Смирнов Иван ИУ7-22Б - 2023г.

Отчет

Задание №1.2

Автоматизация функционального тестирования

Целью данной работы является автоматизация процессов сборки и тестирования.

Задание:

Рассмотрим реализованные скрипты в рамках задания из четвертой задачи пятой лабораторной работы (lab_05_04_02). Для других задач в некоторые скрипты придется внести изменения. В ходе задания были реализованы следующие скрипты:

1) Скрипты отладочной и релизной сборок.

build_debug.sh

```
#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c -O0 -g main.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c -O0 -g file_bin.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-conversion -Wvla -c -O0 -g file_text.c
gcc -o app.exe main.o file_bin.o file_text.o -lm
```

build_release.sh

```
#!/bin/bash

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

В каталоге с исходным кодом программы в файле main.c располагаются скрипты build_debug.sh, build_release.sh, с помощью которых

автоматизируется сборка отладочной и релизной сборок проекта. В скриптах отдельны выделены два этапа сборки: компиляция и компоновка.

2) Скрипты отладочной сборки с санитайзерами

build_asan.sh

```
#!/bin/bash

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

build_msan.sh

```
#!/bin/bash

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

build_ubsan.sh

```
#!/bin/bash

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

Данные скрипты реализуют сборку с address, memory и undefined behavior sanitizer соответственно. Так же реализован скрипт, при запуске которого запускается автоматическое тестирование программы под всеми санитайзерами.

sanitize_check.sh

```
#!/bin/bash

echo "build_asan:"
./build_asan.sh
./testing.sh
./clean.sh

echo ""
echo "build_msan:"
./build_msan.sh
./testing.sh
./clean.sh
```

```
echo ""
echo "build_ubsan:"
./build_ubsan.sh
./testing.sh
./clean.sh
```

collect_coverage.sh

```
#!/bin/bash

./testing.sh

echo ""
echo "Coverage (in %):"
gcov main.c > "tmp.txt"
var=$(cat tmp.txt)
echo "${var#*:}" | sed 's/%*$/ /g' | sed 's/ .*//'
rm -f "tmp.txt"
```

Данный скрипт запускает скрипт *func_tests.sh,* чтобы отобразить информацию о полноте тестирования, а затем с помощью утилиты gcov считает и выводит процент покрытия кода программы *main.c.*

Для правильной работы скрипта, перед его запуском необходимо собрать программу с помощью скрипта *build_gcov.sh*.

build_gcov.sh

```
#!/bin/bash

gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -O0 -g --coverage main.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -O0 -g --coverage file_bin.c
gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-equal -Wfloat-
conversion -Wvla -c -O0 -g --coverage file_text.c
gcc -o app.exe main.o file_bin.o file_text.o --coverage -lm
```

3) Скрипт очистки побочных файлов

clean.sh

```
#!/bin/bash

files="./func_tests/data/*.tmp ./func_tests/scripts/*.txt
./func_tests/scripts/*.exe *.exe *.o *.out *.gcno *.gcda *.gcov"
```

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

Данный скрипт удаляет все побочные файлы.

4) Компаратор для сравнения результата выполнения программы с ожидаемым результатом, который изначально располагается в out-файле.

comparator.sh

```
#!/bin/bash

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

f1=$1
f2=$2


mask="*."

clean_out_prog=$(grep -Eo "$mask" "$f1")
clean_out_test=$(grep -Eo "$mask" "$f2")

if [ "$clean_out_prog" != "$clean_out_test" ]; then
    exit 1
fi

exit 0
```

Принцип работы компаратора:

- На вход подаются файл с выходными данными, который получился при выполнении программы (f1) и файл с выходными данными, который существовал в папке с тестами изначально и считается *ожидаемым результатом* выходных данных (f2).
- С помощью маски (в разных компараторах – разные маски) и функции grep в две отдельные переменные помещается содержимое двух этих файлов (либо целые числа, либо вещественные).
- Сравниваются результаты grep-а, помещенные в переменные. Если они не одинаковы – содержимое разное. Иначе – одинаковое.
- В случае одинакового содержимого возвращается код возврата – 0, в случае разного содержимого – 1.

5) Скрипт, запускающий проверку всех скриптов утилитой shellcheck

check_scripts.sh

```
#!/bin/bash

scripts="./*.sh"

for file in $scripts ; do
      shellcheck "$file"
done

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

for file in $scripts ; do
      shellcheck "$file"
done
```

6) Скрипт, выдающий права на исполнение другим скриптам

chmod.sh

```
#!/bin/bash

files="./*.sh"
for file_in in $files; do
      chmod +x "$file_in"
done
cd ./func_tests/scripts || exit 1
files="./*.sh"
for file_in in $files; do
      chmod +x "$file_in"
done
cd ../../
```

7) Скрипт, запускающий автоматическое тестирование

testing.sh

```
#!/bin/bash

cd ./func_tests/scripts/ || exit 1
./func_tests.sh
cd ../../
```

8) Скрипт для проведения автоматического тестирования

func_tests.sh

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

```
test_ok="0"
files_count=0
count_errors=0

files="../data/pos_??_in.txt"
for file_in in $files; do
      if [ -f "$file_in" ]; then
            number=$(echo "$file_in" | grep -o "[0-9]*")
      else
            echo "No positive tests"
            continue
      fi
      number=$(echo "$file_in" | grep -o "[0-9]*")

      file_out="../data/pos_""$number""_out.txt"
      file_args="../data/pos_""$number""_args.txt"

      if [[ $(head -c 2 < "$file_args") == "ab" ]]; then
            mode="b"
      elif [[ $(head -c 2 < "$file_args") == "ft" ]]; then
            mode="t"
      else
            mode="n"
      fi

      if [ -f "$file_out" ] && [ -f "$file_args" ]; then
            if [[ $mode == "t" ]]; then
                  ./pos_case.sh "$file_in" "$file_out" "$file_args" "$mode"
                  error="$?"
            elif [[ $mode == "b" ]]; then
                  gcc -o cnv.exe conversion.c
                  file_tmp="""$file_in"".tmp"
                  ./cnv.exe "t2b" "$file_in" "$file_tmp"
                  ./pos_case.sh "$file_in" "$file_out" "$file_args" "$mode"
                  error="$?"
            else
                  echo "pos_""$number""_in"": FAIL (Wrong mode)"
                  count_errors=$((count_errors + 1))
                  files_count=$((files_count + 1))
                  continue
            fi
      else
            echo "pos_""$number""_in"": FAIL (No Output/Args File)"
            count_errors=$((count_errors + 1))
            files_count=$((files_count + 1))
            continue
      fi
      if [ "$error" -eq "$test_ok" ]; then
            echo "pos_""$number""_in"": PASS"
      else
            echo "pos_""$number""_in"": FAIL"
            count_errors=$((count_errors + 1))
      fi
      files_count=$((files_count + 1))
done
```

```
echo
files="../data/neg_??_in.txt"
for file_in in $files; do
      if [ -f "$file_in" ]; then
            number=$(echo "$file_in" | grep -o "[0-9]*")
      else
            echo "No negative tests"
            continue
      fi

      file_args="../data/neg_""$number""_args.txt"

      if [[ $(head -c 2 < "$file_args") == "ab" ]]; then
            mode="b"
      elif [[ $(head -c 2 < "$file_args") == "ft" ]]; then
            mode="t"
      else
            mode="n"
      fi

      if [ -f "$file_args" ]; then
            if [[ $mode == "t" ]]; then
                  ./neg_case.sh "$file_in" "$file_args" "$mode"
                  error="$?"
            else
                  gcc -o cnv.exe conversion.c
                  file_tmp="""$file_in"".tmp"
                  ./cnv.exe "t2b" "$file_in" "$file_tmp"
                  ./neg_case.sh "$file_in" "$file_args" "$mode"
                  error="$?"
            fi
      else
            echo "neg_""$number""_in"": FAIL (No Output/Args File)"
            count_errors=$((count_errors + 1))
            files_count=$((files_count + 1))
            continue
      fi

      if [ "$error" -ne "$test_ok" ]; then
            echo "neg_""$number""_in"": PASS"
      else
            echo "neg_""$number""_in"": FAIL"
            count_errors=$((count_errors + 1))
      fi
      files_count=$((files_count + 1))
done

percentage=$(echo "scale=9; ($files_count-$count_errors)/$files_count*100"
| bc )
echo
echo "Tests passed (in %): "
echo "$percentage" | awk '{printf "%.0f\n",$1}'
exit "$count_errors"
```

Принцип работы скрипта:

- Скрипт находит все файлы вида *pos_NN_in.txt*. При их отсутствии программа сообщает, что позитивных тестов нет.
- Далее скрипт находит для каждого in-файла соответствующий out-файл и args-файл. Если out-файла (или args-файла) с очередным номером не существует, то тест с этим номером считается проваленным.
- Далее скрипт определяет, с каким in-файлом он работает. Конкретно для задачи lab_05_04_02 существует 2 типа передаваемых файлов: текстовые и бинарные. С помощью регулярного выражения, скрипт устанавливает режим (mode) работы, где t – текстовый, b – бинарный, n – not defined. Режимы проверяются по заданным в задаче ключам (в данном случае ключ ab - для бинарных файлов, ft – для текстовых файлов). Так как сначала проверяются положительные тесты, то при режиме not defined тест считается проваленным.
- Далее по режиму выбирается тип запуска программы с in-файлом. Если режим открытия – t (то есть in-файл текстовый), то запускаем скрипт положительного случая (pos_case.sh) для текстовых (`$mode`) файлов `$file_in, $file_out, $file_args`. Если режим открытия – b (то есть in-файл бинарный), то сначала запускается конвертер (conversion.c) с ключом t2b, с помощью которого in-файл переводится из текстового в бинарный (изначально все тестовые данные хранятся в папке func_tests/data как текстовые файлы, поэтому при необходимости информацию из текстовых файлов необходимо временно перевести в бинарный вид). Содержимое (бинарный файл) хранится во временном файле `$file_tmp` (именно этот файл изначально указан в args-файле для теста с бинарным файлом). Скрипт положительного случая (pos_case.sh) запускается с текстовыми файлами `$file_in, $file_out, $file_args,` но так как $mode равен b, то программа будет работать с файлом, который указан в $file_args, как с бинарным.
- Возвращаемое значение скрипта `pos_case.sh` проверяется автоматически (если код возврата совпадает с нулем, то тест считается пройденным, иначе - нет).
- Все вышеперечисленные действия аналогично проводятся с негативными тестами, однако есть несколько нюансов. При режиме открытия n – not defined тест не считается проваленным, так как неправильно заданный ключ считается фишкой негативного теста (то, что ключ не валидный проверяется самой программой). Негативный тест считается пройденным, если код возврата скрипта негативного случая (neg_case.sh) отличен от нуля, в противном случае – проваленным.

- Во время выполнения скрипт считает общее количество тестов и количество пройденных тестов. После обработки всех тестов, скрипт выводит информацию о том, сколько процентов от всех тестов оказались пройденными. 100% означает, что все тесты прошли успешно.

9) Скрипт позитивного случая

pos_case.sh

```
#!/bin/bash

ok="0"
fail="1"

if [ $# -ne 4 ]; then
      exit "$fail"
fi

bin_file=$1

out_test=$2
args_test=$3

mode=$4

tmp_out="tmp_out.txt"
command="../../app.exe "

if [ $# -eq 4 ]; then
      command="$command $(cat "$args_test")"
fi

if [[ $mode == "t" ]]; then
    $command > "$tmp_out"
    error="$?"
    if [[ $error -ne 0 ]]; then
        exit "$fail"
    fi
    ./comparator.sh "$tmp_out" "$out_test"
    return_code="$?"
elif [[ $mode == "b" ]]; then
    $command < "$bin_file"
    error="$?"
    if [[ $error -ne 0 ]]; then
        exit "$fail"
    fi
    text_out="bin_out.txt"
    ./cnv.exe "b2t" """$bin_file"".tmp" "$text_out"
    ./comparator.sh "$text_out" "$out_test"
    return_code="$?"
else
    exit "$fail"
```

```
fi


if [[ return_code -eq 0 ]]; then
    exit "$ok"
else
    exit "$fail"
fi
```

Принцип работы скрипта:

- На вход подаются файлы *pos_NN_in.txt*, *pos_NN_out.txt, pos_NN_args.txt*, а также режим работы с in-файлом (mode).
- Команда запуска программы запускается с помощью переменной command. Так как в args-файле содержится информация о всех ключах, необходимых для запуска программы, то их необходимо добавить в конец выполняемой команды (что и делается при проверке на количество переданных аргументов).
- Далее для каждого из режимов работы (mode) выполняются ряд своих инструкций. Если скрипт работает с текстовым файлом, то результат выполнения программы помещается во временный файл $tmp_out. Если возникла ошибка во время выполнения программы (ненулевой код возврата), то скрипт возвращает ненулевой код возврата. Далее с помощью компаратора сравнивается содержимое двух файлов: сформированного программой $tmp_file и изначального out-файла $out_test. При совпадении содержимого файлов возвращается нулевой код возврата, иначе – ненулевой.
- Если скрипт работает с бинарным файлом (`$mode == "b"`), то программа (конкретно в lab_05_04_02) изменяет бинарный файл путем добавления информации из in-файла (редактируется временный бинарный файл `"""$bin_file"".tmp"`). Чтобы сравнить вывод программы с ожидаемым выводом (в out-файле), необходимо запустить конвертер с ключом b2t, который преобразует содержимое бинарного файла в временный текстовый файл $text_out. И далее с помощью компаратора сравнивается содержимое файлов $text_out (получившийся текстовый файл) и $out_test (out-файл). При совпадении содержимого файлов возвращается нулевой код возврата, иначе – ненулевой.

10) Скрипт негативного случая

neg_case.sh

```
#!/bin/bash

if [ $# -ne 3 ]; then
      exit 1
fi

ok="0"
fail="1"

bin_file=$1
args_test=$2

mode=$3

tmp_out="tmp_out.txt"
command="../../app.exe "

if [ $# -eq 3 ]; then
      command="$command $(cat "$args_test")"
fi

if [[ $mode != "b" ]]; then
    $command > "$tmp_out"
    error="$?"
    if [[ $error -ne 0 ]]; then
        exit "$fail"
    fi
else
    $command < "$bin_file"
    error="$?"
    if [[ $error -ne 0 ]]; then
        exit "$fail"
    fi
fi

exit "$ok"
```

Так как под негативном случаем подразумевается возвращение ненулевого кода возврата, то и проверять нужно только его. Поэтому скрипт очень похож на скрипт позитивного случая, в котором проверяется только код возврата и не проверяется результат работы программы. При ненулевом коде возврата возвращается ненулевой код возврата.

11) Конвертер файлов с помощью ключей (t2b, b2t)

conversion.c

```
#include <stdio.h>
#include <string.h>
```

```
#include <stdint.h>
#include <inttypes.h>

#define MAX_NAME_LEN 30
#define MAX_MANUF_LEN 15

typedef struct
{
    char name[MAX_NAME_LEN + 1];
    char manuf[MAX_MANUF_LEN + 1];
    uint32_t price;
    uint32_t count;
} product_t;

void binary_to_text(const char *binary_filename, const char *text_filename)
{
    FILE *binary_file = fopen(binary_filename, "rb");
    if (binary_file == NULL)
    {
        printf("Failed to open binary file %s for reading\n",
binary_filename);
        return;
    }

    FILE *text_file = fopen(text_filename, "w");
    if (text_file == NULL)
    {
        printf("Failed to open text file %s for writing\n", text_filename);
        fclose(binary_file);
        return;
    }

    product_t product;
    while (fread(&product, sizeof(product_t), 1, binary_file) != 0)
    {
        fprintf(text_file, "%s %s %u %u\n", product.name, product.manuf,
product.price, product.count);
    }

    //printf("Binary file %s converted to text file %s\n", binary_filename,
text_filename);
    fclose(binary_file);
    fclose(text_file);
}

void text_to_binary(const char *text_filename, const char *binary_filename)
{
    FILE *text_file = fopen(text_filename, "r");
    if (text_file == NULL)
    {
        printf("Failed to open text file %s for reading\n", text_filename);
        return;
    }

    FILE *binary_file = fopen(binary_filename, "wb");
    if (binary_file == NULL)
```

```
    {
        printf("Failed to open binary file %s for writing\n",
binary_filename);
        fclose(text_file);
        return;
    }

    char name[MAX_NAME_LEN];
    char manufacturer[MAX_MANUF_LEN];
    uint32_t price;
    uint32_t quantity;
    while (fscanf(text_file, "%s %s %u %u\n", name, manufacturer, &price,
&quantity) == 4)
    {
        product_t product;
        strncpy(product.name, name, MAX_NAME_LEN);
        strncpy(product.manuf, manufacturer, MAX_MANUF_LEN);
        product.price = price;
        product.count = quantity;

        fwrite(&product, sizeof(product_t), 1, binary_file);
    }

    //printf("Text file %s converted to binary file %s\n", text_filename,
binary_filename);
    fclose(text_file);
    fclose(binary_file);
}

int main(int argc, char **argv)
{
    if (argc != 4)
    {
        printf("cnv.exe [b2t|t2b] [input filename] [output filename]\n");
        return 1;
    }

    char *mode = argv[1];
    char *input_filename = argv[2];
    char *output_filename = argv[3];

    if (strcmp(mode, "b2t") == 0)
    {
        binary_to_text(input_filename, output_filename);
    }
    else if (strcmp(mode, "t2b") == 0)
    {
        text_to_binary(input_filename, output_filename);
    }
    else
    {
        printf("Invalid mode. Must be 'b2t' or 't2b'.\n");
        return 1;
    }

    return 0;
```

```
}
```

Принцип работы конвертера:

- Конвертер принимает на вход ключ конвертации файла, in-файл, out-файл. Вызывается функция конвертации в соответствии с ключом конвертации.
- В рассматриваемой задаче (lab_05_04_02) в файлах хранятся структуры типа product_t (структура описана в начале конвертера). Если в задаче в файлах содержится
- При вызове функции `binary_to_text` открывается in-файл с режимом (rb) и out-файл в режиме (w). Далее из in-файла считываются структуры, а затем записываются в out-файл таким образом, что каждая структура находится на каждой строке (а поля структуры разделены пробелами).
- При вызове функции `text_to_binary` открывается in-файл с режимом (r) и out-файл в режиме (wb). Далее из in-файла считываются структуры по формату, описанному в предыдущем пункте, а затем записываются в out-файл целой структурой (так как out-файл – бинарный).
- В конце выполнения каждой из функций закрываются открытые файлы.

Заключение

Написанные в ходе задания скрипты помогли автоматизировать процесс тестирования и сборки программы, которая принимает аргументы командной строки, а также работает с текстовыми и бинарными файлами. Цель была успешна выполнена. Данные скрипты используются в курсе “Программирование на Си” для пятой лабораторной работы.