

Информация

- Асташкина Анна Дмитриевна
- БПИ239
- Вариант 21
- Задача о нелюдимых садовниках. Имеется пустой участок земли (двумерный массив размером $M \times N$) и план сада, разбитого на отдельные квадраты. От 10 до 30 процентов (задается случайно) площади сада заняты прудами или камнями. То есть недоступны для ухода. Эти квадраты располагаются на плане произвольным (случайным) образом. Уход за садом выполняют два садовника, которые не хотят встречаться друг другом (то есть, одновременно появляться в одном и том же квадрате). Первый садовник начинает работу с верхнего левого угла сада и перемещается слева направо, сделав ряд, он спускается вниз и идет в обратном направлении, пропуская обработанные участки. Второй садовник начинает работу с нижнего правого угла сада и перемещается снизу вверх, сделав ряд, он перемещается влево и также идет в обратную сторону. Если садовник видит, что участок сада уже обработан другим садовником или является необработываемым, он идет дальше. Если по пути какой-то участок занят другим садовником, то садовник ожидает когда участок освободится, чтобы пройти дальше на доступный ему необработанный участок. Садовники должны работать одновременно со скоростями, определяемыми как параметры задачи. Прохождение через любой квадрат занимает некоторое время, которое задается константой, меньшей чем времени обработки и принимается за единицу времени. Создать многопоточное приложение, моделирующее работу садовников. Каждый садовник — это отдельный поток.

4-5 баллов

В отчете должен быть приведен сценарий, описывающий одновременное поведение представленных в условии задания сущностей в терминах предметной области. То есть, описан сценарий, задающий ролевое поведение субъектов и объектов задачи (интерпретация условия с большей степенью детализации происходящего), а не то, как это будет реализовано в программе.

В задаче фигурируют следующие объекты:

- сад - состоит из грядок, каждая имеет одну из 3х состояний: пустая, недоступная для обработки, обработанная

- садовник 1 - начинает свою работу из верхнего левого угла, двигается змейкой: вправо до упора, вниз, влево до упора, обработанные участки пропускает, при встрече другого садовника ждет окончания обработки и только потом двигается дальше
- садовник 2 - поведение аналогично садовнику 1, но двигается из правого нижнего угла змейкой вверх, влево, вниз и т. д.

Характеристики:

- садовник - имеет скорость обработки участка (v_1/v_2) и скорость прохождения уже обработанного участка - константа (time) одинаковая у обоих садовников. Эти 2 параметра подаются в программу. Кроме них также для удобства имеют координаты начальных столбца и строки, свой номер (id) для обработки направления и ссылку на обрабатываемый сад
- сад - имеет n и m - кол-во столбцов и строк - задаются как параметры задачи

Взаимодействие:

- выполнение задачи имитируется с помощью параллельных потоков
- каждый поток - это садовник, потоки действуют параллельно, это реализуется с помощью POSIX
- независимо друг от друга с разной скоростью, задающейся как параметр с клавиатуры садовники обрабатывают один общий сад
- каждое действие садовников (обработка участка) выводится на экран с нынешним состоянием сада
- садовники не могут находиться на одном участке одновременно, это реализуется с помощью `pthread_mutex_lock` одного для сада, соответственно логика работы остается корректной
- если садовник видит пустой участок он его обрабатывает - меняется статус участка, поток ждет указанное время обработки
- если садовник переходит на заданный участок, то он его проходит - ждет указанное время прохода

Сценарий:

- садовники начинают на указанных позициях (см объекты)
- обрабатывают участки в соответствии с взаимодействием по своей траектории
- продолжают свою работу пока не обойдут все участки на своей траектории
- завершают работу пройдя весь сад параллельно друг другу
- состояние после каждого шага выводится

Описана модель параллельных вычислений, используемая при разработке многопоточной программы.

- Данная задача решается с помощью многопоточного приложения, основанного на модели параллельных вычислений.
- Основной особенностью является то, что потоки работают независимо друг от друга со своими параметрами, можно сказать параллельно. С помощью использования этой модели происходит разделение задачи обработки участка между двумя потоками, благодаря чему выполнение идет более эффективно.
- Кроме того происходит синхронизация потоков, т.к. оба потока не должны одновременно работать над одним участком, для данной цели используется

мьютекс для сада, который является общим для двух потоков. Таким образом включается логика ожидания, описанная в условии: если 2 потока встречаются для обработки одного участка, то один из них ждет, пока другой обработает данный участок и только потом продолжает движение.

- Эффективность разбиения в данной задаче заключается в том, что потоки идут по разным траекториям с разных концов сада => начинают обработку с территорий, где не могут встретиться.
- Для реализации используется библиотека POSIX (pthread.h), а именно техники создания потока - create, присоединения - join, блокирования - lock, уничтожения - destroy
- сами объекты реализованы с помощью структур, сад - на векторе векторов

Описаны входные данные программы, включающие вариативные диапазоны, возможные при многократных запусках.

Входные параметры (передаются при запуске):

- n - количество строк в саду
- m - количество столбцов в саду
- v1 - скорость обработки участка первым садовником
- v2 - скорость обработки участка вторым садовником
- time - константа прохождения участка

Вариативность:

- при каждом запуске сад создается случайным образом: процент занятых (недоступных для обработки) участков и их расположение

Для используемых генераторов случайных чисел описаны их диапазоны и то, как интерпретируются данные этих генераторов

Корректные диапазоны задаются на основе введенных данных:

```
std::uniform_real_distribution<> distr(10, 30);
```

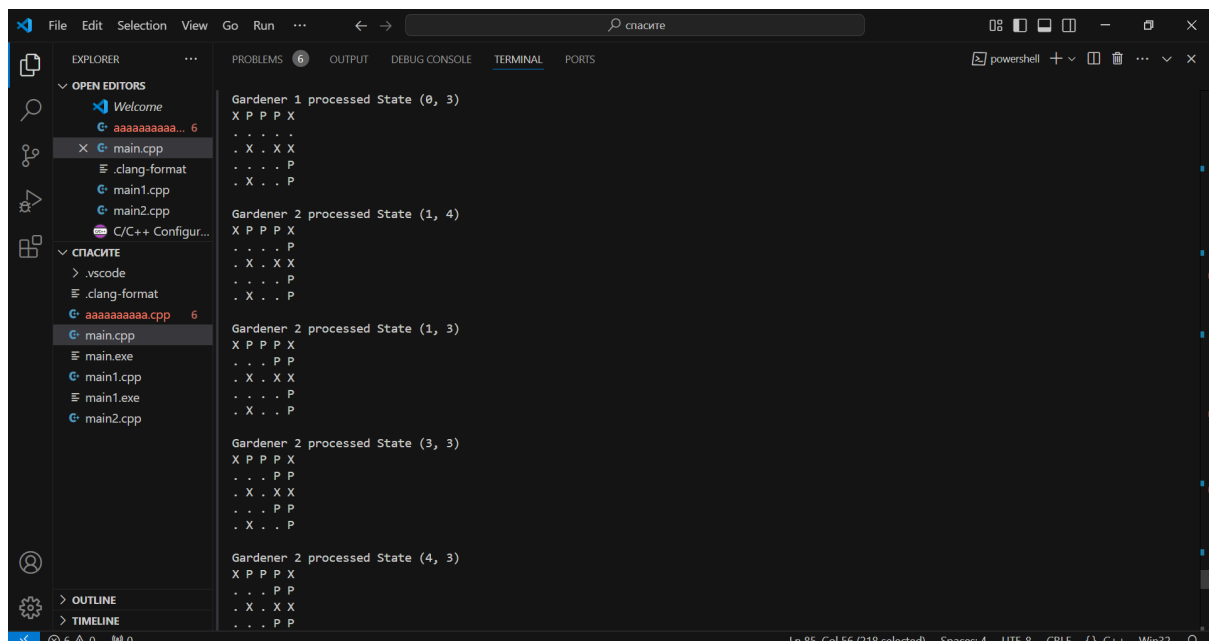
- процент недоступных участков

```
std::uniform_int_distribution<> distrn(0, garden.n - 1);
```

```
std::uniform_int_distribution<> distrm(0, garden.m - 1);
```

- координаты для недоступного участка

Вывод информативный - выводится совершенное действие и текущее состояние поля сада.



Visual Studio Code interface showing the Explorer, Output, and Terminal panels. The Explorer panel shows a project structure with files like main.cpp, main1.cpp, and main2.cpp. The Output panel shows the execution of a program with state updates for 'Gardener 1' and 'Gardener 2'. The Terminal panel shows a PowerShell prompt.

```
Explorer:
- OPEN EDITORS
  - Welcome
  - aaaaaaaa... 6
  - main.cpp
  - .clang-format
  - main1.cpp
  - main2.cpp
  - C/C++ Configur...
- CMAKE
  - .vscode
  - .clang-format
  - aaaaaaaa.cpp 6
  - main.cpp
  - main.exe
  - main1.cpp
  - main1.exe
  - main2.cpp
- OUTLINE
- TIMELINE

Output:
Gardener 2 processed State (4, 2)
X P P P X
. . . P P
. X . X X
. . . P P
. X P P P

Gardener 2 processed State (3, 2)
X P P P X
. . . P P
. X . X X
. . P P P
. X P P P

Gardener 1 processed State (1, 2)
X P P P X
. . P P P
. X . X X
. . P P P
. X P P P

Gardener 1 processed State (1, 1)
X P P P X
. P P P P
. X . X X
. . P P P
. X P P P

Gardener 2 processed State (2, 2)
X P P P X
. P P P P
. X P X X
. . P P P

Terminal:
powershell
```

Visual Studio Code interface showing the Explorer, Output, and Terminal panels. The Explorer panel shows a project structure with files like main.cpp, main1.cpp, and main2.cpp. The Output panel shows the execution of a program with state updates for 'Gardener 1' and 'Gardener 2'. The Terminal panel shows a PowerShell prompt.

```
Explorer:
- OPEN EDITORS
  - Welcome
  - aaaaaaaa... 6
  - main.cpp
  - .clang-format
  - main1.cpp
  - main2.cpp
  - C/C++ Configur...
- CMAKE
  - .vscode
  - .clang-format
  - aaaaaaaa.cpp 6
  - main.cpp
  - main.exe
  - main1.cpp
  - main1.exe
  - main2.cpp
- OUTLINE
- TIMELINE

Output:
Gardener 1 processed State (1, 0)
X P P P X
P P P P P
. X P X X
. . P P P
. X P P P

Gardener 1 processed State (2, 0)
X P P P X
P P P P P
P X P X X
. . P P P
. X P P P

Gardener 2 processed State (3, 1)
X P P P X
P P P P P
P X P X X
. P P P P
. X P P P

Gardener 2 processed State (4, 0)
X P P P X
P P P P P
P X P X X
. P P P P
P X P P P

Gardener 2 processed State (3, 0)
X P P P X
P P P P P
P X P X X
P P P P P

Terminal:
powershell
```

The screenshot shows the Visual Studio Code interface with a project named 'спасите'. The Explorer sidebar on the left shows the file structure under 'OPEN EDITORS' and 'C/C++ CONFIGURATION'. The main editor area displays the 'main.cpp' file, which contains a 4x4 grid of 'P' and 'X' characters. The terminal window at the bottom shows the output of the program, which includes the state of the garden after each step of the gardener's movement. The output is as follows:

```
P X P X X
. . P P P
. X P P P

Gardener 2 processed State (3, 1)
X P P P X
P P P P P
P X P X X
. P P P P
. X P P P

Gardener 2 processed State (4, 0)
X P P P X
P P P P P
P X P X X
. P P P P
P X P P P

Gardener 2 processed State (3, 0)
X P P P X
P P P P P
P X P X X
P P P P P
P X P P P

Final garden state:
X P P P X
P P P P P
P X P X X
P P P P P
P X P P P
```

Каждый шаг нетрудно отследить

В программе присутствуют комментарии, поясняющие выполняемые действия и описание используемых объектов и переменных.

Присутствуют

```
8  main.cpp > main(int, char *[])
9  enum State { EMPTY, NOPROCESS, PROCESSED };
10
11 struct Garden {
12     std::vector<std::vector<State>> grid; // Грядки
13     pthread_mutex_t mutex;
14     int n;
15     int m;
16 };
17
18 struct Gardener {
19     int id;
20     Garden* garden;
21     double speed; // Время обработки одного участка
22     int startRow; // Начальный ряд
23     int startCol; // Начальная колонка
24     double time; // Время прохода занятого или уже обработанного участка
25 };
26
27 void printGarden(const Garden& garden) { // Печать сада
28     for (int i = 0; i < garden.n; ++i) {
29         for (int j = 0; j < garden.m; ++j) {
30             if (garden.grid[i][j] == EMPTY) {
31                 std::cout << ".";
32             } else if (garden.grid[i][j] == NOPROCESS) {
33                 std::cout << "X";
34             } else {
35                 std::cout << "P";
36             }
37         }
```

```
27 void printGarden(const Garden& garden) { // Печать сада
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 void initializeGarden(Garden& garden, int procent) {
43     garden.grid.resize(garden.n, std::vector<State>(garden.m, EMPTY));
44     int occupiedCells = (garden.n * garden.m * procent) / 100; // Количество занятых для свопадения с процентом
45
46     std::random_device rand_dev;
47     std::mt19937 generator(rand_dev());
48     std::uniform_int_distribution<> distrn(0, garden.n - 1);
49     std::uniform_int_distribution<> distrm(0, garden.m - 1);
50
51     for (int i = 0; i < occupiedCells; ++i) { // Генерация координат занятых
52         int indexn = distrn(generator);
53         int indexm = distrm(generator);
54         if (garden.grid[indexn][indexm] == EMPTY) {
55             garden.grid[indexn][indexm] = NOPROCESS;
56         } else {
57             --i; // Повторяем, если клетка уже занята
58         }
59     }
60 }
61
62
63 void* gardenerWork(void* arg) { // Основная функция
64     Gardener* gardener = (Gardener*)arg;
65     int row = gardener->startRow;
66     int col = gardener->startCol;
67
68     while (row >= 0 && row < gardener->garden->n && col >= 0 && col < gardener->garden->m) {
69         pthread_mutex_lock(&gardener->garden->mutex); // Запираем клетку
70     }
```

```
main.cpp > main(int, char *[])
43 void initializeGarden(Garden& garden, int procent) {
62
63 void* gardenerWork(void* arg) { // Основная функция
64     Gardener* gardener = (Gardener*)arg;
65     int row = gardener->startRow;
66     int col = gardener->startCol;
67
68     while (row >= 0 && row < gardener->garden->n && col >= 0 && col < gardener->garden->m) {
69         pthread_mutex_lock(&gardener->garden->mutex); // Запираем клетку
70
71         if (gardener->garden->grid[row][col] == EMPTY) {
72             // Обработка
73             gardener->garden->grid[row][col] = PROCESSED;
74             std::cout << "Gardener " << gardener->id << " processed State (" << row << ", " << col << ") " << std::endl;
75             printGarden(*gardener->garden); // Печать состояния сада после обработки
76             sleep(gardener->speed); // Имитация времени обработки
77         } else {
78             sleep(gardener->time); // Имитация прохождения занятого поля
79         }
80         pthread_mutex_unlock(&gardener->garden->mutex); // Разблокируем
81
82         // Переход к следующему участку
83         if (gardener->id == 1) { // Садовник идет вниз (первый)
84             if (row % 2 == 0 && col < gardener->garden->m - 1) {
85                 col++;
86             } else if (row % 2 == 1 && col > 0) {
87                 col--;
88             } else if (row % 2 == 0) {
89                 row++;
90             } else {
91                 row++;
92             }
93         } else { // Садовник идет вверх (второй)
94             if ((gardener->garden->m - 1 - col) % 2 == 0 && row > 0) {
95                 row--;
96             } else if ((gardener->garden->m - 1 - col) % 2 == 1 && row < gardener->garden->n - 1) {
97                 row++;
98             } else if ((gardener->garden->m - 1 - col) % 2 == 0) {
99                 col--;
100             } else {
101                 col--;
102             }
103         }
104     }
105     return nullptr;
106 }
107
108 int main(int argc, char* argv[]) {
109     if (argc != 6) {
110         std::cout << ":(\n";
111         return 1;
112     }
```

```
main.cpp > main(int, char *[])
63 void* gardenerWork(void* arg) { // Основная функция
64     while (row >= 0 && row < gardener->garden->n && col >= 0 && col < gardener->garden->m) {
65         if (gardener->id == 1) { // Садовник идет вниз (первый)
66             if (row % 2 == 0 && col < gardener->garden->m - 1) {
67                 col++;
68             } else if (row % 2 == 1 && col > 0) {
69                 col--;
70             } else if (row % 2 == 0) {
71                 row++;
72             } else {
73                 row++;
74             }
75         } else { // Садовник идет вверх (второй)
76             if ((gardener->garden->m - 1 - col) % 2 == 0 && row > 0) {
77                 row--;
78             } else if ((gardener->garden->m - 1 - col) % 2 == 1 && row < gardener->garden->n - 1) {
79                 row++;
80             } else if ((gardener->garden->m - 1 - col) % 2 == 0) {
81                 col--;
82             } else {
83                 col--;
84             }
85         }
86     }
87     return nullptr;
88 }
89
90 int main(int argc, char* argv[]) {
91     if (argc != 6) {
92         std::cout << ":(\n";
93         return 1;
94     }
```



```
107
108 int main(int argc, char* argv[]) {
109     if (argc != 6) {
110         std::cout << "(":
111     }
112     return 1;
113 }
114
115 // Основные переменные
116 int n = std::atoi(argv[1]);
117 int m = std::atoi(argv[2]);
118 double v1 = std::stod(argv[3]);
119 double v2 = std::stod(argv[4]);
120 double time = std::stod(argv[5]); // Константа прохождения участка
121
122 std::random_device rand_dev;
123 std::mt19937 generator(rand_dev());
124 std::uniform_real_distribution<> distr(10, 30);
125 double procent = distr(generator); // Генерация процента занятых
126
127 Garden garden; // Создание сада
128 garden.n = n;
129 garden.m = m;
130 pthread_mutex_init(&garden.mutex, nullptr);
131
132 initializeGarden(garden, procent);
133
134 printGarden(garden); // Печать начального состояния сада
135
136 // Создание садовников
137 Gardener gardener1 = {1, &garden, v1, 0, 0, time}; // Первый садовник (идет вниз)
```

```
138 pthread_mutex_init(&garden.mutex, nullptr);
139
140 initializeGarden(garden, procent);
141
142 printGarden(garden); // Печать начального состояния сада
143
144 // Создание садовников
145 Gardener gardener1 = {1, &garden, v1, 0, 0, time}; // Первый садовник (идет вниз)
146 Gardener gardener2 = {2, &garden, v2, n - 1, m - 1, time}; // Второй садовник (идет вверх)
147
148 pthread_t thread1, thread2; // Создание потоков
149 pthread_create(&thread1, nullptr, gardenerWork, &gardener1);
150 pthread_create(&thread2, nullptr, gardenerWork, &gardener2);
151
152 pthread_join(thread1, nullptr);
153 pthread_join(thread2, nullptr);
154
155 std::cout << "Final garden state:\n";
156 printGarden(garden); // Печать конечного состояния сада
157
158 pthread_mutex_destroy(&garden.mutex);
159
160 return 0;
161 }
```

Результаты работы программы представлены в отчете

Пример выше

6-7 баллов

В отчете подробно описан обобщенный алгоритм, используемый при реализации программы исходного словесного сценария. В котором показано, как на программу отображается каждый из субъектов предметной области.

Описан выше, однако повторимся именно с алгоритмом

- считываются данные

```
// Основные переменные
int n = std::atoi(argv[1]);
int m = std::atoi(argv[2]);
double v1 = std::stod(argv[3]);
double v2 = std::stod(argv[4]);
double time = std::stod(argv[5]); // Константа прохождения участка

std::random_device rand_dev;
std::mt19937 generator(rand_dev());
std::uniform_real_distribution<> distr(10, 30);
double procent = distr(generator); // Генерация процента занятых
```

- создаются садовники (потoki) и сад на их основе

```
Garden garden; // Создание сада
garden.n = n;
garden.m = m;
pthread_mutex_init(&garden.mutex, nullptr);

initializeGarden(garden, procent);

printGarden(garden); // Печать начального состояния сада

// Создание садовников
Gardener gardener1 = {1, &garden, v1, 0, 0, time}; // Первый садовник (идет вниз)
Gardener gardener2 = {2, &garden, v2, n - 1, m - 1, time}; // Второй садовник (идет вверх)

pthread_t thread1, thread2; // Созданы потоки
pthread_create(&thread1, nullptr, gardenerWork, &gardener1);
```

```
pthread_create(&thread2, nullptr, gardenerWork, &gardener2);

pthread_join(thread1, nullptr);
pthread_join(thread2, nullptr);
```

- начинают каждый с заданной позиции, идет по своим траекториям

```
Gardener* gardener = (Gardener*)arg;
int row = gardener->startRow;
int col = gardener->startCol;
```

- при прохождении пустого участка - обрабатывают его

```
pthread_mutex_lock(&gardener->garden->mutex); // Запираем клетку
if (gardener->garden->grid[row][col] == EMPTY) {
    // Обработка
    gardener->garden->grid[row][col] = PROCESSED;
    std::cout << "Gardener " << gardener->id << " processed
State (" << row << ", " << col << ")" << std::endl;
    printGarden(*gardener->garden); // Печать состояния сада
    после обработки
    sleep(gardener->speed); // Имитация времени обработки
```

- иначе - проходят участок (или ждут и проходят)

```
} else {
    sleep(gardener->time); // Имитация прохождения занятого
поля
}
pthread_mutex_unlock(&gardener->garden->mutex); //
Разблокируем
```

- логика перехода на следующую клетку:
садовник 1 - если ряд имеет четный индекс, то идет вправо, если нечетный - влево, при конце ряда спускается вниз

```
// Переход к следующему участку
if (gardener->id == 1) { // Садовник идет вниз (первый)
    if (row % 2 == 0 && col < gardener->garden->m - 1) {
        col++;
    } else if (row % 2 == 1 && col > 0) {
        col--;
    } else if (row % 2 == 0) {
        row++;
    } else {
        row++;
    }
}
```

садовник 2 - если колонка имеет четный индекс (с конца) - идет вверх, если нечетный - идет вниз, при конце колонки переходит на колонку влево

```
} else { // Садовник идет вверх (второй)
    if ((gardener->garden->m - 1 - col) % 2 == 0 && row > 0) {
        row--;
    } else if ((gardener->garden->m - 1 - col) % 2 == 1 && row
< gardener->garden->n - 1) {
        row++;
    } else if ((gardener->garden->m - 1 - col) % 2 == 0) {
        col--;
    } else {
        col--;
    }
}
```

- печать происходит на каждом шаге после обработки

```
printGarden(*gardener->garden); // Печать состояния сада после
обработки
```

В программу добавлена генерация случайных данных в допустимых диапазонах.

Добавлена

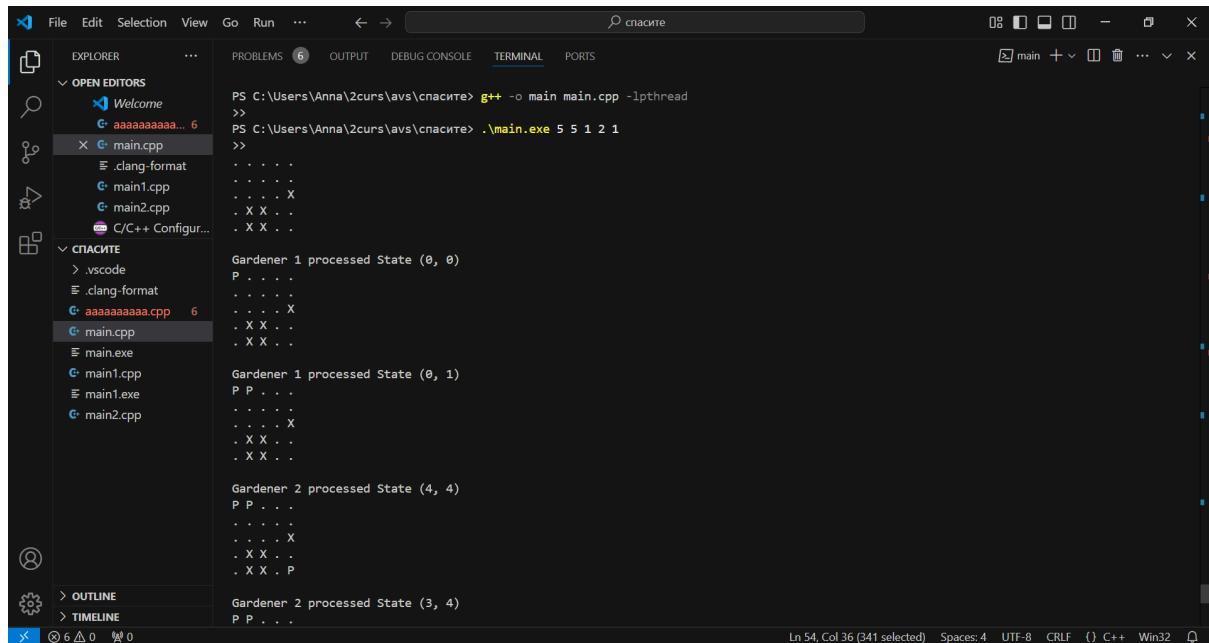
```
std::random_device rand_dev;
std::mt19937 generator(rand_dev());
std::uniform_real_distribution<> distr(10, 30);
double procent = distr(generator); // Генерация процента занятых
```

```
std::random_device rand_dev;
std::mt19937 generator(rand_dev());
std::uniform_int_distribution<> distrn(0, garden.n - 1);
std::uniform_int_distribution<> distrm(0, garden.m - 1);

for (int i = 0; i < occupiedCells; ++i) { // Генерация координат
занятых
    int indexn = distrn(generator);
    int indexm = distrm(generator);
```

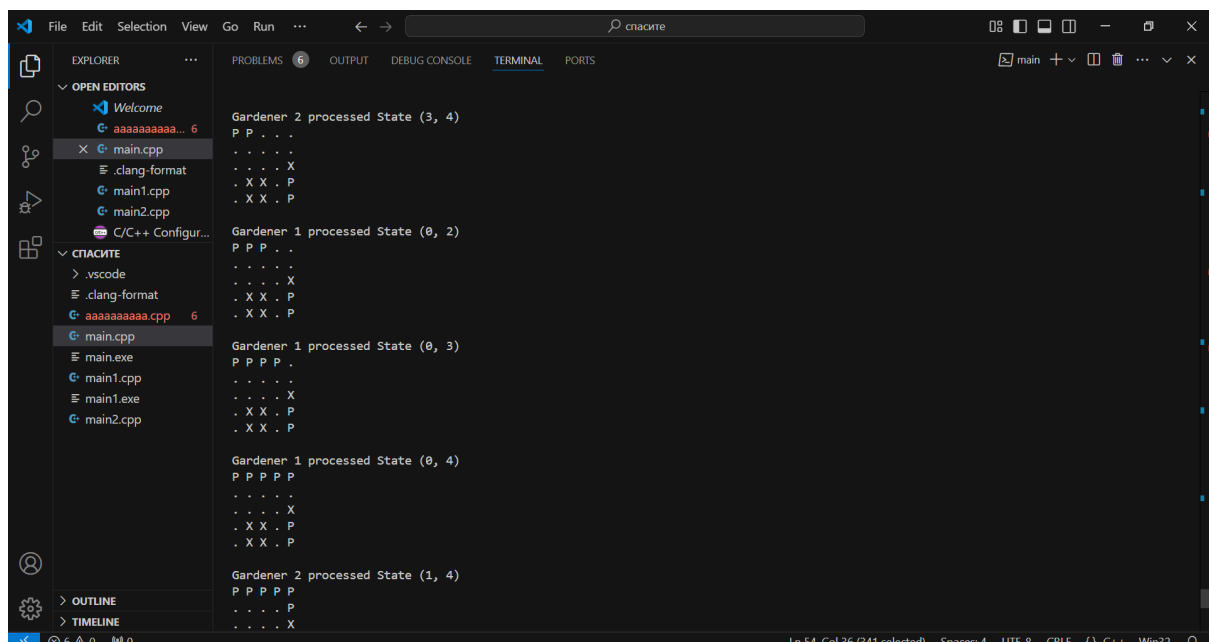
Реализован ввод исходных данных из командной строки
при запуске программы вместо ввода параметров с консоли
во время выполнения программы

Реализован, пример запуска -
g++ -o main main.cpp -lpthread
.\main.exe 5 5 1 4 1



```
File Edit Selection View Go Run ...  
EXPLORED PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
OPEN EDITORS  
Welcome  
G: aaaaaaaa... 6  
X G: main.cpp  
 clang-format  
G: main1.cpp  
G: main2.cpp  
C/C++ Configur...  
СПАСИТЕ  
> .vscode  
 clang-format  
G: aaaaaaaa.cpp 6  
G: main.cpp  
 main.exe  
G: main1.cpp  
 main1.exe  
G: main2.cpp  
OUTLINE  
TIMELINE  
Ln 54, Col 36 (341 selected) Spaces: 4 UTF-8 CRLF () C++ Win32
```

```
PS C:\Users\Anna\2curs\avs\спасите> g++ -o main main.cpp -lpthread  
>>  
PS C:\Users\Anna\2curs\avs\спасите> .\main.exe 5 5 1 2 1  
>>  
. . . .  
. . . .  
. . . . X  
. X X .  
. X X .  
Gardener 1 processed State (0, 0)  
P . . . .  
. . . .  
. . . . X  
. X X .  
. X X .  
Gardener 1 processed State (0, 1)  
P P . . .  
. . . .  
. . . . X  
. X X .  
. X X .  
Gardener 2 processed State (4, 4)  
P P . . .  
. . . .  
. . . . X  
. X X .  
. X X . P  
Gardener 2 processed State (3, 4)  
P P . . .
```



```
File Edit Selection View Go Run ...  
EXPLORED PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
OPEN EDITORS  
Welcome  
G: aaaaaaaa... 6  
X G: main.cpp  
 clang-format  
G: main1.cpp  
G: main2.cpp  
C/C++ Configur...  
СПАСИТЕ  
> .vscode  
 clang-format  
G: aaaaaaaa.cpp 6  
G: main.cpp  
 main.exe  
G: main1.cpp  
 main1.exe  
G: main2.cpp  
OUTLINE  
TIMELINE  
Ln 54, Col 36 (341 selected) Spaces: 4 UTF-8 CRLF () C++ Win32
```

```
Gardener 2 processed State (3, 4)  
P P . . .  
. . . .  
. . . . X  
. X X . P  
. X X . P  
Gardener 1 processed State (0, 2)  
P P P . .  
. . . .  
. . . . X  
. X X . P  
. X X . P  
Gardener 1 processed State (0, 3)  
P P P P .  
. . . .  
. . . . X  
. X X . P  
. X X . P  
Gardener 1 processed State (0, 4)  
P P P P P  
. . . .  
. . . . X  
. X X . P  
. X X . P  
Gardener 2 processed State (1, 4)  
P P P P P  
. . . . P  
. . . . X
```

