

# Programming Errors in C

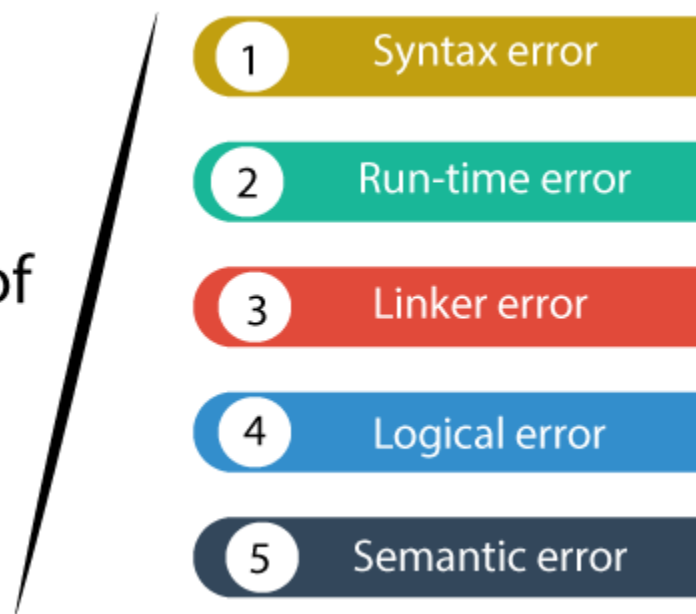
Errors are the problems or the faults that occur in the program, which makes the behavior of the program abnormal, and experienced developers can also make these faults. Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as **debugging**.

These errors are detected either during the time of compilation or execution. Thus, the errors must be removed from the program for the successful execution of the program.

**There are mainly five types of errors exist in C programming:**

- **Syntax error**
- **Run-time error**
- **Linker error**
- **Logical error**
- **Semantic error**

Types of errors



## Syntax error

Syntax errors are also known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers. These errors are mainly occurred due to the mistakes while typing or do not follow the

syntax of the specified programming language. These mistakes are generally made by beginners only because they are new to the language. These errors can be easily debugged or corrected.

### For example:

1. If we want to declare the variable of type integer,
2. `int a; // this is the correct form`
3. `Int a; // this is an incorrect form.`

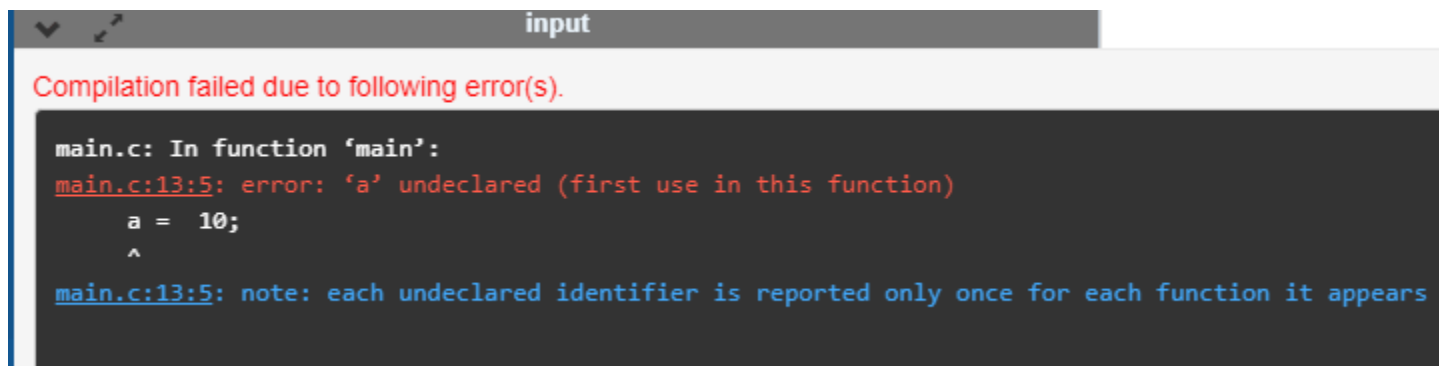
Commonly occurred syntax errors are:

- If we miss the parenthesis (}) while writing the code.
- Displaying the value of a variable without its declaration.
- If we miss the semicolon (;) at the end of the statement.

### Let's understand through an example.

1. `#include <stdio.h>`
2. `int main()`
3. `{`
4. `a = 10;`
5. `printf("The value of a is : %d", a);`
6. `return 0;`
7. `}`

### Output



```
input
Compilation failed due to following error(s).
main.c: In function 'main':
main.c:13:5: error: 'a' undeclared (first use in this function)
    a = 10;
    ^
main.c:13:5: note: each undeclared identifier is reported only once for each function it appears in
```

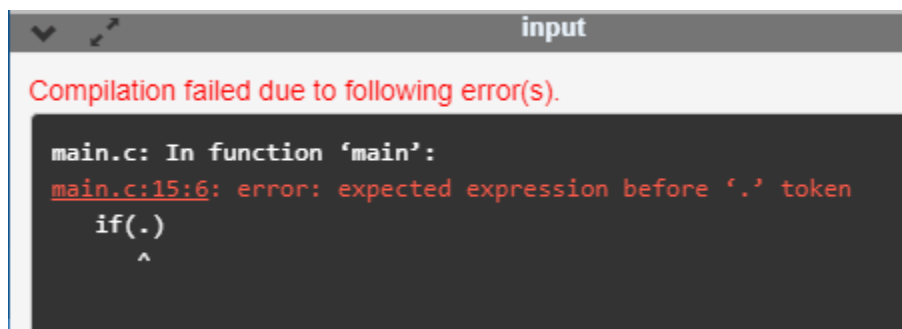
In the above output, we observe that the code throws the error that 'a' is undeclared. This error is nothing but the syntax error only.

There can be another possibility in which the syntax error can exist, i.e., if we make mistakes in the basic construct. Let's understand this scenario through an example.

1. `#include <stdio.h>`
2. `int main()`
3. `{`
4. `int a=2;`
5. `if(.) // syntax error`
6.
7. `printf("a is greater than 1");`
8. `return 0;`
9. `}`

In the above code, we put the (.) instead of condition in 'if', so this generates the syntax error as shown in the below screenshot.

## Output



```
input
Compilation failed due to following error(s).
main.c: In function 'main':
main.c:15:6: error: expected expression before '.' token
if(.)
  ^
```

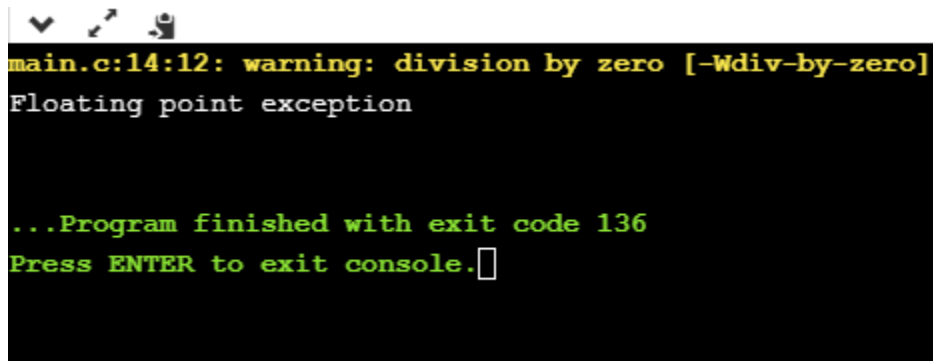
## Run-time error

Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors. When the program is running, and it is not able to perform the operation is the main cause of the run-time error. The division by zero is the common example of the run-time error. These errors are very difficult to find, as the compiler does not point to these errors.

**Let's understand through an example.**

1. `#include <stdio.h>`
2. `int main()`
3. `{`
4. `int a=2;`
5. `int b=2/0;`
6. `printf("The value of b is : %d", b);`
7. `return 0;`
8. `}`

## Output



```
main.c:14:12: warning: division by zero [-Wdiv-by-zero]
Floating point exception

...Program finished with exit code 136
Press ENTER to exit console.█
```

In the above output, we observe that the code shows the run-time error, i.e., division by zero.

## Linker error

Linker errors are mainly generated when the executable file of the program is not created. This can be happened either due to the wrong function prototyping or usage of the wrong header file. For example, the **main.c** file contains the **sub()** function whose declaration and definition is done in some other file such as **func.c**. During the compilation, the compiler finds the **sub()** function in **func.c** file, so it generates two object files, i.e., **main.o** and **func.o**. At the execution time, if the definition of **sub()** function is not found in the **func.o** file, then the linker error will be thrown. The most common linker error that occurs is that we use **Main()** instead of **main()**.

**Let's understand through a simple example.**

1. `#include <stdio.h>`
2. `int Main()`

```
3. {
4.   int a=78;
5.   printf("The value of a is : %d", a);
6.   return 0;
7. }
```

## Output

```
(.text+0x20): undefined reference to `main'
collect2: error: ld returned 1 exit status
```

## Logical error

The logical error is an error that leads to an undesired output. These errors produce the incorrect output, but they are error-free, known as logical errors. These types of mistakes are mainly done by beginners. The occurrence of these errors mainly depends upon the logical thinking of the developer. If the programmers sound logically good, then there will be fewer chances of these errors.

### Let's understand through an example.

```
1. #include <stdio.h>
2. int main()
3. {
4.   int sum=0; // variable initialization
5.   int k=1;
6.   for(int i=1;i<=10;i++); // logical error, as we put the semicolon after loop
7.   {
8.     sum=sum+k;
9.     k++;
10.  }
11. printf("The value of sum is %d", sum);
12. return 0;
13. }
```

## Output

```

The value of sum is 1

...Program finished with exit code 0
Press ENTER to exit console.

```

In the above code, we are trying to print the sum of 10 digits, but we got the wrong output as we put the semicolon (;) after the for loop, so the inner statements of the for loop will not execute. This produces the wrong output.

## Semantic error

Semantic errors are the errors that occurred when the statements are not understandable by the compiler.

The following can be the cases for the semantic error:

- Use of a un-initialized variable.  

```
int i;
i=i+2;
```
- Type compatibility  

```
int b = "javatpoint";
```
- Errors in expressions  

```
int a, b, c;
a+b = c;
```
- Array index out of bound  

```
int a[10];
a[10] = 34;
```

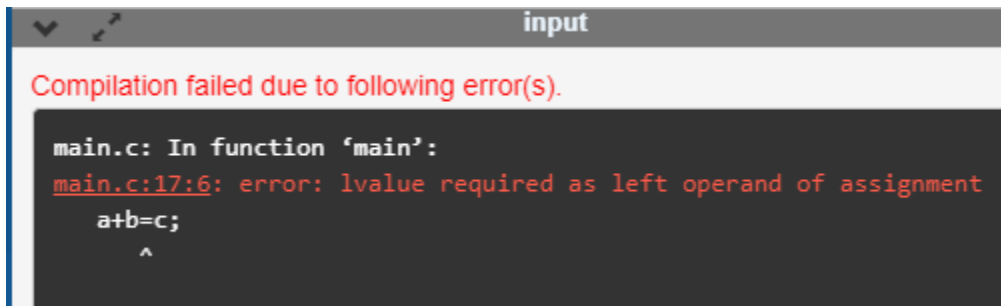
**Let's understand through an example.**

1. `#include <stdio.h>`
2. `int main()`
3. `{`
4. `int a,b,c;`

5. `a=2;`
6. `b=3;`
7. `c=1;`
8. `a+b=c; // semantic error`
9. `return 0;`
10. `}`

In the above code, we use the statement `a+b=c`, which is incorrect as we cannot use the two operands on the left-side.

## Output



```
input
Compilation failed due to following error(s).
main.c: In function 'main':
main.c:17:6: error: lvalue required as left operand of assignment
a+b=c;
  ^
```

