Доколин Георгий ИУ7-22Б

Отчет по Проектно-технологической практике (тестированию, отладке и профилированию ПО)

Задание 1

Цель

Целью данной работы является автоматизация процессов сборки и тестирования.

Пояснение скриптов

1. build_debug.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wvla -Wfloat-conversion -Wfloat-equal -g3 -c main.c
gcc -o app.exe main.c -lm
```

Скрипт **build_debug.sh** используется для сборки программы с различными ключами, которые будут показывать ошибки допущенные в процессе разработки.
В последней строке этого скрипта создаются еще несколько файлов с различными расширениями.

2. build_debug_asan.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

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

Скрипт **build_debug_asan.sh** выполняет роль address sanitizer, который отлавливает ошибки выхода за пределы массива, а также ошибки неверного использования переменных.

3. build_debug_msan.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

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

Скрипт **build_debug_msan.sh** выполняет роль memory sanitizer , который отлавливает ошибки неиспользованных перменных.

4. build_debug_ubsan.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

clang -fsanitize=undefined -fno-omit-frame-pointer -std=c99 -Wall -
Werror -Wpedantic -Wextra -Wvla -Wfloat-conversion -Wfloat-equal
-g3 -c main.c -o main.o
clang main.o -o app.exe -lm -fsanitize=undefined
```

Скрипт **build_debug_ubssan.sh** выполняет роль undefined behavior sanitizer , который отлавливает различные виды неопределенного поведения.

5. build_release.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

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

Скрипт **build_release.sh** используется для сборки программы с различными ключами, которые будут показывать ошибки допущенные в процессе разработки, однако данная сборка проекта уже является предрелизной.
В последней строке этого скрипта создаются еще несколько файлов с различными расширениями.

6. clean.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

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

Скрипт **clean.sh** используется для очистки директории после работы, поскольку в процессе выполнения скрипта создаются файлы нужные только для его работы.

7. func_tests.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
flag=0
cd "$to_source" || exit
if [ "$1" = "-v" ]; then
  VERBOSE=true
else
  VERBOSE=false
fi
ls "$to_source/../data" > save_all.txt
count_files_in=$(grep -Ec 'pos_[0-9]{1,2}_in\.txt' "save_all.txt")
count_files_out=$(grep -Ec 'pos_[0-9]{1,2}_out\.txt' "save_all.txt")
if [ "$count_files_in" -eq "$count_files_out" ];
then
  if [ "$count_files_in" -ne 0 ];
    then
    for ((i=1; i < count_files_in + 1; i++ ))
    do
      if [ "$i" -lt 10 ];
      then
        data_in=$(cat "../data/pos_0${i}_in.txt")
        data_out=$(cat "../data/pos_0${i}_out.txt")
```

```
      bash pos_case.sh "$data_in" "$data_out"
      if [ $? -eq 1 ]; then
        if [ "$VERBOSE" = true ]; then
          echo "pos_0$i - ERROR"
        fi
        flag=$((flag + 1))
      else
        if [ "$VERBOSE" = true ]; then
          echo "pos_0$i - OK"
        fi
      fi
    else
      data_in=$(cat "../data/pos_${i}_in.txt")
      data_out=$(cat "../data/pos_${i}_out.txt")
      bash pos_case.sh "$data_in $data_out"
      if [ $? -eq 1 ]; then
        if [ "$VERBOSE" = true ]; then
          echo "pos_$i - ERROR"
        fi
        flag=$((flag + 1))
      else
        if [ "$VERBOSE" = true ]; then
          echo "pos_$i - OK"
        fi
      fi
    fi
  done
 fi
else
 exit 1
fi
count_files_in=$(grep -Ec 'neg_[0-9]{1,2}_in\.txt' "save_all.txt")
count_files_out=$(grep -Ec 'neg_[0-9]{1,2}_out\.txt' "save_all.txt")
if [ "$count_files_in" -eq "$count_files_out" ];
then
 if [ "$count_files_in" -ne 0 ];
 then
  for ((i=1; i < count_files_in + 1; i++ ))
  do
   if [ "$i" -lt 10 ];
   then
    data_in=$(cat "../data/neg_0${i}_in.txt")
    data_out=$(cat "../data/neg_0${i}_out.txt")
    bash neg_case.sh "$data_in" "$data_out"
    if [ $? -eq 1 ]; then
```

```
        if [ "$VERBOSE" = true ]; then
          echo "neg_0$i - ERROR"
        fi
        flag=$((flag + 1))
      else
        if [ "$VERBOSE" = true ]; then
          echo "neg_0$i - OK"
        fi
      fi
      else
         data_in=$(cat "../data/neg_${i}_in.txt")
         data_out=$(cat "../data/neg_${i}_out.txt")
         bash neg_case.sh "$data_in $data_out"
        if [ $? -eq 1 ]; then
          if [ "$VERBOSE" = true ]; then
            echo "neg_$i - ERROR"
          fi
          flag=$((flag + 1))
        else
           if [ "$VERBOSE" = true ]; then
             echo "neg_$i - OK"
           fi
        fi
   fi
   done
  fi
else
  flag=$((flag + 1))
fi
if [ "$flag" == 0 ]; then
  echo "TESTS - COMPLETE"
  exit 0
else
  echo "TESTS - FAIL"
  exit 1
fi
```

Скрипт **func_tests.sh** используется для того чтобы обработать все положительные и отридцательные тесты, передав входные и выходные файлы для тестирования в следующие скрипты.
Сначала создаются переменные, которые хранят в себе все входные и выходные файлы. Далее создаются перменные хранящие в себе количсества всех файлов.

В цикле скрипт проходит по каждому из файлов и передает их значения в **pos_case.sh.**
В конце обработки положительного теста, скрипт получает значение которое вернул **pos_case.sh,** в зависимости от этого ответа зависит код завершения скрипта.
В случае с негативными тестами, происходит все то же самое, только вызывается скрипт **neg_case.sh.**

8. pos_case.sh

```
#!/bin/bash
if [ $# != 2 ]; then
        echo Error neg_case
        exit 1
fi
echo "$1" > save_in.txt
echo "$2" > save_out.txt
touch save_res.txt
start_file=$(find "../../" -name "app.exe")
"$start_file" < save_in.txt > save_res.txt
bash comparator.sh save_res.txt save_out.txt
if  ! bash comparator.sh save_res.txt save_out.txt; then
        exit 1
fi
exit 0
```

Скрипт **pos_case.sh** используется для того чтобы запустить программу с входными данными и передать ее в скриипт **comparator.sh** для сравнивания выходных данных, заявленных пользователем и фактическим результатом работы программы.
В первых строках идет проверка на наличие двух переданных параметров. Далее происходит запись переданных данных в файлы для дальнейшего запуска компаратора.
Строки **(cd ../../; bash build_release.sh)** и **start_file=$(find "../../" -name "app.exe")** сначала запускают скрипт, который создает необходимые файлы для тестировани, а дальше уже запускается эта программа, с перенаправлением вывода в текстовый файл. Потом, запускается компаратор, в котором сравнивается файл с результатом работы программы и созданный пользователем выходной файл.

9. neg_case.sh

```
#!/bin/bash#!/bin/bash
if [ $# != 2 ]; then
        echo Error pos_case
        exit 1
fi
echo "$1" > save_in.txt
touch save_res.txt
start_file=$(find "../../" -name "app.exe")
"$start_file" < save_in.txt > save_res.txt
if [ $? -eq 1 ]; then
        exit 0
fi
exit 1
```

Скрипт **neg_case.sh** используется для того чтобы запустить программу с входными данными, при которых программа должна выдавать ошибку. Работа скрипта очень похожа на скрипт **pos_case.sh** за исключением того, что файлы не сравниваются друг с другом при помощи компаратора, а просто берется код завершениия работы и сравнивается с ожидаеммым результатом.

10. comparator.sh

```
#!/bin/bash

file_1=$1
file_2=$2

touch save_line_1.txt
touch save_line_2.txt
touch saver.txt
create_line=""

if [ -e "$file_1" ] && [ -e "$file_2" ]; then
        lines_1=$(cat "$file_1")
        lines_2=$(cat "$file_2")
        for line in $lines_1
        do
                create_line+="${line} "
        done
        echo "$create_line" 1> save_line_1.txt
```

```
        create_line=""
        for line in $lines_2
        do
                create_line+="${line} "
        done
        echo "$create_line" 1> save_line_2.txt
        diff save_line_1.txt save_line_2.txt 1> saver.txt
        if [ $? -ne 1 ]; then
                exit 0
        else
                exit 1
        fi
else
   exit 1
fi
```

Скрипт **comparator.sh** используется для того чтобы сравнить два текстовых файла, в которых хранятся ожидаемый результат работы и фактический. Программа считывает каждый из файлов построчно и записывает каждый из них в строчку, потом каждая из этих строк записывается в другой файл, которые сравниваются при помощи функции **diff.** По результату работы, прокрамма возращает соответствующий код.

11. sanitize_check.sh

```
#!/bin/bash

check_tests()
{
   if [ "$VERBOSE" = true ]; then
      echo "Check tests"
      ./func_tests/scripts/func_tests.sh -v
   else
      ./func_tests/scripts/func_tests.sh
   fi
}

if [ "$1" = "-v" ]; then
  VERBOSE=true
```

```
else
  VERBOSE=false
fi

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit

flag=true
if [ "$VERBOSE" = true ]; then
   echo "Check address sanitizer"
fi
bash build_debug_asan.sh
check_tests
if [ "$?" = "1" ]; then
   flag=false
fi

echo
"————————————————————————————————————————————————————————————"

if [ "$VERBOSE" = true ]; then
   echo "Check memory sanitizer"
fi
bash build_debug_msan.sh
check_tests
if [ "$?" = "1" ]; then
   flag=false
fi

echo
"————————————————————————————————————————————————————————————"

if [ "$VERBOSE" = true ]; then
   echo "Check undefinedBehavior sanitizer"
fi
bash build_debug_ubsan.sh
check_tests
if [ "$?" = "1" ]; then
   flag=false
```

```
fi

echo
"————————————————————————————————————————————————————————————"

if [ "$flag" = false ]; then
   echo "Sanitizers - ERROR"
else
   echo "Sanitizers - COMPLETE"
fi
```

Скрипт **sanitize_check.sh** используется для того, чтобы проверить программу при помощи различных санитайзеров, которые вызываются вместе с тестами последовательно и в случае ошибки, будет выведено сообщение, поясняющее что это за ошибка.

12. build_coverage.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit 1

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

Скрипт **build_coverage.sh** используется для того, чтобы собарть программу и подготовить исполняемый файл к проверке на покрытие кода.

13. collect_coverage.sh

```
#!/bin/bash

to_source=$(dirname "$(readlink -f "$0")")
cd "$to_source" || exit
"./clean.sh"
"./build_coverage.sh"
"./func_tests/scripts/func_tests.sh"
gcov ./main.c
```

Скрипт **collect_coverage.sh** используется для того, чтобы запустить тесты и проверить их на покрытие кода, предварительно очистив прошлые файлы.