Пояснительная записка (Д/З №3)

Глазков Максим БПИ208 - Вариант 289 (9, 21)

1. Описание задания

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)	Общие для всех альтернатив переменные	Общие для всех альтернатив функции
Тексты, состоящие из цифр и латинских букв, зашифрованые различными способами	1. Шифрование заменой символов (указатель на массив пар: [текущий символ, замещающий символ]; зашифрованный текст – строка символов) 2. Шифрование циклическим сдвигом кода каждого символа на n (целое число, определяющее сдвиг; зашифрованный текст – строка символов) 3. Шифрование заменой символов на числа (пары: текущий символ, целое число – подстановка при шифровании кода символа в виде короткого целого; зашифрованный текст – целочисленный массив)	Открытый текст – строка символов	Частное от деления суммы кодов незашифрован ной строки на число символов в этой строке (действительн ое число)

2. Описание структуры ВС

Отображение содержимого классов программы

Таблица классов	**Таблица имен**	**Описание**	
Container	__init__	func **данные экземпляра:**	**def __init__(self)** self.messages – список сообщений
	file_init	func	**def file_init(self, messages)** messages – список строк
	rnd_init	func	**def rnd_init(self, length)** length – количество элементов в контейнере
	write	func	**def write(self, outstream)** outstream – поток, в который нужно записать информацию
	average	func	**def average(self)**
	average_only	func	**def average_only (self)**
	-----	**данные класса:**	symb – массив char длиной 62 crypt – массив char длиной 62 rand – экземпляр Random
		func	**def __init__(self)**

Crypter	__init__	**данные класса:**	self.message – строка для шифрования
	file_init	func	**def file_init(self, mes)** mes – сообщение (string)
	rnd_init	func	**def rnd_init(self)**
	pair_crypt	func	**def pair_crypt(self)**
	shift_crypt	func	**def shift_crypt(self)**
	numeric_crypt	func	**def numeric_crypt(self)**
	get_message_code	func	**def get_message_code(self)**
	write	func	**def write(self, outstream)** outstream – поток, в который нужно записать информацию

Отображение на память содержимого модуля container

Память программы	**Таблица имен**	**Память данных**	
def file_init	index	int	<number>
	cr	list (Crypter)	[string]
def rnd_init	index	int	<number>
	cr	list (Crypter)	[string]
def write	----------		
def average	total	int	<number>
def average_only	aver	list	[string, string, ..] (Список string)

Отображение на память содержимого модуля cryptographer

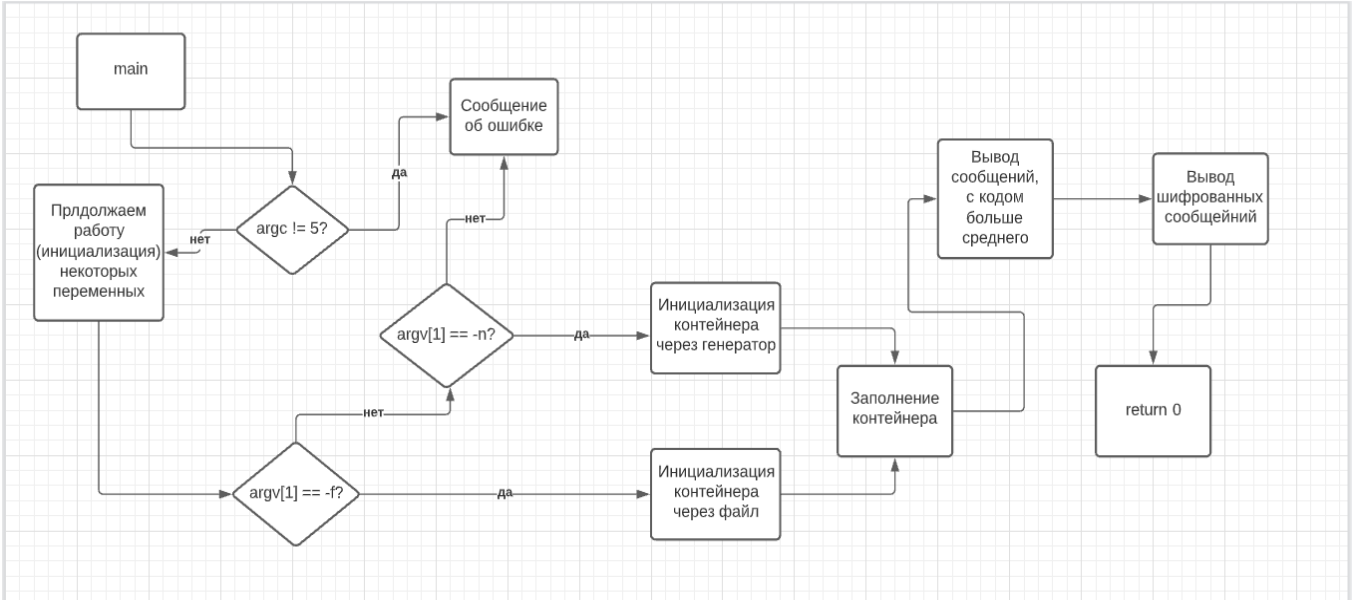
Память программы	**Таблица имен**	**Память данных**	
def file_init	----------		
def rnd_init	length	int	<number>
	i	int	<number>
def pair_crypt	crypted	string	“…”
	index	int	<number>
def shft_crypt	crypted	string	“…”
	shift	int	<number>
def numeric_crypt	crypted	string	“…”
def get_message_code	total	int	<number>
def write	----------		

Отображение содержимого модуля main

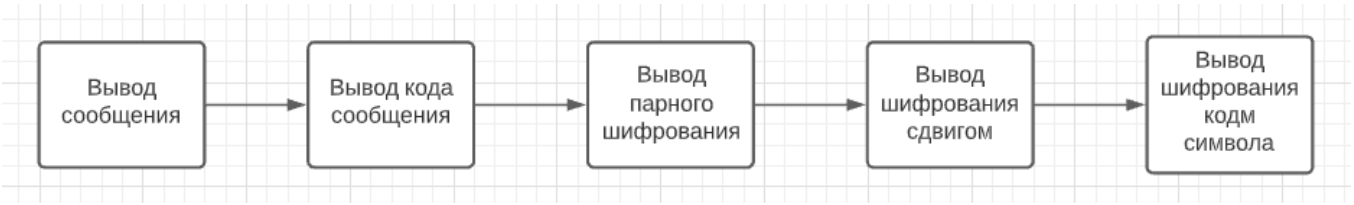
Память программы	**Таблица имен**	**Память данных**	
main.py	start	float	<number>
	container.py	module	
	is_file_input	bool	True or False
	cont	list	[Crypter, Crypter,..] (Список Crypter) или [[string], [string], ..]
	instream	file	“…”
	outstream	file	“…”

	outstream	file	“…”

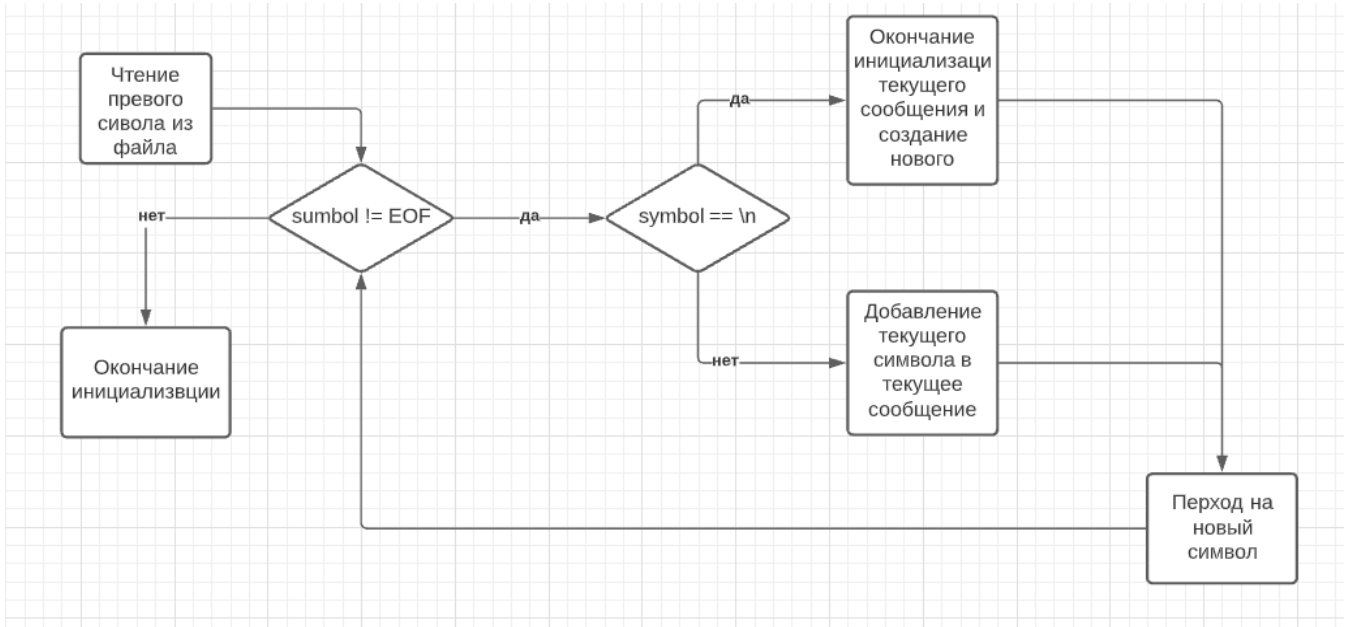
Отработка main



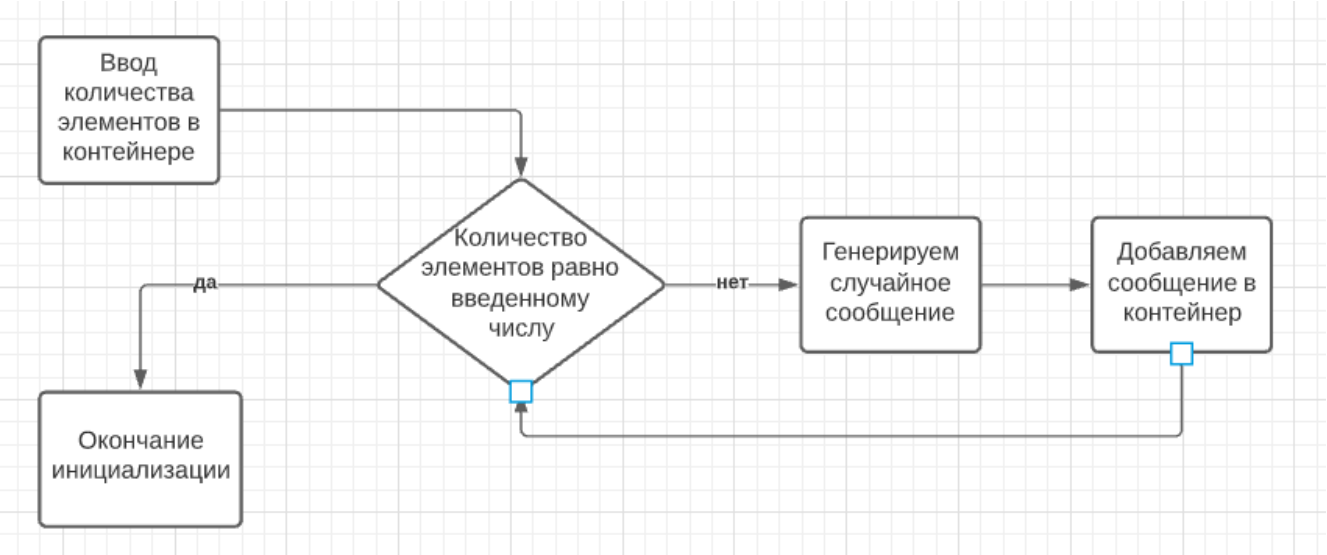
Логика вывода в файл для каждого сообщения в контейнере



Логика при инициализации контейнера через файл



Логика при инициализации контейнера случайными сообщениями



3. Входные и выходные данные

- В консоль поступает консольная команда с указанием на тестовый файл: <Программа.exe> <-f> <Тестовый_файл.txt> <Файл вывода_1.txt> <Файл вывода_2.txt>
- В консоль поступает команда без указания на тестовый файл: <Программа.exe> <-n> <Число> <Файл вывода_1.txt> <Файл вывода_2.txt>
- На вход подается только файл в формате .txt. Пример входных данных:

  My text here
  It is a new way to create a code in python
  8I want to create something special8

  В файле должны быть записаны строки, в которых будут только заглавные и прописные буквы латинского алфавита или цифры от 0 до 9, сообщения должны быть разделены знаком переноса на новую строку (“\n”), иначе весь текст будет восприниматься программой как одно единое сообщение. В случае, если в сообщении присутствуют символы, не соответствующие требованиям, описанным выше, ввод будет являться некорректным
- После обработки данных программа записывает все результаты во введенный файл в следующем виде:

  ==========
  Message text: My text here
  Message code: 94
  Pair crypt: Zd ktmk btqt
  Shift crypt: R~%yj}y%mjwj
  Numeric crypt: 771213211610112011632104101114101
  ==========

Где сначала идет текст сообщения, код сообщения, как целое число (коды сообщений могут совпадать), и три разных способа зашифровки сообщения. Во втором выходном фале записываются только те сообщения, код которых не меньше среднего кода всех сообщений в контейнере. Отображение данных о сообщениях точно такое же как и в приведенном примере. В консоль не выводятся результаты работы программы, только сообщения об ошибках или об успешном завершении работы программы

4. Дополнительные сведения

- Время работы программы будет выводиться в консоль в конце работы программы в случае ее успешного завершения
- Результат работы тестов можно увидеть в подкаталоге “outs” (тесту с названием “test0<i>.txt” соответствуют выводы в файлах “out0<i>.txt” и “average_out0<i>.txt”)
- Результаты работы программы на случайно сгенерированных сообщениях можно увидеть в файлах “rnd_out.txt” и “aver_rnd_out.txt”

5. Сравнение статически типизированного языка с динамическим

- Главное отличие двух подходов, что статически типизированные языки привязывают тип к переменной (если пишем int x = 0, то компилятор уже не даст нам присвоить переменной x значение, не являющееся int, например, “Hello world”. У динамически типизированных языков нет таких ограничений. Компилятор такого языка понимает, что 0 это int, но он не будет привязывать тип к переменной (т.е. в переменную x = 0, в программе можно записать True)
- Тесты на программах написанных на C\C++ отрабатывают быстрее, чем на Python, так как в статически типизированных языках компилятор не тратит время на определение типа переменных
- Динамически типизированный язык обладает большой гибкостью, на таких языках намного быстрее и проще получается писать программы и универсальные алгоритмы (на написание одной и той же программы на C\C++ ушло больше времени, чем на написание ее же на Python, хотя C++ я знаю лучше). Однако писать что-то большое на таких языках может быть неудобно, так как без четкого определения типов можно запутаться что есть что.