Доколин Георгий ИУ7-22Б

Отчет по Проектно-технологической практике (тестированию, отладке и профилированию ПО)
Задание 1
Цель

Целью данной работы является автоматизация процессов сборки и тестирования.

Пояснение скриптов

1. build_debug.sh

```
#!/bin/bash
gcc -std=c99 -Wall -Werror -Wextra -Wfloat-equal -Wfloat-conversion
-Wpedantic -g3 -c main.c
gcc -o app.exe main.c -lm
```

Скрипт **build_debug.sh** используется для сборки программы с различными ключами, которые будут показывать ошибки допущенные в процессе разработки.
В последней строке этого скрипта создаются еще несколько файлов с различными расширениями.

2. build_debug_asan.sh

```
#!/bin/bash

if [ $# != 1 ]; then
	echo Usage: ./build_debug_asan.sh file-name
	exit 1
fi

gcc -std=c99 -Wall -Werror -fsanitize=address -fno-omit-frame-
pointer -g $1 -o build_debug_asan
```

Скрипт **build_debug_asan.sh** выполняет роль address sanitizer, который отлавливает ошибки выхода за пределы массива, а также ошибки неверного использования переменных.

3. build_debug_msan.sh

```
#!/bin/bash

if [ $# != 1 ]; then
  echo Usage: ./msan.sh file-name
  exit 1
fi

gcc -std=c99 -Wall -fsanitize=memory -fPIE -pie -fno-omit-frame-pointer -g $1 -o test_msan
```

Скрипт **build_debug_msan.sh** выполняет роль memory sanitizer , который отлавливает ошибки неиспользованных перменных.

4. build_debug_ubsan.sh

```
#!/bin/bash

if [ $# != 1 ]; then
  echo Usage: ./ubsan.sh file-name
  exit 1
fi

gcc -std=c99 -Wall -fsanitize=undefined -fno-omit-frame-pointer -g $1 -o test_ubsan
```

Скрипт **build_debug_ubssan.sh** выполняет роль undefined behavior sanitizer , который отлавливает различные виды неопределенного поведения.

5. build_release.sh

```
#!/bin/bash
```

```
gcc -std=c99 -Wall -Werror -Wextra -Wfloat-equal -Wfloat-conversion
-Wpedantic -c main.c
gcc -o app.exe main.c -lm
```

Скрипт **build_release.sh** используется для сборки программы с различными ключами, которые будут показывать ошибки допущенные в процессе разработки, однако данная сборка проекта уже является предрелизной.
В последней строке этого скрипта создаются еще несколько файлов с различными расширениями.

6. clean.sh

```
#!/bin/bash
rm -f *.exe
rm -f *.o
```

Скрипт **clean.sh** используется для очистки директории после работы, поскольку в процессе выполнения скрипта создаются файлы нужные только для его работы.

7. func_tests.sh

```
#!/bin/bash
files_in=$(ls ../data | grep -E 'pos_[0-9]{1,2}_in\.txt')
files_out=$(ls ../data | grep -E 'pos_[0-9]{1,2}_out\.txt')
count_files_in=$(ls ../data | grep -E 'pos_[0-9]{1,2}_in\.txt' | wc -l)
count_files_out=$(ls ../data | grep -E 'pos_[0-9]{1,2}_out\.txt' | wc -l)
for ((i=1; i < $count_files_in + 1; i++ ))
do
        if [ $i -lt 10 ];
        then
                data_in=$(cat "../data/pos_0${i}_in.txt")
                data_out=$(cat "../data/pos_0${i}_out.txt")
                bash pos_case.sh "$data_in" "$data_out"
                if [ $? -eq 1 ]; then
                        echo 1
                        exit 1
                fi
        else
                data_in=$(cat "../data/pos_${i}_in.txt")
                data_out=$(cat "../data/pos_${i}_out.txt")
```

```
                bash pos_case.sh "$data_in $data_out"
                if [ $? -eq 1 ]; then
                        echo 1
                        exit 1
                fi
        fi
done
files_in=$(ls ../data | grep -E 'neg_[0-9]{1,2}_in\.txt')
files_out=$(ls ../data | grep -E 'neg_[0-9]{1,2}_out\.txt')
count_files_in=$(ls ../data | grep -E 'neg_[0-9]{1,2}_in\.txt' | wc -l)
count_files_out=$(ls ../data | grep -E 'neg_[0-9]{1,2}_out\.txt' | wc -l)
for ((i=1; i < $count_files_in; i++ ))
do
        if [ $i -lt 10 ];
        then
                data_in=$(cat "../data/neg_0${i}_in.txt")
                data_out=$(cat "../data/neg_0${i}_out.txt")
                bash neg_case.sh "$data_in" "$data_out"
                if [ $? -eq 1 ]; then
                        echo 1
                        exit 1
                fi
          else
                data_in=$(cat "../data/neg_${i}_in.txt")
                data_out=$(cat "../data/neg_${i}_out.txt")
                bash neg_case.sh "$data_in $data_out"
        if [ $? -eq 1 ]; then
                echo 1
                exit 1
        fi
fi

done
echo 0
exit 0
```

Скрипт **func_tests.sh** используется для того чтобы обработать все положительные и отридцательные тесты, передав входные и выходные файлы для тестирования в следующие скрипты.
Сначала создаются переменные, которые хранят в себе все входные и выходные файлы. Далее создаются перменные хранящие в себе количсества всех файлов.

В цикле скрипт проходит по каждому из файлов и передает их значения в **pos_case.sh.**
В конце обработки положительного теста, скрипт получает значение которое вернул **pos_case.sh,** в зависимости от этого ответа зависит код завершения скрипта.
В случае с негативными тестами, происходит все то же самое, только вызывается скрипт **neg_case.sh.**

8. pos_case.sh

```
#!/bin/bash
if [ $# != 2 ]; then
        echo Error
        exit 1
fi
echo $1 > save_in.txt
echo $2 > save_out.txt
(cd ../../; bash build_release.sh)
touch save_res.txt
start_file=$(find "../../" -name "app.exe")
"$start_file" < save_in.txt > save_res.txt
bash comparator.sh save_res.txt save_out.txt
if [ $? != 0 ]; then
        rm save_in.txt
        rm save_out.txt
        rm save_res.txt
        rm save_line_1.txt
        rm save_line_2.txt
        exit 1
fi
rm save_in.txt
rm save_out.txt
rm save_res.txt
rm save_line_1.txt
rm save_line_2.txt
exit 0
```

Скрипт **pos_case.sh** используется для того чтобы запустить программу с входными данными и передать ее в скриипт **comparator.sh** для сравнивания выходных данных, заявленных пользователем и фактическим результатом работы программы.

В первых строках идет проверка на наличие двух переданных параметров. Далее происходит запись переданных данных в файлы для дальнейшего запуска компаратора.
Строки **(cd ../../; bash build_release.sh)** и **start_file=$(find "../../" -name "app.exe")** сначала запускают скрипт, который создает необходимые файлы для тестировани, а дальше уже запускается эта программа, с перенаправлением вывода в текстовый файл. Потом, запускается компаратор, в котором сравнивается файл с результатом работы программы и созданный пользователем выходной файл.
После того, как **comparator.sh** закончил работу, скрипт удаляет все созданные дополнительно файлы и завершает работу с соответствующим кодом.

9. neg_case.sh

```
#!/bin/bash
if [ $# != 2 ]; then
        echo Error
        exit 1
fi
echo $1 > save_in.txt
(cd ../../; bash build_release.sh)
touch save_res.txt
start_file=$(find "../../" -name "app.exe")
"$start_file" < save_in.txt > save_res.txt
if [ $? -eq 1 ]; then
        rm save_in.txt
        rm save_res.txt
        exit 0
fi
rm save_in.txt
rm save_res.txt
exit 1
```

Скрипт **neg_case.sh** используется для того чтобы запустить программу с входными данными, при которых программа должна выдавать ошибку. Работа скрипта очень похожа на скрипт **pos_case.sh** за исключением того, что файлы не сравниваются друг с другом при помощи компаратора, а просто берется код завершениия работы и сравнивается с ожидаеммым результатом.

10. comparator.sh

```
#!/bin/bash

file_1=$1
file_2=$2

touch save_line_1.txt
touch save_line_2.txt
create_line=""

if [ -e "$file_1" ] && [ -e "$file_2" ]; then
        lines_1=$(cat $file_1)
        lines_2=$(cat $file_2)
        for line in $lines_1
        do
                create_line+="${line} "
        done
        echo "$create_line" 1> save_line_1.txt

        create_line=""
        for line in $lines_2
        do
                create_line+="${line} "
        done
        echo "$create_line" 1> save_line_2.txt
        if diff save_line_1.txt save_line_2.txt; then
                exit 0
        else
                exit 1
        fi
else
   exit 1
fi
```

Скрипт **comparator.sh** используется для того чтобы сравнить два текстовых файла, в которых хранятся ожидаемый результат работы и фактический. Программа считывает каждый из файлов построчно и записывает каждый из них в строчку, потом каждая из этих строк записывается в другой файл, которые сравниваются при помощи функции **diff.** По результату работы, прокрамма возращает соответствующий код.