基于单向链表类模板的球员管理系统

2021 年 10 月 11 日

1 基于单向链表类模板的球员管理系统项目概述

1.1 项目内容及要求

各小组自行选题，利用 LinkList 类模板设计一个链表综合应用程序。要求使用**链表的链表**。

1.2 研究人员及分工

序号	学号	姓名	角色及具体贡献
1	20123101	李昀哲	组长，项目设计 Lead-Designer； 贡献：项目设计、类设计、主函数设计编写；程序优化；报告撰写 初期协同研究学习 LinkList 类模板及 Bank、AddressBook 应用示例并开展讨论。
2	20123103	倪思远	类设计 Co-Designer 贡献：协助设计 FB_player 类，初期协同研究学习基于类模板的应用并提出 FB_player 类设计建议
3	20123105	杜佳杰	类设计 Co-Designer 贡献：协助设计 CR 类，初期协同研究学习基于类模板的应用并提出 CR 类设计建议
4	20122283	徐思怡	协同测试 Co-Tester 贡献：提出可能存在的问题，初期协同研究学习基于类模板的应用并提出测试样例设计建议

2 FB_player（球员 Football Player）类的设计

2.1 数据成员设计

```
1.  private:
2.      string  name, club;          //string 类的球员姓名、俱乐部
3.      int id;                      //创建球员分配的序号
4.      LinkList<CR> link;           //“球员”链表的“职业生涯报告”链表
```

2.2 成员函数（友元函数）原型设计

2.2.1 成员函数

2.2.1.1 构造函数

```
1.  //构造函数
2.  FB_player::FB_player(const char* Name, const char* Club, const int Id):name(Name), club(Club), id(Id)
3.  {
4.  }
```

2.2.1.2 获取球员姓名

```
1.  string FB_player::Get_name() //获取球员姓名
2.  {
3.      return name;
4.  }
```

2.2.1.3 设置球员信息

```
1.  void FB_player::Set(const char* Name, const char* Club, const int Id)   //通过外部信息设置球员的姓名、俱乐部、id
2.  {
3.      name = Name;
4.      club = Club;
5.      id = Id;
6.  }
```

2.2.1.4 转会函数

```
1.  void FB_player::Change_Club(const char * Name, const char* Club)       //转会函数
2.  {
3.      name = Name;
4.      club = Club;
5.  }
```

2.2.1.5 球员退役函数

```
1.  void FB_player::Set_Retirment(const char* Name)     //球员退役函数
2.  {
3.      name = Name;
4.      club = "Retirement";
5.      id = 0;
6.  }
```

2.2.1.6 重载强制转换运算符

```
1.  FB_player::operator string() const      //重载 string()运算符，用于显示一个俱乐部全部球员信息
2.  {
3.      return club;
4.  }
```

```
FB_player::operator int() const         //重载 int()运算符
{
    return id;
}
```

2.2.1.7 “球员链表”节点插入“生涯报告”链表

```
void FB_player::AppendCR(const CR& b)   //在“球员”链表中的节点加入“生涯报告链表”
{
    int x;
    double r;
    CR y(b);
    if (link.NumNodes() == 0)
    {
        x = 0;
    }
    else
    {
        link.GoBottom();
        x = link.CurData().total_goals;
    }

    y.total_goals = x + b.goal;
    y.Bonus(y.salary);
    link.Append(y);
}
```

2.2.1.8 显示“球员”链表信息函数

```
void FB_player::Show(ostream& out) const    //显示“球员”链表函数
{
    out << "Name: " << name << '\t' << "Club: " << club << '\t' <<"ID: " << id << endl;
}
```

2.2.2 友元函数

2.2.2.1 输入流运算符的重载

```
istream& operator>>(istream& in, FB_player& a)
    {
        char str[80];
        in >> str;    a.name = str;
        in >> str ;   a.club = str;
        return in;
    }
```

2.2.2.2 输出流运算符的重载

```
ostream& operator<<(ostream& out, const FB_player& u)   //显示信息函数
{

    out << "Name: "<< u.name << '\t' << "Club: " << u.club << '\t' << "ID: " << u.id << endl;
    cout << endl;
    out << "    Date\tGoals\tTotal\tSalary" << endl;
    u.link.PutList(out);
    return out;
}
```

3 CR（球员生涯报告 CareerReport）类的设计

3.1 数据成员设计

```
private:
    int year, month, day;   //进球年、月、日
    int goal, total_goals;  //当日进球、总进球
    double salary;          //薪资（会随进球而浮动）
```

3.2 成员函数（友元函数）原型设计

3.2.1 成员函数

3.2.1.1 构造函数

```
CR::CR(int Year, int Month, int Day, double Goal, double Salary) : year(Year), month(Month), day(Day), goal(Goal), total_goals(0), salary(5000)
```

```
{
}
```

3.2.1.2 设置进球

```
void CR::Set(int Year, int Month, int Day, int Goal)
{
    year = Year;
    month = Month;
    day = Day;
    goal = Goal;
}
```

3.2.1.3 重载强制类型转换运算符

```
CR::operator double() const
{
    return salary;
}
```

3.2.1.4 奖金奖励机制

```
void CR::Bonus(const double &Salary)
{
    salary = Salary *(1 + goal * 0.005);
}
```

3.2.2 友元函数

重载输出流运算符

```
ostream& operator<<(ostream& out, const CR& b)
{
    out << setfill('0');
    out << b.year << '-' << setw(2) << b.month << '-' << b.day << '\t';
    if (b.goal < 0)
        out << '\t' << -b.goal;
    else
        out << b.goal << '\t';
    out << b.total_goals << '\t' << b.salary <<setfill(' ');
    return out;
}
```

4 测试情况

4.1 测试样例设计

3.1.1 基本功能测试

序号	功能	用途	测试样例
1	增加一个球员	添加球员信息	姓名、俱乐部输入为英文或中文
2	输出全部信息	显示所有球员信息和进球信息	/
3	查询信息	共有三个功能模块：	
		① 查看某俱乐部全部球员信息	/
		② 选择一个球员增加进球信息	输入某日期和进球个数查看总进球和薪资的变化
		③ 退役一个球员	操作后查看球员信息是否退役
4	转会	更改一个球员的俱乐部信息	查看俱乐部信息是否更迭
5	删除球员	删除一个球员的全部信息	查看球员是否删除

3.1.2 可靠性测试

序号	功能	测试样例	测试结果
1	增加一个球员	姓名、俱乐部输入为数字，查看情况	“输入错误，输入必须为人名，请重新输入。” “输入错误，输入必须为俱乐部名，请重新输入。”
2	输出全部信息	/	/
3	查看某俱乐部全部球员信息	输入一个不存在的俱乐部，查看指针是否会越界	不会，定位函数中会对未找到球员的情况做区别，并输出“找到 0 个球员”
4	选择一个球员增加进球信息	①选择输入一个不存在的球员 ID，查看对指针越界的处理 ②日期输入不合法日期 ③进球输入为负值或零	①初次测试未考虑，导致指针越界。 修改后会对不存在的 ID 予以排除。 ②增加判定合法日期功能 ③增加判定合法进球功能
5	退役一个球员	选择输入一个不存在的球员 ID，查看对指针越界的处理	同样在初次测试未考虑，修改后修复
6	更改一个球员的俱乐部信息	俱乐部输入为数字，查看情况	“输入错误，输入必须为俱乐部名，请重新输入”
7	删除球员	输入不存在的 ID	同样在初次测试未考虑，修改后修复

3.2 测试结果对程序的改进情况

3.2.1 指针越界问题和逐语句运行的使用

初始版本的最大问题为对 Locate 后的数据指针越界的报错。但在初期测试并未发现是指针越界，而是去 debug 了类中函数的可能错误，但经过逐语句运行调试，定位到了在 Curdata()位置处 throw -1 时出现的问题，才发觉是指针越界问题，再着手修改代码，进行二次测试。

3.2.2 对同名同姓球员的处理和区别

之前未考虑到同名同姓球员，他们可能效力于不同俱乐部，而区分他们的最好方式，就是加入 ID。最初 FB_player 类的数据成员设计中，没有 ID，在测试后，因为创建球员总是会有先后的，现实中，即使是再多胞胎，也是先后出生的，根据这个逻辑，ID 就是类似“时间戳”的存在，以此区别球员。

3.2.3 对非法输入判定的优化预期

现在版本对非法的日期、ID、进球等都有了不同的处理，但都是在主函数中处理，未来可以加入成员函数，在类中直接处理，更好达到面向对象的效果。