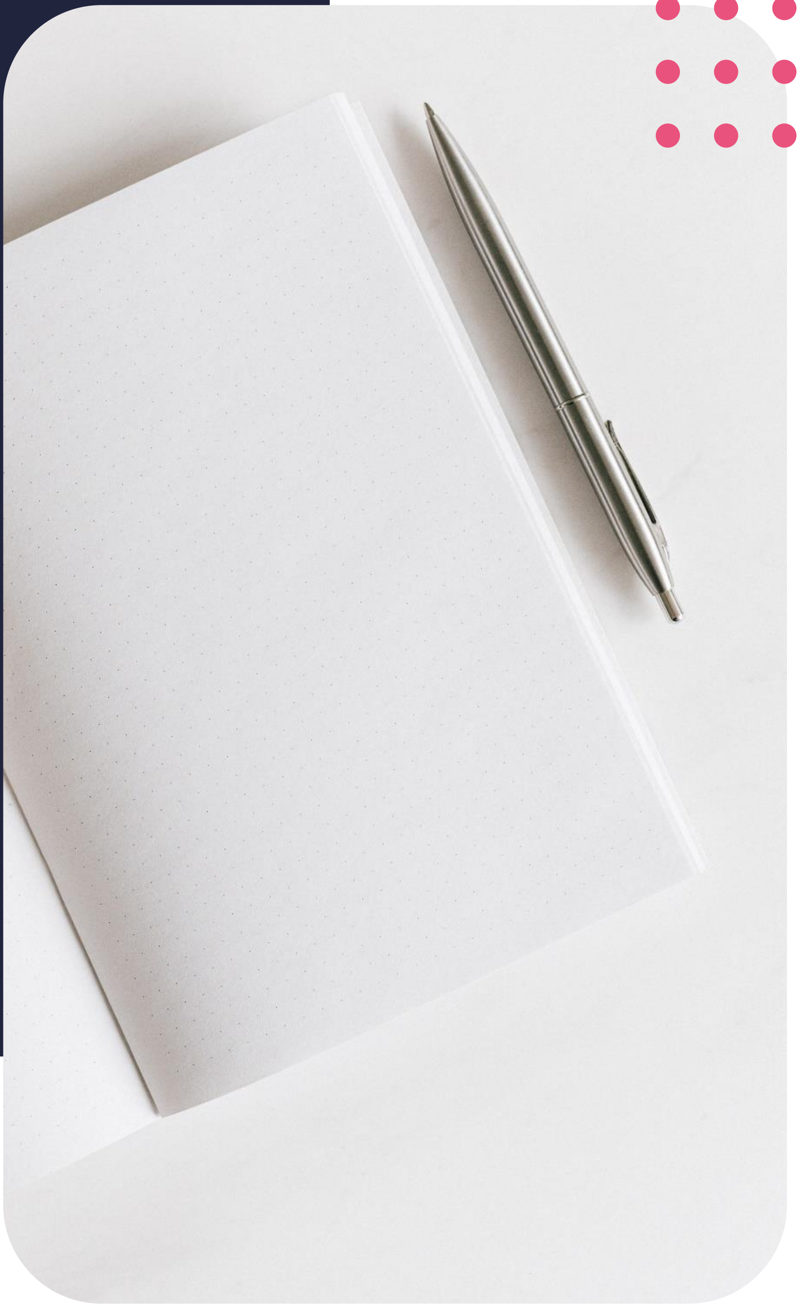
Diploma in Computer Science

Environment setup

Contents



Lesson outcomes

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

- Know the various IDES available for C
- Learn how to write and compile code in an IDE
- Explore the factors that affect your choice of IDE
- Explain the components of a typical IDE

IDEs

It's very rare to mention programming without mentioning IDEs. In modern programming, this is where most of the time in software development is spent. IDEs have taken centre stage in software development. It's worth noting, however, that not using an IDE doesn't mean you can't write code; you can actually write your code in any text editor and compile it manually in the corresponding compiler, but why would you do that to yourself?!

A closer look at IDEs

An IDE, short for integrated development environment, is a software application that provides a complete package of facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger. In simpler language, an IDE combines all of the tools that you require to build an application.

IDEs are typically used by developers because they speed up the development of applications by a long shot, and I mean a really long shot.

Most of the utilities needed for developing applications are already configured and set up as part of the installation process. Imagine walking into a carpentry workshop and you find the glue and nails on the workbench, saws and sanders plugged in and ready to use, and wood laid out. It would be pretty easy, assuming you're already a carpenter of course! This is the same thing with an IDE, it makes your life as a developer so much easier.

An IDE typically consists of a source code editor, a debugger, a compiler and several other plugins and extensions that can be made unique to the programming language.

A source code editor is a text editor that is used for writing software code with features such as syntax highlighting with visual cues, providing language specific auto-completion, and checking for bugs as code is being written. This is probably the first way in which you will know if you have written code with wrong syntax. A text editor within an IDE will typically highlight wrong syntax right after you finish a statement, and in some cases give you hints on what to change in order to correct the statement. Automated code checking is called linting. Some IDEs will warn you if your code deviates from the official style guides. This isn't always perfect, of course. An IDE can only suggest a fix based on satisfying a language specification, and in the wrong context, that can break your code. Some really smart IDEs can correct small errors, which is great for keeping obvious syntax errors out of your code. That said, what's more important is your understanding of the code that you're writing and what you're trying to achieve than relying on an IDE for corrections.

Most IDEs also contain a debugger. A debugger is a program for testing other programs.

It has the capability of graphically displaying the location of a bug in the original code. This is very handy if you are writing a large program with thousands of lines of code. It makes it a lot less tedious to find and fix obvious bugs. An easy way to think of a debugger is “instant replay”, or how you would replay a video sent to you that has a scene that happens too fast; you would play the video in slow mo. and see what happened step by step. The debugger allows you to visualise your code the same way your computer will see it once you compile it. You recall that a program runs from the beginning and halts at the end, this is the case even when running in a debugger. You often know that errors occur in a certain part of your program that is only executed long after you start running the program. For this, a breakpoint is set. A breakpoint tells the debugger where to “stop” execution so that you can take a look at what’s going on.

Another way of using a debugger is called stepping. In stepping, the debugger advances the program one line at a time. This is a powerful way to monitor the effect that your program has on variables.

A debugger will allow you to track the creation of variables, entering and leaving functions, any calculations that you make and tracks control structures, just to mention a few of the super useful things that it allows you to do. Using a debugger is essential to finding semantic errors in the behaviour of your program. It’s going to take a bit of practice to effectively use debuggers, but its most certainly well worth it. As you use a debugger, you will realise that it is a fairly repetitive process

An IDE makes it super easy to do simple repeatable tasks as part of creating a local build of the software like compiling computer source code into binary code, packaging binary code, and running automated tests.

Modern IDEs offer intelligent code completion. This is a context aware process where pieces of code are suggested in the form of pop ups, and the programmer can then select he text in the pop up and it gets appended to the code. This has the direct effect of greatly reducing typos and also reducing the time it takes to complete an entire project.

Some IDEs offer functionality of translating code from one programming language to another. The software that does this is called a source-to-source compiler, transpiler or transcompiler. This software takes source code form one language as input and produces source code of another language as output. The input language and output language are typically at the same level of abstraction. Note that this is different from a traditional compiler, which converts languages across generations, that is, from high level language to low level language. A transpiler is useful when, for example, the existing code will break backward compatibility if used in its current state. The code will be translated to match the current version of the program. If developers decide they want to use a different language for their codebase, they can use a transpiler to change the source code to a different language without having to rewrite the code in that language by hand. It saves a lot of time, but it requires manual intervention from time to time for the code to work properly.

Available options

Let’s now look at some of the options available for C development. It’s important that you try out a few IDEs and see which one works best for your system and works well for you in terms of ease of use among other factors IDEs are like cars, a car that works for me and my job and taste might not be the same as the car that works for you. For the purposes of demonstration in this course, Dev C++ will be used.

Dev C++ is a free, open-source IDE. It is often referred to as one of the best C IDEs out there. The sad part is it only runs on windows; there are no versions for Linux or Mac. Dev C++ has features such as code completion, syntax highlighting, a built in GCC compiler, a debugger, and a profiler. Apart from the provided compiler, you can even add your own.

Visual studio is an IDE developed by Microsoft. It is available for windows. Along with the standard set of features, it has a wealth of additional features such as code snippets, a great quality API and integration with Git. It also has a lot of other goodies, such as a debugger that can debug both source code and machine code, project deployment tools, database integration, server setup and a powerful version of the normal code completion tool called IntelliSense, it uses language semantics and the context of your source code to deliver appropriate code completion suggestions. These goodies, as expected, come at a cost. Expect this IDE to sit on an upwards of 4GB of your storage space, and you need a pretty powerful processor to enjoy the experience. Also, if you want to build commercial applications, you need to purchase a license.

If you're not up for all the bells and whistles that come with Visual Studio, but still want to use an IDE that's close to Visual Studio, then your best bet would be Visual Studio Code. It runs on Windows, Linux and MacOS. It's not really a C IDE, but rather, a code editor for a variety of languages. With the right extensions, you can turn it into a C IDE, complete with a compiler. Visual Studio Code is not focused on projects but is built around a file system. It makes up for this with a handful of features such as GIT integration among others borrowed from Visual Studio.

CodeBlocks is another alternative open-source IDE. It runs on Windows, Linux and MacOS CodeBlocks comes with a compiler and a debugger, and also supports profiling and auto completion, like many other IDEs. You can also add many other plugins using extensions.

Eclipse is a fully featured IDE, available for Windows, Mac, and Linux. It comes with the standard set of features, such as debugging, code compiling, auto code completion, profiling, refactoring, static code analysis, drag and drop, and so on. Eclipse is considered one of the simplest IDEs around, which is a very useful attribute especially for beginners. As with most other IDEs, it is open source, and free, which is also a great plus.

On Android, you can grab AIDE or Mobile C from the Play store. The former even comes with features that you would expect from a desktop IDE, including Git integration. For iOS, a number of options such as C Compiler and Mobile C exist in the app store. The mobile versions can be a bit difficult to use due to the smaller screens that information is displayed on, and mobile devices aren't exactly cut out for typing large amounts of text.

Aside from all these feature rich IDEs, nothing at all stops you from firing up a humble text editor and writing your code there. You lose a lot of helpful features, but if you are really resource constrained, this could actually work in your favour. Other noteworthy IDEs include Sublime text, NetBeans, QT creator and brackets. There are many, many IDEs out there, covering them all would require an entire module. Here we've only covered the most popular ones, which you will probably encounter in your journey as a computer scientist.

GIT

You will probably hear Git being mentioned a lot as you progress with software development. Git is a version control system that is used to track changes in source code during software development. Git was created by Linus Torvalds in 2005 for use in the development of the Linux kernel we also mentioned in module 1 that the Linux kernel is free and open source, so anyone- even you, can go over to the Linux repository and edit the code, contributing to the development of the Linux kernel! Git is free and open source, so you can freely download it and use it in your personal projects. Git is useful especially when programmers are working together on a project; it is a great way to coordinate their work. Git is used to track changes to a set of files; the files are stored in a git repository, and the versions of the code stored there are tracked as a record of the history of changes.

The version control system is distributed, that is, there is a remote repository which is stored in a server and a local repository which is stored in the computer of each developer. This means that the code is not just stored in a central server, but the complete copy of the code is also stored in every developers' computer. This whole idea of having the same stuff everywhere probably sounds like a waste of space and adding more confusion to an already fast-moving cycle. In reality, many developers often work on one project. A version control system will ensure that there are no code conflicts between the developers, for example, the same line of code in a file being edited by different people, or a line of code being deleted by one developer and edited by another developer. On top of that, requirements in a project change often, so if requirements have changed, a git system allows the developers to go back to earlier code easily. It is also useful when several projects are running at the same time and use the same codebase.

We will take a closer look at git when we start working on our project. There are a number of other version control systems, but git is by far the most popular. git is not really necessary for you to write code, but it is worth noting that git has grown to become the de facto standard in the software development industry. It will make a big difference if you learn about it while you're still learning how to write code. When you eventually become a pro and move on to large projects, you will have a lot less surprises in the form of standard procedures that you have to learn.

Choosing and IDE

When choosing an IDE, you consider a number of factors that make the IDE suitable for a particular project.

Cost

A lot of IDEs on the market are free. Better still, a lot are free and open source! Free IDEs have the benefit of costing nothing but paid for packages usually come with the benefit of tons and tons of features. When considering cost, the biggest factor here is the trade-off of features, you need to consider if you need the extra paid for features.

Speed

One of the biggest deal breakers when choosing an IDE is slow speed. An IDE can have tons of really useful tools, but if it is clunky. Performance issues can also impact on compilation, which is an already lengthy process for large projects. Some IDEs are bloated with unnecessary features which you may never use. When choosing an IDE, it is best to choose one which has the set of features that you need, and not bloat your PC with tons of files that you never get to use. Just like any other software package, IDEs have minimum platform requirements, such as processor speed, amount of RAM and so on. These also need to be considered as even the IDE that meets your needs can have massive overheads that can choke your PC. This might sound like nit-picking; to put this into perspective, a study by Sharp found out that the average UK office worker wastes at least 21 days each year due to slow or outdated technology! That is quite a lot of time, especially if you are a computer scientist working with tight deadlines.

Ease of use

What's the use of having an IDE if it's overly complicated?! The IDE you pick must be easy to navigate, well, at least to you. This not only speeds up the development process but saves you a lot of frustration trying to navigate complicated software.

The compiler

Now for the elephant in the room, the compiler. There's really no point in writing code in high level languages if we won't be able to compile it and run it. A lot of IDEs are bundled with compilers, which will make your life a lot easier. It will just be as easy as writing your code then clicking run. the IDE will already be set up to compile and run at the click of a single command. This is a lot easier than writing your code in a random text editor, then using a whole lot of commands to link the text file to a compiler, then compiling the code. The nightmare will turn even darker if you run into any errors.

The choice of a compiler is pretty straightforward. Always aim for stability and compatibility. Typically, the internet will guide you in choosing the most stable compiler. Besides your own observations, reviews speak for themselves. An unstable compiler is just all sorts of trouble from so many angles, from weird program output to unexpected program behaviour, or flat out refusing to compile for mysterious reasons. You might find that in some cases, the solution to code that just won't run is simply switching to a different compiler!

To summarise, the whole point of having an IDE is improving productivity. Of course, you could arbitrarily pick any IDE then get so used to it that it actually makes you productive. The bottom line is, after taking all the factors that we discussed into consideration, always choose the IDE that gets the job done and has the features that you need!

Download and setup

Now comes the part we have probably all been waiting for, the actual installation of the IDE and writing our very first line of code! There are many IDEs out there, in fact, it would be well worth your while to spend some time exploring a handful of them and see the one that works for you. For the purposes of illustration, we will just run through the installation on the three most popular platforms, windows, Android, and Mac.

For the purposes of this demonstration, we are going to be using Dev C++. It is a popular, lightweight windows IDE that can compile C and C++ programs.

See demo for additional content

Setup on Android

The setup process on Android is straightforward. Head over to google play store, and search for AIDE (used for illustrative purposes) tap on it when it pops up, tap install, and after a few moments, your shiny new IDE is ready to use

Running your first program

Now that we have installed our IDE, we are ready to run our very first program. For this we will use the "hello world" program, the first program that nearly every developer writes when learning to code!

A bit of orientation

The Dev C++ IDE is fairly straightforward. Under file, you can create new source files and save them. under execute, you can compile and run your program. Those are the two most important for now. It's always a good idea to play around with the IDE and familiarise yourself with what each button does.

Writing code

Remember the general structure that we learnt about in our previous lesson? We are going to be using that in all the code that we will be writing from this point on.

Compilation and execution

After writing your code, you need to compile it then run it. Like we just mentioned, this is done using the compile button, or pressing F11. The compiler will compiler will compile your code and immediately run the program. The program should show the text "Hello world" This means your program has run successfully! Congratulations!

References

Aditya Sridhar (2018). An introduction to Git: what it is, and how to use it. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/

Best C++ IDEs or Source Code Editors for Programming [2020 (2020). Best C++ IDEs or Source Code Editors for Programming [2020]. [online] Hackr.io. Available at: https://hackr.io/blog/cpp-ide

Booty, L. (2016). Office workers waste more time on slow tech than they spend on holiday. [online] Realbusiness.co.uk. Available at: https://realbusiness.co.uk/office-workers-waste-more-time-on-slow-tech-than-they-spend-on-holiday/

Ci (2019). 5 Must See C, C++ & C# IDEs for iPhone & iPad -. [online] iPhoneNess. Available at: https://www.iphoneness.com/iphone-apps/c-programming-ios-ide/

GeeksforGeeks. (2020). Top 5 IDEs for C++ That You Should Try Once - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/top-5-ides-for-c-that-you-should-try-once/

reddit. (2016). r/androidapps - What’s the best C/C++ compiler and IDE? [online] Available at: https://www.reddit.com/r/androidapps/comments/5t6d53/whats_the_best_cc_compiler_and_ide/

Redhat.com. (2020). What is an IDE? [online] Available at: https://www.redhat.com/en/topics/middleware/what-is-ide

Utah.edu. (2020). Programming - Topics. [online] Available at: https://www.cs.utah.edu/~germain/PPS/Topics/index.html