



Домашна работа 2

курс Увод в програмирането
за специалност Информатика
зимен семестър 2017/18 г.

Задача 2А: Магически квадрати

Нека е дадена квадратна таблица съдържаща числа. Ще казваме, че тя е магически квадрат, ако сбора на числата във всички нейни редове, всички колони и по двата диагонала, е един и същ. Например дадената по-долу таблица е магически квадрат:

4	14	15	1
9	7	6	12
5	11	10	8
16	2	3	13

Напишете програма, която въвежда от потребителя число N -- размерът на таблицата, следвано от N^2 цели числа -- нейните елементи. Считаме, че $N \leq 10$. Елементите на таблицата се въвеждат ред по ред, започвайки от най-горния. Не е задължително всички елементи в таблицата да са различни, например може всички числа в нея да са равни на 1. Програмата трябва да изведе на екрана текст "True", ако таблицата е магически квадрат и "False" в противен случай.

По-долу е дадена примерна таблица с размери 3×3 и как би могла да работи програмата за нея:

4	9	2
3	5	7
8	1	6

```
Enter N: 3
Enter table elements: 4 9 2 3 5 7 8 1 6
True
```

Задача 2Б: Компютърна игра

Трябва да се разработи нова компютърна игра за мобилни телефони. На вас е възложено да напишете една част от нея, която реализира основната логика. По-долу е дадено описание на играта и на това, което трябва да се реши като проблем.

На всеки рунд, играта генерира стая, в която са разположени огледала. Потребителят избира начална точка, от която да тръгне лазерен лъч, който трябва да уцели мишена с формата на топка, която също се намира в стаята.

Лъчът може да се отразява от огледалата или от стените на стаята. Броят на отраженията е ограничен, като ограничението може да се променя за различните рундове. Например, ако на даден рунд може да се направят най-много пет отражения, потребителят ще го спечели, ако лъчът успее да уцели топката след 5 или по-малко отражения и ще го загуби, ако лъчът или не уцели топката или го направи след повече от 5 отражения.

Играта е двуизмерна и потребителят ще вижда стаята като гледа отгоре ѝ. За него тя ще изглежда като един правоъгълник, в който мишената е показана като кръг, а огледалата -- като линии. Така всички отражения ще бъдат в една плоскост, успоредна на пода на стаята.

Вашата задача ще бъде да напишете тази част от играта, която реализира логиката на отраженията и определя дали потребителят е успял да уцели мишената или не.
По-конкретно:

Ще получите библиотека, която трябва да включите в решението си. В библиотеката ще намерите две функции. Едната от тях се използва, за да ви подаде входна информация за един рунд на играта:

- размерите на стаята;
- местоположението и размерите на мишената;
- началната точка, от която тръгва лазерният лъч и накъде е насочен той;
- броят на допустимите отражения;
- броят на огледалата в стаята;
- техните координати;

Втората функция се използва, за да може вашата част от програмата да отговори:

- в кои точки лъчът се е отразил от някоя повърхност (огледало или стена) и

- дали е успял да уцели мишената за по-малко от заложения брой отражения.

Вашето решение трябва да използва информацията от първата функция, за да изчисли дали лъчът уцелва мишената и с помощта на втората, да изчертае лъча. Изчертаването се прави вътрешно от библиотеката, която ще получите, заедно със задачата.

Двете функции имат следните спецификации:

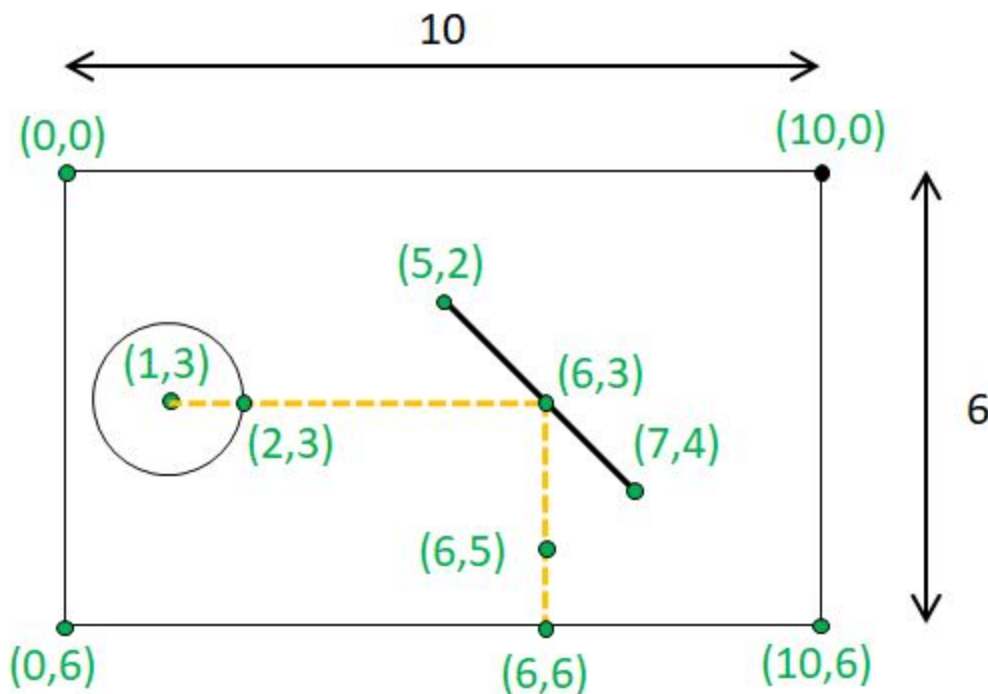
```
void GetInput(double GameData[])
```

На `GetInput` трябва да подадете масив с място за поне 51 елемента. В него функцията ще попълни входните данни. Те ще бъдат организирани както следва:

Индекс в масива	Информация
0	Широчина на стаята
1	Височина на стаята
2	X-координата на центъра на мишената
3	Y-координата на центъра на мишената
4	Радиус на мишената.
5	X-координата на началната точка на лъча
6	Y-координата на началната точка на лъча
7	X-координата на втора точка, която определя посоката на лъча
8	Y-координата на втора точка, която определя посоката на лъча
9	Максимален брой отражения R. Ще бъде цяло число между 0 и 500. Тъй като масивът е от тип <code>double</code> , трябва да преобразувате стойността до <code>int</code> , преди да я използвате.
10	Брой на огледалата M. Ще бъде цяло число между 1 и 10. Тъй като масивът е от тип <code>double</code> , трябва да преобразувате стойността до <code>int</code> , преди да я използвате.
11	Координати на огледалата
...	
$11 + 4 * M - 1$	

Координатите на огледалата се задават с две точки -- началото и края на всяко огледало. Считаме, че огледалото включва и двете точки, т.е. ако лъчът уцели точно някоя от двете точки, той трябва да се отрази. Няма ограничение за това как точно ще бъдат подредени двете точки. Например няма изискване първо да е тази, която е по-наляво или по-надясно в стаята. Също така, възможно е двете точки да съвпадат (да са с еднакви координати). Това е най-малкото възможно огледало в играта (с размер точно една точка). При това положение приемете, че ако лъчът удари това огледало, то той пада перпендикулярно на него. Ако лъчът се удари в огледало под ъгъл 180 градуса, то приемаме че не се отразява (плъзга се по огледалото).

По-долу е дадена схема на възможен рунд в играта, а под нея е показано как ще се съхрани тази информация в масива. На схемата е дадена стая с размери 10 на 6. Мишената е с център точката (1,3) и радиус 1. В стаята има само едно огледало, между точките (5,2) и (7,4). Лъчът е начало в точката (6,6). Посоката му се задава от точката (6,5). Той се отразява само веднъж от огледалото и уцелва мишената в точката (2,3).



Тази информация ще се представи в масива с входна информация по следния начин (считаме, че за това ниво на играта, максималният брой допустими отражения е 50):

10	6	1	3	1	6	6	6	5	50	1	7	4	5	2
----	---	---	---	---	---	---	---	---	----	---	---	---	---	---

За да се избегне объркване, обърнете внимание на няколко неща:

- Въпреки, че на схемата, началната точка е върху една от стените, това не е задължително - може например да бъде в средата на стаята, вътре в мишената, върху някое огледало и т.н.
- Отражението на лъча от някоя повърхност (стена или огледало) може да е под всякакъв ъгъл. Не е задължително да е под 90 градуса, както е в примера;
- Координатната система, която използваме е с начало в горния-ляв ъгъл на стаята. Стените са успоредни на координатните оси. Координатната система е ориентирана така, че оста Y нараства надолу и намалява нагоре -- аналогично на стандартното за компютърния екран.

Прототипът на втората функция, с която ще работите е даден по-долу.

```
void SetRaySegment(double fromX, double fromY, double toX, double toY)
```

Тя се използва, за да подадете на играта един по един сегментите на лъча, които се намират между точките на отражение. Например на схемата по-горе има два такива сегмента - от (6,6) до (6,3) и от (6,3) до (2,3).

Колко пъти ще извикате функцията, зависи от това колко на брой отражения е направил лъча. При първото извикване ще ѝ подадете сегмента, който започва от началната точка и завършва при първото отражение. След това ще подадете втория сегмент и т.н. Например за дадената по-горе схема трябва да се направят две последователни извиквания:

1. `SetRaySegment(6, 6, 6, 3)`
2. `SetRaySegment(6, 3, 2, 3)`

Ако играчът успее да уцели мишената, последният сегмент ще завършва в точка намираща се върху окръжността на мишената. Ако не успее, последната точка ще бъде върху някое огледало или стена в стаята.

За тази задача в Moodle ще намерите архив с готов проект, в който можете да започнете да разработвате своето решение. Проектите са два - един за Visual Studio и един за Code::Blocks.

В директорията на проекта ще намерите и примерен конфигурационен файл, в който се пази информацията за един рунд на играта. Можете да промените този файл и да впишете в него каквито искате данни, за да тествате решението си с различни стаи и местоположения на огледала.

Когато стартирате приложението и извикате функцията `GetInput()`, библиотеката ще зареди конфигурационния файл и ще отвори прозорец, който отговаря на размерите на

стоята. В него ще са изчертани всички огледала. С две щраквания на мишката можете да изберете началната точка на лъча и неговата посока. След това функцията ще приключи своята работа и ще върне на вашето приложение входните данни. С помощта на функцията `SetRaySegment`, вие ще можете да изчертавате отсечки в прозореца.

Задача 2В: Изрязване на изображение

Днес е ден за графика. След работата с лазерите ще пореботим със скенери. Вашата програма ще трябва да реализира подрязване (crop) на изображение, получено от скенер. Изображението е с размери $W \times H$ (W - width, широчина, H - height, височина). Всеки пиксел в него се представя като 32 битово число, байтовете на което (от старши към младши) описват съответно стойностите в [ARGB](#) каналите. За улеснение приемаме, че всички пиксели са плътни (т.е. имат максимална стойност `0xFF` за alpha канала).

Например цветът получаващ се като `red=0x30, green=0x10, blue = 0x50`, ще се представи като следното число: `0xFF301050`.

За решаването на задачата, ще получите две готови функции, едната от които зарежда изображение от `.bmp` файл и втора, която записва изображение в такъв файл. Прототипите на двете функции са както следва:

```
int LoadBitmap(const char* Path,
               unsigned int ImageData[],
               size_t MaxSize
               unsigned int& Width,
               unsigned int& Height);
```

```
int SaveBitmap(const char* Path,
               unsigned int ImageData[],
               unsigned int Width,
               unsigned int Height);
```

И двете функции връщат число, което указва какъв е резултатът от тяхната работа. В таблицата по-долу е описано какви са възможните стойности, които функциите могат да върнат и какво означават те. Препоръчваме, вместо числовите стойности, да използвате дефинираните за тях мнемонични означения, така както е показано на примера в края на този документ.

Код	Макрос	Значение
0	ALL_OK	Функцията е приключила успешно

1	ERR_CANNOT_OPEN_FILE	Не може да се отвори подаденият от потребителя файл. Възможно е пътят да не е коректен, да нямате права за достъп до файла и т.н.
2	ERR_WRONG_FILE_TYPE	Подаденият на LoadBitmap файл не е валиден 24-битов bitmap файл.
3	ERR_BUFFER_TOO_SMALL	Подаденият на LoadBitmap масив не е достатъчно голям, за да съхрани изображението.
4	ERR_FILE_READ_ERROR	Грешка по време на четенето от файла
5	ERR_CANNOT_ALLOCATE_MEMORY	SaveBitmap не може да задели нужната за работата ѝ памет. Вероятно сте подали много голяма стойност за широчина на изображението.

Параметрите на функциите са както следва:

- `Path` е пътят до файла, с който функцията трябва да работи.
- `ImageData` е масив, който вие подавате на функцията. `LoadBitmap` ще върне в него данните на изображението, което зарежда от `Path`. `SaveBitmap` получава в този масив данните на изображение, което записва в `Path`.
- `MaxSize` е размерът на масива, който подавате. Подаваме този параметър на `LoadBitmap`, за да може функцията да определи дали данните за изображението могат да се поберат в масива. Например ако той съдържа само 10 елемента, няма как в него да се запише картинка съдържаща 20 пиксела.
- `Width` е широчината на изображението. `LoadBitmap` връща в този параметър широчината на изображението, което е заредила в `ImageData`. `SaveBitmap` използва този параметър, за да знае колко голямо е изображението, което ще запише във файла.
- `Height` е височината на изображението. За този параметър важи казаното за `Width` по-горе.

Както може да се види от прототипите на функциите, те работят с двумерно изображение, което е представено в едномерен масив. Считаме, че това представяне е същото, като на двумерните масиви в C/C++ в паметта, т.е. в първите `Width` на брой клетки на масива се съдържа първият ред на изображението, в тези непосредствено след тях -- вторият ред и т.н.

По-долу е дадено примерно изображение с размери 4x3 и как то би се представило в едномерен масив:

Изображение



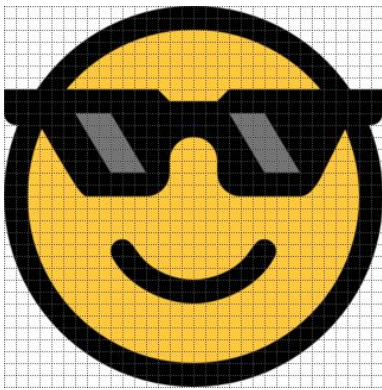
10	20	30	40
50	60	70	80
90	100	110	120

Представяне

10	20	30	40	50	60	70	80	90	100	110	120
----	----	----	----	----	----	----	----	----	-----	-----	-----

Вие трябва да напишете програма, която използва тези функции, за да зареди изображение, да го подреже и след това да запише резултата в нов файл.

За целта програмата ви трябва да въведе от потребителя името на файла, който да се зареди и цяло, 32-битово число, указващо цвета на фона. След това програмата ви трябва да подреже всички редове и колони, по края на изображението, които съдържат само фон. По-конкретно, трябва да подрежете от изображението максимална рамка, така че в първия ред, първата колона, последния ред и последната колона да има поне един пиксел с цвят различен от фона. Накрая програмата трябва да запише резултата в нов файл. По-долу е даден пример за работата на програмата:

Оригинал	Цвят на фона	Резултат
		

Опционално (за допълнителни точки), приемете че фонът на изображението е съставен от пиксели с цвят с определена близост (в проценти) до подадения цвят на фона.

Близостта означаваме с D и тя също трябва да се въвежда от потребителя. Програмата ви трябва да подреже рамка състояща се от всички пиксели намиращи се на разстояние по-малко от D от въведения цвят за фон. За да решите опционалната задача, потърсете подходящи формули за определяне на разстояние и относителна близост на два цвята.

Както и за задачата с лазерите, към формата за предаване на второто домашно в Moodle ще намерите архив съдържащ header файл необходим за работата на функциите за зареждане и записване на данни от/в bitmap файл и .cpp файл, от който можете да започнете решението си.

По-долу е даден кратък фрагмент, който зарежда изображение от файл с име "test.bmp" и записва данните му в масив. Изображението може да се състои от не повече от 100 000 пиксела. След това програмата прави всички пиксели в изображението бели и записва резултата в нов файл, който се казва "test_new.bmp":

```
const size_t MAX_SIZE = 100000; // Можете да промените този размер,
                                // според това с колко големи
                                // изображения искате да работите

unsigned int width, height, image[MAX_SIZE];

int rval;

// В последните три параметъра, функцията връща изображението и
// размерите му
rval = LoadBitmap("test.bmp", image, MAX_SIZE, width, height);

if (rval != ALL_OK)
{
    std::cerr << "Cannot load image data from test.bmp! Error code "
                << rval << "\n";
    return 1;
}

int pixelsCount = width * height;

for(int i = 0; i < pixelsCount; ++i)
    image[i] = 0xFFFFFFFF;

rval = SaveBitmap("test_new.bmp", image, width, height);

if (rval != ALL_OK)
{
```

```
std::cerr << "Cannot save image data to test_new.bmp! Error code  
"  
    << rval << "\n";  
return 2;  
}
```