电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2018091602007

姓 名 王乐卿

（实验） 课程名称 程序设计与算法基础Ⅱ

理论教师 刘峤

实验教师 刘峤

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：**王乐卿 **学号：**2018091602007  **指导教师：**刘峤

**实验地点：**基础实验大楼  **实验时间：**2019.4.21

1. **实验名称：**线性结构及相关算法的设计与应用-学生课程成绩查询程序
2. **实验学时：**8学时
3. **实验目的：**

1. 掌握磁盘读写文件的方法；

2. 单链表的创建、插入、查找、删除节点的方法；

3. 对单链表的排序方法

3. 输出单链表节点的方法；

4. 单链表的定义和使用方法；

5. 栈和队列的使用方法。

1. **实验原理：**

栈是一种先进先出，后进后出的常用数据结构，利用栈可以实现链表反向的功能。

队列是一种先进后出，后进先出的数据结构，利用队列可以动态管理链表中的元素，如实现两个队列的合并。

链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。由于链表的这些特点，可以动态的增添删减元素，因此采用链表作为本次实验的数据结构。

1. **实验内容：**

设计一个学生课程成绩查询程序，设有学生信息文件 student.dat，每个学生记录包括：学号 sno、姓名 sname、性别 sex、专业 major；课程信息文件 course.dat，课程记录包括：课程号 cno、课程名称 cname、课时数 classHours；课程成绩信息文件 courseGrade.dat，成绩记录包括学号 sno、课程号 cno、考试成绩 score。设计应用程序实现如下功能：

1. 输入10个学生记录，其中软件技术专业5人，人工智能专业5人，并存入文件student.dat中；
2. 输入3门课程（数据库、数据结构、程序设计）信息记录，并存入文件course.dat中；
3. 输入上述10位同学分别选修上述三门课程的考试成绩到文件courseGrade.dat中；
4. 从文件student.dat中读出学生信息，生成按照学号升序排列的单向链表，并在屏幕上显示输出；
5. 从文件course.dat中读出课程信息，生成按照课程号升序排列的单向链表，并在屏幕上显示输出；
6. 从文件courseGrade.dat中读出成绩信息，生成按照学号和课程号升序排列的单链表，并在屏幕上显示输出；
7. 查询所有学生所有课程的考试成绩，生成该课程的成绩单链表，要求包括学号、学生姓名、专业、课程名、考试成绩等信息，按照考试成绩降序排列，并将学生的该成绩信息输出到文件studentGrade.dat中，同时在屏幕上显示输出；
8. 在（7）的链表中，查询指定课程号的所有学生的考试成绩，生成该课程的成绩单链表，要求包括学号、学生姓名、专业、课程名、考试成绩等信息，按照考试成绩降序排列输出到屏幕上显示；
9. 在（7）的链表中，查询指定课程号的考试成绩小于60分的学生成绩信息，生成该课程的成绩链表，要求包括学号、学生姓名、专业、课程名、考试成绩等信息，并按照考试成绩降序排列在屏幕上显示输出；
10. 使用栈实现将（4）的单链表中的学生信息逆序生存新的链表；
11. 设计使用链式队列完成问题（7）的要求，建议使用带头指针和尾指针链式队列。
12. **实验器材（设备、元器件）：**

PC机一台，装有C语言集成开发环境。

1. **实验步骤：**
2. **定义结构体，明确学生信息、课程属性、成绩构成：**

学生信息包括学生姓名，学号，性别，专业；课程信息包括课程号，课程名称和学时；成绩信息包括学号，课程号，成绩。其中用字符数组保存学生姓名，学号，课程号，课程名称，用布尔型保存学生性别和专业，用整数型保存学时数与成绩。链表结构包括学生信息，课程信息或成绩信息与下一节点指针。

1. **实现主程序部分：**

菜单栏显示，接受用户输入，实现相应功能。

1. **创建三个链表，实现文件输入和读出：**

链表内容由用户输入，分别创造学生信息、课程信息、学生成绩表三个链表，对其分别进行按照学号升序排列、按照课程号升序排列、按照学号和课程号升序排列，输出到屏幕，最后保存到文件中。

定义并实现：

1. void list\_create\_student(void);

2. void list\_create\_course(void);

3.void list\_create\_grade(void);

4.void list\_read\_student(void);

5.void list\_read\_course(void);

6.void list\_read\_grade(void);

7.void output\_student(void);

8.void output\_course(void);

9.void output\_grade(void);

1. **实现三个归并排序函数**

三个函数分别对应于上述三种排列方式。

定义并实现：

1.StudentList\* MergeSort\_student(StudentList\* node);

2.CourseList\* MergeSort\_course(CourseList\* node);

3.GradeList\* MergeSort\_grade(GradeList\* node);

1. **创建课程成绩单链表**

此链表由学生信息、课程信息、学生成绩三个表合成，在输出是可根据用户选择，进行对三个课程的分别排序，并且最终将其输入保存到文件中。

定义并实现：

1.void create\_stdgra\_table(void);

2.void save\_sg\_file(void);

3.void list\_read\_studentgrade(void);

4.void output\_studentgrade(void);

1. **实现查询课程号生成该课程的单链表的功能。**

该单链表基于课程成绩单链表，最后需要按该成绩降序排列。

定义并实现：

1.void printf\_course\_grade(void);

1. **实现查询制定课程号低于六十分的学生成绩信息**

定义并实现：

1.void printf\_flunk\_course(void);

1. **利用栈实现学生信息的逆序链表**

定义并实现：

1.void Stack(void);

2.void InitStack(void);

3.bool StackEmpty(void);

4.void Push(StudentList \*ps);

5.StudentList \* Pop(void);

6.void output\_stack(void);

1. **使用链式队列实现课程成绩链表的输出**

定义并实现：

1.void Queue(void);

2.void InitQueue(void);

3.bool QueueEmpty(void);

4.void enQueue(StudentgradeList \*q);

5.bool deQueue(StudentgradeList \*\*e);

**10. 反复调试修改，使得能正常实现所有功能。**

1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**
   1. 主程序的实现

|  |
| --- |
| int main(void)  {  int code;  while (1) {  printf("\n");  puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  puts("1. 输入学生信息并保存");  puts("2. 输入课程信息并保存");  puts("3. 输入成绩信息并保存");  puts("4. 读取学生信息，创建按学号升序的单向链表");  puts("5. 读取课程信息，创建按课程号升序的单向链表");  puts("6. 读取成绩信息，创建按学号和课程号升序的单向链表");  puts("7. 读取成绩信息，查询所有学生所有课程的考试成绩");  puts("8. 读取成绩信息，查询所有学生指定课程的考试成绩");  puts("9. 读取成绩信息，查询指定课程成绩小于 60 分的学生");  puts("10. 将 4 中生成的链表反转；");  puts("11. 用链式队列解决 7 的问题；");  puts("输入任意其他字符退出系统");  puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  scanf("%d", &code);  switch (code)  {  case 1:  list\_create\_student();  break;  case 2:  list\_create\_course();  break;  case 3:  list\_create\_grade();  break;  case 4:  list\_read\_student();  MergeSort\_student(std\_table.snode); output\_student();  break;  case 5:  list\_read\_course();  MergeSort\_course(cou\_table.cnode); output\_course();  break;  case 6:  list\_read\_grade();  MergeSort\_grade(gra\_table.gnode); output\_grade();  break;  case 7:  list\_read\_student();  MergeSort\_student(std\_table.snode);  list\_read\_course();  MergeSort\_course(cou\_table.cnode);  list\_read\_grade();  MergeSort\_grade(gra\_table.gnode);  create\_stdgra\_table();  save\_sg\_file();  list\_read\_studentgrade();  break;  case 8:  list\_read\_studentgrade();  printf\_course\_grade();  break;  case 9:  list\_read\_studentgrade();  printf\_flunk\_course();  break;  case 10:  Stack();  output\_stack();  break;  case 11:  list\_read\_studentgrade();  printf\_queue();  break;  default:  return 0;  }  }  return 0;  } |

* 1. 输入学生信息并保存

|  |
| --- |
| void list\_create\_student(void)  {  int i=1;  FILE \*fp\_student;  StudentList \*std;  std\_table.cols = 10;  std\_table.rows = 4;  std = (StudentList\*)malloc(sizeof(StudentList));  printf("输入第%d个学号", i);  scanf("%s", std->sno);//用户出入  printf("输入第%d个姓名", i);  scanf("%s", std->sname);  printf("输入第%d个性别", i);  scanf("%s", std->sex);  printf("输入第%d个专业", i);  scanf("%s", std->major);  std->next = NULL;  std\_table.snode = std;  fp\_student = fopen("student.dat", "w");  if (fp\_student == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  for (i = 2; i <= 10; i++)  {  std = (StudentList\*)malloc(sizeof(StudentList));  printf("输入第%d个学号", i);  scanf("%s", std->sno);//用户出入  printf("输入第%d个姓名", i);  scanf("%s", std->sname);  printf("输入第%d个性别", i);  scanf("%s", std->sex);  printf("输入第%d个专业", i);  scanf("%s", std->major);  std->next = std\_table.snode;  std\_table.snode = std;  }  while (std\_table.snode != NULL)  {  if (fwrite(std\_table.snode, sizeof(StudentList), 1, fp\_student) == 1)  std\_table.snode = std\_table.snode->next;  }  fclose(fp\_student);  } |

* 1. 输入课程信息并保存

|  |
| --- |
| void list\_create\_course(void) {    int i = 1;  FILE \*fp\_course;  CourseList \*course;  cou\_table.cols = 3;  cou\_table.rows = 3;  course = (CourseList\*)malloc(sizeof(CourseList));  printf("输入第%d个课程号", i);  scanf("%s", course->cno);//用户出入  printf("输入第%d个课程名", i);  scanf("%s", course->cname);  printf("输入第%d个课时", i);  scanf("%d", &course->classHours);  course->next = NULL;  cou\_table.cnode = course;  fp\_course = fopen("course.dat", "w");  if (fp\_course == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  for (i = 2; i <= 3; i++)  {  course = (CourseList\*)malloc(sizeof(CourseList));  printf("输入第%d个课程号", i);  scanf("%s", course->cno);//用户出入  printf("输入第%d个课程名", i);  scanf("%s", course->cname);  printf("输入第%d个课时", i);  scanf("%d", &course->classHours);  course->next = cou\_table.cnode;  cou\_table.cnode = course;  }  while (cou\_table.cnode != NULL)  {  if (fwrite(cou\_table.cnode, sizeof(CourseList), 1, fp\_course) == 1)  cou\_table.cnode = cou\_table.cnode->next;  }  fclose(fp\_course);  } |

* 1. 输入成绩信息并保存

|  |
| --- |
| void list\_create\_grade(void)  {  int i = 1;  FILE \*fp\_grade;  GradeList \*grade;  gra\_table.cols = 30;  gra\_table.rows = 3;  i = 1;  grade = (GradeList\*)malloc(sizeof(GradeList));  printf("输入第%d个学号", i);  scanf("%s", grade->sno);//用户出入  printf("输入第%d个课程号", i);  scanf("%s", grade->cno);  printf("输入第%d个成绩", i);  scanf("%d", &grade->score);  grade->next = NULL;  gra\_table.gnode = grade;  fp\_grade = fopen("courseGrade.dat", "w");  if (fp\_grade == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  for (i = 2; i <= 30; i++)  {  grade = (GradeList\*)malloc(sizeof(GradeList));  printf("输入第%d个学号", i);  scanf("%s", grade->sno);//用户出入  printf("输入第%d个课程号", i);  scanf("%s", grade->cno);  printf("输入第%d个成绩", i);  scanf("%d", &grade->score);  grade->next = gra\_table.gnode;  gra\_table.gnode = grade;  }  while (gra\_table.gnode != NULL)  {  if (fwrite(gra\_table.gnode, sizeof(GradeList), 1, fp\_grade) == 1)  gra\_table.gnode = gra\_table.gnode->next;  }  fclose(fp\_grade);  } |

* 1. 读取学生信息，创建按学号升序的单向链表

|  |
| --- |
| void list\_read\_student(void)  {  FILE \*fp\_student;  StudentList \* new\_std;  fp\_student = fopen("student.dat", "rb");  if (fp\_student == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  std\_table.snode = NULL;  for (;;)  {  new\_std= (StudentList\*)malloc(sizeof(StudentList));  if (new\_std == NULL)  {  perror("读入信息分配空间失败：");  exit(EXIT\_FAILURE);  }  if (fread(new\_std, sizeof(StudentList), 1, fp\_student) == 1)  {  if (std\_table.snode== NULL) {  new\_std->next = NULL;  std\_table.snode = new\_std;  }  else {    new\_std->next = std\_table.snode;  std\_table.snode = new\_std;  }      }  else  break;  }  fclose(fp\_student);    }  void output\_student(void)  {  StudentList \*p1;  for (p1 = std\_table.snode; p1 != NULL; p1 = p1->next)  {  printf("\n学号:%-10s姓名:%-10s性别:%-10s专业:%-10s", p1->sno, p1->sname, p1->sex, p1->major);  }  } |

* 1. 读取课程信息，创建按课程号升序的单向链表

|  |
| --- |
| void list\_read\_course(void)  {  FILE \*fp\_course;  CourseList \* new\_cou;  fp\_course = fopen("course.dat", "rb");  if (fp\_course == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  cou\_table.cnode = NULL;  for (;;)  {  new\_cou = (CourseList\*)malloc(sizeof(CourseList));  if (new\_cou == NULL)  {  perror("读入信息分配空间失败：");  exit(EXIT\_FAILURE);  }  if (fread(new\_cou, sizeof(CourseList), 1, fp\_course) == 1)  {  if (cou\_table.cnode == NULL) {  new\_cou->next = NULL;  cou\_table.cnode = new\_cou;  }  else {  new\_cou->next = cou\_table.cnode;  cou\_table.cnode = new\_cou;  }  }  else  break;  }  fclose(fp\_course);  } |

* 1. 读取成绩信息，创建按学号和课程号升序的单向链表

|  |
| --- |
| void list\_read\_grade(void)  {  FILE \*fp\_grade;  GradeList \* new\_gra;  fp\_grade = fopen("courseGrade.dat", "rb");  if (fp\_grade == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  gra\_table.gnode = NULL;  for (;;)  {  new\_gra = (GradeList\*)malloc(sizeof(GradeList));  if (new\_gra == NULL)  {  perror("读入信息分配空间失败：");  exit(EXIT\_FAILURE);  }  if (fread(new\_gra, sizeof(GradeList), 1, fp\_grade) == 1)  {  new\_gra->next = gra\_table.gnode;  gra\_table.gnode = new\_gra;  }  else  break;  }  fclose(fp\_grade);  } |

* 1. 归并排序函数实现

|  |
| --- |
| StudentList\* MergeSort\_student(StudentList\* node) {  //先判断链表长度是否大于1，小于1时无须排序  if (node != NULL && node->next != NULL) {  //运用快慢指针，找到链表的中间节点  StudentList \*fast = node->next;  StudentList \*slow = node;  while (fast != NULL && fast->next != NULL) {  fast = fast->next->next;  slow = slow->next;  }  //将链表分成两部分进行分割  StudentList \*p1 = MergeSort\_student(slow->next);  slow->next = NULL; //设置了第一个链表的结尾  StudentList \*p2 = MergeSort\_student(node);  //对两条子链进行归并  StudentList \*p0 = (StudentList \*)malloc(sizeof(StudentList));  StudentList \*p = p0;  while (p1 != NULL && p2 != NULL) {  if (strcmp(p1->sno, p2->sno) < 0) {  p->next = p1;  p1 = p1->next;  }  else {  p->next = p2;  p2 = p2->next;  }  p = p->next;  }  if (p1 != NULL) {  p->next = p1;  }  if (p2 != NULL) {  p->next = p2;  }  p = p0->next;  free(p0);  return p;  }  return node;  } |

* 1. 查询成绩课程表，创建所有学生所有课程的单向链表

|  |
| --- |
| void create\_stdgra\_table(void) //生成学生成绩表  {  char cno[10];  stdgra\_table.sgnode = NULL;  stdgra\_table.cols = 5;  stdgra\_table.rows = 10;  StudentList \*ps = std\_table.snode;  CourseList \*pc = cou\_table.cnode;  GradeList \*pg = gra\_table.gnode;  StudentgradeList\* new\_sg;  printf("\n按哪门课程排序成绩排序:\n数据库：输入C01\n数据结构：输入C02\n程序设计：输入C03\n");  scanf("%s", cno);    while (ps != NULL&&pg!=NULL)  {  new\_sg = (StudentgradeList\*)malloc(sizeof(StudentgradeList));  strcpy(new\_sg->sno, ps->sno);  strcpy(new\_sg->sname, ps->sname);  strcpy(new\_sg->major, ps->major);  //构建三个科目的成绩  strcpy(new\_sg->cname1, pc->cname);  pc = pc->next;  new\_sg->score1 = pg->score;  pg = pg->next;  strcpy(new\_sg->cname2, pc->cname);  pc = pc->next;  new\_sg->score2 = pg->score;  pg = pg->next;  strcpy(new\_sg->cname3, pc->cname);  pc = cou\_table.cnode; //初始化pc  new\_sg->score3 = pg->score;  pg = pg->next;    ps = ps->next;  //构建链表  new\_sg->next = stdgra\_table.sgnode;  stdgra\_table.sgnode = new\_sg;  if (strcmp(cno, "C01") == 0)  {  //按数据库成绩降序函数  stdgra\_table.sgnode=MergeSort\_studentgrade\_C01(stdgra\_table.sgnode);  }  else if (strcmp(cno, "C02") == 0)  {  //按数据结构成绩降序函数  stdgra\_table.sgnode = MergeSort\_studentgrade\_C02(stdgra\_table.sgnode);  }  else if (strcmp(cno, "C03") == 0)  {  //按程序设计成绩降序函数  stdgra\_table.sgnode = MergeSort\_studentgrade\_C03(stdgra\_table.sgnode);  }    }  output\_studentgrade();  }  void save\_sg\_file(void)//将学生成绩表存放到文件中  {  FILE \*fp\_sg;  fp\_sg = fopen("studentGrade.dat", "w");  if (fp\_sg == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  while (stdgra\_table.sgnode != NULL)  {  if (fwrite(stdgra\_table.sgnode, sizeof(StudentgradeList), 1, fp\_sg) == 1)  stdgra\_table.sgnode = stdgra\_table.sgnode->next;  }  fclose(fp\_sg);  }  void list\_read\_studentgrade(void) //从文件中读出  {  FILE \*fp\_sg;  StudentgradeList\* new\_sg;  fp\_sg = fopen("studentGrade.dat", "rb");  if (fp\_sg == NULL)  {  perror("读入信息文件打开失败：");  exit(EXIT\_FAILURE);  }  stdgra\_table.sgnode = NULL;  for (;;)  {  new\_sg = (StudentgradeList\*)malloc(sizeof(StudentgradeList));  if (new\_sg == NULL)  {  perror("读入信息分配空间失败：");  exit(EXIT\_FAILURE);  }  if (fread(new\_sg, sizeof(StudentgradeList), 1, fp\_sg) == 1)  {  new\_sg->next = stdgra\_table.sgnode;  stdgra\_table.sgnode = new\_sg;  }  else  break;  }  fclose(fp\_sg);  }  void output\_studentgrade(void)  {  StudentgradeList \*p;  for (p = stdgra\_table.sgnode; p != NULL; p = p->next)  {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname1, p->score1);  printf("\n%-10s:%-10d", p->cname2, p->score2);  printf("\n%-10s:%-10d", p->cname3, p->score3);  }  } |

* 1. 查询成绩课程表，输出对应课程的成绩表

|  |
| --- |
| void printf\_course\_grade(void) {  char cno[10];  StudentgradeList\* p;  printf("\n请输入查询课程号:\nC01\nC02\nC03\n");  scanf("%s", cno);  if (strcmp(cno, "C01") == 0)  {  //按CO1成绩降序函数  p=MergeSort\_studentgrade\_C01(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname1, p->score1);  }  }  else if (strcmp(cno, "C02") == 0)  {  //按CO2成绩降序函数  p=MergeSort\_studentgrade\_C02(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname2, p->score2);  }  }  else if (strcmp(cno, "C03") == 0)  {  //按CO3成绩降序函数  p=MergeSort\_studentgrade\_C03(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname3, p->score3);  }  }  else  {  printf("\nerror input");  exit(1);  }  }  StudentgradeList\* MergeSort\_studentgrade\_C01(StudentgradeList\* node) {  //先判断链表长度是否大于1，小于1时无须排序  if (node != NULL && node->next != NULL) {  //运用快慢指针，找到链表的中间节点  StudentgradeList \*fast = node->next;  StudentgradeList \*slow = node;  while (fast != NULL && fast->next != NULL) {  fast = fast->next->next;  slow = slow->next;  }  //将链表分成两部分进行分割  StudentgradeList \*p1 = MergeSort\_studentgrade\_C01(slow->next);  slow->next = NULL;  StudentgradeList \*p2 = MergeSort\_studentgrade\_C01(node);  //对两条子链进行归并  StudentgradeList \*p0 = (StudentgradeList \*)malloc(sizeof(StudentgradeList));  StudentgradeList \*p = p0; //p0作为头指针，p用于移动处理。  while (p1 != NULL && p2 != NULL) {  if (p1->score1>p2->score1) {  p->next = p1;  p1 = p1->next;  }  else {  p->next = p2;  p2 = p2->next;  }  p = p->next;  }  if (p1 != NULL) {  p->next = p1;  }  if (p2 != NULL) {  p->next = p2;  }  p = p0->next; //p指向头指针后第一个位置，即第一个存放数据的位置。  free(p0);  return p;  }  return node;  }  StudentgradeList\* MergeSort\_studentgrade\_C02(StudentgradeList\* node) {  //先判断链表长度是否大于1，小于1时无须排序  if (node != NULL && node->next != NULL) {  //运用快慢指针，找到链表的中间节点  StudentgradeList \*fast = node->next;  StudentgradeList \*slow = node;  while (fast != NULL && fast->next != NULL) {  fast = fast->next->next;  slow = slow->next;  }  //将链表分成两部分进行分割  StudentgradeList \*p1 = MergeSort\_studentgrade\_C02(slow->next);  slow->next = NULL;  StudentgradeList \*p2 = MergeSort\_studentgrade\_C02(node);  //对两条子链进行归并  StudentgradeList \*p0 = (StudentgradeList \*)malloc(sizeof(StudentgradeList));  StudentgradeList \*p = p0; //p0作为头指针，p用于移动处理。  while (p1 != NULL && p2 != NULL) {  if (p1->score2 > p2->score2) {  p->next = p1;  p1 = p1->next;  }  else {  p->next = p2;  p2 = p2->next;  }  p = p->next;  }  if (p1 != NULL) {  p->next = p1;  }  if (p2 != NULL) {  p->next = p2;  }  p = p0->next; //p指向头指针后第一个位置，即第一个存放数据的位置。  free(p0);  return p;  }  return node;  }  StudentgradeList\* MergeSort\_studentgrade\_C03(StudentgradeList\* node) {  //先判断链表长度是否大于1，小于1时无须排序  if (node != NULL && node->next != NULL) {  //运用快慢指针，找到链表的中间节点  StudentgradeList \*fast = node->next;  StudentgradeList \*slow = node;  while (fast != NULL && fast->next != NULL) {  fast = fast->next->next;  slow = slow->next;  }  //将链表分成两部分进行分割  StudentgradeList \*p1 = MergeSort\_studentgrade\_C03(slow->next);  slow->next = NULL;  StudentgradeList \*p2 = MergeSort\_studentgrade\_C03(node);  //对两条子链进行归并  StudentgradeList \*p0 = (StudentgradeList \*)malloc(sizeof(StudentgradeList));  StudentgradeList \*p = p0; //p0作为头指针，p用于移动处理。  while (p1 != NULL && p2 != NULL) {  if (p1->score3 > p2->score3) {  p->next = p1;  p1 = p1->next;  }  else {  p->next = p2;  p2 = p2->next;  }  p = p->next;  }  if (p1 != NULL) {  p->next = p1;  }  if (p2 != NULL) {  p->next = p2;  }  p = p0->next; //p指向头指针后第一个位置，即第一个存放数据的位置。  free(p0);  return p;  }  return node;  } |

* 1. 查询对应课程成绩低于六十的学生，并排序

|  |
| --- |
| void printf\_flunk\_course(void) {  char cno[10];  StudentgradeList\* p;  printf("\n请输入查询课程号:\nC01\nC02\nC03\n");  scanf("%s", cno);  if (strcmp(cno, "C01") == 0)  {  //按CO1成绩降序函数  p = MergeSort\_studentgrade\_C01(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  if (p->score1 < 60) {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname1, p->score1);  }    }  }  else if (strcmp(cno, "C02") == 0)  {  //按CO2成绩降序函数  p = MergeSort\_studentgrade\_C02(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  if (p->score2 < 60) {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname2, p->score2);  }  }  }  else if (strcmp(cno, "C03") == 0)  {  //按CO3成绩降序函数  p = MergeSort\_studentgrade\_C03(stdgra\_table.sgnode);  for (; p != NULL; p = p->next)  {  if (p->score3 < 60) {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", p->sno, p->sname, p->major);  printf("\n%-10s:%-10d", p->cname3, p->score3);  }  }  }  else  {  printf("\nerror input");  exit(1);  }  } |

* 1. 利用栈倒序学生信息表

|  |
| --- |
| void Stack(void) {  InitStack();  StudentList \*ps = std\_table.snode;  StudentList \*p;  while (ps != NULL)  {  Push(ps);  ps = ps->next;  }  new\_std\_table.snode = Pop(); //第一个出栈  p = new\_std\_table.snode;  while ((p->next = Pop()) != NULL)  {  p = p->next;  }  p->next = NULL;      }  void InitStack(void) {  stack\_list= (LinkStNode\*)malloc(sizeof(LinkStNode));  stack\_list->next = NULL;  }  bool StackEmpty(void) {  return(stack\_list->next == NULL);  }  void Push(StudentList \*ps) {  LinkStNode \*p;  p = (LinkStNode\*)malloc(sizeof(LinkStNode));  strcpy(p->sno, ps->sno);  strcpy(p->sname, ps->sname);  strcpy(p->sex, ps->sex);  strcpy(p->major, ps->major);  p->next = stack\_list->next;  stack\_list -> next = p;  }  StudentList \* Pop(void) {  LinkStNode \*p;  StudentList \* temp=NULL;  temp = (StudentList \*)malloc(sizeof(StudentList));  if (StackEmpty())  return NULL;  p = stack\_list->next;  strcpy(temp->sno, p->sno);  strcpy(temp->sname, p->sname);  strcpy(temp->sex, p->sex);  strcpy(temp->major, p->major);  stack\_list->next = p->next;  return temp;  }  void output\_stack(void)  {  StudentList \*p;  for (p = new\_std\_table.snode; p != NULL; p = p->next)  {  printf("\n学号:%-10s姓名:%-10s性别:%-10s专业:%-10s", p->sno, p->sname, p->sex, p->major);  }  } |

* 1. 使用队列实现课程成绩表的顺序输出

|  |
| --- |
| void Queue(StudentgradeList \*Q)  {  StudentgradeList \*pq =Q;  StudentgradeList \*item;  InitQueue();  while (pq != NULL)  {  enQueue(pq);  pq = pq->next;  }  while (deQueue(&item))  {  printf("\n学号:%-10s\n姓名:%-10s\n专业:%-10s", item->sno, item->sname, item->major);  printf("\n%-10s:%-10d", item->cname1, item->score1);  printf("\n%-10s:%-10d", item->cname2, item->score2);  printf("\n%-10s:%-10d", item->cname3, item->score3);  }  return;  }  void InitQueue(void)  {  queue\_list = (LinkQuNode\*)malloc(sizeof(LinkQuNode));  queue\_list->front = queue\_list->rear = NULL;  }  bool QueueEmpty(void)  {  return queue\_list->rear == NULL;  }  void enQueue(StudentgradeList \*q)  {  StudentgradeList \*p;  p = (StudentgradeList\*)malloc(sizeof(StudentgradeList));  strcpy(p->sno, q->sno);  strcpy(p->sname, q->sname);  strcpy(p->major, q->major);  strcpy(p->cname1, q->cname1);  p->score1 = q->score1;  strcpy(p->cname2, q->cname2);  p->score2 = q->score2;  strcpy(p->cname3, q->cname3);  p->score3 = q->score3;  p->next = NULL;  if (QueueEmpty())  queue\_list->front = queue\_list->rear = p;  else  {  queue\_list->rear->next = p;  queue\_list->rear = p;  }  }  bool deQueue(StudentgradeList \*\*e)  {  StudentgradeList \*t;  if (QueueEmpty())  return false;  t = queue\_list->front;  if (queue\_list->front == queue\_list->rear)  queue\_list->front = queue\_list->rear = NULL;  else  queue\_list->front = queue\_list->front->next;  \*e = t;  return true;  }  void printf\_queue(void) {  char cno[10];  StudentgradeList\* p;  printf("\n请输入查询课程号:\nC01\nC02\nC03\n");  scanf("%s", cno);  if (strcmp(cno, "C01") == 0)  {  //按CO1成绩降序函数  p = MergeSort\_studentgrade\_C01(stdgra\_table.sgnode);  Queue(p);  }  else if (strcmp(cno, "C02") == 0)  {  //按CO2成绩降序函数  p = MergeSort\_studentgrade\_C02(stdgra\_table.sgnode);  Queue(p);  }  else if (strcmp(cno, "C03") == 0)  {  //按CO3成绩降序函数  p = MergeSort\_studentgrade\_C03(stdgra\_table.sgnode);  Queue(p);  }  else  {  printf("\nerror input");  exit(1);  }  } |

1. **总结及心得体会：**
2. 编写代码过程中应当尽量避免小错误的发生，往往一个小小的错误可能需要debug半天，以下是容易碰到的错误点：其一，赋值与判等符号容易搞错，或者说容易忘记加一个等号；其二，特殊情况需要额外考虑，如指针为空，这一点在插入一个信息是额外需要注意；其三，拼写错误，比如printf写成pirntf，打字快了就可能出错，然后中英文的逗号分号也要注意；其四，指针的重数，函数的调用，全局变量的值都应该小心谨慎。
3. 各种括号、语句的缩进应当注意，否则会显得很乱，过一段时间就看不懂了，故而在程序编写过程中应当注意缩进风格的异同。
4. 注释的使用、变量名、函数名的选择都很重要。这既决定了你的代码能否被别人所理解，也决定了你写代码的速度，拥有显然直观的名字可以在很大程度上减少代码的正确性。
5. 应当重视并熟练掌握指针的运用，指针感觉就像是C语言的灵魂，所有的操作大多都基于指针，没有指针寸步难行。指针的存在对各种功能的实现和简化有不可或缺的作用。
6. **对本实验过程及方法、手段的改进建议：**

本实验重复内容过多，建议精简实验过程，可以避免很多时间资源浪费。

**报告评分：**

**指导教师签字：**