МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

ТЕМА: РАЗРАБОТКА СИСТЕМЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ ДЛЯ ПРОВЕРКИ HTML-СТРАНИЦЫ НА ВАЛИДНОСТЬ И ПРОГРАММЫ ДЛЯ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ

Студент гр. 6303 _____ Сергеенков М.Ю.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Сергеенков Михаил Юрьевич

Группа 6303

Тема работы: **РАЗРАБОТКА СИСТЕМЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ ДЛЯ ПРОВЕРКИ HTML-СТРАНИЦЫ НА ВАЛИДНОСТЬ И ПРОГРАММЫ ДЛЯ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ**

Содержание пояснительной записки:

- Содержание
- Введение
- Постановка задачи
- Описание решения задачи
- Примеры работы программы
- Заключение
- Список использованных источников
- Приложение: Исходный код программы

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 12.03.2017

Дата сдачи реферата: .05.2017

Дата защиты реферата: .05.2017

Студент _____ Сергеенков М.Ю.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В данной работе был создан проект и с использованием языков программирования C и Python, который позволяет генерировать тестовые данные и проверять, корректно ли работают программы. Для решения поставленной задачи были реализованы:

1. Генераторы тестов:
   1) Генератор html-страницы;
   2) Генератор дерева каталогов в файловой системе;
2. Эталонные решения:
   1) Эталонное решения задачи о проверке валидности html-страницы;
   2) Эталонное решение задачи о сортировке чисел из всех файлов заданного дерева каталогов;
3. Скрипты для запуска проверяемых программ и сверки результатов их работы с результатами работы эталонного решения.

В ходе работы над проектом созданы и описаны необходимые функции, управляющие генерацией исходных данных. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов. Приведено полное описание исходного кода.

СОДЕРЖАНИЕ

АННОТАЦИЯ 3
ВВЕДЕНИЕ 5
ЦЕЛЬ РАБОТЫ 5
ФОРМУЛИРОВКА ЗАДАЧИ 5
Проверка html-страницы на валидность 5
Работа с файловой системой 6
РЕШЕНИЕ ЗАДАЧИ 7
ЧЕКЕР ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СТРАНИЦЫ 7
Реализация функция для генерации тестовых данных 7
Реализация эталонного решения задачи 9
Реализация функции для проверки пользовательского решения 12
ЧЕКЕР ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ 13
Реализация функции для генерации дерева каталогов 13
Реализация эталонного решения задачи 15
Реализация функции для проверки пользовательского решения 16
Реализация функции для очистки текущей директории 17
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ 18
ЧЕКЕР ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СРАНИЦЫ 18
ЧЕКЕР ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ 20
ЗАКЛЮЧЕНИЕ 21
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22
ПРИЛОЖЕНИЯ 23
ПРИЛОЖЕНИЕ №1: ИСХОДНЫЙ КОД ЧЕКЕРА ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СТРАНИЦЫ 23
ПРИЛОЖЕНИЕ №2: ИСХОДНЫЙ КОД ЧЕКЕРА ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ 26
ПРИЛОЖЕНИЕ №3: ИСХОДНЫЙ КОД ЭТАЛОННОГО РЕШЕНИЯ НА ЯЗЫКЕ С ДЛЯ ПРОВЕРКИ ВАЛИДНОСТИ HTML-СТРАНИЦЫ 30
ПРИЛОЖЕНИЕ №4: ИСХОДНЫЙ КОД ЭТАЛОННОГО РЕШЕНИЯ НА ЯЗЫКЕ С ДЛЯ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ 33

ВВЕДЕНИЕ

ЦЕЛЬ РАБОТЫ

Создать проект, представляющий собой систему тестирования программы для проверки html-страницы на валидность и программы для работы с файловой системой.

ФОРМУЛИРОВКА ЗАДАЧИ

Проверка html-страницы на валидность

Требуется написать программу, получающую на вход html-страницу, представляющую собой код “простой” html-страницы и проверяющую ее на валидность. Программа должна: - вывести correct, если страница валидна или wrong в ином случае.

html-страница, состоит из тегов и их содержимого, заключенного в эти теги. Теги представляют собой некоторые ключевые слова, заданные в треугольных скобках. Например, <tag> (где tag - имя тега). Область действия данного тега распространяется до соответствующего закрывающего тега </tag> который отличается символом /. Теги могут иметь вложенный характер, но не могут пересекаться <tag1><tag2></tag2></tag1> - верно <tag1><tag2></tag1></tag2> - не верно

Существуют теги, не требующие закрывающего тега.

Валидной является html-страница, в коде которой всякому открывающему тегу соответствует закрывающий (за исключением тегов, которым закрывающий тег не требуется)

Во входной строке могут встречаться любые парные теги, но гарантируется, что в тексте символы < и > не встречаются. Аттрибутов у тегов также нет. Теги, которые не требуют закрывающего тега:
, <hr>

Работа с файловой системой

Дана некоторая корневая дирректория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида <filename>.txt

В каждом текстовом файле хранится одна строка, начинающаяся с числа вида:

<число><пробел><латинские буквы, цифры, знаки препинания> (“124 string example!”)

Требуется написать программу, которая, будучи запущенной в корневой дирректории, выведет строки из файлов всех поддиректорий в порядке возрастания числа, с которого строки начинаются.

РЕШЕНИЕ ЗАДАЧИ

ЧЕКЕР ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СТРАНИЦЫ

Для реализации поставленной задачи принято решение использовать возможности образовательной платформы Stepik, в связи с чем выбрана следующая архитектура чекера:

- Функция generate() – используется для генерации тестовых данных;
- Функция solve() – представляет собой эталонное решение поставленной задачи;
- Функция check() – используется для проверки пользовательского решения.

Реализация функция для генерации тестовых данных

1) Для хранения тестовых данных в функции generate() инициализируется список test_cases[];
2) В список сохраняются два теста, написанных вручную (впоследствии они будут показаны пользователю как примеры работы программы), их содержимое задается в соответствующих константах TEST_1 и TEST_2;
3) В список сохраняется ряд тестов (их количество задается константой AUTO_TESTS). Генерация отдельно взятого теста происходит следующим образом:
    - Создается список для хранения тегов tags_list[], в который записываются сгенерированные теги (их содержимое генерируется посимвольно с использованием специально написанной функции rand_text_gen(), генерирующей слово,

размер которого колеблется от MIN_TAG_LEN до MAX_TAF_LEN):

```
def rand_text_gen(min_len, max_len,
with_br_or_hr):
    text=""
    if (with_br_or_hr):
        for i in range (random.randint(min_len,
max_len)):

text+=random.choice(string.ascii_uppercase +
string.ascii_lowercase)
            if (random.random()):
                text+="<br>"
            elif (random.random()==0):
                text+="<hr>"
            if (random.randint(MIN_POSIBILITY,
MAX_POSIBILITY)== RAND_CONSTANT):
                text+="\n"
    else:
        for i in range (random.randint(min_len,
max_len)):

text+=random.choice(string.ascii_uppercase +
string.ascii_lowercase)
    return text
```

- Создается символьная строка case, в которую сначала заносятся теги из списка tags_list и между ними с контролируемой вероятностью (для чего заведены константы MIN_POSIBILITY, MAX_POSIBILITY и RAND_CONSTANT) заносятся текстовые последовательности, а также с произвольной частотой символы табуляции и переноса строки;

- В строку case добавляются теги в обратном порядке с добавлением символа «/» после «<» (таким образом, тег становится закрывающим). Закрывающие теги появляются с вероятностью 50%. Между тегами аналогично предыдущему пункту записываются символьные последовательности;
- В список test_cases[] добавляется строка case

4) Завершая свою работу, функция generate() возвращает список test_cases.

Реализация эталонного решения задачи

1. **Эталонное решение на языке Python**
   1) Для реализации эталонного решения принято использовать принцип работы стека, для чего инициализируется список stack[];
   2) Производится посимвольный анализ входной строки, в ходе которого теги из строки помещаются в tags_list[];
   3) Производится анализ списка tags_list[]. Если тег является открывающим, он кладется на вершину стека. Если тег является закрывающим, производится его сравнение с вершиной стека и, если он является парным для вершины стека, производится снятие вершины.
   4) После обработки списка тегов стек проверяется на пустоту. В случае, если он пуст, последовательность тегов считается корректной, в противном случае – нет. В зависимости от этого возвращается соответственно “correct” или “wrong”.

2. **Эталонное решение на языке C**

   Для решения поставленной задачи было принято решение использовать следующий набор функций (ниже приведены функции и краткое описание их работы):

- `bool checkTagsSequence(char *check_str)` — функция для проверки html-страницы на валидность. На вход подается строка длины INPUT_STR_LEN. Функция возвращает *true* в случае, если страница валидна, и *false* в противном случае.

Интерфейс для работы со стеком реализован следующим набором функций:

- `char **initStack()` - функция для инициализации стека, которая занимается выделением памяти под двумерный массив и возвращает указатель на его начало.

- `void push(char **stack, char *data)` — функция для добавление нового элемента в стек. На вход подается указатель на стек и строка *data*, после чего создается новая вершина стека со значением *data*.

- `void pop(char **stack)` — функция удаляет вершину стека

- `char *top(char **stack)` — функция возвращает элемент, находящийся на вершине стека без его удаения.

- `bool stackIsEmpty(char **stack)` — функция для проверки степка на пустоту.

- `int findTop(char **stack)` — функция возвращает индекс элемента массива, который является вершиной стека.

Ниже приведено описание алгоритма работы программы.

1) Со стандартного ввода считывается строка, которая подается на вход функции checkTagsSequence.

2) Инициализируется стек tags_stack с помощью функции initStack. Выполняется посимвольный анализ строки и поиск тегов. Если встречен открывающий тег, он добавляется в стек функцией push, если же тег является закрывающим, производится его сравнение с вершиной стека, которую возвращает функция top. В случае, если тег и вершина стека равны, вершина стека удаляется с помощью функции pop.

3) После завершения посимвольного анализа строки стек проверяется на пустоту с помощью функции stackIsEmpty. В случае, если стек пуст, последовательность тегов считается валидной, в противном случае — нет.

4) В зависимости от возвращаемого значения функции checkTagsSequence на стандартный вывод выводится строка **«correct»** или **«wrong»**.

Реализация функции для проверки пользовательского решения

Функция сравнивает вывод эталонного решения с выводом пользовательской программы. Ниже приведена функция check():

```
def check(reply, clue):
    return str(reply) == str(clue)
```

ЧЕКЕР ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ

Для реализации поставленной задачи принято решение использовать следующую архитектуру:

- Функция для генерации дерева каталогов в текущей директории;
- Функция, представляющая собой эталонное решение задачи;
- Функция для запуска пользовательского решения на сгенерированных тестовых данных и сравнения результатов его работы с результатами работы эталонного решения;
- Функция для очистки текущей директории от сгенерированного дерева каталогов.

Реализация функции для генерации дерева каталогов

1) Объявлен набор констант для управления генерацией:

```
#Управление максимальной длиной пути
MAX_PATH_LEN = 200
#Управление размером названия файла
MIN_FILENAME = 1
MAX_FILENAME = 20
#Управление количеством директорий на текущем уровне
MIN_DIRS = 1
MAX_DIRS = 3
#Управление глубиной
MIN_DIR_DEPTH = 1
MAX_DIR_DEPTH = 3
#Управление количеством файлов на текущем уровне
MIN_FILES = 1
MAX_FILES = 3
```

#Управление диной текста

MIN_TEXT_LEN = 1

MAX_TEXT_LEN = 3

#Управление диапазоном чисел

MIN_NUMBER = -2147483648

MAX_NUMBER = 2147483647

#Управление длиной названий директорий

MIN_DIR_NAME = 1

MAX_DIR_NAME = 5

2) Реализован набор вспомогательных функций
   - Функция для генерации названия файла:
     `generateFileName()`
   - Функция для генерации символьной строки:
     `generateString(min, max)`
     Здесь min и max служат ограничениями на длину строки;
   - Функция для генерации случайного числа для файла
     `generateNumber()`
     Возвращает произвольное целое число в границах MIN_NUMBER и MAX_NUMBER;
   - Функция для генерации содержимого файла:
     `generateFileContent()`
     Возвращает строку, состоящую из числа и слова;
   - Функция для генерации файлов в директории path:
     `generateFiles(path)`
     Создает файлы и записывает в них данные.
   - Функция для генерации набора поддиректорий в текущей директории path:
     `generateDirsAndFilesOnCurrentLevel(path)`
     Используя модуль os, генерирует поддиректории.

3) Реализована функция `generate()`:
    - В текущей директории создается поддиректория Cases;
    - В вышеуказанной поддиректории генерируется ряд файлов;
    - В вышеуказанной поддиректории генерируется ряд поддиректорий;
    - В зависимости от заданной глубины генерации поддиректорий производится рекурсивный обход поддиректорий с помощью os.walk() и в каждой происходит генерация фалов и поддиректорий.

Реализация эталонного решения задачи

1. **Эталонное решение на языке Python**
    1) Реализована функция solve(), получающая на вход текущую директорию и возвращающая строку, в которую помещен отсортированный список данных из файлов;
    2) В ходе работы функции производится рекурсивный обход поддиректорий с помощью os.walk(), в ходе которого данные из файлов парами (число и слово) помещаются в список list_of_files_content[];
    3) Список list_of_files_content[] сортируется по первому элементу;
    4) Формируется строка solution, в которую помещаются последовательно элементы вышеуказанного списка с добавлением символа переноса строки после каждого элемента.

2. **Эталонное решение на языке C**

    Для реализации принято решение использовать следующий набор функций:

    - `void dirTraveler(char *dirname, int deep)`

Рекурсивная функция для обхода вложенных директорий, принимает на вход путь и глубину;

- `void sort()`

  Функция для сортировки массива структур по первому элементу;

- `void print()`

  Функция для печати массива структур;

Описана структура FileContent, содержащая следующие поля:

- number – целое число;
- text – символьная строка;

Ниже приведен краткий алгоритм работы программы:

- Вызывается функция dirTraveler, в качестве аргументов ей передается путь до директории с тестами и глубина MAX_DEPTH;
- Функция dirTraveler, используя функции, описанные в заголовочном файле dirent.h, просматривает директорию. В случае обнаружения файла последний открывается для чтения и информация из него записывается в массив content. В случае обнаружения директории последняя передается в качестве аргумента в функцию dirTraveler;
- Выполняется сортировка массива content по возрастанию поля number;
- Выполняется печать массива content.

Реализация функции для проверки пользовательского решения

Для запуска и проверки пользовательской программы реализована функция check(). Ниже приведен алгоритм ее работы:

- В цикле с количеством итераций, контролируемом константой TESTS происходит сравнение результатов пользовательского и эталонного решений;

- Генерируется дерево директорий с помощью функции generate();
- На сгенерированных тестовых данных запускается эталонное решение, результат работы которого записывается в переменную clue;
- С помощью модуля os производится компиляция и запуск эталонного решения, результат работы которого перенаправляется в файл result_file:

  os.system('gcc solution.c -o solution')

  os.system('./solution > result_file')
- Содержимое вышеуказанного файла помещается в переменную reply;
- Сравниваются значения переменных reply и clue. Если они равны, выводится строка “Test N - correct”, где N – номер теста, в противном случае выводится строка “Failed test N””, где N – номер теста, после чего цикл прерывается.

Реализация функции для очистки текущей директории

С помощью модуля shutil производится рекурсивное удаление дерева директорий, а также временных файлов:

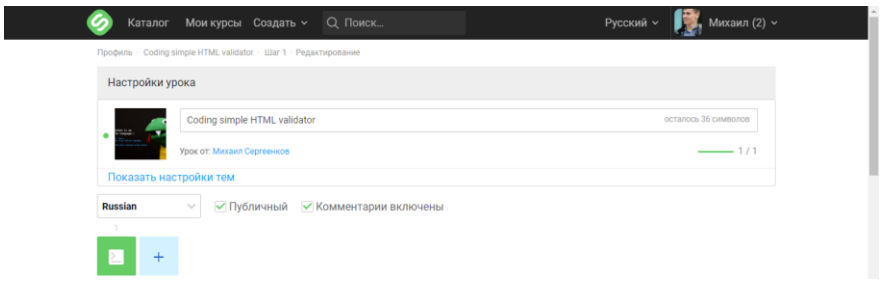
```
def clean():
     shutil.rmtree(os.getcwd()+"/Cases/")
     os.remove('./solution')
     os.remove('./result_file')
```

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

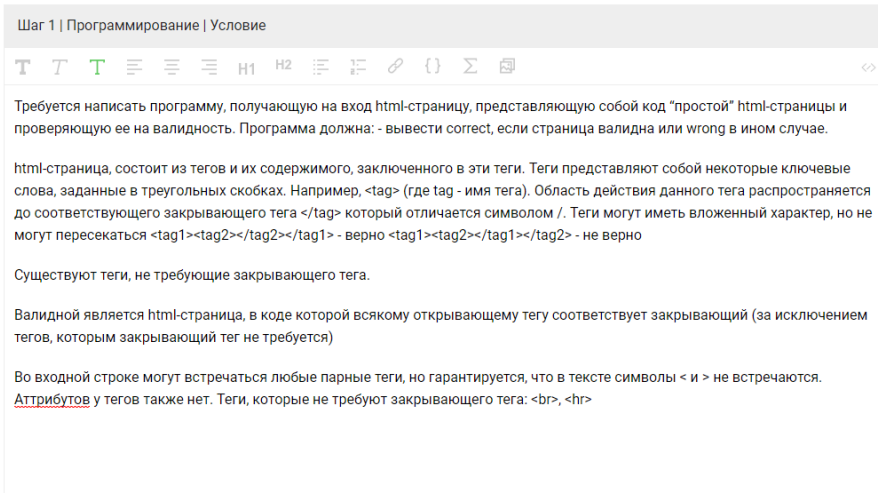
ЧЕКЕР ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СРАНИЦЫ

Для демонстрации работы программы будет использована образовательная платформа Stepik.

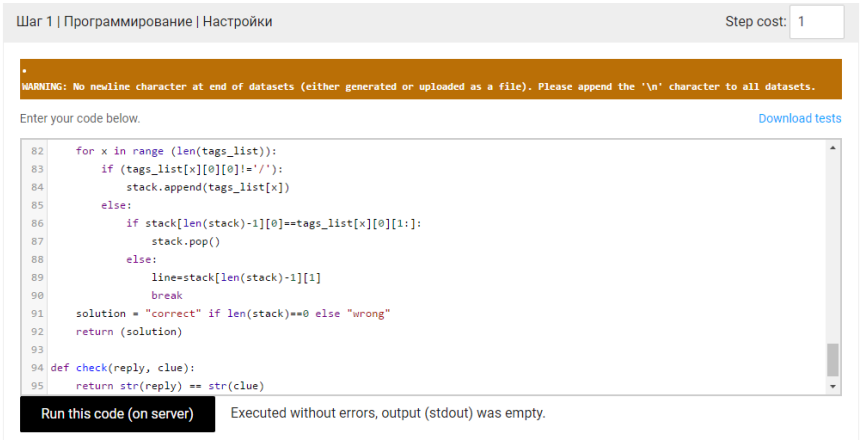
1) Создан урок на платформе Stepik:



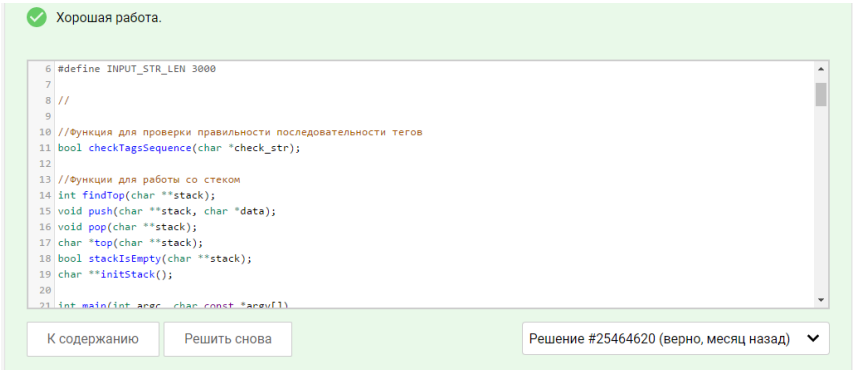
2) Написано условие задачи:



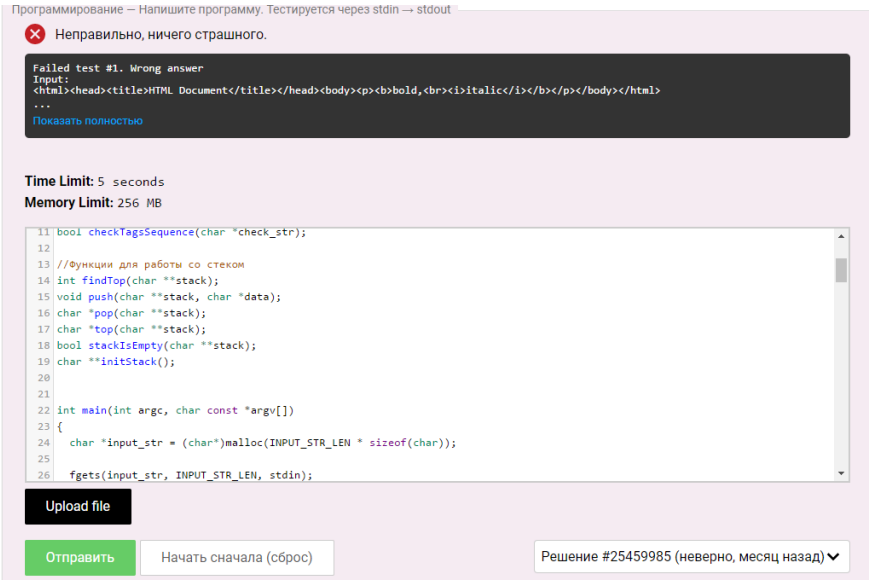
3) Код чекера внесен в специальную область:



4) Выполнена проверка эталонного решения на языке C:



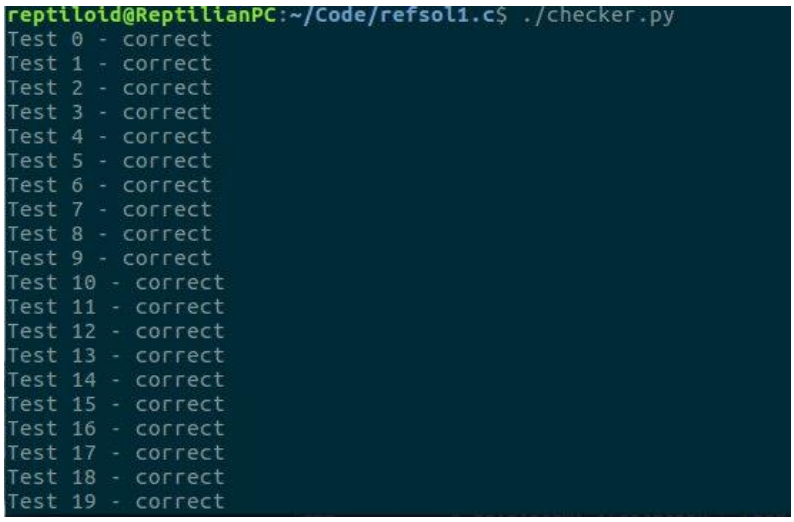
5) В эталонное решение намеренно внесены изменения, после чего снова запущена проверка:



Можно наблюдать, что проверяющая программа обнаружила ошибку в решении.

ЧЕКЕР ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ

1) В текущую директорию помещен файл checker.py и пользовательское решение solution.c, в данном случае – эталонное решение на языке C;
2) Запущен скрипт checker.py, ниже приведены результаты его работы:



3) Намеренно изменен исходный код эталонного решения, чтобы программа выводила неверный результат, и вновь запущен checker.py:

Как можно наблюдать, проверяющая программа обнаружила ошибку в решении.

ЗАКЛЮЧЕНИЕ

В ходе работы создан проект, представляющий собой систему тестирования программы для проверки html-страницы на валидность и программы для работы с файловой системой. Для реализации проверяющей системы использованы средства языка программирования Python. Помимо этого, реализовано эталонное решение двух вышеуказанных задач на языке программирования C.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.
2. Моли Б. UNIX/Linux: теория и практика программирования. М.: КУДИЦ-ОБРАЗ, 2004. 576 с.
3. Васильев А.Н. Python на примерах. Практический курс по программированию. СПб.: Издательство "Наука и техника", 2016. 431 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ №1: ИСХОДНЫЙ КОД ЧЕКЕРА ДЛЯ ПРОГРАММЫ, ПРОВЕРЯЮЩЕЙ ВАЛИДНОСТЬ HTML-СТРАНИЦЫ

```
#!/usr/bin/python3
import random
import string
MIN_TAG_LEN = 1
MAX_TAG_LEN = 4
MIN_TEXT_LEN = 0
MAX_TEXT_LEN = 100
TAGS = 20
AUTO_TESTS = 50
MIN_POSIBILITY = 0
MAX_POSIBILITY = 200
RAND_CONSTANT = 30
TEST_1            =            "<html>\n\t<head>\n\t\t<title>HTML
Document</title>\n\t</head>\n\t<body>\n\t\t<p><b>Этот текст будет
полужирным,<br>\n\t\t<i>a        этот        —        ещё        и
курсивным</i></b></p>\n\t</body>\n</html>"
TEST_2            =            "<html>\n\t<head>\n\t\t<title>HTML
Page</title>\n\t</head>\n\t<body>\n\t\t<p><b>Страница    выглядит
правильной,<br><i>но    где-то    тут    мы    потеряли    закрывающий
тег</b></p>\n\t</body>\n</html>"

#function generates random string with or without <br> and <hr>
tags
def rand_text_gen(min_len, max_len, with_br_or_hr):
      text=""
      if (with_br_or_hr):
         for i in range (random.randint(min_len, max_len)):
               text+=random.choice(string.ascii_uppercase        +
               string.ascii_lowercase)
               if (random.random()):
                  text+="<br>"
               elif (random.random()==0):
                  text+="<hr>"
               if (random.randint(MIN_POSIBILITY, MAX_POSIBILITY)==
               RAND_CONSTANT):
                  text+="\n"
      else:
         for i in range (random.randint(min_len, max_len)):
```

```
            text+=random.choice(string.ascii_uppercase        +
            string.ascii_lowercase)
      return text

def generate():

      test_cases = []

      #first two hand-made tests (human-readable)
      test_cases.append(TEST_1)
      test_cases.append(TEST_2)
      #auto tests
      for x in range (3, AUTO_TESTS):
         tags_list=[]
         for i in range (2, TAGS):
               tags_list.append('<'+rand_text_gen(MIN_TAG_LEN,
               MAX_TAG_LEN, 0)+'>')
         case=""
         for i in range (len(tags_list)):
               case+=tags_list[i]
               if random.randint(MIN_POSIBILITY, RAND_CONSTANT):
                  case+='\n'
                  if                random.randint(MIN_POSIBILITY,
                  MAX_POSIBILITY):
                     case+='\t'
               case+=rand_text_gen(MIN_TEXT_LEN, MAX_TEXT_LEN, 1)
         tags_list.reverse()
         for i in range (len(tags_list)):
               case+=rand_text_gen(MIN_TEXT_LEN, MAX_TEXT_LEN, 1)
               if random.random():
                  case+='</'
                  case+=tags_list[i][1:]
         test_cases.append(str(case))

      return test_cases


def solve(dataset):
      string=dataset
      solution=""
      tags_list=[]
      flag=0
      tag=""
      stack=[]
```

```
        for i in range (len(dataset)):
           if (flag==0) and (dataset[i])=='<':
                flag=1
           elif (flag==1) and (dataset[i]!='>'):
                tag+=dataset[i]

           else:
                flag=0
                if (tag!="") and (tag!="br") and (tag!="hr"):
                   tags_list.append(tag)
                tag=""
        for x in range (len(tags_list)):
           if (tags_list[x][0][0]!='/'):
                stack.append(tags_list[x])
           else:
                if stack[len(stack)-1][0]==tags_list[x][0][1:]:
                   stack.pop()
                else:
                   line=stack[len(stack)-1][1]
                   break
        solution = "correct" if len(stack)==0 else "wrong"
        return (solution)

def check(reply, clue):
        return str(reply) == str(clue)
```

ПРИЛОЖЕНИЕ №2: ИСХОДНЫЙ КОД ЧЕКЕРА ДЛЯ ПРОГРАММЫ, РАБОТАЮЩЕЙ С ФАЙЛОВОЙ СИСТЕМОЙ

```
#!/usr/bin/python3

import os
import random
import string
import fnmatch
import shutil

#Управление количеством тестов
TESTS = 20
#Управление максимальной длиной пути
MAX_PATH_LEN = 100
#Управление размером названия файла
MIN_FILENAME = 1
MAX_FILENAME = 4
#Управление количеством директорий на текущем уровне
MIN_DIRS = 1
MAX_DIRS = 3
#Управление глубиной
MIN_DIR_DEPTH = 1
MAX_DIR_DEPTH = 3
#Управление количеством файлов на текущем уровне
MIN_FILES = 1
MAX_FILES = 3
#Управление диной текста
MIN_TEXT_LEN = 1
MAX_TEXT_LEN = 3
#Управление диапазоном чисел
MIN_NUMBER = -100
MAX_NUMBER = 100
#Управление длиной названий директорий
MIN_DIR_NAME = 1
MAX_DIR_NAME = 4

#Функция генерирует название файла
def generateFileName():
    result = ""
    for x in range(random.randint(MIN_FILENAME, MAX_FILENAME)):
```

```
        result    +=    random.choice(string.ascii_uppercase    +
string.ascii_lowercase)
    result += ".txt"
    return result

#Функция генерирует символьную строку
def generateString(min, max):
     result = ""
     for x in range (random.randint(min, max)):
         result    +=    random.choice(string.ascii_uppercase    +
string.ascii_lowercase + string.digits)
     return result

#Функция генерирует случайное число для файла
def generateNumber():
        result = int(random.randint(MIN_NUMBER, MAX_NUMBER))
        return result
#Функция генерирует содержимое файла
def generateFileContent():
    result     =     str(generateNumber())     +     "     "     +
generateString(MIN_TEXT_LEN, MAX_TEXT_LEN)
    return result

#Функция генерирует файлы в директории path
def generateFiles(path):
    for x in range(random.randint(MIN_FILES, MAX_FILES)):
        file_path = path + "/" + generateFileName()
        file_to_generate = open(file_path, 'w')
        file_to_generate.write(generateFileContent())
        file_to_generate.close()

#Функция генерирует набор директорий в текущей директории
def generateDirsAndFilesOnCurrentLevel(path):
    for i in range(random.randint(MIN_DIRS, MAX_DIRS)):
        if random.random():
            dir_to_generate     =     str(path)     +     "/"     +
generateString(MIN_DIR_NAME, MAX_DIR_NAME)
            if    (not    os.path.exists(dir_to_generate))    and
(len(dir_to_generate) < MAX_PATH_LEN):
                os.makedirs(dir_to_generate)
                generateFiles(dir_to_generate)

def generate():
    #current_dir = str(os.getcwd())
```

```
    root_dir = os.getcwd() + "/Cases"
    os.makedirs(root_dir)
    generateFiles(root_dir)
    generateDirsAndFilesOnCurrentLevel(root_dir)

    for x in range(random.randint(MIN_DIR_DEPTH, MAX_DIR_DEPTH)):
        for root, dirs, files in os.walk(root_dir, topdown=False):
            for name in dirs:

generateDirsAndFilesOnCurrentLevel(os.path.join(root, name))
    return root_dir

def solve(root_dir):
    pattern = '*.txt'
    list_of_files_content = [] #Лист для хранения данных из файлов
    pair = []
    for root, dirs, files in os.walk(root_dir):
        for filename in fnmatch.filter(files, pattern):
            file_to_read = open(os.path.join(root, filename), 'r')
            number, string = file_to_read.read().split(" ")
            pair.append(int(number))
            pair.append(string)
            list_of_files_content.append(pair)
            file_to_read.close()
            pair = []
    #Сортировка по числу
    list_of_files_content.sort(key=lambda item: item[0])
    solution = ""
    list_length = len(list_of_files_content)
    for x in range(list_length):
         solution  +=  str(list_of_files_content[x][0])  +  "  "  +
list_of_files_content[x][1]
         if x < list_length - 1:
             solution += "\n"
    return solution

def clean():
    shutil.rmtree(os.getcwd()+"/Cases/")
    os.remove('./solution')
    os.remove('./result_file')

def check():
    for x in range(TESTS):
        generate()
```

```
        clue = solve('./Cases')
        os.system('gcc solution.c -o solution')
        os.system('./solution > result_file')
        result_file = open('./result_file', 'r')
        reply = str(result_file.read())
        result_file.close()
        if (reply == clue):
            print('Correct')
        else:
            print('Failed test {}'.format(x))
            break
    clean()
check()
```

ПРИЛОЖЕНИЕ №3: ИСХОДНЫЙ КОД ЭТАЛОННОГО РЕШЕНИЯ НА ЯЗЫКЕ С ДЛЯ ПРОВЕРКИ ВАЛИДНОСТИ HTML-СТРАНИЦЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define INPUT_STR_LEN 3000

//Функция для проверки валидности html страницы
bool checkTagsSequence(char *check_str);

//Функции для работы со стеком
int findTop(char **stack); //получение индекса вершины стека
void push(char **stack, char *data); // добавление элемента в стек
void pop(char **stack); //удаление вершины стека
char *top(char **stack); //получение вершины стека
bool stackIsEmpty(char **stack); //проверка стека на пустоту
char **initStack(); //инициализация стека

int main(int argc, char const *argv[])
{
  char *input_str = (char*)malloc(INPUT_STR_LEN * sizeof(char));

  fgets(input_str, INPUT_STR_LEN, stdin);

  fprintf(stdout, checkTagsSequence(input_str) ? "correct\n" : "wrong\n");

  free(input_str);

  return 0;
}

bool checkTagsSequence(char *check_str)
{
  char **tags_stack = initStack();
  bool inside_tag_flag = false;
  bool tag_is_closer = false;
  char tag[INPUT_STR_LEN];
  int tag_sym_counter = 0;

  for (int i = 0; i < INPUT_STR_LEN; ++i)
  {
    if (inside_tag_flag)
    {
      if(check_str[i] == '/')
      {
        tag_is_closer = true;
        continue;
      }
      else
      {
        tag[tag_sym_counter] = check_str[i];
        ++tag_sym_counter;
      }
    }
    if (check_str[i] == '<')
    {
```

```
      inside_tag_flag = true;
      tag_sym_counter = 0;
      memset(tag, 0, INPUT_STR_LEN);
    }
    else if(check_str[i] == '>')
    {
      inside_tag_flag = false;
       if ((strncmp(tag, "br", 2) != 0) &&
           (strncmp(tag, "hr", 2) !=0))
       {
        if (tag_is_closer)
        {
          if (strncmp(tag, top(tags_stack), INPUT_STR_LEN) == 0)
          {
            pop(tags_stack);
          }
          tag_is_closer = false;
        }
        else
        {
          push(tags_stack, tag);
        }
      }
    }
  }
  return stackIsEmpty(tags_stack) ? true : false;
  free(tags_stack);
}

//функция инициализации стека
char **initStack()
{
  char **stack = (char**)malloc(INPUT_STR_LEN * sizeof(char*));
  for (size_t i = 0; i < INPUT_STR_LEN; i++)
  {
    stack[i] = (char*)malloc(INPUT_STR_LEN * sizeof(char));
    memset(stack[i], 0, sizeof(char));
  }
  return stack;
}

//функция возвращает индекс вершины стека
int findTop(char **stack)
{
  int i;
  for (i = 0; i < INPUT_STR_LEN; ++i)
  {
    if (strlen(stack[i]) == 0) break;
  }
  return i - 1;
}

//функция для добавления элемента в стек
void push(char **stack, char *data)
{
  int top_index = findTop(stack);
  if (top_index == (INPUT_STR_LEN - 1))
  {
    exit(1); //stack overflow
  }
  else
  {
    ++top_index;
```

```
    strncpy(stack[top_index], data, INPUT_STR_LEN);
  }
}

//функция снятия элемента с вершины стека
void pop(char **stack)
{
  int top_index = findTop(stack);
  memset(stack[top_index], 0, INPUT_STR_LEN);
}

//функция возвращает вершину стека без снятия
char *top(char **stack)
{
  return stack[findTop(stack)];
}

//функция для проверки стека на пустоту
bool stackIsEmpty(char **stack)
{
  return strlen(stack[0]) == 0 ? true : false;
}
```

ПРИЛОЖЕНИЕ №4: ИСХОДНЫЙ КОД ЭТАЛОННОГО РЕШЕНИЯ НА ЯЗЫКЕ С ДЛЯ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>

#define ROOT_PATH "./Cases/"

#define MAX_PATH_LEN 3000
#define ARRAY_LEN 1000
#define MAX_TEXT_LENGTH 100
#define MAX_DIR_NAME 255
#define MAX_DEPTH 100

typedef struct FileContent
{
  int number;
  char text[MAX_TEXT_LENGTH];
} FileContent;

FileContent content[ARRAY_LEN];
int arr_index = 0;

//Рекурсивная функция для обхода поддиректорий
void dirTraveler(char *dirname, int deep);
//функция для сортировки массива структур
void sort();
//функция для печати массива структур
void print();

int main(int argc, char const *argv[])
{

  char path[MAX_PATH_LEN];
  strcat(path, ROOT_PATH);
  dirTraveler(path, MAX_DEPTH);
  sort();
  print();
  return 0;
}

void dirTraveler(char *dirname, int deep)
```

```
{
  if (deep==0)
      return;
  DIR *dir;
  struct dirent *entry;
  if ((dir = opendir(dirname)) == NULL)
  {
      exit(1);
  }
  while ((entry = readdir(dir)) != NULL)
  {
      if (strcmp(entry->d_name,".")&&strcmp(entry->d_name,".."))
      {
          char s[MAX_DIR_NAME];
          s[0]='\0';
          strcat(s,dirname);
          strcat(s,entry->d_name);
          if (opendir(s)==NULL)
          {
              FILE * file = fopen(s, "r");
              int number;
              fscanf(file,  "%d  %s",  &content[arr_index].number,
content[arr_index].text);
              ++arr_index;
              fclose(file);
          }
          else
          {
              strcat(s,"/");
              dirTraveler(s,deep-1);
          }
      }
  }
  if (closedir(dir) != 0)
  {
      exit(1);
  }
  return;
}

void sort()
{
  for (int i = 0; i < arr_index - 1; i++)
  {
```

```
    for (int j = 0; j < (arr_index - i - 1); j++)
    {
      if (content[j].number > content[j+1].number)
      {
        FileContent buf = content[j];
        content[j] = content[j+1];
        content[j+1] = buf;
      }
    }
  }
}

void print()
{
  for (int i = 0; i < arr_index; i++)
  {
    printf("%d %s\n", content[i].number, content[i].text);
  }
}
```