

## Практическая работа №5

## Тема: «Стэк и очередь»

**Цель работы:** изучить СД «стэк» и «очередь» научиться их программно реализовывать.

Вариант 18.

Реализовать систему, представленную на рисунке 1.

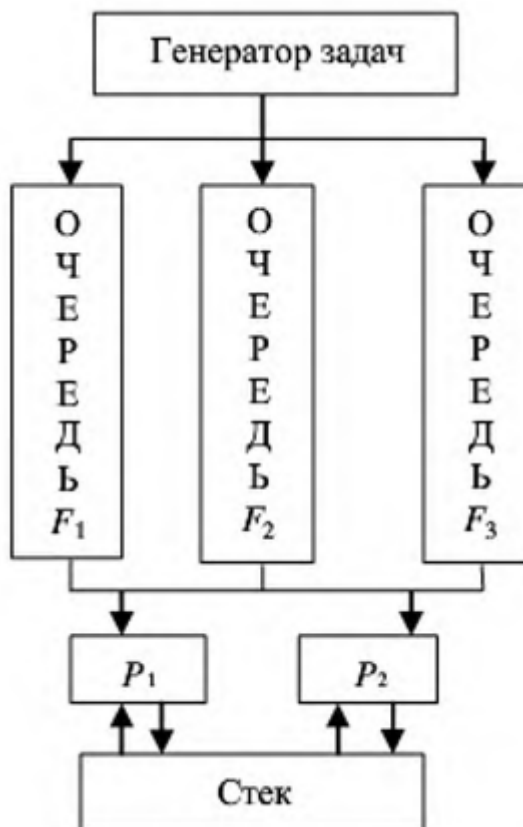


Рисунок 1 - Система для реализации

Задачи из очередей берутся по очереди. Задачи из первой и второй очереди выполняются либо на первом процессоре, либо на втором процессоре, если оба свободны, то на первом. Задачи из очереди третьей выполняются на первом и втором, если оба свободны то на втором.

Реализуем генератор задач, который будет состоять из структуры и класса, который предоставляет доступ к полям (листинг 1).

### Листинг 1. Генератор задач.

```
from dataclasses import dataclass
```

					<div style="text-align: center;"> <h1> <i>АиСД.09.03.02.220000 ПР</i> </h1> </div>					
Изм.	Лист	№ докум.	Подпись	Дата	<div style="text-align: center;"> <h2>                     Практическая работа № — Тема                 </h2> </div>			Лит.	Лист	Листов
Разраб.	Третьяк И.Н.									
Проверил	Береза А.Н.									
Реценз										
Н. Контр.										
Утверд.										

```

from numpy import random as rnd

@dataclass()
class TaskData:
    time: int = None
    type_of_task: int = None

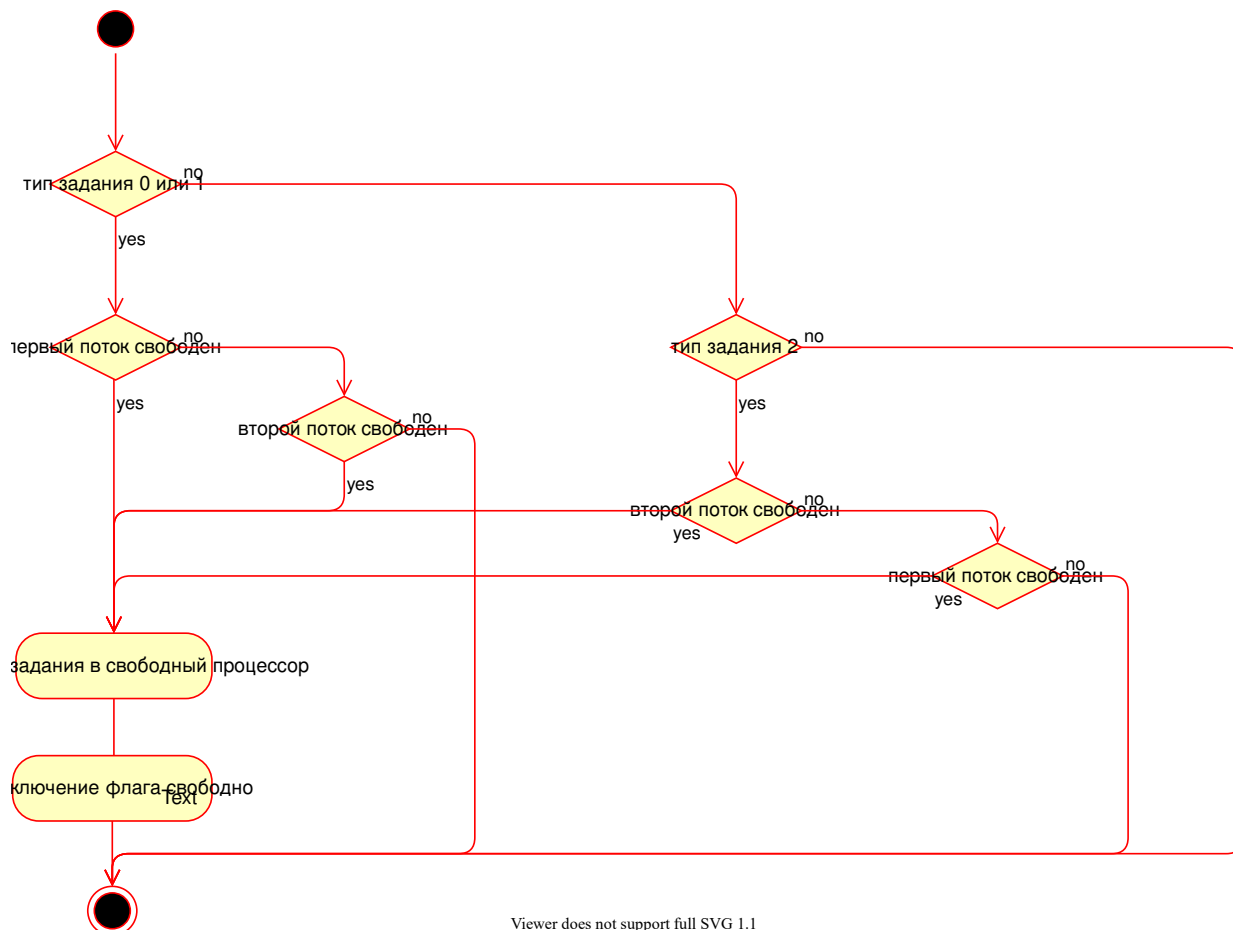
class Task():
    def __init__(self):
        time_work = [3, 6, 9]
        type_of_task = rnd.randint(high=3, low=0)
        self.current_task = TaskData()
        self.current_task.time = time_work[type_of_task]
        self.current_task.type_of_task = type_of_task

    def get_time(self):
        return self.current_task.time

    def get_type(self):
        return self.current_task.type_of_task

```

Реализуем процессор, у которого будет два потока, которые представим структурой (листинг 2). Диаграмма деятельности для добавления задачи на выполнение представлена на рисунке 2.



Viewer does not support full SVG 1.1

Рисунок 2 - Диаграмма деятельности для добавления задачи

## Листинг 2. Класс процессора.

```

from dataclasses import dataclass
from task import Task

```

```

@dataclass()

```

```

class Thread:

```

```

    time_work: int = None

```

```

    task_type: int = None

```

```

    idle: bool = True

```

```

class Processor():

```

```

    def __init__(self):

```

```

        self.p1 = Thread()

```

```

        self.p2 = Thread()

```

```

    def add_task(self, task: Task):

```

```

        if task.get_type() in [0,1]:

```

```

            if self.p1.idle:

```

```

                self.p1.time_work = task.get_time()

```

```

                self.p1.task_type = task.get_type()

```

```

                self.p1.idle = False

```

```

        elif self.p2.idle:
            self.p2.time_work = task.get_time()
            self.p2.task_type = task.get_type()
            self.p2.idle = False
    elif task.get_type() == 2:
        if self.p2.idle:
            self.p2.time_work = task.get_time()
            self.p2.task_type = task.get_type()
            self.p2.idle = False
        elif self.p1.idle:
            self.p1.time_work = task.get_time()
            self.p1.task_type = task.get_type()
            self.p1.idle = False

    def __task_perform_p1(self):
        self.p1.time_work -= 1
        if self.p1.time_work <= 0:
            self.p1.idle = True
            self.p1_task_type = None

    def __task_perform_p2(self):
        self.p2.time_work -= 1
        if self.p2.time_work <= 0:
            self.p2.idle = True
            self.p2.task_type = None

    def __str__(self):
        string = "|proc|type|time|idle|"
        if not self.p1.idle:
            string += "\n|1          |{:<4}|{:<4}|{:<4}|".format(str(self.p1.task_type), str(self.p1.time_work), str(self.p1.idle))
        else:
            string += "\n|1      |None|None|True|"
        if not self.p2.idle:
            string += "\n|2          |{:<4}|{:<4}|{:<4}|".format(str(self.p2.task_type), str(self.p2.time_work), str(self.p2.idle))
        else:
            string += "\n|2      |None|None|True|"
            string += "\n|{:<4}|{:<4}|{:<4}|{:<4}|\n\n".format("____", "____", "____", "____")
        return string

    def work(self):
        if not self.p1.idle:

```

```

        self.__task_perform_p1()
    else:
        self.p1.idle = True
    if not self.p2.idle:
        self.__task_perform_p2()
    else:
        self.p2.idle = True

def idle_proc(self):
    return self.p2.idle or self.p1.idle

```

Реализуем класс очереди, диаграммы деятельности для добавления задачи в очереди и ее удаления из очереди представлены на рисунках 4 и 3 соответственно, листинг класса представлен листинге 3.

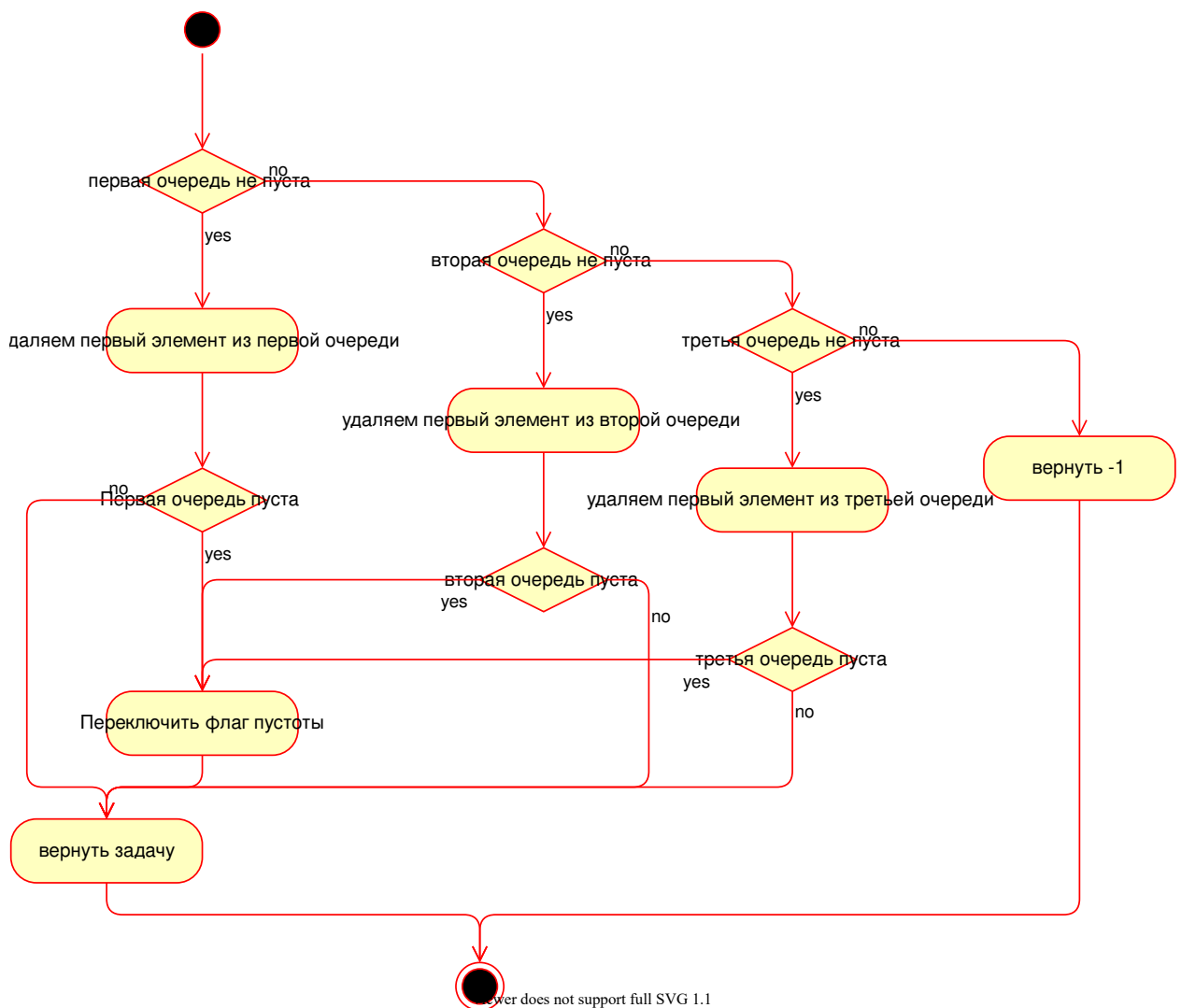
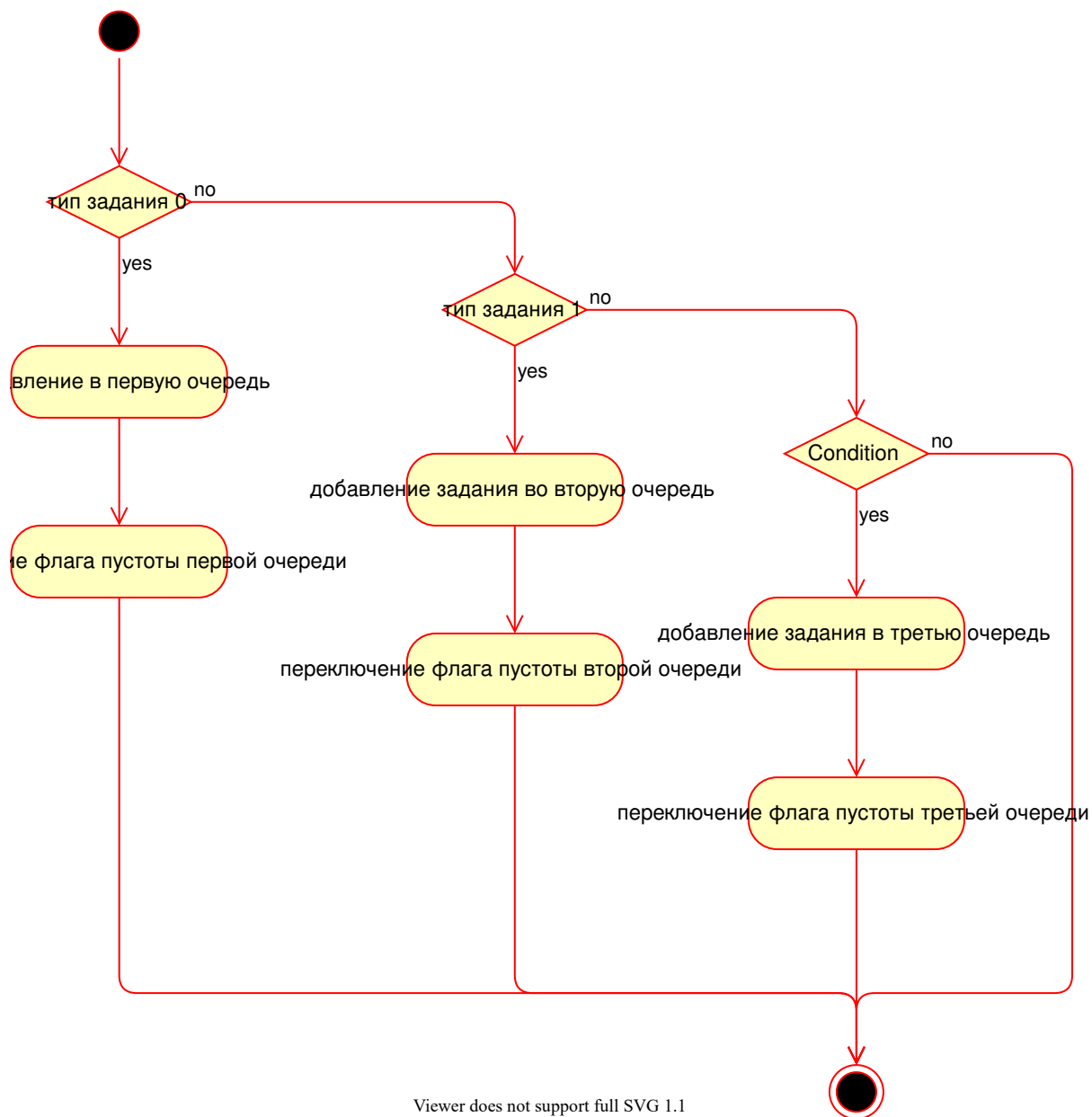


Рисунок 3 - Удаление элемента из очереди



Viewer does not support full SVG 1.1

Рисунок 4 - Добавление задачи в очередь

### Листинг 3. Очередь задач.

```

from dataclasses import dataclass
from task import Task

```

```

@dataclass()
class QueueData:
    list_of_task = []
    is_empty: bool = True

class Queue():

    def __init__(self):
        self.q1 = QueueData()

```

```

        self.q2 = QueueData()
        self.q3 = QueueData()

    def add_task(self, task:Task):
        if task.get_type() == 0:
            self.q1.list_of_task.append(task)
            self.q1.is_empty = False
        elif task.get_type() == 1:
            self.q2.list_of_task.append(task)
            self.q2.is_empty = False
        elif task.get_type() == 2:
            self.q3.list_of_task.append(task)
            self.q3.is_empty = False

    def del_task(self):
        if not self.q1.is_empty:
            task = self.q1.list_of_task.pop(0)
            if len(self.q1.list_of_task) == 0:
                self.q1.is_empty = True
        elif not self.q2.is_empty:
            task = self.q2.list_of_task.pop(0)
            if len(self.q2.list_of_task) == 0:
                self.q2.is_empty = True
        elif not self.q3.is_empty:
            task = self.q3.list_of_task.pop(0)
            if len(self.q3.list_of_task) == 0:
                self.q3.is_empty = True
        else:
            task = -1
        return task

    def __str__(self):
        return str(str(self.q1.list_of_task) +
str(self.q1.is_empty) + str(self.q2.list_of_task) +
str(self.q2.is_empty) + str(self.q3.list_of_task) +
str(self.q3.is_empty))

    def get_queue_empty_flag(self):
        return self.q1.is_empty and self.q2.is_empty and
self.q3.is_empty

```

Реализуем стэк задач, диаграмма деятельности для добавления в стэк и удаления задачи из стека представлена на рисунках 5 и 6 соответственно. Листинг реализации представлен в 4.



Рисунок 5 -  
Добавление  
задачи в стэк



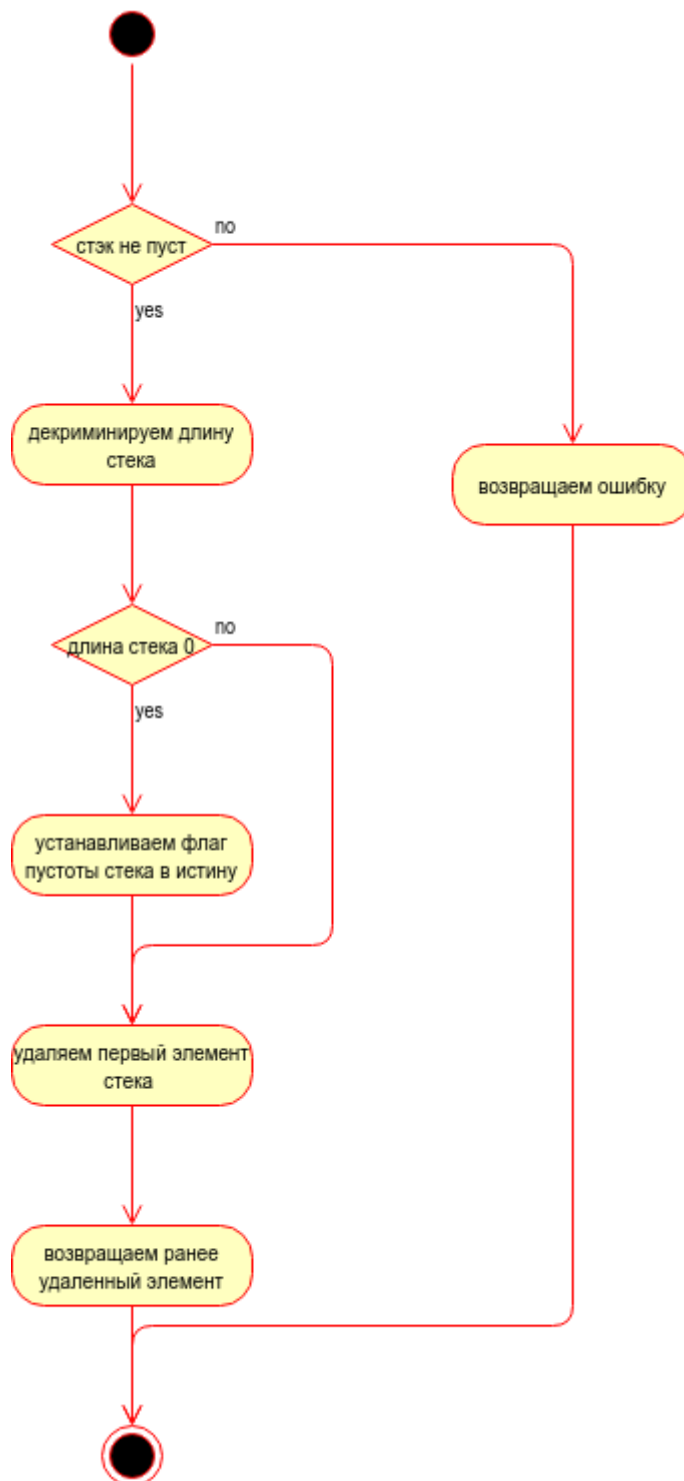



Рисунок 6 - Удаление элемента из стека

Листинг 4. Реализация стека задач.

```
from dataclasses import dataclass
```

```
@dataclass()
class TaskStack:
    list_of_task = []
    is_empty = True
    length = 0
```



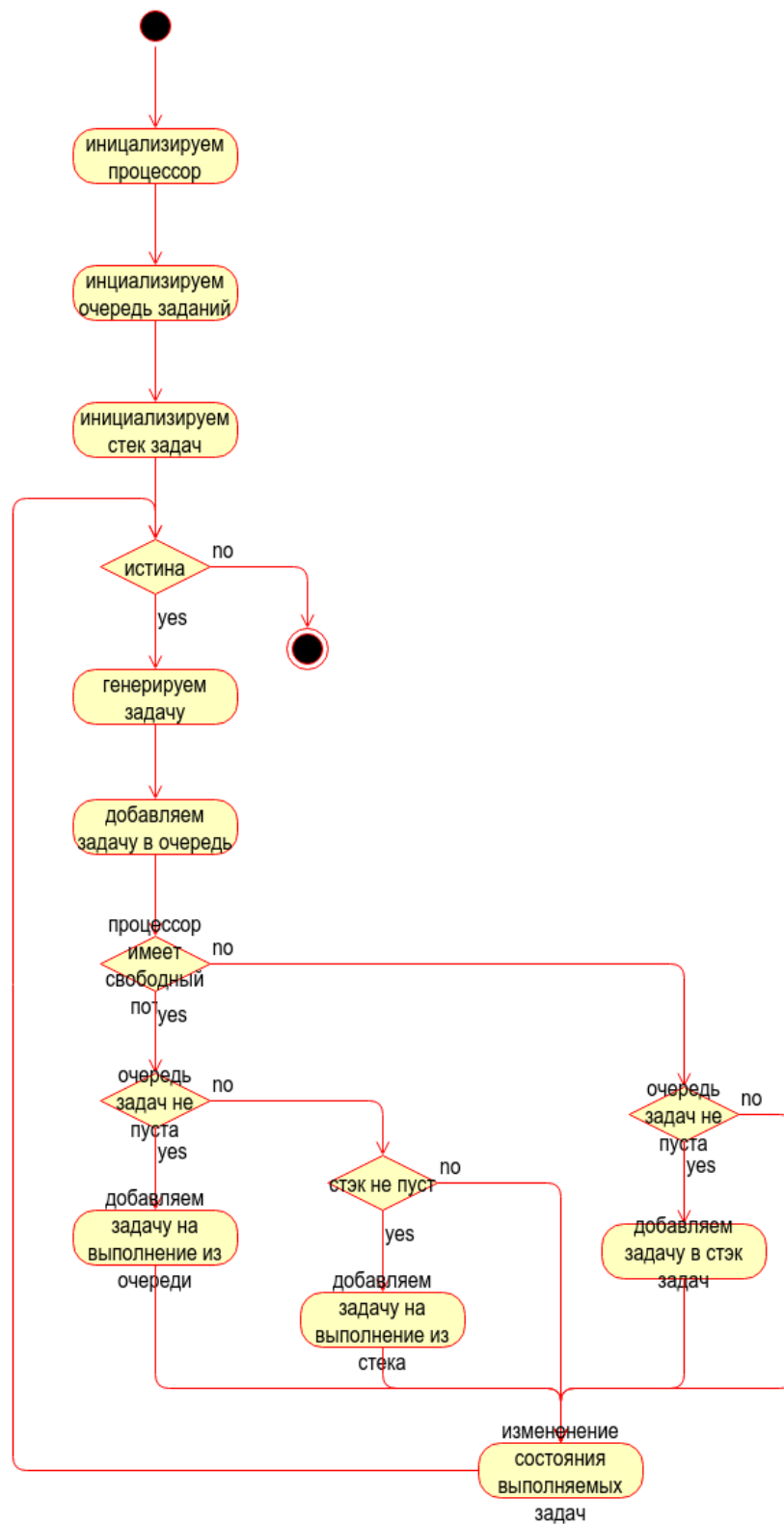


Рисунок 7 - Диаграмма деятельности для главной логики программы

Листинг 5. Файл main.py.

```

from processor import Processor
from queue import Queue
from task import Task
from stack import Stack

```

```

if __name__ == "__main__":
    proc = Processor()
    task_queue = Queue()
    task_stack = Stack()
    while True:
        a = Task()
        task_queue.add_task(a)
        if proc.idle_proc():
            if not task_queue.get_queue_empty_flag():
                proc.add_task(task_queue.del_task())
            elif not task_stack.check_is_empty():
                proc.add_task(task_stack.del_item())
        else:
            if not task_queue.get_queue_empty_flag():
                task_stack.add_item(task_queue.del_task())
    print(proc)
    print(task_stack)
    print(task_queue)
    proc.work()

```

Вывод в ходе работы были изучена структуры данных стек и очередь.