

АВС ИДЗ 1

Власов Николай Алексеевич, БПИ229

14 октября 2023 г.

Условие

Вариант 3: Разработать программу, которая вводит одномерный массив A , состоящий из N элементов (значение N вводится при выполнении программы), после чего формирует из элементов массива A массив B из сумм соседних элементов A по следующим правилам: $B_0 = A_0 + A_1, B_1 = A_1 + A_2, \dots$. Будем считать, что массив B имеет размер $N - 1$, из чего следует, что $2 \leq N \leq 10$ (при $N = 1$ массив B будет иметь неинтересный размер 0).

Тесты, демонстрирующие проверку разработанных программ.

Листинг 1: Код, генерирующий тесты

```
1  from random import randint, seed
2
3  seed(10)
4
5  rand_size = lambda: randint(1, 10)
6  rand_array = lambda size: [randint(-1000, 1000) for _ in range(size)]
7  get_output_array = lambda size, array: [array[i] + array[i + 1]
8                                          for i in range(size - 1)]
9  array_to_string = lambda array, sep: sep.join(map(str, array))
10
11 def generate():
12     for i in range(1, 11):
13         s = rand_size()
14         a = rand_array(s)
15         b = get_output_array(s, a)
16
17         print(f"CASE#{i}:")
18         print(f"Size_{s}\nInput_array:_{array_to_string(a, ' ')}\n"
19               "Output_array:_{array_to_string(b, ' ')}\n~~~~~")
20
21 if __name__ == "__main__":
22     generate()
```

Листинг 2: Файл test_data.asm, в котором лежат сгенерированные для тест-кейсов данные

```
1  .data # test cases
2  # wrong size tests
3  negative_n: .word -1
4  zero_n: .word 0
5  big_n: .word 11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
6
7  # simple tests
8  n1: .word 5
9  arr1: .word 1, 2, 3, 4, 5
10 b1: .word 3, 5, 7, 9
11 n2: .word 10
12 arr2: .word 1, 3, 5, 7, 9, 11, 13, 15, 17, 19
13 b2: .word 4, 8, 12, 16, 20, 24, 28, 32, 36
14
15 # random tests
16 test_n1: .word 10
17 test_a1: .word -934, -122, -12, 183, -970, -578, -53, 665, 6, 686
18 test_b1: .word -1056, -134, 171, -787, -1548, -631, 612, 671, 692
19 test_n2: .word 5
20 test_a2: .word 338, 660, -672, -930, 66
21 test_b2: .word 998, -12, -1602, -864
22 test_n3: .word 8
23 test_a3: .word -329, -845, -489, 950, 951, 526, -261, -909
24 test_b3: .word -1174, -1334, 461, 1901, 1477, 265, -1170
25 test_n4: .word 7
26 test_a4: .word 761, -716, 235, -273, -219, -138, -420
27 test_b4: .word 45, -481, -38, -492, -357, -558
28 test_n5: .word 5
29 test_a5: .word -65, -643, 404, -380, 355
30 test_b5: .word -708, -239, 24, -25
31 test_n6: .word 6
32 test_a6: .word -728, -65, 572, 761, -510, 985
33 test_b6: .word -793, 507, 1333, 251, 475
34 test_n7: .word 8
35 test_a7: .word 256, -232, -910, 193, -992, -518, -726, -601
36 test_b7: .word 24, -1142, -717, -799, -1510, -1244, -1327
37 test_n8: .word 5
38 test_a8: .word 98, -251, 581, 839, -509
39 test_b8: .word -153, 330, 1420, 330
40 test_n9: .word 6
41 test_a9: .word 365, 124, -78, -108, -38, -867
42 test_b9: .word 489, 46, -186, -146, -905
43 test_n10: .word 10
44 test_a10: .word -336, 728, 27, -680, 724, -541, 986, -155, -512, -925
45 test_b10: .word 392, 755, -653, 44, 183, 445, 831, -667, -1437
```

Результаты тестовых прогонов для различных исходных данных.

```
Enter 0 to exit, 1 to run the program or 2 to run the tests: 2
Wrong size tests:
Test passed.
Test passed.
Test passed.

Simple tests:
Test passed.
Test passed.

Random tests:
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
```

```
Enter 0 to exit, 1 to run the program or 2 to run the tests: 1
Enter size of array between 1 and 10: 6
Enter elements of array (one per line):
1
2
1
2
1
2
Output array: 3 3 3 3 3
Enter 0 to exit, 1 to run the program or 2 to run the tests: 1
Enter size of array between 1 and 10: 4
Enter elements of array (one per line):
100
1000
1000000
1000
Output array: 1100 1001000 1001000
Enter 0 to exit, 1 to run the program or 2 to run the tests: 0

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

Как видно, все тесты программа проходит успешно

Тексты программы на языке ассемблера.

Листинг 3: main.asm

```
1  # includes
2  .include "messages.asm"
3  .include "macrolib.asm"
4
5  .text
6      main:
7          print_str(prompt1)
8          input_int(s1) # input command
9
10         li t0 1
11         li t1 2
12
13         mv s2 sp # write sp to recover it for efficient use of the stack
14
15         beqz s1 jal_exit # if s1 == 0 go to exit
16         beq s1 t0 jal_user # if s1 == 1 go to user input
17         beq s1 t1 jal_test # if s2 == 2 go to running tests
18
19         print_str(wrong_choice) # else wrong command was chosen
20
21         j main # repeat solution
22
23     jal_exit:
24         j exit # go to exit
25
26     jal_user:
27         jal run_user # call function for user input
28         mv sp s2 # recover sp
29         j main # repeat solution
30
31     jal_test:
32         jal run_test # call function for running tests
33         mv sp s2 # recover sp
34         j main # repeat solution
35
36     exit: # exit
37         li a7 10
38         ecall
39
40 # includes
41 .include "user.asm"
42 .include "test.asm"
43 .include "input.asm"
44 .include "output.asm"
45 .include "solution.asm"
```

Листинг 4: user.asm

```

1  .text
2      run_user:
3          mv s3 ra # write ra because we will use jal later
4
5          jal input_array # call function for reading and writing array
6          beqz a1 wrong_size_error # check flag (look input.asm)
7          mv a3 sp # write B.begin() to a3
8          jal solve # call function for solving problem
9          addi a4 a1 -1 # write B.size() to a4
10         jal output_array # call function for printing array
11
12         mv ra s3 # recover ra
13         ret
14
15     wrong_size_error:
16         mv ra s3 # recover ra
17         ret

```

Листинг 5: output.asm

```

1  .text
2  output_array: # params: B.begin() in a3, B.size() in a4
3      mv t0 a3 # write B.begin() to t0
4      mv t1 a4 # write B.size() to t1
5
6      print_str(prompt4)
7      output_while: # print elements of array
8          addi t0 t0 -4
9          lw t2 4(t0)
10         print_int(t2)
11         print_str(tab)
12         addi t1 t1 -1
13         bnez t1 output_while
14
15     print_str(endl)
16
17     ret

```

Листинг 6: input.asm

```

1  .text
2  input_array:  # return A.size() in a1, A.begin() in a2
3      print_str(promt2)
4
5      # lines from 7 to 11 are tested by test_corner
6
7      input_int(a1)  # write A.size() to a1
8      li t0 1
9      ble a1 t0 fail  # check left bound for N
10     li t0 10
11     bgt a1 t0 fail  # check right bound for N
12
13
14     mv t0 a1
15     li t1 -4
16     mul t0 t0 t1
17
18     mv a2 sp  # write A.begin() to a2
19     add sp sp t0  # move sp to A.end()
20
21     mv t0 a1
22     mv t1 a2
23
24     print_str(promt3)
25     input_while:  # input elements of array
26         input_int(t2)
27         addi t1 t1 -4
28         sw t2 4(t1)
29         addi t0 t0 -1
30         bnez t0 input_while
31
32     ret
33
34
35 fail:
36     print_str(wrong_size)
37     li a1 0  # flag indicating that an incorrect size has been entered
38     ret

```

Листинг 7: macrolib.asm

```

1  .macro print_str(%str)  # macro for printing string from address %str
2      la a0 %str
3      li a7 4
4      ecall
5
6  .end_macro
7
8  .macro print_int(%reg)  # macro for printing integer from register %reg
9      mv a0 %reg
10     li a7 1
11     ecall
12 .end_macro
13
14 .macro input_int(%reg)  # macro for reading integer and writing to register %reg
15     li a7 5
16     ecall
17     mv %reg a0
18 .end_macro
19
20 .macro test_corner(%n)  # macro for testing cases with wrong array size
21     la t2 %n
22     lw t1 (t2)
23     li t0 1
24     ble t1 t0 fail  # check left bound for N
25     li t0 10
26     bgt t1 t0 fail  # check right bound for N
27     j ok  # bound are correct
28
29     fail:
30         print_str(test_passed)  # program successfully detect an error
31         j ex
32
33     ok:
34         print_str(test_failed)  # program don't detect an error
35
36     ex:
37 .end_macro
38
39 .macro prep_test_case(%n, %a)
40     # macro for preparing a test case with size
41     # in address %n and input array in address %a
42     la a1 %n
43     la t3 %a
44     lw a1 (a1)
45
46     mv t0 a1
47     li t1 -4
48     mul t0 t0 t1
49
50     mv a2 sp  # write A.begin() to a2
51     add sp sp t0  # move sp to A.end()
52
53     mv t0 a1
54     mv t1 a2
55

```

```

56     input_while: # writing elements of array to stack
57         lw t2 (t3)
58         addi t3 t3 4
59         addi t1 t1 -4
60         sw t2 4(t1)
61         addi t0 t0 -1
62         bnez t0 input_while
63
64     mv a3 sp # writing B.begin() to a3
65 .end_macro
66
67 .macro equal_array(%result_size, %result_begin, %answer_size, %answer_begin)
68     # macro for checking equality of two arrays
69     bne %result_size %answer_size fail # if arrays' sizes are not equal
70
71     mv t0 %result_size
72     mv t2 %answer_begin
73     mv t1 %result_begin
74
75     while: # element-wise comparison
76         lw t3 (t1)
77         lw t4 (t2)
78         bne t3 t4 fail # if current elements of arrays are not equal
79
80         addi t1 t1 -4
81         addi t2 t2 4
82
83         addi t0 t0 -1
84         bnez t0 while
85
86     j ok # element-wise comparison has been ended successfully
87
88     fail: # arrays are not equal
89         print_str(test_failed)
90         j ex
91     ok: # arrays are equal
92         print_str(test_passed)
93     ex:
94 .end_macro

```


Листинг 8: messages.asm

```

1 .data # string data used in program
2     prompt1: .asciz "Enter_0_to_exit_,_1_to_run_the_program_or_2_to_run_the_tests:_"
3     prompt2: .asciz "Enter_size_of_array_between_1_and_10:_"
4     prompt3: .asciz "Enter_elements_of_array_(one_per_line):\n"
5     prompt4: .asciz "Output_array:_"
6     wrong_choice: .asciz "Wrong_choice._You_should_enter_number_0,_1_or_2.\n"
7     wrong_size: .asciz "Wrong_size._You_should_enter_number_between_1_and_10.\n"
8     test_passed: .asciz "Test_passed.\n"
9     test_failed: .asciz "Test_failed.\n"
10    wrong_size_tests: .asciz "Wrong_size_tests:\n"
11    simple_tests: .asciz "Simple_tests:\n"
12    random_tests: .asciz "Random_tests:\n"
13    tab: .asciz "_"
14    endl: .asciz "\n"

```

Листинг 9: solution.asm

```

1 .text
2
3 solve: # params: A.size() in a1, A.begin() in a2, B.begin() in a3
4     mv t0 a2
5     mv t1 a3
6     addi t6 a1 -1 # B.size()
7
8     solve_while: # counting current element of result array
9         lw t2 (t0) # A[i]
10        lw t3 -4(t0) # A[i + 1]
11        add t4 t2 t3 # t4 = A[i] + A[i + 1]
12        sw t4 (t1) # B[i] = (t4) = A[i] + A[i + 1]
13
14        addi t0 t0 4
15        addi t1 t1 4
16        addi t6 t6 1
17
18        bnez t6 solve_while
19
20    mv sp t1 # move sp to the B.end()
21    ret

```

```

1  .include "test_data.asm"
2
3  .macro test_case(%n, %a, %b)
4      # macro was added to make writing run_test faster,
5      # so it is not in the macrolib
6      prep_test_case(%n, %a)
7      jal solve
8      addi a4 a1 -1
9      la t0 %n
10     lw t0 (t0)
11     addi t0 t0 -1
12     la t1 %b
13     equal_array(a4, a3, t0, t1)
14 .end_macro
15
16 .text
17     run_test:
18         mv s4 ra # write ra because we will use jal later
19
20         # run test cases
21         print_str(wrong_size_tests)
22         test_corner(negative_n)
23         test_corner(zero_n)
24         test_corner(big_n)
25         print_str(endl)
26         print_str(simple_tests)
27         test_case(n1, arr1, b1)
28         test_case(n2, arr2, b2)
29         print_str(endl)
30         print_str(random_tests)
31         test_case(test_n1, test_a1, test_b1)
32         test_case(test_n2, test_a2, test_b2)
33         test_case(test_n3, test_a3, test_b3)
34         test_case(test_n4, test_a4, test_b4)
35         test_case(test_n5, test_a5, test_b5)
36         test_case(test_n6, test_a6, test_b6)
37         test_case(test_n7, test_a7, test_b7)
38         test_case(test_n8, test_a8, test_b8)
39         test_case(test_n9, test_a9, test_b9)
40         test_case(test_n10, test_a10, test_b10)
41
42         mv ra s4 # recover ra
43         ret

```

Дополнительная информация, подтверждающая выполнение задания в соответствии требованиям на предполагаемую оценку.

- Приведено решение задачи на ассемблере. Ввод данных осуществляется с клавиатуры. Вывод данных осуществляется на дисплей. ✓
- В программе должны присутствовать комментарии, поясняющие выполняемые действия. ✓
- Допускается использование требуемых подпрограмм без параметров и локальных переменных. ✓
- В отчете должно быть представлено полное тестовое покрытие. Приведены результаты тестовых прогонов. Например, с использованием скриншотов. - **см. страница 3** ✓
- В программе необходимо использовать подпрограммы с передачей аргументов через параметры, отображаемые на стек. ✓
- Внутри подпрограмм необходимо использовать локальные переменные, которые при компиляции отображаются на стек. - **не потребовалось (кроме записи ga)** ✓
- В местах вызова функции добавить комментарии, описывающие передачу фактических параметров и перенос возвращаемого результата. При этом необходимо отметить, какая переменная или результат какого выражения соответствует тому или иному фактическому параметру. - **описаны около определения функций** ✓
- Разработанные подпрограммы должны поддерживать многократное использование с различными наборами исходных данных, включая возможность подключения различных исходных и результирующих массивов. - **функция main позволяет поддерживать многократное использование программы** ✓
- Реализовать автоматизированное тестирование за счет создания дополнительной тестовой программы, осуществляющей прогон подпрограммы обработки массивов с различными тестовыми данными (вместо ввода данных). Осуществить прогон тестов обеспечивающих покрытие различных ситуаций. Тестовые данные можно формировать в различных исходных массивах. - **см. страница 10** ✓
- Добавить в программу использование макросов для реализации ввода и вывода данных. Макросы должны поддерживать повторное использование с различными массивами и другими параметрами. . - **см. страницы 7-8** ✓
- Программа должна быть разбита на несколько единиц компиляции. При этом подпрограммы ввода-вывода должны составлять унифицированные модули, используемые повторно как в программе, осуществляющей ввод исходных данных, так и в программе, осуществляющей тестовое покрытие. ✓
- Макросы должны быть выделены в отдельную автономную библиотеку. - **macrolib.asm** ✓

ВАЖНО!!! В эмуляторе RARS НЕ НУЖНО включать компиляцию всех открытых файлов, компилировать и запускать ТОЛЬКО файл main.asm



Кот для привлечения внимания и поднятия настроения