**顺序表的各功能实现以及与单链表的功能对比实验**

190110304王君涛

功能一：产生顺序表

具体函数代码：

void SeqList::InitList()

{

p = new char[maxSize];

last = 0;//Last用来记录表中的元素个数

}

算法说明：

时间复杂度为O(1)级别，运用new来动态分配存储空间，方便释放

功能二：向顺序表中批量导入元素

具体函数代码：

void SeqList::import\_data(const char\* t)

{

if (strlen(t) +last> maxSize)

{

cout << "导入的字符串过长！" << endl;

return;

}

for (int i = last; i < strlen(t); i++)

{

ListInsert(i + 1, \*(t + i));

}

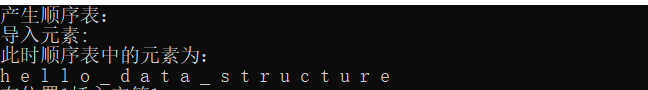
}

算法说明：

首先检测批量导入元素后，是否超过表的储存的最大值。之后从第Last+1个位置开始向表后插入元素，实现批量插入元素

与链表的对比：

主要区别为功能三的时间复杂度的对比



功能三：向顺序表中某个位置插入元素

具体函数代码：

void SeqList::ListInsert(int i, char e)

{

if (last>= maxSize)

{

cout << "储存空间已满！" << endl;

return ;

}

if (i > last + 1)

{

cout << "该顺序表仅有" << last << "个元素，插入位置错误！" << endl;

return;

}

if (last == 0)

{

\*p = e;

last++;

return;

}

if (i==last+1)

{

\*(p + last) = e;

last++;

return;

}

for (int n = last - 1; n >= i - 1; n--)

{

\*(p + n + 1) = \*(p + n);

}

\*(p + i - 1) = e;

last++;

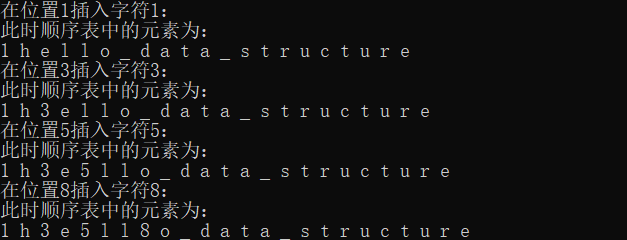
}

算法说明：

首先判断是否储存空间已满，之后判断插入位置是否存在，如果表中无元素在，则只能插入到位置1，若插入位置为表尾，则直接插入，时间复杂度为O(1)级，若插入的元素位置i为表中，则从最后一个元素开始到第i个元素为止依次将其位置+1，将第i个位置的元素数值赋值给待插入元素值，因此，总的平均时间复杂度为O(n)级

与链表的对比：

若已经确定插入位置，链表只需要改变对应位置的前驱的指针和插入元素的后继指针，时间复杂度为O(1)级别而顺序表由上述说明，时间复杂度为O(n）级别



功能四：删除对应位置的元素

具体函数代码：

void SeqList::ListDelete(int i)

{

if (last == 0)

{

cout << "表中无数据用来删除" << endl;

return;

}

if (i > last)

{

cout << "该顺序表仅有" << last << "个元素，操作错误！" << endl;

return;

}

for (int n = i - 1; n <= last - 2; n++)

{

\*(p + n) = \*(p + n + 1);

}

last--;

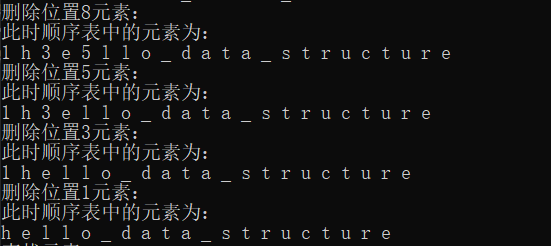
}

算法说明：

首先检测是否为空顺序表，如果是则无法进行删除操作，之后检测输入的删除位置，若位置不合适，则无法删除。之后从第i个元素开始到倒数第二个元素为止不断将元素后移，第i个元素被第i+1个元素替代，实现元素的删除，时间复杂度为O(n)级别。

与链表的对比：

同插入操作，只要确定删除元素的指针，则只需要改变第i-1个元素的后继指针，指向第i+1个元素，时间复杂度仅为O(1)



功能五：打印顺序表元素

具体函数代码：

void SeqList::printdata()

{

cout << "此时顺序表中的元素为：" << endl;

for (int i = 0; i < last ; i++)

cout << \*(p + i) << " ";

cout << endl;

}

算法说明：

将当前顺序表的各个元素按顺序打印出来，便于观察

功能五：查找与给定元素相等的顺序表中的元素位置

具体函数代码：

void SeqList::search\_elem(char e)

{

bool flag = false;

for (int i = 0; i < last; i++)

{

if (\*(p + i) == e)

{

cout << "第" << i + 1 << "个元素的值等于" << e << endl;

flag = true;

}

}

if (flag == false)

cout << "顺序表中无等于" << e << "的元素" << endl;

}

算法说明：

遍历顺序表查找元素值为给定元素的位置，时间复杂度为O(n)级别。

与链表的对比：

单链表的该功能与顺序表类似，也是从第一个元素遍历整个表，返回与查找值相等的元素位置



功能六：删除顺序表

具体函数代码：

void SeqList::DestroyList()

{

last = 0;

delete []p;

}

算法说明：

顺序表的元素个数改为0，释放内存空间。



源码：

|  |
| --- |
| #include <iostream>  using namespace std;  #define maxSize 100  class SeqList  {  private:  char\* p;  int last;  public:    void InitList();  void DestroyList();  void ListInsert(int i, char e);  void ListDelete(int i);  void printdata();  void import\_data(const char\* t);  void search\_elem(char e);  };  void SeqList::InitList()  {  p = new char[maxSize];  last = 0;  }  void SeqList::DestroyList()  {  last = 0;  delete []p;  }  void SeqList::ListInsert(int i, char e)  {  if (last>= maxSize)  {  cout << "储存空间已满！" << endl;  return ;  }  if (i > last + 1)  {  cout << "该顺序表仅有" << last << "个元素，插入位置错误！" << endl;  return;  }  if (last == 0)  {  \*p = e;  last++;  return;  }  if (i==last+1)  {  \*(p + last) = e;  last++;  return;  }  for (int n = last - 1; n >= i - 1; n--)  {  \*(p + n + 1) = \*(p + n);  }  \*(p + i - 1) = e;  last++;  }  void SeqList::ListDelete(int i)  {  if (last == 0)  {  cout << "表中无数据用来删除" << endl;  return;  }  if (i > last)  {  cout << "该顺序表仅有" << last << "个元素，操作错误！" << endl;  return;  }  for (int n = i - 1; n <= last - 2; n++)  {  \*(p + n) = \*(p + n + 1);  }  last--;  }  void SeqList::printdata()  {  cout << "此时顺序表中的元素为：" << endl;  for (int i = 0; i < last ; i++)  cout << \*(p + i) << " ";  cout << endl;  }  void SeqList::import\_data(const char\* t)  {  if (strlen(t) +last> maxSize)  {  cout << "导入的字符串过长！" << endl;  return;  }  for (int i = last; i < strlen(t); i++)  {  ListInsert(i + 1, \*(t + i));  }  }  void SeqList::search\_elem(char e)  {  bool flag = false;  for (int i = 0; i < last; i++)  {  if (\*(p + i) == e)  {  cout << "第" << i + 1 << "个元素的值等于" << e << endl;  flag = true;  }  }  if (flag == false)  cout << "顺序表中无等于" << e << "的元素" << endl;  }  int main()  {  SeqList L;  const char\* t = "hello\_data\_structure";  cout << "产生顺序表：" << endl;  L.InitList();  cout << "导入元素:" << endl;  L.import\_data(t);  L.printdata();  cout << "在位置1插入字符1：" << endl;  L.ListInsert(1, '1');  L.printdata();  cout << "在位置3插入字符3：" << endl;  L.ListInsert(3, '3');  L.printdata();  cout << "在位置5插入字符5：" << endl;  L.ListInsert(5, '5');  L.printdata();  cout << "在位置8插入字符8：" << endl;  L.ListInsert(8, '8');  L.printdata();  cout << "删除位置8元素：" << endl;  L.ListDelete(8);  L.printdata();  cout << "删除位置5元素：" << endl;  L.ListDelete(5);  L.printdata();  cout << "删除位置3元素：" << endl;  L.ListDelete(3);  L.printdata();  cout << "删除位置1元素：" << endl;  L.ListDelete(1);  L.printdata();  cout << "查找元素:" << endl;  L.search\_elem('a');  L.search\_elem('b');  cout << "成功删除顺序表！" << endl;  L.DestroyList();  return 0;  } |