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: трапеция, ромб, пятиугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: стек.

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 в первом смысле – реализация динамической памяти. В языке C 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.

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      TStack<void *>_free_blocks;
16  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab6$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
1
Enter bigger base: 10
Enter smaller base: 10
Enter left side: 10
Enter right side: 10
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
3
Enter side: 10
Enter index = 0
Choose an operation:
```

```
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
2
Enter side: 10
Enter smaller angle: 10
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
5
idx: 0   Sides =  10,type: pentagon

idx: 1   Smaller base = 10,bigger base = 10,left side = 10,right side = 10,type:
trapeze

idx: 2   Side = 10,smaller_angle = 10,type: rhomb


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
```

```
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
6
Side = 10,smaller_angle = 10,type: rhomb
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
1
Enter bigger base: 10
Enter smaller base: 10
Enter left side: 10
Enter right side: 10
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
3
Enter side: 10
```

```
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
5
idx: 0   Smaller base = 10,bigger base = 10,left side = 10,right side = 10,type:
trapeze

idx: 1   Side = 10,smaller_angle = 10,type: rhomb

idx: 2   Sides =  10,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
0
```