Московский авиационный институт

(национально исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра прикладной математики и информатики

Лаборатория №6 по курсу «Объектно-ориентируемое программирование»

Студент: Рождественских Аднрей Сергеевич

Преподаватель: А. В. Поповкин

Группа: М80-206Б

Вариант: 15

Дата:

Оценка:

Подпись:

Москва, 2018.

1 Цель работы

• Закрепление навыков по работе с памятью в C++.

• Создание аллокаторов памяти для динамических структур данных.

2 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.

3 Описание

Аллокатор памяти – часть программы (как прикладной, так и операционной системы), обрабатывающая запросы на выделение и освобождение оперативной памяти или запросы на включение заданной области памяти в адресное пространство процессора.

Основное назначение аллокатора памяти в первом смысле – реализация динамической памяти. В языке С динамическое выделение памяти производится через функцию malloc.

Программисты должны учитывать последствия динамического выделения памяти и дважды обдумать использование функции malloc или оператора new. Легко убедить себя, что вы не делаете так уж много аллокаций, а значит большого значения это не имеет, но такой тип мышления распространяется лавиной по всей команде и приводит к медленной смерти. Фрагментация и потери в производительности, связанные с использование динамической памяти, не будучи пресеченными в зародыше, могут иметь катастрофические трудноразрешаемые последствия в вашем дальнейшем цикле разработки. Проекты, где управление и распределение памяти не продумано надлежащим образом, часто страдают от случайных сбоев после длительной сессии из-за нехватки памяти и стоят сотни часов работы программистов, пытающихся освободить память и реорганизовать ее выделение.

4 Изменения

Добавились файлы TAllocatorBlock.cpp TallocatorBlock.h, изменились файлы TbinaryTreeItem.cpp TbiaryTreeItem.h

5 Выводы

В результате выполнения был разработан аллокатор памяти для массива, оптимизирован вызов операции malloc, переопределены операторы new-delete. Для хранения свободных блоков использовался массив памяти.