**Смирнов Иван ИУ7-22Б - 2023г.**

**Отчет**

**Задание №1**

**Автоматизация функционального тестирования**

*Целью* данной работы является автоматизация процессов сборки и тестирования.

**Задание:**

В ходе задания были реализованы следующие скрипты:

1. Скрипты отладочной и релизной сборок.

**build\_debug.sh**

#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c -O0 -g main.c

gcc main.o -o app.exe -lm

**build\_release.sh**

#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c main.c

gcc main.o -o app.exe -lm

В каталоге с исходным кодом программы в файле main.c располагаются скрипты build\_debug.sh, build\_release.sh, с помощью которых автоматизируется сборка отладочной и релизной сборок проекта.

1. Скрипты отладочной сборки с санитайзерами

**build\_asan.sh**

#!/bin/bash

clang -std=c99 -Wall -fsanitize=address -fno-omit-frame-pointer -g main.c -o test\_asan

**build\_msan.sh**

#!/bin/bash

clang -std=c99 -Wall -fsanitize=memory -fPIE -pie -fno-omit-frame-pointer -g main.c -o test\_msan

**build\_ubsan.sh**

#!/bin/bash

clang -std=c99 -Wall -fsanitize=undefined -fno-omit-frame-pointer -g main.c -o test\_ubsan

Данные скрипты реализуют сборку с address, memory и undefined behavior sanitizer соответственно.

1. Скрипт очистки побочных файлов

**clean.sh**

#!/bin/bash

files="./func\_tests/scripts/\*.txt \*.exe \*.o \*.out \*.gcno \*.gcda \*.gcov"

for file in $files; do

    rm -f "$file"

done

Данный скрипт удаляет побочные файлы.

1. Компаратор для сравнения последовательностей действительных (и целых) чисел, располагающихся в двух текстовых файлах, с игнорированием остального содержимого.

**comparator.sh (для целых чисел)**

#!/bin/bash

if [ $# -ne 2 ]; then

    exit 1

fi

f1=$1

f2=$2

mask="[0-9]+"

clean\_out\_prog=$(grep -Eo "$mask" "$f1")

clean\_out\_test=$(grep -Eo "$mask" "$f2")

if [ "$clean\_out\_prog" != "$clean\_out\_test" ]; then

    exit 1

fi

exit 0

**comparator.sh (для вещественных чисел)**

#!/bin/bash

if [ $# -ne 2 ]; then

    exit 1

fi

f1=$1

f2=$2

mask="[-+]?[0-9]+[.][0-9]\*"

clean\_out\_prog=$(grep -Eo "$mask" "$f1")

clean\_out\_test=$(grep -Eo "$mask" "$f2")

if [ "$clean\_out\_prog" != "$clean\_out\_test" ]; then

    exit 1

fi

exit 0

Принцип работы компаратора:

* На вход подаются файл с выходными данными, который получился при выполнении программы (f1) и файл с выходными данными, который существовал в папке с тестами изначально и считается *ожидаемым результатом* выходных данных (f2).
* С помощью маски (в разных компараторах – разные маски) и функции grep в две отдельные переменные помещается содержимое двух этих файлов (либо целые числа, либо вещественные).
* Сравниваются результаты grep-a, помещенные в переменные. Если они не одинаковы – содержимое разное. Иначе – одинаковое.
* В случае одинакового содержимого возвращается код возврата – 0, в случае разного содержимого – 1.

1. Компаратор для сравнения содержимого двух текстовых файлов, располагающегося после первого вхождения подстроки «Result:␣».

В отличие от предыдущих компараторов, данный проверяет все выходные данные, включающие подстроку «Result:␣». Маска отличается.

**comparator.sh**

#!/bin/bash

if [ $# -ne 2 ]; then

    exit 1

fi

f1=$1

f2=$2

mask="Result: .\*"

clean\_out\_prog=$(grep -Eo "$mask" "$f1")

clean\_out\_test=$(grep -Eo "$mask" "$f2")

if [ "$clean\_out\_prog" != "$clean\_out\_test" ]; then

    exit 1

fi

exit 0

1. Скрипт pos\_case.sh для проверки позитивного тестового случая по определённым правилам.

**pos\_case.sh**

#!/bin/bash

ok="0"

fail="1"

if [ $# -ne 2 ]; then

    exit "$fail"

fi

in\_test=$1

out\_test=$2

tmp\_out="tmp\_out.txt"

command="../../app.exe"

$command < "$in\_test" > "$tmp\_out"

error="$?"

if [[ $error -ne 0 ]]; then

    exit "$fail"

fi

./comparator.sh $tmp\_out $out\_test

return\_code="$?"

if [[ return\_code -eq 0 ]]; then

    exit "$ok"

else

    exit "$fail"

fi

Принцип работы скрипта:

* На вход подаются файлы *pos\_NN\_in.txt* и *pos\_NN\_out.txt*, которые заранее заданы в папке /func\_tests/data.
* С помощью перенаправления ввода-вывода скрипт запускает программу с входными данными из in-файла. Результат программы сохраняет во временном файле *tmp\_out.txt.*
* Если код возврата программы отличен от нуля, то тест считается проваленным, так как в программе произошла ошибка (возвращается код возврата – 1). Если код возврата равен нулю, то скрипт продолжает работу (см. след. пункт).
* Далее скрипт запускает компаратор, который сравнивает содержимое файлов *tmp\_out.txt* и *pos\_NN\_out.txt* (см. выше). Если код возврата компаратора – 0, значит содержимое (с точки зрения компаратора) совпадает и тест считается успешным (возвращается код возврата - 0), иначе тест считается проваленным (код возврата - 1).

1. Скрипт neg\_case.sh для проверки негативного тестового случая по определённым правилам.

**neg\_case.sh**

#!/bin/bash

if [ $# -ne 1 ]; then

    exit 1

fi

ok="0"

fail="1"

in\_test=$1

tmp\_out="tmp\_out.txt"

command="../../app.exe"

$command < "$in\_test" > "$tmp\_out"

error="$?"

if [[ $error -ne 0 ]]; then

    exit 0

else

    exit 1

fi

Так как под негативным тестом подразумевается, что при попытке программы обработать входные данные – вернется код ошибки, то скрипт возвращает код возврата (0), если код возврата программы с тестовыми данными отличен от нуля, и наоборот.

1. Скрипты для обеспечения автоматизации функционального тестирования.

**func\_tests.sh**

#!/bin/bash

test\_ok="0"

files\_count=0

count\_errors=0

files="../data/pos\_??\_in.txt"

for file\_in in $files; do

    if [ -f "$file\_in" ]; then

        number=$(echo "$file\_in" | grep -o "[0-9]\*")

    else

        echo "No positive tests"

        continue

    fi

    file\_out="../data/pos\_""$number""\_out.txt"

    if [ -f "$file\_out" ]; then

        ./pos\_case.sh $file\_in $file\_out

        error="$?"

    else

        echo "pos\_""$number""\_in"": FAIL (No Output File)"

        count\_errors=$((count\_errors + 1))

        files\_count=$((files\_count + 1))

        continue

    fi

    if [ "$error" -eq "$test\_ok" ]; then

        echo "pos\_""$number""\_in"": PASS"

    else

        echo "pos\_""$number""\_in"": FAIL"

        count\_errors=$((count\_errors + 1))

    fi

    files\_count=$((files\_count + 1))

done

echo

files="../data/neg\_??\_in.txt"

for file\_in in $files; do

    if [ -f "$file\_in" ]; then

        number=$(echo "$file\_in" | grep -o "[0-9]\*")

    else

        echo "No negative tests"

        continue

    fi

    ./neg\_case.sh $file\_in

    error="$?"

    if [ "$error" -eq "$test\_ok" ]; then

        echo "neg\_""$number""\_in"": PASS"

    else

        echo "neg\_""$number""\_in"": FAIL"

        count\_errors=$((count\_errors + 1))

    fi

    files\_count=$((files\_count + 1))

done

percentage=$(echo "scale=9; ($files\_count-$count\_errors)/$files\_count\*100" | bc )

echo

echo "Tests passed (in %): "

echo $percentage | awk '{printf "%.0f\n",$1}'

exit "$count\_errors"

Принцип работы скрипта:

* Скрипт находит все файлы вида *pos\_NN\_in.txt.* При их отсутствии программа сообщает, что позитивных тестов нет.
* Далее скрипт находит для каждого in-файла соответствующий out-файл. Если out-файла с очередным номером не существует, то тест с этим номером считается проваленным. При существовании out-файла скрипт запускает скрипт *pos\_case.sh* c данными файлами на вход.
* Если код возврата скрипта *pos\_case.sh* – 0, то тест считается пройденным. Иначе – проваленным.
* Далее все предыдущие пункты аналогично выполняются для негативных тестов.
* Во время выполнения скрипт считает общее количество тестов и количество пройденных тестов. После обработки всех тестов, скрипт выводит информацию о том, сколько процентов от всех тестов оказались пройденными. 100% означает, что все тесты прошли успешно.

**collect\_coverage.sh**

#!/bin/bash

cd ./func\_tests/scripts/ || exit 1

./func\_tests.sh

cd ../../

echo ""

echo "Coverage (in %):"

gcov main.c > "tmp.txt"

var=$(cat tmp.txt)

echo ${var#\*:} | sed 's/%\*$/ /g' | sed 's/ .\*//'

rm -f "tmp.txt"

Данный скрипт запускает скрипт *func\_tests.sh,* чтобы отобразить информацию о полноте тестирования, а затем с помощью утилиты gcov считает и выводит процент покрытия кода программы *main.c.*

Для правильной работы скрипта, перед его запуском необходимо собрать программу с помощью скрипта *build\_gcov.sh*.

**build\_gcov.sh**

#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c -O0 -g --coverage main.c

gcc main.o -o app.exe --coverage -lm

**Заключение**

Написанные в ходе задания скрипты помогли автоматизировать процесс тестирования и сборки программы. Цель была успешна выполнена. Данные скрипты используются в курсе “Программирование на Си” для лабораторных работ. Скрипты изначально были созданы для первой лабораторной работы, однако они написаны таким образом, чтобы в следующих лабораторных работах приходилось делать как можно меньше изменений (в идеале только comparator.sh).

**Список литературы:**

1. Курс “Проектно-технологическая практика (знакомство с Linux)”:

<https://e-learning.bmstu.ru/iu7/course/view.php?id=76>

1. Крус “Проектно-технологическая практика (тестирование, отладка и профилирование ПО)”:

<https://e-learning.bmstu.ru/iu7/course/view.php?id=73>

1. Тестировщик регулярных выражений:

<https://regex101.com/>

1. Тестировщик команды Sed:

<https://sed.js.org/index.html>

1. Статья на тему “Команда Sed для Linux/Unix с примерами”:

<https://www.geeksforgeeks.org/sed-command-in-linux-unix-with-examples/>