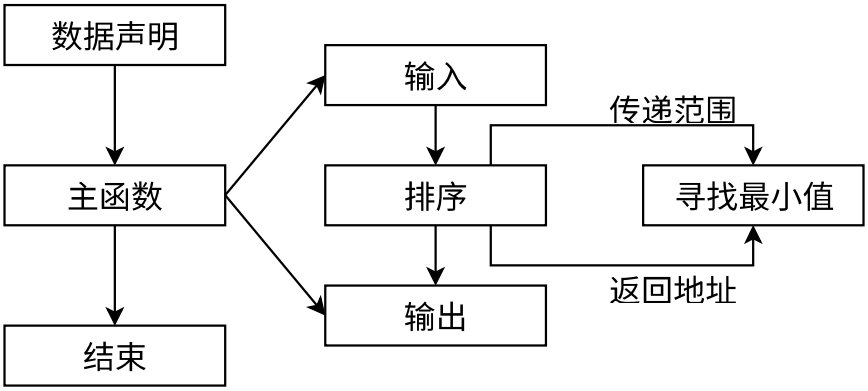
程序结构设计

为了方便理解，我们仿照C语言来对程序进行分解。按照功能模块来分，程序可以分为 6 个部分：数据定义、主函数、输入部分、输出部分、排序部分、寻找最小值部分。

注意，虽然这里使用了“函数”一词，但这只是为了便于理解，事实上并不需要写成一个完整的“函数”，只需能将不同的功能区分开，便于理解即可。



1. 首先，程序需要声明几个内存区域来存储数列、字符串、临时变量等。
2. 然后来到主函数，主函数依次调用输入函数、排序函数和输出函数。
3. 排序函数循环调用寻找最小值部分，传递给寻找最小值部分想要遍历的范围，而寻找最小值部分则返回一个最小值的地址，以供排序部分进行交换。
4. 最终，程序结束，完成功能。

关于通用寄存器的使用，实际上是一种**约定**，为了方便程序员之间的沟通而制定的规则。

1. 对于 s 寄存器而言，被调用者需要保证寄存器中的值在调用前后不能发生改变——对应到实际操作中，如果你想要编写一个子函数，那么在这个子函数中使用的所有 s 寄存器，都必须要在函数的开头入栈，在函数的结尾出栈，以确保其值在这个函数被调用前后不会发生变化。
2. 对于 t 寄存器而言则刚好相反，你编写的子函数中用到 t 寄存器的地方无需做任何保存，随意使用即可——因为维护 t 寄存器是上层函数的任务。这也就是所谓的 “s 寄存器由被调用者维护，t 寄存器由调用者维护”。
3. 需要注意的是，这仅仅是一个约定而已。你当然可以不遵守这种规范，但代价就是程序的合作和维护将变得举步维艰。在同学们编写 mips 代码的时候，尤其是涉及到子函数的编写时，我们推荐严格按照 mips 的规范进行，确保一个函数在被调用前后，其中所用到的 s 寄存器的值不会被改变。

一个调用者（即父函数）并不能预知其将要调用的子函数（即被调用者）会使用到哪些 t 寄存器，但可能在调用时并不想失去自己正在使用的某个 t 寄存器中的数据。

在这种情况下，为了维持 t 寄存器中的数据，调用者有两种选择：一是将所有 t 寄存器中的数据移至 s 寄存器，函数调用结束之后再移回来；二是将自己希望保留的 t 寄存器压入栈中，函数调用结束之后再弹回来。

第一种方法看似简单，但实际上引入了很多潜在的问题，比如：s 寄存器用完了怎么办？怎么确保子函数一定不会破坏 s 寄存器中的数据？在自动生成汇编代码（如编译）的过程中，怎样确定哪些 s 寄存器是可以用来保存 t 寄存器中的数据的？

因此，采用第二种方法，是一个更优雅，也更规范的做法。在第二种方法里，不再需要去考虑寄存器之间如何倒腾，只需要借助 sp 指针，不停地用栈去存取自己需要的数据就可以了。这减少了程序员的心智负担，规范了函数调用的过程，也方便了编译器的实现。

总而言之，调用者维护 t 寄存器，被调用者维护 s 寄存器的意义，就在于让代码更易于模块化。在这种约定下，调用者不需要去考虑被调用者的具体细节，被调用者也不需要去考虑自己被调用的方式。这使得 mips 代码可以以函数为单位进行模块化开发。