Отчет по проектно-технологической практике

Задание 1

Пояснение всех скриптов

Оглавление

build_debug.sh

```
#!/bin/bash
gcc-13 -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-conversion
-Wfloat-equal -g3 -c main.c
gcc-13 main.o -o app.exe -lm
```

Скрипт build_debug.sh предназначен для компиляции программы из исходного кода на языке C с параметрами, настроенными для создания отладочной версии приложения.

Компилятор gcc-13 используется потому что на MACOS используется gcc под капотом у которого clang.

-lm опциональный ключ, добавляется когда используется библиотека math.h.

build_debug_asan.sh

```
#!/bin/bash
clang -fsanitize=address -fno-omit-frame-pointer -std=c99 -Wall -
Werror -Wpedantic -Wextra -Wfloat-conversion -Wfloat-equal -g main.c
-o app.exe
```

Скрипт build_debug_asan.sh используется для компиляции программы из исходного кода на языке C с помощью компилятора Clang с включенным AddressSanitizer (ASan) для обнаружения ошибок в работе с памятью.

build_debug_msan.sh

```
#!/bin/bash
clang -fsanitize=memory -fPIE -pie -fno-omit-frame-pointer -std=c99 -
Wall -Werror -Wpedantic -Wextra -Wfloat-conversion -Wfloat-equal -g
main.c -o app.exe
```

Скрипт build_debug_msan.sh предназначен для компиляции программы из исходного кода на языке C с помощью компилятора Clang с включенным MemorySanitizer (MSan) для обнаружения ошибок в работе с памятью.

Описание такое же как и для **ASAN**, но все таки между ними есть отличие:
Главное отличие между ними заключается в том, что **MSAN** сосредоточен на обнаружении неинициализированной памяти, тогда как **ASAN** более широко ориентирован и обнаруживает ряд других проблем, таких как переполнение буфера и утечки памяти.

build_debug_ubsan.sh

```
#!/bin/bash
clang -fsanitize=undefined -fno-omit-frame-pointer -std=c99 -Wall -
Werror -Wpedantic -Wextra -Wfloat-conversion -Wfloat-equal -g main.c
-o app.exe
```

Скрипт build_debug_ubsan.sh предназначен для компиляции программы из исходного кода на языке C с помощью компилятора Clang с включенным UndefinedBehaviorSanitizer (UBSan) для обнаружения неопределенного поведения в программе.

build_release.sh

```
#!/bin/bash
gcc-13 -std=c99 -Werror -Wall -Wpedantic -Wextra -Wfloat-conversion -
Wfloat-equal -c main.c
gcc-13 main.o -o app.exe -lm
```

Скрипт build_release.sh используется для компиляции программы из исходного кода на языке C с помощью компилятора GCC в релизную версию, оптимизированную для производительности.

collect_coverage.sh

```
#!/bin/bash
gcc-13 main.c --coverage -o app.exe -lm
./func_tests/scripts/func_tests.sh
gcov main.c
```

Скрипт collect_coverage.sh используется для сбора данных о покрытии кода тестами

clean.sh

```
#!/bin/bash

rm -f ./func_tests/scripts/*.txt
rm -f ./*.txt
rm -f ./*.exe
rm -f ./*.o
rm -f ./*gcno
rm -f ./*gcda
rm -f ./*gcov
```

Скрипт clean.sh используется для очистки всех временных и побочных файлов

comparator.sh

```
#!/bin/bash

file1="$1"
file2="$2"

if [ -f "$file1" ] && [ -f "$file2" ]; then
    grep -oE "[+-]?[0-9]+([.][0-9]+)?" "$file1" > file_new_1.txt
    grep -oE "[+-]?[0-9]+([.][0-9]+)?" "$file2" > file_new_2.txt

    if diff file_new_1.txt file_new_2.txt; then
        exit 0
    else
        exit 1
    fi
else
    exit 1
fi
```

Скрипт comparator.sh сделан для того, чтобы сравнивать два файла, если файлы одинаковы (т.е имеют одно и то же содержимое), то скрипт возвращает 0, иначе 1. Так же в скрипте реализована проверка на то, что подаются файлы, если хотя бы один из аргументов не будет являться файлом, то скрипт вернет 1.

pos_case.sh

```
#!/bin/bash

working_dir=$(dirname "$0")
file_stream_in="$1"
file_stream_out_expected="$2"
executable="$working_dir/../../app.exe"
"$executable" < "$file_stream_in" > "result.txt"

if [ $# -eq 2 ] && [ -f "$file_stream_in" ] && [ -f
"$file_stream_out_expected" ]; then
   if "$working_dir/comparator.sh" "result.txt"
"$file_stream_out_expected" > /dev/null; then
        exit 0
    else
        exit 1
    fi

else
    exit 1
fi
```

Скрипт pos_case.sh предназначен для выполнения позитивного теста программы, который заключается в сравнении вывода программы с ожидаемым результатом. Если выход совпадает с ожидаемым, то скрипт вернет 0, иначе 1

neg_case.sh

```
#!/bin/bash

file_stream_in="$1"
working_dir=$(dirname "$0")

if [ $# -eq 1 ] && [ -f "$1" ]; then
    executable="$working_dir/../../app.exe"
    "$executable" < "$file_stream_in" > /dev/null

    if "$executable" < "$file_stream_in" > /dev/null; then
        exit 1
    else
        exit 0
    fi
else
    exit 1
fi
```

Скрипт neg_case.sh предназначен для выполнения негативного теста программы. Скрипт сравнивает выход программы на входном негативном тесте, если не 0, то возвращает 0, иначе 1.

func_tests.sh

```
#!/bin/bash

working_dir=$(dirname "$0")
failed_tests=0

for pos_test_file_in in "$working_dir/../data/pos_"*"_in.txt"; do
    test_num=$(basename "$pos_test_file_in" | cut -d '_' -f 2)
    pos_test_file_out="$working_dir/../data/pos_${test_num}_out.txt"
    "$working_dir/pos_case.sh" "$pos_test_file_in"
"$pos_test_file_out"
    pos_test_exit_code=$?
    if [ "$pos_test_exit_code" -eq 0 ]; then
        echo "Позитивный тест $test_num пройден"
    else
        echo "Позитивный тест $test_num не пройден"
        ((failed_tests++))
    fi
done

if [ ! -f "$working_dir/../data/neg_01_in.txt" ]; then
    exit 1
fi

for neg_test_file in "$working_dir/../data/neg_"*"_in.txt"; do
    test_num=$(basename "$neg_test_file" | cut -d '_' -f 2)
    "$working_dir/neg_case.sh" "$working_dir/../../app.exe" <
"$neg_test_file"
    neg_test_exit_code=$?
    if [ "$neg_test_exit_code" -eq 0 ]; then
        echo "Негативный тест $test_num пройден"
    else
        echo "Негативный тест $test_num не пройден"
        ((failed_tests++))
    fi
done

exit "$failed_tests"
```

Скрипт func_test предназначен для автоматизации запуска тестов для программа. Скрипт позволяет автоматически запускать наборы позитивных и негативных тестов и определять количество проваленных тестов.

sanitize_check.sh

```
#!/bin/bash

failed_tests=0
working_dir=$(dirname "$0")
app_path="$working_dir/app.exe"
tests_dir="$working_dir/func_tests/data"


asan_path="$working_dir/build_debug_asan.sh"
msan_path="$working_dir/build_debug_msan.sh"
ubsan_path="$working_dir/build_debug_ubsan.sh"


for sanitizer_path in "$asan_path" "$msan_path" "$ubsan_path"; do


    echo "Тестирование санитайзера: $sanitizer_path"

    # Извлекаем название санитайзера из пути
    sanitizer_name=$(basename "$sanitizer_path" | cut -d '_' -f 4 |
cut -d '.' -f 1)

    if eval "$sanitizer_path"; then

        for test_file in "$tests_dir/pos_"*"_in.txt"; do
            if ! "$app_path" < "$test_file" > /dev/null; then
                echo "Тест \"$test_file\" не пройден с санитайзером
\"$sanitizer_name\""
                ((failed_tests++))
            fi
        done
    else
        echo "Ошибка при сборке программы с санитайзером
\"$sanitizer_name\""
    fi
done


if [ "$failed_tests" -eq 0 ]; then
    echo "Все тесты пройдены успешно"
else
    echo "Не пройдено тестов: $failed_tests"
fi
```

Вспомогательный скрипт sanitize_check.sh предназначен для автоматизированного тестирования программы app.exe с различными санитайзерами (AddressSanitizer, MemorySanitizer, UndefinedBehaviorSanitizer).