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.

2) 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.

3) 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.

4) 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-а, помещенные в переменные. Если они не одинаковы – содержимое разное. Иначе – одинаковое.
- В случае одинакового содержимого возвращается код возврата – 0, в случае разного содержимого – 1.

5) 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
```

6) 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).

7) 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.

8) 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* с 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

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

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

3. Тестировщик регулярных выражений: https://regex101.com/

4. Тестировщик команды Sed: https://sed.js.org/index.html

5. Статья на тему “Команда Sed для Linux/Unix с примерами”: https://www.geeksforgeeks.org/sed-command-in-linux-unix-with-examples/