Лабораторная работа №6

Целью лабораторной работы является:

- Закрепление навыков по работе с памятью в C++.
- Создание аллокаторов памяти для динамических структур данных

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: октагон, квадрат, треугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: связный список.

1 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 - уменьшение количества операций по выделению памяти, в следствие чего, программа работает эффективнее

2 Исходный код

Описание классов фигур и класса-контейнера остается неизменным.

TAllocationBlock.cpp	
TAllocationBlock(int32t size, int32t count);	Конструктор класса
void *Allocate();	Выделение памяти
void Deallocate(void *ptr);	Освобождение памяти
bool Empty();	Проверка, пуст ли аллокатор
int32t Size();	Получение количества выделенных блоков
virtual ~TAllocationBlock();	Деконструктор класса

```
1
2  class TAllocationBlock
3  {
4  public:
5      TAllocationBlock(int32_t size, int32_t count);
6      void *Allocate();
7      void Deallocate(void *ptr);
8      bool Empty();
9      int32_t Size();
10
11     virtual ~TAllocationBlock();
12
13 private:
14     Byte *_used_blocks;
15     TNList<void *>_free_blocks;
16 };
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab6$ ./run
TAllocationBlock: Memory init
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
1
Enter side A: 2
Enter side B: 3
Enter side C: 4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
3
Enter side: 10
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
2
Enter side: 17
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
```

```
5) Print list
6) Print list using iterator
0) Exit
5
idx: 0   Side A = 2,side B = 3,side C = 4,type: Triangle

idx: 1   Side = 10,type: Octagon

idx: 2   Side = 17,type: Foursquare


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
0) Exit
5
idx: 0   Side A = 2,side B = 3,side C = 4,type: Triangle

idx: 1   Side = 10,type: Octagon


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Print list using iterator
```

```
0) Exit
0
TAllocationBlock: Memory freed
```

4 Выводы

В данной лабораторной работе был реализован аллокатор памяти, свободные блоки которого хранятся в контейнере второго уровня – списке. Аллокатор оптимизирует работу с памятью.