АВС: ИДЗ-3

Ноябрь 2024

Гобец Иван Евгеньевич. БПИ 237. Вариант 18

Условие варината 18

Разработать программу, заменяющую все согласные буквы в заданной ASCII–строке на заглавные.

Отчет на 10

Отчет сразу начинается с оценки на 10, так как, спросив у семинариста, он сказал, что можно так сделать.

- Далее буду предоставлять блоки по различным частям кода (для понятности, чтобы не было все подряд).
- Ещё я пытался, все строчки кода комментировать для более понятного пояснения в отчете.

Файл производящий работу со строкаму (string_processor.asm)

- Начну с малой “функции” это копирование строки с адресса **a5** в **a6**. Цикл **loop_copy** в котором происходит само копирование строки. В **t0** загружаем значение с **a5** и записываем его в **a6**, после проверям текущий символ это конец строки или нет (т.е. 0), если это так, то заканчиваем копирование, иначе к добаляем в **a5** и **a6**, т.е. переходим к следующему символу и запускаем цикл заново.

```
56  strcpy:
57          loop_copy:
58                  lb      t0, (a5)
59                  sb      t0, (a6)
60                  beqz    t0, end
61                  addi    a5, a5, 1
62                  addi    a6, a6, 1
63                  b       loop_copy
64
65          end:
66                  ret
67
```

- Для начала снова повторим условие варианта: **Разработать программу, заменяющую все согласные буквы в заданной ASCII–строке на заглавные.** Далее я разберу свой код и поясню логику его работы.
- Мне нужно все согласнеы буквы заменить на заглавные. Для начала я загружу в a1, a2 ASCII значения строчной буквы a и z. И сразу проверю, если текущий символ не лежит в диапозоне $[a_1, a_2]$, значит это не строчная буква, соедовательно я сразу такой символ пропускаю.

- Далее я проверяю, что текущий символ это не согласная строчая буква. Всего будет 5 “if” на каждую гласную букву, если текущий символ гласная буква, то я пропускаю его.
- После всех проверок я могу удостовериться, что текущий символ это строчная согласная буква. Теперь мне достаточно от текущего символа отнять **32** и это будет соответствующая согласная буква, но уже заглавная. Потом меняем текущую букву в строке и все готово.

```
.globl  string_refactor strcpy

string_refactor:
        mv      a0      s3                      # Пермещаем нидекс начала строки в a0
        mv      a3      a0                      # Сохраняем адресс начала строки

        loop:
                lbu     t0      (a0)            # Считываем симовол
                beqz    t0      fin             # Если это конец строки то прыгаем в подпрограмму завершения цикла (fin)

                # Отсекаем все ненужные сиволы, т.е. все что не строчная буква
                #-------------------------------------------------------------------------------------------------------------------------
                li      a1      'a'
                li      a2      'z'

                blt     t0      a1      continue        # Если ASCCII код меньше чем сивол 'a' то пропускаем данный символ
                bgt     t0      a2      continue        # Если ASCCII код больше чем сивол 'z' то пропускаем данный символ
                #-------------------------------------------------------------------------------------------------------------------------


                # Если у нас считанный символ гласная буква то перекодим к следующему
                #-------------------------------------------------------------------------------------------------------------------------
                li      a1      'a'
                beq     t0      a1      continue

                li      a1      'e'
                beq     t0      a1      continue

                li      a1      'i'
                beq     t0      a1      continue

                li      a1      'o'
                beq     t0      a1      continue

                li      a1      'u'
                beq     t0      a1      continue
                #-------------------------------------------------------------------------------------------------------------------------

                # То есть мы все "нехорошие" символы отсекли и у нас остались только согласные буквы.
                # Теперь вычитаем из ASCII текущей буквы 32, чтобы получить заглавную версию буквы.
                #-------------------------------------------------------------------------------------------------------------------------
                addi    t0      t0      -32
                sb      t0      (a0)

                j       continue
                #-------------------------------------------------------------------------------------------------------------------------

        continue:
                addi    a0      a0      1               # Переходим на следующий символ
                j       loop                            # Возвращаемся обратно в цикл

        fin:
                mv      a0      a3                      # Возвращаем в a0 индекс начала нашей "строки"
                ret
```

Библиотека макросов (macro-syscalls.m)

- **print_int (%x)** - макрос для печати содержимого заданного ригистра как целого.
- **print_imm_int (%x)** - макрос для печати непосредственного целочисленного значения.
- **print_str (%x)** - макрос для печати строковой константы, ограниченной нулевым символом.

```
#-----------------------------------------------------------------------------
# Печать содержимого заданного регистра как целого
.macro print_int (%x)
        li a7, 1
        mv a0, %x
        ecall
.end_macro
#-----------------------------------------------------------------------------
# Печать непосредственного целочисленного значения
.macro print_imm_int (%x)
        li a7, 1
        li a0, %x
        ecall
.end_macro

#-----------------------------------------------------------------------------
# Печать строковой константы, ограниченной нулевым символом
.macro print_str (%x)
   .data
str:
   .asciz %x
   .align 2     # Выравнивание памяти. Возможны данные размером в слово
   .text
   push (a0)
   li a7, 4
   la a0, str
   ecall
   pop  (a0)
.end_macro

#-----------------------------------------------------------------------------
```

- **print_char(%x)** - макрос для вывода отдельного заданного символа.
- **new_line** - макрос для перевода строки.
- **read_int_a0** - макрос для ввода целого числа и запись его в регист **a0**
- **read_int(%x)** - макрос для ввода целого числа с консоли в указанный регистр, исключая регистр a0.

```
#-----------------------------------------------------------------------------
# Печать отдельного заданного символа
.macro print_char(%x)
   li a7, 11
   li a0, %x
   ecall
.end_macro

#-----------------------------------------------------------------------------
# Печать перевода строки
.macro newline
   print_char('\n')
.end_macro

#-----------------------------------------------------------------------------
# Ввод целого числа с консоли в регистр a0
.macro read_int_a0
   li a7, 5
   ecall
.end_macro

#-----------------------------------------------------------------------------
# Ввод целого числа с консоли в указанный регистр, исключая регистр a0
.macro read_int(%x)
   push (a0)
   li a7, 5
   ecall
   mv %x, a0
   pop  (a0)
.end_macro
```

- **str_get(%strbuf, %size)** - макрос для ввода строки в буфер заданного размера с заменой перевода строки нулем. **%strbuf** - адрес буфера, **%size** - целая константа, ограничивающая размер вводимой строки.
- **open** - макрос для открытия файла для чтения, записи, дополнения
  - .eqv READ_ONLY 0 - Открыть для чтения
  - .eqv WRITE_ONLY 1 - Открыть для записи
  - .eqv APPEND 9 - Открыть для добавлени
- **read** - макрос для чтения информации из открытого файла.

```
#-------------------------------------------------------------------------------
# Ввод строки в буфер заданного размера с заменой перевода строки нулем
# %strbuf - адрес буфера
# %size - целая константа, ограничивающая размер вводимой строки
.macro str_get(%strbuf, %size)
    la      a0 %strbuf
    li      a1 %size
    li      a7 8
    ecall
    push(s0)
    push(s1)
    push(s2)
    li  s0 '\n'
    la  s1      %strbuf
next:
    lb  s2  (s1)
    beq s0      s2      replace
    addi s1 s1 1
    b   next
replace:
    sb  zero (s1)
    pop(s2)
    pop(s1)
    pop(s0)
.end_macro

#-------------------------------------------------------------------------------
# Открытие файла для чтения, записи, дополнения
.eqv READ_ONLY  0       # Открыть для чтения
.eqv WRITE_ONLY 1       # Открыть для записи
.eqv APPEND         9   # Открыть для добавления
.macro open(%file_name, %opt)
    li          a7 1024         # Системный вызов открытия файла
    la      a0 %file_name   # Имя открываемого файла
    li          a1 %opt         # Открыть для чтения (флаг = 0)
    ecall                       # Дескриптор файла в a0 или -1)
.end_macro

#-------------------------------------------------------------------------------
# Чтение информации из открытого файла
.macro read(%file_descriptor, %strbuf, %size)
    li   a7, 63         # Системный вызов для чтения из файла
    mv   a0, %file_descriptor       # Дескриптор файла
    la   a1, %strbuf    # Адрес буфера для читаемого текста
    li   a2, %size              # Размер читаемой порции
    ecall               # Чтение
.end_macro
```

- **read_addr_reg(%file_descriptor, %reg, %size)** - макрос для чтения информации из открытого файла, когда адрес буфера в регистре
- **close(%file_descriptor)** - макрос для закрытия файла.
- **allocate(%size)** - макрос для выделения области динамической памяти заданного размера.
- **exit** - макрос для завершения работы программы.

- **push(%x)** - макрос для сохранения заданного регистра на стеке.
- **pop(%x)** - макрос для выталкивания значения из стека в заданный регистр.

```
#-------------------------------------------------------------------------------
# Чтение информации из открытого файла,
# когда адрес буфера в регистре
.macro read_addr_reg(%file_descriptor, %reg, %size)
    li   a7, 63         # Системный вызов для чтения из файла
    mv   a0, %file_descriptor       # Дескриптор файла
    mv   a1, %reg       # Адрес буфера для читаемого текста из регистра
    li   a2, %size              # Размер читаемой порции
    ecall               # Чтение
.end_macro

#-------------------------------------------------------------------------------
# Закрытие файла
.macro close(%file_descriptor)
    li   a7, 57       # Системный вызов закрытия файла
    mv   a0, %file_descriptor  # Дескриптор файла
    ecall             # Закрытие файла
.end_macro

#-------------------------------------------------------------------------------
# Выделение области динамической памяти заданного размера
.macro allocate(%size)
    li a7, 9
    li a0, %size        # Размер блока памяти
    ecall
.end_macro

#-------------------------------------------------------------------------------
# Завершение программы
.macro exit
    li a7, 10
    ecall
.end_macro

#-------------------------------------------------------------------------------
# Сохранение заданного регистра на стеке
.macro push(%x)
        addi    sp, sp, -4
        sw      %x, (sp)
.end_macro

#-------------------------------------------------------------------------------
# Выталкивание значения с вершины стека в регистр
.macro pop(%x)
        lw      %x, (sp)
        addi    sp, sp, 4
.end_macro
```

- **accept_if(%save, %first, %second)** - макрос для записывания в %save 1, если %first и %second равны, иначе 0.

- **print_string(%x)** - макрос для вывода строки в консольб
- **strcpy** - макрос для копирования строки, содержащий подпрограмму **strcpy**.
- **print_new_line_if(%x)** - макрос для выводы новой строки, если в **%s** лежит 1.

```
#-----------------------------------------------------------------------------
# Записывает в %save 1, если %first и %second равны, иначе 0
.macro accept_if(%save, %first, %second)
        beq     %first  %second accept
        bne     %first  %second skip

        accept:
                li      %save   1
                j       end
        skip:
                li      %save   0
                j       end

        end:
.end_macro

#-----------------------------------------------------------------------------
# Макрос для вывода строки
.macro print_string(%x)                 # Определение макроса print_string, принимающего метку %x (адрес строки).
        la, a0, %x                      # Загружаем адрес строки (метки) %x в регистр a0.
        li, a7, 4                       # Загружаем код системного вызова 4 (print_string) в регистр a7 для вывода строки.
        ecall                           # Выполняем системный вызов.
.end_macro

#-----------------------------------------------------------------------------
# Макрос для копирования строки
.macro strcpy (%s1, %s2)
        la a5 %s1
        la a6 %s2
        jal strcpy
.end_macro

#-----------------------------------------------------------------------------
# Макрос для выводы новой строки если в %s лежит 1
.macro print_new_line_if(%s)
        beqz    %s      end

        print_str("\n")

        end:
.end_macro
```

Файл с настройками (settings.asm)

Данная подпрограмма **get_settings** собирает информации у пользователя: 1) Хочет ли он, чтобы подпрограмма протестировалась автоматически; 2) Хочет ли он, чтобы результат вывелся в консоль.

Разберем работу данной подпрограммы:

- **get_settings_autotest** - Выводим пользователю строку - “Autotest the program (Y/N): “ и считываем его ответ. Далее записываем в **a1** ввод пользовател и в **2** ASCII код символа ‘Y’. Далее вызываем макрос **accept_if**, в котором произойдет сравнивание двух строк и если они равна, то в **s7** будет лежать 1, иначе 0.
- **get_settings_output** - Все идентично предыдущей подпрограммы, но тут уже вывожид “Output results to console (Y/N): “, и записываем в **s8** 1, если пользователь ввел ‘Y’, иначе 0.

```
.globl  get_settings
.include "macro-syscalls.m"

.eqv    SETTINGS_SIZE 16        # Размер буфера для конфигурации работы приложения

get_settings:
        get_settings_autotest:
                print_str("Autotest the program (Y/N): ")
                str_get(settings, SETTINGS_SIZE)

                la      a0      settings
                lbu     a1      (a0)
                li      a2      'Y'

                accept_if(s7, a1, a2)

        get_settings_output:
                print_str("Output results to console (Y/N): ")
                str_get(settings, SETTINGS_SIZE)

                la      a0      settings
                lbu     a1      (a0)
                li      a2      'Y'

                accept_if(s8, a1, a2)

        ret

back:
        ret
```

Файл работы с выводом (iomod.asm)

Подпрограмма **output** для вывода пользователю результата. Помним, что в **s8** хранится 1, если результат нужно вывести, иначе 0. В начале подрограммы проверяем это условие, если 0, то прыгаем в попрограмму **no_output_console** и просто вывозращаемся по адрессу. Если же 1, то выводим строку “Output: “ и после выводим результат и вывозращаемся по адрессу.

```
.globl output
.include "macro-syscalls.m"

output:
        beq     s8      zero    no_output_console       # Если в s8 лежит 0, т.е. false то мы пропускаем вывод
        j       output_console                          # Прыгаем в подпрограмму вывода в консоль

        output_console:
                # Вывод текста на консоль
                print_str("Output: ")
                li      a7      4
                ecall

                ret

        no_output_console:
                ret
```

Работа с файлами (file_utils.asm)

- **read_file** - подпрограмма для чтения файла. Поснение описано подробно в коммментариях. Где **file_name** - имя файлу и **TEXT_SIZE** количество считываемых символов.

```
.globl  read_file save_in_file
.include "macro-syscalls.m"

.eqv    TEXT_SIZE 512           # Размер буфера для текста

read_file:
        open(file_name, READ_ONLY)
        li              s1      -1                              # Проверка на корректное открытие
        beq             a0      s1 er_name                      # Ошибка открытия файла
        mv              s0      a0                              # Сохранение дескриптора файла

        # Выделение начального блока памяти для для буфера в куче
        allocate(TEXT_SIZE)                                     # Результат хранится в a0
        mv              s3,     a0                              # Сохранение адреса кучи в регистре
        mv              s5,     a0                              # Сохранение изменяемого адреса кучи в регистре
        li              s4,     TEXT_SIZE                       # Сохранение константы для обработки
        mv              s6,     zero                            # Установка начальной длины прочитанного текста

        read_loop:
                # Чтение информации из открытого файла
                read_addr_reg(s0, s5, TEXT_SIZE)                # чтение для адреса блока из регистра

                # Проверка на корректное чтение
                beq     a0      s1 er_read                      # Ошибка
                mv      s2      a0                              # Сохранение длины текста
                add     s6,     s6,     s2                      # Размер текста увеличивается на прочитанную порцию

                # При длине прочитанного текста меньшей, чем размер буфера,
                # необходимо завершить процесс.
                bne     s2      s4      end_loop

                # Иначе расширить буфер и повторить
                allocate(TEXT_SIZE)                     # Результат здесь не нужен, но если нужно то...
                add     s5      s5      s2              # Адрес для чтения смещается на размер порции
                b read_loop                             # Обработка следующей порции текста из файла
        end_loop:
                close(s0)                               # Закрытие файла
                mv      t0      s3                      # Адрес буфера в куче
                add     t0      t0      s6              # Адрес последнего прочитанного символа
                addi    t0      t0      1               # Место для нуля
                sb      zero    (t0)                    # Запись нуля в конец текста

        ret
```

```
save_in_file:
        # Сохранение прочитанного файла в другом файле
        open(file_name, WRITE_ONLY)
        li              s1      -1                      # Проверка на корректное открытие
        beq             a0      s1      er_name         # Ошибка открытия файла
        mv              s0      a0                      # Сохранение дескриптора файла

        # Запись информации в открытый файл
        li              a7,     64                      # Системный вызов для записи в файл
        mv              a0,     s0                      # Дескриптор файла
        mv              a1,     s3                      # Адрес буфера записываемого текста
        mv              a2,     s6                      # Размер записываемой порции из регистра
        ecall                                           # Запись в файл

        ret

er_name:
        # Сообщение об ошибочном имени файла
        la              a0      er_name_mes
        li              a7      4
        ecall
        # И завершение программы
        exit
er_read:
        # Сообщение об ошибочном чтении
        la              a0      er_read_mes
        li              a7      4
        ecall
        # И завершение программы
        exit
```

- **save_in_file** - подпрограмма для сохранения данных в файл.
- **er_name** - в случае ошибки в имени файла, выводим строку “An error occurred - the file name is incorrectn” и завершаем работу программы.
- **er_read** - в случае ошибки в чтении файла, выводим строку “An error occurred during the read operationn” и завершаем работу программы.

Main

В **main** происходит логика вся работы программы. Для начала мы прыгаем в подрограмму **get_settings** (описание данной подпрограмы было написано выше), далее прыгаем в подпрограмму **state_machine**. Для начала записываем в стек адресс **ra** после, если в **s7** лежит 1, то прыгаем в подпрограмму **autotest** (разбора этой попрограммы, будет разобран далее), иначе продолжаем работу подпрограмму. Подпрограмма **read_data** выводим “Enter the path to the file: “ и запрашиваем у пользователя имя файла через макрос **str_get** далее прыгаем в подпрограмму **read_file** и считываем содержимое файла. Далее выполнятся подпрограмма **string_refactor** и выполням бизнесс логику. После прыграем в подпрограмму **output**, где выводится результат (описание данной подпрограмы было написано выше), если **s8** равно 1, иначе не выводим. После в подпрограмме **save_data** звпрашиваем у пользователя имя файла для записи и сохраненям результат через макрос **save_in_file**. После со стека снимаем адресс **ra** с помощью макроса **pop** и возвращаемся в main, где завершаем работу программы через макрос **exit**.

```
.globl  main file_name er_name_mes er_read_mes settings

.include "macro-syscalls.m"

.eqv    TEXT_SIZE 512           # Размер буфера для текста
.eqv    NAME_SIZE 256           # Размер буфера для имени файла
.eqv    SETTINGS_SIZE 16        # Размер буфера для конфигурации работы приложения

.data
# Buffers
string_buf:     .space  TEXT_SIZE

file_name:      .space  NAME_SIZE
settings:       .space  SETTINGS_SIZE

# Strings
er_name_mes:    .asciz "An error occurred - the file name is incorrect\n"
er_read_mes:    .asciz "An error occurred during the read operation\n"

.text
main:
        # return s7, s8 - flags
        jal     get_settings
        jal     state_machine

        exit()

state_machine:
        push(ra)
        bne     s7      zero    autotest

        read_data:
                print_str("Enter the path to the file: ")
                str_get(file_name, NAME_SIZE)
                jal     read_file

        refactor_data:
                jal     string_refactor

        output_data:
                jal     output
                print_str("\n")

        save_data:
                print_str("Enter name of file: ")
                str_get(file_name, NAME_SIZE)
                jal     save_in_file

        return:
                pop(ra)
                ret
```

Тестирование

Для начала предоставлю ручное тестирование.

- Протестируем строку **Ajsdjhfuiy823yjahfxznmvcnejwr90 iasfmnadfn asd** лежащую по адрессу **test1.txt** с парметром вывода - **да**. И именем файла для сохранения результата **output**.

```
Autotest the program (Y/N): **** user input : N
Output results to console (Y/N): **** user input : Y
Enter the path to the file: **** user input : TestData/test1.txt
Output: AJSDJHFuiY823YJaHFXZNMVCNeJWR90 iaSFMNaDFN aSD
Enter name of file: **** user input : output
```

- Получили строку в файле **output** - **AJSDJHFuiY823YJaHFXZNMVCNeJWR90 iaSFMNaDFN aSD**
- Работа программы прошла успешно.

- Протестируем строку **aa_ii_uu_oo_aa_oo_pp_bb** лежащую по адрессу **test6.txt** с парметром вывода - **нет**. И именем файла для сохранения результата **output123**.

```
Autotest the program (Y/N): **** user input : N
Output results to console (Y/N): **** user input : N
Enter the path to the file: **** user input : TestData/test6.txt

Enter name of file: **** user input : output123
```

- Получили строку в файле **output123** - **aa_ii_uu_oo_aa_oo_PP_BB**
- Работа программы прошла успешно.

Дале будет описана логика выполнения автотестирования Объявим имена для файлов, которые будут браться в качестве читаемых и имена выходных файлов. И динамическую память для записи хранения там строк (будет удобно для тестирования).

```
.globl  autotest
.include "macro-syscalls.m"

.eqv    NAME_SIZE 256           # Размер буфера для имени файла

.data
test_case_1_input:      .asciz  "TestData/test1.txt"
test_case_1_output:     .asciz  "TestData/test1_result.txt"

test_case_2_input:      .asciz  "TestData/test2.txt"
test_case_2_output:     .asciz  "TestData/test2_result.txt"

test_case_3_input:      .asciz  "TestData/test3.txt"
test_case_3_output:     .asciz  "TestData/test3_result.txt"

test_case_4_input:      .asciz  "TestData/test4.txt"
test_case_4_output:     .asciz  "TestData/test4_result.txt"

test_case_5_input:      .asciz  "TestData/test5.txt"
test_case_5_output:     .asciz  "TestData/test5_result.txt"

test_case_6_input:      .asciz  "TestData/test6.txt"
test_case_6_output:     .asciz  "TestData/test6_result.txt"

test_file_name_input:   .space  NAME_SIZE
test_file_name_output:  .space  NAME_SIZE
```

Опишем логику выполнения тестирования. Для начала мы запоминаем адресс **ra** в стек. Далее копируем в **file_name** значение **test_file_name_input** (далее будет описано как мы с ним работаем) и считваем файл, рефакторим строку, выводим если пользователь поставил такой параметр и сохраняем результат в файл (имя выходного файла мы скопировали с test_file_name_output). После снимаем адресс **ra** со стека и возвращаемся в **autotest**.

```
test_logic:
        push(ra)

        print_str("\n")
        print_str("File path input: ")
        print_string(test_file_name_input)

        strcpy(test_file_name_input, file_name)
        jal     read_file

        jal     string_refactor

        print_new_line_if(s8)
        jal     output

        strcpy(test_file_name_output, file_name)
        jal     save_in_file

        print_str("\n")

        print_str("File name ouput: ")
        print_string(test_file_name_output)

        print_str("\n")

        pop(ra)
        jalr    ra
```

Все тесты, их 6 штук. В каждой подпрограмме **test_i** мы копируем значение в **test_file_name_imput** из **test_case_i_input** и анологично для **test_file_name_output** из **test_case_i_output**. После прыгаем в саму логику тестирования. После всех тестов завершаем программу.

```
.text
autotest:
        test_1:
                strcpy(test_case_1_input, test_file_name_input)
                strcpy(test_case_1_output, test_file_name_output)
                jal     test_logic

        test_2:
                strcpy(test_case_2_input, test_file_name_input)
                strcpy(test_case_2_output, test_file_name_output)
                jal     test_logic

        test_3:
                strcpy(test_case_3_input, test_file_name_input)
                strcpy(test_case_3_output, test_file_name_output)
                jal     test_logic

        test_4:
                strcpy(test_case_4_input, test_file_name_input)
                strcpy(test_case_4_output, test_file_name_output)
                jal     test_logic

        test_5:
                strcpy(test_case_5_input, test_file_name_input)
                strcpy(test_case_5_output, test_file_name_output)
                jal     test_logic

        test_6:
                strcpy(test_case_6_input, test_file_name_input)
                strcpy(test_case_6_output, test_file_name_output)
                jal     test_logic


        exit()
```

- Рузультат вывода автотестирования с параметром вывода - **да**.

```
Autotest the program (Y/N): **** user input : Y
Output results to console (Y/N): **** user input : Y

File path input: TestData/test1.txt
Output: AJSDJHFuiY823YJaHFXZNMVCNeJWR90 iaSFMNaDFN aSD
File name ouput: TestData/test1_result.txt

File path input: TestData/test2.txt
Output: 2312`123`1231`124123412341234aiaiai
123aSD
41
234
1234eeeeWT
1234
aaD
File name ouput: TestData/test2_result.txt

File path input: TestData/test3.txt
Output: aSDFaSDFKaSDJFioaSDFKMQeRJNFMNM,FNa,SDMFN,MaSD
File name ouput: TestData/test3_result.txt

File path input: TestData/test4.txt
Output: ASNDFJASDJASIOFJASFNJASFAS"DA"SF
File name ouput: TestData/test4_result.txt

File path input: TestData/test5.txt
Output: )!@#)($*!@__+{ASD)(@)#$**!#@*SKFJ
File name ouput: TestData/test5_result.txt

File path input: TestData/test6.txt
Output: aa_ii_uu_oo_aa_oo_PP_BB
File name ouput: TestData/test6_result.txt

-- program is finished running (0) --
```

- Рузультат вывода автотестирования с параметром вывода - **нет**.

```
Autotest the program (Y/N): **** user input : Y
Output results to console (Y/N): **** user input : N

File path input: TestData/test1.txt
File name ouput: TestData/test1_result.txt

File path input: TestData/test2.txt
File name ouput: TestData/test2_result.txt

File path input: TestData/test3.txt
File name ouput: TestData/test3_result.txt

File path input: TestData/test4.txt
File name ouput: TestData/test4_result.txt

File path input: TestData/test5.txt
File name ouput: TestData/test5_result.txt

File path input: TestData/test6.txt
File name ouput: TestData/test6_result.txt

-- program is finished running (0) --
```

- Работа всех тестов прошла успешно.

Вывод

Работа соответствует всем требованиям на каждую оценкую. Тестирование реализовано очень удобно и понятно. Программа работает корректно и без ошибок.