**Memory Management Assessment**

**Question 1**

**Team member: BoNan Mai, HuaYi Lai, RuiMing Zhu**

# Code Score Statistics

The specific score and total score of each task have been recorded in the following table

(Table 1)

Table 1:Score Statistics

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task 1 | Task2 | Task 3 | Task 4 | Task 5 | Total Marks |
| 10/10 | **10/10** | **10/10** | **10/10** | **10/10** | **50/50** |

The following are some screenshots of the **final score** of each task after the program runs(Figure 1 to Figure 5)

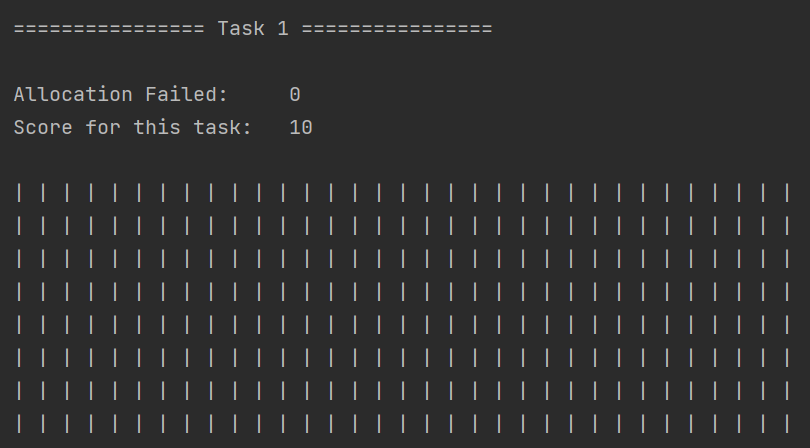


Figure 1:Score of Task 1

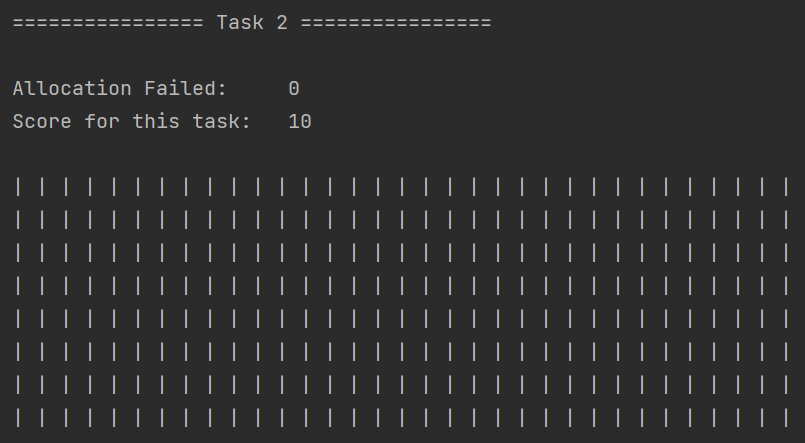


Figure 2:Score of Task 2

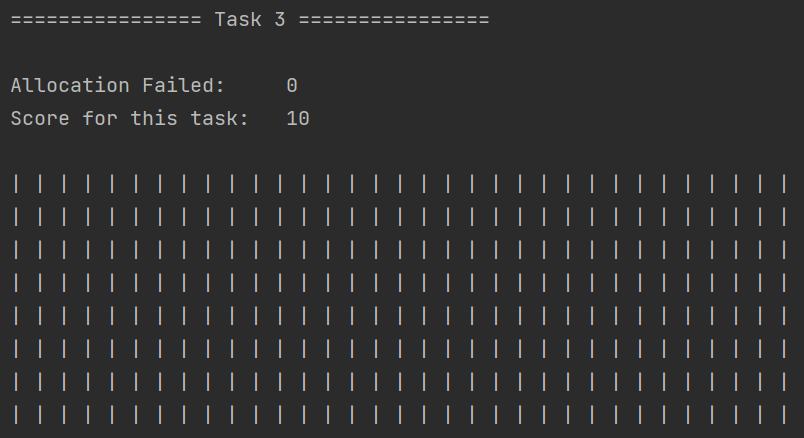


Figure 3:Score of Task 3

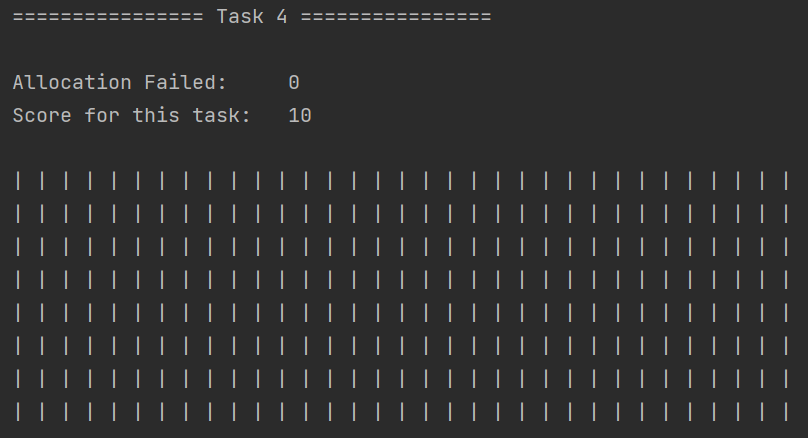


Figure 4:Score of Task 4

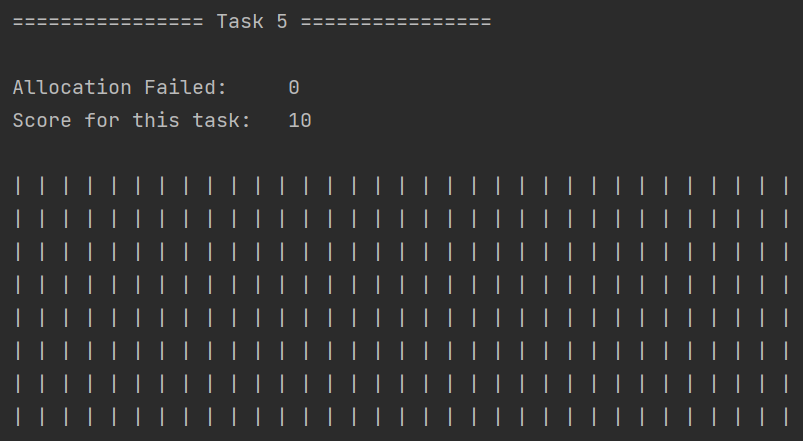


Figure 5:Score of Task 5

# Code that We Use

Here is a screenshot of the algorithm that we use(Figure 6):

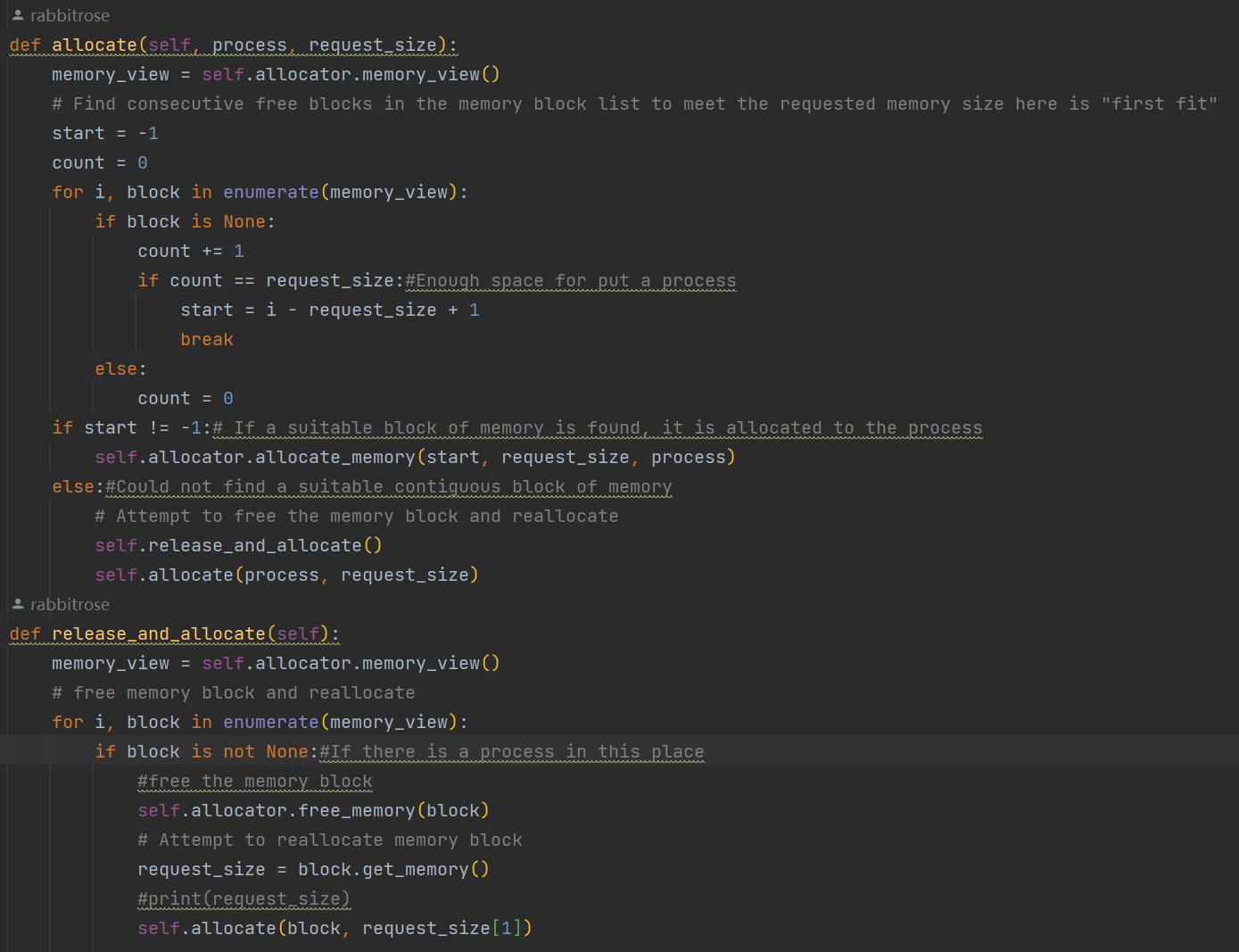


Figure 6: The algorithm that we designed

Here is the code that we designed (Seen in the following block),more detailed information about our algorithm will be explained in details at “Question 2” part

|  |
| --- |
| The *allocate* Function and the *release\_and\_allocate* Function |
| Function: that is the algorithm that we use |
| def allocate(self, process, request\_size):  memory\_view = self.allocator.memory\_view()  *# Find consecutive free blocks in the memory block list to meet the requested memory size here is "first fit"*  start = -1  count = 0  for i, block in enumerate(memory\_view):  if block is None:  count += 1  if count == request\_size:*#Enough space for put a process*  start = i - request\_size + 1  break  else:  count = 0  if start != 1:*# If a suitable block of memory is found, it is allocated to the process*  self.allocator.allocate\_memory(start, request\_size, process)  else:*#Could not find a suitable contiguous block of memory*  *# Attempt to free the memory block and reallocate*  self.release\_and\_allocate()  self.allocate(process, request\_size)  def release\_and\_allocate(self):  memory\_view = self.allocator.memory\_view()  *# free memory block and reallocate*  for i, block in enumerate(memory\_view):  if block is not None:*#If there is a process in this place*  *#free the memory block*  self.allocator.free\_memory(block)  *# Attempt to reallocate memory block*  request\_size = block.get\_memory()  self.allocate(block, request\_size[1]) |