Задание 1

1. build_debug.sh

```
1  #!/bin/bash
2
3
4  gcc -std=c99 -Wall -Werror -Wpedantic -Wextra -Wfloat-conversion -Wfloat-
equal -g3 -c main.c
5  gcc main.o -o app.exe -lm
6
```

Build_debug.sh

Скрипт представленный выше реализует отладочную сборку скрипта.

2. build_release.sh

```
1  #!/bin/bash
2
3  gcc -std=c99 -Wall -Wpedantic -Wextra -Wfloat-conversion -Wfloat-equal -c
main.c
4  gcc main.o -o app.exe -lm
5
```

Build_release.sh

Данный скрипт осуществляет релизную сборку скрипта.

3. comporator_number.sh

```
1  #!/bin/bash
2
3  file1="$1"
4  file2="$2"
5
6  if [ -f "$file1" ] && [ -f "$file2" ]; then
7      grep -oE "[+-]?[0-9]+([.][0-9]+)?" "$file1" > file_new_1.txt
8      grep -oE "[+-]?[0-9]+([.][0-9]+)?" "$file2" > file_new_2.txt
9
10      if diff file_new_1.txt file_new_2.txt; then
11          exit 0
12      else
13          exit 1
14      fi
15  else
16      exit 1
17  fi
18
```

Comporator_number.sh

Данный компоратор получает на вход два файла: file1 – результат работы скрипта app.exe, file2 – ожидаемый результат работы программы. В строчках 7-8 с помощью утилиты grep отбираемый из файлов только вещественные числа и записываем их в новые файлы file_new_1.txt и file_new_2.txt, которые мы сравниваем при помощи утилиты diff, в случае если файлы совпадают, то утилита ничего не выведет и скрипт закончит свою работу, с кодом возврата 0, в случае если файлы отличаются или файлы, которые получает скрипт на вход не существуют, то скрипт завершит свою работу с кодом возврата 1.

4. comporator_result.sh

```
1  #!/bin/bash
2
3  file1=$1
4  file2=$2
5
6  if [ -f "$file1" ] && [ -f "$file2" ]; then
7      arr1=($(grep -Eo "Result:[a-Z0-9]+" "$file1"))
8      arr2=($(grep -Ed "Result:[a-Z0-9]+" "$file2"))
9      string1="${arr1[0]}"
10     string2="${arr2[0]}"
11     if diff < "$string1" < "$string2"; then
12         exit 0
13     else
14         exit 1
15     fi
16  else
17      exit 1
18  fi
```

Comporator_result.sh

Данный компоратор получает на вход два файла: file1 – результат работы скрипта app.exe, file2 – ожидаемый результат работы программы. В 6 строчке осуществляется проверка, что компоратор получил 2 файла. В строчках 7-8 в массивы заносятся все строки начинающиеся со слова “Result:”. В строчках 9-10 в переменные string1, string2 заносятся первые вхождения строк, начинающихся со слова “Result:”. Затем в строке 11 сравниваются 2 строки, в случае, если они равны то компоратор завершает свою работу с кодом возврата 0, если же строки не равны или возникли ошибки при передаче позиционных аргументов, то скрипт завершает свою работу с кодом возврата 0.

5. pos_case.sh

```
1  #!/bin/bash
2
3  file_stream_in="$1"
4  file_stream_out_expect="$2"
```

```
5  current_dir=$(dirname "$0")
6  add_args=''
7
8  if [ -n "$3" ]; then
9      if [ -f "$3" ]; then
10         add_args=$(cat "$3")
11     fi
12 fi
13
14 if [ -n "$USE_VALGRIND" ]; then
15     if valgrind --log-file=file.log --quiet "$current_dir/../../app.exe"
"add_args" < "$file_stream_in" > "result.txt"; then
16
17         content_file=$(cat "file.log")
18
19         if [ -z "$content_file" ]; then
20             if  "$current_dir/./comporator_number.sh"
"$file_stream_out_expect" "result.txt" > /dev/null; then
21                 exit 0
22             else
23                 exit 1
24             fi
25         else
26             if "$current_dir/./comporator_number.sh"
"$file_stream_out_expect" "result.txt" > /dev/null; then
27                 exit 2
28             else
29                 exit 3
30             fi
31         fi
32     fi
33 else
34     if "$current_dir/../../app.exe" "add_args" < "$file_stream_in" >
"result.txt"; then
35         if "$current_dir/./comporator_number.sh"
"$file_stream_out_expect" "result.txt" > /dev/null; then
36             exit 0
37         else
38             exit 1
39         fi
40     fi
41 fi
42
```

Pos_case.sh

Скрипт pos_case.sh осуществляет выполнение скрипта app.exe, зачем передает компоратору на вход результирующий файл и файл с ожидаемыми данным. В случае, если тест прошел успешно, то скрипт завершается с кодом возврата 0, если тестирование было выполнено с ошибками, то скрипт завершается с кодом возврата 0. В случае, если поднят флаг USE_VALGRIND, в выполняется другая ветка кода, в которой проверяется не только работоспособность программы, но и проверка на память. В случае, если проверка на работоспособность и память прошла успешно, то скрипт завершает свою

работу с кодом возврата 0, если тест работает неверно, но в памяти нет ошибок, то скрипт завершает свою работу с кодом возврата 0, если тест проходит успешно, но с ошибками в памяти, то pos_case.sh завершает свою работу с кодом возврата 2 и если же тест не проходит по памяти и выдает неверный результат, то скрипт завершает свою работу с кодом возврата 3. Чтобы проверить, что скрипт работает без ошибок в памяти, следует запускать app.exe с помощью valgrind и результат работы выводится в файл file.log при помощи параметра –log-file, чтобы valgrind работал в тихом режиме, следует использовать параметр –quiet.

6. neg_case.sh

```
#!/bin/bash

file_stream_in="$1"
current_dir=$(dirname "$0")
add_args=''

if [ -n "$2" ]; then
    if [ -f "$2" ]; then
        add_args=$(cat "$2")
    fi
fi

if [ -n "$USE_VALGRIND" ]; then
    if valgrind --log-file=file.log --quiet "$current_dir/../../app.exe" "add_args" < "$file_stream_in" > /dev/null; then

        return_code=$?
        content_file=$(cat "file.log")

        if [ -z "$content_file" ]; then
            if [ $return_code -ne 0 ]; then
                exit 0
            else
                exit 1
            fi
        else
            if [ $return_code -ne 0 ]; then
                exit 2
            else
                exit 3
            fi
        fi
    fi
else
    if "$current_dir/../../app.exe" < "$file_stream_in" > /dev/null;then

        return_code=$?

        if [ "$return_code" -ne 0 ]; then
            exit 0
        else
            exit 1
        fi
    fi
```

```
44  fi
45
```

Neg_case.sh

Neg_case.sh работает почти аналогично pos_case.sh, коды возврата соответствуют тому, что описано в pos_case.sh, но neg_case.sh получает на вход файл с входными данным и файл с аргументами (при наличии), в случае если при выполнении app.exe скрипт получает код возврата отличный от нуля, то это означает, что негативный тест был пройден. Если был получен код возврата 0, то негативный тест работает неверно, тогда скрипт завершает свою работу с кодом возврата 1. (Случай с USE_VALGRIND описан в pos_case.sh)

7. func_tests.sh

```
1  #!/bin/bash
2
3  current_dir=$(dirname "$0")
4
5  i_pos=0
6  i_neg=0
7
8  arr_pos_in=($(ls "$current_dir/../data/" | grep -Eo "pos_[0-9][0-
9]_in.txt" | sort))
9  arr_pos_out=($(ls "$current_dir/../data/" | grep -Eo "pos_[0-9][0-
9]_out.txt" | sort))
10  arr_neg_in=($(ls "$current_dir/../data/" | grep -Eo "neg_[0-9][0-
9]_in.txt" | sort))
11  arr_neg_args=($(ls "$current_dir/../data/" | grep -Eo "neg_[0-9][0-
12  arr_pos_args=($(ls "$current_dir/../data/" | grep -Eo "pos_[0-9][0-
9]_args.txt" | sort))
13
14  len_pos=${#arr_pos_in[@]}
15
16  for ((i=0; i<len_pos;i++))
17  do
18      if [ -n "$USE_VALGRIND" ]; then
19          "$current_dir/./pos_case.sh"
"$current_dir/../data/${arr_pos_in[i]}"
"$current_dir/../data/${arr_pos_out[i]}"
"$current_dir/../data/${arr_pos_args[i]}"
20          return_code=$?
21          if [ $return_code -eq 0 ]; then
22              echo "TEST: $((i+1)): PASSED MEMORY: PASSED"
23              i_pos=$((i_pos+1))
24          fi
25          if [ $return_code -eq 1 ]; then
26              echo "TEST: $((i+1)): FAILED MEMORY: PASSED"
27          fi
28          if [ $return_code -eq 2 ]; then
29              echo "TEST: $((i+1)): PASSED MEMORY: FAILED"
30              i_pos=$((i_pos+1))
31          fi
```

```
32          if [ $return_code -eq 3 ]; then
33              echo "TEST: $((i+1)): FAILED MEMORY: FAILED"
34          fi
35      else
36          "$current_dir/./pos_case.sh"
"$current_dir/../data/${arr_pos_in[i]}"
"$current_dir/../data/${arr_pos_out[i]}"
"$current_dir/../data/${arr_pos_args[i]}"
37          return_code=$?
38          if [ $return_code -eq 0 ]; then
39              i_pos=$((i_pos+1))
40              echo "TEST $((i+1)): PASSED"
41          else
42              echo "TEST $((i+1)): FAILED"
43          fi
44      fi
45  done
46
47  len_neg=${#arr_neg_in[@]}
48
49  for ((i=0; i<len_neg;i++))
50  do
51      if [ -n "$USE_VALGRIND" ]; then
52          "$current_dir/./pos_case.sh"
"$current_dir/../data/${arr_neg_in[i]}"
"$current_dir/../data/${arr_neg_args[i]}"
53          return_code=$?
54          if [ $return_code -eq 0 ]; then
55              echo "TEST: $((i+1)): PASSED MEMORY: PASSED"
56              i_neg=$((i_neg+1))
57          fi
58          if [ $return_code -eq 1 ]; then
59              echo "TEST: $((i+1)): FAILED MEMORY: PASSED"
60          fi
61          if [ $return_code -eq 2 ]; then
62              echo "TEST: $((i+1)): PASSED MEMORY FAILED"
63              i_neg=$((i_neg+1))
64          fi
65          if [ $return_code -eq 3 ]; then
66              echo "TEST: $((i+1)): PASSED MEMORY FAILED"
67          fi
68
69      else
70          "$current_dir/./neg_case.sh"
"$current_dir/../data/${arr_neg_in[i]}"
"$current_dir/../data/${arr_neg_args[i]}"
71          return_code=$?
72          if [ $return_code -eq 0 ]; then
73              i_neg=$((i_neg+1))
74              echo "TEST $((i+1)): PASSED"
75          else
76              echo "TEST $((i+1)): FAILED"
77          fi
78      fi
79  done
80
81  echo "Positive tests: correct $i_pos of $len_pos"
82  echo "Negative tests: correct $i_neg of $len_neg"
83
```

```
84  number_fail=$((len_neg+len_pos-i_pos-i_neg))
85
86  exit $number_fail
87
```

Func_tests.sh

Скрипт представленный выше выполняет перебор всех тестовых файлов, находящихся в директории /data, в строчках 8-12 в массивы считываются все файлы, которые необходимы для тестирования app.exe. переменные i_pos и i_neg подсчитывают количество пройденных положительных и отрицательных тестов. Переменные len_pos и neg_pos хранят количество положительных и отрицательных тестов. В случае если поднят флаг USE_VALGRIND, то сначала выполняется скрипт pos_case или neg_case, а затем проверяются коды возврата этих скриптов и выводятся сообщения об успешности прохождения теста и прохождения памяти. Если же USE_VALGRIND не поднят, то проверяется только успешность прохождения теста и выводится соответствующее сообщение об успешности. В конце скрипта выводится количество успешно пройденных позитивных и негативных тестов.

8. collect_coverage.sh

```
1  #!/bin/bash
2
3  gcc main.c --coverage -o app.exe
4  ./func_tests/scripts/func_tests.sh
5  gcov main.c
6
7
```

Collect_coverage.sh

Скрипт представленный выше выполняет компиляцию файла с параметром –coverage, чтобы утилита gcov могла собирать данные по покрытию кода, при прохождении тестов. Скрипт выводит в консоль в процентах покрытие кода.

9. clean.sh

```
1  #!/bin/bash
2
3  rm -f ./func_tests/scripts/*.txt
4  rm -f ./func_tests/scripts/*.log
5  rm -f ./*.txt
6
```

Clean.sh

Данный скрипт выполняет удаление файлов, которые создаются тестирующей системой. Удаляются дополнительные текстовые файлы и логи.