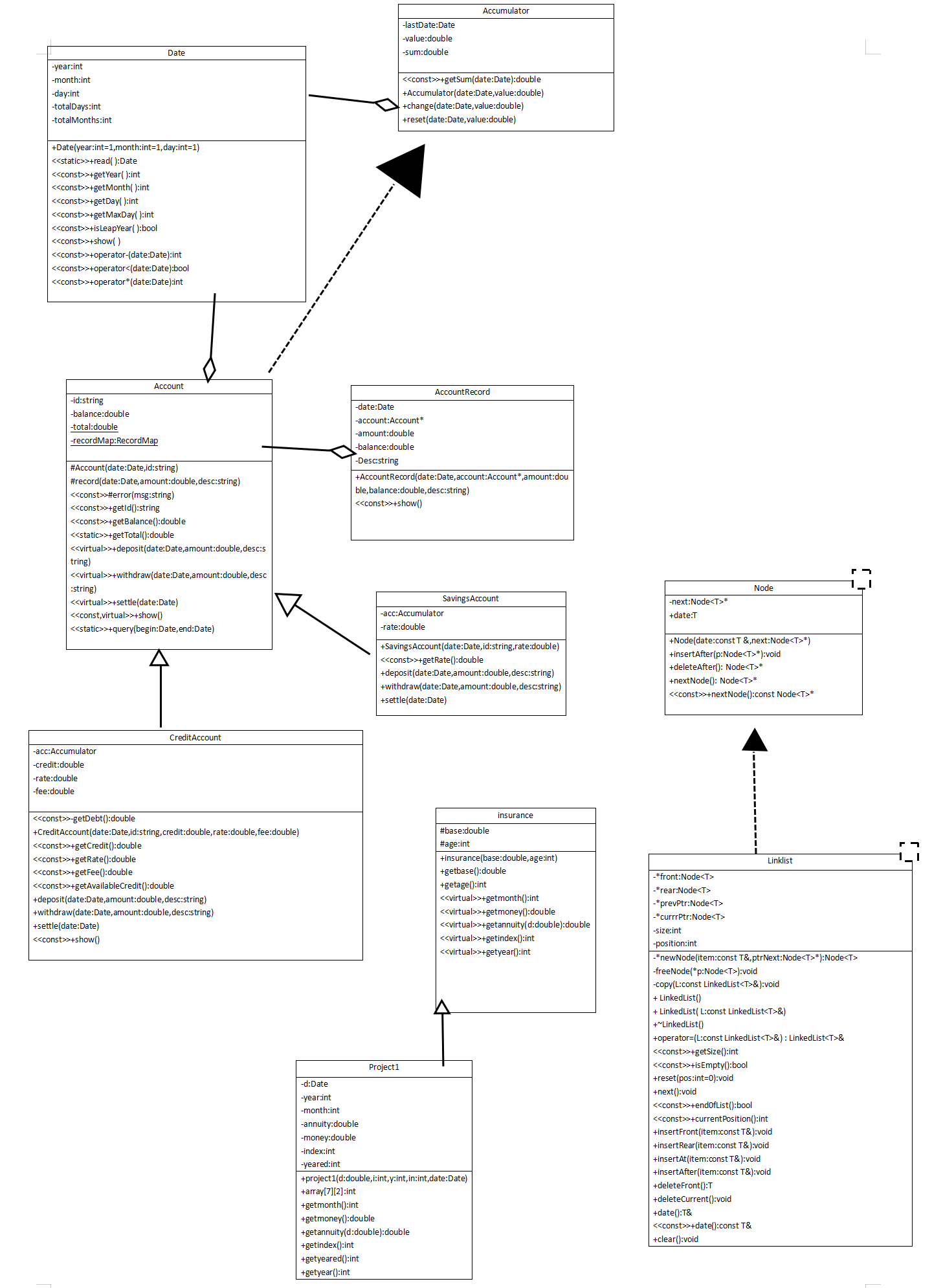
2.设计说明书

**2.1类设计**

2.1.1类图



2.1.2关于保险类的注释：

在insurance类中，

数据成员：

int 型数据age 用于存储用户预计退休的年龄；

double型数据base用于存储用户上一年度的年工资；

函数成员：

getbase函数用于获取用户上一年度年工资的值；

getage函数用于获取用户的预计退休年龄；

虚函数getmonth用于获取用户养老金的计发月份；

虚函数getmoney用于获取每月应缴纳养老保险的金额数；

虚函数getannuity用于获取退休之后每月应获得的养老金；

虚函数getindex用于获取用户的卡号；

虚函数getyear用于获取用户预计缴纳养老保险的年数；

在project1类中：

数据成员：

二维数组array用于存储国家标准中用户退休年龄与养老金计发月份的对应数据；

int型数据 month用于存储计发月数；

Int型数据year用于存储预计缴纳养老保险年数；

Double型数据annuity用于存储用户退休后的每月养老金；

Double型数据money用于存储用户每月应缴纳的养老保险数额；

int型数据index用于存储所用卡号；

Int型数据yeared用于记录用户目前的年龄；

函数成员：

getmonth重构insurance中的同名虚函数，用于获取用户养老金的计发月份并代入计算；getmoney重构insurance中的同名虚函数，用于获取每月应缴纳养老保险的金额数；

getannuity重构insurance中的同名虚函数，用于获取退休之后每月应获得的养老金；

getindex重构insurance中的同名虚函数，用于获取用户的卡号，所有缴纳操作将在此账户中进行；

getyear重构insurance中的同名虚函数，用于获取用户预计缴纳养老保险的年数；

getyeared用于获取用户目前开始购买养老保险的年龄；

**2.2主要技术难点及实现方案/算法设计**

**（1）主要技术难点：**

1.统计计发月数。

2.输入数据多时太过繁琐，消耗用户时间。

3.在养老保险中实现自动扣费与自动发放服务。

4.在主函数中的时间发生改变时，自动对账户进行扣钱或者存钱操作。且用户选择很丰富，可以跳转到下一个月，也可以跳转到下一年。但是用户是选择一个月一个月的跳转还是一年一年的跳转或者是两者混合使用，我们是无法预知的，这给计算扣钱总数、账户总金额等带来了极大的困难。

5.如果用户要购买多份保险产品，且不知道要买多少份，那么就需要一个动态数组来保存这些保险类型的数据。而且由于保险有许多不同的种类，所以需要充分利用多态的性质简化程序。

6.链表类模板的实例化。虽然模板是现成的，但是在教材及参考书上没有类似实例可供参考的情况下来使用，仍然是一件比较困难的事，常常不知道自己使用时的格式或者其他一些细节地方有什么错误。

**（2）实现方案：**

1.使用二维数组记录数据，第一维用来记录退休年龄，第二维用来记录相应的养老 金的计发月份，当按照要求输入退休年龄时，用循环找到该年龄在数组中的位置， 从这个年龄的位置来确定相应的养老金的计发月份的位置，从而找到了对应的计发月份。

2.采用循环来多次输入数据，同时，以五年作为分段点，第五年之前的数据，按真 实值精确输入，第五年之后的数据，以第五年的数据为养老保险缴费基数。即若打算缴 纳养老保险的年限不大于五年，可得到精确的数据，若打算缴纳养老保险的年限大于五 年，第五年之后缴纳费用按照第五年计算，退休之后的养老金发放的计算公式为：

获得每月养老金=预计退休后每个月的市平均月薪\*1%+

预计缴纳的养老保险总金额与每月获得的养老保险金是对应的，故没有问题。

3.记录年份，当年份处于需要缴纳养老保险的日期中时，使用account类的外部接口中的withdrew函数每过一个月自动扣除养老保险的金额，当缴纳日期满时，停止缴纳。同时可以调用函数查看每年的缴纳情况。当用户年龄达到可以退休年龄之后，使用deposit函数实现自动发放功能，每个月的月初向用户的账号中发放用户的养老金。

4.经过修改和改进后，我们小组的主函数中能够改变时间的操作有3种：在这个月里改变天数、跳转到下一个月、跳转到下一年。由于保险是按月扣钱或发钱的，所以我们关心的是后面两项。

对于账户利息的计算，我们在“跳转至下一年”的操作里模仿使用了“跳转至下一月”的实现方案，即将目前时间传入到settle(Date)函数里进行结算。

对于保险扣钱或存钱的计算，我们设置了一个静态变量lastmonth，用来保存上次进行时间操作时的月份。比如说我在2009年3月1日购买了该保险，跳转到下一月：2009年4月1日，那么我的lastmonth就是3。再使用当前月份减去lastmonth就能得出我应该扣的钱是(date.getmonth()-lastmonth)\*(每月扣的钱)。然后将lastmonth改成4。再比如我使用“跳转到下一年”的操作，又从2009年4月1日跳转到2010年1月1日。那么我度过的月份就是12-lastmonth+1。计算扣钱再将lastmonth改成1。对于存钱的操作类似。

5.虽然我们的程序只对保险类派生出了一个产品类，但实际生活中却有很多种保险，一个人可能买很多种不同的保险。

所以我们利用了vector模板创建了基类指针类型的动态数组insurances.push\_back(ins);，这样就能实现动态数据结构的管理，并利用模板内的函数实现得到数组大小、增添数据等功能。

另外，我们将基类insurance设置成了抽象类，并将其成员函数设置成虚函数，而且vector数组中的每个元素都是一个指向能保存派生类数据的动态内存空间的指针ins = new project1(base, age,year,index,date);，这样就能实现不同派生类对函数调用时的动态绑定，实现多态性。

向数组中添加元素的操作我们参照了书上原有代码中对account类的操作。访问具体对象的成员，可以通过（insurances[i]->类成员）的方式实现。最后调用deleter()函数释放内存空间。

6.先自学链表类的相关知识并做几道测试题练练手。通过public接口了解这个类模板具有哪些功能，引入该类，实现对其的实例化和调用。在屡次出现bug时仔细检查调用的数据成员是否是公共的，函数成员函数的参数传入格式是否正确。如果仍然不能达到预期效果，就应该仔细研读类具体是怎样实现法，去看看函数的具体定义。如果还是不行，我想与同学交流讨论或者向老师求助都是不错的选择。