

Clicker Questions

CS202

Clicker Questions for Today

Questions 1 through 5

Behold the following program:

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

int main()
{
    int a, b;
    char c, d;

    cin >> a >> b;
    c = 'a' + b;
    d = 'a' + (a-b);

    cout << a % b << endl;
    cout << a / b << endl;
    cout << a << b << endl;
    cout << c << endl;
    cout << d << endl;
    return 0;
}
```

I compile this program to **a.out** and then I run it with:

```
UNIX> echo 9 4 | ./a.out
```

Question 1: What is the first line of output?

Question 2: What is the second line of output?

Question 3: What is the third line of output?

Question 4: What is the fourth line of output?

Question 5: What is the fifth line of output?

Clicker question answers:

Just copy it, compile and run. If you don't understand an answer, tweak the input and try to understand why the output is what it is.

```
UNIX> echo 9 4 | a.out
```

```
1
```

```
2
```

```
94
```

```
e
```

```
f
```

```
UNIX>
```

Clicker Questions

- **Question 1:** What is the output of **p1.cpp** when it runs with **input-1.txt** as standard input?
- **Question 2:** What is the output of **p2.cpp** when it runs with **input-1.txt** as standard input?
- **Question 3:** What is the output of **p3.cpp** when it runs with **input-2.txt** as standard input?

input-1.txt

```
aa bb cc dd
ee ff gg hh
```

input-2.txt

```
4 5
3 9
```

p1.cpp

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

int main()
{
    string s1, s2, s3;

    while (cin >> s1 >> s2 >> s3) {
        cout << s1[0] << s2[0] << s3[0];
    }
    cout << endl;
    return 0;
}
```

p2.cpp

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

int main()
{
    vector <string> sv(3);

    while (cin >> sv[0] >> sv[1] >> sv[2]) {
        cout << sv[0] << sv[1] << sv[2];
    }
    cout << endl;
    return 0;
}
```

p3.cpp

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

int main()
{
    int sum, i;

    sum = 0;
    while (!cin.eof()) {
        cin >> i;
        sum += i;
    }
    cout << sum << endl;
    return 0;
}
```

Answers

Question 1: The first iteration reads "aa", "bb" and "cc". The second iteration reads "dd", "ee", and "ff". The third iteration fails, because there are only two words left. After the words are read, the first character of each word is printed:

abcdef

Question 2: The same strings are read as in Question 1, only this time into a vector rather than three separate variables. All of the characters are printed:

aabbccddeeff

Question 3: This is a buggy program, because `cin.eof()` only returns `true` after it has failed to read the input. So this loop runs five times, and on the last iteration, the `(cin >> i)` statement fails, leaving `i` equaling nine. Thus, 9 gets added to the sum twice. The answer is $4 + 5 + 3 + 9 + 9 = 30$

30

Clicker Questions

- **Question 1:** What is the output of **p1.cpp** when it runs with **input-1.txt** as standard input?
- **Question 2:** What is the output of **p2.cpp** when it runs with **input-1.txt** as standard input?
- **Question 3:** What is the output of **p3.cpp** when it runs with **input-2.txt** as standard input?

(BTW, I'm omitting the "include" and "using" lines to conserve space. Just pretend that the correct ones are there.)

input-1.txt

```
Dear Earl,  
Great Job!  
Love Madison
```

input-2.txt

```
4  
5  
6  
seven  
3
```

p1.cpp

```
int main()  
{  
    string s, r;  
  
    while (cin >> s) {  
        r += s;  
    }  
    cout << r.size() << endl;  
    return 0;  
}
```

p2.cpp

```
int main()  
{  
    string s;  
    vector <string> v;  
    size_t i;  
    int total;  
  
    while (cin >> s) v.push_back(s);  
    total = 0;  
    for (i = 1; i < v.size(); i++) {  
        total += (v[i][0] - v[i-1][0]);  
    }  
    cout << total << endl;  
    return 0;  
}
```

p3.cpp

```
int main()  
{  
    int i;  
    int total;  
  
    total = 0;  
    while (!cin.fail()) {  
        cin >> i;  
        total++;  
    }  
    cout << total << endl;  
    return 0;  
}
```

Clicker Answers

Question 1:

The program reads six strings:

1. "Dear" -- 4 characters.
2. "Earl," -- 5 characters (the comma counts).
3. "Great" -- 5 characters
4. "Job!" -- 4 characters (the exclamation point counts).
5. "Love" -- 4 characters
6. "Madison" -- 7 characters

Thus, **r** will be the sum of the characters. The answer is 29.

Question 2:

This reads in the same six strings. It then sums up the difference between each adjacent pair of first characters:

1. ('E' - 'D') = 1.
2. ('G' - 'E') = 2.
3. ('J' - 'G') = 3.
4. ('L' - 'J') = 2.
5. ('M' - 'L') = 1.

The sum is 9.

Question 3:

Does this question seem familiar? It should -- it's pretty much the same as the **eof()** question from the last set. **cin.fail()** doesn't return **true** until you try to read "seven". Thus, that **while** loop is executed four times -- the answer is 4.

Clicker Questions

- **Question 1:** What is the output of the program on the right when it runs with **input-1.txt** as standard input?
- **Question 2:** What is the output of the program on the right when it runs with **input-2.txt** as standard input?
- **Question 3:** What is the output of the program on the right when it runs with **input-3.txt** as standard input?
- **Question 4:** What is the output of the program on the right when it runs with **input-4.txt** as standard input?

(BTW, I'm omitting the "include" and "using" lines to conserve space. Just pretend that the correct ones are there.)

(Also, don't print the final "newline" -- just answer with the string that is printed.)

```
int main()
{
    int i, n;
    string s;
    string rv;

    rv = "";
    while (cin >> n) {
        for (i = 0; i < n; i++) {
            if (cin >> s) rv.push_back(s[0]);
        }
    }
    cout << rv << endl;
    return 0;
}
```

<u>input-1.txt</u>	<u>input-2.txt</u>	<u>input-3.txt</u>	<u>input-4.txt</u>
1 Fred	2 Marco Polo 3 Drop Dead Fred	3 A BB CCC 2 Xerxes Yassin Zelda	3 7 5 3 4 7 6 8 3 9 8 3

Clicker Answers

Question 1:

- The program reads an integer **n**, and then iterates **n** times.
- For each iteration, it reads a string, and if that was successful, appends the first character of the string to the string **rv**.
- The program repeats this process until reading **n** fails.
- Then it prints out **rv**.

So, with input 1, it reads the integer 1, then reads the string "Fred", and appends the 'F' to **rv**. The answer is

F

Question 2: Now, the main **while()** loop iterates twice:

- It reads the number 2 and then "Marco" and "Polo". At this point, **rv** is "MP".
- It reads the number 3 and then "Drop", "Dead" and "Fred". At this point, **rv** is "MPDDF".

The answer is:

MPDDF

Question 3: The main **while()** loop iterates twice:

- It reads the number 3 and then "A", "BB" and "CCC". At this point, **rv** is "ABC".
- It reads the number 2 and then "Xerxes" and "Yassin". At this point, **rv** is "ABCXY".
- It tries to read a number, but it fails because the next word is "Zelda". So it falls out of the **while()** loop.

The answer is:

ABCXY

Question 3: Now the main loop iterates three times:

- It reads the number 3 and then "7", "5" and "3". At this point, **rv** is "753".
- It reads the number 4 and then "7", "6", "8" and "3". At this point, **rv** is "7537683".
- It reads the number 9 and then "8" and "3". At this point, **rv** is "753768383".
- It tries to continue reading strings, but it reaches EOF, so the **cin** statement fails.
- It now tries to read an integer, but the **cin** is still in a failed state, so it out of the **while()** loop.

The answer is:

753768383

Clicker Questions

Question 1: When I put the keyword **const** in front of a reference parameter, it means that:

- It will seg-fault if you try to change it.
 - The procedure or method will not modify it.
 - It is a constant, like Pi.
 - The parameter is a fixed size data type, like **int** or **char**.
 - The compiler will emit lots of errors and warnings.
-

Question 2: When I put the keyword **public** in front of a class' variable, it means that:

- A procedure that has an instance of the class may access or modify the variable.
 - It is **const** by default.
 - It is a global variable.
 - The variable can only be modified in one of the class' methods.
 - The compiler will emit lots of errors and warnings.
-

Questions 3-8: In all of the following, the answers are either "Like", "Dislike" or "Sometimes Like / Sometimes Dislike".

- **Question 3:** What does Dr. Plank think of declaring variables in the middle of procedures.
- **Question 4:** What does Dr. Plank think of tertiary expressions?
- **Question 5:** What does Dr. Plank think of Python?
- **Question 6:** What does Dr. Plank think of putting executable code in header files?
- **Question 7:** What does Dr. Plank think of reference parameters?
- **Question 8:** What does Dr. Plank think of The **const** keyword?

Answers Clicker Questions

Question 1: When I put the keyword **const** in front of a reference parameter, it means that the procedure or method will not modify it. If it does, then the compiler will flag an error. The keyword is nice, because whoever calls such a procedure or method knows that its parameter won't change. Please see <https://web.eecs.utk.edu/~jplank/plank/classes/cs202/Notes/Procedures/index.html> for more discussion.

Question 2: When I put the keyword **public** in front of a class' variable, it means that a procedure that has an instance of the class may access or modify the variable. If it's **private** or **protected**, then it may only be accessed or modified within implementations of the class (there's a little more to it than this because of inheritance, but just go with that for now).

Question 3: What does Dr. Plank think of declaring variables in the middle of procedures? Dislike. I like my variable declarations at the top of a procedure or method. That way, you know where to find them.

Question 4: What does Dr. Plank think of tertiary expressions? Sometimes like, sometimes dislike. I like them when they are simple and readable. When they are convoluted, or embedded in convoluted expressions, I dislike them.

Question 5: What does Dr. Plank think of Python? Dislike. Maybe that will change someday, but I find the language to be too much of a free-for-all. I like structure and explicit types. This doesn't mean that you have to dislike Python -- I think it is an important language that has expanded the reach of machine learning, and that if you learn it, it will only help your careers. I simply don't like it.

Question 6: What does Dr. Plank think of putting executable code in header files? Dislike. In my opinion, header files should have procedure prototypes and class definitions, and no running code. Headers should be static, and implementations should be dynamic. It's a matter of readability, and being able to trace through what your code is doing.

Question 7: What does Dr. Plank think of reference parameters? Like. They avoid copying and read better than pointers, and sometimes you want your procedure to be able to modify its arguments.

Question 8: What does Dr. Plank think of The **const** keyword? Like. It's really nice to know when things are being modified and when they aren't, and const lets you know.

Clicker Questions

In case you've forgotten:

AND: this is a single ampersand:	&
OR: this is a single vertical bar:	
XOR: this is a single carat:	^
NOT: this is a single tilde:	~
Left-shift: this is two less-than signs:	<<
Right-shift: this is two greater-than signs:	>>

For i between 1 and 6, Question i is to tell me what gets printed on line i of the program to the right.

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

int main()
{
    int a, b, c, d, e;

    a = 32;
    b = 10;
    c = 0x3b53a;
    d = 0x3b93a;
    e = 0x8372a;

    printf("%d\n", 5 & 9);           // Question 1
    printf("%d\n", 3 | 12);         // Question 2
    printf("0x%x\n", a + b);        // Question 3
    printf("0x%x\n", c ^ d);        // Question 4
    printf("0x%x\n", e << 8);       // Question 5
    printf("0x%x\n", e >> 4);       // Question 6
    return 0;
}
```

Clicker Question Answers

Here's the short version

```
UNIX> g++ q1.cpp
UNIX> a.out
1
15
0x2a
0xc00
0x8372a00
0x8372
UNIX>
```

Here's the longer version

Question 1:

```
0101 -- 5 in binary
1001 -- 9 in binary
----
0001 -- so the answer is 1
```

Question 2:

```
0011 -- 3 in binary
1101 -- 12 in binary
----
1111 -- so the answer is 15
```

Question 3: 32 is 2 times 16, so the hex digit in the 16's place is 2. 10 is 0xa in hex. So the answer is 0x2a.

Question 4: In hex, each digit is four bits, so you can simply do the xor in hex:

```
3 b 5 3 a
3 b 9 3 a
-----
0 0 c 0 0 -- the answer is 0xc00
```

Question 5: Remember that left and right shifting by multiples of 4 allow you to simply work on the hex digits. So you simply append two zeros to the hex: 0x8372a00.

Question 6: And here you simply chop off one digit: 0x8372.

For these questions, each answer is four numbers. Simply enter them on Turning point separated by spaces or commas.

I'll also put the mathematical definition of open addressing here. From the lecture notes:

With open addressing, you test the keys in order, h_0, h_1, h_2, \dots , until you find what you're looking for or you find an empty space in the hash table. h_i is defined as follows:

$$h_i = H(\text{key}) + F(i, \text{key}), \text{ modulo the table size.}$$

$H()$ is the hash function, and $F()$ is a function that defines how to resolve collisions. It is usually convenient to make sure that $F(0, \text{key})$ equals zero, and it is necessary for $F(i, \text{key})$ not to equal zero when i doesn't equal zero.

- With linear probing, $F(i, \text{key}) = i$.
- With quadratic probing, $F(i, \text{key}) = i^2$.
- With double hashing, $F(i, \text{key}) = H_2(\text{key}) * i$.

Question 1: Suppose you are looking up a value in a hash table that has 100 elements, and the hash table is pretty full. The hash function for the value returns 847298. You are using linear probing – what are the first four indices that you will test when you look up the value?

Question 2: Now, suppose you are using quadratic probing. What are the first four indices that you will test?

Question 3: Now, suppose you are using double-hashing, and the second hash function returns 92821. What are the first four indices that you will test?

Answers

This one came from the spring 2014 midterm. Here are the answers:

Question 1: You start at 98, and look at successively higher indices mod 100: 98, 99, 0, 1.

Question 2: Now, you start at 98, and add 0^2 , 1^2 , 2^2 and 3^2 : 98, 99, 2, 7.

Question 3: Now, you start at 98, and add $0*21$, $1*21$, $2*21$ and $3*21$: 98, 19, 40, 61.

Grading

1 point per answer, with the following caveats:

- **Question 1:** To receive full credit on the 2nd, 3rd and 4th probes, they had to equal the previous probe plus one, mod 100.
- **Question 1:** If you said 1 instead of 0, you got half credit.
- **Question 1:** If you forgot the mod 100 when you added one, you got half credit.
- **Question 2:** To get full credit, your answer for the first probe had to equal the answer for the first probe in Question 1.
- **Question 2:** The other probes needed to be relative to the first probe.
- **Question 2:** If you forgot the mod 100, you got half credit.
- **Question 2:** On the third and fourth probes, if you were off by one, you got half credit.
- **Question 3:** To get full credit, your answer for the first probe had to equal the answer for the first probe in Question 1.
- **Question 3:** The other probes needed to be relative to the first probe.
- **Question 3:** If you forgot the mod 100, you got half credit.
- **Question 3:** On the second, third and fourth probes, if you were off by one, you got half credit.

Name	H1	H2
-----	-----	-----
Aidan Nepal	9bc6cc	42757b
Alexander Aeronautic	4345ae	93249d
Austin Prissy	20b849	8c1ef7
Charlie Maiden	0d6fc1	e5855d
Claire Tva	fe30bf	8124af
Gavin Parallelepiped	0e9b65	db8914
Hunter Fabric	b112a2	78d98d
Isabella Stack	4c30e6	f7193f
Joshua Polonium	831e15	eeba38
Madison Willoughby	1874dc	911d61
Max Mere	fc7e5b	8d1814
Maya Paddy	107b1e	5e84ff
Natalie Sober	c624de	b46fa3
Noah Porous	ae3e75	2dbc99
Riley Predecessor	f001c1	385e34
Savannah Tradesman	a3d7ee	ab7176
Sophia Keys	69147a	493120

On the left, I have a table of 17 names, and what each name hashes to with two different hash functions, H1 and H2. The hash values are given in hexadecimal.

On the right, I have a 16-element hash table that has been filled in with some of these names.

Please answer the questions below. Do not answer the questions as if one affects the other. Answer them all with respect to the tables shown here. For example, you should not answer part C as if Austin Prissy were inserted into the table. Instead, you simply answer with respect to the table above.

In all of the questions, assume that hash function H1 is used to insert into the table, and if double-hashing is used, then hash function H2 is used as the second hash function.

0.	-----
1.	Charlie Maiden
2.	Hunter Fabric
3.	-----
4.	-----
5.	Gavin Parallelepiped
6.	Isabella Stack
7.	-----
8.	-----
9.	-----
10.	Sophia Keys
11.	Max Mere
12.	Aiden Nepal
13.	Madison Willoughby
14.	Savannah Tradesman
15.	Claire Tva

1. What is the load factor of the table (you can give a fraction here)?
2. Into which index will Austin Prissy go into the table, using linear probing?
3. Into which index will Joshua Polonium go into the table, using quadratic probing?
4. Into which index will Maya Paddy go into the table, using double hashing?
5. Into which index will Noah Porous go into the table, using linear probing?
6. Into which index will Riley Predecessor go into the table, using double hashing?

Answers

This came from the 2019 midterm for COSC140.

Since the table size is 16, the last hex digit is the hash index.

- Question 1: There are 10/16 entries full, so the load factor is 10/16.
- Question 2: Austin Prissy's index is 9, which is empty, so the answer is 9.
- Question 3: Joshua Polonium's index is 5. Five is taken, so we check $5+1^2 = 6$. That is taken, so we check $5+2^2 = 9$. That is empty, so the answer is 9.
- Question 4: Maya Paddy's index is $0xd = 13$. 13 is taken, so we'll need to use the second hash value, which is $0xf = 15$. This means that we'll be looking at successively smaller indices -- 12, 11, 10, and finally 9.
- Question 5: Noah Porous' index is 5. Five is taken, so we'll look at successively larger indices -- 6, which is taken, and then 7, which is empty. The answer is 7.
- Question 6: Riley Predecessor's index is 1. 1 is taken, so we'll need to use the second hash value, which is 4. 5 is taken, but 9 is empty, so the answer is nine.

Please read over the code to the right, and assume that this is compiled to **a.out**.

Question 1: What is the output of:

```
UNIX> echo A | ./a.out
```

Question 2: What is the output of:

```
UNIX> echo B | ./a.out
```

Question 3: What is the output of:

```
UNIX> echo C | ./a.out
```

Question 4: What is the output of:

```
UNIX> echo D | ./a.out
```

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

string a(const string &s)
{
    if (s == "A") throw (string) "B";
    try {
        if (s == "C") throw (string) "D";
        return "E";
    } catch (const string &t) {
        cout << t;
    }
    return "F";
}

int main()
{
    string t;

    cin >> t;
    try {
        cout << a(t);
    } catch (const string &s) {
        cout << s;
    }
    cout << endl;
    return 0;
}
```

Answers

```
UNIX> g++ code.cpp
UNIX> echo A | ./a.out      # Question 1
B
UNIX> echo B | ./a.out      # Question 2
E
UNIX> echo C | ./a.out      # Question 3
DF
UNIX> echo D | ./a.out      # Question 4
E
UNIX>
```

Explanations:

- When "A" is entered and passed to the procedure `a()`, the procedure throws the string "B". That **throw** statement is not inside a **try**, so the exception is passed to the calling procedure `main()`. `main()` catches the exception and prints "B". It then prints a newline and exits.
- When "B" is entered and passed to the procedure `a()`, both **if** statements are false, so no exceptions are thrown. `a()` returns "E" to the caller, which prints it out, and then prints a newline.
- When "C" is entered and passed to the procedure `a()`, the procedure throws the string "D". That is caught in `a()`, which prints "D". After the **catch** code, it returns "F". `main()` prints "R" and a newline.
- Any string that is not "A" or "C" will be identical to question 2.

Behold the program to the right.

- **Question 1:** What is the first line of this program's output?
- **Question 2:** What is the second line of this program's output?
- **Question 3:** What is the third line of this program's output?
- **Question 4:** What is the fourth line of this program's output?
- **Question 5:** What is the fifth line of this program's output?
- **Question 6:** What is the sixth line of this program's output?
- **Question 7:** What is the seventh line of this program's output?
- **Question 8:** On which line does a memory leak occur?

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

int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */

    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */

    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */

    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */

    jv.push_back(*(iv[1])); /* Line 18 */
    jv.push_back(*(iv[0])); /* Line 19 */
    /* Line 20 */

    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
    /* Line 23 */

    cout << *ip << endl; /* Line 24 */
    cout << *jp << endl; /* Line 25 */
    cout << *kp << endl; /* Line 26 */
    cout << *(iv[0]) << endl; /* Line 27 */
    cout << *(iv[1]) << endl; /* Line 28 */
    cout << jv[0] << endl; /* Line 29 */
    cout << jv[1] << endl; /* Line 30 */
    return 0; /* Line 31 */
} /* Line 32 */
```

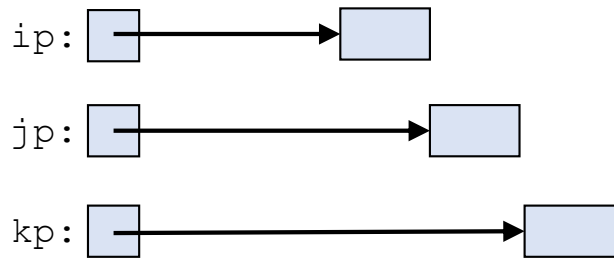
ip:

jp:

kp:

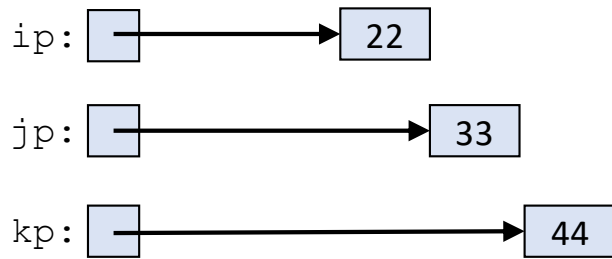
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Lines 7-9



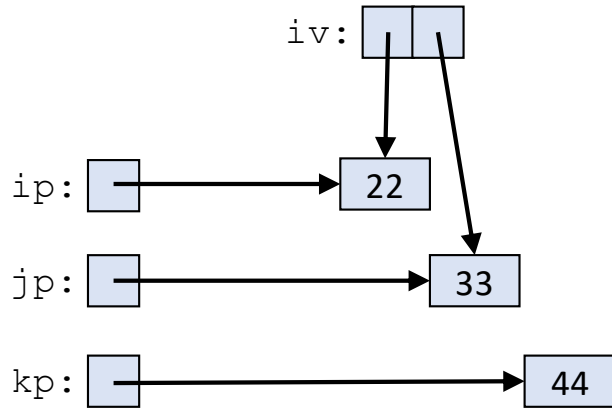
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Lines 11-13



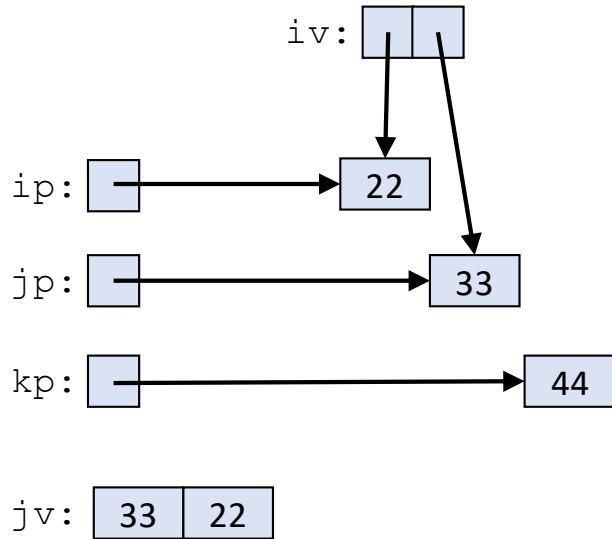
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Lines 15-16



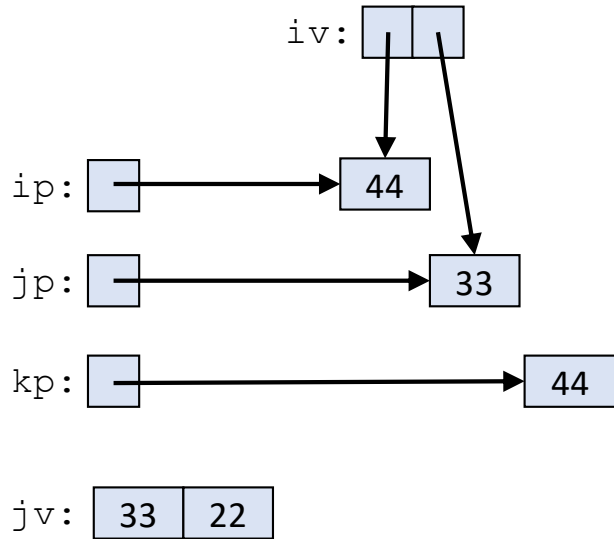
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```


Lines 18-19



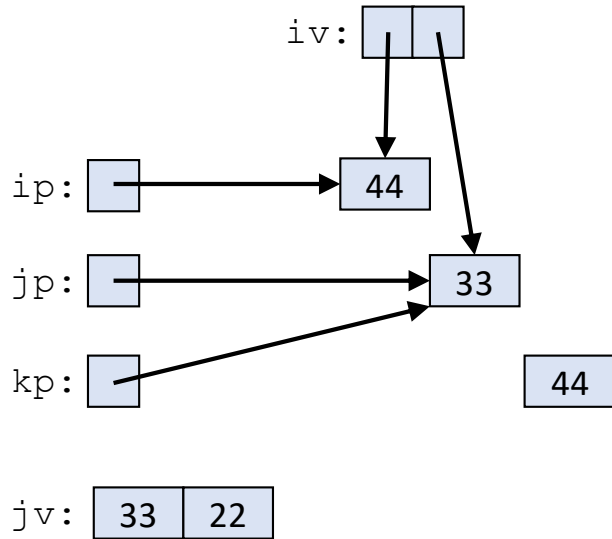
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Line 21



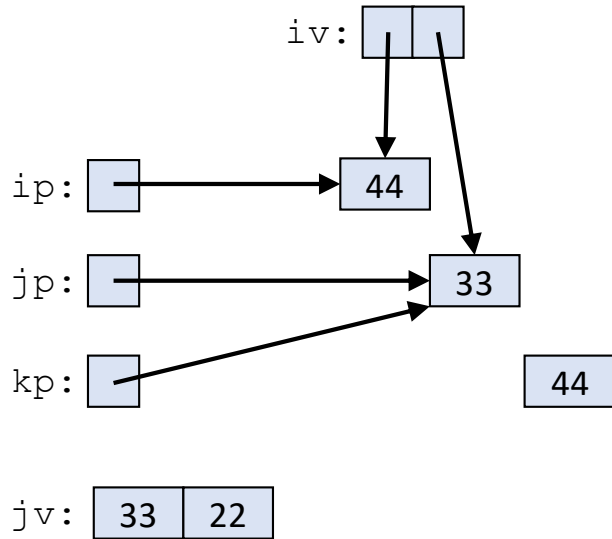
```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Line 22



```
int main() /* Line 1 */
{ /* Line 2 */
  vector <int *> iv; /* Line 3 */
  vector <int> jv; /* Line 4 */
  int *ip, *jp, *kp; /* Line 5 */
  /* Line 6 */
  ip = new int; /* Line 7 */
  jp = new int; /* Line 8 */
  kp = new int; /* Line 9 */
  /* Line 10 */
  *ip = 22; /* Line 11 */
  *jp = 33; /* Line 12 */
  *kp = 44; /* Line 13 */
  /* Line 14 */
  iv.push_back(ip); /* Line 15 */
  iv.push_back(jp); /* Line 16 */
  /* Line 17 */
  jv.push_back(*iv[1]); /* Line 18 */
  jv.push_back(*iv[0]); /* Line 19 */
  /* Line 20 */
  *(iv[0]) = *kp; /* Line 21 */
  kp = jp; /* Line 22 */
}
```

Line 22



Output:

```
*ip:      44
*jp:      33
*kp:      33
*iv[0]:   44
*iv[1]:   33
jp[0]:    33
jp[1]:    22
```

```
int main() /* Line 1 */
{ /* Line 2 */
    vector <int *> iv; /* Line 3 */
    vector <int> jv; /* Line 4 */
    int *ip, *jp, *kp; /* Line 5 */
    /* Line 6 */
    ip = new int; /* Line 7 */
    jp = new int; /* Line 8 */
    kp = new int; /* Line 9 */
    /* Line 10 */
    *ip = 22; /* Line 11 */
    *jp = 33; /* Line 12 */
    *kp = 44; /* Line 13 */
    /* Line 14 */
    iv.push_back(ip); /* Line 15 */
    iv.push_back(jp); /* Line 16 */
    /* Line 17 */
    jv.push_back(*iv[1]); /* Line 18 */
    jv.push_back(*iv[0]); /* Line 19 */
    /* Line 20 */
    *(iv[0]) = *kp; /* Line 21 */
    kp = jp; /* Line 22 */
}
```

Memory leak is on line 22.

Behold the program to the right. We compile it to the file **a.out** and then we run it with:

UNIX> **echo 30 50 70 | ./a.out**

- **Question 1:** What is the first line of output?
- **Question 2:** What is the second line of output?
- **Question 3:** What is the third line of output?
- **Question 4:** What is the fourth line of output?
- **Question 5:** What is the fifth line of output?
- **Question 6:** What is the sixth line of output?
- **Question 7:** What is the seventh line of output?
- **Question 8:** What is the eighth line of output? Enter a dash if there is no eighth line.
- **Question 9:** What is the ninth line of output? Enter a dash if there is no ninth line.

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

int main()
{
    vector <int *> v1;
    vector <int> v2;
    vector <int> *v3;
    int i;
    int *p;

    v3 = &v2;
    while (cin >> i) {
        p = new int;
        *p = i;
        v1.push_back(p);
        v2.push_back(*p);
    }


    for (i = 0; i < v1.size(); i++) *(v1[i]) += 2;
    for (i = 0; i < v2.size(); i++) v2[i] += 10;

    for (i = 0; i < v1.size(); i++) cout << *(v1[i]) << endl;
    for (i = 0; i < v2.size(); i++) cout << v2[i] << endl;
    for (i = 0; i < v3->size(); i++) cout << v3->at(i) << endl;
    return 0;
}
```

We start with all three vectors being empty, then v3 points to v2:

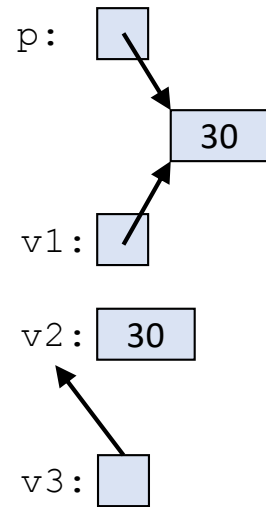
v1:

v2:

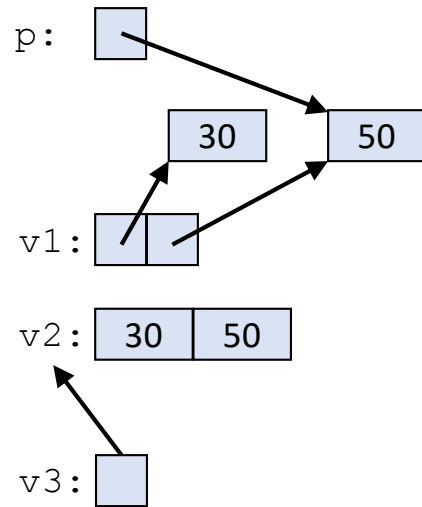
v3: 



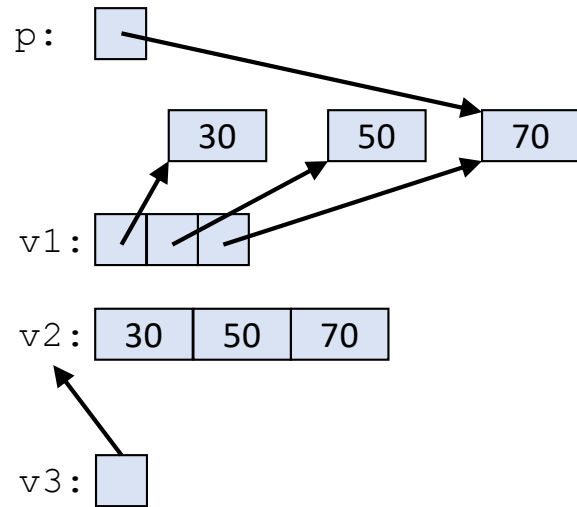
We then read 30. A new integer is allocated and set to 30. A pointer to that is put onto v1, And a copy of it is put onto v2:



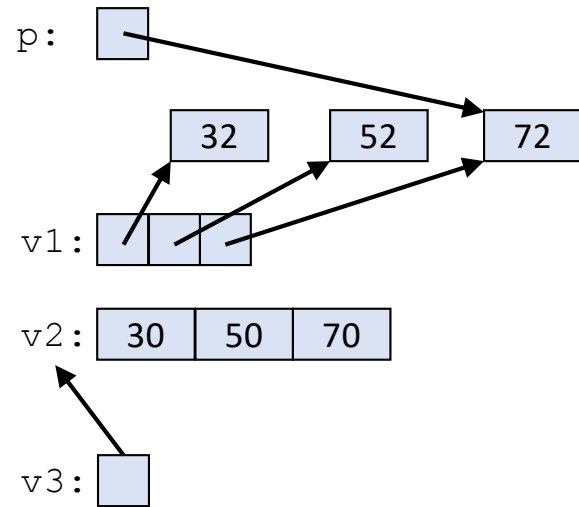
We then read 50. A new integer is allocated and set to 50. A pointer to that is put onto v1, And a copy of it is put onto v2:



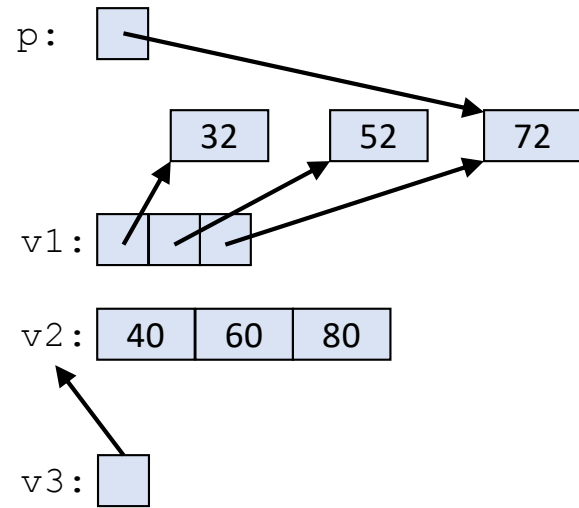
We then read 70. A new integer is allocated and set to 70. A pointer to that is put onto v1, And a copy of it is put onto v2:



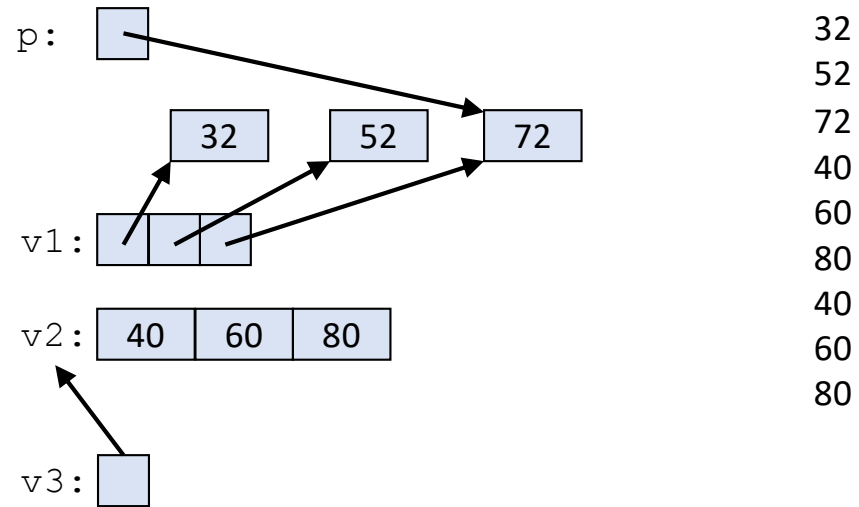
We add two to every value pointed to by v1:



We add 10 to every value in v2:



Now we print out three values for v1, v2 and v3:



On the right are two programs that both do the same thing. If you care, `rand()` returns a pseudorandom number between 0 and 2^{31} . When I time `p1` on one computer, the following takes three seconds:

```
UNIX> echo 10000 100000 | ./p1
```

- **Question 1:** Roughly how many seconds will the following take?

```
UNIX> echo 10000 400000 | ./p1
```

- **Question 2:** Roughly how many seconds will the following take?

```
UNIX> echo 30000 700000 | ./p1
```

I move to a different computer, and the following takes three seconds:

```
UNIX> echo 10000 100000 | ./p2
```

- **Question 3:** Roughly how many seconds will the following take?

```
UNIX> echo 10000 400000 | ./p2
```

- **Question 4:** Roughly how many seconds will the following take?

```
UNIX> echo 30000 700000 | ./p2
```

p1.cpp, compiled to p1.

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

int main()
{
    int vs, ns;
    vector <int> v;
    int matches, i, j, n;

    cin >> vs >> ns;

    for (i = 0; i < vs; i++) {
        v.push_back(rand()%100000);
    }

    matches = 0;
    for (i = 0; i < ns; i++) {
        n = rand()%100000;
        for (j = 0; j < v.size(); j++) {
            if (v[j] == n) matches++;
        }
    }
    cout << matches << endl;
    return 0;
}
```

p2.cpp, compiled to p2.

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

int main(int argc, char **argv)
{
    int vs, ns;
    vector < vector <int> > v;
    int matches, i, j, n;

    cin >> vs >> ns;
    v.resize(vs);

    for (i = 0; i < vs; i++) {
        n = rand()%100000;
        v[n%vs].push_back(n);
    }

    matches = 0;
    for (i = 0; i < ns; i++) {
        n = rand()%100000;
        for (j = 0; j < v[n%vs].size(); j++) {
            if (v[n%vs][j] == n) matches++;
        }
    }
    cout << matches << endl;
    return 0;
}
```

Question 1

The second **for** loop now runs 400,000 iterations instead of 100,000. It is doing the same amount of work in each iteration (looping through 10,000) numbers. So this will take four times as long. The answer is 12.

Question 2

The second **for** loop now runs 700,000 iterations instead of 100,000. It is traversing a vector that is three times the size as before, so each iteration will take three times as long. The whole program will thus take 21 times as long. The answer is 63.

We don't need to factor in the time for the first loop -- it is going to be *so* much faster than the second loop that we don't need to care about it.

Question 3

Once again, the second **for** loop now runs 400,000 iterations instead of 100,000. It is doing the same amount of work in each iteration (looking for numbers in a hash table that uses separate chaining). As with Question 2, it will take four times as long. The answer is 12. You'll note that the computer running this program must be a lot slower than the computer in Questions 1 and 2.

Question 4

We triple the size of our hash table, but its load factor is still 1. So the inner loop of the second **for** loop is doing the same amount of work as in Question 3. Thus, the slowdown is only 7. The answer is 21. The difference between the two loops will be less than in Questions 1 and 2, but the first loop is fast enough to be considered in the noise, compared to the second loop.

Question 1

Behold the declaration of the **MyClass** class to the right. Please answer the following true/false questions:

- A. The copy constructor is declared on line 3.
- B. The copy constructor is declared on line 4.
- C. The copy constructor is declared on line 5.
- D. The method **A()** cannot change **s**.
- E. The method **B()** cannot change **s**.
- F. The method **C()** cannot change **s**.
- G. The method **D()** cannot change **s**.
- H. The method **A()** cannot change **v**.
- I. The method **B()** cannot change **v**.
- J. The method **C()** cannot change **v**.
- K. The method **D()** cannot change **v**.
- L. When you call **A()**, you will call **Otherclass'** copy constructor.
- M. When you call **B()**, you will call **Otherclass'** copy constructor.
- N. When you call **A()**, you will call **Otherclass'** regular constructor.
- O. When you call **B()**, you will call **Otherclass'** regular constructor.
- P. When **A()** returns, it will call the destructor for **s**.
- Q. When **B()** returns, it will call the destructor for **s**.
- R. This header won't compile, because **F** should be protected.
- S. If **x** and **y** are declared as **MyClass** variables, I'm allowed to say "**x=y**", even though I didn't declare an assignment overload.
- T. There is a memory leak, because there is no destructor to free up the memory corresponding to the variable **v**.
- U. If **Copy()** calls **new**, I should have a destructor that calls **delete**.

```
/* 1 */ class MyClass {
/* 2 */     public:
/* 3 */         MyClass();
/* 4 */         MyClass *Copy();
/* 5 */         MyClass(const MyClass &mc);
/* 6 */         void A(Otherclass &s) const;
/* 7 */         void B(Otherclass s);
/* 8 */         void C(Otherclass &s);
/* 9 */         void D(const Otherclass &s);
/* 10 */        int F;
/* 11 */        protected:
/* 12 */        vector <int> v;
/* 13 */ };
```

CS140 Midterm Exam - October 15, 2019 Answers and Grading

James S. Plank

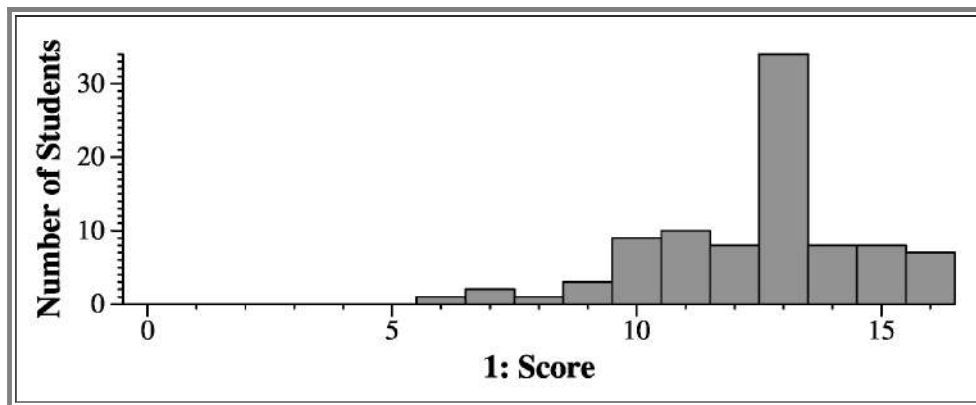
Question 1: 16 Points

- Part *A*: The copy constructor is on line 5. **False**
- Part *B*: The copy constructor is on line 5. **False**
- Part *C*: The copy constructor is on line 5. **True**
- Part *D*: The reference parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *E*: The parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *F*: The reference parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *G*: The reference parameter is declared **const**, so the method cannot change *s*. **True**
- Part *H*: The method is declared **const**, so it cannot change *v*. **True**
- Part *I*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *J*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *K*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *L*: Since *s* is a reference parameter, there is no constructor called when **A()** is called, as **A()** simply gets a reference to the caller's *s*: **False**
- Part *M*: Since *s* is not a reference parameter, it will make a copy of the caller's parameter, and that will use the copy constructor: **True**.
- Part *N*: See part *L*: No constructor is called: **False**
- Part *O*: See part *M*: The copy constructor is called: **False**
- Part *P*: Since nothing was constructed, nothing needs to be destructed. **False**
- Part *Q*: The copy needs to be destructed when the method returns: **True**
- Part *R*: Variables may be public: **False**
- Part *S*: This is fine -- it simply calls a default assignment operator, which will make copies of **F** and **v**: **True**
- Part *T*: Vectors and strings are destroyed automatically. **False**
- Part *U*: This is identical to the **Copy()** method in the bitmatrix lab. It is up to the caller to free the pointer. You'll note, there's nowhere to store the pointer in the class definition, so it can't delete anything: **False**

Grading:

0.75 points for parts *A* through *Q*, and parts *S* through *U*. 1 point for part *R*. Here are counts of correct answers (91 students):

A: 89 F	B: 44 F	C: 55 T	D: 68 F	E: 61 F
F: 86 F	G: 89 T	H: 72 T	I: 85 F	J: 86 F
K: 81 F	L: 68 F	M: 67 T	N: 60 F	O: 55 F
P: 78 F	Q: 77 T	R: 89 F	S: 64 T	T: 81 F
U: 73 T				



Question 2: 10 Points

I know, no one likes to trace through code, but how else to get you to demonstrate to me that you understand the basics of **try/catch**? Let's walk through it:

- At the first **cout** statement, *i* is 0, so: **0**.
- Since 0 is even, we don't throw, so *i* becomes 3 and we skip the catch clause: **3**.
- We add two to *i*, which becomes 5, and go back to the top of the while loop. It prints: **5**.
- Now, five is odd, so we throw it. The **catch** clause prints 5 again, and then adds 5 + 5 to set 5 to 10. After the **catch** clause, it prints: **10**.
- At the end of the while loop, it increments *i* by two again, to turn it into 12. The while loop exits, and we print: **12**.

Tell me the output of the program to the right. Put your answer all on one line -- if there's a newline in the program's output, simply replace it with a space, or even no space. Both will be fine.

So, for example, if the output is:

```
A
B
C
```

Then your answer should be "ABC" or "A B C".

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

class Fred {
public:
    Fred();
    Fred(const Fred &f);
    ~Fred();
};

Fred::Fred()
{
    cout << "A" << endl;
}

Fred::Fred(const Fred &f)
{
    cout << "B" << endl;
}

Fred::~~Fred()
{
    cout << "C" << endl;
}

void proc1(Fred f, int i)
{
    cout << "X" << endl;
    if (i != 0) throw (string) "E";
}

void proc2(const Fred &f)
{
    cout << "Y" << endl;
}

int main()
{
    Fred f;
    Fred *f2;

    try {
        proc1(f, 0);
        proc2(f);
        f2 = new Fred;
        proc1(f, 1);
    } catch (const string &s) {
        cout << s << endl;
        return 0;
    }

    cout << "F" << endl;
    return 0;
}
```

Answer to the Clicker Question

- The very first thing that happens is the f's constructor is called, because it has been declared inside **main()**. That prints "A".
- Next "proc1(f,0)" is called. That calls the copy constructor for the parameter. It prints "B".
- "proc1(f,0)" prints "X".
- "proc1(f,0)" tries the if statement, which is false. It returns, calling the destructor for its parameter: "C".
- "proc2(f)" is called. It doesn't call a constructor, because its parameter f is a reference parameter. It prints "Y" and returns.
- "f2 = new Fred" is called. That calls the constructor for "Fred". It prints "A".
- "proc1(f,1)" is called. That calls the copy constructor for the parameter. It prints "B".
- "proc1(f,1)" prints "X".
- "proc1(f,1)" throws the exception, which makes the procedure return. Before returning it calls the destructor for its parameter f: "C".
- The exception is caught in the "catch" statement. That prints "E".
- The "catch" statement now returns, which calls the destructor for main's f. That prints "C".
- You'll note that the destructor for f2 is not called -- you would have to call "delete" for that.

The answer is "A B X C Y A B X C E C" or "ABXCYABXCEC".

Clicker Questions

- **Question 1:** What is the output of **p1.cpp** when it runs with **input-1.txt** as standard input?
- **Question 2:** What is the output of **p2.cpp** when it runs with **input-2.txt** as standard input?

input-1.txt

Samuel
Amelia
Lucas
Ian
Cole
Brooke
Bailey
Lucas
Charlie
Samuel

input-2.txt

Gabriel
Wyatt
Eva
Lily
Sarah
Benjamin
Nicholas
Michael
Landon
Madelyn

p1.cpp

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

int main()
{
    set <string> n;
    set <string>::const_iterator nit;
    string s;

    while (cin >> s) n.insert(s);
    for (nit = n.begin(); nit != n.end(); nit++) {
        cout << nit->at(0) ;
    }
    cout << endl;
    return 0;
}
```

p2.cpp

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

int main()
{
    set <string> n;
    set <string>::const_iterator nit;
    string s;

    while (cin >> s) n.insert(s);
    for (nit = n.begin(); nit != n.end(); nit++) {
        cout << nit->at(0) ;
        nit++;
    }
    cout << endl;
    return 0;
}
```

Answers to Clicker Questions

Question 1

Remember that you don't insert duplicate elements into a set.

```
UNIX> g++ p1.cpp
UNIX> ./a.out < input-1.txt
ABBCCILS
UNIX>
```

Question 2

```
UNIX> g++ p2.cpp
UNIX> ./a.out < input-2.txt
BGLMS
UNIX>
```

Clicker Questions

For each of these, please use the following multiple choice answers:

- A: $O(1)$
- B: $O(\log n)$
- C: $O(n)$
- D: $O(n \log n)$
- E: $O(n^2)$

Question 1: Inserting an element into a multiset with n elements.

Question 2: Inserting n elements into an empty multiset.

Question 3: Inserting n elements into a multiset with n elements.

Question 4: Erasing the first element of a vector with n elements.

Question 5: `for (i = 0; i < n; i++) for (j = 0; j < i; j++) k++;`

Question 6: Calling `push_back()` on a vector with n elements.

Question 7: Creating Pascal's triangle with n levels.

Clicker Answers:

Question 1: *B: $O(\log n)$* : Basic operation on a set/map/multiset/multimap.

Question 2: *D: $O(n \log n)$* : Just memorize this for now. Actually, you can prove it by using the answer to Question 3 to show that inserting the second half of elements is $O(n \log n)$.

Question 3: *D: Still $O(n \log n)$* . This is easier -- you are doing n operations, and each of them is between $O(\log n)$ and $O(\log 2n)$. Since $\log 2n$ is equal to $(\log n)+1$, we have that the n operations are bounded by $O(n \log n + n)$, which is $O(n \log n)$.

Question 4: *C: $O(n)$* : This is why you should never use this method on a vector.

Question 5: *E: $1 + 2 + 3 + 4 + \dots + n = O(n^2)$* .

Question 6: *A: $O(1)$* : Basic operation on a vector.

Question 7: *E: Each level i has i entries -- this is the same as Question 5: $O(n^2)$*

Clicker Questions

Question 1: Suppose I have a square lawn with a width of n feet. And suppose I want to fence it in the following way: I'm going to set a 4"x4"by4' wooden post every six feet (potentially less in the corners). And I am going to run three 1"x3"x6' wooden boards between every consecutive pair of posts. Please give me the cleanest big-o expression of number boards and posts I will have to purchase.

Question 2: Suppose it takes me 2 seconds to mow a 6'x1' rectangle in my field. Please give me the cleanest big-o expression of number of minutes that it takes me to mow my lawn.

Question 3: Suppose my field is an equilateral triangle whose sides are each n feet long. Please give me the cleanest big-o expression of the number of pieces of wood that I have to purchase in order to fence it like I did in question 1.

Question 4: It's n miles to to the mountains, and there are 5 stop lights. I have to wait up to 50 seconds at a light. If I'm really unlucky and reach each stop light just as it's turning red, but otherwise I drive at an average of 50 MPH, please give me the cleanest big-o expression for the number of hours it takes me to get to the mountains.

Question 5: Suppose I have a computer memory with n bits, and I'm going to store just one number on it. Please give me the cleanest big-o expression for how many different numbers can I store in the memory?

Clicker Answers:

Question 1: $O(n)$: For each side of the fence, you'll need roughly $n/6$ posts and $3(n/6)$ boards. Their sum is $O(n)$ and four times this is also $O(n)$.

Question 2: $O(n^2)$: The area is n^2 square feet, and you can partition that into 6 square-foot regions to mow. That's roughly $n^2/6$, which is $O(n^2)$. On your clicker answer, you can write " n^2 " or " $n*n$ ".

Question 3: Now there are three sides instead of four. It's still $O(n)$.

Question 4: Your time is $n/50$ hours plus $5*50$ seconds. This is an equation of the form $a*n + b$, where a and b are constants, which is $O(n)$.

Question 5: You can hold 2^n numbers in an n -bit memory, so the answer is $O(2^n)$.

Clicker Questions

<pre>#include "stack.hpp" #include <iostream> #include <sstream> #include <vector> using namespace std; void a(Stack *s) { vector <Stack> v; v.resize(s->Size(), *s); // Line F return; // Line G }</pre>	<pre>int main() { int n; int i; Stack s1; Stack s2; Stack *s3; ostringstream ss; cin >> n;</pre>	<pre>for (i = 0; i < n; i++) { ss.clear(); ss.str(""); ss << i; s1.Push(ss.str()); } s2 = s1; // Line A s3 = new Stack; // Line B *s3 = s2; // Line C a(s3); // Line D delete s3; // Line E while (!s2.Empty()) cout << s2.Pop() << endl; return 0; }</pre>
--	---	--

The answer to each of these will be $O(f(n))$. In your answer, just specify $f(n)$.

Question 1: What is the big-O running time of Line A?

Question 2: What is the big-O running time of Line B?

Question 3: What is the big-O running time of Line C?

Question 4: What is the big-O running from the beginning of Line D until just before Line F executes?

Question 5: What is the big-O running time of Line F?

Question 6: What is the big-O running from the beginning of Line G until just before Line E executes?

Question 7: What is the big-O running time of Line E?

Clicker Answers

- Question 1: This will call the assignment overload, and copy each of the n nodes from $s1$ to $s2$: $O(n)$.
- Question 2: This simply sets a pointer from the memory allocator. The pointer is to a few bytes, so this is $O(1)$.
- Question 3: This once again calls the assignment overload, which copies each of the n nodes from $s2$ to $s3$. $O(n)$.
- Question 4: This calls the procedure and copies the pointer, not the stack, as a parameter. This is $O(1)$.
- Question 5: The vector has n Stacks and each one is copied from a stack with n elements: $O(n^2)$.
- Question 6: Now, the destructor is called on each of those stacks, so it will delete n^2 Stacknodes: $O(n^2)$.
- Question 7: This calls the destructor on the stack, which deletes the n nodes: $O(n)$.

Suppose that the program to the right is compiled into the executable `a.out`. In questions 1 through 3, please tell me the output of that command when typed into the shell.

Question 1: `echo B | ./a.out`

Question 2: `echo AA | ./a.out`

Question 3: `echo BAD | ./a.out`

Question 4: If the string entered on standard input has n characters, what is the big-O running time of the program?

Question 5: If the string entered on standard input has n characters, what is the big-O memory usage of "the stack"?

Bonus Question: Suppose the parameter `s` were not a reference parameter. Then what is the big-O memory usage of "the stack"?

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

void a(const string &s, int index)
{
    if (index == s.size()) return;

    cout << s[index];

    if (s[index] == 'A') cout << "1";

    a(s, index+1);

    cout << s[index];
}

int main()
{
    string s;

    cin >> s;
    a(s, 0);
    cout << endl;
    return 0;
}
```

Answers to today's clicker questions:

Question 1: This makes one recursive call, which returns instantly, so it simply prints 'B' twice. The answer is "BB".

Question 2: Let's go through what happens here:

- `a("AA",0)` first prints 'A'.
- `a("AA",0)` then prints '1', because character 0 is 'A'.
- `a("AA",0)` calls `a("AA",1)`.
- `a("AA",1)` prints 'A'
- `a("AA",1)` then prints '1', because character 1 is 'A'.
- `a("AA",1)` calls `a("AA",2)`.
- `a("AA",2)` returns instantly.
- `a("AA",1)` prints 'A' again, and then returns.
- `a("AA",0)` prints 'A' again, and then returns.

So the answer is "A1A1AA".

Question 3: Let's go through what happens:

- `a("BAD",0)` first prints 'B'.
- `a("BAD",0)` calls `a("BAD",1)`.
- `a("BAD",1)` prints 'A' and then '1'.
- `a("BAD",1)` calls `a("BAD",2)`.
- `a("BAD",2)` prints 'D'.
- `a("BAD",2)` calls `a("BAD",3)`.
- `a("BAD",3)` returns instantly.
- `a("BAD",2)` prints 'D' again and then returns.
- `a("BAD",1)` prints 'A' again and then returns.
- `a("BAD",0)` prints 'B' again and then returns.

The answer is "BA1DDAB"

Question 4: There are n recursive calls, and each call to `a()` does $O(1)$ work. The answer is $O(n)$.

Question 5: The stack contains local variables and procedure parameters for each recursive call. There are no local variables, and two procedure parameters, each of which is $O(1)$ (the reference parameters is a pointer, which is most commonly 8 bytes, and the integer is 4 bytes). So each context on the stack is $O(1)$ and there are n of these. The answer is therefore $O(n)$.

Bonus Question: If s is not a reference parameter, then a copy of the string will be stored on each stack context. That means each recursive call consumes $O(n)$ on the stack. The answer is therefore $O(n^2)$.

Question 1: Assuming open addressing, what is the running time of inserting an element into a hash table with n elements and a load factor of 0.5?

Question 2: Assuming open addressing, what is the running time of inserting an element into a hash table with n elements and only 10 empty slots?

Question 3: Assuming separate chaining, what is the running time of inserting an element into a hash table with n elements and a load factor of f ?

Question 4: What is the running time of the following code:

```
k = 0;
for (i = 0; i < 1000*n; i++) k++;
```

Question 5: What is the running time of the following code:

```
k = 0;
for (i = 0; i < 2*n; i++) {
    for (j = 0; j < 10000; j++) k++;
}
```

Question 6: What is the running time of the following code:

```
k = 0;
for (i = 1; i < (n * n); i *= 2) k++;
```

Answers to clicker questions

Question 1: When half of the table is empty, your expected number of probes to find an empty slot is 2, so the answer is $O(1)$.

Question 2: With only 10 empty slots, your expected number of probes becomes $n/10$, so the answer is $O(n)$. I won't tell you how to calculate this, just trust me.

Question 3: With a load factor of f , the average size of the vector in a hash table entry is f , so the answer is $O(f)$.

Question 4: The loop runs $1000n$ times, which is $O(n)$.

Question 5: The outer loop runs $2n$ times, so its number of iterations is $O(n)$. The inner loop runs 10,000 times so its number of iterations is $O(1)$. $O(n) * O(1) = O(n)$.

Question 6: The loop runs $\log(n^2)$ times.

$$\log(n^2) = 2 * \log(n)$$

Therefore, the loop's running time is $O(\log n)$.

Suppose that the program to the right is compiled into the executable `a.out`. In questions 1 through 3, please tell me the output of that command when typed into the shell. If the output is multiple lines, simply type it on one line separated by spaces.

Question 1: `echo 1 6 | ./a.out | tail -n 1`

Question 2: `echo 2 2 | ./a.out | tail -n 1`

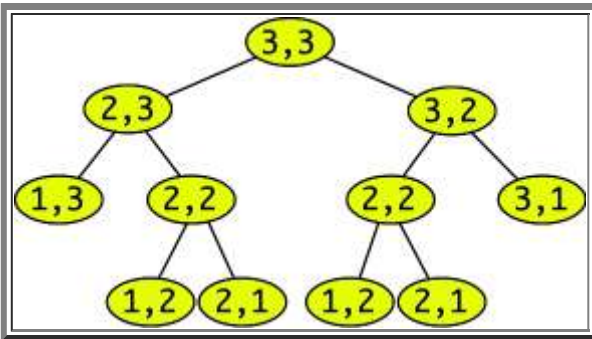
Question 3: `echo 2 3 | ./a.out | head -n 3`

Question 4: The tree drawn below represents the recursive calls that are made when you do the following:

```
echo 3 3 | ./a.out | tail -n 1
```

What kind of traversal would you use on this tree to calculate what gets printed?

Question 5: What is the output? (Use the tree)



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

int a(int b, int c)
{
    cout << b << ", " << c << endl;
    if (b == 1 || c == 1) return c*b;
    return a(b-1,c) * a(b,c-1);
}

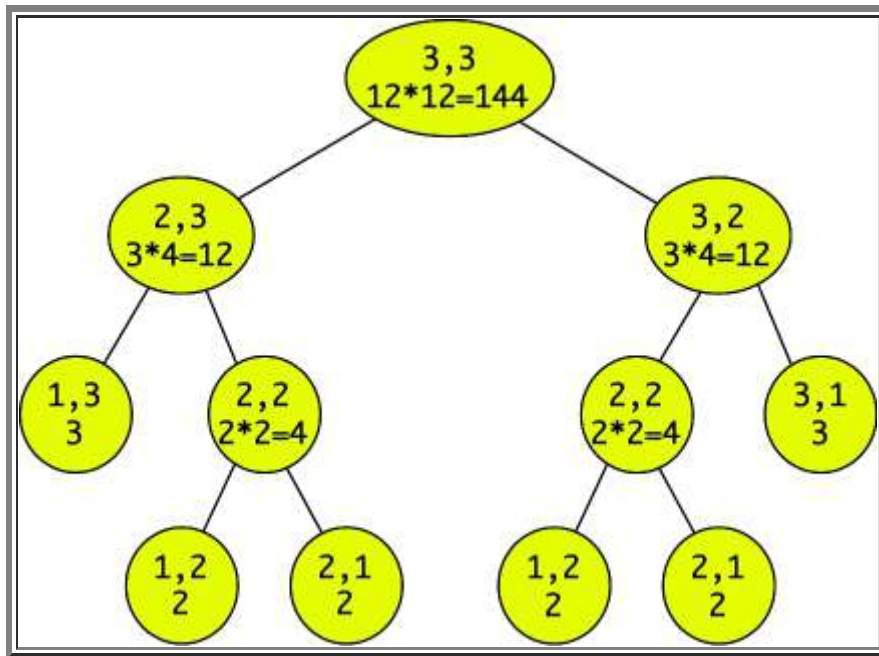
int main()
{
    int b, c, ans;

    cin >> b >> c;

    ans = a(b,c);
    cout << ans << endl;
    return 0;
}
```

Answers to the Clicker Questions

- **Question 1:** It returns $1*6 = 6$ instantly.
- **Question 2:** $a(2,2)$ calls $a(1,2)$ and $a(2,1)$ and returns their product. Both $a(1,2)$ and $a(2,1)$ return 2, so the answer is 4.
- **Question 3:** $a(2,3)$ calls $a(1,3)$, which returns 3, and then $a(2,2)$. So the answer is "2,3 1,3 2,2".
- **Question 4:** You calculate the answer with a postorder traversal -- only after you have the return values of $a(b-1,c)$ and $a(b,c-1)$ can you calculate your return value.
- **Question 5:** Here's the tree filled out with the values from the postorder traversal. The answer is 144.



Suppose that the program to the right is compiled into the executable `a.out`. Please answer with the output of the following:

Question 1: `echo A | ./a.out`

Question 2: `echo AA | ./a.out`

Question 3: `echo BA | ./a.out`

Question 4: `echo BBBAA | ./a.out`

Question 5: `echo AAABB | ./a.out`

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

void a(const string &s, int index)
{
    if (index >= s.size()) return;

    if (s[index] == 'A') {
        cout << '(';
        a(s, index+1);
        cout << ')';
    } else {
        cout << "()";
        a(s, index+1);
    }
}

int main()
{
    string s;

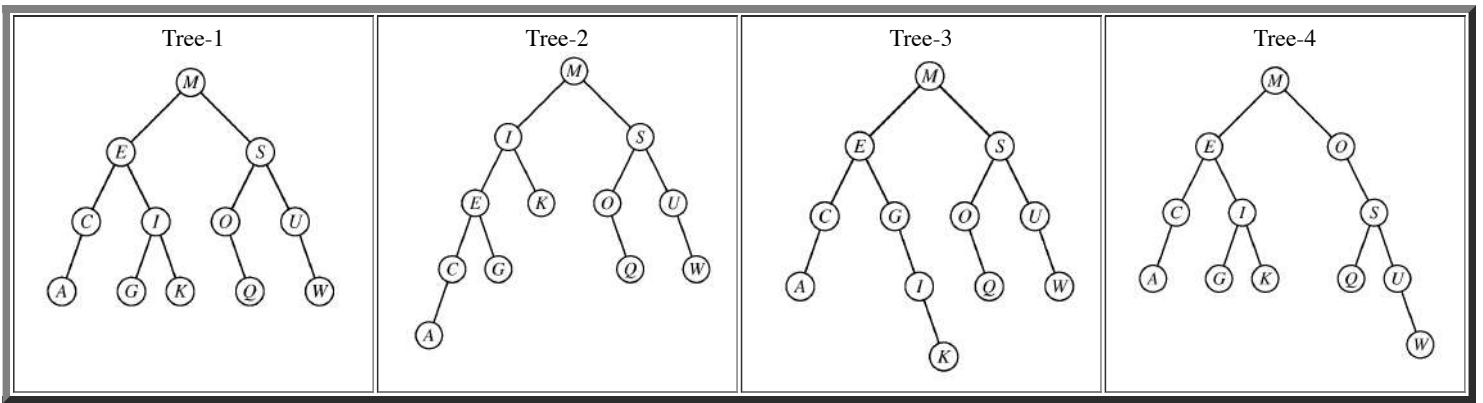
    cin >> s;
    a(s, 0);
    cout << endl;
    return 0;
}
```

Clicker Answers

Just compile and run:

```
UNIX> echo A | ./a.out  
(  
UNIX> echo AA | ./a.out  
((  
UNIX> echo BA | ./a.out  
(()  
UNIX> echo BBBAA | ./a.out  
(())()  
UNIX> echo AAABB | ./a.out  
(((())  
UNIX>
```

Below are four trees:



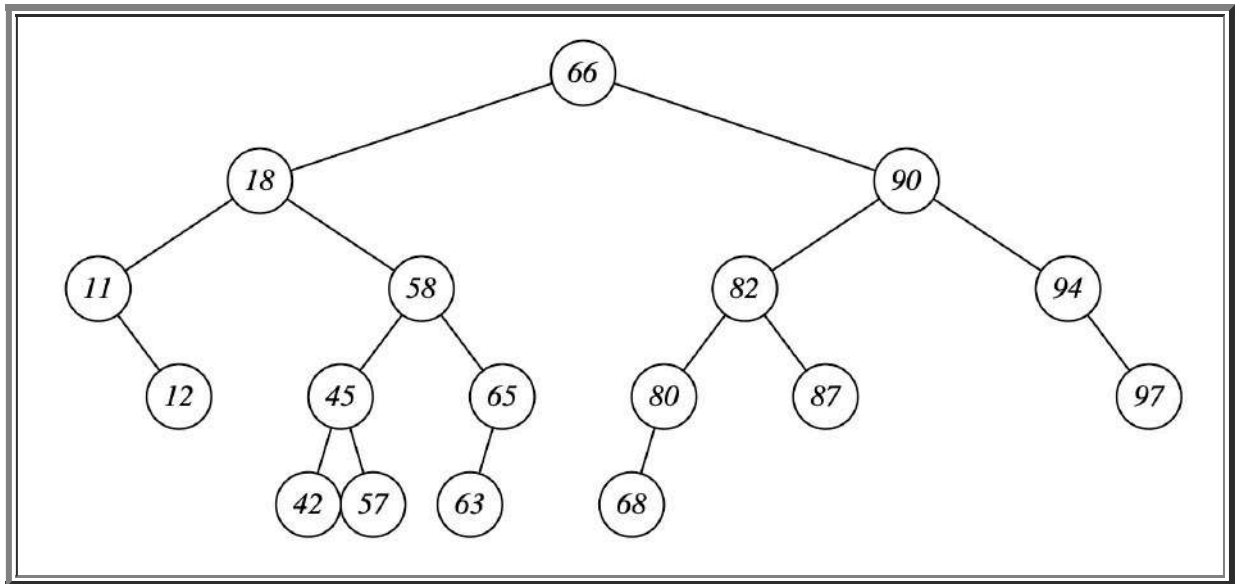
Each question has an answer which is one letter.

- **Question 1:** About which node do we rotate to turn Tree-1 into Tree-2?
- **Question 2:** About which node do we rotate to turn Tree-1 into Tree-3?
- **Question 3:** About which node do we rotate to turn Tree-1 into Tree-4?
- **Question 4:** Tree-2 is not an AVL tree. Name a node that is imbalanced.
- **Question 5:** Tree-3 is not an AVL tree. Name a node that is imbalanced.
- **Question 6:** Tree-4 is not an AVL tree. Name a node that is imbalanced.

Answers to the clicker questions.

- **Question 1:** For this and the next two questions, you look for the node who swaps parent-child relationship from the first tree to the second tree. The answer is the node that was the child. In this case: **I**.
- **Question 2:** **G**.
- **Question 3:** **O**.
- **Question 4:** **I**: Its left subtree has a height of 3, and its right subtree has a height of 1. All other nodes are balanced.
- **Question 5:** **G**: Its left subtree has a height of 0, and its right subtree has a height of 2. All other nodes are balanced.
- **Question 5:** **O**: Its left subtree has a height of 0, and its right subtree has a height of 3. All other nodes are balanced.

All of the questions below pertain to the tree to the right. For each pair of questions, ignore the previous questions -- in other words, each pair of questions pertains to this tree and not to the results of previous AVL tree operations.



Question 1: If I insert 72 into the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

Question 2: About which node will I perform the rotation(s)?

Question 3: If I insert 99 into the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

Question 4: About which node will I perform the rotation(s)?

Question 5: If I insert 19 into the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

Question 6: About which node will I perform the rotation(s)?

Question 7: If I delete 94 from the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

Question 8: About which node will I perform the rotation(s)?

Question 9: If I delete 87 from the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

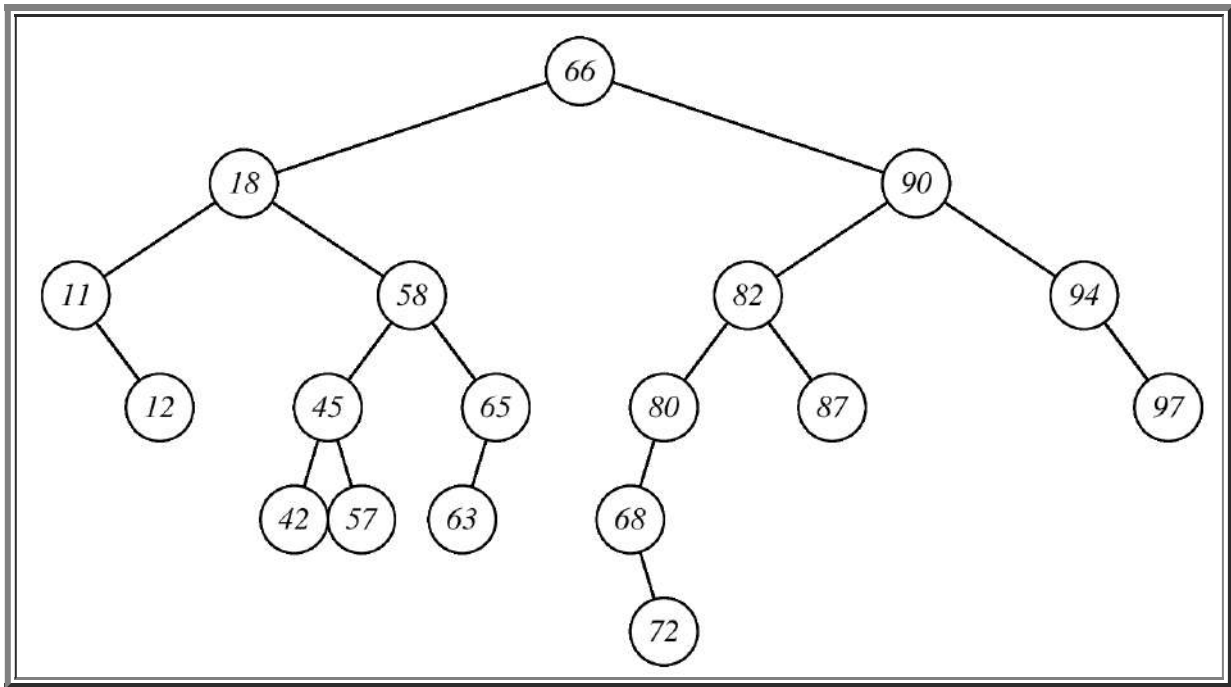
Question 10: About which node will I perform the rotation(s)?

Question 11: If I delete 18 from the tree, what type of rebalancing operation will I do (answer "zigzig" or "zigzag")?

Question 12: About which node will I perform the rotation(s)?

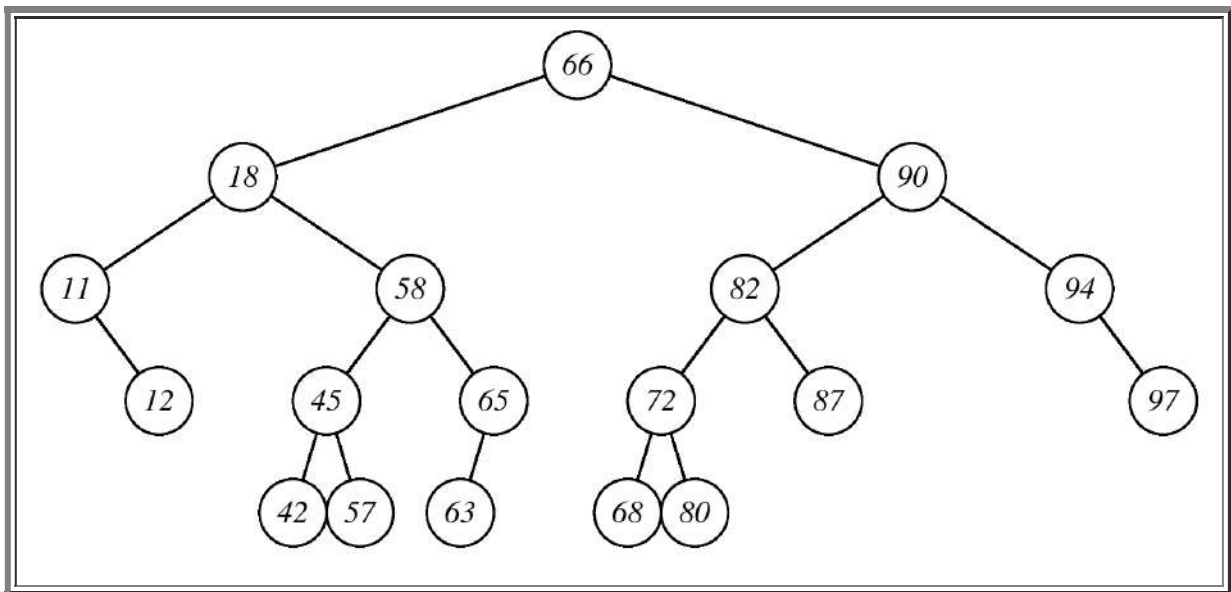
Answers

Question 1: When I insert 72, I get:

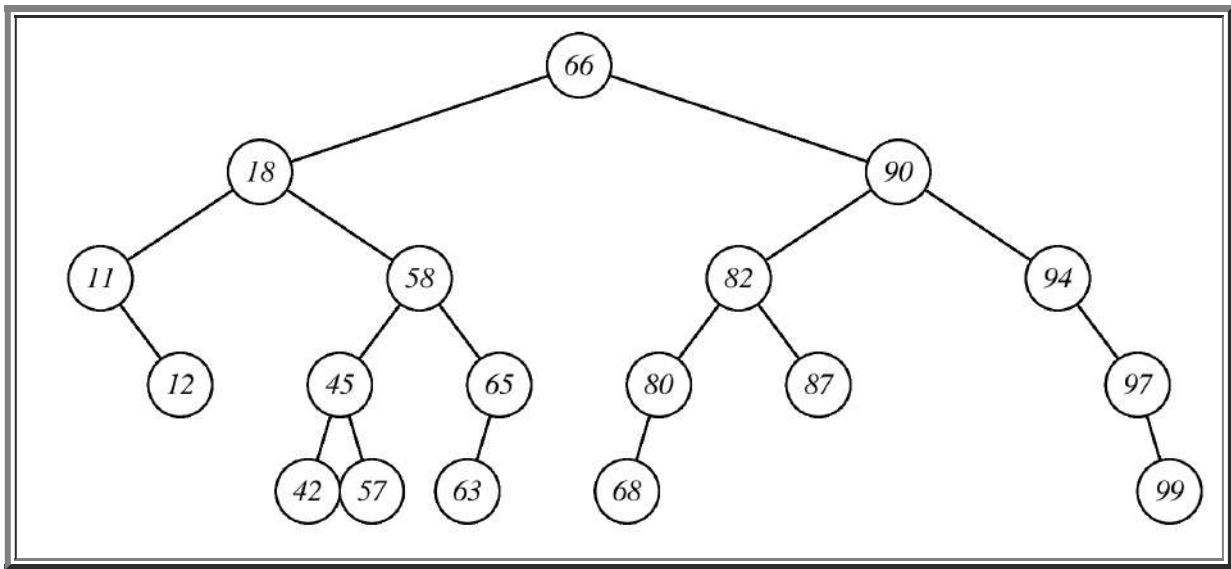


Node 80 is imbalanced, and the imbalance goes left-right. So it's a zigzag.

Question 2: To fix, I rotate twice about the grandchild of the imbalanced node. That's node 72. Here's the result.

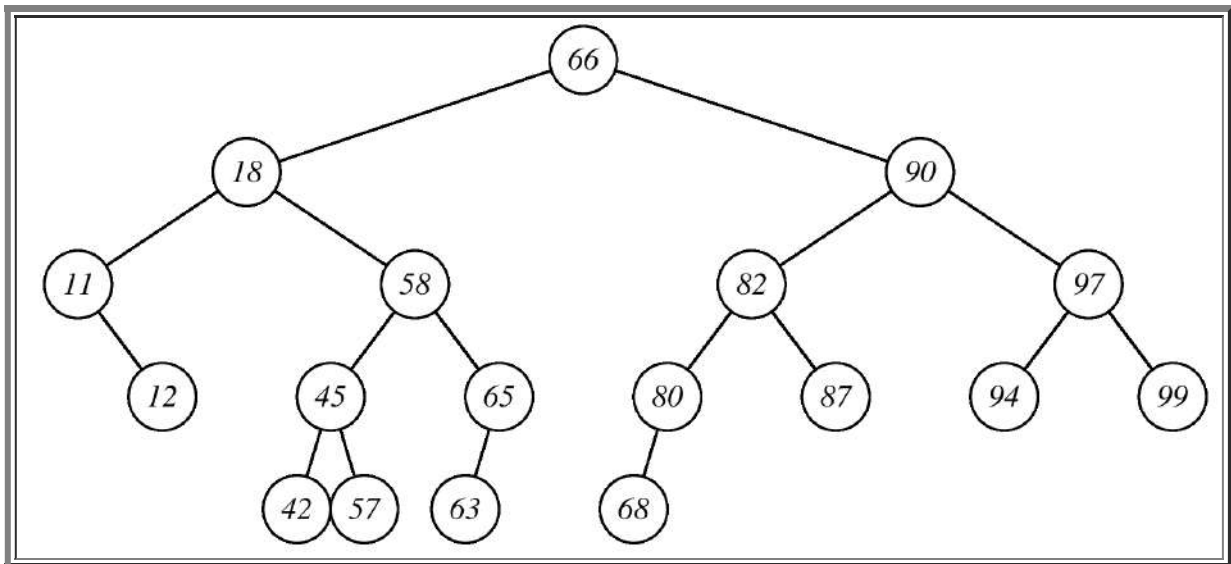


Question 3: When I insert 99, I get:

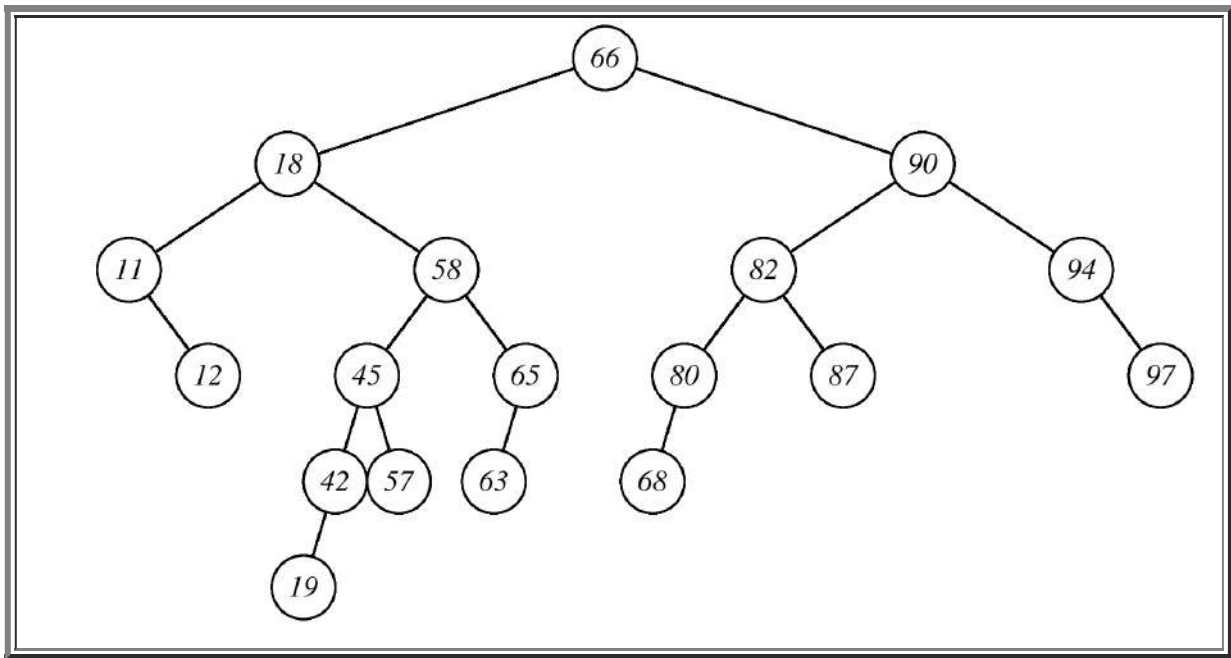


Node 94 is imbalanced, and the imbalance goes right-right. So it's a zigzig.

Question 4: To fix, I rotate once about the child of the the imbalanced node. That's node 97. Here's the result.

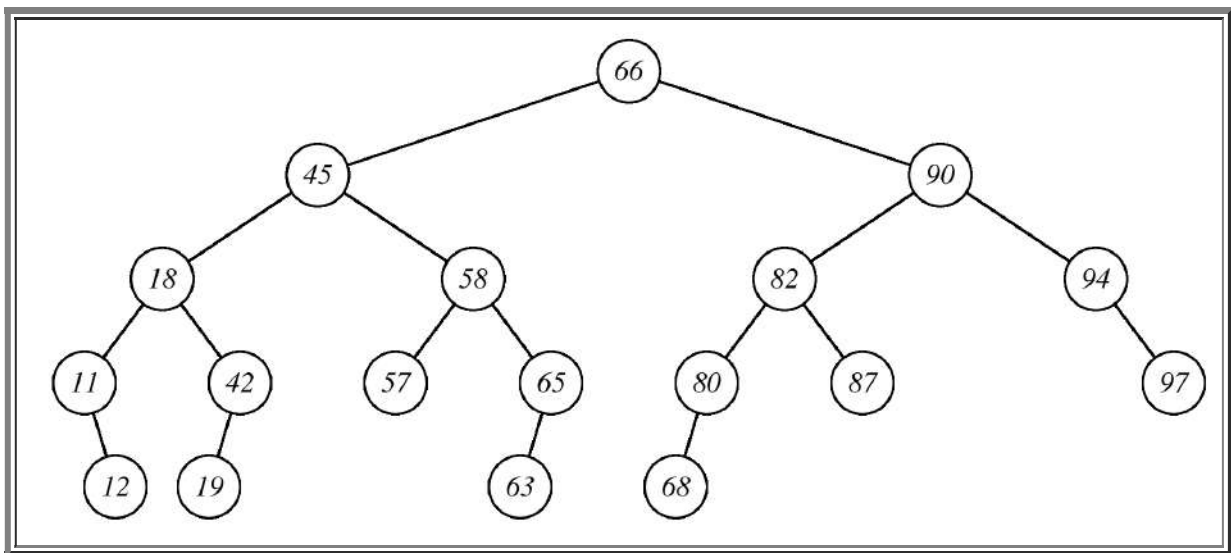


Question 5: When I insert 19, I get:

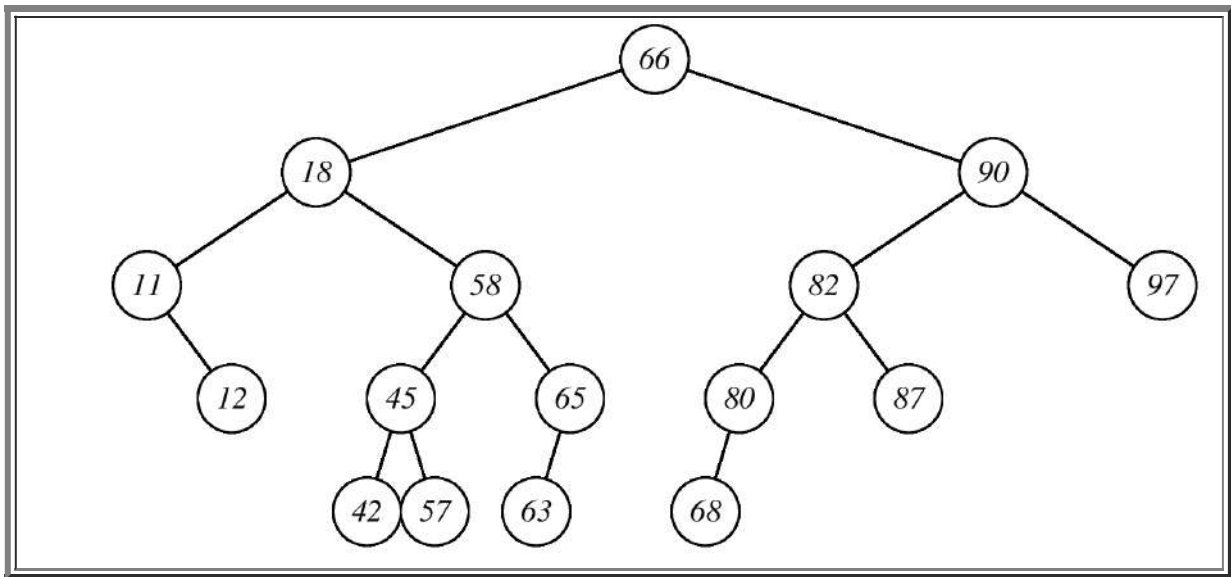


Node 18 is imbalanced, and the imbalance goes right-left. So it's a zigzag.

Question 6: To fix, I rotate twice about the grandchild of the the imbalanced node. That's node 45. Here's the result.

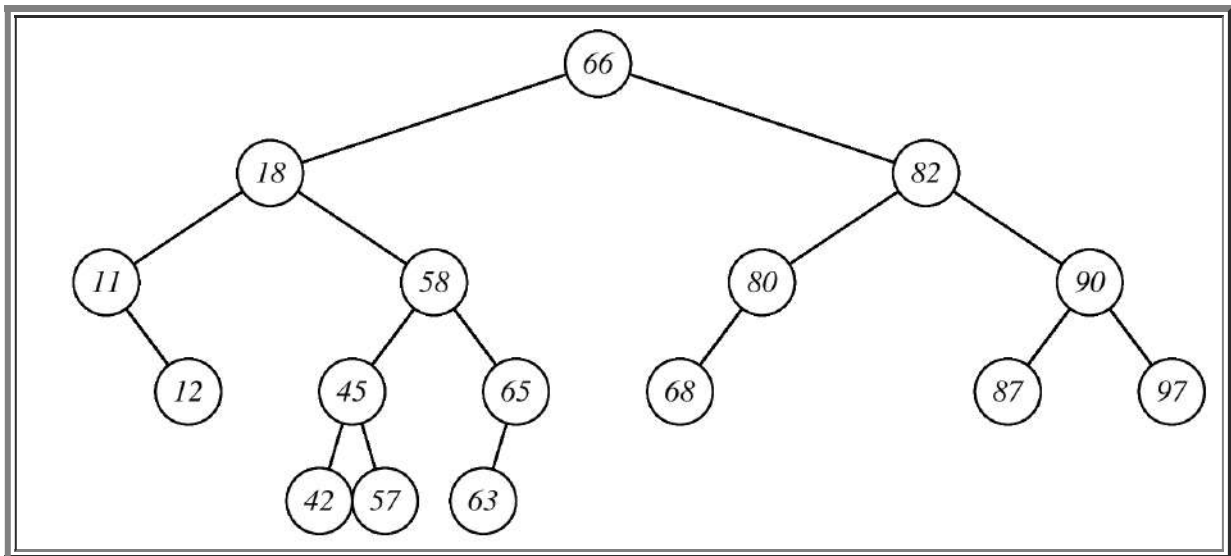


Question 7: When I delete 94, I replace it with node 97. The result is:

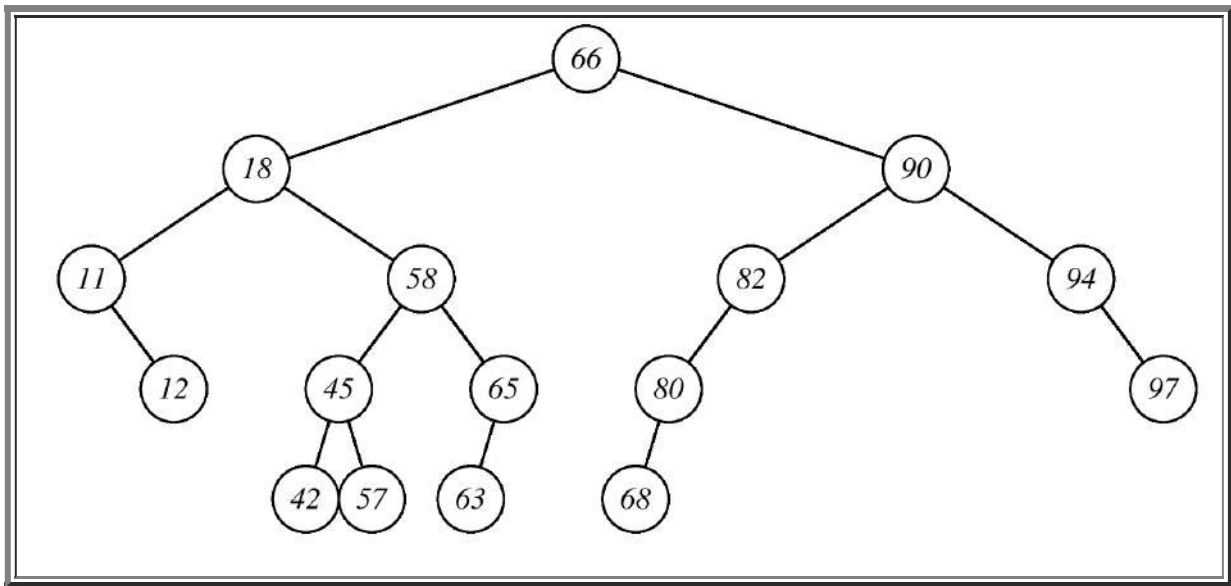


Node 90 is imbalanced, and the imbalance goes left-left. So it's a zigzig.

Question 8: To fix, I rotate once about the child of the the imbalanced node. That's node 82. Here's the result.

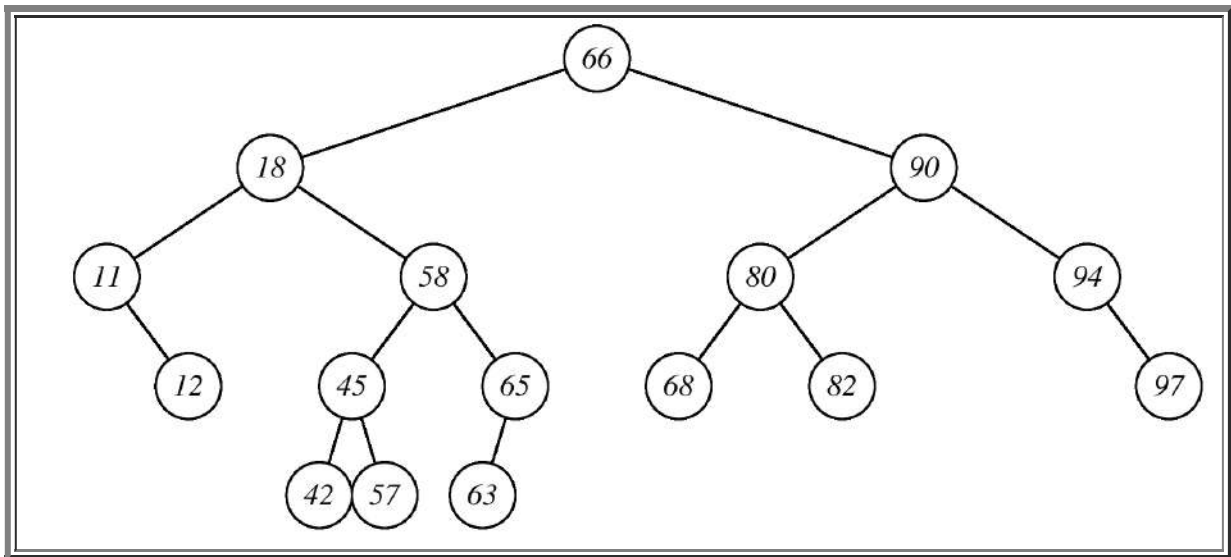


Question 9: When I delete 87, I get:

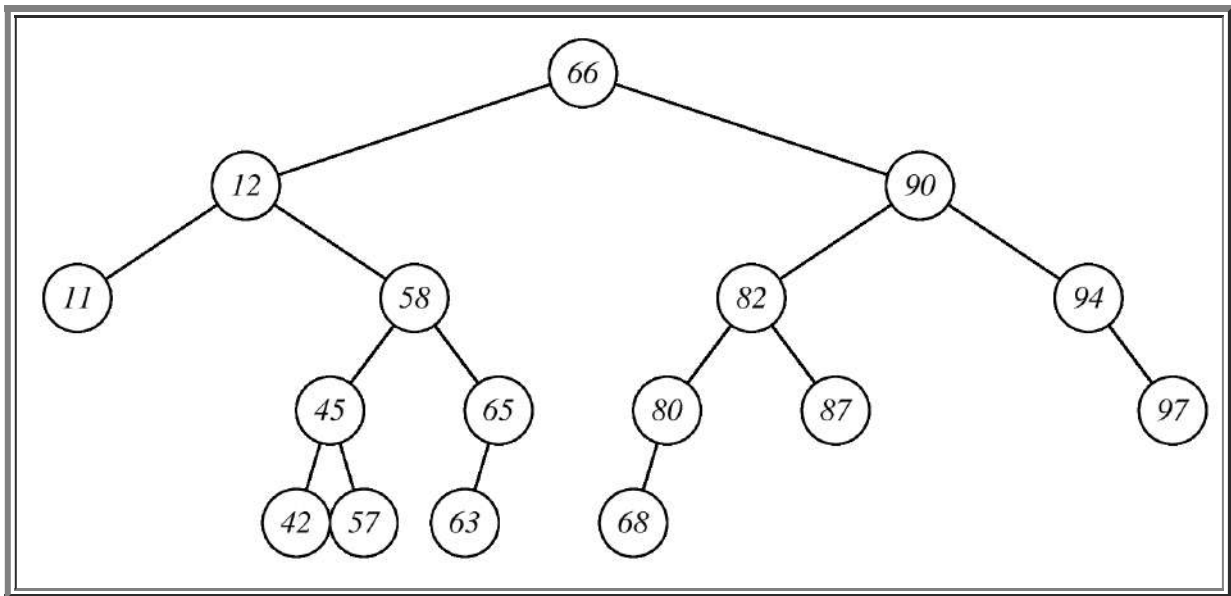


Node 82 is imbalanced, and the imbalance goes left-left. So it's a zigzig.

Question 10: To fix, I rotate once about the child of the the imbalanced node. That's node 80. Here's the result.

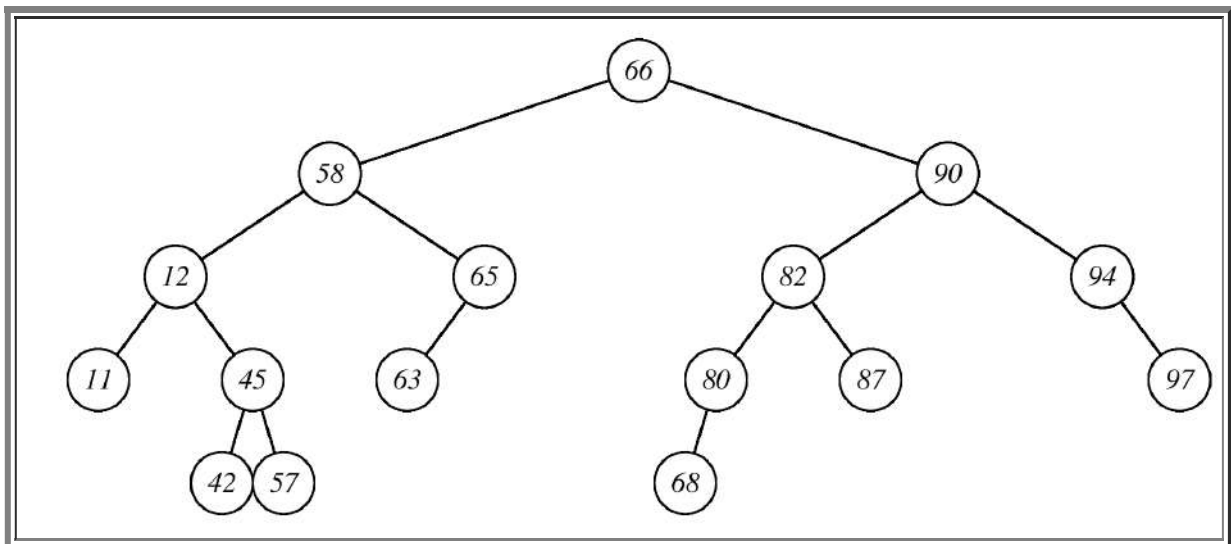


Question 11: When I delete 18, I replace it with the largest node in its left subtree -- 12 -- and then delete node 12. Here's the result:



Node 12 is imbalanced, and this is a case where the two subtrees of the child (58) are the same height. You treat this as a zigzig.

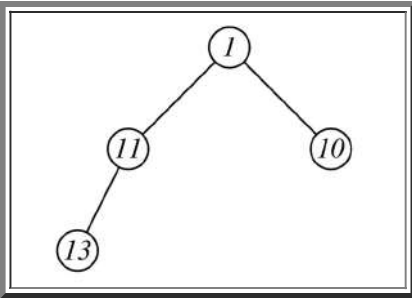
Question 12: To fix, I rotate once about the child of the the imbalanced node. That's node 58. Here's the result.



Clicker Questions

Question 1:

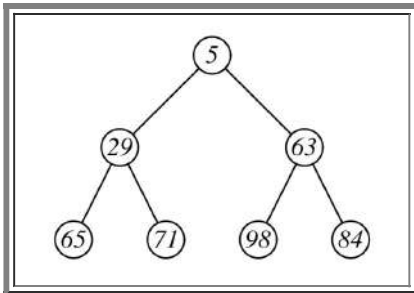
Is the tree to the right a binary heap? Please answer "Yes" or "No".



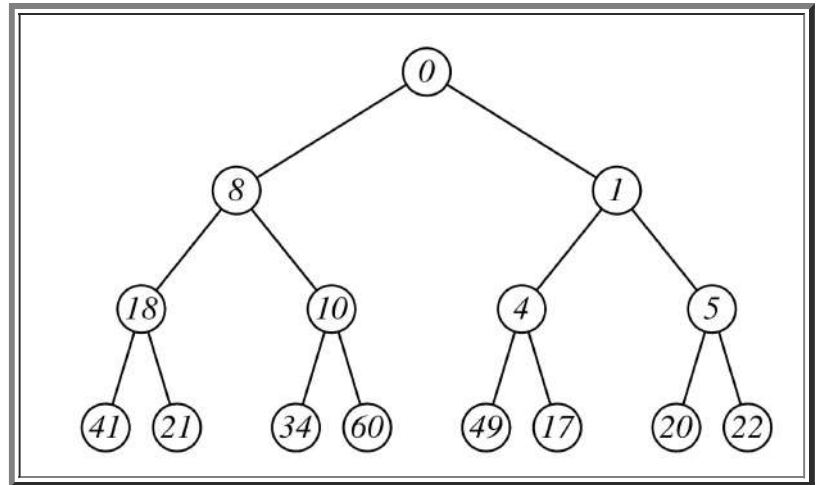
Question 2: When the following binary heap is stored in a vector, what is the index of the number 65?

Question 3: When the following binary heap is stored in a vector, what is the index of the number 84?

Question 4: If the value 33 is pushed onto the heap, in what index of the vector will it be stored?



Question 5: In the following heap, we call **Pop()**. What will be the index of the value 22 in the vector version of the heap, when **Pop()** is done?



Answers

If any of these are unclear, please go over the lecture notes for heaps.

- **Question 1:** Yes.
- **Question 2:** 3.
- **Question 3:** 6.
- **Question 4:** 3.
- **Question 5:** 12.