Паламарчук Андрей ИУ7-23Б

Автоматизация функционального тестирования. Аргументы командной строки приложения

Цель работы

Целью данной работы является автоматизация процессов сборки и тестирования.

Задачи

1. Реализовать скрипты отладочной и релизной сборок.
2. Реализовать скрипты отладочной сборки с санитайзерами.
3. Реализовать скрипт очистки побочных файлов.
4. Реализовать компаратор для сравнения содержимого двух текстовых файлов.
5. Реализовать скрипт pos_case.sh для проверки позитивного тестового случая по определённым далее правилам.
6. Реализовать скрипт neg_case.sh для проверки негативного тестового случая по определённым далее правилам.
7. Обеспечить автоматизацию функционального тестирования.

Реализация

1. Скрипт релизной сборки

```
#!/bin/bash


# Компиляция

gcc -std=c99 -c main.c func.c add_entry.c sort.c print_substr.c


# Комановка

gcc -o main_release.exe main.o func.o add_entry.o sort.o
print_substr.o -lm
```

2. Скрипт отладочной сборки

```
#!/bin/bash


# Компиляция

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -g3 main.c func.c add_entry.c sort.c
print_substr.c
```

```
# Комановка(-o название исполняемого файла)

gcc -o main_debug.exe main.o func.o add_entry.o sort.o print_substr.o
–lm
```

3. Скрипт отладочной сборки с адрес санитайзером

```
#!/bin/bash


clang -std=c99 -Wall -fsanitize=address -fno-omit-frame-pointer -g
main.c func.c add_entry.c sort.c print_substr.c -o main_asan.exe
```

4. Скрипт отладочной сборки с санитайзером памяти

```
#!/bin/bash


clang -std=c99 -Wall -fsanitize=memory -fno-omit-frame-pointer -g
main.c func.c add_entry.c sort.c print_substr.c -o main_msan.exe
```

5. Скрипт отладочной сборки с санитайзером UB

```
#!/bin/bash


clang -std=c99 -Wall -fsanitize=undefined -fno-omit-frame-pointer -g
main.c func.c add_entry.c sort.c print_substr.c -o main_udsan.exe
```

6. Скрипт очистки побочных файлов

```
#!/bin/bash


# *.txt *.exe *.o *.out *.gcno *.gcda *.gcov

junk_files1="./func_tests/*/*.out"

junk_files2="./func_tests/*/*.exe"

junk_files3="./*.exe ./*.o ./*.out ./*.gcno ./*.gcda ./*.gcov"

# Проверить, существует ли файл

for el in $junk_files1 $junk_files2 $junk_files3;

do
```

```
    if [[ -f $el ]]; then
        rm "$el"
    fi
done
```

7. Компаратор для сравнения двух файлов

```
#!/bin/bash

# Проверка количества аргументов
if [ $# -ne 2 ]; then
    exit 1
fi

out_prog=$1
out_test=$2

# сравниваем их
rc=$(cmp -s "$out_prog" "$out_test")
if [ ! "$rc" ]; then
    exit 1
fi
exit 0
```

8. Скрипт pos_case.sh

```
#!/bin/bash

# Все делаем из папки lab_!!!
# Проверка количества аргументов
if [ $# -ne 3 ]; then
    exit 1
fi
```

```
file_in=$1
file_out=$2
file_args=$3

# Коды ошибок
test_pass="0"
test_failed="1"

flag=$(head -n 1 "$file_args")
file_substr=$(sed '2q;d' "$file_args")

# Если передали входной и выходной файлы,
# то передаём их исполняемому файлу *.exe

touch ./func_tests/scripts/prog_pos.out
prog="./func_tests/scripts/prog_pos.out"

# Переводим текстовые файлы в бинарные
sh ./t2b.sh "$file_in" "$file_in"
# Если не флаг печати переводим file_out
if [[ "$flag" != "fb" ]]; then
      sh ./t2b.sh "$file_out" "$file_out"
fi

if [[ "$flag" == "ab" ]]; then
      touch ./func_tests/data/add.out
      add_entry="./func_tests/data/add.out"
      # Вытаскиваем запись, которую нужно добавить из file_args
      sed -n '3,7p' "$file_args" > "$add_entry"
```

```
	command="./*.exe ${flag} ${file_in}"
	$command < "$add_entry" > "$prog"
else
	command="./*.exe ${flag} ${file_in} ${file_substr}"
	$command > "$prog"
fi

return_code="$?"

# Проверка завершения программы
if [ "$return_code" -ne 0 ]; then		# не нулевой код ошибки
	sh ./b2t.sh "$file_in" "$file_in"
	# Если не флаг печати переводим file_out
	if [[ "$flag" != "fb" ]]; then
		sh ./b2t.sh "$file_out" "$file_out"
	fi
	exit "$test_failed"
fi

if [[ "$flag" == "ab" ]]; then
	# сравниваем выходные данные программы и данные в тесте
	if sh ./func_tests/scripts/comparator.sh "$file_in" "$file_out";
then	# неверный тест
		# Переводим бинарные файлы в текстовые
		sh ./b2t.sh "$file_in" "$file_in"
		sh ./b2t.sh "$file_out" "$file_out"
		exit "$test_failed"
	else
	# верный тест
		# Переводим бинарные файлы в текстовые
		sh ./b2t.sh "$file_in" "$file_in"
		sh ./b2t.sh "$file_out" "$file_out"
```

```
            exit "$test_pass"
      fi
else
      # сравниваем выходные данные программы и данные в тесте
      if sh ./func_tests/scripts/comparator.sh "$prog" "$file_out";
then        # неверный тест
            # Переводим бинарные файлы в текстовые
            sh ./b2t.sh "$file_in" "$file_in"
            # Если не флаг печати переводим file_out
            if [[ "$flag" != "fb" ]]; then
                  sh ./b2t.sh "$file_out" "$file_out"
            fi
            exit "$test_failed"
      else
      # верный тест
            # Переводим бинарные файлы в текстовые
            sh ./b2t.sh "$file_in" "$file_in"
            # Если не флаг печати переводим file_out
            if [[ "$flag" != "fb" ]]; then
                  sh ./b2t.sh "$file_out" "$file_out"
            fi
            exit "$test_pass"
      fi
fi
```

9. Скрипт neg_case.sh

```
#!/bin/bash

# Все делаем из папки lab_!!!
# Проверка количества переданных файлов
if [ $# -ne 2 ]; then
      exit 1
```

```
fi

file_in=$1
file_args=$2

# Коды ошибок
test_pass="0"
test_failed="1"

flag=$(head -n 1 "$file_args")
file_substr=$(sed '2q;d' "$file_args")

# Если передали входной и выходной файлы,
# то передаём их исполняемому файлу *.exe

touch ./func_tests/scripts/prog_neg.out
prog="./func_tests/scripts/prog_neg.out"

# Переводим текстовые файлы в бинарные
sh ./t2b.sh "$file_in" "$file_in"

if [[ "$flag" == "ab" ]]; then
      touch ./func_tests/data/add.out
      add_entry="./func_tests/data/add.out"
      # Вытаскиваем запись, которую нужно добавить из file_args
      sed -n '3,7p' "$file_args" > "$add_entry"

      command="./*.exe ${flag} ${file_in}"
      $command < "$add_entry" > "$prog"
else
      command="./*.exe ${flag} ${file_in} ${file_substr}"
```

```
      $command > "$prog"
fi

return_code="$?"

# Переводим бинарные файлы в текстовые
sh ./b2t.sh "$file_in" "$file_in"
# Проверка завершения программы
if [ "$return_code" -ne 0 ]; then          # не нулевой код ошибки возврата
      exit "$test_pass"                    # верный тест
else
      exit "$test_failed"                  # неверный тест
fi
```

10. Скрипт func_tests.sh

Скрипт последовательно вызывает pos_case.sh, neg_case.sh и передает им все входные и выходные тестовые файлы (позитивные и негативные соответственно). А также выводит дополнительную информацию о пройденных/проваленных тестах.

```
#!/bin/bash

# Количество ошибочных тестов
count_err=0
pos=0
neg=0

# Коды ошибок
test_pass="0"

# Позитивные тесты
```

```
files="./func_tests/data/pos_??_in.txt"
for file_in in $files; do

      # Вытаскиваем номер теста
      number=$(echo "$file_in" | grep -o "[0-9]*")

      # Проверка на наличие тестов(-z длина строки = 0)
      if [ -z "$number" ]; then
            break
      fi
      # Флаг наличия поз. тестов
     pos=1

      # Название выходного тестового файла
      file_out="./func_tests/data/pos_""$number""_out.txt"
      # Название файла с аргументами
      file_args="./func_tests/data/pos_""$number""_args.txt"

      tmp="./tmp.out"
      cp -r "$file_in" "$tmp"
      # Выходной файл существует => передаем входной и выходной файлы в
pos_case.sh
      # Не существует, то тест провален, переходим к следующему тесту

      if [ -f "$file_out" ]; then
            command="sh ./func_tests/scripts/pos_case.sh ""$file_in
""$file_out ""$file_args"
      else
            echo "POS_""$number"": FAILED"
            count_err=$((count_err + 1))
            continue
      fi
```

```
	$command
	return_code="$?"

	# Результат в соответствии с кодом возврата ./pos_case.sh
	if [ "$return_code" = "$test_pass" ]; then
		echo "POS_""$number"": PASSED"
	fi
	if [ "$return_code" != "$test_pass" ]; then
		echo "POS_""$number"": FAILED"
		count_err=$((count_err + 1))
	   pos=$((pos + 1))
	fi

	cp -r "$tmp" "$file_in"
done

# Негативные тесты
files="./func_tests/data/neg_??_in.txt"
for file_in in $files; do

	# находим номер теста
	number=$(echo "$file_in" | grep -o "[0-9]*")

	# проверка на наличие тестов(-z длина строки = 0)
	if [ -z "$number" ]; then
		break
	fi
	# Флаг наличия нег. тестов
    neg=1
```

```
	# Название файла с аргументами
	file_args="./func_tests/data/neg_""$number""_args.txt"
	# Передаем входной тестовый файл в ./neg_case.sh
	command="sh ./func_tests/scripts/neg_case.sh ""$file_in
""$file_args"

	$command
	return_code="$?"

	# Результат в соответствии с кодом возврата ./neg_case.sh
	if [ "$return_code" = "$test_pass" ]; then
		echo "NEG_""$number"": PASSED"
	fi
	if [ "$return_code" != "$test_pass" ]; then
		echo "NEG_""$number"": FAILED"
		count_err=$((count_err + 1))
	fi
done

# Дополнительная информация

if [ "$count_err" = 0 ]; then
	echo "All tests passed."
else
	echo "Failed $count_err tests."
fi

if [ "$pos" = 0 ]; then
	echo "No positive tests."
fi

if [ "$neg" = 0 ]; then
```

```
      echo "No negative tests."
fi


exit "$count_err"
```

11. Скрипт сборки с утилитой gcov

```
#!/bin/bash


# Компиляция
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -g3 --coverage main.c func.c add_entry.c sort.c
print_substr.c


# Комановка(-o название исполняемого файла)
gcc -o main_gcov.exe main.o func.o add_entry.o sort.o print_substr.o -
-coverage –lm
```

12. Скрипт - результат покрытия кода

```
#!/bin/bash


gcov main.c func.c add_entry.c sort.c print_substr.c
```

13. Скрипт проверки shellcheck-ом

```
#!/bin/bash


this_path="./*.sh"
scripts_path="./func_tests/scripts/*.sh"


for file in $this_path $scripts_path ; do
      shellcheck "$file"
done
```

14. Скрипт tests.sh

```
#!/bin/bash


sh ./func_tests/scripts/func_tests.sh
```

15. Скрипт go.sh

Скрипт последовательно запускает:

1. Скрипт, который формирует исполняемый файл
2. Скрипт, который проводит все позитивные и негативные тесты программы
3. Скрипт, который очищает ненужные файлы

Так проходит по сборкам со всеми санитайзерами, релизную и отладочную сборки, а также сборку с утилитой gcov.

```
#!/bin/bash


echo Result testing build_debug:
sh ./build_debug.sh
sh ./func_tests/scripts/func_tests.sh
sh ./clean.sh
echo ""

echo Result testing build_release:
sh ./build_release.sh
sh ./func_tests/scripts/func_tests.sh
sh ./clean.sh
echo ""

echo Result testing build_debug_asan:
sh ./build_debug_asan.sh
sh ./func_tests/scripts/func_tests.sh
sh ./clean.sh
```

```
echo ""

echo Result testing build_debug_msan:
sh ./build_debug_msan.sh
sh ./func_tests/scripts/func_tests.sh
sh ./clean.sh
echo ""

echo Result testing build_debug_ubsan:
sh ./build_debug_ubsan.sh
sh ./func_tests/scripts/func_tests.sh
sh ./clean.sh
echo ""

echo Result gcov:
sh ./build_gcov.sh
sh ./func_tests/scripts/func_tests.sh
sh ./collect_coverage.sh
sh ./clean.sh
echo ""
echo Temporary files removed
echo ""
```

16. Скрипт b2t.sh

Скрипт формирует исполняемый файл, для перевода бинарного файла в текстовый, и запускает его с переданными аргументами (исходный файл и конечный файл).

```
#!/bin/bash

# Проверка количества аргументов
if [ $# -ne 2 ]; then
    exit 1
fi
```

```
sh ./func_tests/b2t/build_b2t.sh
touch ./func_tests/b2t/tmp1.out
tmp_1="./func_tests/b2t/tmp1.out"
echo -n > "$tmp_1"
touch ./func_tests/b2t/tmp2.out
tmp_2="./func_tests/b2t/tmp2.out"
echo -n > "$tmp_2"

cp -r "$1" "$tmp_1"
./func_tests/b2t/b2t.exe "$tmp_1" "$tmp_2"
cp -r "$tmp_2" "$2"

if [[ -f "$tmp_1" ]]; then
    rm "$tmp_1"
fi
if [[ -f "$tmp_2" ]]; then
    rm "$tmp_2"
fi
```

17. Скрипт сборки b2t.exe

```
#!/bin/bash

# Компиляция
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -g3 ./func_tests/b2t/b2t.c

# Комановка(-o название исполняемого файла)
gcc -o ./func_tests/b2t/b2t.exe b2t.o -lm
```

18. Скрипт t2b.sh

Скрипт формирует исполняемый файл, для перевода текстового файла в бинарный, и запускает его с переданными аргументами (исходный файл и конечный файл).

```
#!/bin/bash

# Проверка количества аргументов
if [ $# -ne 2 ]; then
    exit 1
fi

sh ./func_tests/t2b/build_t2b.sh
touch ./func_tests/t2b/tmp1.out
tmp_1="./func_tests/t2b/tmp1.out"
touch ./func_tests/t2b/tmp2.out
tmp_2="./func_tests/t2b/tmp2.out"

cp -r "$1" "$tmp_1"
./func_tests/t2b/t2b.exe "$tmp_1" "$tmp_2"
cp -r "$tmp_2" "$2"

if [[ -f "$tmp_1" ]]; then
    rm "$tmp_1"
fi
if [[ -f "$tmp_2" ]]; then
    rm "$tmp_2"
fi
```

19. Скрипт сборки t2b.exe

```
#!/bin/bash

# Компиляция
```

```
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -g3 ./func_tests/t2b/t2b.c


# Комановка(-o название исполняемого файла)

gcc -o ./func_tests/t2b/t2b.exe t2b.o -lm
```