МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт радиоэлектроники и информационных технологий

Кафедра «Информационные радиосистемы»

Контрольная работа по дисциплине «Системное программирование»

Выполнил:

Студент гр. 23-РЗ _____

Проверил:

к.т.н., доцент кафедры ИРС Сидоров С.Б.

Нижний Новгород
2024

Содержание

1. Постановка задачи

Применяя парадигму абстрактных типов данных и инкрементную модель разработки, создать программную систему для решения поставленной задачи. Все исходные данные должны вводиться со стандартного устройства ввода (с клавиатуры), то есть запрашиваться у пользователя. Результаты обработки должны быть выданы на стандартное устройство вывода (дисплей). Кроме окончательного варианта программной системы должны быть предоставлены и её промежуточные версии.

Вариант 27.

Дана последовательность точек на декартовой плоскости $\{a_1, a_2, \ldots, a_n\}$, где $a_i=(x_i, y_i)$. Найти пару наиболее удалённых друг от друга точек, то есть указать их координаты и расположение в исходной последовательности.

2. Архитектура программной системы

В качестве решения для хранения переменных приложения было решено использовать структуру АТД. Что предполагает собой некий тип данных, который будет передаваться в качестве аргумента функции, и будет содержать все переменные данного приложения. Рассмотрим используемую реализацию этого подхода.

```
typedef struct Application{

   int n;
   std::vector<std::pair<int, int>> array;
   std::pair<std::pair<int,int>, std::pair<int,int>> distanced;

}Application;
```

Первое поле - число n, что является количеством точек на рассматриваемой декартовой плоскости.

Второе поле — вектор целочисленных пар. Массив точек рассматриваемой декартовой плоскости.

Третье — двухуровневая пара целочисленных. Хранение наиболее удалённых точек. Первая и вторая точки, и их X/Y координаты.

В качестве исполняемых функций используются набор трёх функций. Ввод, обработка и вывод. И предшествующая им основная запускающая функция run.

```
int app_run(Application& app);
bool app_start(Application& app);
bool app_process(Application &app);
bool app_end(Application& app);
```

Для работы с массивом координат вводится отдельный модуль cartesian, который отвечает за добавление точек в массив, запуск функции поиска наиболее удалённых точек, и функции расчёта расстояния между двумя точками. Познакомимся с структурой этого модуля поближе.

bool cartesian_add(int x, int y, std::vector<std::pair<int, int>> &array);

bool cartesian_find(std::vector<std::pair<int, int>>& array, std::pair<std::pair<int,int>, std::pair<int,int>>& distanced);

double cartesian_find_distance(std::pair<int, int>& pair_one, std::pair<int, int>& pair_two);

Первая функция выполняет задачу добавления точки в массив, принимая на вход два целочисленных элемента и ссылку на массив.

Вторая функция выполняет поиск наиболее удалённых друг от друга точек, принимая на вход ссылку на массив точек, и пару переменной для хранения наиболее удалённых точек.

Третья — функция вычисления квадрата расстояния между двумя точками, возвращает число с двумя знаками после запятой. Принимая на вход две пары, первой и второй точек.

3. Алгоритм обработки

Программа начинает свою работы с вывода информации о назначении, собственно, программы. — Cartesian dot finder!

Далее мы объявляем переменную app по типу нашего АТД Application. После объявления мы вызывает функцию run, передавая ей нашу переменную по типу АТД.

Переходя в функцию выполнения приложения, пользователю будет предложено ввести число точек т. е. n. После этого, по количеству точек, у пользователя будут «спрашиваться» координаты X/Y для каждой точки, добавляя их в массив.

Как только весь необходимый набор точек будет получен, будет вызвана функция поиска наиболее удалённой пары точек cartesian_find. Данная функция сравнивает все точки между собой, сохраняя наибольшее из полученных расстояний в пару distanced.

После выполнения функции поиска, на стандартное устройство вывода будет предоставлена информация о подходящих точках. В противном случае, например, если пользователь указал только одну точку, то выведется сообщение об ошибке.

Приложение 1.

Версия 1.0

Файл main.cpp

```
#include "application.h"

int main (){
   std::cout << "Cartesian dot finder!" << std::endl;
   Application app;
   int ret = app_run(app);
   return ret;
}
```

Файл application.h

```
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <iostream>
#include "cartesian.h"

typedef struct Application{

   int n;
   std::vector<std::pair<int, int>> array;
   std::pair<std::pair<int,int>, std::pair<int,int>> distanced;
   //distanced.first - dot 1
   //distanced.second - dot 2
   //distanced.first.first/second - x/y of a 1nd dot
   //distanced.second.first/second - x/y of a 2nd dot

}Application;

int app_run(Application& app);
bool app_start(Application& app);
bool app_process(Application &app);
bool app_end(Application& app);
```

```
#endif //NNTU_APPLICATION_H
```

Файл application.cpp

```
#include "application.h"

int app_run(Application& app){
   app_start(app);
   if (app_process(app)){
      app_end(app);
      return 0;
   }
   else{
      std::cout << "An error has occured." << std::endl;
      return 1;
   }
}

bool app_start(Application& app){
   int temp_x = 0, temp_y = 0;
   std::cout << "START START" << std::endl;
   std::cout << "Input a n number:" << std::endl;
   std::cin >> app.n;
   for (int i = 0; i < app.n; ++i){
      std::cout << "[" << i+1 << "/" << app.n << "]" << " set is being assigned:" <<
std::endl;
      std::cout << "Input a x:" << std::endl;
      std::cin >> temp_x;
      std::cout << "Input an y:" << std::endl;
      std::cin >> temp_y;
      cartesian_add(temp_x, temp_y, app.array);
   }
   return true;
}

bool app_process(Application& app){
   std::cout << "PROCESSING" << std::endl;
   if (app.array.size() > 1){
      cartesian_find(app.array, app.distanced);
```

```
        return true;
    }
    std::cout << "Insufficient data. ABORTING..." << std::endl;
    return false;
}

bool app_end(Application& app){
    std::cout << "ENDING" << std::endl;
    std::cout << "The farthest pair of points are: ("
        << app.distanced.first.first << ", " << app.distanced.first.second << ") and ("
        << app.distanced.second.first << ", " << app.distanced.second.second << ")"
<< std::endl;

    for (int i = 0; i < app.array.size(); ++i) {
        if (app.array[i] == app.distanced.first) {
            std::cout << "First point index in the sequence: " << i+1 << std::endl;
        }
        if (app.array[i] == app.distanced.second) {
            std::cout << "Second point index in the sequence: " << i+1 << std::endl;
        }
    }
    return true;
}
```

Файл cartesian.h

```
#ifndef CARTESIAN_H
#define CARTESIAN_H

#include <vector>
#include <utility>
#include <math.h>

bool cartesian_add(int x, int y, std::vector<std::pair<int, int>> &array);
bool cartesian_find(std::vector<std::pair<int, int>>& array,
std::pair<std::pair<int,int>, std::pair<int,int>>& distanced);
double cartesian_find_distance(std::pair<int, int>& pair_one, std::pair<int, int>&
pair_two);
```

```
#endif //CARTESIAN_H
```

Файл cartesian.cpp

```
#include "cartesian.h"

bool cartesian_add(int x, int y, std::vector<std::pair<int, int>> &array){
   std::pair<int, int> temp_pair = std::make_pair(x, y);
   array.push_back(temp_pair);
   return true;
}

bool cartesian_find(std::vector<std::pair<int, int>>& array,
std::pair<std::pair<int,int>, std::pair<int,int>>& distanced){
   double distance_max = 0;
   for (int i = 0; i < array.size(); ++i){
      for (int j = i+1; j < array.size(); ++j){
         double distance = cartesian_find_distance(array[i], array[j]);
         if (distance > distance_max){
            distance_max = distance;
            distanced = {array[i], array[j]};
         }
      }
   }
   return true;
}

double cartesian_find_distance(std::pair<int, int>& pair_one, std::pair<int, int>&
pair_two){
   return pow(pair_two.first - pair_one.first, 2) + pow(pair_two.second -
pair_one.second, 2);
}
```

Версия 0.6

Файл main.cpp

```
#include "application.h"

int main (){
   std::cout << "Cartesian dot finder!" << std::endl;
   Application app;
   int ret = app_run(app);
   return ret;
}
```

Файл application.h

```
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <iostream>
#include "cartesian.h"

typedef struct Application{

   int n;
   std::vector<std::pair<int, int>> array;
   std::pair<std::pair<int,int>, std::pair<int,int>> distanced;
   //distanced.first - dot 1
   //distanced.second - dot 2
   //distanced.first.first/second - x/y of a 1nd dot
   //distanced.second.first/second - x/y of a 2nd dot

}Application;

int app_run(Application& app);
bool app_start(Application& app);
bool app_process(Application &app);
bool app_end(Application& app);

#endif //NNTU_APPLICATION_H
```

Файл application.cpp

```
#include "application.h"

int app_run(Application& app){
}

bool app_start(Application& app){
   std::cout << "START START" << std::endl;
   return true;
}

bool app_process(Application& app){
   std::cout << "PROCESSING" << std::endl;
   return false;
}

bool app_end(Application& app){
   std::cout << "ENDING" << std::endl;
   return true;
}
```

Файл cartesian.h

```
#ifndef CARTESIAN_H

#define CARTESIAN_H

#include <vector>
#include <utility>
#include <math.h>

bool cartesian_add(int x, int y, std::vector<std::pair<int, int>> &array);
bool cartesian_find(std::vector<std::pair<int, int>>& array,
std::pair<std::pair<int,int>, std::pair<int,int>>& distanced);
double cartesian_find_distance(std::pair<int, int>& pair_one, std::pair<int, int>&
pair_two);

#endif //CARTESIAN_H
```

Файл cartesian.cpp

```
#include "cartesian.h"

bool cartesian_add(int x, int y, std::vector<std::pair<int, int>> &array){
   return true;
}

bool cartesian_find(std::vector<std::pair<int, int>>& array,
   return true;
}
```

Версия 0.1

Файл main.cpp

```
#include <iostream>

int main (){
   std::cout << "Cartesian dot finder!" << std::endl;
   return 0;
}
```