**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.

1. 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 |

1. 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.

1. 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-a, помещенные в переменные. Если они не одинаковы – содержимое разное. Иначе – одинаковое.
* В случае одинакового содержимого возвращается код возврата – 0, в случае разного содержимого – 1.

1. 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 |

1. 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 ../../ |

1. Скрипт, запускающий автоматическое тестирование

**testing.sh**

|  |
| --- |
| #!/bin/bash  cd ./func\_tests/scripts/ || exit 1  ./func\_tests.sh  cd ../../ |

1. 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.

1. 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.

1. 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.

1. 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.