Answer to 1 (a)

1. (a) Fill in the code below to produce the Output on the right:

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

i. `print(` [ ] `],` [ ] `])`

Output:

```
Spring Tuesday
```

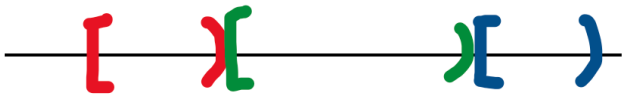
Explanation: This problem tests index and slicing in a string or list.

Word Spring is in string seasons

Letter in left index is included letters in right index is **not** included

index	0	1	2	3	4	5	6	7																	
seasons	+	S	p	r	i	n	g	+	S	u	m	m	e	r	+	F	a	l	l	+	W	i	n	t	e	r
Index starting from right	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

To get Spring from string seasons, if we count from left to right, then the leftmost index is 0, and index is increased one a time when moving from left to right. The index of the first letter in Spring is 1, while the last index is 6, to represent in index of Python, where the start index is included **but** the right index is not included – this makes dividing a segment easier, for example,



Also, [start, end) implies there are end – start elements in this range.

To get Spring, write seasons[1: 7], you can verify that 7-1 = 6, which are the number of characters in “Spring” without double quotes.

Warning: cannot write seasons[1:7] as seasons[1,7], column symbol between 1 and 7 is like to go from 1 to 7. If you use comma, it is seasons is a two-dimensional array, while 1 is the index of the first dimension and 7 is the index of the second dimension.

If you want to count from right to left, then the rightmost index is -1 and each time when you move to the left, index is increased by 1. It is like when you move from west to east, the coordinate is smaller and smaller.

As an alternative, you can write use seasons[-25, -19] to extract “Spring”.

Either seasons[1: 7] or seasons[-25: -19] gives us “Spring”. Note that whichever way you use, the left index is always less than or equal to the right index, also, 7 – 1 = -19 – (-25) = 6, which are the number of letters in the string “Spring” you extract.

Rule of thumb: if the target item is close to the left, then we start to count from left to right, if the target is close to the right, then start to count from right to left, otherwise, the target is close the middle, use whichever counting you feel comfortable.

To get Tuesday, which is in workdays = “Monday?Tuesday?Wednesday?Thursday?”, I will count from left to right since the target is close to the left end. The answer is workdays[7: 14].

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
workdays	M	o	n	d	a	y	?	T	u	e	s	d	a	y	?

1. (a) Fill in the code below to produce the Output on the right:

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

ii.
```
days = long_weekend[          ].split(          )
print("Our weekend has", len(          ),"days.")
```

Output:

```
Our weekend has 3 days.
```

Method split of a string can divide a string into a list of words. For example, print(workdays.split('?')) returns ['Monday', 'Tuesday', 'Wednesday', 'Thursday', ''], where the last '' is the empty string after removing ?.

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
```

Similarly, print(workdays.split(“day”)) outputs ['Mon', '?Tues', '?Wednes', '?Thurs', '?']

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
```

You can think split is to cut the string by pieces.

```
long_weekend = "Friday_Saturday_Sunday"
```

print(long_weekend.split('_'))
output ['Friday', 'Saturday', 'Sunday']

print(long_weekend[:].split('_')) # long_weekend[:] means all the letters in long_weekend.
output ['Friday', 'Saturday', 'Sunday']

days = long_weekend[:].split('_') #save ['Friday', 'Saturday', 'Sunday'] to days
print("Our weekend has", len(days), "days")

The above two statements produce

```
Our weekend has 3 days.
```

1. (a) Fill in the code below to produce the Output on the right:

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

iii.
```
for d in [          ]
    print([          ])
```

Output:

```
FRIDAY
SATURDAY
SUNDAY
```

for d in days:
  print(d)

Code to test the above problem is as follows.

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"

print(seasons[1:7], workdays[7:14])
#cannot replace : by , ie, seasons[1,7] is wrong.
#print(seasons[-25:-19], workdays[7:14]) #also work
```

```
print(workdays.split('?'))
#output ['Monday', 'Tuesday', 'Wednesday', 'Thursday', '']

print(workdays.split('day'))
#output ['Mon', '?Tues', '?Wednes', '?Thurs', '?']

print(long_weekend.split('_'))
# output ['Friday', 'Saturday', 'Sunday']

print(long_weekend[:].split('_'))
# output ['Friday', 'Saturday', 'Sunday']

days = long_weekend[:].split('_')
print("Our weekend has", len(days), "days")

for d in days:
    print(d)
```

Answer to 1 (b)

(b) Consider the following shell commands:

```
$ pwd
/Users/guest
$ ls
bronx.png  circuit.txt   nand.txt  nyc.png    temp
```

So here is the structure. The current folder is guest.

/ -----
|__ Users
|___ guest
|____ bronx.png
|____ circuit.txt
|____ nand.txt
|____ nyc.png
|____ temp

Root directory / has a subdirectory (folder) named Users and Users in turn has a subdirectory guest. Under guest, we have files Bronx.png, circuit.txt, nand.txt, and nyc.png and subdirectory temp. Normally, it is a file name is followed by a suffix, like .png, .txt, .py, and so on, while a folder name does not has a suffix.

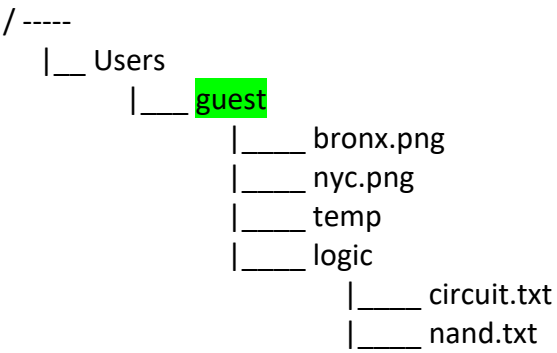
i. What is the output for:

```
$ mkdir logic
$ mv *txt logic
$ ls
```

After command **mkdir logic**, we get

/ -----
|__ Users
|___ guest
|____ bronx.png
|____ circuit.txt
|____ nand.txt
|____ nyc.png
|____ temp
|____ logic

After **mv *.txt logic**, which moves all the files ended with .txt to folder logic, we get



Run command **ls**, which list the contents of the current directory, we get

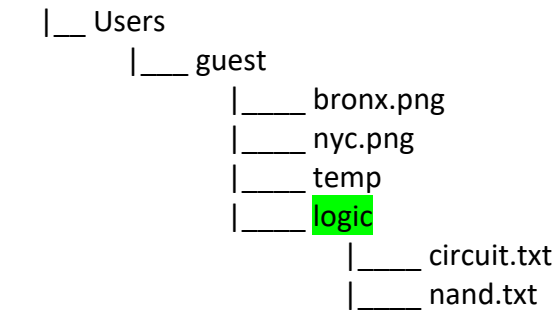
bronx.png nyc.png temp logic

Problem 1 (b) ii

ii. What is the output for:

```
$ cd logic
$ ls
```

After **cd logic**, the current directory is logic.



After **ls**, the output is

circuit.txt nand.txt

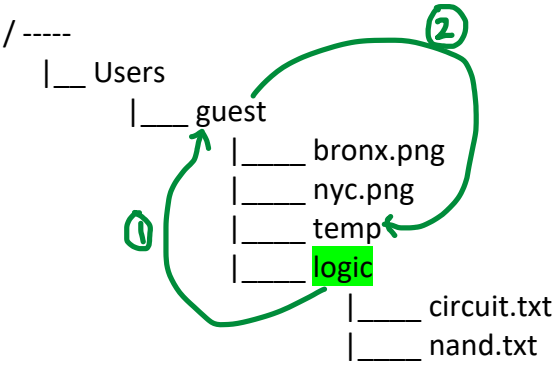
Problem 1 (b) iii

iii. What is the output for:

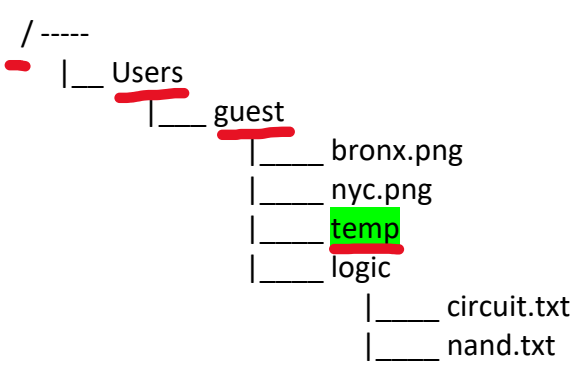
```
$ cd ../temp
$ pwd
```

After cd **../temp**, where .. is the parent directory guest of current directory logic. Then move to temp

Before:



After:



Run command **pwd**, which return the path of current directory. That is,

/Users/guest/temp

In terminal of Mac or WSL.

(1) Type in the commands, that is, **contents after %** with return key. Suppose I am in Downloads directory. Type in cd ~ and press enter key. See current directory changes from Downloads to ~.

```
laptopuser@LaptopUsersMBP2 Downloads % cd ~
laptopuser@LaptopUsersMBP2 ~ %
```

(2) Enter pwd and enter return key.

```
laptopuser@LaptopUsersMBP2 ~ % pwd
```

(3) Output the current directory. My user name is laptopuser, your username can be different.
/Users/laptopuser

(4) Create a guest_dir and move to it. Enter mkdir guest_dir && cd $_ with return key. See the current directory changes from ~ (home directory) to guest_dir.

```
laptopuser@LaptopUsersMBP2 ~ % mkdir guest_dir && cd $_
laptopuser@LaptopUsersMBP2 guest_dir %
```

(5) Command touch is to create an empty file.

```
laptopuser@LaptopUsersMBP2 guest_dir % touch bronx.png
laptopuser@LaptopUsersMBP2 guest_dir % touch circuit.txt
laptopuser@LaptopUsersMBP2 guest_dir % touch nand.txt
laptopuser@LaptopUsersMBP2 guest_dir % touch nyc.png
laptopuser@LaptopUsersMBP2 guest_dir % mkdir temp
laptopuser@LaptopUsersMBP2 guest_dir % ls
bronx.png       nand.txt        temp
circuit.txt     nyc.png
```

(6) Create a directory called logic under current directory. Move all the files ended with .txt to logic.

```
laptopuser@LaptopUsersMBP2 guest_dir % mkdir logic
laptopuser@LaptopUsersMBP2 guest_dir % mv *.txt logic
laptopuser@LaptopUsersMBP2 guest_dir % ls
bronx.png       logic           nyc.png                 temp
```

(7) Move to logic and display its contents.

```
laptopuser@LaptopUsersMBP2 guest_dir % cd logic
```

```
laptopuser@LaptopUsersMBP2 logic % ls
circuit.txt        nand.txt
```

(8) Move to temp directory of parent directory. Display path information of temp.

```
laptopuser@LaptopUsersMBP2 logic % cd ../temp
laptopuser@LaptopUsersMBP2 temp % pwd
/Users/laptopuser/guest_dir/temp
```

(9) Remove guest_dir if we no longer need it or after we finish the purpose of testing. We were in temp directory when we type in cd ../.., where .. means parent directory. So we move from temp to its parent directory guest_dir, then move to the parent directory of guest_dir, which is ~ (home directory).

```
laptopuser@LaptopUsersMBP2 temp % cd ../..
laptopuser@LaptopUsersMBP2 ~ % rm -r guest_dir
```

Be very careful of rm command, after running it, the contents cannot be restored. Unlike move to trash, you can still have a chance to recover the contents. After rm command runs successfully, the things removed are gone. Option -r means recursion. This is useful when you want to remove a non-empty directory, but again, **double check before you press the return key after a rm command.**

Problem 2

Problem 2 (a) i

2. (a) Select the correct option.

r g b

i. What color is tina after this command? `tina.color(1.0,0.0,1.0)`

☐ black ☐ red ☐ white ☐ gray ☐ purple

(1.0, 0.0, 1.0) means red component is 1, green component is 0, and blue component is 1. Red and blue together makes purple.

Problem 2 (a) ii

ii. Select the SMALLEST Binary number:

☐ 1011 ☐ 1101 ☐ 1111 ☐ 1010 ☐ 1110

(1) We are talking about unsigned number. Compare the leftmost (most significant) bit, every number has that bit as 1.
(2) Move to the second most significant digit, ie, 2nd digit from left. Since we are looking for the smallest number, only the one with 0 are possible candidates. We have 1011 and 1010 left.
(3) Compare 1011 and 1011. The third bit (the third most significant bit) from left is the same.
(4) Move to the least significant bit, ie, the rightmost bit. The smallest number is 1010.

It is like to compare number in decimal system, compare the digits in the most significant digit to the least significant one. For example,

(a) 123 is smaller than 200, since the hundred digit 1 in 123 is smaller than hundred digit 2 in 200.
(b) 123 is smaller than 139. With the same hundreds digit, tens digit of 123 is smaller than the tens digit 139.
(c) 123 is smaller than 125. With the same hundreds and tens digits, compare ones digit.

Problem 2 (a) iii

iii. Select the LARGEST Hexadecimal number:

☐ AA ☐ BA ☐ DC ☐ CC ☐ CD

Hexadecimal number is similar to decimal number, the only difference hexadecimal digit is 0, 1, …, 9, A (equivalent to 10), B (equivalent to 11), …, F (equivalent to 15).

Compare the most significant digit, The largest one is D. The answer is DC. It is like decimal number 70 is larger than 69 since the most significant digit in 70 is 7, while the most significant digit in 69 is 6, and 7 is larger than 6.

Problem 2 (a) iv

iv. What is the binary number equivalent to decimal 14?

☐ 1011 ☐ 1101 ☐ 1111 ☐ 1010 ☐ 1110

(1) Divide 14 by 2, the quotient is 7 and the remainder is 0.

```
2  |  14
   +-------
      7      0
```

(2) Divide 7 by 2, the quotient is 3 and the remainder is 1. It is like to divide 7 pens among 2 students, each student get 3 pens, and there is one pen left.

```
2  |  14
   +-------
   2  |  7      0
      +---
      3        1
```

(3) Divide 3 by 2, the quotient is 1 and the remainder is 1. In fact, if the number is odd, when it is divided by 2, the remainder is 1, otherwise, the remainder is 0.

```
2  |  14
   +-------
   2  |  7      0
      +---
   2  |  3      1
      +----
         1      1
```

(4) Divide 1 by 2, the quotient is 0 and the remainder is 1.

```
2  |  14
   +-------
   2  |  7      0
      +---
   2  |  3      1
      +----
    2  |  1     1
       +----
          0     1
```

(5) When the quotient is 0, we can stop. And string the remainders backwards. The answer is 1110.

(6) Quick verify (optional): binary number 1110 is decimal number 14.

exponent	3	2	1	0
rank	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Binary number	1	1	1	0
rank * bit when bit is not zero	1 * 8	1*4	1*2	

Add 1*8 + 1*4 + 1*2 = 14.

Problem 2 (a) v

v. What is the hexadecimal number equivalent to decimal 170?

☐ AA ☐ BA ☐ DC ☐ CC ☐ CD

Answer: same as above, the only different is that the base of hexadecimal number is 16.

(1) Divide 170 by 16, the quotient is 10, the remainder is 10, which is A in hexadecimal system.

```
16 | 170
   +--------
      10        10 = A16
```

(2) Divide 10 by 16, the quotient is 0, the remainder is 10, which is A.

```
16 | 170
   +--------
16 | 10        10 = A16
   +------
     0        10  = A16
```

(3) Stop when quotient is 0. String remainders backwards. We get AA_{16}, which is equivalent to 170 in decimal system.

(4) Verify (optional): hexadecimal number AA is equivalent to 170 in decimal system.

exponent	1	0
rank	$16^1 = 16$	$16^0 = 1$
Hexadecimal number	A	A
Multiple rank and digit	Hexadecimal A is same as 10 in decimal system, 10 * 16	10 * 1

Add 10 * 16 + 10 * 1 = 170.

Problem 2 b (i)

(b) Fill in the code to produce the Output on the right:

```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]

i. for i in range( [ ] , [ ] ):
       print(nums[i], end=" ")
```

Output:

```
23 98 45 11 4 33 29
```

Answer:
There are two occurrences of 23. If we choose the first one, the indices are not evenly spaces so we cannot use range function.

Wrong choice:

```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

Correct choice:

```
Index          0.   1    2.   3    4    5    6    7    8    9    10
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

The answer is as follows. Note that element at index 3 is included, but element at index 10 is not. Also 10 – 3 = 7, which means this range includes 7 elements. Is that pretty?

```
for i in range(3, 10):
    print(nums[i], end=" ")
```

Problem 2 (b) ii

```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]

ii. for j in range( [ ] , [ ] , [ ] ):
        print(nums[j], end=" ")
```

Output:

```
45 45 29
```

Answer:

```
Index          0   1    2    3    4    5    6    7    8    9    10   11
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

Use (start, end, step) version of range. Note that the items we selected are indexed at 1, 5, 9, respectively. Said differently, start at 1, end at 10, increase by 4 each time.

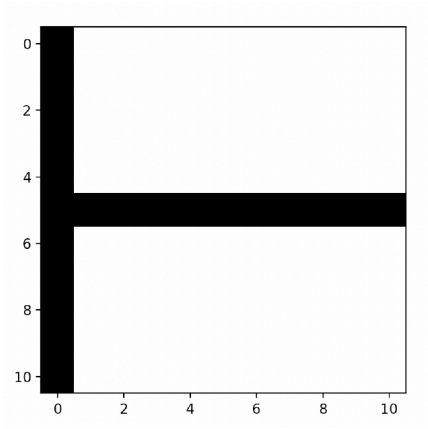
```
for i in range(1, 10, 4):
    print(nums[i], end=" ")
```

Problem 2 b (iii)

iii.

```
import numpy as np
import matplotlib.pyplot as plt
img = np.ones( (11,11,3) )

img[ ____ , ____ , :]  = 0 # black row

img[ ____ , ____ , :]  = 0 # black column
plt.imshow(img)
plt.show()
```

Output:



Explanation:

(1) For the horizontal line, row is indexed at 5, column index is all, so use img[5, :, :]. The third dimension is rgb (red, green, blue) channels, use : means choose all of them. When r, g, b are all zeros, the color is black.

(2) For the vertical line, row index is all and column index is 0, so use img[:, 0, :].

Complete code is as follows.

```
import matplotlib.pyplot as plt
import numpy as np

img = np.ones((11, 11, 3))
img[5, :, :] = 0 #set row
img[:, 0, :] = 0 #set column

plt.imshow(img)
plt.show()
```

Problem 3 (a) i

3. (a) What is the value (True/False):

i.
```
in1 = False
in2 = False
out =  (not in1 and in2) or (not in1 or in2)
```
☐ True ☐ False

When in1 is False and in2 is False, we have

(1) not in1 as True
(2) in2 as False
(3) Then (not in1 and in2) is (True and False), which is False.
(4) Also (not in1 or in2) is (True or False), which is True.
(5) (not in1 and in2) or (not in1 or in2) is False or True, which is True.

Problem 3 (a) ii

ii.
```
in1 = True
in2 = False
in3 = ( not in1 ) or ( not in2 )
out = (not in1 or not in2) and (not in2 and in3)
```

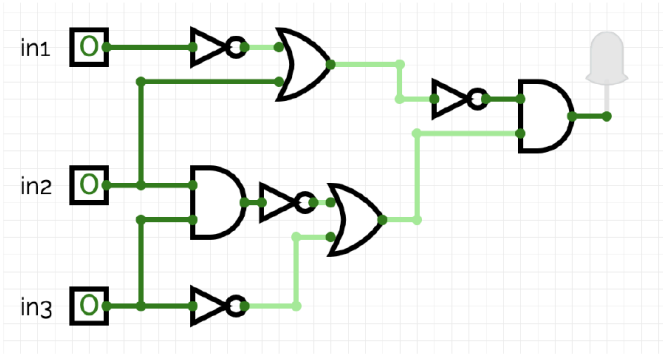
When in1 is True, in2 is False

(1) not in1 is False
(2) not in2 is True
(3) in3 = (not in1) or (not in2) is False or True, which is True.
(4) out = (not in1 or not in2) and (not in2 and in3)
    (4a) (not in1 or not in2) is (False or True), which is True
    (4b) (not in2 and in3) is (True and True) which is True.
    Remember, NOT has higher precedence than AND, which in turn has higher precedence than OR. So we will run not in2 first, before we use the result of not in2 to and with in3.

    so True and True is True.

Problem 3 (a) iii

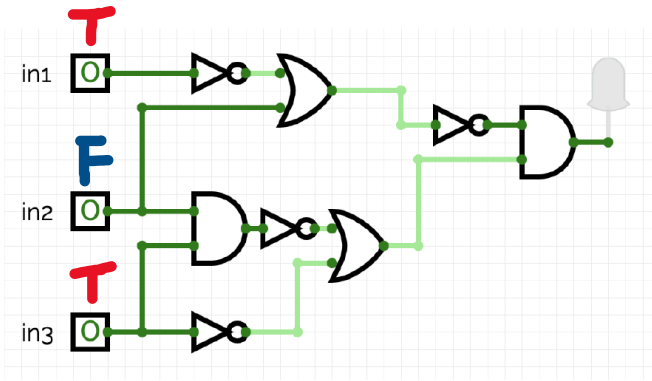
iii.



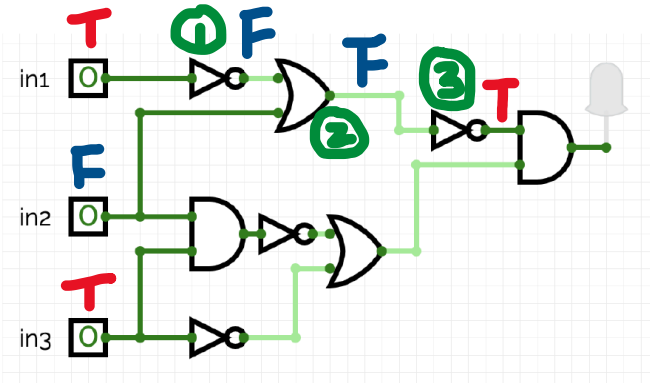

```
in1 = True
in2 = False
in3 = True
```

☐ True ☐ False

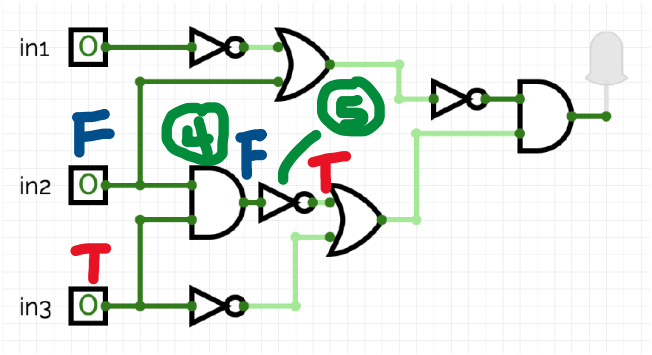
Answer:



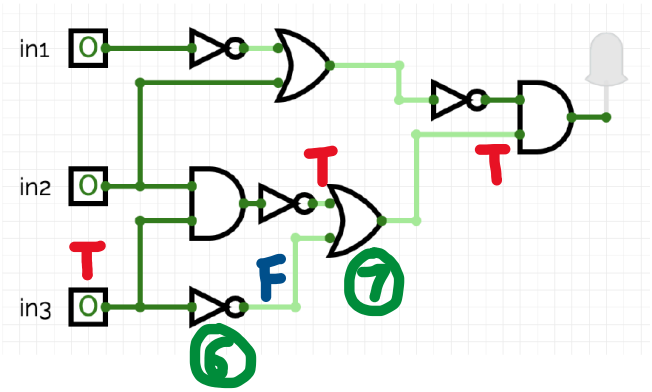
(1) Then not in1 is F, and F or F (for the top OR door) is F, and not F (for the rightmost NOT door) is T.



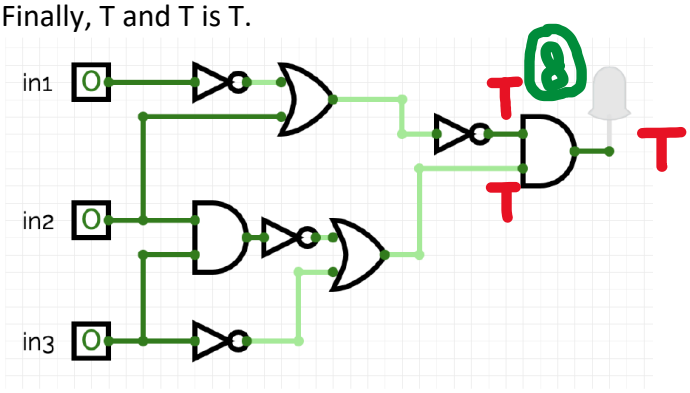
(2) Next, in the left AND door, F and T is F. In the middle row NOT door, not F is T.



(3) Third, in the bottom NOT door, not T is F. In the bottom OR door, T or F is T.



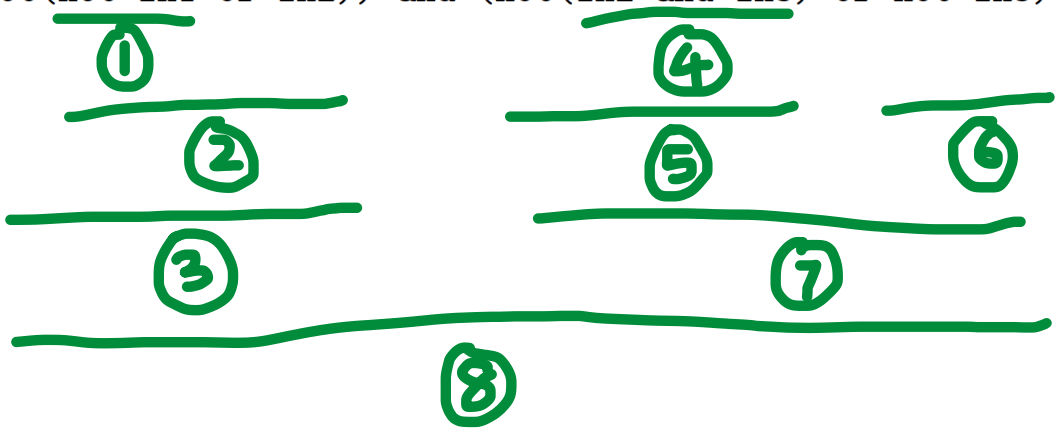
(4) Finally, T and T is T.



Problem 3 (b)

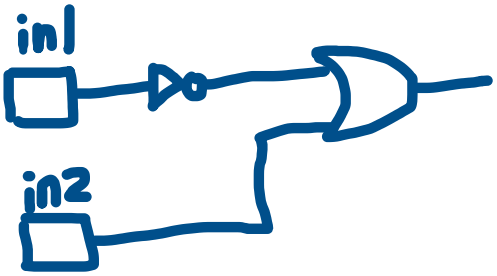
(b) Draw a circuit that implements the logical expression:

```
(not(not in1 or in2)) and (not(in2 and in3) or not in3)
```

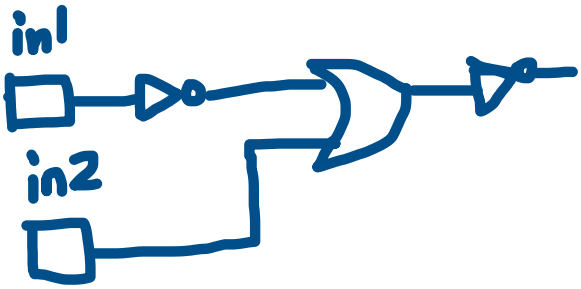


(1) Step 1: not in1

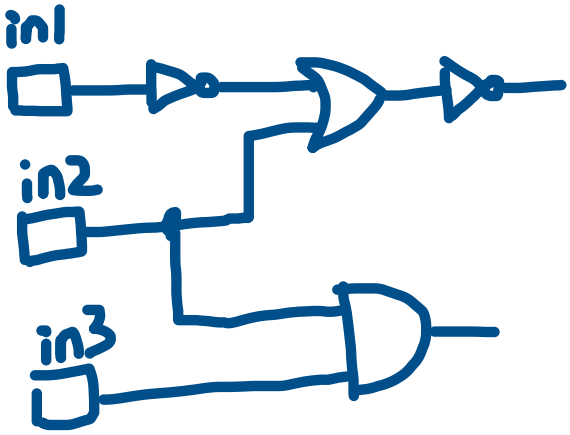
(2) Step 2: not in1 or in2



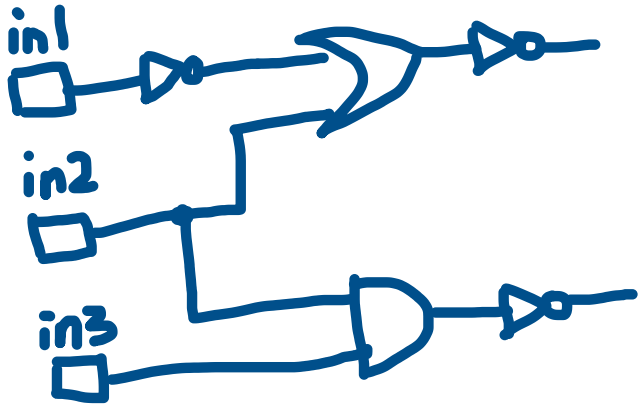
(3) Step 3: (not (not in1 or in2))



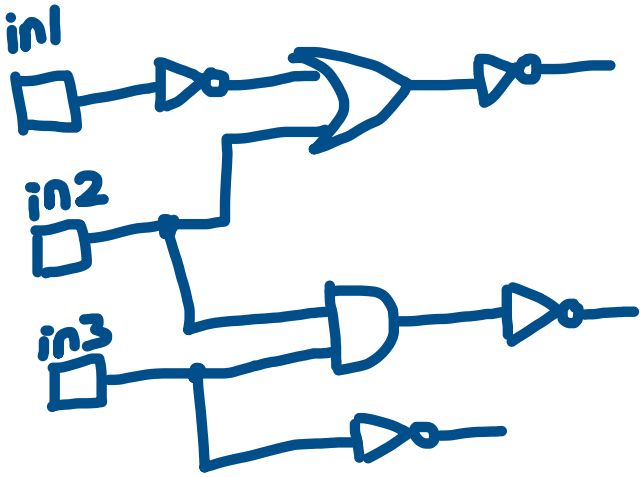
(4) Step 4: in2 and in3



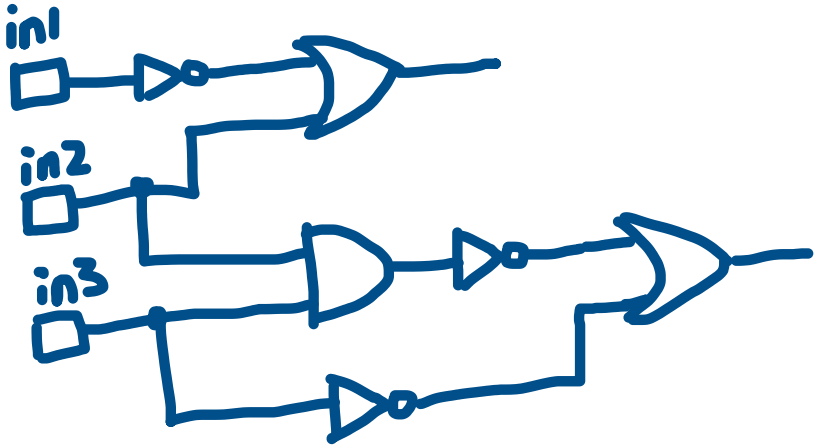
(5) Step 5: not (in2 and in3)



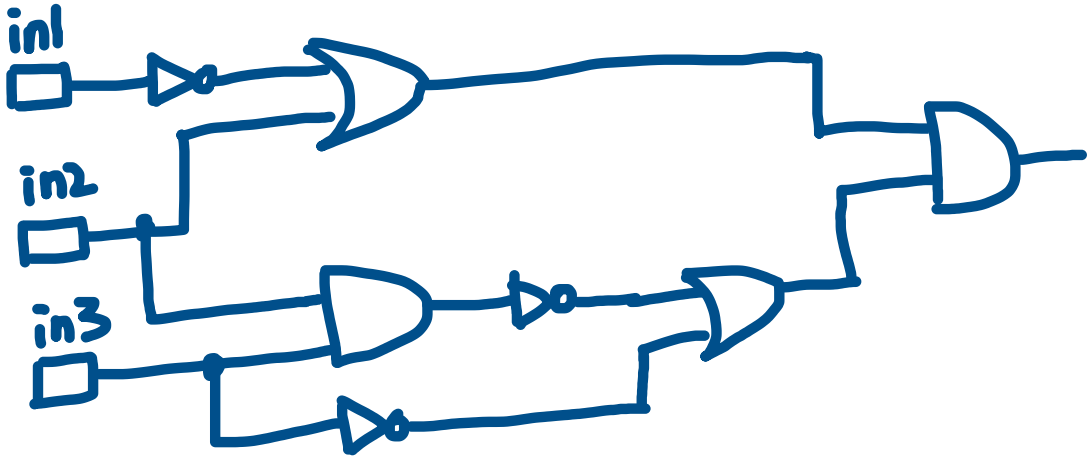
(6) Step 6: not in3



(7) Step 7: not (in2 and in3) or not in3



(8) Step 8: (not(not in1 or in2)) and (not(in2 and in3) or not in3)



Problem 4

4. Consider the following functions:

```
def whoop(n, smile):
   for i in range(1,n+1):
      screech(i, smile)
      print()
```

```
def screech(i, smirk):
    for j in range(i):
      print(smirk, end='  ')


def main():
      whoop(3, '^_^')
```

(a) What are the formal parameters for screech()?

Answer: formal parameters are those parameters in parentheses of function definition. That is, go to the definition of screech, in the function header, find out those parameters, ie, i, smirk.

```
def screech(i, smirk):
```

(b) What are the actual parameters for whoop()?

Answer: actual parameter are the parameters in parentheses when that function is being called. Function whoop is called by function main. Those actual parameters are 3 and '^_^'

```
def main():
      whoop(3, '^_^')
```

(c) How many calls are made to screech() after calling main()?

Answer: when function main is called, it in turn calls whoop(3, '^_^').

```
def main():
      whoop(3, '^_^')
```

```
def whoop(n, smile):
   for i in range(1,n+1):
      screech(i, smile)
      print()
```

(1) First, formal parameters in whoop function are initialized by actual parameters. That is, n is initialized to be 3 and smile is initialized to be '^_^'.
(2) Second, control is switched to function whoop. Move inside function whoop. In loop for i in range(1, n+1):, function range(1, n+1) is range(1, 4) since n is 3, which returns [1, 2,

3] since left end 1 is included but not right end 4, and default increment step is 1. Variable i loops through 1, 2, and 3, function screech inside the loop runs 3 times.

Conclusion: there are three calls to screech function.

(d) What is the output after calling main()?

Answer: In main function, whoop is called. So the problem asks for the output of whoop(3, '^_^'). When we work on for statement, we normally list a table, initialization loop variable (before loop), run statements in loop, one round a row.

```
for i in range(1,n+1): #When n is 3, n+1 is 4, so the loop is for i in range(1, 4): Then i is in [1, 2, 3]
    screech(i, smile)
    print()
```

statements in loop body in the loop of whoop

i	screech(i, '^_^')	print()

1	screech(1, '^_^')
	`def screech(i, smirk):`
	`  for j in range(i):`
	`  print(smirk, end=' ')`
	3 spaces
	(1) Variable i is initialized to be 1, and smirk is initialized to be '^_^'.
	(2) Control switches to function screech.
	(3) In loop for j in range(i), when i is 1, range(i) returns [0], so we only run the statement inside the loop once. That is, print(smirk, end=' '). That is, print '^_^', followed by a space. Or you can think how the loop works.
	(4) Now screech function finishes, return to its caller whoop, and prepare to run the next statement following screech(i, smirk), that is, print() statement.
		Print a new line.
		^_^
		cursor

j	print(smirk, end = ' ')
0	print "^_^ " to the screen. The screen looks like ^_^ (cursor is here, following three spaces)

<table>
<tr>
<td>2</td>
<td>
screech(2, '^_^')
<pre>
def screech(i, smirk):
    for j in range(i):
        print(smirk, end=' ')
</pre>
(1) Variable i is initialized to be 2, and smirk is initialized to be '^_^'.
(2) Control switches to function screech.
(3) In loop for j in range(i), when i is 2, range(i) returns [0, 1], so we run the statement inside the loop twice. That is, print(smirk, end=' ') twice. That is, print '^_^', followed by a space, do it for twice. Here is the screen output.
<table>
<tr><td>j</td><td>Print(smirk, end = ' ')</td></tr>
<tr><td>0</td><td>print "^_^ " to screen. The first line is from previous output. So the screen looks like
^_^
^_^ (the cursor is here, following three spaces), where the first line was drawn before.</td></tr>
<tr><td>1</td><td>print "^_^ " to screen.
^_^
^_^ ^_^ (the cursor is here, following three spaces)</td></tr>
</table>
(4) Now screech function finishes, return to its caller whoop, and prepare to run the next statement following screech(i, smirk), that is, print() statement.
</td>
<td>Print() prints a new line.
The screen looks like
^_^
^_^ ^_^
(cursor is here)</td>
</tr>
</table>

<table>
<tr>
<td>3</td>
<td>
screech(3, '^_^')
<pre>
def screech(i, smirk):
    for j in range(i):
        print(smirk, end=' ')
</pre>
(1) Variable i is initialized to be 3, and smirk is initialized to be '^_^'.
(2) Control switches to function screech.
(3) In loop for j in range(i), when i is 3, range(i) returns [0, 1, 2], so we run the statement inside the loop three times. That is, print(smirk, end=' '). That is, print '^_^', followed by a space, do it for three times. Here is the screen output.
<table>
<tr><td>j</td><td>Print(smirk, end = ' ')</td></tr>
<tr><td>0</td><td>print "^_^ " to screen. The first two lines are from previous output. So the screen looks like
^_^
^_^ ^_^
^_^ (cursor is here, following three spaces)</td></tr>
<tr><td>1</td><td>print "^_^ " to screen.
^_^
^_^ ^_^
^_^ ^_^ (the cursor is here, following three spaces)</td></tr>
<tr><td>2</td><td>Print '^_^' to screen.
^_^
^_^ ^_^
^_^ ^_^ ^_^ (the cursor is here, following three spaces)</td></tr>
</table>
(4) Now screech function finishes, return to its caller whoop, and prepare to run the next statement following screech(i, smirk), that is, print() statement.
</td>
<td>Print a new line character. The screen looks like
^_^
^_^ ^_^
^_^ ^_^ ^_^
(the cursor is here)</td>
</tr>
</table>

Conclusion: the screen looks as follows

^_^
^_^ ^_^
^_^ ^_^ ^_^
(the cursor is here)

Problem 5

Design an algorithm that asks the user for the name of an image file and the quarter ['TL', 'TR', 'BL', 'BR'] they wish to "black-out", where 'TL' stands for Top Left, 'BL' stands for Bottom Right and so on. The algorithm then saves a new image where that quarter of the image is black. The name of the new image is 'XXblack.png' where XX is replaced by one of ['TL', 'TR', 'BL', 'BR'] that the user entered. You must write detailed pseudocode as a precise list of steps that completely and precisely describe the algorithm.

Answer: The original answer is pretty detailed, so I just copy and paste.

(1) Libraries (if any): pyplot and numpy, where pyplot is used for read a png file and put the return to an array object. Then numpy manipulates the array by "blacking out" part of it.

(2) Input: The file name and the quarter

(3) Output: An image where the corresponding quarter is black

Principal Mechanisms (select all that apply):

☐ Single Loop ☐ Nested Loop ☑ Conditional (if/else) statement

☑ Indexing / Slicing ☐ `split()` ☑ `input()`

(a) Ask the user for the name of an image file
(b) Ask the user for the name of a quarter, one of ['TL', 'TR', 'BL', 'BR']
(c) Use pyplot to read the image into a numpy array and give it a name, say img
(d) Use img.shape to find the height and width of the image, with height = img.shape[0] and width = img.shape[1]
(e) Use conditionals (if/elif/else statements) to determine which quarter should be black and use slices to set the color of that quarter to black.

```
if quarter == 'TL',
  img[ : height//2, : width // 2, : ] = 0
elif quarter == 'BL',
    img[ height//2 : , : width // 2, : ] = 0
elif quarter == 'TR',
    img[ : height//2, width // 2 : , : ] = 0
else: #BR
      img[ height//2 : , width // 2 : , : ] = 0
```

(f) use pyplot to save the image to a file with name quarter + "black.png",
plt.iamsave(quarter + "black.png", img)

The above code to set the corresponding region to be black.

You just think
top means 0:height//2 or :height//2
bottom means height//2:
left means 0:width//2 or :width//2
right means width//2:

Top left means img[:height//2, :width//2].
Top right means img[:height//2, width//2:]
Bottom left means img[height//2:, :width//2]
Bottom right means img[height//2:, width//2:]

In short, this is pseudocode, but you need to write key parts in python codes. For example, when it is TL, what operations are done to the image to "black out" that corner.

Problem 6

Consider boeing.csv from the "Military Stocks during Russia-UkraineWar " dataset from kaggle, reporting the Boeing Company's stock prices (in USD $) from January 2010 to May 2022. Each row in the dataset corresponds to the stock values for one day of trading. A snapshot of the data is given in the image below:

Date	Open	High	Low	Close	Volume
2010-01-04	55.720001	56.389999	54.799999	56.180000	6186700
2010-01-05	56.250000	58.279999	56.000000	58.020000	8867800
2010-01-06	58.230000	59.990002	57.880001	59.779999	8836500
2010-01-07	59.509998	62.310001	59.020000	62.200001	14379100
■	■	■	■	■	■
2022-04-28	156.610001	156.789993	149.000000	154.220001	13518800
2022-04-29	153.440002	157.029999	148.520004	148.839996	10880300
2022-05-02	148.020004	149.449997	143.380005	148.610001	12390700

Fill in the Python program below:

```
#Import the libraries for plotting and data frames
import matplotlib.pyplot as plt #plot data
import pandas as pd #process csv file
```

```
#Prompt user for input file name:
fin = input("Enter file name: " )

#Read input data into data frame:
boeing = pd.read_csv(fin)

#Print the average opening value
print(boeing['Open'].mean()) #'Open' is a column from data frame
boeing. Find its average using mean function.

#Print the lowest closing value
print(boeing['Close'].min()) #'Close' is a column from data
frame boeing. Find its min. In python, you can use either
"Close" or 'Close', either way is fine.

#Create a new column called "Range" that computes
#the difference between the highest and lowest value of the
stock
boeing["range"] = boeing["high"] - boeing["low"]

#Plot the newly computed range against the date
boeing.plot(x="Date", y="Range") #created column can be used
after it were the original column in table
plt.show()
```

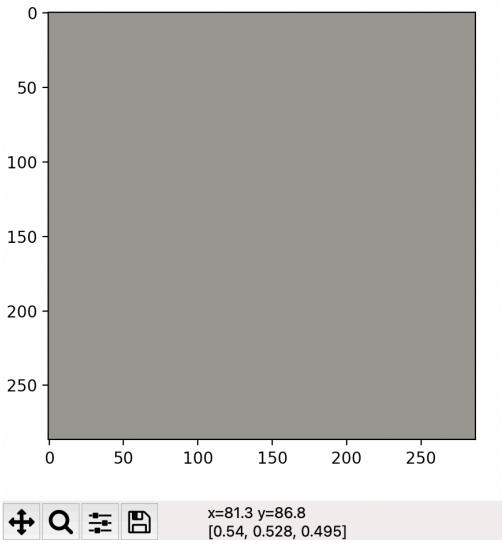
Problem 7

Fill in the following functions that are part of a program that averages the color in an image:
_ getData(): asks the user for the name of an image file and returns a numpy array of the pixels
_ getAvg(): computes and returns the average (r, g, b) values in img
_ avgImg(): returns an image of size rows, cols, with color r, g, b

```
import numpy as np
import matplotlib.pyplot as plt

def getData():
    """
    Asks the user for the name of an image file
    Returns a numpy array of the pixels
    """
    file_name = input("Enter file name: ")
    #plt is used to read a png file and return a numpy array of
the pixels
    img = plt.imread(file_name)
```

```
    return img

def getAvg(img):
    """
    Computes and returns the average (r, g, b) values in img
    """
    r = img[:, :, 0].mean()
    #the first : means the first dimension, which are rows.
    #the second : means the second dimension, which are columns.
    # 0 means red channel in img.

    g = img[:, :, 1].mean() # 1 means green channel in img
    b = img[:, :, 2].mean() # 2 means blue channel in img

    return r, g, b


def avgImg(rows, cols, r, g, b):
    """
    Creates and returns an image of size rows, cols, with color
r, g, b
    """
    avg_img = np.zeros((rows, cols, 3))
    avg_img[:, :, 0] = r
    avg_img[:, :, 1] = g
    avg_img[:, :, 2] = b
    return avg_img

def main():
    img = getData()
    r, g, b = getAvg(img)
    rows = img.shape[0]
    cols = img.shape[0]
    img2 = avgImg(rows, cols, r, g, b)

    plt.imshow(img2)
    plt.show()

if __name__ == '__main__':
   main()
```

Test the above program with csBridge.png, we get


0
50
100
150
200
250
0
50
100
150
200
250
x=81.3 y=86.8
[0.54, 0.528, 0.495]

Problem 8

Review: keys to MIPS program are as follows. Adapted from
http://homepage.divms.uiowa.edu/~ghosh/6016.2.pdf

(1) Use registers $s0, …, $s7, $t0, …, $t7 as local variables, where $t0-$t7 are temporary, not saved after the program, and $s0-$s7 are contents saved for late use. The difference are after the program finishes, the contents of $s0-$s7 are what you change in your program, but the contents of $t0-$t7 might be changed, since operating system or other programs can take these registers as needed.
(2) Use $v0-$v1 to save result. We only use $v0 in our example.
(3) Use $a0-$a3 to save arguments (aka, serve as formal parameter in function calling). We only use $a0 in our example, since print string function takes only one parameter.
(4) Use $sp as stack pointer.
(5) R-type operation (assembly language is case insensitive, so ADD or add does not matter).

```
ADD $s1, $s2, $t0 #add $s2 and $t0, and put the sum to $s1.
ADDI $s2, $s2, -1 #subtract 1 from $s2
```

(6) I-type operation

```
SB $t0, 0($sp)  # save content of $t0
                # to memory address $sp + 0.
```

Save the content of memory address of $sp (base address) + 0 (offset) to register $t0, where S means Save and B means Byte. There are load commands LB (Load Byte) or LW (Load Word) that load the contents from memory address to registers, which are not covered in our example.

(7) J-type operation

```
J SETUP # jump back to SETUP
```

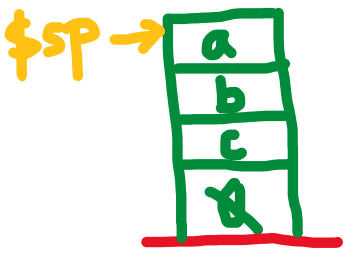
(8) Register `$zero` has value 0.

The program we are doing are just print some strings.

(1) Put contents to memory associated $sp. Note that a string needs to be ended by 0 (null character). For example, if you want to print “abc” without quotes, the data should be stored in memory as follows.

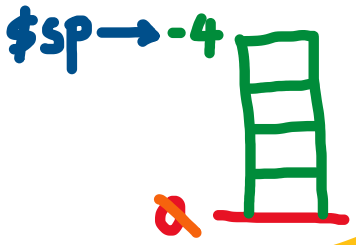
Before

After



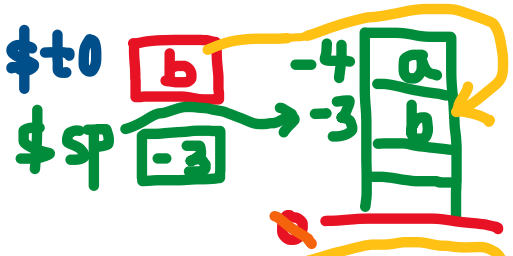
A technique is to subtract 4 (three letters plus null character) from $sp,

addi $sp, $sp, -4

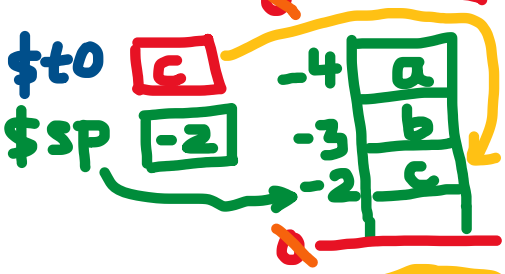


addi $t0, $zero, 97 # $t0 <- ascii of a
sb $t0, 0($sp)

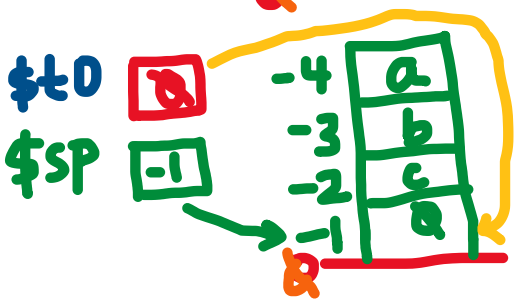
addi $t0, $t0, 1 # $t0 <- ascii of b
addi $sp, $sp, 1
sb $t0, 0($sp)



addi $t0, $t0, 1 # $t0 <- ascii of c
addi $sp, $sp, 1
sb $t0, 0($sp)

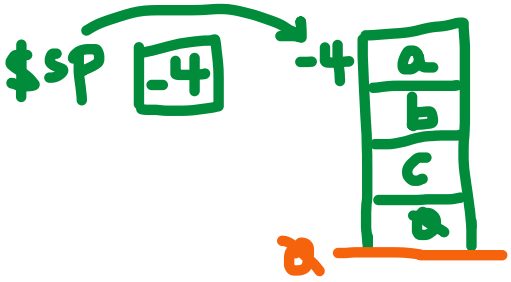


addi $t0, $zero, 0 # $t0 <- null
addi $sp, $sp, 1
sb $t0, 0($sp)



(2) Move $sp to the beginning of data. Note that $sp is -1, the target is -4, both are relative address (you can think the current value of $sp is 1000, move back 4 spaces to 1000 – 4 = 996), so run the following command

```
ADDI $sp, $sp, -3
```



(3) To print a string, we put 4 to $v0, point $a0 to $sp, then call syscall.

```
addi $v0, $zero, 4 # to print string, set $v0 to be 4
addi $a0, $sp, 0 # Set $a0 to stack pointer for printing
syscall
```

Rewrite the above code to use loop
A complete code to print “abc” (without quotes) to the screen is as follows.

```
addi $sp, $sp, -4
addi $t0, $zero, 97 # $t0 <- ascii of a
sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1    # $t0 <- ascii of b
sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1    # $t0 <- ascii of c
sb $t0, 0($sp)
addi $sp, $sp, 1

addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -3

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

Find out the redundant code. Add / modify some comments for consistence.

```
# initialize $sp and $t0
addi $sp, $sp, -4
addi $t0, $zero, 97 # $t0 <- ascii of a

sb $t0, 0($sp)      # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1    # update $sp
addi $t0, $t0, 1    # $t0 <- next letter

sb $t0, 0($sp)      # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1    # update $sp
addi $t0, $t0, 1    # $t0 <- next letter

sb $t0, 0($sp)      # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1    # update $sp
addi $t0, $t0, 1 # Adding this line makes rewriting a loop easy,
                 # but itself does not change running result,
                 # since addi $t0, $zero, 0 follows immediately.
```

```
# save null to indicate end of string
addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -3

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
Syscall
```

Rewrite highlighted part as a loop.

```
sb $t0, 0($sp)     # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1   # update $sp, preparing for next round
addi $t0, $t0, 1   # $t0 <- next letter, preparing for next round

sb $t0, 0($sp)     # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1   # update $sp, preparing for next round
addi $t0, $t0, 1   # $t0 <- next letter, preparing for next round

sb $t0, 0($sp)     # save $t0 to 0($sp), memory pointed by $sp
addi $sp, $sp, 1   # update $sp, preparing for next round
addi $t0, $t0, 1   # $t0 <- next letter, preparing for next round
```

It is equivalent to do the following.

Repeat the following for three times:

```
sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1
```

We do not have a high-level language like repetition statement in MIPS. However, we have j (unconditional jump) and beq (branch if equal) statements. Rewrite as codes in MIPS.

```
Initialize $s2 to be 3
SETUP:
    sb $t0, 0($sp)    # save $t0 to 0($sp), memory pointed by $sp
    addi $sp, $sp, 1  # update $sp, preparing for next round
    addi $t0, $t0, 1  # add 1 to $t0, preparing for next round

    addi $s2, $s2, -1 # deduct 1 from $s2
    beq $s2, $zero, DONE # when $s2 becomes zero,
                         # jump to label DONE:, effectively
                         # move out of loop starting at SETUP:
    j SETUP
DONE:
```

A complete code to print “abc” using loops is as follows.

```
addi $sp, $sp, -4
addi $t0, $zero, 97 # $t0 <- ascii of a
addi $s2, $zero, 3  # store 3 letters to correct position

SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1
addi $s2, $s2, -1
beq $s2, $zero, DONE
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -3

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

More exercises on MIPS

(1) Modify the above code to print alphabet letters.

```
addi $sp, $sp, -27
addi $t0, $zero, 97     # $t0 <- ascii of a
addi $s2, $zero, 26     # store 26 letters to correct position

SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1
addi $s2, $s2, -1
beq $s2, $zero, DONE
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -26

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

(2) Modify the above code to print acegi.

```
addi $sp, $sp, -6   #acegi + null, a total of 5 letters
addi $t0, $zero, 97 # $t0 <- ascii of a
addi $s2, $zero, 5  # store 5 letters to correct position

SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 2    # the gap between adjacent letters is 2
addi $s2, $s2, -1
beq $s2, $zero, DONE
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -5

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

(3) Modify the above code to print ZYXWVU.

```
addi $sp, $sp, -7   # ZYXWVU + null, a total of 7 letters
addi $t0, $zero, 90 # $t0 <- ascii of Z, 90
addi $s2, $zero, 6  # store 6 letters to correct position
SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, -1   # ascii of next letter -
                    # ascii of current letter = -1
addi $s2, $s2, -1
beq $s2, $zero, DONE
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -6

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

(4) Modify the above code to print ZZZZZZZZZ (ten occurrences of letter Z).

```
addi $sp, $sp, -11    # 10 Z's + null, a total of 11 letters
addi $t0, $zero, 90 # $t0 <- ascii of Z, 90
addi $s2, $zero, 10  # store 10 letters to correct position
SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
#addi $t0, $t0, -1    # no update of $t0
addi $s2, $s2, -1
beq $s2, $zero, DONE
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -10

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

(5) Modify the above code to print from b to h, where h is not included.

```
addi $sp, $sp, -7 # bcdefg, 6 letters, plus one null
addi $t0, $zero, 98 # $t0 <- ascii of b, 98
addi $s2, $zero, 104  # $s2 <- ascii of h, 104

SETUP: sb $t0, 0($sp)
addi $sp, $sp, 1
addi $t0, $t0, 1
#addi $s2, $s2, -1      #commented, no update $s2
#beq $s2, $zero, DONE  #commented, not compare $s2 with zero
beq $s2, $t0, DONE # when $t0 reaches $s2 ('h'), leave loop,
                   # so 'h' is not stored in $sp.
j SETUP

DONE: addi $t0, $zero, 0  # $t0 <- null
sb $t0, 0($sp)

ADDI $sp, $sp, -6

ADDI $v0, $zero, 4  # to print string, set $v0 to be 4
ADDI $a0, $sp, 0    # Set $a0 to stack pointer for printing
syscall
```

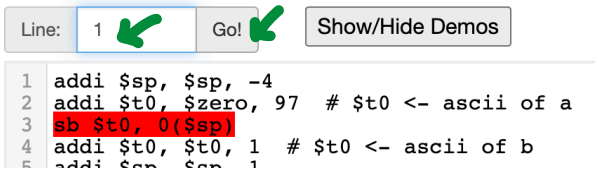
Debug process

Server wemips (google “online wemips”) is tricky. If you get the following errors, here are fixes.

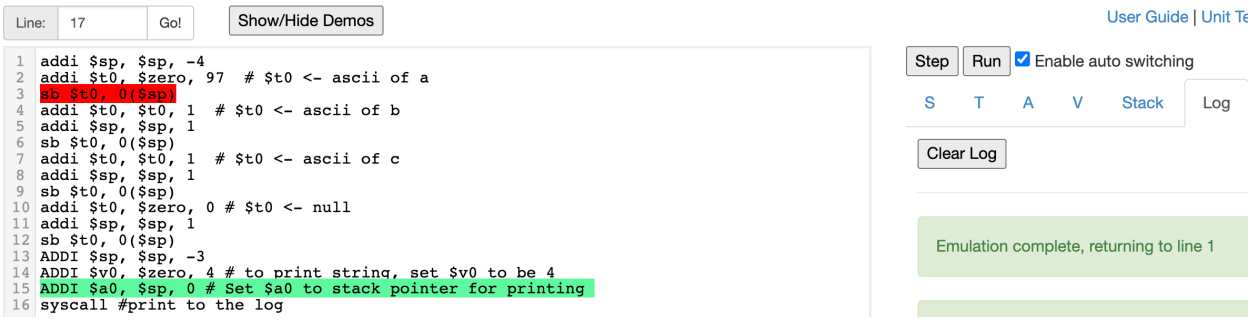
(1) Invalid stack address

Line 3: Invalid stack address (1479859900). Valid stack addresses are between 0 and 1479859899.

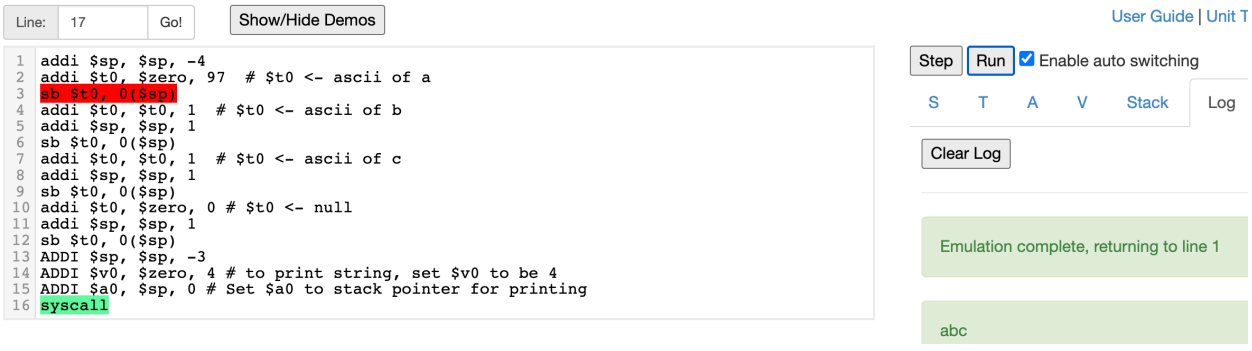
Fix: Put cursor to the MIPS editor, then enter 1 in textbox next to Line, click Go! Then click Run in the right end Step Run .



(2) Code runs ok, but no output. The line before syscall is highlighted green. This means syscall is not properly called.

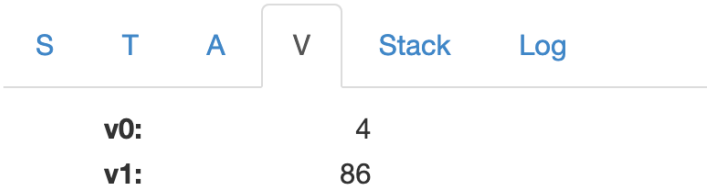


Reason: syscall does not run, that is, the function is not called.
Fix: remove any letters (especially spaces) after syscall, then re-run the code again.



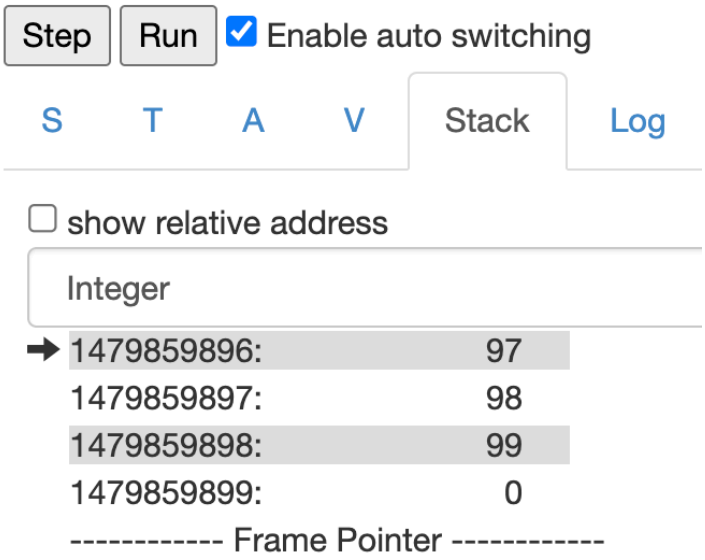
View Register Value

Suppose you want to check the value of $v0, Click V tab.

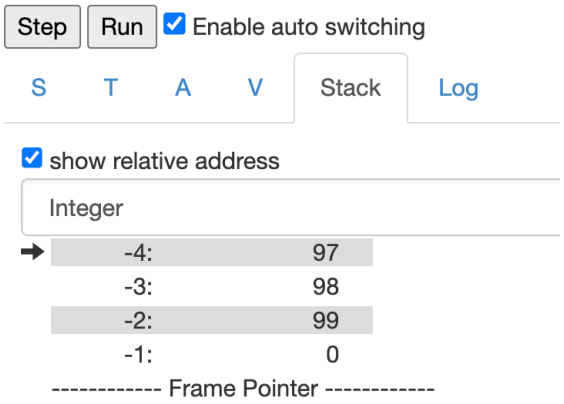


View Stack address

You can click Stack, then see stack address as follows.



Click the textbox next to Use Relative address, you will see



Run code line by line

Click Step  button to see how the code runs step by step.

We start with a simplified version. What does the following problem do?

```
ADDI $sp, $sp, -3 # Set up stack
ADDI $t0, $zero, 90 # 90 is the ASCII code of letter 'Z',
                    #Set $t0 at 90 (Z)
ADDI $s2, $zero, 2 # Use to test when you reach 2
SETUP: SB $t0, 0($sp) # Next letter in $t0
ADDI $sp, $sp, 1 # Increment the stack
ADDI $s2, $s2, -1 # subtract 1 from s2
BEQ $s2, $zero, DONE # Jump to done if $s2 == 0
J SETUP  # If not, jump back to SETUP for loop
DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
SB $t0, 0($sp) # Add null to stack
ADDI $sp, $sp, -2 # Set up stack to print
ADDI $v0, $zero, 4 # to print string, set $v0 to be 4
ADDI $a0, $sp, 0 # Set $a0 to stack pointer for printing
syscall
```

Line: 19 Go! Show/Hide Demos

```
1  #Loop through characters
2  ADDI $sp, $sp, -3 # Set up stack
3  ADDI $t0, $zero, 90 # 90 is the ASCII code of letter 'Z',
4                      #Set $t0 at 90 (Z)
5  ADDI $s2, $zero, 2 # Use to test when you reach 2
6  SETUP: SB $t0, 0($sp) # put $t0 to the top of $sp
7  ADDI $sp, $sp, 1 # Increment the stack
8  ADDI $s2, $s2, -1 # subtract 1 from s2
9  BEQ $s2, $zero, DONE # Jump to done if $s2 == 0
10 J SETUP  # If not, jump back to SETUP for loop
11 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
12 SB $t0, 0($sp) # Add null to stack
13 ADDI $sp, $sp, -2 # Set up stack to print
14 ADDI $v0, $zero, 4 #4 for print string
15 ADDI $a0, $sp, 0 # Set $a0 to stack pointer for printing
16 syscall
17
```

Click Run button, we see output.

Step Run Enable auto switching

S T A V Stack Log

Clear Log

Emulation complete, returning to line 1

ZZ

Problem 9

```
#include <iostream>
using namespace std;
int main()
{
    for(      ; i <=15;      ){
(a)     cout << i-1 << endl;
    }
    return 0;
}
```

Output:

```
3
5
7
9
11
13
```

i-1	i
3	4
5	6
7	8
9	10
11	12
13	14

Observation: when we see problem like this, a sequence of numbers of even gap, we need to use repetition. To work on repetition problems, need to answer following questions.

(1) Where does the variable start?
(2) Where does the variable stop? Or said differently, in what condition should the variable continue to move?
(3) What do you do in each round?
(4) How to update the variables (check difference in adjacent items)

For this problem, the answers are as follows.

(1) The variable i starts at 4.
(2) As long the number <= 15, we stay inside the loop. Said different, the process continue. Leave the loop when the variable > 15.
(3) We print the number in each round.
(4) The loop variable is increased by 2 in each round (or when we move to next round, variable is increased by 2).

List a table as follows.

Variable num is initialized to be 3			
Variable number	Variable number <= 15	Print number	Update loop variable by increasing by 2
4	yes	4	6

6	yes	6	8
8	yes	8	10
10	yes	10	12
12	yes	12	14
14	yes	14	16
16	NO (stop)		

```
#include <iostream>
using namespace std;

int main()
{
    for (int i = 4; i <= 15; i += 2)
        cout << i -1 << endl;

    return 0;
}
```

Problem 9 (b)

(b)

```
#include <iostream>
using namespace std;
int main()
{
     int n=12, m=-5;

     while(n+m      ){
          cout << n << " " << m << endl;
          n-=2;
          m++;
     }
     return 0;
}
```

Output:

```
12 -5
10 -4
8 -3
6 -2
4 -1
2 0
0 1
```

From the problem, we can see that the loop variable is n + m, so we need to find out the changes of n + m.

n	m	n+m	n -= 2	m++	Each time n is decreased by 2 and m is increased by 1, then n + m is decreased by 1.
12	-5	7	10	-4	
10	-4	6	8	-3	
8	-3	5	6	-2	
6	-2	4	4	-1	
4	-1	3	2	0	
2	0	2	0	1	
0	1	1	-2	2	

So n + m starts from 7, as long as n + m >= 1 or equivalently, n + m > 0 since both n and m are integers, we stay in the loop, print n and m respectively. Then decrease n by 2 and increase m by 1 to prepare for the next round. An answer is as follows.

```
#include <iostream>
using namespace std;

int main()
{
     int n = 12, m = -5;

     while (n + m > 0) //or n + m >= 1
     {
          cout << n << " " << m << endl;
          n -= 2;
          m++;
     }

     return 0;
}
```

Problem 9 (c)

```
#include <iostream>
using namespace std;
int main(){
    for ( [          ] ){
(c)     for( [          ] ){
            cout << i << j-i << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:

```
88 87 86 85 84 83 82 81 80
77 76 75 74 73 72 71 70
66 65 64 63 62 61 60
55 54 53 52 51 50
44 43 42 41 40
33 32 31 30
```

By the output, we find out the value of i and j-i (so derive j's value as well).

output	i	j-i	j = j – i + i
88 87 86 85 84 83 82 81 80	8	8 7 6 5 4 3 2 1 0	16 15 14 13 12 11 10 9 8

i j-i	i = 8
	for (j = 16; j >= 8; j--) //adjacent items of j is decreased by 1
	cout << i << j – i << " ";
	cout << endl;
	
	If represent 16 and 8 with i, since these values are related with i, we have
	
	i = 8
	for (j = 2 * i; j >= i; j--)
	cout << i << j – i << " ";
	cout << end;		

77 76 75 74 73 72 71 70	7	7 6 5 4 3 2 1 0	14 13 12 11 10 9 8 7

	i = 7
	for (j = 14; j >= 7; j--)
	cout << i << j – i << " ";
	cout << endl;
	
	Represent 14 and 7 by i, we have
	i = 7
	for (j = 2 * i; j >= i; j--)
	cout << i << j – i << " ";
	cout << endl;		

66 65 64 63 62 61 60	6	6 5 4 3 2 1 0	12 11 10 9 8 7 6
	i = 6		

<table>
<tr><td></td><td colspan="3">for (j = 12; j >= 6; j--)
 cout << i << j – i << “ “;
cout << endl;

Represent 12 and 6 by i, we have
i = 6
for (j = 2 * i; j >= i; j--)
 cout << i << j – i << “ “;
cout << endl;</td></tr>
<tr><td>55 54 53 52 51 50</td><td>5</td><td>5 4 3 2 1 0</td><td>10 9 8 7 6 5</td></tr>
<tr><td></td><td colspan="3">i = 5
for (j = 10; j >= 5; j--)
 cout << i << j – i << “ “;
cout << endl;

Represent 10 and 5 by i, we have
i = 5
for (j = 2 * i; j >= i; j--)
 cout << i << j – i << “ “;
cout << endl;</td></tr>
<tr><td>44 43 42 41 40</td><td>4</td><td>4 3 2 1 0</td><td>8 7 6 5 4</td></tr>
<tr><td></td><td colspan="3">i = 4
for (j = 8; j >= 4; j--)
 cout << i << j – i << “ “;
cout << endl;

Represent 8 and 4 by i, we have
i = 4
for (j = 2 * i; j >= i; j--)
 cout << i << j – i << “ “;
cout << endl;</td></tr>
<tr><td>33 32 31 30</td><td>3</td><td>3 2 1 0</td><td>6 5 4 3</td></tr>
<tr><td></td><td colspan="3">i = 3
for (j = 6; j >= 3; j--)
 cout << i << j – i << “ “;
cout << endl;

Represent 6 and 3 by i, we have
i = 3
for (j = 2 * i; j >= i; j--)
 cout << i << j – i << “ “;
cout << endl;</td></tr>
</table>

Extract the code.

```
i = 8
for (j = 2 * i; j >= i; j--)
      cout << i << j – i << “ “;
cout << end;

i = 7
for (j = 2 * i; j >= i; j--)
     cout << i << j – i << “ “;
cout << endl;

i = 6
for (j = 2 * i; j >= i; j--)
     cout << i << j – i << “ “;
cout << endl;

i = 5
for (j = 2 * i; j >= i; j--)
     cout << i << j – i << “ “;
cout << endl;

i = 4
for (j = 2 * i; j >= i; j--)
     cout << i << j – i << “ “;
cout << endl;

i = 3
for (j = 2 * i; j >= i; j--)
     cout << i << j – i << “ “;
cout << endl;
```

Rewrite the code as a nested loop

```
int i, j;
for (i = 8; i >= 3; i--)
{ //this pair of curly braces cannot be omitted, since loop body it enclosed has >= 2 statements
  //the first is a for statement, the second is a statement to print a new line.
     for (j = 2 * i; j >= i; j--)
     { //this pair of curly braces can be omitted, since loop body it encloses has only one
statement
          cout << i << j – i << “ “;
     }
     cout << endl;
}
```

A complete code is as follows. We declare i and j inside for-head.

```
#include <iostream>
using namespace std;

int main()
{
    for (int i = 8; i >= 3; i--) {
        for (int j = 2*i; j >= i; j--)
            cout << i << j - i << " ";

        cout << endl;
    }

    return 0;
}
```

Problem 10

Write a complete C++ program that repeatedly asks the user for two integers until their sum is even, then outputs the sum.

pseudocode:

Declare integer variables i and j

enter i and j
as long as i + j is not even, that is, (i + j) % 2 != 0
begin
    prompt that the sum is not even
    ask user to re-input i and j
end

output the sum of i and j

A complete C++ code is as follows.

```
#include <iostream>
using namespace std;

int main()
{
    int i, j;
    cout << "Enter i and j such that i + j is even: ";
    cin >> i >> j;
    while ( (i + j) % 2 != 0 )
    {
        cout << "The sum needs to be even. Re-enter: ";
        cin >> i >> j; //let user to make updates on i and j
    }

    cout << "the sum is: " << i + j << endl;
    return 0;
}
```

Problem 10 (b)

Write a complete C++ program that asks the user for an amount and computes the number of years it takes to triple the amount, if it is subject to an increase of 5% each year.

Key ideas:
Suppose amount is 100 for illustration purpose. You can use other value like 200, 1000, or whatever, it should be double type since the number can contain decimal numbers.

year	balance in end of that year
0	100
1	100 * 1.05, increase 0.05 in one year, starting balance is 100
2	100 * 1.05 * 1.05, increase 0.05 in one year, starting balance is 100 * 1.05
3	100 * 1.05 * 1.05 * 1.05, increase 0.05 in one year, starting balance is the end balance of previous year, that is, 100 * 1.05 * 1.05
4	100 * 1.05 * 1.05 * 1.05 * 1.05
…	
Continue until the balance is less than 300. Or, stop once balance is >= 300.	

Pseudocode:
int year = 0
Enter initial balance and put in variable amount.
double target = 3 * amount
while (amount < target)
begin

    increase year by 1
    amount is increased by 0.05, that is, amount *= 1.05;
    report year and balance
end
print amount and years to get that amount.

A complete C++ code is as follows.

```
#include <iostream>
using namespace std;

int main()
{
    int year = 0;
    double amount;
    cout << "Enter initial amount: ";
    cin >> amount;

    double target = 3 * amount;
    while (amount < target)
    {
        year++;
        amount *= 1.05;
        cout << year << "\t" << amount << endl;
    }

    cout << "It takes " << year << " to get "
         << amount << endl;

    return 0;
}
```