

Задание №1 часть 2 в рамках вычислительного практикума

Автоматизация функционального тестирования

Содержание

Условия задач	2
Описание скриптов	3
Заключение	16

Цель работы: автоматизация процессов сборки и тестирования.

Условия задач

1. Реализовать скрипты отладочной и релизной сборки
2. Реализовать скрипты отладочной сборки с санитайзерами.
3. Реализовать скрипт очистки побочных файлов.
4. Реализовать компаратор для сравнения последовательностей действительных чисел, располагающихся в двух текстовых файлах, с игнорированием остального содержимого
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.

Описание скриптов:

1. Скрипты отладочной и релизной сборки:

build_release.sh

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

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

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

- -std=c99: указывает компилятору GCC использовать стандарт языка C99
- -Wall: включает все предупреждения компилятора
- -Werror: превращает все предупреждения компилятора в ошибки
- -Wpedantic: включает строгий режим соответствия стандарту
- -Wfloat-equal: предупреждает о сравнении вещественных чисел на равенство с помощью оператора ==
- -Wfloat-conversion: Этот ключ предупреждает о потере точности при неявном преобразовании вещественных чисел.
- -Wextra: включает дополнительные предупреждения, которые не включены в -Wall, но являются полезными для обнаружения потенциальных проблем.
- -c: Этот ключ указывает компилятору создать только объектный файл (main.o) без создания исполняемого файла.
- -o app.exe: Этот ключ указывает компилятору GCC использовать имя app.exe для создаваемого исполняемого файла.
- -lm: Этот ключ указывает компилятору линковать с библиотекой математических функций, которая обычно содержит функции, такие как sin, cos, sqrt и т. д.

build_debug.sh

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

```
gcc -std=c99 -Wall -Werror -Wpedantic -Wfloat-equal -Wfloat-conversion -Wextra -  
Wvla -c -O0 -g3 *.c
```

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

Здесь, помимо прошлых ключей были добавлены:

- -O0: отключает все уровни оптимизации компилятора, что облегчает отслеживание ошибок.
- -g3: указывает компилятору GCC включить максимальное количество отладочной информации в объектные файлы.

build_gcov.sh

```
#!/bin/bash

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

Скрипт `build_gcov.sh` предназначен для сборки программы `app.exe` с поддержкой измерения покрытия кода тестами. Он компилирует все файлы с расширением `.c` в текущей директории с включением флага покрытия `-coverage`. Это позволяет собирать информацию о покрытии кода тестами с помощью утилиты `gcov`.

collect_coverage.sh

```
#!/bin/bash

SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

"$SCRIPT_DIR/build_gcov.sh
"$SCRIPT_DIR/func_tests/scripts/func_tests.sh
echo

gcov main.c

exit 0
```

Скрипт `collect_coverage.sh` предназначен для сбора информации о покрытии кода тестами и её анализа. Он вызывает сначала скрипт `build_gcov.sh` для сборки исполняемого файла с поддержкой измерения покрытия кода, а затем запускает функциональные тесты с помощью скрипта `func_tests.sh`. После завершения тестирования скрипт `gcov` анализирует информацию о покрытии, собранную для файла `main.c`, и выводит результаты на экран.

2. Скрипты отладочной сборки с санитайзерами

build_debug_asan.sh

Обнаруживает ошибки:

- Выход за пределы локального/глобального/динамического массива.

- Неверное использование локальных переменных.

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

```
clang -std=c99 -Wall -Werror -Wpedantic -Wfloat-equal -Wfloat-conversion -
Wextra -Wvla -g -fno-omit-frame-pointer -fsanitize=address -c *.c
clang -fno-omit-frame-pointer -fsanitize=address -o app.exe *.o -lm
```

build_debug_msan.sh

Обнаруживает ошибки:

- Использование не инициализированных локальных и «динамических» переменных.

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

```
clang -std=c99 -Wall -Werror -Wpedantic -Wfloat-equal -Wfloat-conversion -
Wextra -Wvla -g -fno-omit-frame-pointer -fsanitize=memory -fPIE -c *.c
clang -fno-omit-frame-pointer -fsanitize=memory -pie -o app.exe *.o -lm
```

build_debug_ubsan.sh

Обнаруживает ошибки:

- Различные виды неопределенного поведения

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

```
clang -std=c99 -Wall -Werror -Wpedantic -Wextra -Wvla -g -fno-omit-frame-
pointer -fsanitize=undefined -c *.c
clang -fno-omit-frame-pointer -fsanitize=undefined -o app.exe *.o -lm
```

run_all_sanitizers.sh

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

```
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")
```

```
# Запускаю программы с каждым санитайзером
```

```
echo "=====
```

```
echo "Running AddressSanitizer"
```

```
"$SCRIPT_DIR"/build_debug_asan.sh
```

```
"$SCRIPT_DIR"/func_tests/scripts/func_tests.sh
```

```
echo
```

```
echo "=====
```

```
echo "Running MemorySanitizer"
```

```
"$SCRIPT_DIR"/build_debug_msan.sh
"$SCRIPT_DIR"/func_tests/scripts/func_tests.sh

echo
echo "=====
echo "Running UndefinedBehaviorSanitizer"
"$SCRIPT_DIR"/build_debug_ubsan.sh
"$SCRIPT_DIR"/func_tests/scripts/func_tests.sh

echo "Cleaning files"
"$SCRIPT_DIR"/clean.sh
```

Этот скрипт предназначен для запуска функциональных тестов программы с различными санитайзерами - AddressSanitizer, MemorySanitizer и UndefinedBehaviorSanitizer.

Он начинается с определения пути к директории скрипта. Затем запускаются три последовательные команды, каждая из которых запускает сборку программы с соответствующим санитайзером (AddressSanitizer, MemorySanitizer, UndefinedBehaviorSanitizer) и после этого запускает функциональные тесты.

После выполнения всех тестов скрипт вызывает скрипт `clean.sh`, предположительно для очистки временных файлов и результатов тестирования.

3. Скрипт очистки побочных файлов.

clean.sh

```
#!/bin/bash

files="./func_test/scripts/*.txt *.txt *.exe *.o *.out *.gc*"

for file in $files; do
    rm -f "$file"
done
```

Удаляет указанные файлы из директорий

4. Компаратор для сравнения содержимого двух текстовых файлов

comparator.sh

```
#!/bin/bash

if [ ! -f "$2" ]; then
    exit 2
fi

# Извлечение содержимого из файла 1
output_a=$(cat "$1")

# Извлечение содержимого из файла 2
output_b=$(cat "$2")

# Сравнение содержимого файлов
if [ "$output_a" != "$output_b" ]; then
    exit 1
fi

exit 0
```

5. Реализовать скрипт pos_case.sh для проверки позитивного тестового случая по определённым далее правилам.

pos_case.sh

```
#!/bin/bash

# Проверка аргументов командной строки
if [ $# -lt 2 ]; then
    echo "Usage: $0 input_file expected_file [args_file]" >&2
    exit 2
fi

input_file=$1
expected_file=$2
args_file=${3:-}

# Проверка наличия входных файлов
if [ ! -f "$input_file" ]; then
```

```

    echo "Input file '$input_file' does not exist" >&2
    exit 2
fi

if [ ! -f "$expected_file" ]; then
    echo "Expected file '$expected_file' does not exist" >&2
    exit 2
fi

# Получение пути к каталогу скрипта
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")
output_file="$SCRIPT_DIR/../../output.txt"
app="$SCRIPT_DIR/../../app.exe"

# Проверка наличия исполняемого файла приложения
if [ ! -f "$app" ]; then
    echo "Application '$app' not found" >&2
    exit 2
fi

# Проверка, есть ли файл с аргументами
if [ -n "$args_file" ]; then
    args=$(cat "$args_file")
    binary_mode=false

# Проверка, есть ли в аргументах бинарный файл
for arg in "${args[@]"; do
    if [[ $arg == *.bin ]]; then
        binary_mode=true
        bin_file=$arg
        if [[ $arg == *pos_*_in.bin ]]; then
            # Выполняю импорт данных
            "$app" import "$input_file" "$bin_file"
            #echo "we dont know"
            if [ $? -ne 0 ]; then
                exit 1
            fi
        fi
    fi
done

# Запуск приложения
if $binary_mode; then
    "$app" "${args[@]}" > "$output_file"
    if [ $? -ne 0 ]; then
        exit 1
    fi

# Выполняю экспорт данных, если был бинарный файл
if [ -n "$bin_file" ]; then
    "$app" export "$bin_file" "$output_file"
    if [ $? -ne 0 ]; then

```



```

        exit 1
    fi
fi
else
"$app" ${args[@]} < "$input_file" > "$output_file"
if [ $? -ne 0 ]; then
    echo "eqeq"
    exit 1
fi
fi
else
"$app" < "$input_file" > "$output_file"
if [ $? -ne 0 ]; then
    exit 1
fi
fi

# Запуск компаратора
comparator="$SCRIPT_DIR/comparator.sh"
if [ ! -f "$comparator" ]; then
    echo "Comparator script '$comparator' not found" >&2
    exit 2
fi

"$comparator" "$output_file" "$expected_file"
if [ $? -ne 0 ]; then
    exit 1
fi

exit 0

```

Скрипт предназначен для автоматизации тестирования программы app.exe, включая обработку бинарных файлов. Он принимает три аргумента: input_file, expected_file, и необязательный args_file. Сначала он проверяет наличие указанных файлов. Затем определяет путь к исполняемому файлу app.exe и путь к директории скрипта. Если указан файл с аргументами, скрипт выполняет импорт данных из бинарных файлов, если они есть, и запускает приложение с этими аргументами. После выполнения приложения скрипт также, в случае наличия бинарных файлов в аргументах, выполняет экспорт данных из бинарного файла в выходной текстовый. В конце скрипт вызывает компаратор, в который передает выходной и ожидаемый файл, проверит успешность его выполнения и завершает работу с соответствующим кодом возврата.

6. Реализовать скрипт neg_case.sh для проверки негативного тестового случая по определённым далее правилам.

neg_case.sh

```
#!/bin/bash

# Получаю путь к каталогу скрипта
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

# Проверка наличия аргументов (хотя бы одного)
if [ $# -lt 1 ]; then
    echo "Usage: $0 input_file [args_file]" >&2
    exit 3
fi

input_file=$1
args_file=$2

# Проверка существования входного файла
if [ ! -f "$input_file" ]; then
    echo "Input file '$input_file' does not exist" >&2
    exit 2
fi

# Получаю абсолютный путь к файлу app.exe
app_path="$SCRIPT_DIR/../../app.exe"

# Проверка существования файла app.exe
if [ ! -f "$app_path" ]; then
    echo "app.exe not found at $app_path" >&2
    exit 2
fi

# Чтение аргументов из файла, если он существует
args=""
if [ -n "$args_file" ] && [ -f "$args_file" ]; then
    args=$(cat "$args_file")
fi

# Запуск программы с аргументами и перенаправление
# ввода и вывода
if ! "$app_path" $args < "$input_file" > /dev/null; then
    exit 0 # Ожидается ошибка, возвращаем ноль в случае
# успеха
```

```
fi
```

```
# Возврат ненулевого кода в случае успешного  
завершения (что не ожидается)  
exit 1
```

Скрипт `neg_case.sh` предназначен для тестирования отрицательных сценариев работы программы `app.exe`. Он принимает как минимум один аргумент - `input_file`, и необязательный `args_file`. Сначала скрипт проверяет наличие указанных файлов. Затем определяет путь к исполняемому файлу `app.exe` и путь к директории скрипта. Если указан файл с аргументами, скрипт считывает их из этого файла. После этого запускается программа `app.exe` с указанными аргументами и входным файлом. Если программа успешно завершает свою работу, скрипт возвращает код выхода 1, что не соответствует ожиданиям для отрицательного сценария, в противном случае возвращает 0.

7. Обеспечить автоматизацию функционального тестирования

func_tests.sh

```
#!/bin/bash

VERBOSE=false
total_positive_tests=0
total_negative_tests=0
positive_tests_passed=0
negative_tests_passed=0

# Проверяю, задан ли не тихий режим
if [[ "$1" == "-v" ]]; then
    VERBOSE=true
fi

# Получаю путь к каталогу скрипта
SCRIPT_DIR=$(dirname "$(readlink -f "$0")")

# Функция для запуска позитивных тестов
run_positive_tests() {
    if $VERBOSE; then
        echo "-----"
        echo "RUNNING POSITIVE TESTS"
        echo
    fi

    for in_file in "$SCRIPT_DIR"/../data/pos_*_in.txt; do
        # Проверяю, существует ли соответствующий файл с ожидаемым
        # выходом
        out_file="${in_file/_in/_out}"
        if [ ! -f "$out_file" ]; then
            continue
        fi

        # Проверяю, существует ли соответствующий файл с аргументами
        ((total_positive_tests++))
        args_file="${in_file/_in/_args}"
        if [ -f "$args_file" ]; then
            if "$SCRIPT_DIR"/pos_case.sh "$in_file" "$out_file" "$args_file";
        then
            ((positive_tests_passed++))
            if $VERBOSE; then
                echo -e "${(basename "$in_file")}: \e[32mPASSED\e[0m"
            fi
        else
            if $VERBOSE; then
                echo -e "${(basename "$in_file")}: \e[31mFAILED\e[0m"
            fi
            echo
        fi
    done
}
```

```

        echo -e "Input data:\n$(sed 's/^\\t/' "$in_file")"
        echo -e "Expected output:\n$(sed 's/^\\t/' "$out_file")"
        echo -e "Actual output:\n$(sed 's/^\\t/' output.txt)"
        echo
    fi
fi
else
    if "$SCRIPT_DIR"/pos_case.sh "$in_file" "$out_file"; then
        ((positive_tests_passed++))
        if $VERBOSE; then
            echo -e "$(basename "$in_file"): \\e[32mPASSED\\e[0m"
        fi
    else
        if $VERBOSE; then
            echo -e "$(basename "$in_file"): \\e[31mFAILED\\e[0m"
            echo
            echo -e "Input data:\n$(sed 's/^\\t/' "$in_file")"
            echo -e "Expected output:\n$(sed 's/^\\t/' "$out_file")"
            echo -e "Actual output:\n$(sed 's/^\\t/' output.txt)"
            echo
        fi
    fi
fi
done
}

# Функция для запуска негативных тестов
run_negative_tests() {
    if $VERBOSE; then
        echo "-----"
        echo "RUNNING NEGATIVE TESTS"
        echo
    fi

    for in_file in "$SCRIPT_DIR"/../data/neg_*_in.txt; do
        ((total_negative_tests++))
        args_file="{in_file/_in/_args}"
        if [ -f "$args_file" ]; then
            if "$SCRIPT_DIR"/neg_case.sh "$in_file" "$args_file"; then
                ((negative_tests_passed++))
                if $VERBOSE; then
                    echo -e "$(basename "$in_file"): \\e[32mPASSED\\e[0m"
                fi
            else
                if $VERBOSE; then
                    echo -e "$(basename "$in_file"): \\e[31mFAILED\\e[0m"
                    echo
                    echo -e "Input data:\n$(sed 's/^\\t/' "$in_file")"
                    echo -e "Actual output:\n$(sed 's/^\\t/' output.txt)"
                    echo
                fi
            fi
        fi
    fi
}

```

```

else
    if "$SCRIPT_DIR"/neg_case.sh "$in_file"; then
        ((negative_tests_passed++))
        if $VERBOSE; then
            echo -e "$(basename "$in_file"): \e[32mPASSED\e[0m"
        fi
    else
        if $VERBOSE; then
            echo -e "$(basename "$in_file"): \e[31mFAILED\e[0m"
            echo
            echo -e "Input data:\n$(sed 's/^\\t/' "$in_file")"
            echo -e "Actual output:\n$(sed 's/^\\t/' output.txt)"
            echo
        fi
    fi
fi
done
}

# Запуск ПОЗИТИВНЫХ ТЕСТОВ
positive_tests_exist=false
for file in "$SCRIPT_DIR"/../data/pos*_in.txt; do
    if [ -f "$file" ]; then
        positive_tests_exist=true
        break
    fi
done

if $positive_tests_exist; then
    run_positive_tests
else
    if $VERBOSE; then
        echo '-----'
        echo "Positive tests were not found"
        echo
    fi
fi

# Запуск НЕГАТИВНЫХ ТЕСТОВ
negative_tests_exist=false
for file in "$SCRIPT_DIR"/../data/neg*_in.txt; do
    if [ -f "$file" ]; then
        negative_tests_exist=true
        break
    fi
done

if $negative_tests_exist; then
    run_negative_tests
else
    if $VERBOSE; then
        echo '-----'

```

```

        echo "Negative tests were not found"
        echo
    fi
fi

# Вывод статистики
echo "======"
echo "TESTING IS OVER"
echo "STATS DOWN BELOW"
echo "======"
echo
echo "Total positive tests: $total_positive_tests"
if [ "$positive_tests_passed" -eq "$total_positive_tests" ]; then
    echo -e "Positive tests passed:
\e[32m$positive_tests_passed/$total_positive_tests\e[0m"
else
    echo -e "Positive tests passed:
\e[31m$positive_tests_passed\e[0m/\e[32m$total_positive_tests\e[0m"
fi
echo "-----"
echo
echo "Total negative tests: $total_negative_tests"
if [ "$negative_tests_passed" -eq "$total_negative_tests" ]; then
    echo -e "Negative tests passed:
\e[32m$negative_tests_passed/$total_negative_tests\e[0m"
else
    echo -e "Negative tests passed:
\e[31m$negative_tests_passed\e[0m/\e[32m$total_negative_tests\e[0m"
fi
echo "-----"
echo
echo "======"
if [ "$((positive_tests_passed + negative_tests_passed))" -eq
"$((total_negative_tests + total_positive_tests))" ]; then
    echo -e "\e[32mALL TESTS HAVE BEEN PASSED\e[0m"
else
    echo -e "\e[31mNOT ALL TESTS HAVE BEEN PASSED\e[0m"
fi
echo "======"
echo

```

Скрипт `func_tests.sh` предназначен для запуска функциональных тестов программы `app.exe`, включая позитивные и негативные сценарии, с возможностью вывода подробной информации о процессе выполнения тестов.

Он начинается с проверки наличия аргумента `-v` для активации режима вывода подробной информации. Затем определяет путь к директории, в которой располагается скрипт.

Функция `run_positive_tests` запускает позитивные тесты, проверяя наличие соответствующих входных и ожидаемых выходных файлов. Если аргументы заданы,

скрипт передает их в `pos_case.sh` для выполнения теста. Аналогично, функция `run_negative_tests` запускает негативные тесты.

После выполнения тестов выводится статистика: количество выполненных позитивных и негативных тестов, количество успешно пройденных тестов, и общий результат тестирования.

Заключение

Выполняя лабораторную работу, я научился автоматизировать процессы сборки и тестирования