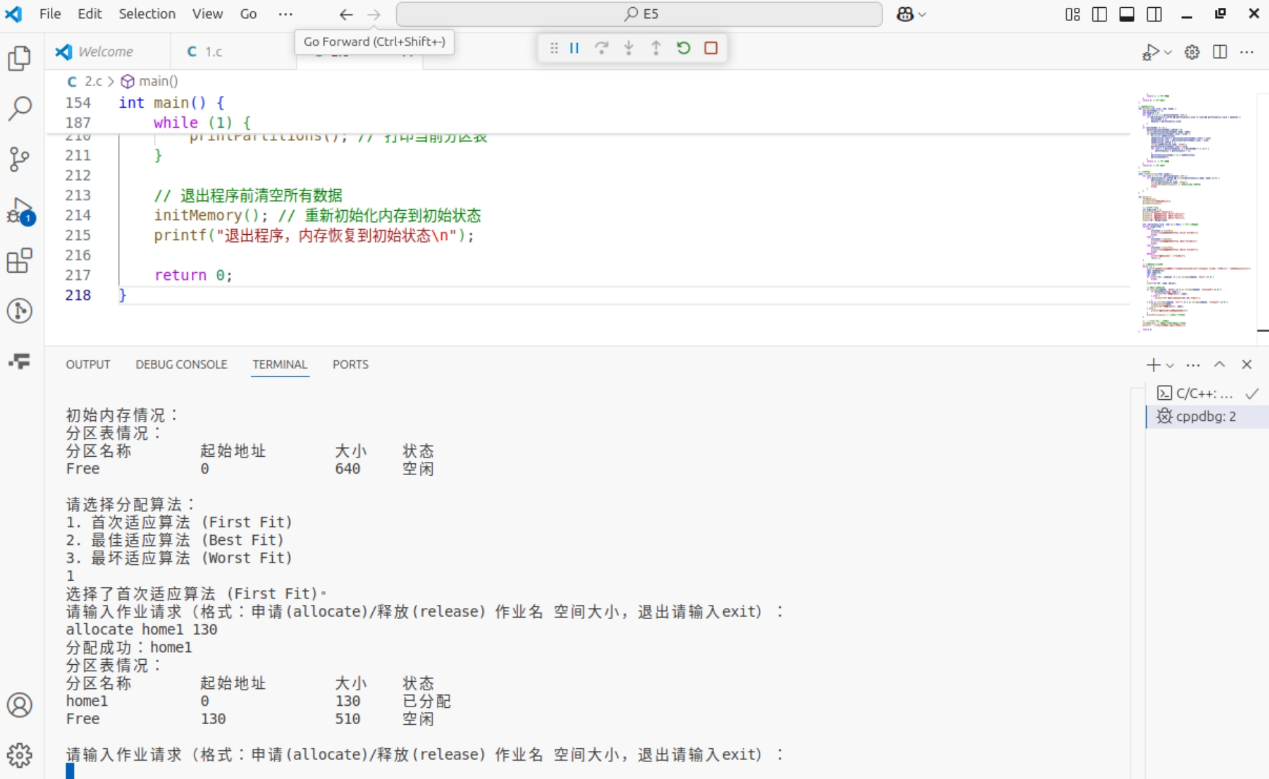
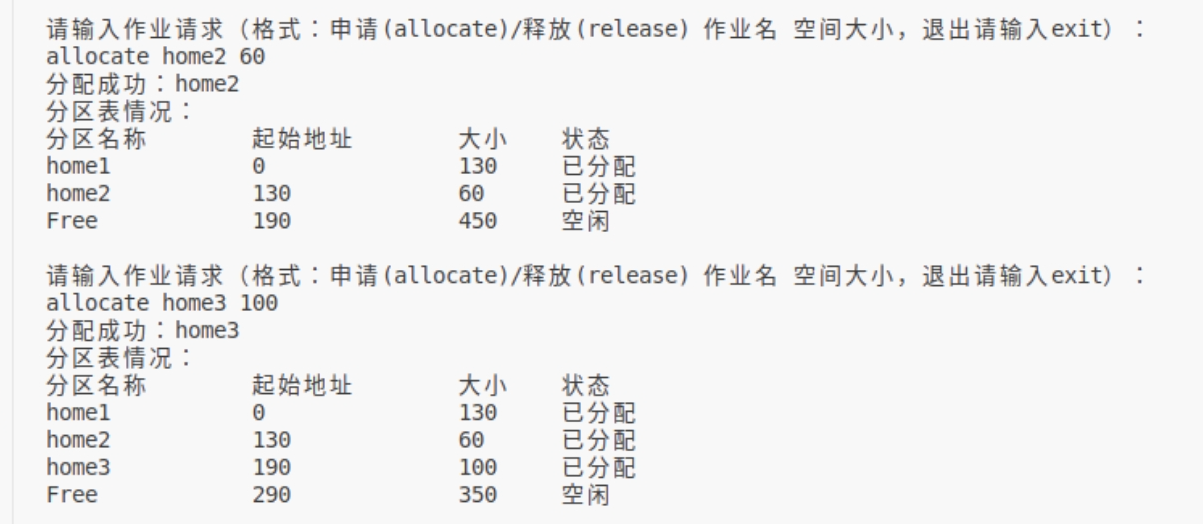
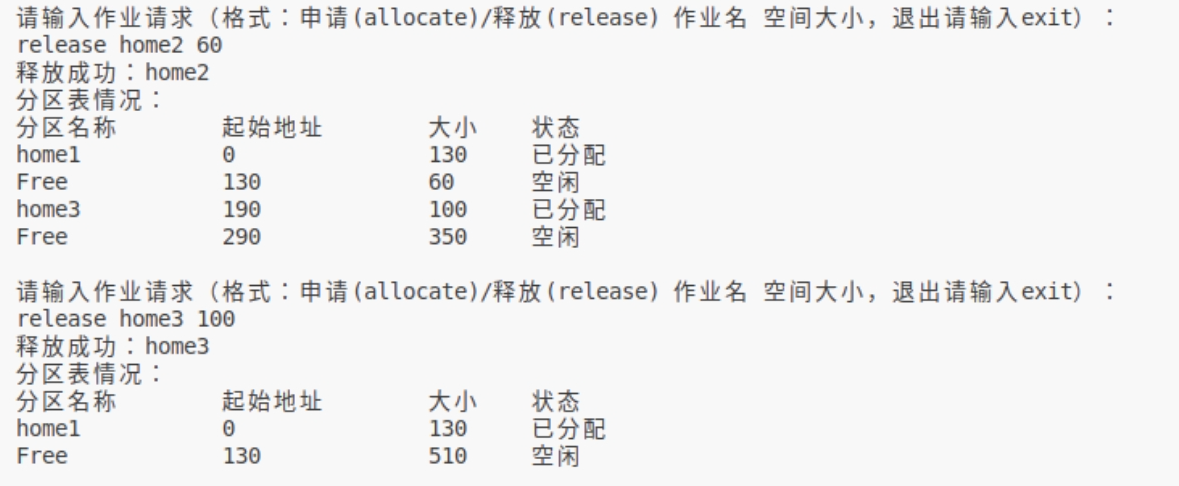
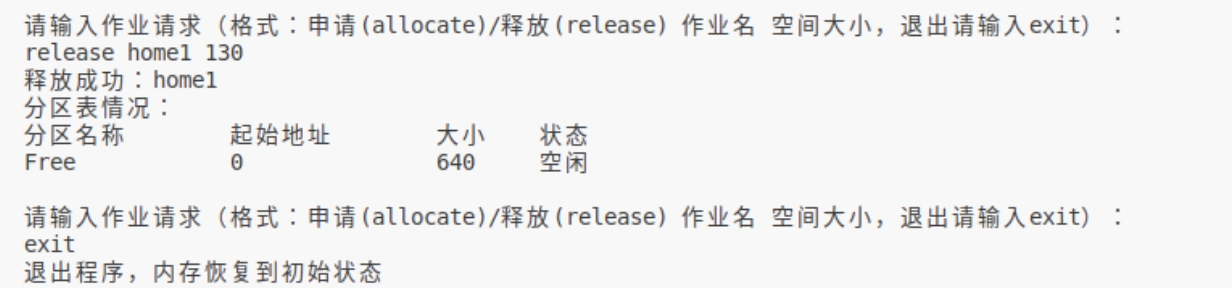
1. **首次适应算法**

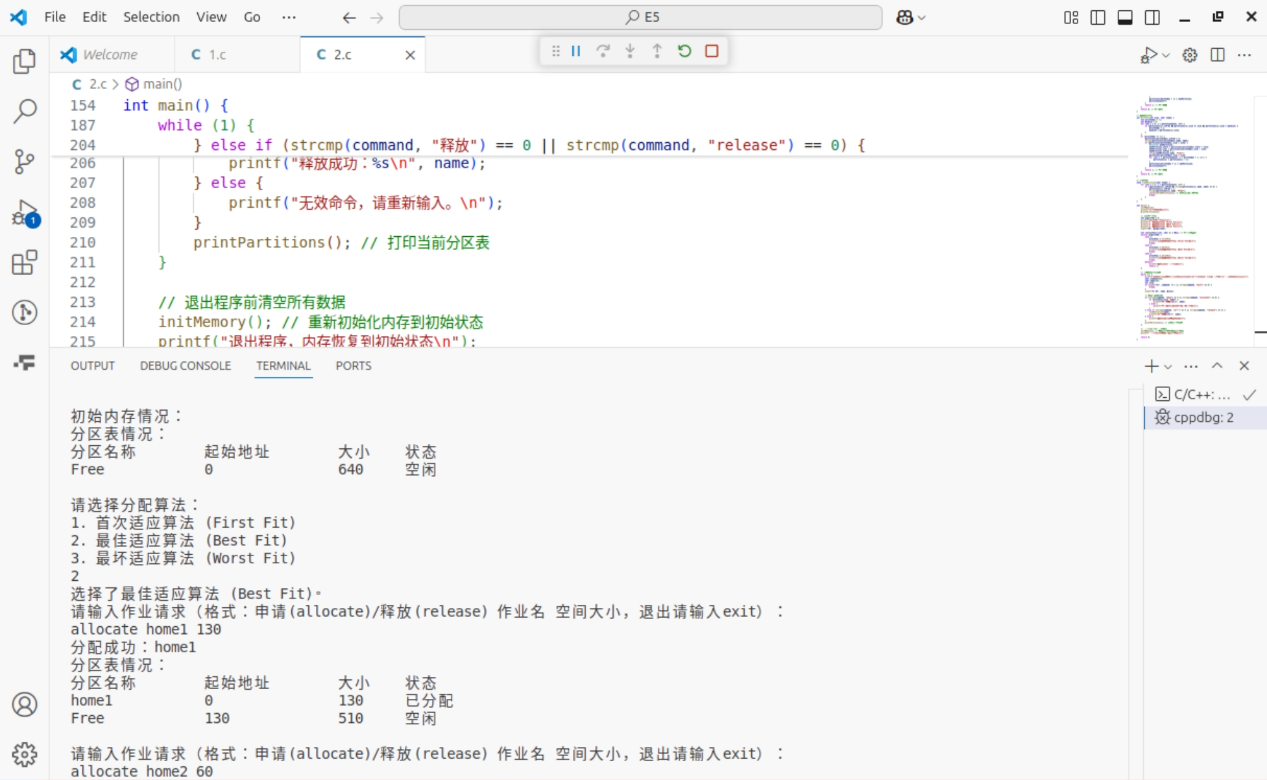
****

****

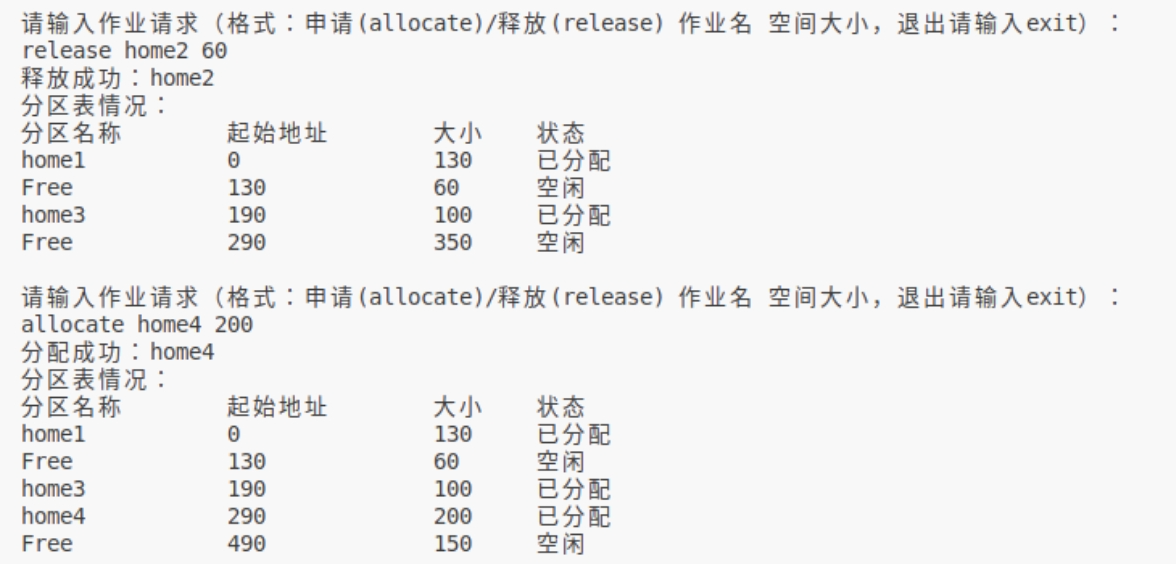
****

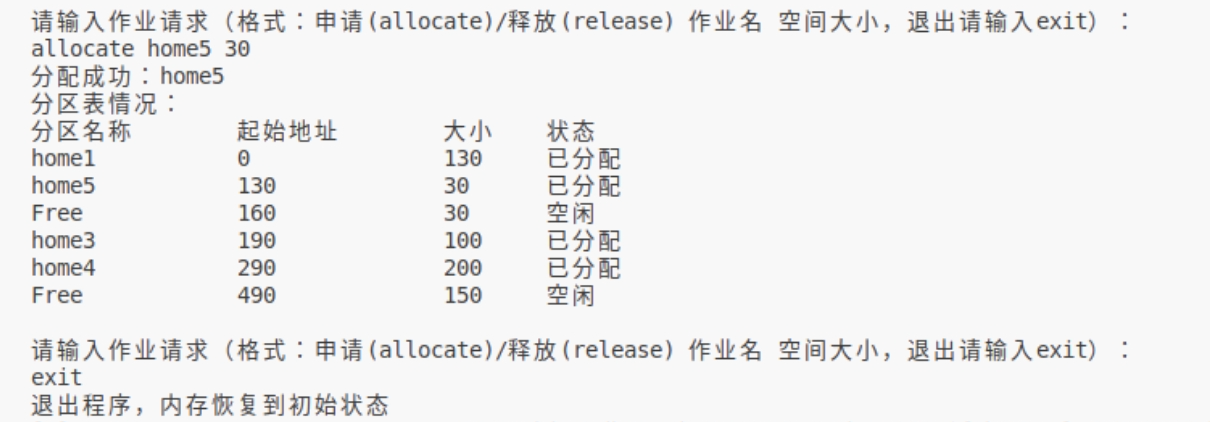
****

1. **最佳适应算法**

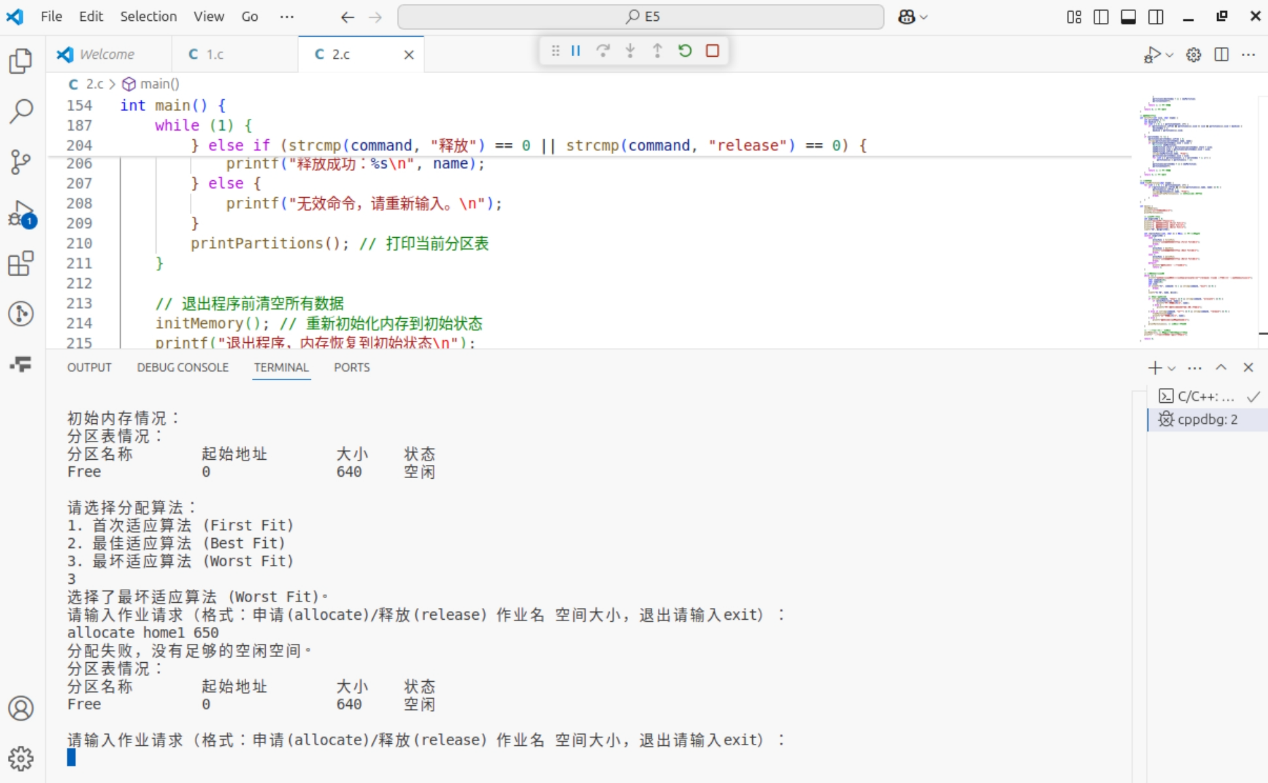
****

****

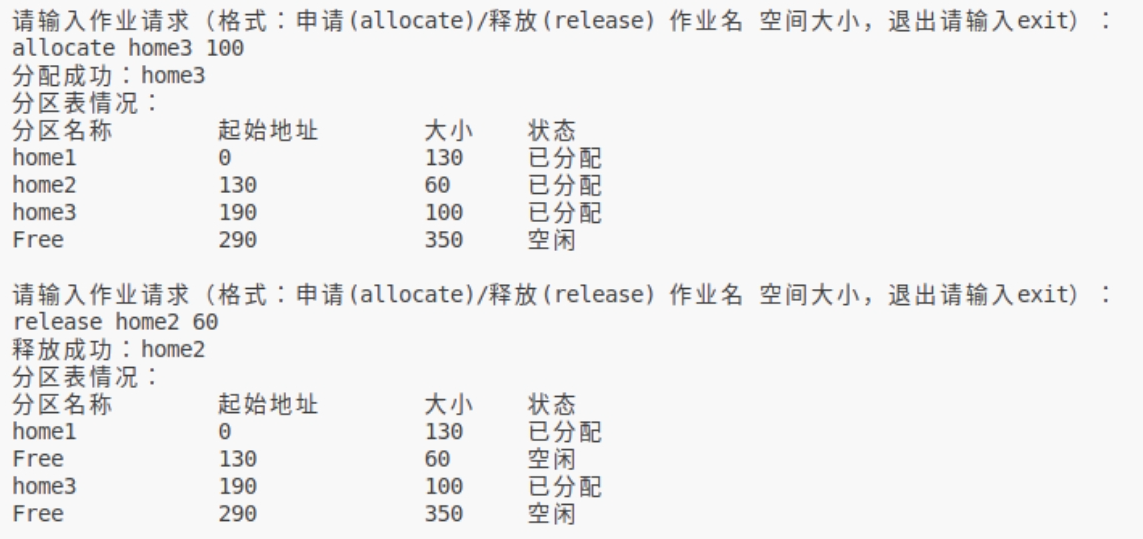
****

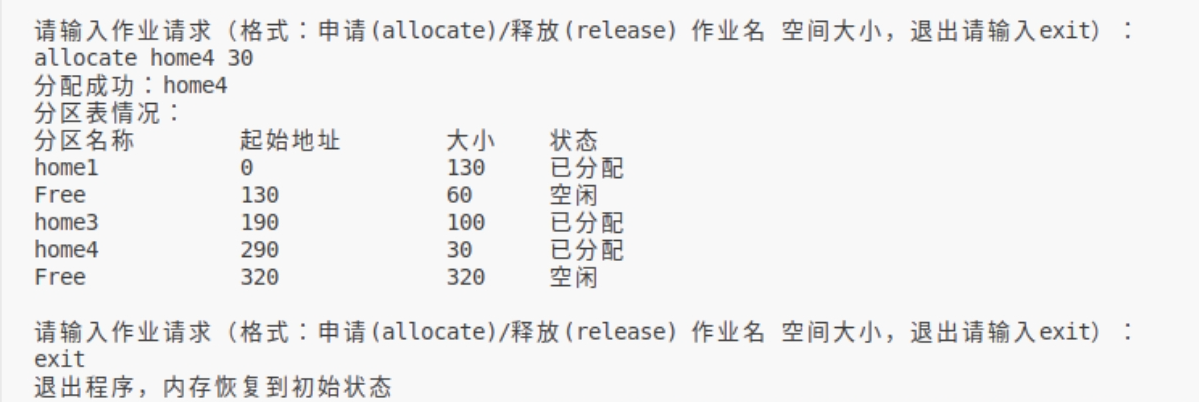
****

1. **最差适应分配**

****

****

****

****

1. **文字说明**
2. **首次适应算法（First Fit）**
   * 从头开始遍历内存分区列表。
   * 找到第一个足够大的空闲分区来满足内存请求。
   * 如果找到的空闲分区大于请求的大小，则将该分区分割成两部分：一部分分配给作业，另一部分保持空闲。
3. **最佳适应算法（Best Fit）**
   * 遍历整个内存分区列表。
   * 找到最小的、足够大的空闲分区来满足内存请求。
   * 同样，如果找到的空闲分区大于请求的大小，则将其分割。
4. **最坏适应算法（Worst Fit）**
   * 遍历整个内存分区列表。
   * 找到最大的、足够大的空闲分区来满足内存请求。
   * 如果找到的空闲分区大于请求的大小，则将其分割。
   * 这种方法试图保留小的空闲分区，以便未来可能的小型作业可以使用它们，但可能会导致大的空闲分区被分割，从而增加碎片。
5. **回收算法**

* 当一个分区被释放时，检查它前后的分区是否也是空闲的。
* 如果前一个分区是空闲的，并且两个分区是连续的（即前一个分区的结束地址加上其大小等于当前分区的起始地址），则将前一个分区的大小增加，以包含新释放的分区。
* 如果后一个分区是空闲的，并且两个分区是连续的，则将新释放的分区的大小增加，以包含后一个分区。
* 合并后，从分区数组中移除合并的分区，以保持数组的准确性。
* 这个过程确保了内存分区的连续性，减少了小的、不连续的空闲分区的数量，从而提高了内存的利用率。