Задание №1 в рамках вычислительного практикума

Автоматизация функционального тестирования

Содержание

Цель работы: автоматизация процессов сборки и тестирования.

Условия задач

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 последовательностей действительных чисел, располагающихся в двух текстовых файлах, с игнорированием остального содержимого
5. Реализовать компаратор для сравнения содержимого двух текстовых файлов, располагающегося после первого вхождения подстроки «Result: ␣».
6. Реализовать скрипт pos_case.sh для проверки позитивного тестового случая по определённым далее правилам.
7. Реализовать скрипт neg_case.sh для проверки негативного тестового случая по определённым далее правилам
8. 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
-c main.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 -c -O0 -g3 main.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.

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

gcc -std=c99 -Wall -Werror -Wpedantic -Wfloat-equal -Wfloat-conversion -Wextra -c -O0 -lm -g3 main.c
gcc -fsanitize=address -fno-omit-frame-pointer -g -o app.exe main.o
```

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 -c -O0 -lm -g3 main.c
clang -fsanitize=memory -fPIE -pie -fno-omit-frame-pointer -g -o app.exe main.o
```

build_debug_ubsan.sh

Обнаруживает ошибки:

• Различные виды неопределенного поведения

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

```
gcc -std=c99 -Wall -Werror -Wpedantic -Wfloat-equal -Wfloat-conversion -Wextra
-c -O0 -lm -g3 main.c
gcc -fsanitize=undefined -fno-omit-frame-pointer -g -o app.exe main.o
```

3. Скрипт очистки побочных файлов.

clean.sh

```
#!/bin/bash

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

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

Удаляет указанные файлы из директорий

4. Компаратор для сравнения последовательностей действительных чисел, располагающихся в двух текстовых файлах

comporator.sh

```
#!/bin/bash

if [ $# -ne 2 ]; then
   echo "Need 2 files to compare" >&2
   exit 3
fi

if [ ! -f "$1" ]; then
   echo "File '$1' does not exist" >&2
   exit 2
fi

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

value_1=$(grep -oP "[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)" "$1")
value_2=$(grep -oP "[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)" "$2")

if [ "$value_1" != "$value_2" ]; then
```

```
    exit 1
fi
exit 0
```

С помощью регулярного выражения из файлов 1 и 2 извлекаются вещественные числа и записываются в переменные value_1 и value_2 соответственно. Если содержимое этих переменных не совпадает, скрипт возвращает код выхода 1, что означает несовпадение. В противном случае скрипт возвращает код выхода 0, указывая на идентичность содержимого файлов.

5. **Реализовать компаратор для сравнения содержимого двух текстовых файлов, располагающегося после первого вхождения подстроки «Result: ␣».**

comparator.sh

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
    exit 1
fi

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

if [ ! -f "$file1" ] || [ ! -f "$file2" ]; then
    exit 1
fi
```

```
index=$(grep -m 1 -aob 'Result: ' "$file1" | cut -d: -f1)

if [ -n "$index" ]; then
   content_file1=$(tail -c +$(($index + 8)) "$file1" | tr -d '\r' | sed
's/^[[:space:]]*//;s/[[:space:]]*$//')
else
   exit 1
fi

index=$(grep -m 1 -aob 'Result: ' "$file2" | cut -d: -f1)

if [ -n "$index" ]; then
   content_file2=$(tail -c +$(($index + 8)) "$file2" | tr -d '\r' | sed
's/^[[:space:]]*//;s/[[:space:]]*$//')
else
   exit 1
fi

if [ "$content_file1" = "$content_file2" ]; then
   exit 0
else
   exit 1
fi
```

Скрипт сначала проверяет наличие двух аргументов командной строки и существование файлов. Затем он находит индекс первого вхождения подстроки "Result: " в каждом файле и извлекает содержимое, начиная с этой подстроки. После этого скрипт сравнивает содержимое двух файлов: если оно одинаково, скрипт завершается успешно (код 0), в противном случае - с ошибкой (код 1).

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

pos_case.sh

```
#!/bin/bash

# Проверка наличия аргументов
if [ $# -ne 2 ]; then
   echo "Need 2 files to compare" >&2
   exit 3
fi

if [ ! -f "$1" ]; then
```

```
    echo "File '$1' does not exist" >&2
    exit 2
fi

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

# Считывание аргументов
file_stream_in="$1"
file_stream_out_expect="$2"

# Запуск программы с входным файлом
if ! "../../app.exe" < "$file_stream_in" > current.txt; then
    exit 1
fi

# Запуск компаратора для сравнения вывода с эталонным файлом
if ! "./comparator.sh" "$file_stream_out_expect" "current.txt"; then
    exit 1
fi

# Возврат нулевого кода в случае успешного завершения
exit 0
```

Скрипт предназначен для проверки позитивных тестовых случаев. После считывания аргументов скрипт запускает программу app.exe, используя в качестве входного потока файл, переданный как первый аргумент. Результат выполнения программы записывается в файл current.txt. Затем скрипт запускает компаратор comparator.sh, чтобы сравнить содержимое файлов current.txt и file_stream_out_expect. Если вывод программы не соответствует ожидаемому выводу, скрипт завершается с кодом выхода 1. Иначе с кодом выхода 0.

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

neg_case.sh

```
#!/bin/bash

# Проверка наличия аргументов
if [ ! -f "$1" ]; then
   echo "File '$1' does not exist" >&2
   exit 2
fi

# Считывание аргументов
file_stream_in="$1"

# Запуск программы с входным файлом
"../../app.exe" < "$file_stream_in" > /dev/null

# Проверка кода завершения программы
if ! "../../app.exe" < "$file_stream_in" > /dev/null; then
   exit 0
fi

exit 1
```

Скрипт предназначен для проверки негативных тестовых случаев. Он проверяет, что при подаче входного файла file_stream_in программа app.exe завершается с ненулевым кодом возврата.

8. Обеспечить автоматизацию функционального тестирования

func_tests.sh

```
#!/bin/bash

# Проведение тестов позитивных случаев
input_files=$(ls ../data/pos_*_in.txt 2>/dev/null)
if [ -z "$input_files" ]; then
  echo "No positive tests"
fi

for test_file in $input_files; do
  test_number=$(basename "$test_file" | grep -oE '[0-9]+')
  output_file="../data/pos_${test_number}_out.txt"
  if ./pos_case.sh "$test_file" "$output_file"; then
    echo "POS_${test_number}: PASSED"
  else
    echo "POS_${test_number}: FAILED"
  fi
done

echo

# Проведение тестов негативных случаев
input_files=$(ls ../data/neg_*_in.txt 2>/dev/null)
if [ -z "$input_files" ]; then
  echo "No negative tests"
fi

for test_file in $input_files; do
  test_number=$(basename "$test_file" | grep -oE '[0-9]+')
  output_file="../data/neg_${test_number}_out.txt"
  if ./neg_case.sh "$test_file" "$output_file"; then
    echo "NEG_${test_number}: PASSED"
  else
    echo "NEG_${test_number}: FAILED"
  fi
done
```

Сначала скрипт ищет файлы с именами вида pos_*_in.txt в директории ../data/. Если таких файлов не найдено, выводится сообщение "No positive tests". Для каждого найденного входного файла скрипт запускает pos_case.sh, передавая входной файл и ожидаемый файл вывода. Результат выполнения скрипта pos_case.sh сохраняется. Если скрипт pos_case.sh завершился успешно (с кодом выхода 0), выводится сообщение "POS_${test_number}: PASSED". В противном случае выводится сообщение "POS_${test_number}: FAILED". Аналогично для негативных тестов

Заключение

Выполняя лабораторную работу, я научился автоматизировать процессы сборки и тестирования