What is this output from program:

#include <iostream>

using namespace std;

```
int main(int argc, const char * argv[]) {
    int i{6}, j{7}, k{8}, x{13};

    if ( i<j)
        cout << "I'm here!\n";
    if ( i<j  &&  j>x  &&  k>i )
        cout << "Not here.\n";
    if ( i==j  ||  j==x  ||  k!=i )
        cout << "And Here too.\n";
    if ( i==j  ||  j<x  &&  k>i )
        cout << "What about me!\n";
    if ( i<j  ||  j>x  &&  k<i )
        cout << "and me!\n";
    if ( i==j  ||  !(j<x  &&  k>i) )
        cout << "And this!\n";

    retrn 0;

}
```

Your Answer:

I'm here!

And Here too.

What about me!

and me!

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(int argc, const char * argv[]) {
    cout << "Samheeta Mistry \t\t CIST004A \n";
    cout << setw(10) << "Number" <<
    setw(10) << "Number*2" <<
    setw(10) << "Number^2" << '\n';

    for (int ct = 1; ct < 16;) {
        cout << setw(10) << ct <<
        setw(10) << ct*2 <<
        setw(10) << ct*ct << '\n';
        ct += 1;
    }
    return 0;
}
```

Prompt the user to enter a number between 1 and 100 inclusive. Accept input of only a valid integer and compute the factorial of that number.

```
main() {
int num;
do {
      cout << “Enter a number between
      1 and 100, inclusive: “;
      cin >> num;
} while (num < 1 || num > 100);
for (int i = num-1; i >= 1; - - i) {
      num *= i;
}
cout << “The factorial is “ << num;
```

Write code to accept from the user a valid integer between 20 and 47, inclusive

```
int number;
do
{
    cout << "Enter a number between 20 and 47 inclusive: ";
    cin >> number;
    if (number < 20 || number > 47)
        cout << "Invalid entry, try again." << endl;
} while (number < 20 || number > 47);

cout << "The number is " << number << ".\n";
```

Write code to count by 2s from 50 to 1000

```
for ( int i{50}; i<= 1000; i+=2)
      cout << "The number is " << i << endl;
```

```
The number is 50
The number is 52
The number is 54
...
The number is 998
The number is 1000
Program ended with exit code: 0
```

Write code to roll 2 dice (9-sided) and give the sum

```
sum = (1 + rand()%9) + (1 + rand()%9);
cout << "The sum of 2 dice is " << sum << endl;
```

First Dice   Second Dice

Write code to roll 2 dice (11-sided) 10 times and give the sum each time

```
int sum;
for ( int i = 1 ; i <=10; i++ ) {
    sum = (1 + rand()%11) +(1 + rand()%11);
    cout << "The sum of 2 dice is " << sum << endl;
}
```

```
The sum of 2 dice is 5
The sum of 2 dice is 13
The sum of 2 dice is 15
The sum of 2 dice is 8
The sum of 2 dice is 10
The sum of 2 dice is 19
The sum of 2 dice is 2
The sum of 2 dice is 22
The sum of 2 dice is 13
The sum of 2 dice is 11
Program ended with exit code: 0
```

float/double/ long double x; /* block comment: Floating point types…Imprecise */
<**unsigned**> **int/long int/long long int** x; // line comment: Integer types
char **x;** /*8-bot character */ **bool** **x;** /***True** or **False** */ **void** // Nothing as a return type
Promotion: Larger, up the list. Coercion make type smaller, down the list.
Computer only does math / compares on SAME type objects! Use **static_cast**<type>(num) to force

Data scopes as: **Global** (outside any function..seen by all functions),
block (inside **{ . . . }**…seen only inside **{. . . }**), (Remember scope in **for** loops too!)
function parameters and defined variable seen only in function.
Use **::varname** to get to global variables! **using namespace std;** //**std::cout** . . . simplified

math: **( )** first, then ***** **/** **%** left to right. Then **+** **-** left to right as you hit them.
X **=** X **+** 1; X **+=** 1; //(+ - / * % operators only)
++X (increment then use); X**++** (Use then increment) (++ --)
= is assignment operator. Values from right (r-, return- or result- value), store left (l-or location- value)

if (***condition***) { // True code } **else** { // False code }
switch (*value*) { **case 1:** ; **case 2:** **;** **default:** }
break; //-> Get you out of **switch** or any **loop**

do { // first time and while True } **while** (***condition***);
while (***condition***) { // True code }
for (Initialization Phase ; ***condition*** ; Post Phase) {// True code }
//Initialization Phase done once, at start
//Post phase done after the ending ‘ }’ is hit for the **for** loop
continue; //Continues the any loop. No impact on **switch** or **if**…**else**

{ . . . } optional. Counts as only 1 Statement!! KNOW where they go if missing

(a **==** b) // **==** Equal **!=** Not Equal **<=** Less or Equal **>=** Greater or Equal
! - Not (True->False | False ->True) a= !a; // Flip a from True to False or False to True
&& - **and** (both sides True, Stop first False) **||** - **or** (either side True, Stop first True)

Functions defined or prototyped before usage
returnType functionName (type par1, type par2=2, type par3=’a’) { . . . } <= Definition
returnType functionName (type par1, type par2, type par3); <= Prototype
functionName (type, type, type)<= Signature: Name & parameters types in order. , used to match Functions
Function get COPIES of the calling parameters.
myFun (int a, int b); Get copies of the <u>values</u> of the ints. They disappear (Poof) at exit
myFun (int **&** a, int **&** b); Note: ‘&’; Get references to the ints (Addresses of) Originals can be modified
(offset + rand() % scale) -> Random Number from a low of offset through (offset + scale-1))

formatting: Special Char **\n** newline, **\t** tab; ‘\\’ is \ ‘\’ is an escape character…next one is special
cin >>variable; // get the data from console of the type specified by the variable
cout <<variable << “string \n”; // print out the variable, or string
fixed, **scientific** only apply to floating point numbers! **setprecision**(#) used only on these 2