Тимохина Софья Константиновна, БПИ207

Пояснительная записка к домашнему заданию 2.

Архитектура статически типизированного универсального языка программирования, ориентированная на объектно-ориентированный подход

Описание задания

Вариант 267, задание 1, функция 20.
Программа должна содержать следующие структуры:

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)
Плоская геометрическая фигура, размещаемые в координатной сетке.	1. Круг (целочисленные координата центра окружности, радиус) 2. Прямоугольник (целочисленные координаты левого верхнего и правого нижнего углов) 3. Треугольник (целочисленные координаты трех углов)

Общая для всех альтернатив переменная - цвет фигуры, перечислимый тип = {красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый}.
Общая для всех альтернатив функция: вычисление площади фигуры (действительное число).
Дополнительная функция: удаление из контейнера тех элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, то есть функция вычисления площади, больше, чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы передвинуть к началу контейнера с сохранением порядка.

Также нужно:

1. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных. Количество тестовых наборов данных – не менее пяти. Число уникальных элементов в тестовых наборах должно варьироваться от нуля до 10000. При необходимости программа должна правильно обрабатывать

переполнение по данным. Тестовые наборы до 20 элементов должны вводиться из заранее подготовленных тестовых файлов. Тестовые данные с большим числом элементов должны порождаться программно с использованием генераторов случайных наборов данных. Данные формируемые генератом случайных наборов должны поддерживать допустимые значения. Управление вводом данных задается из командной строки.

2. Описать структуру используемой ВС с наложением на нее обобщённой схемы разработанной программы.
3. Зафиксировать для отчета основные характеристики программы, такие как: число интерфейсных модулей (заголовочных файлов) и модулей реализации (фалов с определением программных объектов), общий размер исходных текстов, полученный размер исполняемого кода (если он формируется), время выполнения программы для различных тестовых наборов данных.

Структурная схема архитектуры ВС

Таблица типов

Название	Память (байты)

Container: class {
len: int
storage: Shape*
}
	80004
	4[0]
	80000[4]

Shape: class {
color Color::color
}
	4
	4[0]

Triangle: Shape {
p1: Point
p2: Point
p3: Point
}
	28
	8[0]
	8[8]
	8[16]

Rectangle: Shape {
p1: Point
p2: Point
}
	20
	8[0]
	8[8]

Circle: Shape {
center: Point
radius: int
}
	16
	8[0]
	4[8]

Color: class {
enum color
}
	4
	4

Point: class {
x: int
y: int
}
	8
	4[0]
	4[4]

Random: class {
first: int
last: int
}
	8
	4[0]
	4[4]

Описание работы функции main (main.cpp)

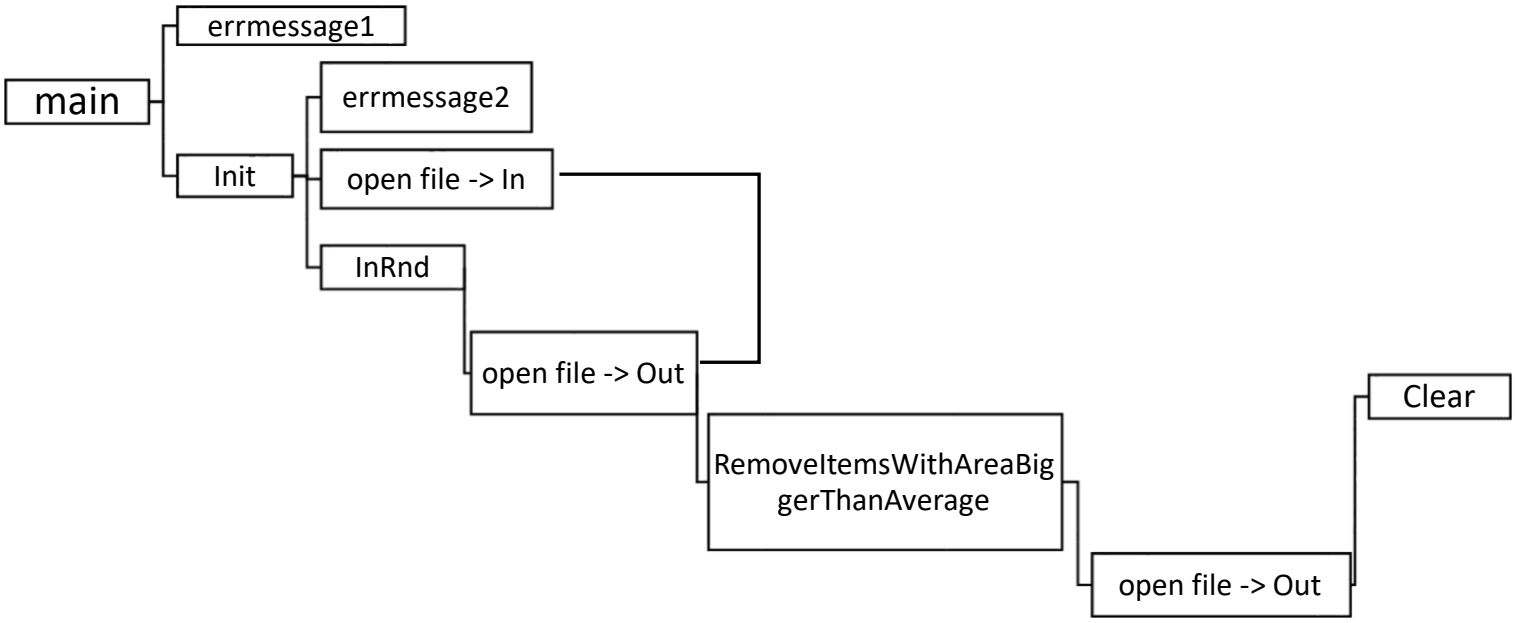
Память программы
int main(int argc, char* argv[])

Stack
InRnd / In
Out
RemoveItemsWithAreaBiggerThanAverage
Out
Clear (вызывается во время delete)

Память данных	Размер (байты)
int argc	4[0]
char* argv[]	8[4]
Container* c	8[12]
clock_t startTime	8[20]
int size	4[28]
ifstream ifst	
ofstream ofst1	
ofstream ofst2	

Heap	
Ввод из файла	Генерация
main	
-f	-n
in.txt	int size
out1.txt	
out2.txtx	

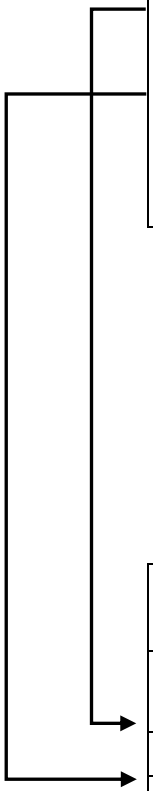
Блок-схема возможного стека в результате работы функции main (main.cpp)



Описание работы функции RemoveItemsWithAreaBiggerThanAverage

Функция

```
void Container::RemoveItemsWithAreaBiggerThanAverage() {
    int currentLen = 0;
    double averageArea = AverageArea();
    for (int i = 0; i < len; i ++) {
        if (storage[i]->Area() <= averageArea) {
            storage[currentLen] = storage[i];
            ++currentLen;
        }
    }
    len = currentLen;
}
```



Stack
RemoveItemsWithAreaBiggerThanAverage
AverageArea
Area

Память данных	Размер (байты)
double averageArea	8[0]
int currentLen	4[8]
int i	4[12]

Описание работы функции AverageArea

Функция

double Container::AverageArea() {
  double sum = 0;
  for (int i = 0; i < len; i ++) {
  sum += storage[i]->Area();
  }
  return sum / len;
}

Stack
Area

Память данных	Размер (байты)
double sum	8[8]
int i	4[16]

Описание работы функции StaticInRnd (shape.cpp)

Функция

```
Shape *Shape::StaticInRnd() {
    int k = Random::Get(1, 3);
    Shape* sp = nullptr;
    switch(k) {
        case 1:
            sp = new Rectangle;
            break;
        case 2:
            sp = new Circle;
            break;
        case 3:
            sp = new Triangle;
            break;
        default:
            return nullptr;
    }
    sp->InRnd();
    return sp;
}
```

Heap
shape [0]

Stack
InRnd

Память данных	Размер (байты)
Shape* sp	8[0]
int k	8[8]

Основные характеристики программы

Число заголовочных файлов: 9
Число исходных файлов: 7
Общий размер кода: 560 строк (86 КБ)
Время выполнения программы для различных тестов:

Номер теста	Время (с)
1	0.000611
2	0.000897
3	0.0009
4	0.000808
5	0.00086

Время выполнения программы для автогенерируемых тестов:

Количестов элементов	Время (с)
50	0.000917
100	0.001182
2000	0.010827
5000	0.020645
10000	0.037599

Сравнительный анализ

В ходе предыдущего домашнего задания была проработана архитектура статически типизированного универсального языка программирования, ориентированная на процедурный подход (далее процедурный подход), в данном задании была проработана архитектура архитектура статически типизированного универсального языка программирования, ориентированная на объектно-ориентированный подход (далее ООП).

В процедурном программировании задача разбирается на несколько задач, то есть на процедуры, которые могут быть вызваны как последовательно, во время другой процедуры, так и отдельно. Программа разбивается на набор переменных и методов (подпрограмм, процедур).
В ООП каждая задача решается в терминах структур данных (объектов), их методов. Каждый объект содержит данные, а также методы для работы с ними. То есть программа разбивается на объекты, которые имеют набор данных и методов.

Итак, в процедурном подходе задачи разбиваются на набор переменных и методов, а в ООП задачи разбиваются на объекты, которые владеют переменными и методами для работы с ними.

Как мне показалось, ООП чаще более удобен в плане разработки, потому что на основе объекта можно создать несколько других объектов с одинаковыми методами, но разной реализацией, например, с помощью наследования, а также из-за инкапсюляции, позволяющей закрыть доступ к некоторым элементам объекта. В процедурном подходе нельзя получить несколько копий одного объекта, а также расширить процедуру, сделать несколько вариаций.