Отчет ИДЗ-1

Исполнитель: Светличный Лев Алексеевич

Группа: БПИ 225

Вариант задания: №3

Условие.

Разработать программу, которая вводит одномерный массив A, состоящий из N элементов (значение N вводится при выполненпии программы),после чего формирует из элементов массива A новый массив B по правилам, указанным в варианте, и выводит его. Память под массивы может выделяться статически, на стеке, автоматичеси по выбору разработчика с учетом требований к оценке работы.

При решении задачи необходимо использовать подпрограммы для реализации ввода, вывода и формирования нового массива массива. Допустимы (при необходимости) дополнительные подпрограммы. Максимальное количество элементов в массиве не должно превышать 10 (ограничение обуславливается вводом данных с клавиатуры). При этом необходимо обрабатывать некорректные значения как для нижней, так и для верхней границ массивов в зависимости от условия задачи.

Сформировать массив B из сумм соседних элементов A по следующим правилам: $B_0 = A_0 + A_1, B_1 = A_1 + A_2, \dots$

Код (main.asm)

```
.include "input_output/read_number.s"
.include "input_output/read_array.s"
.include "input_output/write_array.s"

.data
A_str:  .asciz "\nFirst array input:\n"
B_str:  .asciz "\nGenerated array according to rule:\n"
.align 2
# arrays A and B
array_A:  .space 40
array_B:  .space 40
.text
main:
  li  s1 2    # setting boundaries for input
  li  s2 10
  # macro: reads number within boundaries and saves it to given register (read more in macros/read_number)
  read_number(s0, s1, s2)

  # loading array "pointers"
  la  s1 array_A
  la  s2 array_B
  # macro: reads array of given size and saves it in given address (read more in macros/read_array)
  read_array(s1, s0)

  addi  sp sp -12 # saving arguments on stack:
  sw  s0 8(sp)  # - size
  sw  s1 4(sp)  # - source
  sw  s2 (sp) # - destination
  jal generate3 # calling function that generates B basaed on A with arguments on stack
  lw  s3 (sp) # return value exception code, saved on stack
  addi  sp sp 4 # "0" - all good, no exception
```

```
  bnez  s3 exit # "1" - overflow while counting -> program should be ended
  # if overflow occured during B generation, program ends here

  la  s3 A_str  # providing func with necessary string
  # macro: writes string and array of given size
  write_array(s3, s1, s0)
  addi  s0 s0 -1  # setting size to size of B (it is 1 element less than A)
  la  s3 B_str  # providing func with necessary string
  # macro: writes string and array of given size
  write_array(s3, s2, s0)

exit:
  li  a7 10 # exit program
  ecall

.include "generator.s"
```

Код generator.s - файла с основной подпрограммой

```
.data
overf:  .asciz "\nOverflow occured while adding elements: "
space_g:  .asciz " "
.text
generate3:
  lw  t0 4(sp)    # loading array addresses from stack
  lw  t1 (sp)
  addi  sp sp 8
  li  t2 1      # adding counter

loop_g:
  lw  t3 (t0)   # loading A_i, A_{i+1}
  lw  t4 4(t0)
  xor t5 t3 t4    # check for overflow
  bgez t5 overflow_warning

return:       # come back here if check is false
  add t3 t3 t4    # adding them
  sw  t3 (t1)   # saving the result to B_{i}

  addi  t0 t0 4   # moving counters
  addi  t1 t1 4
  addi  t2 t2 1
  lw  t3 (sp)   #loading array size
  blt t2 t3 loop_g  # if not the end -> back to loop
  addi  sp sp 4   # popping the stack because we no longer need size
  ret

overflow_warning:   # determining the sign of possible oveflow
  add   t5 t3 t4
  bgtz  t3 pos_overflow
  bltz  t3 neg_overflow
  b return

pos_overflow:     # if pos + pos = neg -> overflow
  bltz  t5 overflow
  b return

neg_overflow:     # if neg + neg = pos -> overflow
  bgtz  t5 overflow
  b return

overflow:
  la  a0 overf    # notification output
  li  a7 4
  ecall
  mv  a0 t3   # showing which numbers caused overflow
```

```
  li  a7 1      # first number
  ecall
  la  a0 space_g  # separating them
  li  a7 4
  ecall
  mv  a0 t4   # second number
  li  a7 1
  ecall
  sw  a7 (sp)   # saving info that there is exception on stack
  ret
```

Код (test.asm)

```
.data
success:  .asciz "\nProgram finished successfully. All tests passed."
fail:   .asciz "\nProgram failed current test - "
.align 2
A:  .space 40
B:  .space 40
.text

# Test 1 - Classic
  la  s0 A
  la  s1 B
  li  s2 10

  # adding data to array
  li  a0 1
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 2
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 3
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 4
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 5
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 6
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 7
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 8
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 9
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 10
  sw  a0 (s0)
  addi  s0 s0 4

  # generating array
  la  s0 A
  li  s2 10
  addi  sp sp -12 # saving arguments on stack:
  sw  s2 8(sp)  # - size
  sw  s0 4(sp)  # - source
  sw  s1 (sp) # - destination
  jal generate3 # calling function that generates B basaed on A with arguments on stack
```

```
  lw  s3 (sp) # return value exception code, saved on stack
  addi  sp sp 4 # "0" - all good, no exception
  bnez  s3 exit

  # checking generated data
  la  s0 B
  lw  a0 (s0)
  li  a1 3
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 5
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 7
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 9
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 11
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 13
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 15
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 17
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 19
  bne a0 a1 failed
  addi  s0 s0 4

# Test 2 - Negative numbers too
  la  s0 A
  la  s1 B
  li  s2 8

  # adding data to array
  li  a0 -1
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 -7
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 9
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 0
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 24
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 -11
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 19
  sw  a0 (s0)
```

```
  addi  s0 s0 4
  li  a0 -19
  sw  a0 (s0)
  addi  s0 s0 4

  # generating array
  la  s0 A
  li  s2 10
  addi  sp sp -12 # saving arguments on stack:
  sw  s2 8(sp)  # - size
  sw  s0 4(sp)  # - source
  sw  s1 (sp) # - destination
  jal generate3 # calling function that generates B basaed on A with arguments on stack
  lw  s3 (sp) # return value exception code, saved on stack
  addi  sp sp 4 # "0" - all good, no exception
  bnez  s3 exit

  # checking generated data
  la  s0 B
  lw  a0 (s0)
  li  a1 -8
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 2
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 9
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 24
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 13
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 8
  bne a0 a1 failed
  addi  s0 s0 4
  lw  a0 (s0)
  li  a1 0
  bne a0 a1 failed
  addi  s0 s0 4

# Test 3 - Positive Overflow
  la  s0 A
  la  s1 B
  li  s2 8

  # adding data to array
  li  a0 2000000000
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 1000000000
  sw  a0 (s0)
  addi  s0 s0 4

  # generating array
  la  s0 A
  li  s2 2
  addi  sp sp -12 # saving arguments on stack:
  sw  s2 8(sp)  # - size
  sw  s0 4(sp)  # - source
  sw  s1 (sp) # - destination
  jal generate3 # calling function that generates B basaed on A with arguments on stack
```

```
  lw  s3 (sp) # return value exception code, saved on stack
  addi  sp sp 4 # "0" - all good, no exception
  beqz  s3 failed

# Test 4 - Negative Overflow
  la  s0 A
  la  s1 B
  li  s2 8

  # adding data to array
  li  a0 -2000000000
  sw  a0 (s0)
  addi  s0 s0 4
  li  a0 -1000000000
  sw  a0 (s0)
  addi  s0 s0 4

  # generating array
  la  s0 A
  li  s2 2
  addi  sp sp -12 # saving arguments on stack:
  sw  s2 8(sp)  # - size
  sw  s0 4(sp)  # - source
  sw  s1 (sp) # - destination
  jal generate3 # calling function that generates B basaed on A with arguments on stack
  lw  s3 (sp) # return value exception code, saved on stack
  addi  sp sp 4 # "0" - all good, no exception
  beqz  s3 failed
# If all is good
  la  a0 success
  li  a7 4
  ecall
  b exit
failed:
  la  a0 fail
  li  a7 4
  ecall
exit:
  li  a7 10
  ecall

.include "generator.s"
```

Код из библиотеки ввода/вывода

read_number.s

```
.data
input:  .asciz "Input number of elements ["
to:   .asciz " to "
bracket:  .asciz "]: "
wrong:  .asciz "Wrong input. Try again.\n"

# macro: writes the string with two numbers in it (to avoid repetition in the other macro in this file)
# 1st argument: 1st number
# 2nd argument: 2nd number
.macro print_str (%min, %max)
  la  a0 input
  li  a7 4
  ecall
  mv  a0 %min
  li  a7 1
  ecall
  la  a0 to
  li  a7 4
```

```
  ecall
  mv  a0 %max
  li  a7 1
  ecall
  la  a0 bracket
  li  a7 4
  ecall
.end_macro

# macro: reads number within boundaries and saves it to given register (to avoid repetition in the other macro in this file)
# 1st argument: register to save the number
# 2nd argument: min value of number to be read
# 3rd argument: max value of number to be read
.macro read_number(%reg, %min, %max)
loop:
  print_str(%min, %max)

  li  a7 5      # n input
  ecall

  blt a0 %min again # branching if input is incorrect
  bgt a0 %max again
  b end     # if input is correct, finish program

again:
  la  a0 wrong    # indicate that input was incorrect
  li  a7 4
  ecall
  b   loop      # return to loop: try again
end:
  mv  %reg a0
.end_macro
```

read_array.s

```
.data
input_a:  .asciz "\nEnter array elements on separate lines:\n"

# macro: reads array of given size and saves it in given address
# 1st argument: destination address
# 2nd argument: size of array to be saved
.macro read_array (%dest, %size)
  la  a0 input_a  # requesting array input
  li  a7 4
  ecall
  mv  t0 %dest    # saving array address locally
  mv  t1 zero
loop:
  li  a7 5      # element input
  ecall
  sw  a0 (t0)   # saving element
  addi  t0 t0 4   # moving index
  addi  t1 t1 1
  blt t1 %size loop # check if enough
.end_macro
```

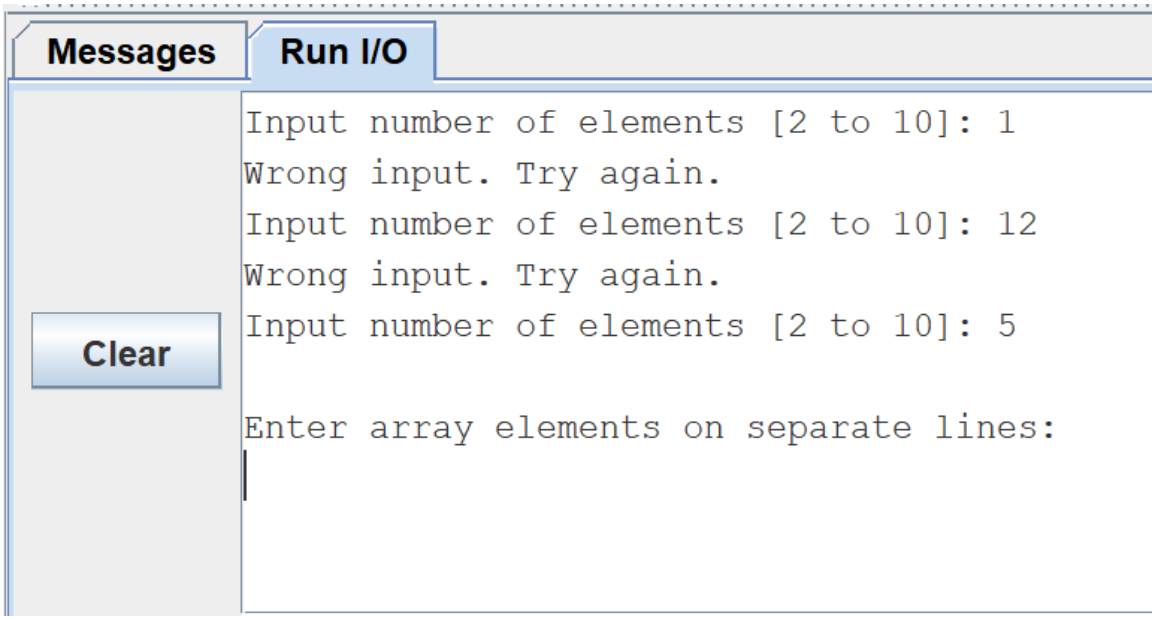
write_array.s

```
.data
n_line: .asciz "\n"
space:  .asciz " "
#macro: writes string and array of given size (to avoid repetition in the other macro in this file)
# 1st argument: string corresponding to array
```

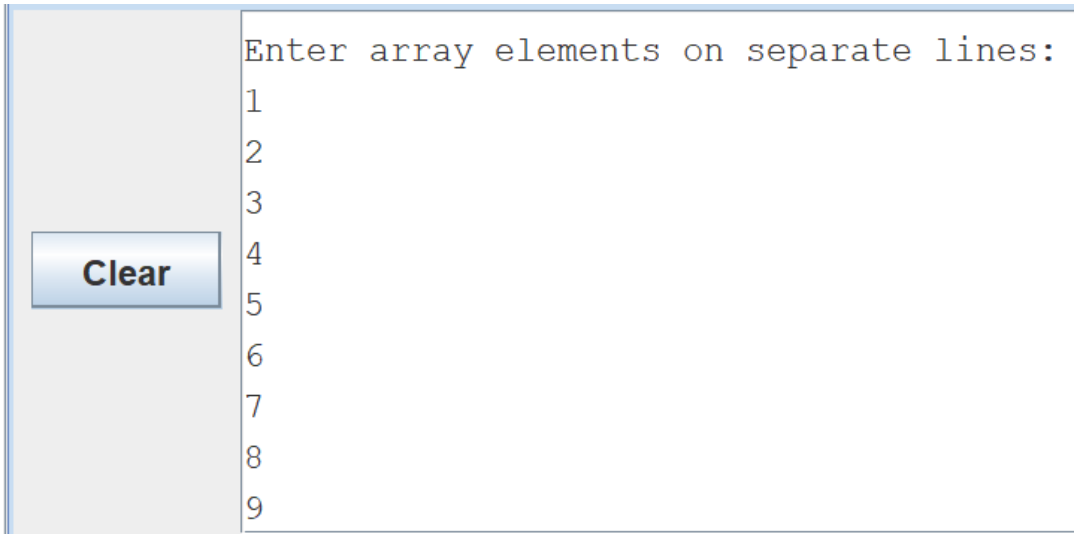
```
# 2nd argument: array address to be put out
# 3rd argument: size of the array
.macro write_array(%str, %arr, %size)
  mv  a0 %str   # string to output
  li  a7 4
  ecall
  mv  t0 %arr   # loading array A start address
  mv  t1 zero

loop:
  lw  a0 (t0)   # loading array element
  li  a7 1
  ecall
  la  a0 space    # spacing
  li  a7 4
  ecall
  addi  t0 t0 4   # increasing counter
  addi  t1 t1 1
  blt t1 %size loop
.end_macro
```

Тесты

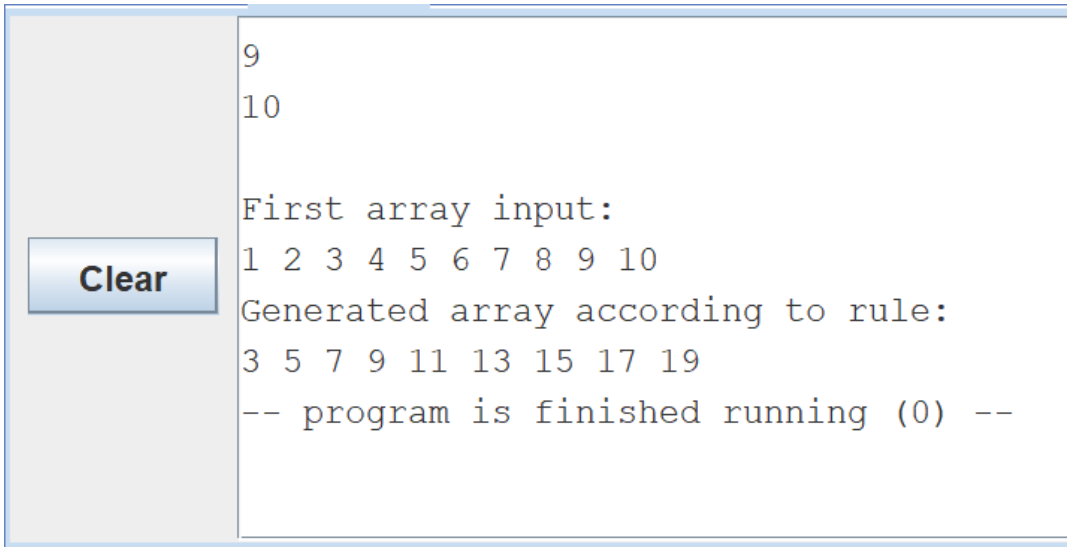
Неправильный ввод количества элементов (10 - верхняя граница из условия, 2 - нижняя, т.к. если в A будет меньше элементов, невозможно сгенерировать B).



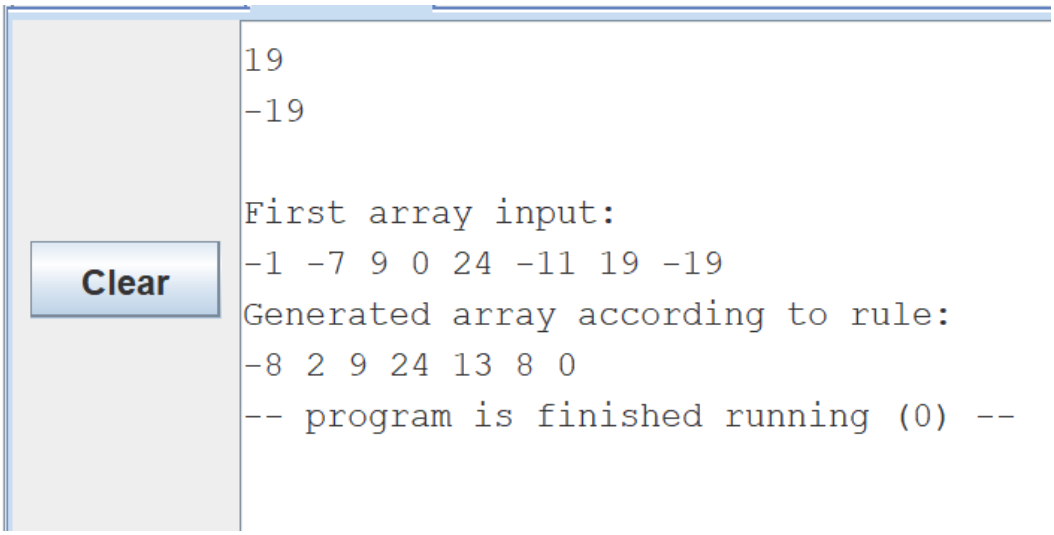
Как организован ввод:



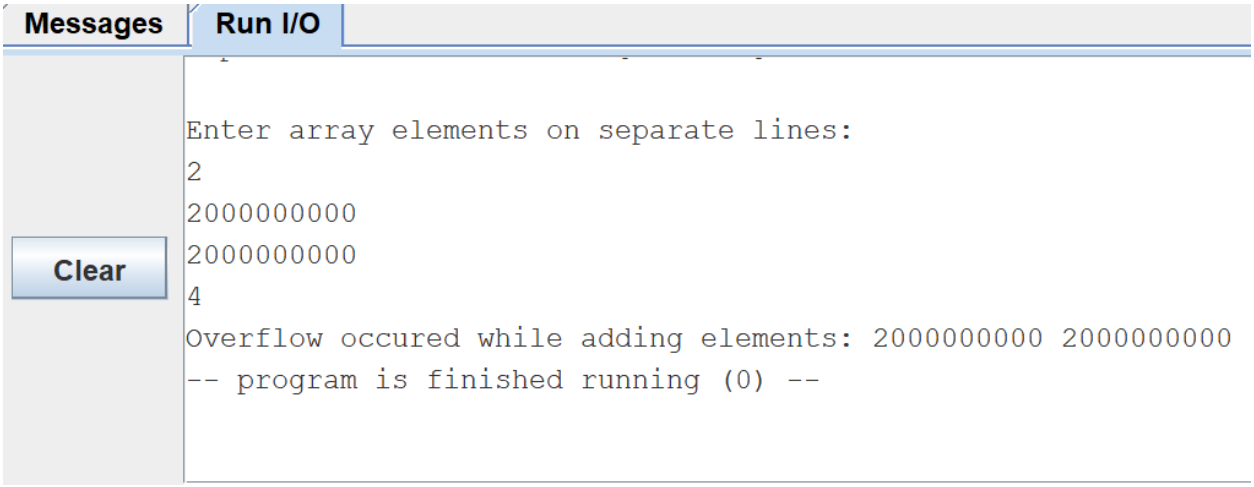
Вывод:



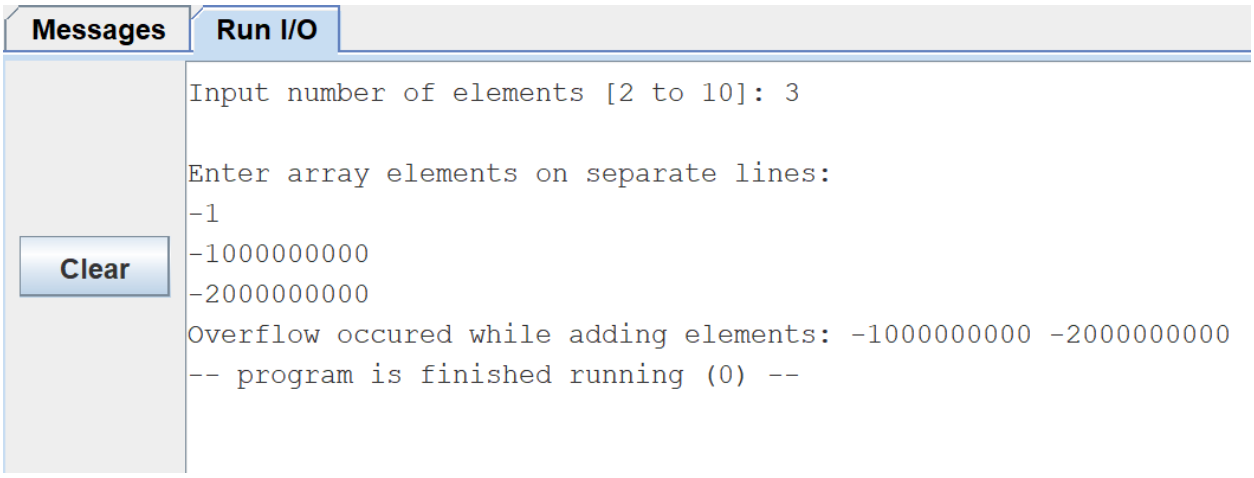
Также работает при отрицательных числах:



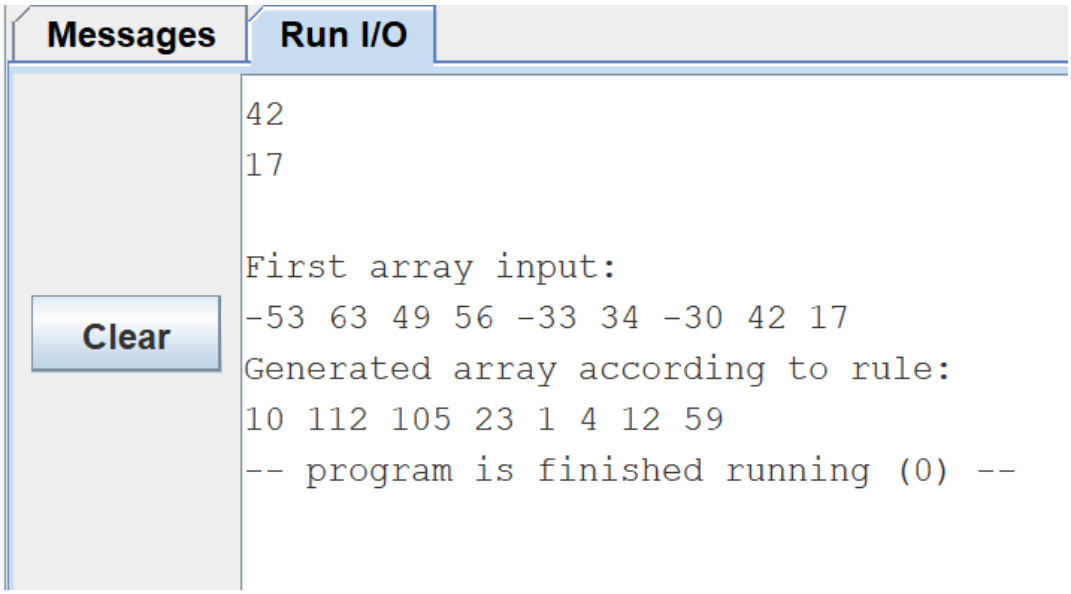
Если случается переполнение при генерации массива B, вызвавшие это дело элементы выводятся на экран, и программа завершается:

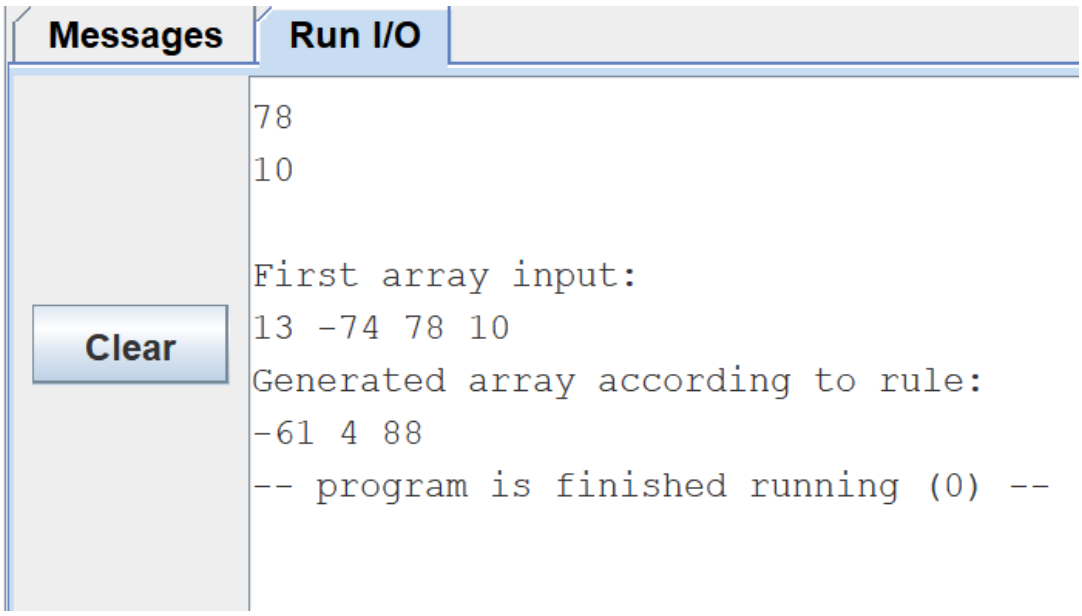


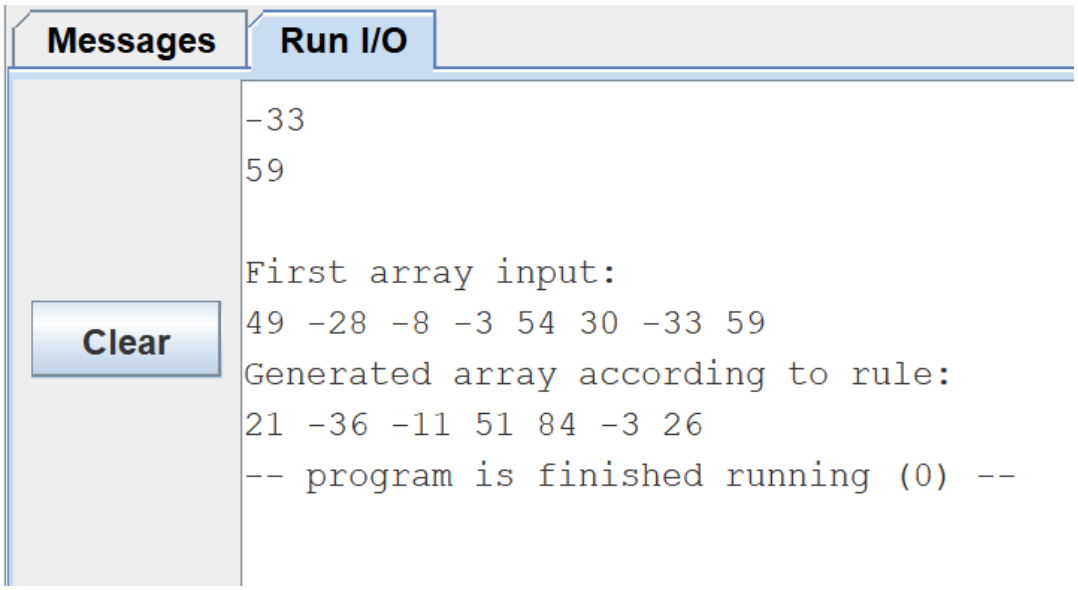
Работает также и при отрицательном переполнении:



Тестовые прогоны







Подтверждение критериев

4-5 баллов

- Решение приведено. Ввод осуществлен с клавиатуры, вывод на дисплей. (см. код).
- Комментарии присутствуют (см. код).
- В отчете представлены результаты тестирования и тестовых прогонов со случайно сгенерированными значениями (см. пункты выше).
- Использование подпрограмм без параметров и локальных переменных допускается, но не используется, т.к. выполнены критерии на более высокий балл.

6-7 баллов

- Используются подпрограммы с передачей аргументов через параметры, отображаемые на стек (см. подпрограмму generate3, где аргументы и возвращаемые значения записаны на стек).
- Внутри подпрограмм используются локальные переменные, которые во время компиляции отображаются на стек.
- В местах вызова функции присутствуют комментарии, описывающие передачу фактических параметров и перенос возвращаемого результата.

8 баллов

- Разработанные программы поддерживают многократное использование с различными наборами исходных данных, включая возможность подключения различных исходных и результирующих массивов.
- Реализовано автоматизированное тестирование за счет создания дополнительной тестовой программы tets.asm, осуществляющей прогон подпрограммы обработки массивов с различными тестовыми данными (вместо ввода данных). Осуществлен прогон тестов обеспечивающих покрытие различных ситуаций.

9 баллов

- Вв программу используются макросы для реализации ввода и вывода данных. Макросы поддерживают повторное использование с различными массивами и другими параметрами.
- Уже разработанные подпрограммы юыли обернуты макросами.

10 баллов

- Программа разбита на несколько единиц компиляции (main.asm и test.asm). При этом подпрограммы ввода–вывода составляют унифицированные модули, используемые повторно как в main.asm, так и в test.asm.
- Макросы выделены в отдельную автономную библиотеку input_output.
- Отчет расширен, дополнен новыми данными.

Дополнительная информация

Ссылка на github: https://github.com/VoRtEx19/AoCS-IHW-1-Var-3