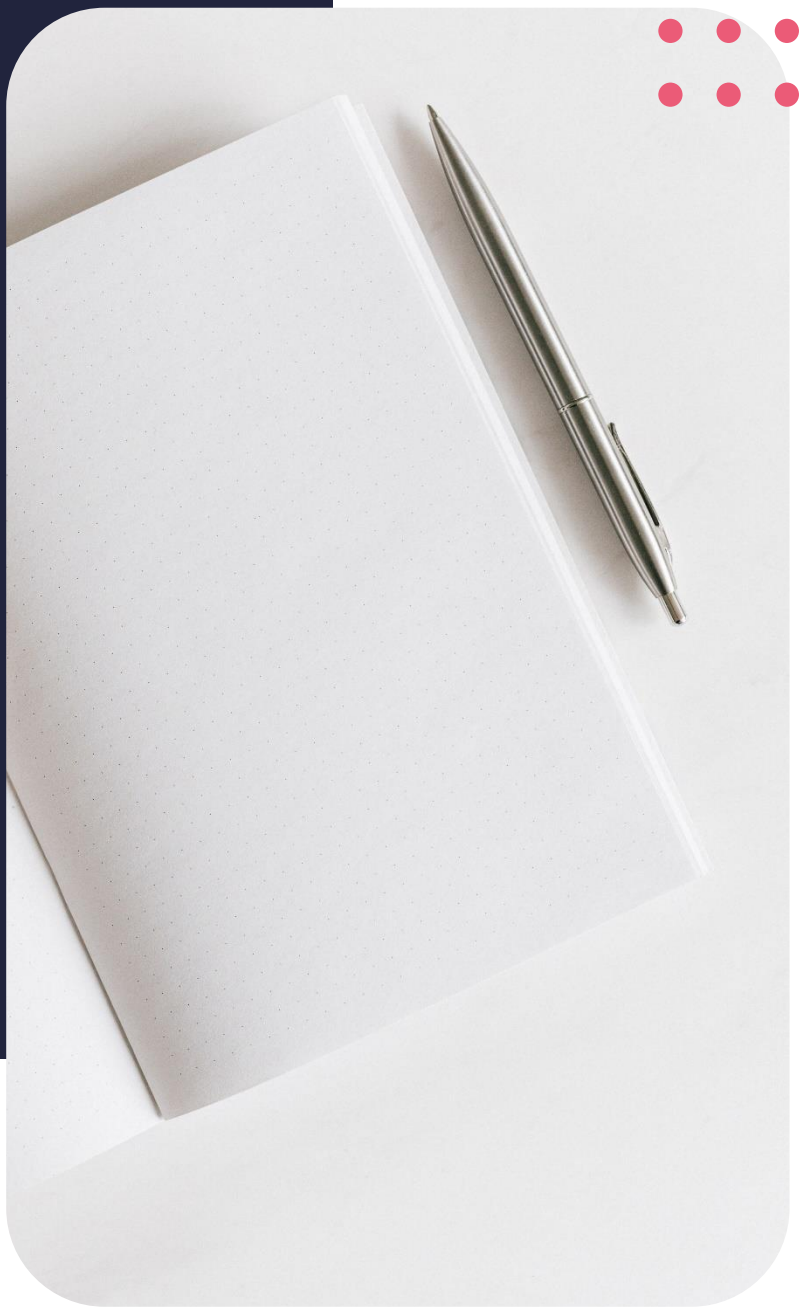
Diploma in Computer Science

Error Handling

Contents

Introduction	3
Programming errors	3
Errors, bugs and exceptions	3
Error handling methods	4
Error handling methods: prevention	4
Error handling methods: Termination	4
Error handling methods	5
Rules for using global error indicators	5
Errno error codes	5
How to ascertain which error handling method to use…	6
Moving on with our project	7
References	7



Lesson objectives

By the end of this lesson, you should be able to:

- Identify the types of errors in programming
- Explore error handling techniques
- Appreciate built-in ways of dealing with errors
- Design a new solution

Introduction

Welcome to the sixth lesson in our series. In this lesson, we build upon our previous lesson on bugs. Normally, a program should be able to deal with the input that it gets. If, for example, your program is dealing with the total amount of stock in a warehouse, then the computer would get digits with no decimals, but have you ever thought what would happen if the program got text instead? That is our focus today, as we look at how to prepare your code for unexpected input. We have used a rudimentary version of this in our previous demo on file handling, and now we are going to look at it in greater detail

Programming errors

In our last lesson, we looked at bugs as software eventualities that prevent it from working in the way the programmer intended. You may be wondering why we have yet another lesson on seemingly pretty much the same thing. Well, it is, in fact, a different realm of things, as we shall see shortly.

Errors, bugs and exceptions

This is a seemingly confusing topic, but in actual fact, it's pretty straightforward. Not all errors are bugs and just as well, not all bugs show up as errors. You might see these referred to in other programming languages as exceptions.

An exception is an unique event that prevents the code from working as intended. Take for example, if you're busy copying your birthday video to a flash drive, and your cat playfully pulls it out because it has a flashing red light. The computer can no longer proceed with copying the file, because the media to which the file is being written is not on the system, so the computer shows an error. This is not because the software is buggy, but something happened during the execution, which wasn't supposed to happen.

Errors, bugs and exceptions

A bug is simply a result of bad programming. The program itself is the problem here; it does something that's in not supposed to do. If you remember from our last lesson, things like a program that refuses to show a menu or performs incorrect calculations are examples of bugs.

In short, if you do not get the expected result as a result of the program, then it is a bug, but if it is a result of external factors, then it is an error.

That said, the program should be able to handle foreseeable errors, such as the user entering text in a field that should accept only integers. Back to our cat and flash drive example, the computer should present the user with a comprehensible error message, such as "the storage device is no longer available, please check that it is still connected" so that the user knows exactly what to do. It generally isn't good programming practice to present users with cryptic error messages that require a master's degree in wizardry to interpret.

Error handling

C does not provide any direct support for error handling. But all is not lost, as there are a few methods and variables that are defined in the error.h header file that can be used to point out errors using the return statement in a function. Most of

the error handling is left to the programmer so let's look at some of the methods that C offers the programmer.

Error handling methods

There are a number of ways for handling errors in C. These include prevention, termination, the global indication error, local error indicator, return values, assertions, signals, goto chains, non-local jumps, and error call-backs. We are going to look at some of these in a bit more detail but before we do that, a quick note…

Error handling is separated into two distinct groups, namely error detection and recovery. Detection refers to the program being able to tell if there is an error, and recovery refers to the program being able to recover from the error and not fall into a not so good state such as crashing.

Error handling methods: prevention

In prevention, we attempt to write the code in such a way that the errors don't happen in the first place. For example, if you know that the program is going to ask the user for a pin, you write the code in such a way that the user cannot enter anything other than digits. You have probably seen this in action on one of your devices. When you're supposed to just enter digits, the keyboard only shows those and so you won't have any way of entering anything else.

That said, there are other instances that make prevention impractical and reduce the functionality of the program, for example, the tan(x) function can overflow for values of x near n*$\pi\pi$/2, and n here can be any odd integer.

Error handling methods: Termination

Termination is a method of ascertaining the reason why a program exited. It is beautifully summarised in this table.

Each function is listed together with what it does when invoked. These functions are useful when the function determines that execution must not continue. This seems like a rather drastic measure, but if you think of what we did using files in one of our demos, it quickly becomes apparent. This should be used with care though, because when an application-independent function such as a library function can detect an error that does not mandate termination, it cannot recover from the error because it doesn't know how the application's error recovery policy. The function can only indicate that the error has occurred and leave the error recovery to some other function further up the call chain, which is when you, the programmer, come in. You need to provide a function that handles this error. This method of error handling is referred to as fail hard. This means that the program cannot continue and is terminated

Function	**Closes open streams**	**Flushes stream buffers**	**Removes temporary files**	**Calls exit handlers**	**Terminates program**
Abort()	Implementation defined	Implementation defined	Implementation defined	No	Abnormal
_exit()	Implementation defined	Implementation defined except in POSIX	Implementation defined	No	Normal
Quick_exit()	Implementation defined	Implementation defined	Implementation defined	At quick_exit()	Normal
Exit()	yes	yes	yes	Atexit()	Normal
Return from main()	yes	yes	yes	Atexit()	Normal

Error handling methods: Global error indicator

The global error indicator is a static variable or a data structure that will indicate the error status of a function call.

There are three types,: errno, feraiseexcept() (floating-point error indicator) and Microsoft Visual C's getlasterror(). These pretty much do the same job, they just exist in different IDEs. They're all fail-soft, meaning the program can still continue after encountering these errors.

The little voice in your head is probably saying yay! But don't celebrate just yet! It's both a nifty little thing to have and the worst thing that can ever happen to you in a decade. Back to our earlier example on files, imagine if the program kept going, writing junk to your files with no way of recovering the data except starting from scratch! Ouch! That simply means they need to be used with great care, and you need to REALLY understand what your code is doing.

Rules for using global error indicators

A number of rules apply when using these functions. We will go with erron since it is the most universal of the three.

- Set errno to zero before calling a function, and use it only after the function returns a
- Value indicating failure
- The value of errno is zero at program start up but must never be set to zero by any library function.
- The value of errno may be set to nonzero by a library function call whether or not there is an error, provided the use of errno is not documented in the description of the function in the C standard.
- It is only meaningful for a program to inspect the contents of errno after an error has been reported. More precisely, errno is only meaningful after a library function that sets errno on error has returned an error code.

Errno error codes

This table summarises what error codes exist under errno and what each one represents:

Errno value	**Error**
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	I/O error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file number

10	No child processes
11	Try again
12	Out of memory
13	Permission denied

Error handling methods: non-global error indicators

Non-global error indicators are, as the name suggests, variables that indicate an error state. These are typically used with files, with the C standard providing 3 notable functions for this purpose: feof(), ferror() and clearerr(). These work in pretty much the same way as errno, so we will not go into much further detail here.

Error handling methods: Function return values

Function return values are values that are used by functions to indicate errors by returning a value. A lot of the standard functions in C have this functionality-built in. You may remember that when we covered specific functions in previous lessons, we specified a return value.

A very vivid example of this is the malloc function, which if you recall, returns a null pointer if there's some kind of error in the allocation process.

The caller function is responsible for checking for error conditions in this case. These work like magic, except that you cannot use the return value for other purposes. This can be a punch in the face, especially if you wanted your code to rely on return values to pass function states or other information like that. Also, this can make your code significantly longer.

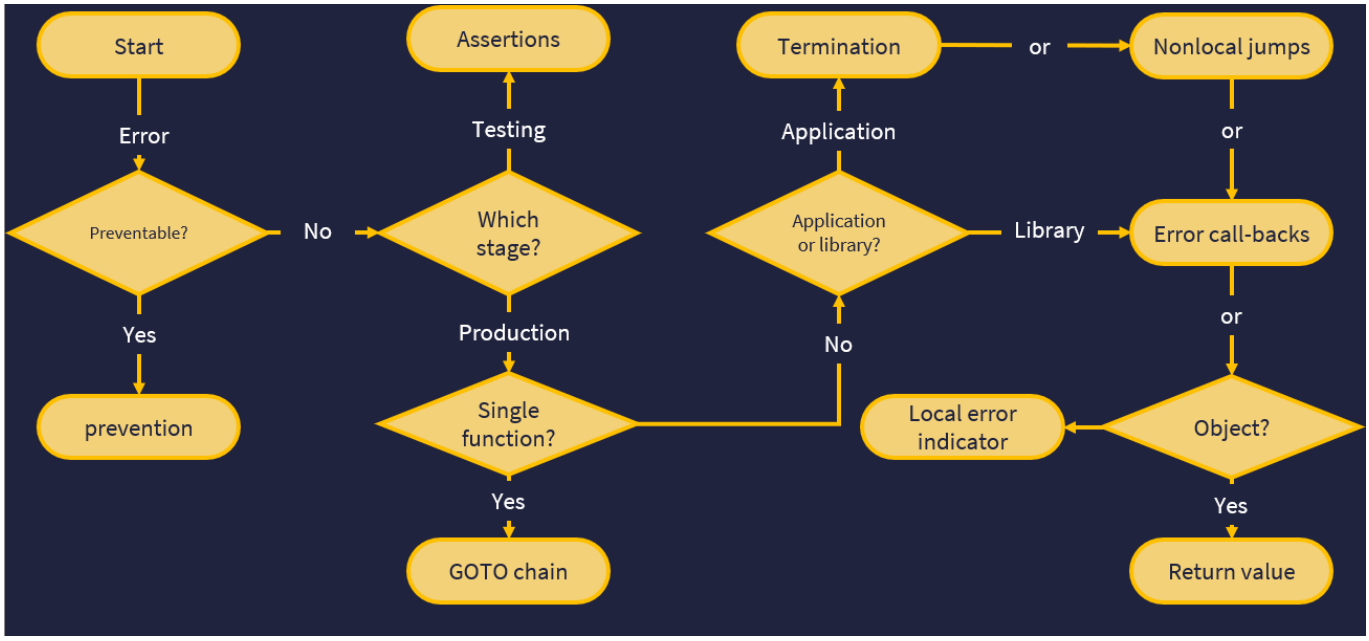
It's also pretty easy to ignore return values, and this has the potential of creating explosive chaos, especially concerning code security.

Error handling methods: Jumps

You can also use jumps to manage errors. If the condition that causes an error is encountered, the code jumps to a predetermined line and resumes execution from there. This is not really a good way of writing code, as it has the potential to create untold problems with your code that are notoriously difficult to debug. It is better to use structured loops wherever possible and avoid jumps as much as possible. To put this into perspective, think of what would happen if you had set a jump to a certain line, then you decide you don't want that line anymore, but then forget to remove the jump. The computer would likely not compile your code, but you might have trouble figuring out where the problem is, and spend hours trying to figure out what went wrong.

How to ascertain which error handling method to use...

This flowchart summarises the process you can follow to ascertain which error handling method you can use in your code.



Moving on with our project

The last part of this lesson is entirely dedicated to our project.

You probably have some sort of flowchart already, perhaps even some pseudocode or even actual code! That is totally fine, as there are many ways of killing a cat. In this lesson though, we will look at my approach to the pseudocode and flowchart. You can use this as a guideline if you ever get lost while trying to build your own solution. Let's get the camera rolling!

References

GeeksforGeeks. (2017). Error Handling in C programs - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/error-handling-c-programs/

Gupta, N. (2019). What is difference between Defect, Bug , Error ? - The Startup - Medium. [online] Medium. Available at: https://medium.com/swlh/what-is-difference-between-defect-bug-error-b477e76b5502

Svoboda, D. (n.d.). Beyond errno Error Handling in C. [online] . Available at: https://resources.sei.cmu.edu/asset_files/Presentation/2016_017_101_484207.pdf

Tutorialspoint.com. (2021). C - Error Handling - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/cprogramming/c_error_handling.htm

W3schools.in. (2021). C Error Handling. [online] Available at: https://www.w3schools.in/c-tutorial/error-handling/