Лабораторная работа №6

Задача: Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР №5) спроектировать и разработать аллокатор памяти для динамической структуры данных.

Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти.
Аллокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-ого уровня, согласно варианту задания).
Для вызова аллокатора должны быть переопределены операторы new и delete у классов-фигур.

Фигуры: треугольник, квадрат, прямоугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: бинарное дерево.

1 Введение

Поскольку памятью в C++ приложении программист управляет самостоятельно, то и за фрагментацией памяти приходится следить самим. Память фрагментируется, если приложение работает достаточно долго и при этом активно работает с памятью.
Аллокатор - это действующая по определённой логике высокоуровневая прослойка между запросами памяти под динамические объекты и стандартными сервисами выделения памяти (new/malloc или другими (например запросами напрямую к ядру о.с.)), конечно же прослойка берет на себя и вопросы управлением отдачей уже ненужной памяти назад. По другому можно сказать - что аллокатор это реализация стратегии управления памятью.

2 Код программы

TAllocationBlock.h

```
#ifndef ALLOCATIONBLOCK_H
#define ALLOCATIONBLOCK_H

class AllocationBlock
{
public:
AllocationBlock();
AllocationBlock(size_t s,size_t c);

void *allocate();
void deallocate(void *pointer);

bool HasFreeBlocks();

virtual ~AllocationBlock();
private:
size_t size;
size_t count;

char* used_blocks;
void** free_blocks;

size_t free_count;
```

```
};

#endif
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"
#include <iostream>

AllocationBlock::AllocationBlock() : size(0),count(0),used_blocks(0),free_blocks(0),fr
{}

AllocationBlock::AllocationBlock(size_t s,size_t c) : size(s),count(c)
{
used_blocks = (char*)malloc(size*count); // выделили память под указанное число
блоков размера size
free_blocks = (void**)malloc(sizeof(void*)*count); // создали указатели
for (int i = 0; i <count; i++)
free_blocks[i] = used_blocks + i*size;
free_count = count;
std::cout <<"TAllocationBlock: Memory init" <<std::endl;
}

void *AllocationBlock::allocate()
{
void *result = nullptr;
if (free_count >0)
{
result = free_blocks[free_count -1];
free_count--;
std::cout <<"AllocationBlock: Allocate " <<(count -free_count) <<" of " <<count
<<std::endl;
}
else
{
std::cout <<"TAllocationBlock: No memory exception :-)" <<std::endl;
}
return result;
}

void AllocationBlock::deallocate(void *pointer)
```

```
{
std::cout <<"TAllocationBlock: Deallocate block " <<std::endl;
free_blocks[free_count] = pointer;
free_count++;
}

bool AllocationBlock::HasFreeBlocks()
{
return free_count>0;
}

AllocationBlock::~AllocationBlock()
{
if (free_count<count)
std::cout <<"TAllocationBlock: Memory leak?" <<std::endl;
else
std::cout <<"TAllocationBlock: Memory freed" <<std::endl;
delete free_blocks;
delete used_blocks;
}
```

3 Вывод программы:

```
TAllocationBlock: Memory init
----------------------------------------
-------------------МЕНЮ-------------------
|1-Добавить треугольник                  |
|2-Добавить прямоугольник                |
|3-Добавить квадрат                      |
|4-Удалить фигуру                        |
|5-Расспечатать список                   |
|6-Выход                                 |
Выберете действие:
2
Введите значение a:9
Введите значение b:2
Введите индекс: 0
Список создан
Выберете действие:
1
```

```
Введите значение a:2
Введите значение b:2
Введите значение c:2
Введите индекс: 0
Список создан
Выберете действие:
3
Введите значение a:1
Введите индекс: 1
Список создан
Выберете действие:
2
Введите значение a:1
Введите значение b:1
Введите индекс: 2
Список создан
Выберете действие:
5
Треугольник со сторонами [2,2,2]
Прямоугольник со сторонами [9,2]
Квадрат со стороной [1]
Прямоугольник со сторонами [1,1]
Выберете действие:
4
Введите индекс: 1
Выберете действие:
5
Треугольник со сторонами [2,2,2]
Квадрат со стороной [1]
Прямоугольник со сторонами [1,1]
Выберете действие:
6
TAllocationBlock: Memory freed
```

4 Вывод

В данной лабораторной работе я познакомилась с таким полезным понятием, как аллокатор памяти. Использование аллокаторов позволяет добиться существенного повышения производительности в работе с динамическими объектами (особенно с объектами-контейнерами).