#4: Memory Manager

Вопрос эффективного управления памятью являются одним из ключевых при разработке игрового движка, и зачастую, от эффективности менеджера памяти зависит скорость работы игры. Зачастую в сложных программных системах используются различные менеджеры памяти, которые используются в различных сценариях выделения памяти в зависимости от времени жизни и размера выделяемого участка памяти.

Задание

В вашу задачу входит реализация универсального менеджера памяти на основе использования подходов Fixed-size Memory Allocation (FSA) и Coalesce Allocation с Free-List. FSA должен использоваться для выделения блоков размером 16, 32, 64, 128, 256 и 512 байт, все запросы блоков размером больше 10Мб передаются непосредственно ОС.

```
class MemoryAllocator
```

должен поддерживать следующие операции:

- `MemoryAllocator();`
  Создает неинициализированный аллокатор. Инициализация и деинициализация аллокатора являются критическими и затратными операциями, поэтому их выполнение реализуется явно специальными методами. Отладочная версия аллокатора должна во всех методах assert-ить была ли выполнена инициализация.
- `virtual ~MemoryAllocator();`
  Деструктор освобождает все блоки памяти, запрошенные у операционной системы. Отладочная версия должна также assert-ить, что был вызван метод деинициализации аллокатора.
- `virtual void init();`
  Выполняет инициализацию аллокатора, запрашивая необходимые страницы памяти у ОС и, при необходимости, резервируя адресное пространство под будущие запросы.

- `virtual void destroy();`
  Выполняет деинициализацию аллокатора, возвращая всю запрошенную память ОС. Отладочная версия аллокатора должна assert-ить в случае, если при деинициализации имеется занятая память (возникла утечка памяти).
- `virtual void *alloc(size_t size);`
  Выделяет блок памяти размером size байт, выровненный по границе в 8 байт.
- `virtual void free(void *p);`
  Освобождает занимаемую память, на которую указывает указатель p. Отладочная версия аллокатора также проверяет, что указатель p является корректным указателем на начало блока памяти, управляемым этим аллокатором.
- `virtual void dumpStat() const;`
  Метод должен быть присутствовать только в отладочной версии аллокатора. Выводит в стандартный поток вывода статистику по аллокатору: количество занятых и свободных блоков, список блоков запрошенных у ОС и их размеры. Можно также подсчитывать статистику аллокаций: сколько памяти и какого размера было запрошено, освобождено, построить гистограмму распределения количества запросов памяти в зависимости от размера блока FSA и т.д.
- `virtual void dumpBlocks() const;`
  Метод должен быть присутствовать только в отладочной версии аллокатора. Выводит в стандартный поток вывода список всех этим занятых блоков: их адреса и размеры.

Пример использования: запросим память для хранения переменных типа целое число, число с плавающей точкой двойной точности и массива из 10 целых чисел, затем распечатаем статистику.

```
MemoryAllocator allocator;
allocator.init();

int *pi = (int *)allocator.alloc(sizeof(int));
double *pd = (double *)allocator.alloc(sizeof(double));
```

```
int *pa = (int *)allocator.alloc(10 * sizeof(int));

allocator.dumpStat();
allocator.dumpBlocks();

allocator.free(pa);
allocator.free(pd);
allocator.free(pi);
allocator.destroy();
```