

西安电子科技大学计算机科学与技术学院

1. **实验目的**

1、完成Date类并用“1970年1月1日之后的天数”实现它。

2、定义类Histogram。

3、定义类Expr进行简单表达式操作。

1. **实验环境**

IDE：clion cmake mingw

os：windows

硬件：Dell g15 5511笔记本

1. **实验内容**
2. 通过对Date类输入一个年月日格式的日期，返回1970年1月1日到这一天的天数。
3. 由输入的一组参数定义一个直方图，再由一组参数作为直方图的数据，构造直方图并打印。处理越界。
4. 输入一个无括号四则运算表达式，要求eval（）返回表达式的值，print打印表达式的完全括号形式、后缀形式、汇编形式。
5. **实验步骤**

1、

1. 构造函数输入一个年月日，并立即调用计算函数，将算得的天数存储在类成员变量numD中，输出则直接调用这个天数进行打印。
2. 天数计算方式：

先遍历年份，numD加对应天数（闰年判断调用另外的类成员函数），直到1970年。再遍历每个月、将对应天数加到类成员变量里面，直到1月1号。再处理多余天数。

1. 一个函数可以返回参数表示的月份的最大天数。
2. 一个函数可以通过输入的年月日判断是否是闰年。

剩余算法无需赘述。

2、

1. 定义map<int,int>boundsMap，左边存直方图分界点，右边存分界点右方、下一个分界点左方的数据的数量。若只有一个分界点，则throw错误。
2. 保存了smallestBound和biggestBound（分界点的最大最小值），以处理边缘情况（包括越界）。另外，biggestBound不会存入boundsMap（因为只存左分界值）。
3. add（）输入单个数据，通过map的upper\_bound函数快速获得右分界点，再通过自减得到左分界点。然后在boundsMap里查找到该分界点，令second值自增，完成统计。若数据越界，则throw错误，但不中断输入与计算。
4. addAll通过遍历输入整个list里的数据。
5. 对boundsMap遍历进行画图。另有一些美化用的逻辑控制，都不再赘述。

3、

1. 此题的核心就是把表达式表示成二叉树。非叶子结点存储运算符号；叶子结点存储数字。优先级更低运算符的作为父结点。
2. 定义了内部类：

class TNode{

public:

string str;

TNode\* lchild;

TNode\* rchild;

TNode(string nextFormula);

};

作为树结点的数据结构。在str中使用string形式存储运算符或者数字。构造函数通过输入的子表达式来构建树。（如果用栈很难处理print等）

1. 建树过程：

（一言以蔽之：以运算符分割表达式）

调用TNode构造函数，传入“以该结点为根的子树”对应的子表达式。

**从后往前**遍历该表达式以寻找分割点。

（注：1+2-3则割‘-’；1\*2/3则割‘/’）

1. 如果是纯数字，则左右子树置空，str内存入传入的string。
2. 如果第一个非数字找到的是‘+’或‘-’，则以此为分割点，str存入该运算符（转string存）。
3. 如果第一个非数字找到的是‘\*’或‘/’，则接着遍历，如果找到‘+’或‘-’则把分割点移走，否则把分割点放在原位。然后同情况b进行操作。

如果不是纯数字（即情况b、c），则lchild=new TNode（分割点左侧子串），rchild=new TNode（分割点右侧子串）。

那么，类成员变量里面存根结点TNode\* root，调用root=new TNode（原始表达式）；就可以自动建整个树出来。

1. eval计算过程：

使用int getAns（TNode）进行递归。返回值是对应子树的对应表达式的计算结果值。

递归逻辑：

如果是叶子结点，那么本结点是数字，直接返回str内的字符串转换出的int。否则：

getAns（lchild）计算左右子树的值，并根据自己str内所存的符号来对这两个值进行加减乘除，然后返回计算后的值。

于是getAns（root）即可得到所求的值。

1. print（）过程：

类成员变量里面保存了各个表达式形式的string。

表达式的中间求得形式有**两种保存方法**：

1. 直接加在对应的类成员变量string上（函数返回值可为void）。

b、作为递归的返回值抛回，每一层递归都把字符串收集起来加起来， 直到最初的函数入口得到结果，再给类成员变量赋值（函数返回值必 须为string）。

其实三种形式的算法都可以使用a方法，因为这几个树的递归调用不需要对返回值进行额外的运算或是多次输出，过程也独立，因此无论是直接输出还是返回给上一级递归都会得到同样的结果。

我在全括号、后缀形式的算法里使用了b方法递归，因为它直观好看。

但在汇编形式里我不得不使用a方法，因为汇编里一个树的结果要保存在寄存器里面，我必须让函数返回结果寄存器的名字而不是用来递归，否则会极度麻烦。

-全括号形式递归逻辑：（中序）（b方法）

若是叶子结点，则返回str。否则：

返回‘(’+左子树的全括号表达式+str+右子树的全括号表达式+‘)’。

于是对root递归即可得到所求表达式形式。

-后缀形式递归逻辑：（后序）（b方法）

若是叶子结点，则返回str。否则：

返回左子树后缀表达式+右子树后缀表达式+str。

于是对root递归即可得到所求表达式形式。

-汇编形式：（后序）（a方法）

递归函数返回所在子树的答案所存储的寄存器的名字。

递归函数有个引用型参数map<char,bool> &regMap，用来保存寄存器名字与使用状态。true表示闲置，false表示占用。初值为{{'A',true},{'B',true},{'C',true},{'D',true}}。（注：验收时我并未进行寄存器回收利用，无限使用新寄存器，在写报告前更新了功能）

由于这题的表达式没有括号，因此寄存器最多只要三个，寄存器D其实是不需要的。例如1+2\*3即可达到最大寄存器量，分别保存1、2、3然后下一步进行2\*3。即使是1\*2+3\*4\*5依旧只要三个寄存器。如果题目改成有括号的表达式，寄存器可能需求量将达到无限多，需要在不足时调用新建寄存器函数来扩展regMap。

接下来讲解其递归逻辑：

如果是叶子结点，则是要进行MOV操作，先查找闲置寄存器，保存其名字在rig中，并将该寄存器设为占用（false）；然后compilationForm+=MOV+reg+,+str+\n；并return reg;否则：

进行左子树的汇编操作，得到存左子树结果的寄存器名字regL；进行右子树的汇编操作，得到存右子树结果的寄存器名字regR。然后根据str内的运算符对这俩寄存器进行ADD、SUB、MUL、DIV，compilationForm+=运算符号+regL+,+regR+\n；之后将regR设为闲置（true），并return regL;（因为汇编每一步结果都存在左边的寄存器里，右边的就不需要了）。

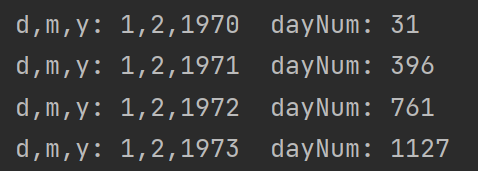
于是对root递归即可得到所求表达式形式。（而且最终结果都会存在AX里）

然后print对三种形式调用三个递归入口函数，最后输出类成员变量即可。

1. **实验结果**

针对不同输入，该程序都存在哪几类可能出现的情况，你的测试数据要完全覆盖了你所想到的这些情况，并给出测试结果截图及说明。

1、



间隔刚好是365、365、366，无问题。

2、

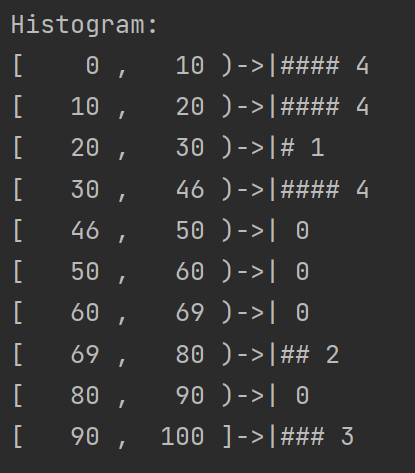
直方图分界点：

set<int> inBounds = {0, 10, 20, 30, 46, 50, 60, 69, 80, 90, 100};

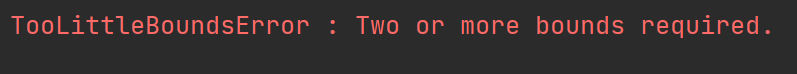
数据：

list<int> valueList = {0, 100, 96, 96,2, 1, 15, 16, 37, 10, 24, 35, 32, 12, 5, 78, 45, 78};

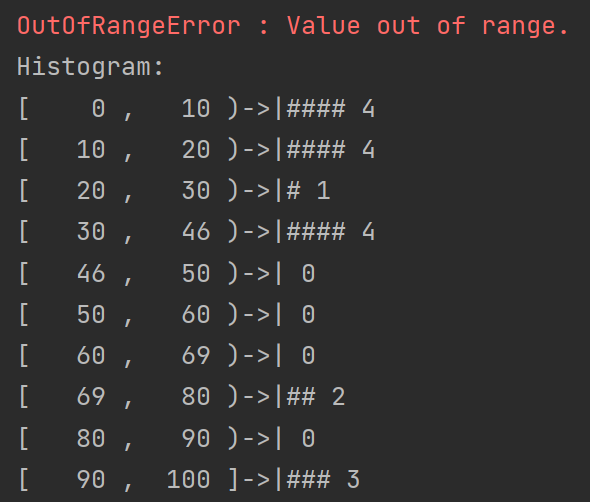
结果：



若inBounds={0}（只有一个分界点）则：

报错并直接跳过接下来的add。

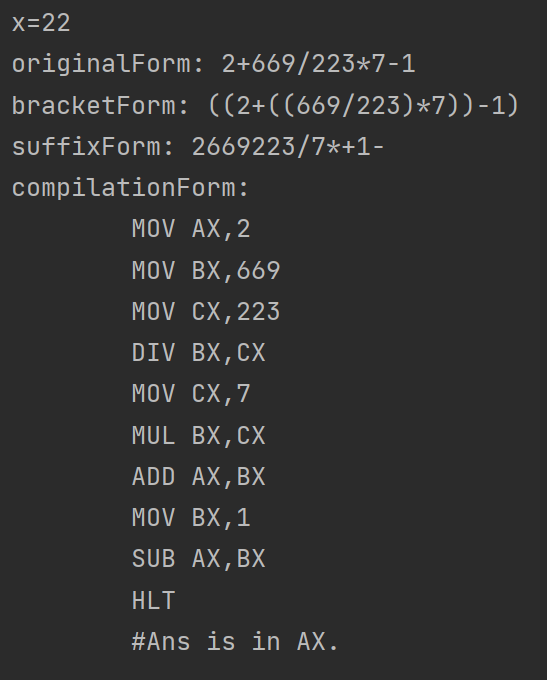
若给valueList里塞个-1（越界值），则：



报错但不中断程序。

3、

测试表达式为2+669/223\*7-1，结果：



eval计算结果没问题。运算顺序正确，三种形式的表达式均无任何问题。汇编形式正确地回收了寄存器，且确实只用了三个寄存器。

1. **实验总结**

类成员变量是一种非常优秀的“全局变量”，它不会干扰类外的代码逻辑，而且非常容易管理（回收、修改等）。很多类成员函数都不需要很多参数，因为它可直接调用类成员变量，而且这些类成员变量往往是很有意义的，比如进一步计算或者打印；它们对于一个对象来说也是必要的存储空间，不会产生无谓的浪费。

另外，第三题（特别是汇编）让我对树遍历的认识加深了不少，特别是汇编用到的a方法直接输出存储，并且这也算是类成员变量的一个应用。更重要的是，说来惭愧，我写这道题的时候才想到表达式可以用二叉树表示（我之前只知道用栈，在这个题目上碰壁了，发现了新大陆）。