Ean Dodge

Lab2-1

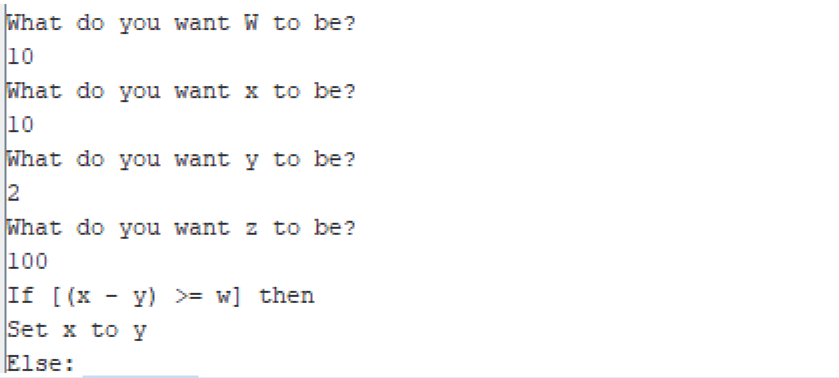
```
.macro print (%problem)
li $v0 4
       la $a0 %problem            #Macro to print
       syscall
       .end_macro
.text

####################################
print(w_output)
                        #Getiing W
li $v0 5
la $a0 w_input
li $a1 6
syscall
move $s0, $v0
###################################

print(x_output)
                        #Getting x
li $v0 5
la $a0 x_input
li $a1 6
syscall
move $s1, $v0
####################################
print(y_output)
                        #Getting y
li $v0 5
la $a0 y_input
li $a1 6
syscall
move $s2, $v0
######################################
print(z_output)
                        #Getting z
li $v0 5
la $a0 z_input
li $a1 6
syscall
move $s3, $v0

```

```
#################################
print(problem1)
        print(problem2)
        print(problem3)  #printing the problem
        print(problem4)
        print(problem5)
        print(problem6)
######################################

sub $t0, $s1, $s2 # x - y
slt $t1, $s0, $t0  #w<(x-y)
bne $t1, $zero, xtoy #if w<(x-y) then x to y
beq $s0, $t0, xtoy #if w == (x-y) then x to y
j xtoz #else x to z
#####################################
xtoz:
print(resultz)
add $s4, $s3, $zero #x to y
j exit
##################################
xtoy:
print(resulty)
add $s4, $s2, $zero #x to z
j exit
###############################
exit:
add $a0, $s4, $zero        #will print out the x value at the end
li $v0 1
syscall
##################################

.data
w_input: .space 10
x_input: .space 10
y_input: .space 10
z_input: .space 10

w_output: .asciiz "What do you want W to be?\n"
x_output: .asciiz "What do you want x to be?\n"
y_output: .asciiz "What do you want y to be?\n"
z_output: .asciiz "What do you want z to be?\n"


problem1: .asciiz "If [(x - y) >= w] then\n"
```

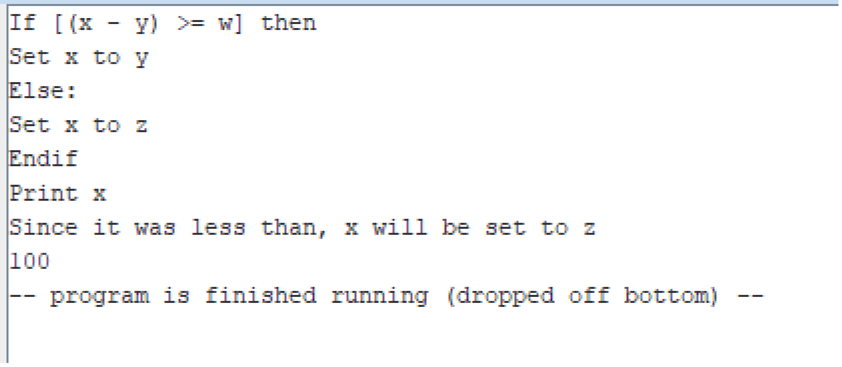
```
86.	problem2: .asciiz "Set x to y\n"
87.	problem3: .asciiz "Else:\n"
88.	problem4: .asciiz "Set x to z\n"
89.	problem5: .asciiz "Endif\n"
90.	problem6: .asciiz "Print x\n"
91.
92.	resulty: .asciiz "Since it was greater than, or equal to, x will be set to y\n"
93.	resultz: .asciiz "Since it was less than, x will be set to z\n"
```

Brief summary:

On line 1, I used a macro to create a print. This made my code look much cleaner. Then after this, I would print a prompt for w, and would then input w. I did this for w, x, y and z. I also thought it would look neat if I outputted the problem, this is on lines 43-48. On line 58 I subtact y from x, since this is in the parenthesis. I then do is w < (x – y). I have to do the opposite result value since I switched this around, so I said if it does not equal zero, meaning that it is a true statement, then I set it to y. If they are equal, then x will be set to x. I do a jump to xtoy(x to y), and in line 64 I set x to be y. If the statement is not true, then it will jump to xtoz(x to z). On line 59, it sets x to be z. In both, I make it jump to exit, where it will print x and end the program.



```
What do you want W to be?
10
What do you want x to be?
10
What do you want y to be?
2
What do you want z to be?
100
If [(x - y) >= w] then
Set x to y
Else:
```



```
If [(x - y) >= w] then
Set x to y
Else:
Set x to z
Endif
Print x
Since it was less than, x will be set to z
100
-- program is finished running (dropped off bottom) --
```

Lab2-2

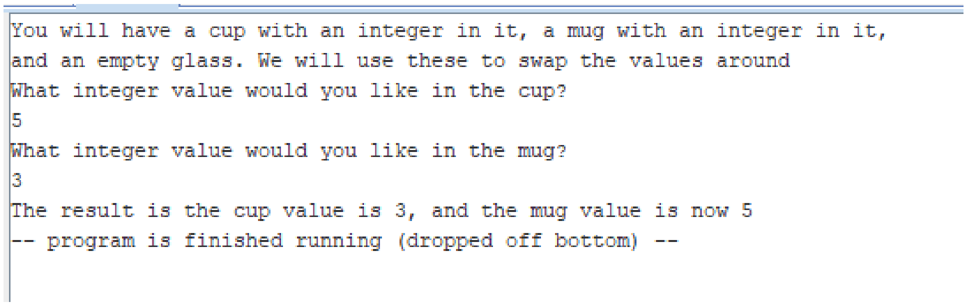
```
1.	.macro print (%problem)
```

```
li $v0 4
	la $a0 %problem		#Macro to print
	syscall
	.end_macro
.text
print(intro_print) #prompt
print(cup_print)   #asking for cup value
li $v0 5
la $a0 get_cup     #getting cup vaue
syscall
move $s0, $v0

print(mug_print)   #asking for mug value
li $v0 5
la $a0 get_mug     #getting mug vaue
syscall
move $s1, $v0

##########################################################

add $t0, $s0, $zero
add $s0, $s1, $zero			#algorithm to swap
add $s1, $t0, $zero

##########################################################
print(result)
add $a0, $s0, $zero
li $v0 1
syscall

print(result1)
add $a0, $s1, $zero
li $v0 1
syscall


.data
intro_print: .asciiz "You will have a cup with an integer in it, a mug with an integer in it, \nand an empty glass. We will use these to swap the values around\n"
cup_print: .asciiz "What integer value would you like in the cup?\n"
mug_print: .asciiz "What integer value would you like in the mug?\n"

get_cup: .space 10
get_mug: .space 10
```

```

result: .asciiz "The result is the cup value is "
result1: .asciiz ", and the mug value is now "
```

Brief summary:

I use the same print macro to print things. This makes my code look much neater. I used a cup, mug, and glass for this swap. This is something that I used in CS1 and brought it back for this. The idea is that I got the value to put in the cup. I got the value for the mug, and then used a temp value to swap them. I put the cup value into the temp. Then I put the mug value into the cup, and then put the temp value into the mug. To swap these values, I used a simple add with zero to move the values around. Once this is all finished, the value in the mug and cup are swapped. I then print them to make sure.



```
You will have a cup with an integer in it, a mug with an integer in it,
and an empty glass. We will use these to swap the values around
What integer value would you like in the cup?
5
What integer value would you like in the mug?
3
The result is the cup value is 3, and the mug value is now 5
-- program is finished running (dropped off bottom) --
```

Conclusion:

One big thing I learned to do were the macros. It is cool to have this thing similar to a function to help make my code neater. I had trouble using the v0 codes, as I kept mixing them up, and it took me forever to find the problem.