实习题目：内存管理模型的设计与实现

【需求规格说明】

内存管理模型的设计与实现

对内存的可变分区申请采用链表法管理进行模拟实现。要求：

（1） 对于给定的一个存储空间自己设计数据结构进行管理，可以使用单个链表，也可以使用多个链表，自己负责存储空间的所有管理组织，要求采用分页方式（指定单元大小为页，如 4K，2K，进程申请以页为单位）来组织基本内容；

（2） 当进程对内存进行空间申请操作时，模型采用一定的策略（如：首先利用可用的内存进行分配，如果空间不够时，进行内存紧缩或其他方案进行处理）对进程给予分配；

（3） 从系统开始启动到多个进程参与申请和运行时，进程最少要有 3 个以上，每个进程执行申请的时候都应能对系统当前的内存情况进行查看；

（4） 对内存的申请进行内存分配，对使用过的空间进行回收，对给定的某种页面调度进行合理的页面分配。

（5） 利用不同的颜色代表不同的进程对内存的占用情况，动态更新这些信息。

【算法设计】

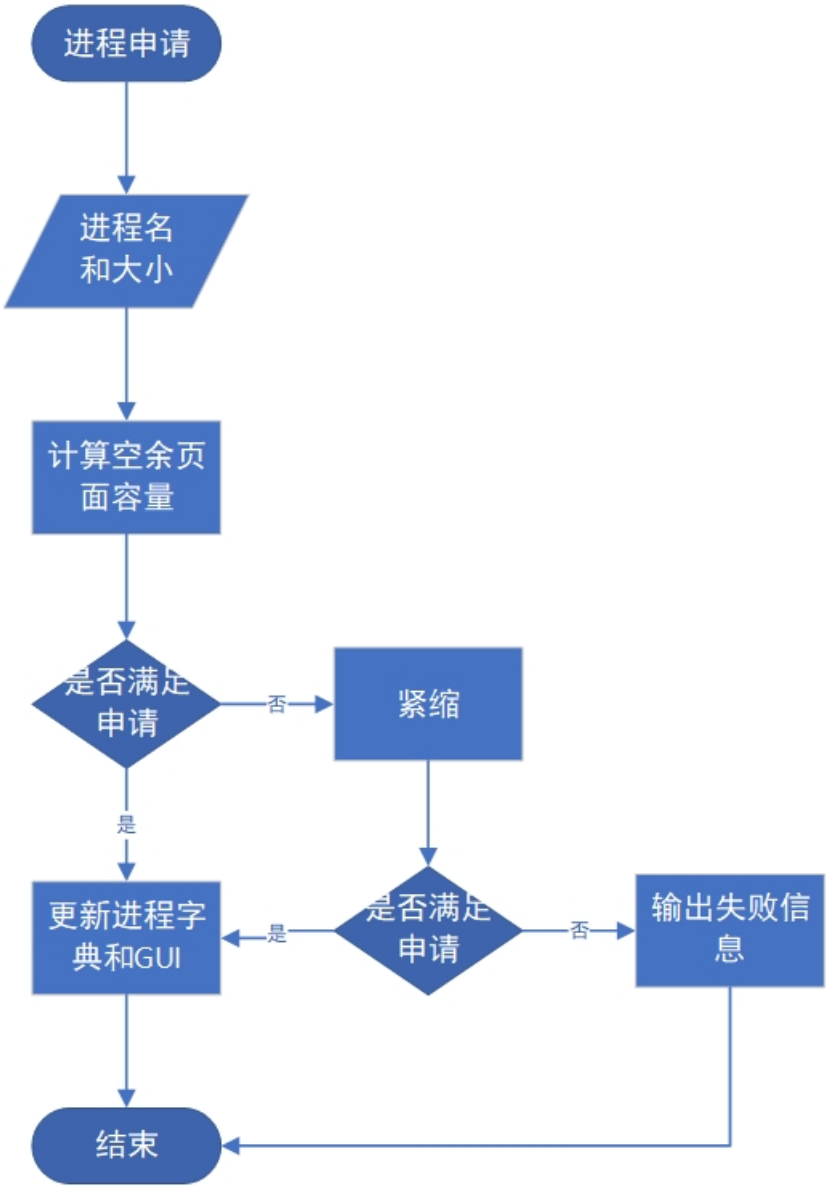
(程序基于 Python 3.6.3 和 Pyforms 3.0.0)

（1）**设计思想**：

约定内存页数为 16（索引地址为 0-15），进程用字典表示，键是进程的标识符，值是进程的起始地址和占用内存页数。

每当有新的进程申请，检测内存中是否有足够的剩余空间，若没有，则紧缩内存，再次检查，若还是没有空间则输出失败信息；若有，根据选择的算法进行内存空间分配

（2）**设计表示**：



（3）**详细设计表示：**

（GUI 组件和布局见附录）

主要变量：

self.lastAddress 记录上一次分配的内存地址，用于邻近适度算法。

self.processes 记录进程信息，包括进程的进程名、起始地址和大小。

主要方法：

getEmptyBlocks 获取空闲区域的起始地址和大小，函数会遍历整个内存，记录连续出现的‘N’（表示该页空闲）的位置和长度。

tightening 紧缩内存空间，从内存索引为 0 开始重新赋予进程起始地址，并根据起始地址更新 GUI。

apply 申请进程，从 GUI 获取进程的名字和大小，检查内存剩余空间是否满足进程需求，满足需求时根据不同算法调度内存空间。算法见下：

首次适度算法：从头遍历 getEmptyBlocks 返回的记录（数据类型为列表），找到第一个满足要求的索引，从该索引更新进程信息。

最优适度算法：从 getEmptyBlocks 返回的数据中减去进程需求，找到最小值对应的索引，从该索引更新进程信息。

邻近适度算法，遍历 getEmptyBlocks 返回的数据，第一次从索引大于 self.lastAddress 的索引中查找满足条件的索引，若找到，从该索引更新进程信息；否则，从头开始查找。

【调试报告】

问题 1：内存紧缩后程序崩溃
原因：紧缩后没有重新计算空闲区块，导致后续逻辑矛盾
解决：紧缩后重新计算空闲区块
问题 2：临近适度算法运行结果总是和首次适度算法相同
原因：上一次内存分配的地址和空闲页的地址在比较上出现逻辑错误
解决：修正比较算法

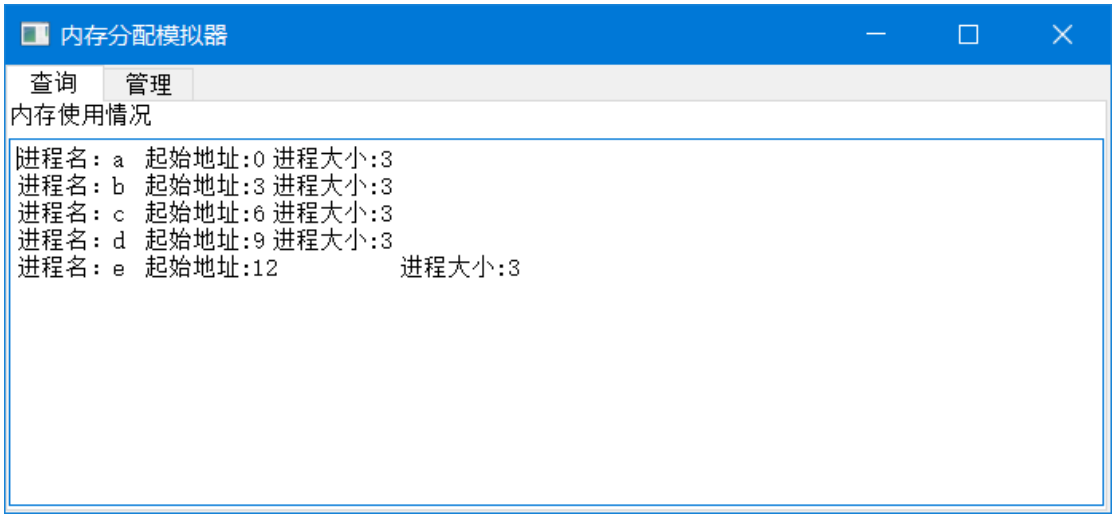
【附录】

运行截图

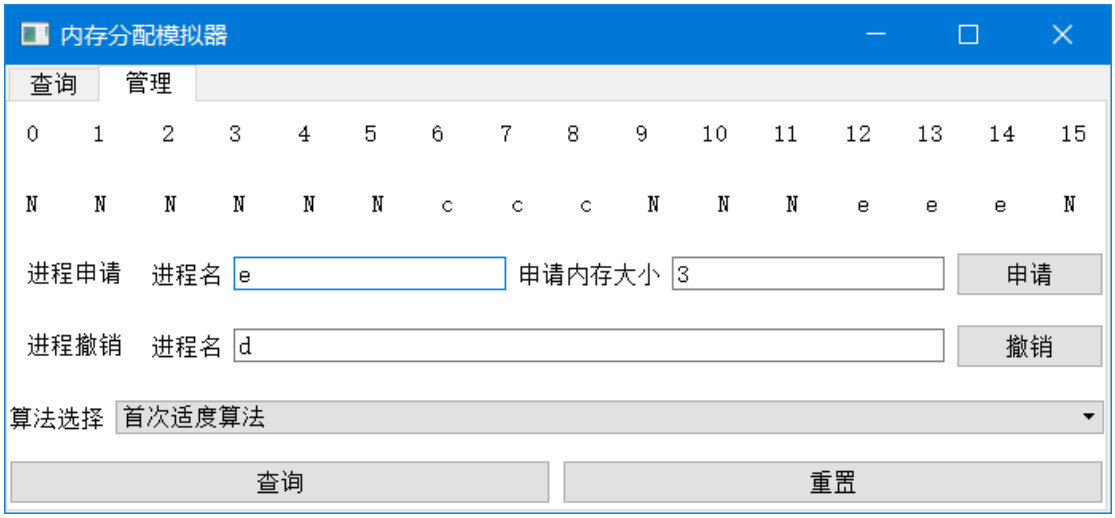
程序初始化
第一行表示内存页地址
第二行表示该页是否为空（N 表示空）

分别申请进程名为 a、b、c、d、e，内存大小都为 3 的进程，算法使用首次适度算法

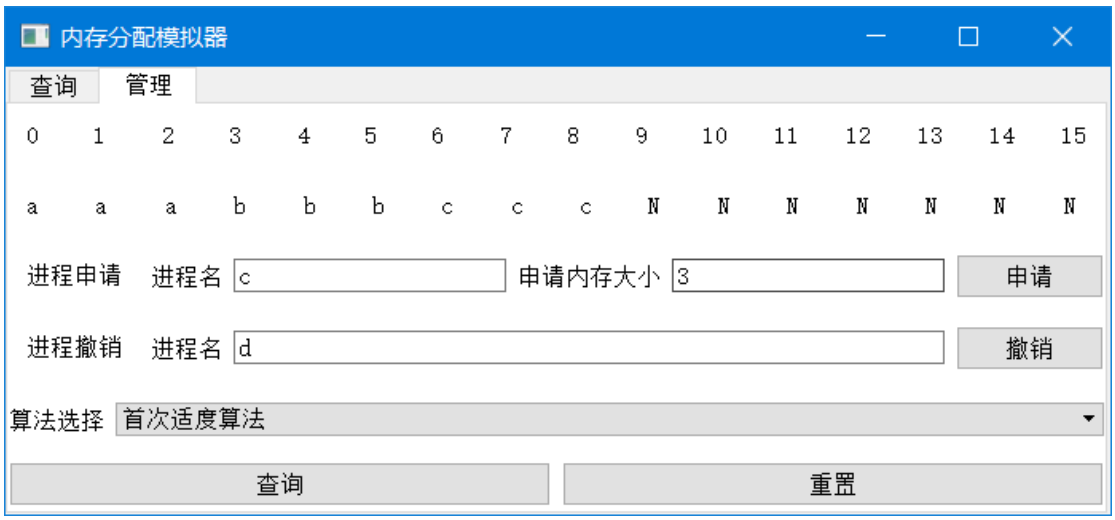
查询内存使用状态



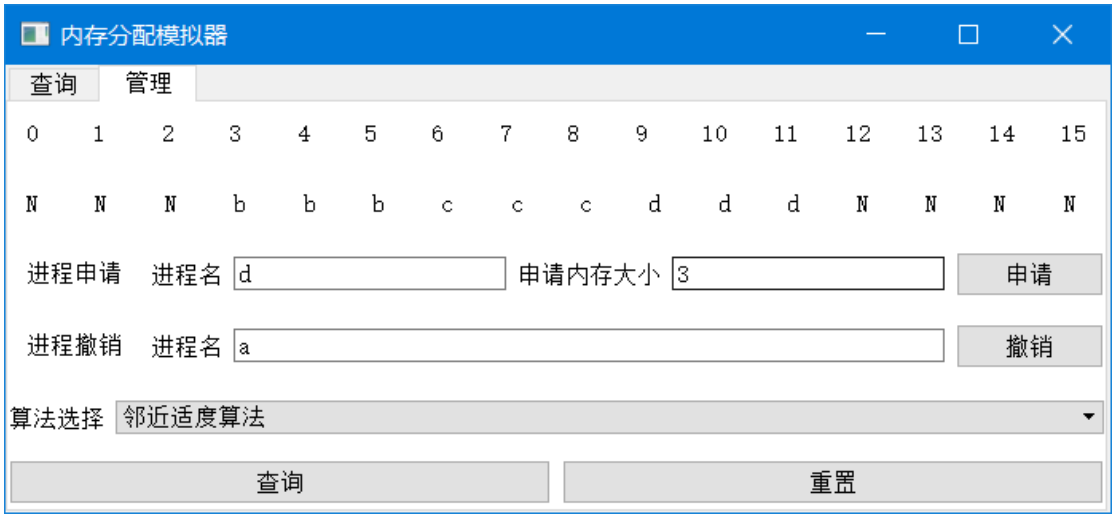
撤销进程 a、b、d，采用最优适度算法申请大小为 3 的进程 f



重置内存，分别申请进程名为 a、b、c 的 3 个进程，进程大小都为 3，采用首次适度算法

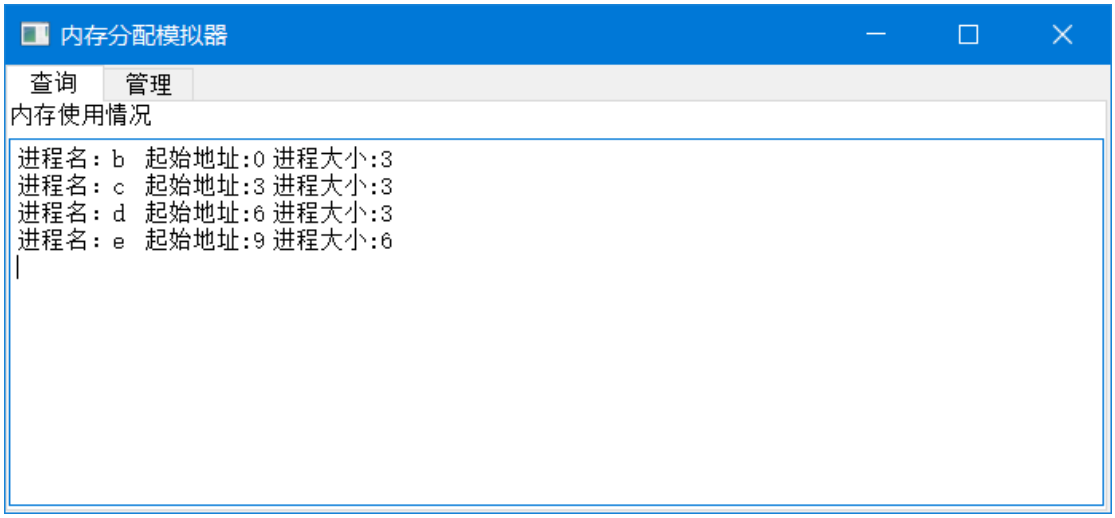


撤销进程 a，采用邻近适度算法申请大小为 3 的进程 d



通过内存紧缩，申请大小为 6 的进程 e

查询内存使用状态



源代码

```
import pyforms
from pyforms import BaseWidget
from pyforms.controls import *


class Memory(BaseWidget):

    def __init__(self):
        super(Memory, self).__init__("内存分配模拟器")
        # 进程字典
        # 键值对为编号和实例
        self.processes = dict()
        # GUI
        # 查询页
        self.textArea = ControlTextArea('内存使用情况')
        # 管理页
        # 第一行 内存地址
        self.memoryMark_0 = ControlLabel('0')
        self.memoryMark_1 = ControlLabel('1')
        self.memoryMark_2 = ControlLabel('2')
        self.memoryMark_3 = ControlLabel('3')
        self.memoryMark_4 = ControlLabel('4')
        self.memoryMark_5 = ControlLabel('5')
        self.memoryMark_6 = ControlLabel('6')
        self.memoryMark_7 = ControlLabel('7')
        self.memoryMark_8 = ControlLabel('8')
        self.memoryMark_9 = ControlLabel('9')
        self.memoryMark_10 = ControlLabel('10')
```

```
        self.memoryMark_11 = ControlLabel('11')
        self.memoryMark_12 = ControlLabel('12')
        self.memoryMark_13 = ControlLabel('13')
        self.memoryMark_14 = ControlLabel('14')
        self.memoryMark_15 = ControlLabel('15')
        # 第二行 内存使用进程标记
        self.memoryUser_0 = ControlLabel('')
        self.memoryUser_1 = ControlLabel('')
        self.memoryUser_2 = ControlLabel('')
        self.memoryUser_3 = ControlLabel('')
        self.memoryUser_4 = ControlLabel('')
        self.memoryUser_5 = ControlLabel('')
        self.memoryUser_6 = ControlLabel('')
        self.memoryUser_7 = ControlLabel('')
        self.memoryUser_8 = ControlLabel('')
        self.memoryUser_9 = ControlLabel('')
        self.memoryUser_10 = ControlLabel('')
        self.memoryUser_11 = ControlLabel('')
        self.memoryUser_12 = ControlLabel('')
        self.memoryUser_13 = ControlLabel('')
        self.memoryUser_14 = ControlLabel('')
        self.memoryUser_15 = ControlLabel('')
        self.memoryUser = [self.memoryUser_0, self.memoryUser_1, self.memoryUser_2,
self.memoryUser_3,
                           self.memoryUser_4, self.memoryUser_5, self.memoryUser_6,
self.memoryUser_7,
                           self.memoryUser_8, self.memoryUser_9, self.memoryUser_10,
self.memoryUser_11,
                           self.memoryUser_12, self.memoryUser_13,
self.memoryUser_14, self.memoryUser_15
                           ]
        self.memoryUserMark = [False] * 16
        for i in self.memoryUser:
            i.value = 'N'
        # 第三行 进程申请内存
        self.applyLabel = ControlLabel('进程申请')
        self.applyName = ControlText('进程名')
        self.applySpace = ControlText('申请内存大小')
        self.applyBtn = ControlButton('申请')
        # 第四行 进程撤销
        self.deleteLabel = ControlLabel('进程撤销')
        self.deleteName = ControlText('进程名')
        self.deleteBtn = ControlButton('撤销')
        # 第五行 算法选择
```

```
        self.algorithm = ControlCombo('算法选择')
        self.algorithm.add_item('首次适度算法', 0)
        self.algorithm.add_item('最优适度算法', 1)
        self.algorithm.add_item('邻近适度算法', 2)
        # 第六行 查询和重置
        self.query = ControlButton('查询')
        self.reset = ControlButton('重置')
        # 绑定按钮和事件
        self.applyBtn.value = self.apply
        self.deleteBtn.value = self.delete
        self.query.value = self.query_s
        self.reset.value = self.setMemoryEmpty
        # GUI 布局
        self.formset = [{
            '管理': [(
                    'memoryMark_0', 'memoryMark_1', 'memoryMark_2', 'memoryMark_3',
                    'memoryMark_4', 'memoryMark_5', 'memoryMark_6', 'memoryMark_7',
                    'memoryMark_8', 'memoryMark_9', 'memoryMark_10',
'memoryMark_11',
                    'memoryMark_12', 'memoryMark_13', 'memoryMark_14',
'memoryMark_15'
                ),
                (
                    'memoryUser_0', 'memoryUser_1', 'memoryUser_2', 'memoryUser_3',
                    'memoryUser_4', 'memoryUser_5', 'memoryUser_6', 'memoryUser_7',
                    'memoryUser_8', 'memoryUser_9', 'memoryUser_10',
'memoryUser_11',
                    'memoryUser_12', 'memoryUser_13', 'memoryUser_14',
'memoryUser_15',
                ),
                ('applyLabel', 'applyName', 'applySpace', 'applyBtn'),
                ('deleteLabel', 'deleteName', 'deleteBtn'),
                ('algorithm'),
                ('query', 'reset')
            ],
            '查询':['textArea']
        }]
        # 上次分配地址
        self.lastAddress = 0

    # 获取空闲内存页
    def getEmptyBlocks(self):
        blocks = list()
        i = 0
```

```
        while i < 16:
            if self.memoryUser[i].value == 'N':
                size = 1
                blocks.append(i)
                while i < 15:
                    i += 1
                    if self.memoryUser[i].value == 'N':
                        size += 1
                    else:
                        blocks.append(size)
                        break
            i += 1
        if len(blocks) & 1 == 1:
            temp = blocks[len(blocks)-1]
            blocks.append(16-temp)
        return blocks

    # 用进程名更新进程占用的内存页
    def changeMemory(self, name, index, size):
        for i in range(index, index+size):
            self.memoryUser[i].value = name

    # 撤销进程后重置对应内存页为空
    def resetMemory(self, index, size):
        for i in range(index, index+size):
            self.memoryUser[i].value = 'N'

    # 重置数据为空
    def setMemoryEmpty(self):
        for i in range(16):
            self.memoryUser[i].value = 'N'
        self.processes = dict()
        self.lastAddress = 0

    # 紧缩
    def tightening(self):
        for i in range(16):
            self.memoryUser[i].value = 'N'
        lastIndex = 0
        for i in self.processes:
            process = self.processes[i]
            process[0] = lastIndex
            self.changeMemory(i, process[0], process[1])
            self.lastAddress = lastIndex
```

```
            lastIndex += process[1]

    # 进程申请
    def apply(self):
        blocks = self.getEmptyBlocks()
        index = list()
        size = list()
        for i in range(int(len(blocks)/2)):
            index.append(blocks[i*2])
            size.append(blocks[i*2+1])
        need = int(self.applySpace.value)
        name = self.applyName.value
        # 若剩余空间不满足申请要求则紧缩
        if need > max(size):
            self.tightening()
        # 在紧缩的基础上若剩余空间不满足申请要求则输出失败信息
        blocks = self.getEmptyBlocks()
        index = list()
        size = list()
        for i in range(int(len(blocks)/2)):
            index.append(blocks[i*2])
            size.append(blocks[i*2+1])
        need = int(self.applySpace.value)
        name = self.applyName.value
        if need > max(size):
            s = '进程'
            s += name
            s += '申请失败\n'
            s += '当前最大空闲区域页数为'
            s += str(max(size))
            self.textArea.value = s
        # 若满足申请要求则根据选择的算法进行分配
        else:
            # 首次适度算法
            if self.algorithm.value == 0:
                for i in range(len(index)):
                    if need <= size[i]:
                        self.changeMemory(name, index[i], need)
                        self.processes[name] = [index[i], need]
                        self.lastAddress = index[i]
                        break
            # 最优适度算法
            elif self.algorithm.value == 1:
                bestSize = [i - need for i in size]
```

```
                size = []
                for i in bestSize:
                    if i >= 0:
                        size.append(i)
                bestIndex = 0
                bestMin = 16
                for i in range(len(size)):
                    if bestMin > size[i]:
                        bestMin = size[i]
                        bestIndex = i
                self.changeMemory(name, index[bestIndex], need)
                self.processes[name] = [index[bestIndex], need]
                self.lastAddress = index[bestIndex]
            # 邻近适度算法
            else:
                compare = self.lastAddress
                for i in range(len(index)):
                    if compare <= index[i] and need <= size[i]:
                        self.changeMemory(name, index[i], need)
                        self.processes[name] = [index[i], need]
                        self.lastAddress = index[i]
                        return
                for i in range(len(index)):
                    if need <= size[i]:
                        self.changeMemory(name, index[i], need)
                        self.processes[name] = [index[i], need]
                        self.lastAddress = index[i]

# 进程撤销
def delete(self):
    name = self.deleteName.value
    if name in self.processes:
        process = self.processes[name]
        del self.processes[name]
        self.resetMemory(process[0], process[1])

# 查询
def query_s(self):
    showValue = ''
    for i in self.processes:
        process = self.processes[i]
        s = '进程名：' + i
        s += '\t 起始地址:'
        s += str(process[0])
```

```
                s += '\t 进程大小:'
                s += str(process[1])
                s += '\n'
                showValue += s
            self.textArea.value = showValue


if __name__ == "__main__":
                                    pyforms.start_app(Memory)
```