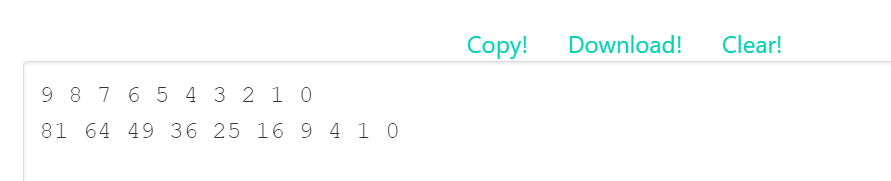
**实验报告**

**练习3：**

运行结果：



实现函数源代码：

.globl map

.text

main:

jal ra, create\_default\_list

add s0, a0, x0 # a0 = s0 is head of node list

#print the list

add a0, s0, x0

jal ra, print\_list

# print a newline

jal ra, print\_newline

# load your args

add a0, s0, x0 # load the address of the first node into a0

# load the address of the function in question into a1 (check out la on the green sheet)

### YOUR CODE HERE ###

la a1,square

# issue the call to map

jal ra, map

# print the list

add a0, s0, x0

jal ra, print\_list

# print another newline

jal ra, print\_newline

addi a0, x0, 10

ecall #Terminate the program

map:

# Prologue: Make space on the stack and back-up registers

### YOUR CODE HERE ###

addi sp,sp,-12

sw ra,0(sp)

sw s0,4(sp)

sw s1,8(sp)

beq a0, x0, done # If we were given a null pointer (address 0), we're done.

add s0, a0, x0 # Save address of this node in s0

add s1, a1, x0 # Save address of function in s1

# Remember that each node is 8 bytes long: 4 for the value followed by 4 for the pointer to next.

# What does this tell you about how you access the value and how you access the pointer to next?

# load the value of the current node into a0

# THINK: why a0?

### YOUR CODE HERE ###

lw a0,0(s0)

# Call the function in question on that value. DO NOT use a label (be prepared to answer why).

# What function? Recall the parameters of "map"

### YOUR CODE HERE ###

jalr ra,a1,0

# store the returned value back into the node

# Where can you assume the returned value is?

### YOUR CODE HERE ###

sw a0,0(s0)

# Load the address of the next node into a0

# The Address of the next node is an attribute of the current node.

# Think about how structs are organized in memory.

### YOUR CODE HERE ###

lw a0,4(s0)

# Put the address of the function back into a1 to prepare for the recursion

# THINK: why a1? What about a0?

### YOUR CODE HERE ###

add a1,x0,s1

# recurse

### YOUR CODE HERE ###

jal ra,map

done:

# Epilogue: Restore register values and free space from the stack

### YOUR CODE HERE ###

lw ra,0(sp)

lw s0,4(sp)

lw s1,8(sp)

addi sp,sp,12

jr ra # Return to caller

square:

mul a0 ,a0, a0

jr ra

create\_default\_list:

addi sp, sp, -12

sw ra, 0(sp)

sw s0, 4(sp)

sw s1, 8(sp)

li s0, 0 # pointer to the last node we handled

li s1, 0 # number of nodes handled

loop: #do...

li a0, 8

jal ra, malloc # get memory for the next node

sw s1, 0(a0) # node->value = i

sw s0, 4(a0) # node->next = last

add s0, a0, x0 # last = node

addi s1, s1, 1 # i++

addi t0, x0, 10

bne s1, t0, loop # ... while i!= 10

lw ra, 0(sp)

lw s0, 4(sp)

lw s1, 8(sp)

addi sp, sp, 12

jr ra

print\_list:

bne a0, x0, printMeAndRecurse

jr ra # nothing to print

printMeAndRecurse:

add t0, a0, x0 # t0 gets current node address

lw a1, 0(t0) # a1 gets value in current node

addi a0, x0, 1 # prepare for print integer ecall

ecall

addi a1, x0, ' ' # a0 gets address of string containing space

addi a0, x0, 11 # prepare for print string syscall

ecall

lw a0, 4(t0) # a0 gets address of next node

jal x0, print\_list # recurse. We don't have to use jal because we already have where we want to return to in ra

print\_newline:

addi a1, x0, '\n' # Load in ascii code for newline

addi a0, x0, 11

ecall

jr ra

malloc:

addi a1, a0, 0

addi a0, x0 9

ecall

jr ra

实验4：