**项目介绍：**

利用huffman树原理实现常见类型文件无损压缩。通过数据结构堆进行排序

**项目流程：**

利用哈希思想统计文件中某字符出现的次数，讲字符出现次数做权重构建huffman树，根据生成的huffman生成每个字符对应的huffman编码 。

压缩：将所有字符出现次数先写在文件头方便解压时重构huffman树，后将每个字符转换成对应的huffman编码写入生成压缩文件。

解压：读取每个字符出现的次数重构huffman树，读取huffman编码进行转换。

**项目特点：**运用仿函数、模板、堆排序、构建huffman树、转化huffman编码实现基本功能。

通过光标定位函数实现简单UI，通过多线程实现进度条增长功能

项目过程：

1. 版本0.1完成main主函数以及简单switch case选择界面。
2. 版本0.2实现模板堆的头文件 实现向下调整 测试成功
3. 版本0.3实现堆的push 向上调整函数实现pop、top等接口函数
4. 版本0.4解决堆向上调整函数问题、优化向上调整函数。
5. 版本0.5实现通过堆来构建huffman树头文件。通过模板实现树类以及结点结构体
6. 版本0.6编写仿函数来比较huffman结点的大小，修改构建huffman树函数没有添加父亲节点的bug，解决不能访问字母块结构体里边的私有成员无法比较问题。
7. 版本0.7修改从堆获取数据时while循环的条件错误，修改没有获取堆里边最后一个根节点的问题。增加获取huffman树根节点函数。
8. 版本0.8编写生成对应huffmna编码函数。解决huffman编码反向问题
9. 版本0.9增加数据块结构体，把每个字符以及对应次数写到压缩文件的开头并且以结构体count为-1表示最后一个。把对应huffman编码写道对应文件。
10. 版本1.0编写解压缩函数
11. 版本1.1解决解压缩后文件内容变成乱码问题，解决解压缩文件输出同一个字符问题
12. 版本1.2解决解压缩后文件末尾少数据的问题，测试多种文件类型，成功！
13. 版本1.3增加光标移动函数，在程序开始增加程序名称、制作人以及版本信息。
14. 版本1.4减少编写时代码冗余问题，增强函数以及变量命名规范。
15. 版本1.5增加压缩、解压缩过程中进度条。修改进度条显示bug。
16. 版本1.6通过多线程方式解决进度条问题。解决进度条无法停止问题。修改多线程程序执行错误问题。
17. 版本1.7增加注释，改进耗时显示错误问题。

项目思路：

1. 简单的界面，选择压缩文件还是解压文件。
2. 文件压缩类里边有两个函数一个是压缩函数一个是解压函数可以来调用。私有成员是一个结构体数组\_Info，结构体里边保存的是关于某个字符的信息，首先要包括这个字符是啥，然后这个字符在文件里边出现的次数，以及这个字符最终被转换成的huffman编码是啥。
3. 在构造函数的时候把私有成员结构体数组进行初始化，一共只有256个字符所以把数组大小定义成256就行，然后让\_info[i]=i;即可，并且每个字符对应的count都设置为0；
4. 对文件进行压缩的话，先以二进制读的方式打开我们的文件，用fread函数来制定每次读取的大小是一个char类型的大小，然后通过while循环不断的读取，把读取到的字符找到他对应的结构体，然后把对应字符的count次数++,直到文件读取完遇到eof。
5. 然后根据字符出现的次数为权重开始构建huffman树，huffma树的结点是个单独的结构体huffmantreenode包括left，right，parent以及T类型的数据。这里要注意的时候，huffman编码不能访问我们的\_Info里边的数据，所以我们传了一个非法info进入，就是count=0的info，如果info[i]==非法制就说明这个i对应的字符没有出现，我们就不把他加到树里边的，这里就需要给info重载一个!=函数
6. 然后在构建huffman树之前我们先创建一个堆，用来出现次数最少的那个数据，堆寻找速度更快一点，这里创建一个小堆，然后把info里边count不是空的结点都插入到堆里边。
7. 堆我们用的是STL里边的vector来实现的。通过数组来模拟一个堆，给vector开辟一个相等大小的空间，然后把数据拷贝进来，运用向下调整函数把我们的堆调整成一个小堆。这里的vector里边放的都是huffman树的结点
8. 向下调整函数是从某个root开始，这里我们是用数组来实现的所以root就是一个下标。然后开始让child等于root\*2+1，来比较大小，比较之前先找到孩子里边小的那个，因为和大的交换没有意义，必须是父亲，左孩子右孩子三个里边最小的在堆顶也就是在父亲结点。所以我们先判断一个右孩子是不是比左孩子更小，如果更小就让右孩子和我们的parent去比。
9. 然后比较孩子结点和父亲结点，那个小，如果孩子结点更小就让父亲结点和孩子结点进行交换。
10. 一直往下调整，一直到最后一个元素。
11. 这里我们还需要写一个向上调整函数，因为我们的vector用的是pushback尾插，所以插入的结点都是在最后，这时候就要把新插入的结点向上调整，让新插入的结点不断的和他们的父亲来比，如果他比父亲要小就让他们两个交换，直到发现它比父亲结点要大，或者说到堆顶了就退出，这里没必要比较他和他的兄弟的关系，因为堆就是这样，他不是有序的只是，堆顶的一定是最小的，父亲一定要比孩子要小，没有限制兄弟结点之间的关系。
12. 给堆假如push、pop、size、top等函数。这里主要是说push和pop 的思想，push之后一定要进行一个向上调整，pop 的话，vector也是不支持头删的，但是我们每次取走的都是堆顶的数据也就是数组第一个数据，这时候我们就需要让第一个元素和最后一个元素进行交换，然后让最后一个出去，开始从0开始进行向下调整。
13. 我们把所有的huffman结点都入到堆里边，然后每次我们都从堆顶获取数据，先获取一个堆顶的数据他是最后一个叶子结点，然后让堆进行pop，之后再获取一个堆顶的数据，他是另一个孩子结点，然后创建一个父亲结点，结点的权重就是前边两个孩子结点之和。然后把他们的指针链接起来，这时候再把新的parent结点push到堆里边，堆会再次进行调整这时候再去获得他的前两个数据，算是每次出来两个禁区一个，直到堆的大小是2了，注意不能是1或者0，因为我们每次执行循环里边都要拿两个数据出来。然后让我们huffman的根节点等于堆顶的数据就可以了。
14. Huffman算是创建结束了，然后这里要添加一个函数就是获得huffman树的根节点，因为这是私有成员不能访问，所以必须增加一个函数。
15. 然后我们就需要把对应的字符生成他们的huffman编码单独写一个函数通过递归来实现。我们是倒着找的所以先递归找到你想编码的那个字符，然后开始倒着往回找，这就是为什么我们给huffman树节点添加了父亲结点，如果他等于父亲结点的左孩子那这个编码就是0，否则就是1，然后一直往上找，知道parent等于空了，这时候把我们获得的huffman编码先进行翻转，然后把他写入到对应字符结构体里边。这样我们就把所有的字符对应的huffman编码求出来了。
16. 然后开始压缩首先给新生成的文件取一个名字，再取一个独一无二的后缀这里要注意就是我们的huffman树是独一无二的，是根据我们文件特定生成的，所以很有可能你修改一个字符你的huffman树就有很大的改变，这里我们需要把huffman树给记录下来，不然等程序关了之后，没办法找到这个huffman树了，所以这里我们首先要做的就是把我们的huffman树给记录下来。
17. 这里我们用一个结构体coutblock里边包含每个字符，然后还有他对应出现的次数。然后把这些结构体一个一个写到压缩文件的开头，然后写完之后再写一个count=-1的结构体当成分界线。
18. 然后开始把生成的huffman编码写到文件里边，这时候对着我们的原文件来写，源文件读一个字符，然后把这个字符对应的huffman编码写到我们的新文件里边。
19. 这里需要注意的是为什么huffman会可以压缩文件，就是因为假如我们一个文章1000个字然后有一个字出现了600次以前这个字符就占了600\*1个字节，但是我们现在它对应的huffman编码可能就就是一个1.他就占了600/8个字节。
20. 我们读取一个字符然后通过结构体数组访问到他的huffman编码，我们往文件里边写的时候是以字节为单位的，所以这里需要注意要存在一个计数器，当计数器满8个了，说明够一个自己饿了，这时候执行一个写操作。我们一位一位的遍历他的huffman编码，如果遇到0就不操作了，因为本来我们value就是设置的0，然后如果遇到1了就让我们的value右移一下，让value|=1就是把最新的这一位变成0.然后count每次都要++如果到了8就说明要写了。
21. 这里千万要注意，读取完你的数据之后很有可能还有一部分你的编码没有写进去，因为每8个他才会写一次，可能最后不够8个，这里记着要补0，然后把补满的8个字节写进去。
22. 到这就算把我们的文件压缩完成了，关闭文件描述符就可以了。
23. 之后就是解压部分解压第一部分自然就是恢复我们的huffman树，每次读取一个数字块的大小，数字块就是我们刚刚在压缩文件开始写的，把每个字符和他对应的次数封装到了一个结构体里边，然后把一个一个结构体写到了文件头部。这时候我们通过fread每次读取一个数字块的大小，并且把读出来的数据给了我们的类成员里边的私有成员字母块。然后自然是和压缩的时候，把我们的字母块给传给huffman树的构造函数，这构造出来的huffman肯定是一样的。构建完之后就可以开始恢复我们的文件。
24. 打开一个新的文件，这时候自然还是需要去读取我们压缩后的文件来对比着解压。每次读取一个字符大小的数据，这里需要做一个判断，很有可能，我们第一次读就直接读到文件结束符了，很有可能就一个字符或者很少的，所以要判断一下。之后就是正常的读，while循环。如果这一位是0代表往左走，这一位是1代表往右走，如果走到不能往下走了说明到了叶子结点，把叶子节点结构体里边的ch也就是对应字符写入进去。然后因为huffman是出现的权重，所以根节点里边的count就是总共所有字符出现了多少次，所以每写进去一个就要让count--，直到小于等于0就说明所有字符都写完了。然后每八次循环之后就从压缩文件里边读取一个字节的huffman编码。
25. 就算解压完成了。