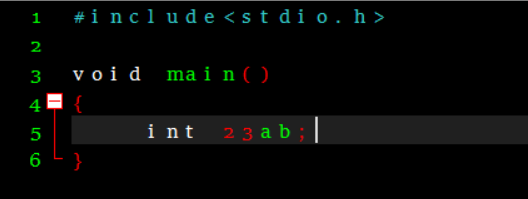
Aum Amriteswaryai Namah

**19CSE205 – Program Reasoning – Assignment – 2**

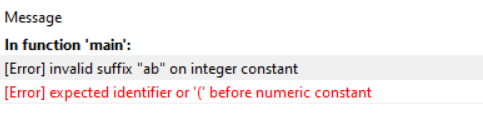
S. Abhishek

AM.EN.U4CSE19147

1. Write a C program which should include the following statement, compile the program and write down the error you get. Ensure no other errors are in the program before trying this.
   * int 23ab;

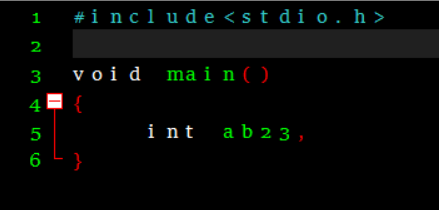


Error :

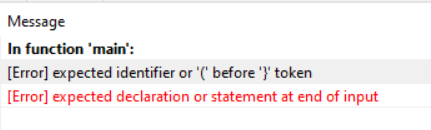


Here the declared string is not recognized as a Identifier and this comes under lexical Correctness. Finite state automation doesn’t recognize it as a Identifier and this throws an Error.

1. Now change the statement as follows. Note there is a comma instead of semicolon at the end. Compile and write down the error you get.
   * int ab23,



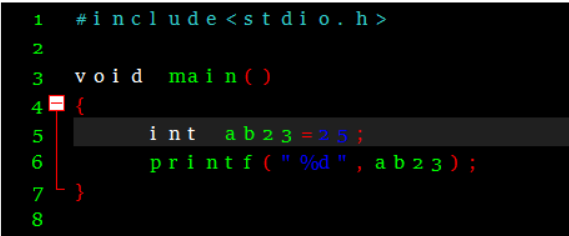
Error :



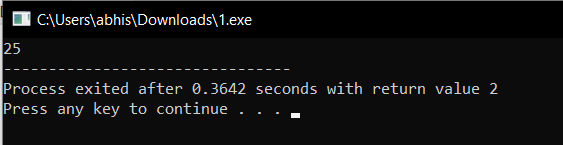
1. Replace comma by semicolon.
   * int ab23;

Add the following statements below the declaration, **one at a time**, compile-run the program and note down your observations.

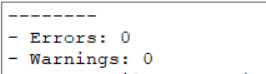
* 1. ab23 = 25;



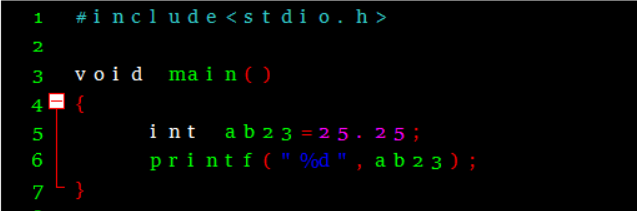
Output:



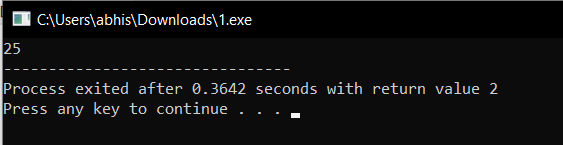
Error:



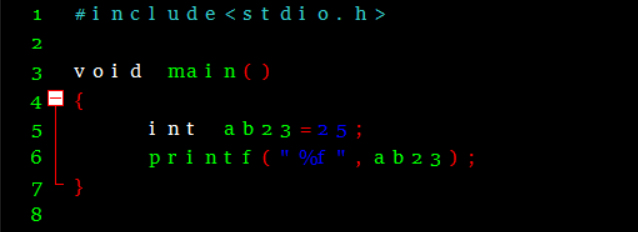
* 1. ab23 = 25.25;



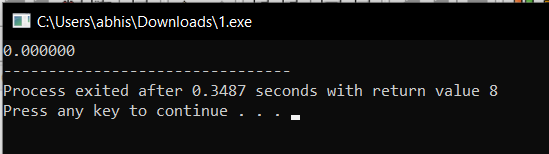
Output :



Here it is automatically Type casted into Int type by the compiler.

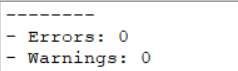


Output:

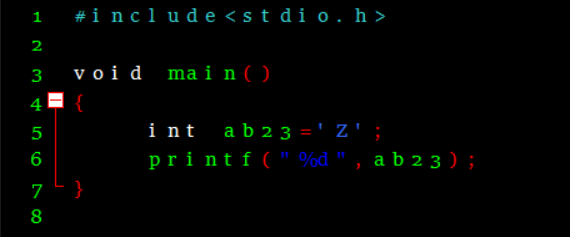


Here the Identifier is of type Int and we are trying to print it in Float. Here to avoid the Error the data type should be either Float or the value assigned to the identifier must be a Floating Point.

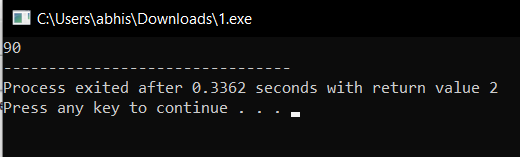
Error:



* 1. ab23 = ‘Z’;



Output:

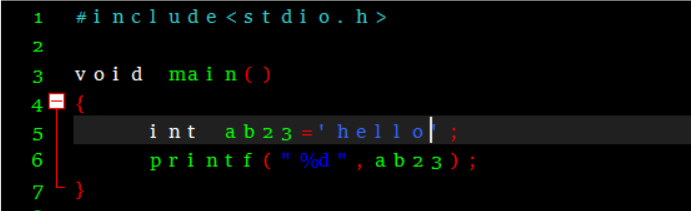


Here the value is automatically Converted into ASCII and it avoids from prompting Error.

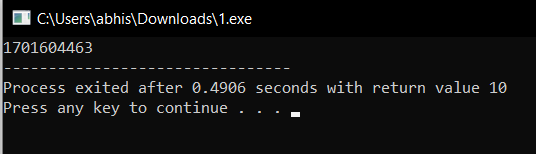
Error:



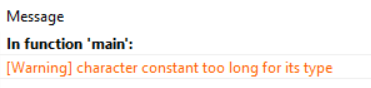
* 1. ab23 = “hello”;



Output :

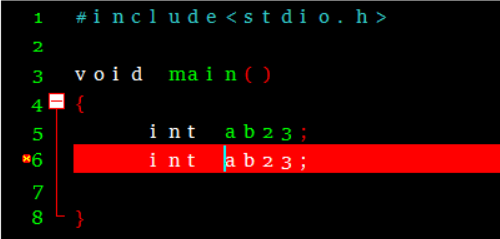


Error :

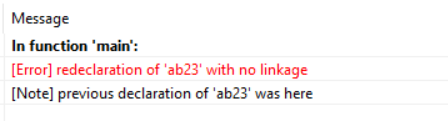


Here the String is assigned to a INT type Identifier.

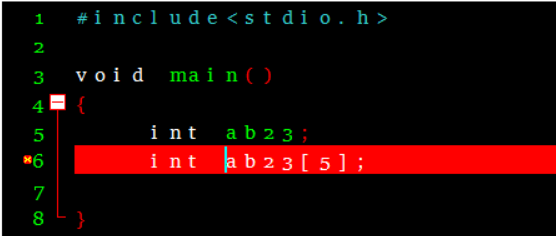
* 1. int ab23; // i.e. another declaration



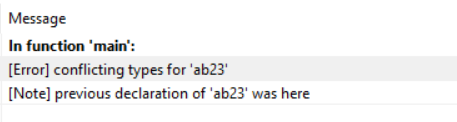
Error :



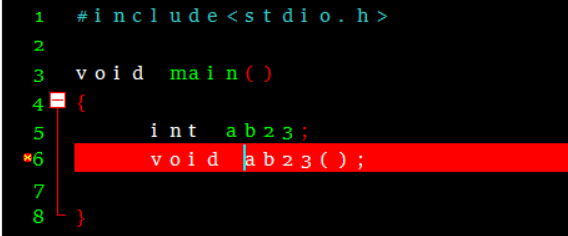
* 1. int ab23[5]; // i.e. declare array variable with same name



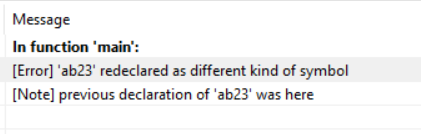
Error:



* 1. void ab23(); // i.e. declare function prototype with same name .

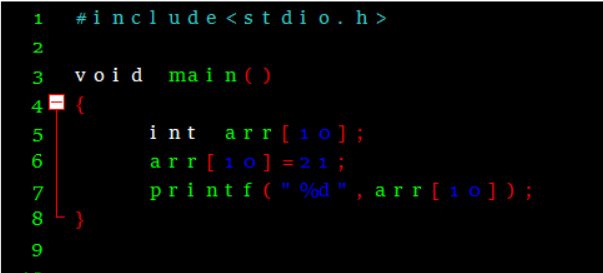


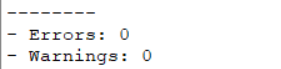
Error :

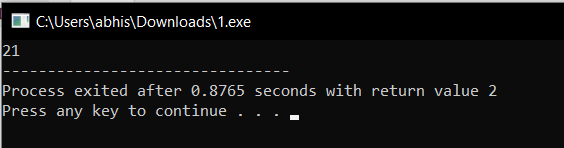


1. Declare an array as follows and assign a value.
   1. int arr[10]; arr[10] = 21;

Does it produce compiler error or runtime error? What type of error it is?







Sometimes it causes Index out of Bound Error or Segmentation Fault.

1. Write a basic program with pointers as directed below.
   1. Declare a pointer to an integer variable ptr.

int \* ptr;

* 1. Use malloc to dynamically allocate memory for ptr.

ptr = (int \*) malloc( sizeof(int) );

* 1. Assign an integer value to the memory pointed to by ptr.

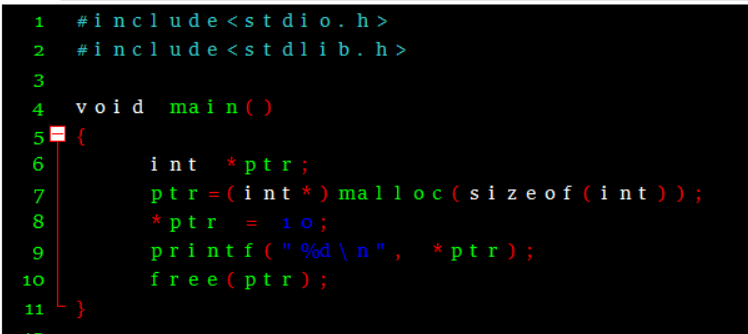
\*ptr = 10;

* 1. Print the value pointed to by ptr to the terminal.

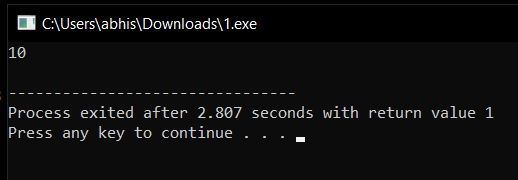
printf(“%d\n”, \*ptr);

* 1. Free the ptr memory.

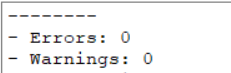
free(ptr);



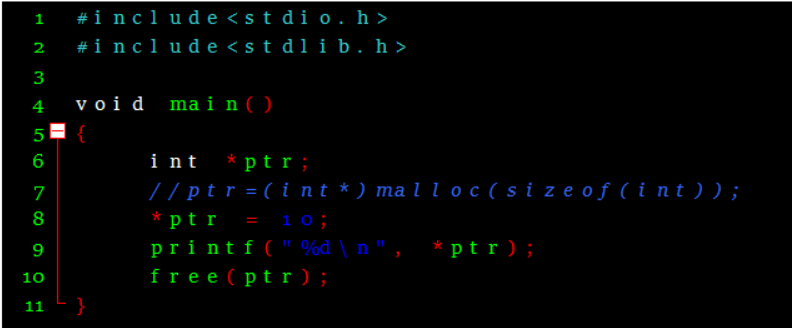
Output :



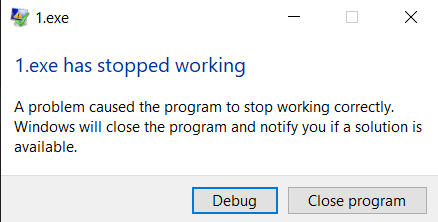
Error :



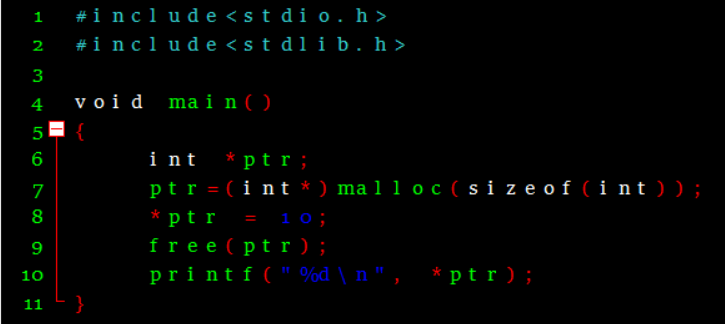
1. **Perils of pointers**: Variations of Qn 5 to simulate the problems due to mishandling of pointers. All these are semantic errors.
   1. **A case of null pointer**: Access value without allocating the memory.
      1. Perform steps a and c (i.e. without b).
      2. Note down the error.



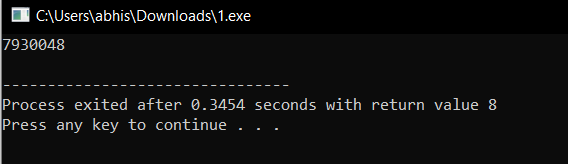
Error :



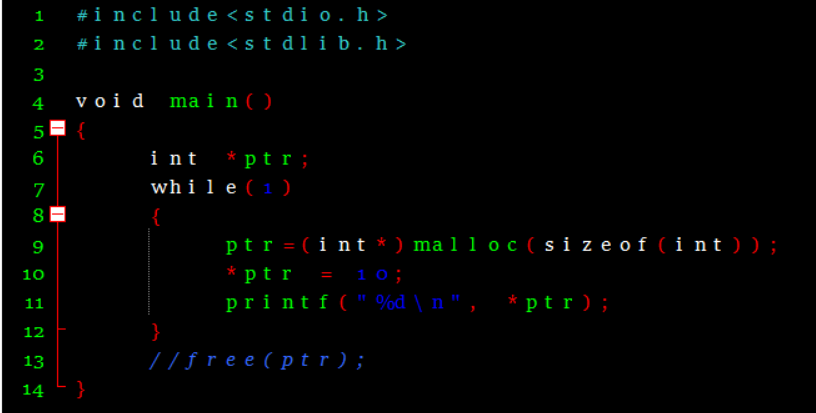
* 1. **Another case of null pointer**: Access value after freeing the memory
     1. Perform steps a, b, c, e and then d.
     2. Note down the error.



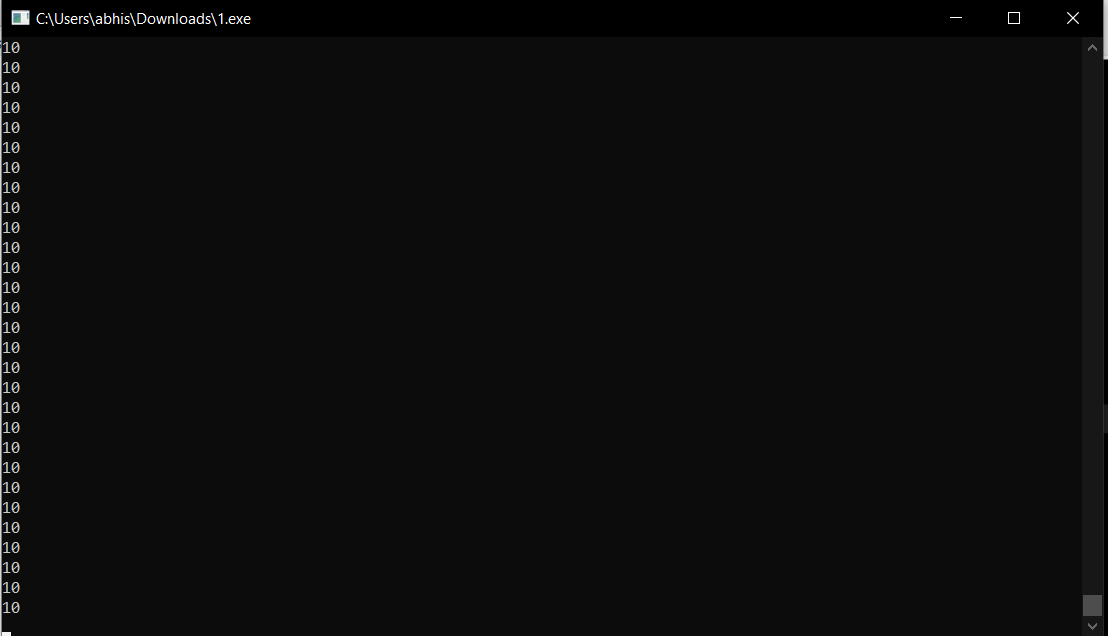
Output :



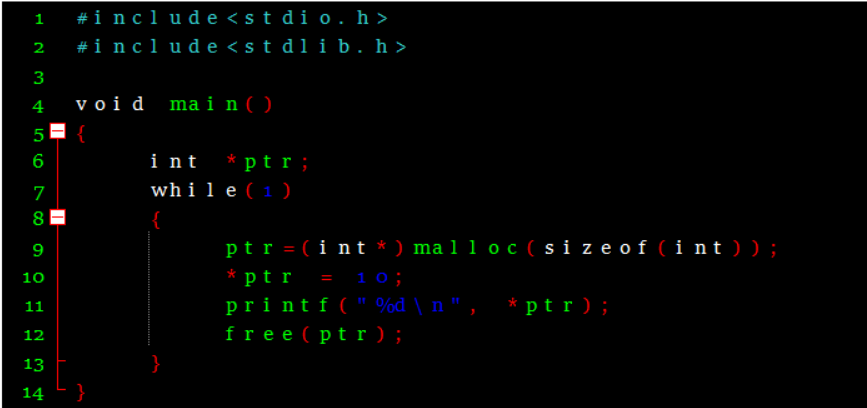
* 1. **A case of memory leak**: Allocating memory without freeing. (Try this last after answering all questions, since system will gradually slow down and eventually crash.)
     1. In an infinite loop, do steps a, b, c and d (i.e. without e).
     2. Run the program for as long as it can.
     3. The program + system will crash after some time. Restart your computer.



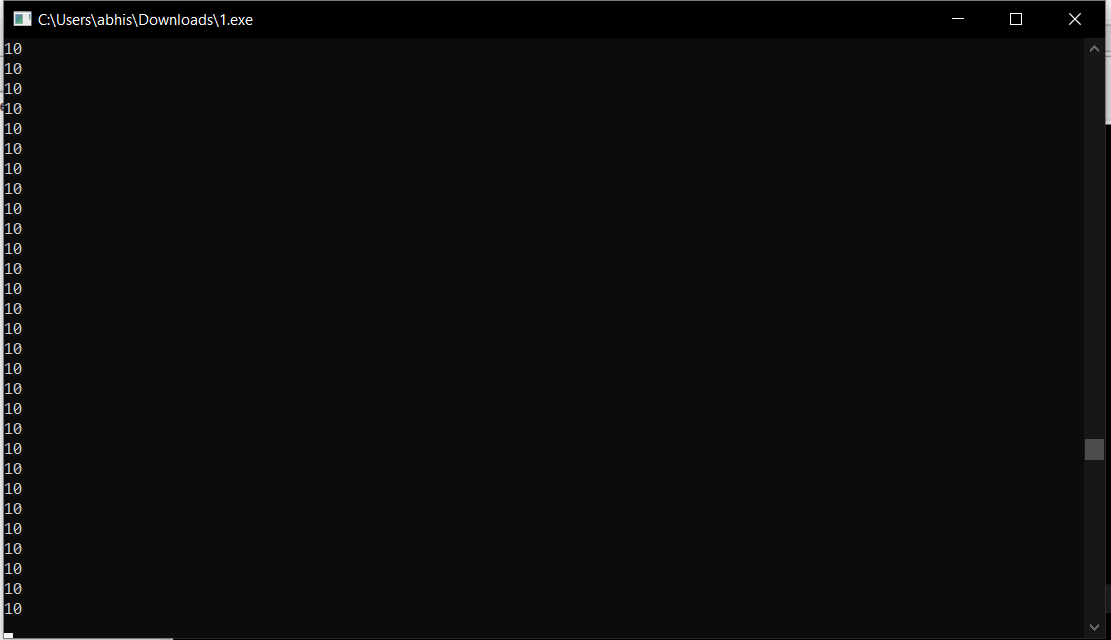
Output :



* 1. **A case of not allowing memory leak**: Allocating memory with freeing
     1. In an infinite loop, do steps a, b, c, d and e.
     2. The program should run forever without crashing.
     3. Press CTRL+C to stop execution.



Output:



* 1. **A case of lost pointer and memory leak**: Re-assigning a pointer to another location

int \* p = (int \*) malloc( sizeof(int) ); // p points to a location\_1 in memory

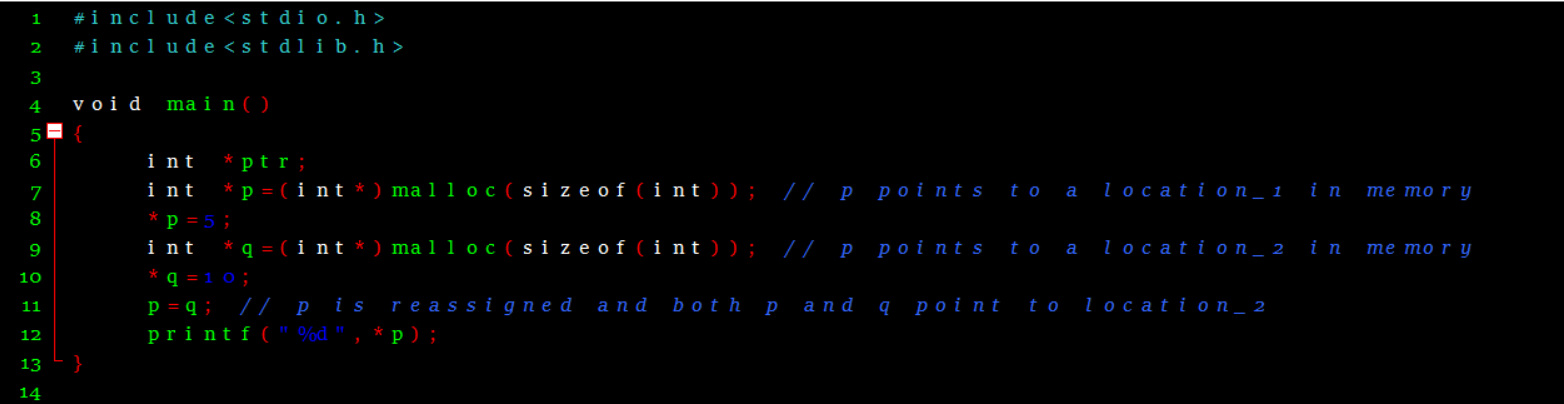
\*p = 5;

int \* q = (int \*) malloc( sizeof(int) ); // p points to a location\_2 in memory

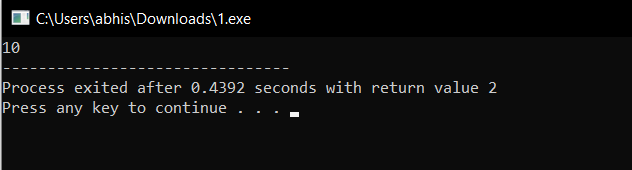
\*q = 10;

p = q; // p is reassigned and both p and q point to location\_2

* + 1. The access to location\_1 is lost. It is impossible to retrieve the value 5.
    2. It can’t be freed either since the pointer is lost. This leads to memory leak.



Output:



* 1. **Another case of lost pointer**: Re-assigning a pointer to a new memory

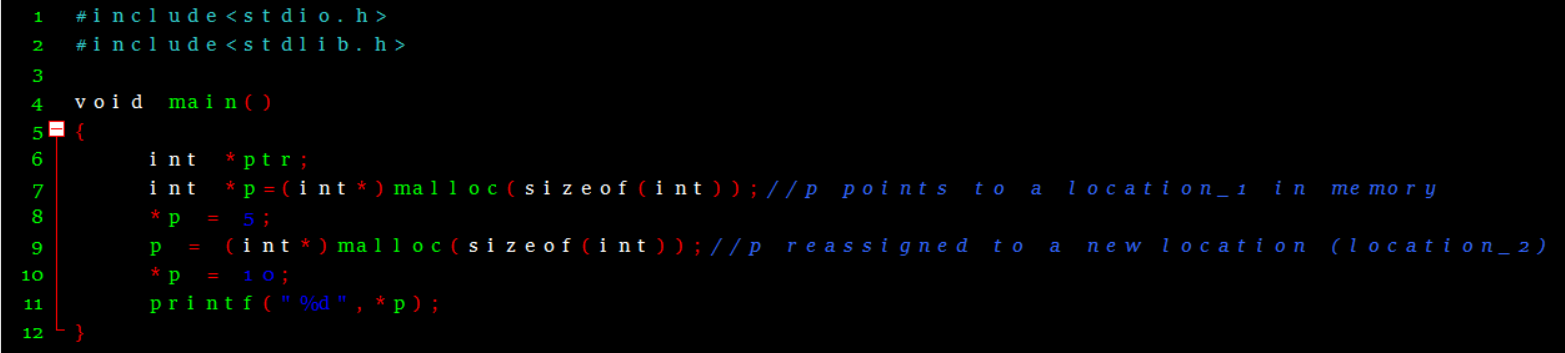
int \* p = (int \*) malloc( sizeof(int) ); // p points to a location\_1 in memory

\*p = 5;

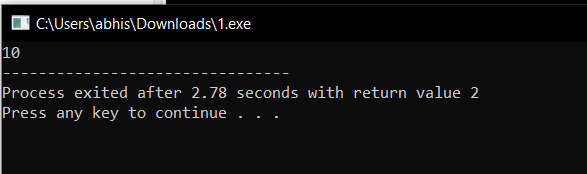
p = (int \*) malloc( sizeof(int) ); // p reassigned to a new location (location\_2)

\*p = 10;

* + 1. The access to location\_1 is lost. It is impossible to retrieve the value 5.
    2. It can’t be freed either since the pointer is lost. This leads to memory leak.



Output:



* 1. **A case of dangling pointer**: Freeing up a pointer when another is accessing the same location.

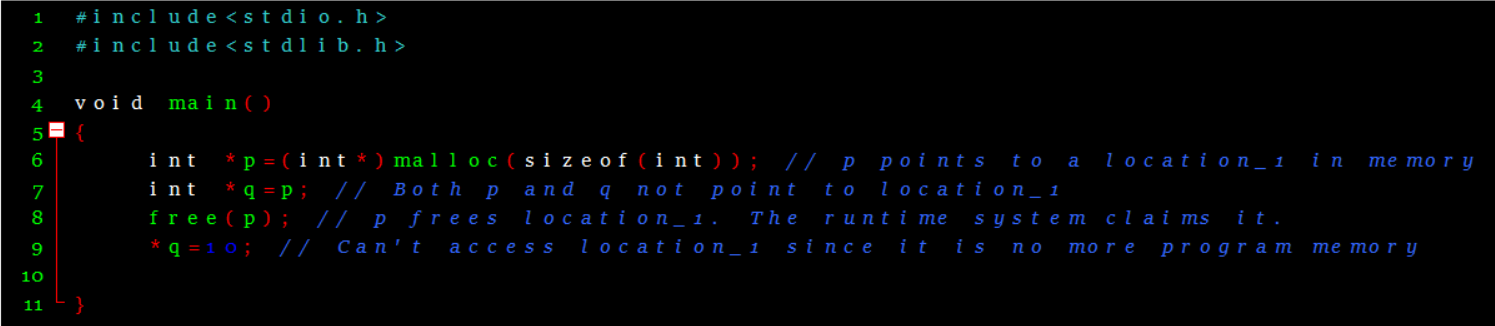
int \* p = (int \*) malloc( sizeof(int) ); // p points to a location\_1 in memory

int \* q = p; // Both p and q not point to location\_1

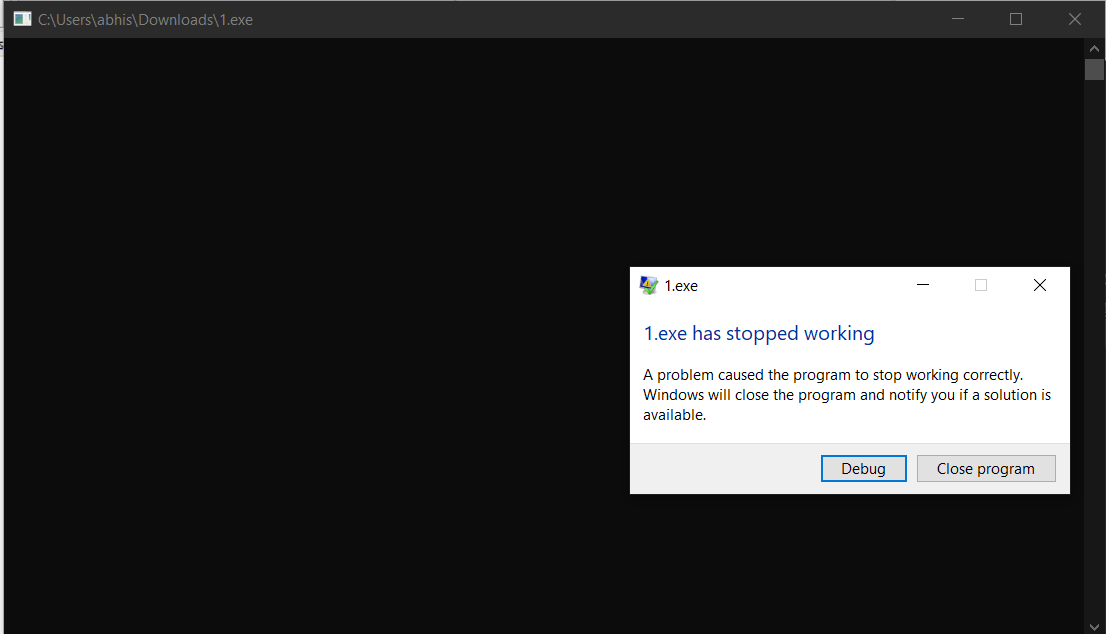
free(p); // p frees location\_1. The runtime system claims it.

\*q = 10; // Can’t access location\_1 since it is no more program memory

* + 1. In short, q points to a location which does not legally belong to the program
    2. Note down the error



Output:

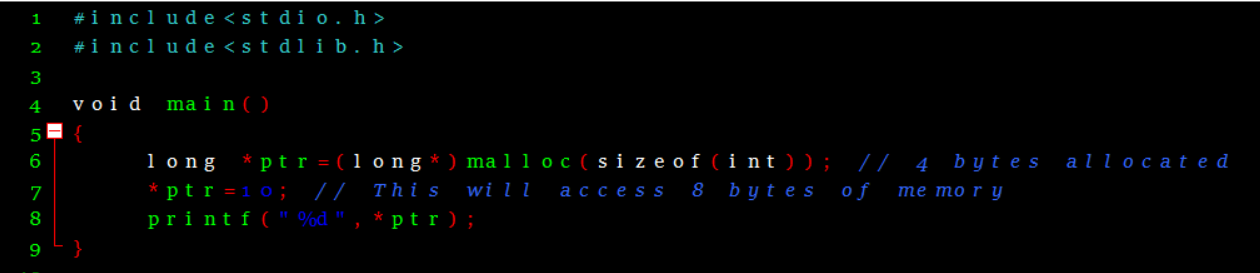


* 1. **A case of messing up with pointer**: incorrect type casting leads

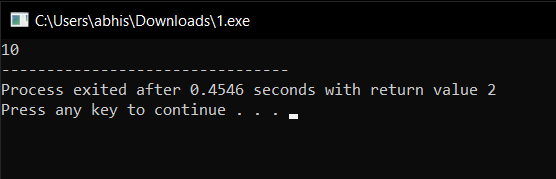
long \* ptr = (long \*) malloc( sizeof(int) ); // 4 bytes allocated

\*ptr = 10; // This will access 8 bytes of memory

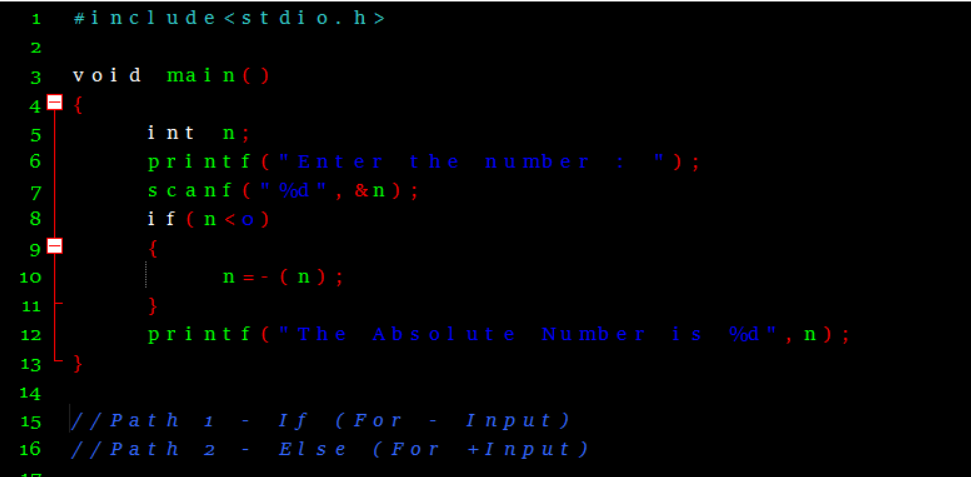
* + 1. Out of 8 bytes, only first 4 can be legally accessed.
    2. The runtime system will report an error when trying to assign value 10.
    3. Note down the error



Output:

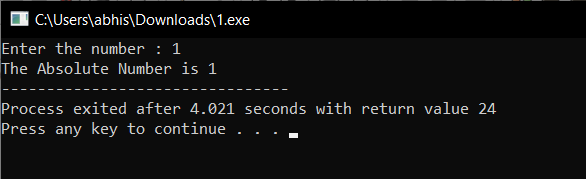


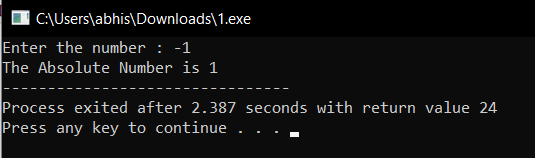
1. Write a program for computing absolute value of an integer. abs(x) = x if x is positive, -x, if x is negative. Don’t use the library call. How many test cases are required to check each execution path the program can take?



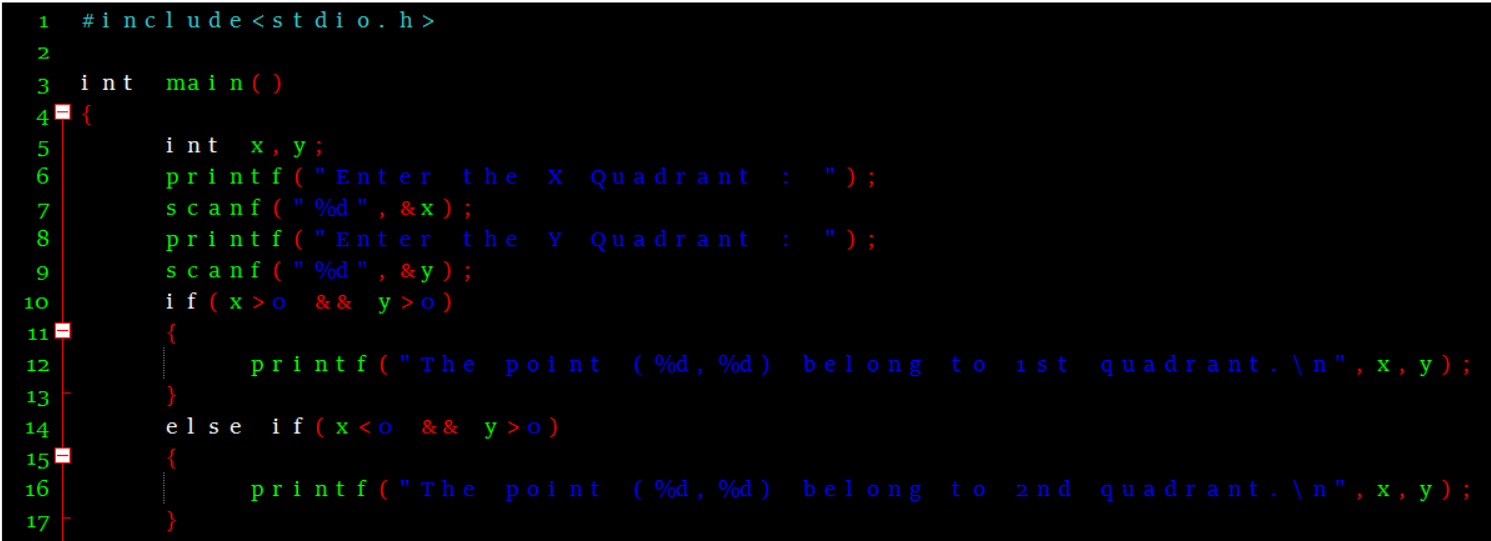
* Two test cases are required to Check its execution Path.
* One test case is to check the if condition and the other one is to check the else condition.
* Here we have given the first input as positive so it goes to the path 2.
* The second input is -1 so it goes to the path 1.
* In this program two test cases are necessary to check its correctness.

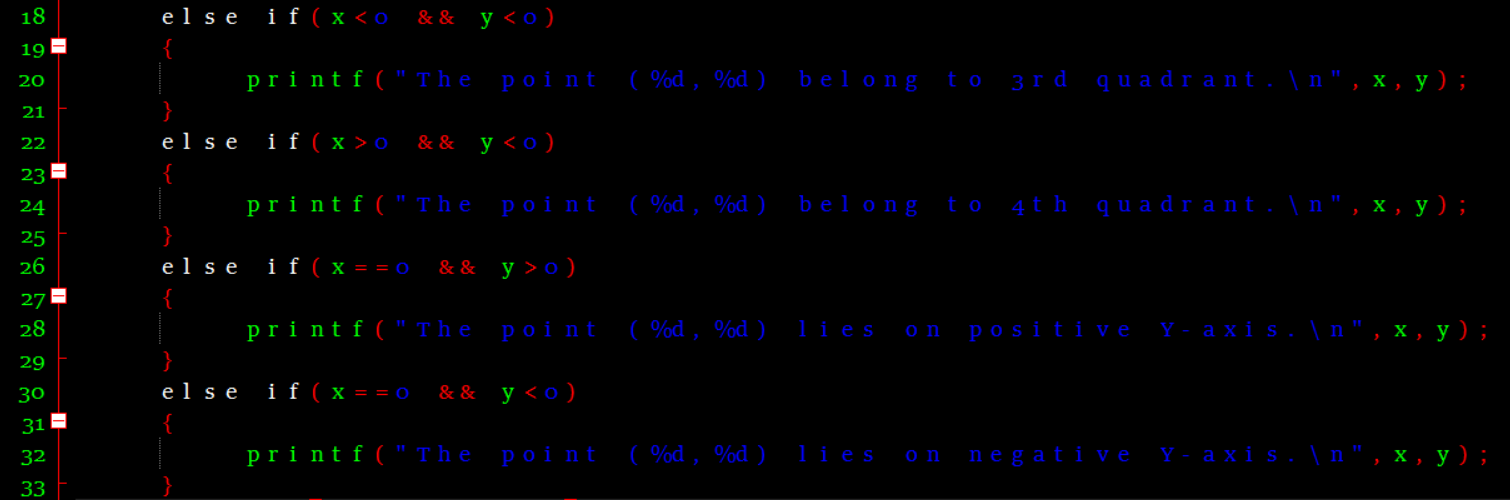
Output:

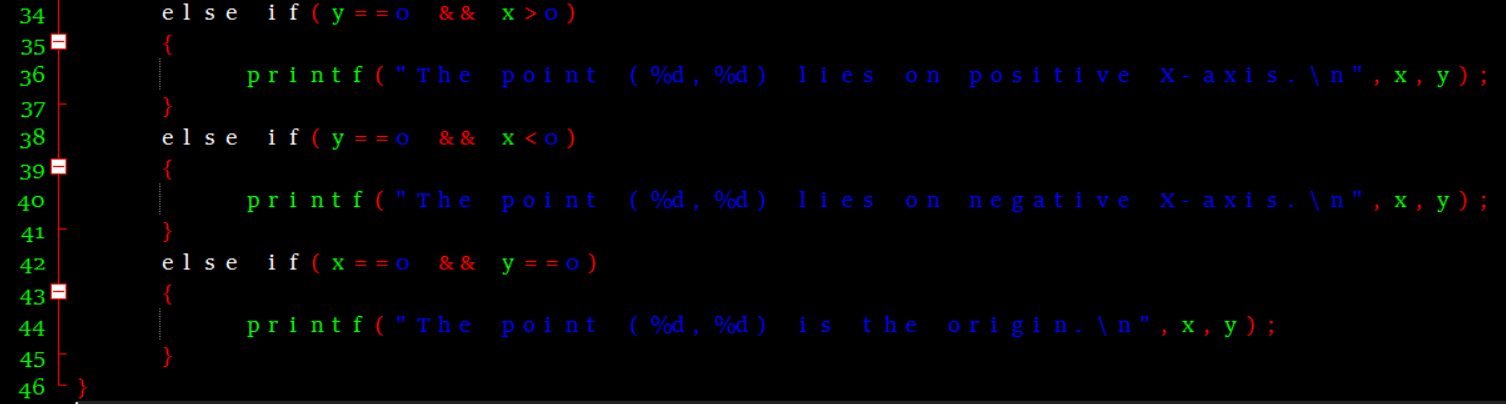




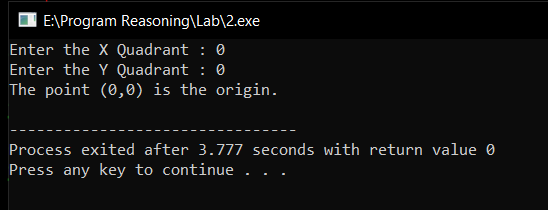
1. Write a program that takes x and y as input, and prints which quadrant (x,y) belongs to. For instance, given (4,-3) as input, it should print Q4. Provide exhaustive set of test cases that will ensure all execution paths are covered.

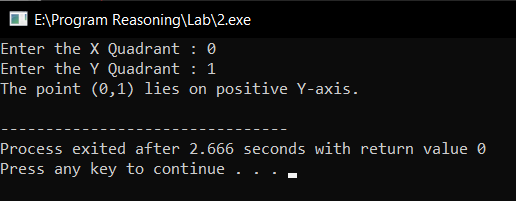


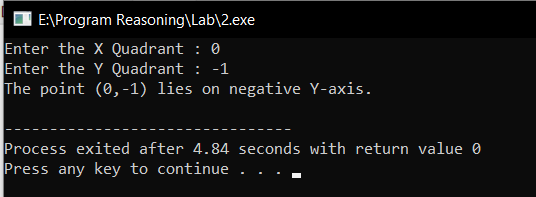


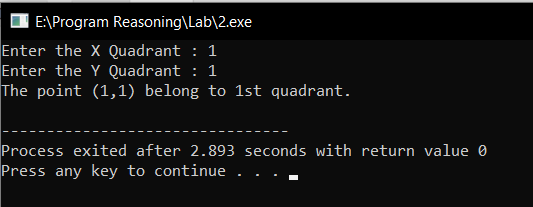


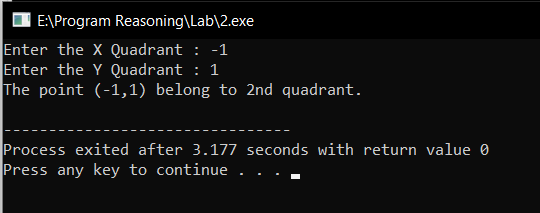
Test Cases:

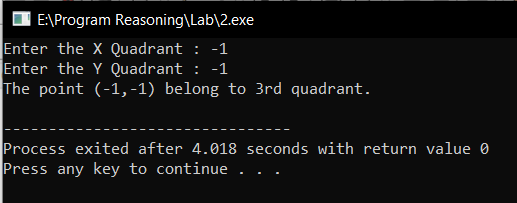


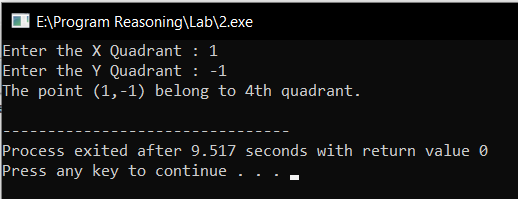


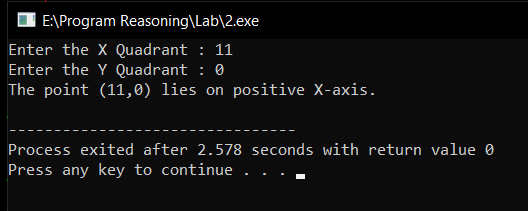


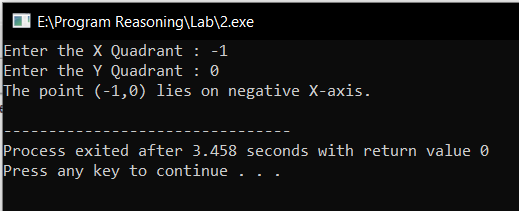












* Nine test cases are required to Check its correction including Origin since there are 4 Quadrants with (1,1) , (1,-1),(-1,-1),(-1,1) and four other cases respective to X as Origin or Y as Origin or Both X and Y as Origin (0,0),(1,0),(0,1),(-1,0),(0,-1).

Thank You!!