果p不为空，说明哈希表中不存在这个这个关键字的节点  L = H.arr[key % H.size];//获取关键字所处的链表  p = L;  while(p->next != NULL && p->next->data != key){  //找到删除节点的前一个节点  p = p->next;  }  tmp = p->next;//找到了删除的节点  p->next = p->next->next;  free(tmp);//释放待删除的节点  printf("删除成功!!!\n");  }else{  printf("关键字%d在哈希表中不存在\n",key);  }  }  void display(HashTable &H){  int i;  Node L;  for(i = 0; i < H.size; i++){  L = H.arr[i]->next;  if(L == NULL){  printf("NULL\n");  }else{  while(L != NULL){  printf("%5d",L->data);  L = L->next;  }  printf("\n");  }  }  }  int main(){  HashTable h;  int n,i,key;  printf("请输入哈希表的大小:");  scanf("%d",&n);  init(h,n);  printf("请输入元素的个数:");  scanf("%d",&n);  printf("请输入各个关键字:");  for(i = 0; i < n; i++){  scanf("%d",&key);  insert(h,key);  }  while(1){  printf("请输入选项: 1、插入 2、查找 3、删除 4、遍历哈希表 0、退出\n");  scanf("%d",&n);  switch(n){  case 1:  printf("请输入待插入数字:");  scanf("%d",&key);  insert(h,key);  break;  case 2:  printf("请输入待查找数字:");  scanf("%d",&key);  if(find(h,key)){  printf("找到了，所处的链表下标为%d\n",key % h.size);  }else{  printf("哈希表中无法找到%d\n",key);  }  break;  case 3:  printf("请输入待删除数字:");  scanf("%d",&key);  deleteElement(h,key);  break;  case 4:  display(h);  break;  case 0:  printf("退出系统");  exit(0);  }  }  return 0;  }  /\*  线性探测法:  发生冲突的时候，往后移，知道找到第一个空的  单元，那么就将数字压入到这个单元即可  \*/  #include<stdio.h>  #include<stdlib.h>  #define Empty 0  #define Occupy 1  typedef struct HASHTABLE HashTable;  typedef struct ELEMENT Element;  struct ELEMENT{  int val;//值  int state;//表示这个元素是否为空，如果是0，表示空，否则不为空  };  struct HASHTABLE{  Element \*arr;//用一个整形数组来表示哈希表  int size;//哈希表的大小  };  //初始化哈希表  void init(HashTable &H,int size){  H.arr = (Element \*)malloc(sizeof(Element) \* size);  if(H.arr == NULL){  printf("哈希表数组分配空间失败!!!\n");  exit(0);  }  int i;  //初始化哈希表，表示是一个空表  for(i = 0; i < size; i++){  H.arr[i].state = Empty;  }  H.size = size;  }  //定义哈希函数  int hash(HashTable &H,int key){  return key % H.size;  }  //查找哈希表中key对应的下标  int findElement(HashTable &H,int key){  int pos;  pos = hash(H,key);//找到关键字在哈希表中散列的地址  while(H.arr[pos].state != Empty && H.arr[pos].val != key){  //如果当前下标不为空，并且对应的值不是我们要找的关键字，那么就后移  pos++;  /\*  pos %= H.size;  \*/  if(pos >= H.size)  pos -= H.size;//如果后移的时候，对应的pos大于等于H.size，那么我们需要回到下标为pos - H.size的位置  }  /\*  这时候返回的pos有两种情况，可能是哈希表中pos下标为空，那么我们  在进行相应的操作的时候，需要判断当前的元素是否为空才可以进行，  也有可能pos下标不为空，同时这个下标对应的元素值就是我们要找的关键字  \*/  return pos;  }  //插入元素  void insertElement(HashTable &H,int key){  int pos;  pos = findElement(H,key);//查找key在哈希表中的位置  /\*  判断pos下标在哈希表中是否为空，如果为空，那么就说明没有找到  这时候我们将key压入到pos下标，否则不用进行任何操作，从而避免了  插入重复数字  \*/  if(H.arr[pos].state == Empty){  H.arr[pos].val = key;  H.arr[pos].state = Occupy;//插入元素之后，同时需要说明这个下标已经有值了  }  }  /\*  删除元素：利用懒惰删除，我们并没有真正的将这个元素从  哈希表中删除，然后删除元素后的数都要往前移，相反，我们只是  将当前删除元素对应的状态标记为Empty，这时候，下次插入元素的时候，就可以  在这个下标插入了，从而达到删除的效果  \*/  void deleteElement(HashTable &H,int key){  int pos;  pos = findElement(H,key);  if(H.arr[pos].state == Occupy){  //判断是否能找到要删除的元素，如果是Occupy，说明找到了  printf("删除的元素%d在哈希表中的下标为%d\n",key,pos);  H.arr[pos].state = Empty;//删除之后，将这个下标对应的状态标记为Empty  }  }  void display(HashTable &H){  int i;  for(i = 0; i < H.size; i++){  if(H.arr[i].state == Empty){  printf("%5d: NULL\n",i);  }else{  printf("%5d: %d\n",i,H.arr[i].val);  }  }  }  int main(){  HashTable h;  int n,i,key,pos;  printf("请输入哈希表的大小:");  scanf("%d",&n);  init(h,n);//初始化哈希表  printf("请输入元素的个数:");  scanf("%d",&n);  printf("请输入各个关键字:");  for(i = 0; i < n; i++){  scanf("%d",&key);  insertElement(h,key);  }  while(1){  printf("请输入选项: 1、插入 2、查找 3、删除 4、遍历哈希表 0、退出\n");  scanf("%d",&n);  switch(n){  case 1:  printf("请输入待插入数字:");  scanf("%d",&key);  insertElement(h,key);  break;  case 2:  printf("请输入待查找数字:");  scanf("%d",&key);  pos = findElement(h,key);  if(h.arr[pos].state == Occupy){  /\*  获取key的关键字之后，我们需要判断这个下标在哈希表中是否为空，如果  为空，说明没有办法在哈希表中找到key，否则找到了  \*/  printf("找到了，所处的哈希表下标为%d\n",pos);  }else{  printf("哈希表中无法找到%d\n",key);  }  break;  case 3:  printf("请输入待删除数字:");  scanf("%d",&key);  deleteElement(h,key);  break;  case 4:  display(h);  break;  case 0:  printf("退出系统");  exit(0);  }  }  return 0;  }  #include<stdio.h>  #include<stdlib.h>  #define Empty 0  #define Occupy 1  typedef struct HASHTABLE HashTable;  typedef struct ELEMENT Element;  struct ELEMENT{  int val;//值  int state;//表示这个元素是否为空，如果是0，表示空，否则不为空  };  struct HASHTABLE{  Element \*arr;//用一个整形数组来表示哈希表  int size;//哈希表的大小  };  //初始化哈希表  void init(HashTable &H,int size){  H.arr = (Element \*)malloc(sizeof(Element) \* size);  if(H.arr == NULL){  printf("哈希表数组分配空间失败!!!\n");  exit(0);  }  int i;  //初始化哈希表，表示是一个空表  for(i = 0; i < size; i++){  H.arr[i].state = Empty;  }  H.size = size;  }  //定义哈希函数  int hash(HashTable &H,int key){  return key % H.size;  }  int hash2(int key,int p){  return p - (key % p);  }  //查找哈希表中key对应的下标  int findElement(HashTable &H,int key){  int pos,oldPos,d,count;  count = 0;//count表示冲突次数  oldPos = hash(H,key);//找到关键字在哈希表中散列的地址  d = hash2(key,7);//这里将p的值为7  pos = oldPos;  while(H.arr[pos].state != Empty && H.arr[pos].val != key){  //如果当前元素不为空，并且不是要找的关键字，那么继续散列  count++;  //双散列  pos = oldPos + count \* d;  while(pos >= H.size)  pos -= H.size;  }  /\*  这时候返回的pos有两种情况，可能是哈希表中pos下标为空，那么我们  在进行相应的操作的时候，需要判断当前的元素是否为空才可以进行，  也有可能pos下标不为空，同时这个下标对应的元素值就是我们要找的关键字  \*/  return pos;  }  //插入元素  void insertElement(HashTable &H,int key){  int pos;  pos = findElement(H,key);//查找key在哈希表中的位置  /\*  判断pos下标在哈希表中是否为空，如果为空，那么就说明没有找到  这时候我们将key压入到pos下标，否则不用进行任何操作，从而避免了  插入重复数字  \*/  if(H.arr[pos].state == Empty){  H.arr[pos].val = key;  H.arr[pos].state = Occupy;//插入元素之后，同时需要说明这个下标已经有值了  }  }  /\*  删除元素：利用懒惰删除，我们并没有真正的将这个元素从  哈希表中删除，然后删除元素后的数都要往前移，相反，我们只是  将当前删除元素对应的状态标记为Empty，这时候，下次插入元素的时候，就可以  在这个下标插入了，从而达到删除的效果  \*/  void deleteElement(HashTable &H,int key){  int pos;  pos = findElement(H,key);  if(H.arr[pos].state == Occupy){  //判断是否能找到要删除的元素，如果是Occupy，说明找到了  printf("删除的元素%d在哈希表中的下标为%d\n",key,pos);  H.arr[pos].state = Empty;//删除之后，将这个下标对应的状态标记为Empty  }  }  void display(HashTable &H){  int i;  for(i = 0; i < H.size; i++){  if(H.arr[i].state == Empty){  printf("%5d: NULL\n",i);  }else{  printf("%5d: %d\n",i,H.arr[i].val);  }  }  }  int main(){  HashTable h;  int n,i,key,pos;  printf("请输入哈希表的大小:");  scanf("%d",&n);  init(h,n);  printf("请输入元素的个数:");  scanf("%d",&n);  printf("请输入各个关键字:");  for(i = 0; i < n; i++){  scanf("%d",&key);  insertElement(h,key);  }  while(1){  printf("请输入选项: 1、插入 2、查找 3、删除 4、遍历哈希表 0、退出\n");  scanf("%d",&n);  switch(n){  case 1:  printf("请输入待插入数字:");  scanf("%d",&key);  insertElement(h,key);  break;  case 2:  printf("请输入待查找数字:");  scanf("%d",&key);  pos = findElement(h,key);  if(h.arr[pos].state == Occupy){  printf("找到了，所处的哈希表下标为%d\n",pos);  }else{  printf("哈希表中无法找到%d\n",key);  }  break;  case 3:  printf("请输入待删除数字:");  scanf("%d",&key);  deleteElement(h,key);  break;  case 4:  display(h);  break;  case 0:  printf("退出系统");  exit(0);  }  }  return 0;  } |

1. 该表信息将会作为你报名申请的重要依据，请认真仔细填写。
2. 培训班有严格的制度，请认真阅读规则并结合自身情况填写该表。
3. 完成该表填写后以“xxx学习情况”命名，及时上传作业。

最后希望大家能够加入我们，一起努力，共同进步！