Allocator：

在该程序中，我们采用int动态数组memory作为内存来储存进程，memory[i]=-1表示该内存空间未被进程占用，为空；memory[i]=k表示该内存空间被进程Pk占用。

对于main()主函数，首先以命令行的形式，用argv[]数组从控制台承接memory的空间大小memory\_size，并初始化memory。

然后用command读入指令并判断。

当指令为EXIT时，用break退出循环，直接结束程序

指令为STATE时，调用state()函数，打印当前内存使用状况。

指令为C时，调用c()函数，合并未使用的内存空洞。

指令为RL时，需要再读取其后的进程编号p，然后将其传入release\_memory()函数中，让该函数找到并释放该进程。

指令为RQ时，需再读取其后的进程编号p，所需空间大小s和内训分配方式f，将其传入request\_memory()函数中，令该函数找到合适的内存空洞并放入该进程。

state()函数：需遍历memory，当遇到前后memory[i]内的数值不同时，即前一段内存与后一段内存的储存不同，按照格式输出前一段内存的开始位置和结束位置（用hole\_begin和hole\_end记录），并检查该段内存中储存的内容。若值为-1，表示该段内存未被使用；若值为k，表示被进程Pk使用。

c()函数：同样，需遍历memory，发现不同的内存段；不过与state()不同的是，当遇到一段未使用的内存段时，需要将其后紧跟着的进程移到前面来，即将空内存与满内存互换；完成后，遍历i的位置跳转到满内存之后的位置，继续向后检索，直到遍历完成。

release\_memory(int program)函数：得到需要释放的进程编号program的输入后，函数需在memory寻找对应的进程，若没有找到，则报错；否则，将对应进程的内存空间位置改为-1表示空，即可。

request\_memory(int program，int size，int fit)函数：跟据传入的fit，函数判断应用哪种方式分配内存，其对应关系为。

fit=0 对应first fit方式，调用first\_fit()函数

fit=1 对应worst fit方式，调用worst\_fit()函数

fit=2 对应best fit方式， 调用best\_fit()函数。

first\_fit(int program，int size)：用同样的方式遍历内存，找到空的内存段，当遇到一个大于size的空内存段时，将编号为program的进程写入，然后直接结束函数。

worst\_fit(int program，int size)：用同样的方式遍历内存，但每找到一段内存空洞，就与当前记录的最大的空洞作比较；直到遍历结束后，就可以得到现存的最大内存空洞的信息，若最大空洞都小于size，则该进程无法被写入，报错；否则就将编号为program的进程写入最大空洞中。

best\_fit(int program，int size)：用同样的方式遍历内存，但每找到一段内存空洞，需与size和当前记录的最小空洞比较，若大于size且小于最小空洞，则更新最小空洞的信息；直到遍历结束后，就可以得到最合适的内存空洞（即大于size的最小空洞）。需判断最小空洞信息是否被更新过，若没有更新过，意味着所有的内存空洞都小于size，则该进程无法被写入，报错；否则，将编号为program的进程写入该空洞中即可。

运行结果如下



