Noroff
School of technology
and digital media

Noroff Accelerate
C++ bootcamp study plan

1. Introduction

C++ is a cornerstone of computer programming within the computer science field, renowned for its speed and efficiency among programming languages. Despite being one of the oldest languages, C++ remains in high demand due to its ever-expanding range of applications. These include, but are not limited to, 1) operating systems, 2) game development and computer graphics, 3) GUI-based applications, 4) database management systems, 5) libraries (particularly for other languages such as Python), and 6) embedded systems.

As of November 2022[1], the TIOBE Programming Community index, which measures the popularity of programming languages, ranks C++ in 4th place, while its predecessor, C, holds the 2nd spot after Python. The TIOBE index is calculated based on the number of skilled engineers worldwide, the availability of courses, and the presence of third-party vendors. This ranking underscores the immense potential and numerous opportunities associated with learning C++, as it continues to be a critical skill in the rapidly evolving world of technology and software development.

The bootcamp is designed to provide participants with the technical skills necessary to proficiently write computer programs in C++. Considering the continued demand and usage of the C language, C++'s predecessor, the course will also highlight the differences between the two languages and delve into C's unique features. To accomplish this, the bootcamp covers a comprehensive range of essential C++ elements, including constructs, standard libraries, and object orientation, supplemented by assignments of varying difficulty levels and a capstone project. The curriculum adheres to the latest version, C++20, as its reference.

The bootcamp's primary focus is on the language features and standard libraries, as the industry often requires "high proficiency in C++" regardless of the specific job title. Nevertheless, further exploration of additional libraries and topics such as computer vision, machine learning, and communication libraries is encouraged through follow-up study programs, allowing participants to broaden their C++ knowledge and application in various domains.

2. Target group

The bootcamp is designed for individuals pursuing a career in programming and development in the IT industry, encompassing various domains such as game and embedded development. Participants can expect to gain foundational knowledge in computer programming, algorithms, and problem-solving skills. While C++ is a core component of computer science, its applicability extends to other fields of study, including physics and mathematics. Consequently, candidates from these disciplines, even at advanced levels such as master's or PhD, can benefit from the latest updates and insights on C++ provided by the bootcamp, enhancing their programming and problem-solving abilities. In addition to these groups, the bootcamp is also suitable for professionals working in industries where C++ is commonly utilized, such as finance or robotics, as well as educators and trainers who teach programming or computer science and wish to stay current with the latest C++ features and best practices.

[1] https://www.tiobe.com/tiobe-index/

Overall, this C++ bootcamp at the bachelor level caters to a diverse audience, all seeking to strengthen their expertise in the C++ programming language.

3. Educational profile

This C++ bootcamp aims to empower candidates with the general, core features of the C++ programming language as well as problem-solving skills as the objectives of this bootcamp. It will equip them with the necessary knowledge of C++ and skills to be able to write intermediate to advanced-level programs. The candidates will in addition learn the basics of algorithms and data structures as a part of problem-solving skills. Since the goal of this bootcamp is to teach C++ in a general manner, candidates develop mostly applications working with command prompt/terminal to keep the UI simple, but with a more focus on the application logic and C++ standard libraries.

After completing this course, candidates can write up to advanced C++ programs using standard libraries, can implement best programming practices in their code, can document and maintain their code well (using repositories), and can also write in the C programming language.

4. Language of teaching and assessment

The teaching, curriculum, and syllabus are in English. Candidates' learning resources and literature will be in English. Most learning resources and terminology come from English-language sources, such as manuals or equipment and software, lecture videos, and articles from the Internet. In the end, candidates will be assessed in English.

5. Career possibilities

Upon successful completion of the bootcamp, candidates possess the knowledge, skills, and general competence to work as a C++ developer. There is a range of developer positions that need proficiency in C++. Job positions associated with this role include 1) game and game engine developer, 2) embedded developer, 3) network communication developer, 4) security product developer, 5) computer graphics/vision/GPU developer, 6) blockchain development, and 7) real-time and robotic development. Due to the speed and efficiency of C++, compared with other languages, it can be used in other similar situations with such needs. The bootcamp does not focus on specific technical details needed in any of the above job positions, but it is kept general to cover a wide range of jobs in need of proficiency in C++.

However, the sector constantly changes as technology development and cyber shifts rapidly. Therefore, occupational and job descriptions will vary and change over a relatively short period. The main objective of this course is to provide candidates with practical and industry-related education within their studies.

6. Required prerequisite knowledge

- Basic computer literacy is a compulsory prerequisite for this bootcamp. Candidate should be able to work with an operating system such as Windows, Linux, or macOS, and can explore the web and use its related software without any problem.
- Some basic understanding of algorithms or preliminary knowledge of any programming language is a plus and recommended.

7. Overall learning outcomes

The overall objectives are formulated based on what a candidate knows, can do, and is capable of doing as a result of a learning process. The outcomes of the completed learning process are described in three categories:

- **Knowledge**: understanding theories, facts, principles and procedures in the discipline, subject area and/ or occupations.
- **Skills**: The ability to utilise knowledge to solve problems or tasks (cognitive, practical, creative, and communication skills)
- **General competence:** The ability to utilise knowledge and skills in an independent manner in different situations.

The outcomes of the completed learning process are outlined below:

Knowledge

On successful completion of the bootcamp, the candidate:

- Has knowledge of concepts, processes and tools used in developing software programs in C++.
- Has insight into relevant standards and quality requirements of programming in C++.
- Has knowledge of the software development industry and is familiar with the field of work.
- Can update their vocational knowledge.
- Understand the importance of software programming from a societal and value-creation perspective.

Skills

On successful completion of the bootcamp, the candidate:

- Can apply C++ software programming knowledge to practical and theoretical problems.
- Masters relevant C++ programming tools, materials, techniques, and styles.
- Can write C++ programs and use debugging tools.
- Can find information and material that is relevant to a software development problem.
- Can study a situation and identify software programming issues and what measures need to be implemented.

General competence

On successful completion of the bootcamp, the candidate:

- Understands the principles that apply in software programming and has developed an attitude of practising the discipline.
- Can carry out work based on the needs of selected target groups and build relations with peers and external target groups.
- Can develop methods, products, and/or services relevant to practising as a C++ software developer.

8. Organisation and structure

The table below shows how the bootcamp are distributed across courses.

Course	Course name	Number of lectures
Course 1	**Getting started:** • Introduction to computer programming: architecture (CPU, memory, I/O, data representation), the role of operating systems, algorithms • Introduction to C++ programming: history, features, advantages, and differences between C and C++. • Setup the environment: Microsoft Visual Studio	1

Course 2	**Fundamentals of C++:** • Basics of programming: program structure, data types, variables, expressions, operators, and namespaces • Control structures: if-else statements, loops, and switch-case statements • Arrays and pointers: one-dimensional and multi-dimensional arrays, pointer variables, and dynamic memory allocation • Functions: defining and calling functions, recursion, and function overloading • Structures and unions: defining structures and unions, accessing structure members, and nested structures	4
Course 3	**Object-Oriented Programming:** • Classes and objects: defining classes and objects, constructors and destructors, access modifiers, and member functions • Inheritance: deriving classes, base and derived class relationships, and virtual functions • Polymorphism: function overloading, operator overloading, and virtual functions • Object-oriented design: SOLID principles • Templates: function templates and class templates, as well as template specialization and template metaprogramming	4
Course 4	**Exceptions and Input/Output (I/O):** • Exception handling: try-catch blocks, throwing and catching exceptions, and handling multiple exceptions • File input and output: opening and closing files, reading and writing files, text/binary files, stream manipulation	2
Course 5	**Standard Template Library (STL):** • Containers: sequence and associative containers, container adapters • Iterators: input, output, forward, bi-directional, and random-access • Algorithms: non-modifying and modifying sequence algorithms, Sorting and numeric operations • Functors: predefined and custom function objects • Allocators: allocation and deallocation in STL containers, custom allocators	2
Course 6	**Advanced C++ Techniques:** • Smart pointers: unique_ptr, shared_ptr, and weak_ptr for modern memory management • Lambda expressions: defining and using lambda expressions • Concurrency: creating threads, synchronization, and mutexes • Connectivity: sockets and network communication in C++	3
Course 7	**Capstone project:** • Working individually on a case project selected from a number of projects	-

9. Scope

The bootcamp takes place over 10 weeks or 20 teaching days comprising a total of 3 hours per week of class time and a maximum of 6 hours per week of contact time with a lecturer. The bootcamp will be presented remotely as outlined below.

	Total for bootcamp	Per week
Duration	Ten weeks part-time	-
SME-facilitated sessions (lectures)	30 hours	2 x 90 minutes synchronous webinars per week
Supervised group problem-solving and feedback	40 hours	2 x 2 hours
Live Q&A session with instructor	20 hours	1 x 2 hours
Self-study	50 hours recommended	5 x 1 hours recommended per week
Total hours:	140 hours	14 hours

10. Teaching, Learning and Work methods

The foundation of Noroff's pedagogic concept is that candidates learn best by actively participating in industry-lead learning activities. Therefore, teaching time applies practical learning methods to accumulate relevant knowledge, skills, and general competence.

The subject-matter expert (SME) facilitating the bootcamp acts as a supervisor, mentor, and instructor. Close monitoring of individual learning needs is crucial for candidates' achievements.

Our teaching methodology is founded on a problem-based learning model, which ensures that the candidates have independently acquired theoretical and technical knowledge through lessons, syllabi, and course materials provided before lectures and teaching hours.

The instruction is tailored to the needs of the candidates. Candidates completing the bootcamp have the responsibility of preparing for lessons and working through the course material. Through observation, testing, industry-relevant tasks, course evaluations and guidance, subject-matter experts review and adapt their teaching methods to the greatest extent possible. SMEs adjust the course materials and remain differentiated based on the candidate's goal achievement.

A regular teaching day consists of the following:

Teaching and teacher-supported work

Candidates have a subject-matter expert (SME) available for guidance, supervision, and professional follow-up. Teaching consists of various learning activities where the SME lectures, guides, and instructs. The methods include exercises, lectures, demonstrations, and candidate presentations or discussions. Skills are developed through course-specific activities that may run over one or more days. The methods used in these sessions are adapted as far as possible to candidate groups and individual candidate learning achievement.

Candidates can access bootcamp learning resources and material used for assessments through the Learning Management System (LMS) used by Accelerate.

Self-study

The time set for self-study is used for completing assessments related to the course's described learning outcome(s). These assessments are completed over time, and candidates must apply their knowledge and skills. The SME and Succelerators (Accelerate Success Managers) are available for questions and follow-up. After submission, assignments are assessed and graded in the LMS based on the assessment criteria in the assignment description and the course's learning outcomes.

Noroff's teaching model sets the candidate's responsibility for self-learning in focus. This requires a clarification of expectations and motivation for the candidates from the first day of the bootcamp. In today's working life, candidates must solve various problems using all available aids. The candidates' independent use of resources in the form of assignment descriptions, video explanations, online resources and similar will be crucial for their learning outcomes and achievements throughout the bootcamp.

Individual candidate follow-up

Candidates are followed up both on their participation in the LMS and through their work requirements. The SME can assess individual academic needs by evaluating a candidate's participation.

Assessment submissions will reveal if a candidate is struggling. Assessments must be submitted and verified. They will be contacted if a candidate has not delivered at a given time. After submission, candidates receive feedback on the work that has been delivered.

11. Assessments

The learning outcomes of candidates are assessed with a set of assessments. These assessments are evaluated as passed or failed, after which verbal or written feedback from the SME indicates the level and the submitted works strengths and further developments needed.

12. Equipment

Listed below are the equipment requirements for this bootcamp. The party responsible for procuring the equipment and software is also detailed in the table.

	Responsible for equipment procurement	
Required Equipment	**Noroff Accelerate**	**Candidate**
PC		X
Powerful Graphics Card		
Internet Connect		X
Noroff LMS	X	
Office Tools		
Software for prototyping		

Software for programming		X
Software for image editing		
Software for Vector Graphics		
Recommended Equipment	**Noroff Accelerate**	**Candidate**
External storage device		
Noise-cancelling headset		X
Two high-resolution monitors		X

13. Course descriptions

The seven courses are outlined below:

Course 1: Getting Started

Academic content

This course focuses on providing a solid foundation in C++ programming, starting with an overview of its history, features, and advantages. Candidates will gain an understanding of the evolution of C++ as a programming language, as well as its core features that make it a popular choice for various software development tasks. The course will highlight the advantages of using C++ in terms of performance, flexibility, and its extensive use in industry applications. In addition, this module will guide candidates through setting up their programming environment using Microsoft Visual Studio, a widely used integrated development environment (IDE) for C++ development. By the end of this module, candidates will be well-equipped with the essential knowledge and tools required to begin their journey in C++ programming and dive deeper into more advanced topics in subsequent modules.

To ensure candidates start with the required basic knowledge of computer programming, the bootcamp starts with an introduction to computer programming. In this part, candidate learn to identify and describe the basic components of a computer system, including the CPU, memory, input/output devices, and storage. They also gain an understanding of data representation concepts, such as binary numbers, bits, and bytes, and their role in computer systems. Additionally, candidates recognize the function of operating systems in managing computer resources and facilitating the execution of programs. Lastly, they explore the relationship between computer hardware and software, as well as their interaction during program execution, to gain a comprehensive understanding of the inner workings of computer systems.

Course relevance

This course establishes a strong foundation for learners who are new to the language or seeking to deepen their understanding. By exploring the history, features, and advantages of C++, candidates gain valuable context and appreciation for the language's significance and versatility across various domains. This foundational knowledge is essential for learners to grasp the rationale behind certain programming constructs and techniques they will encounter throughout the bootcamp. Furthermore, the inclusion of setting up the programming environment using Microsoft Visual Studio ensures that candidates become familiar with a widely used IDE, equipping them with the practical skills necessary to develop, test, and debug their C++ code efficiently. The comprehensive

introduction provided by this module sets the stage for subsequent, more advanced topics, enabling candidates to seamlessly progress through the bootcamp curriculum and effectively apply their C++ knowledge in real-world scenarios.

Learning outcomes

Knowledge

On successful completion of this course, the candidate can:

- Identify and describe the basic components of a computer system, including the CPU, memory, input/output devices, and storage.
- Comprehend the concept of data representation, including binary numbers, bits, and bytes, and their role in computer systems.
- Recognize the function of operating systems in managing computer resources and facilitating the execution of programs.
- Understand the history of C++ programming, including its evolution and origins, and its relationship to the C programming language.
- Recognize the key features of C++ that differentiate it from other programming languages.
- Identify the advantages of using C++ for various software development tasks and applications.
- Gain familiarity with Microsoft Visual Studio, a widely used integrated development environment (IDE) for C++ programming.

Skills

On successful completion of this course, the candidate will be able to:

- Successfully set up the programming environment using Microsoft Visual Studio for efficient C++ development.
- Run basic C++ programs that utilize variables, data types, loops, and conditional statements.
- Apply the knowledge of data representation, such as binary numbers, bits, and bytes, to solve basic problems related to data storage and manipulation in computer systems.
- Assess the features of C++ to determine its suitability for specific software development tasks and applications, based on its differentiating factors from other programming languages.

General competence

On successful completion of this course, the candidate will have:

- An understanding of computer systems, their components, and data representation to adapt to various software development scenarios, regardless of the programming language or development environment.
- Draw upon the historical context of C++ programming, its relationship to the C programming language, and its distinguishing features to make informed decisions about its suitability for specific projects or tasks.

Course 2: Fundamentals of C++

Academic content

This course delves into the fundamental building blocks of programming, ensuring that candidates develop a strong understanding of essential concepts required for effective C++ development. The module begins by covering the basics of programming, including program structure, data types, variables, expressions, and operators. This foundational knowledge provides a clear and concise introduction to the components that make up a C++ program. Next, the module explores control structures, such as if-else statements, loops, and switch-

case statements, teaching candidates how to direct the flow of a program based on specific conditions. The course then moves on to functions, covering topics such as defining and calling functions, recursion, and function overloading, which are critical for modular and maintainable code. Additionally, the module delves into arrays and pointers, discussing one-dimensional and multi-dimensional arrays, pointer variables, and dynamic memory allocation techniques for effective memory management. Lastly, the course module addresses structures and unions, teaching candidates how to define and access structure members, work with nested structures, and understand the differences between structures and unions. Overall, this module equips candidates with a comprehensive understanding of core programming concepts essential for success in C++ development.

Course relevance

The academic content outlined in the module covering fundamental programming concepts is highly relevant to a C++ bootcamp course, as it provides the groundwork necessary for learners to build a strong understanding of the language and develop effective programming skills. By addressing the basics of programming, control structures, functions, arrays and pointers, and structures and unions, this module ensures that candidates acquire a solid grasp of the essential building blocks of C++ development. These core concepts form the foundation upon which more advanced topics and techniques can be introduced and effectively understood by learners. Moreover, this module's focus on essential programming constructs, such as control structures and functions, empowers candidates to write modular and maintainable code, which is crucial for creating scalable and robust software applications. Additionally, the inclusion of arrays, pointers, structures, and unions reinforces the importance of efficient memory management and data manipulation in C++ programming. By mastering these fundamental concepts, candidates will be better prepared to tackle advanced topics and real-world challenges throughout the bootcamp and beyond, ultimately enhancing their overall programming expertise and career prospects in the field.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Can explain the fundamental concepts of programming,
- Has the knowledge of different data types found in C++.
- Understands the difference between compile-time and runtime errors.
- Knows control structures and how they operate.
- Understands recursion as a problem-solving method.
- Explains how multi-dimensional variables are defined.
- Has the knowledge of pointers, pointers to pointers, and their use in data structures.
- Knows what a data structure is and how they are defined using struct and union, and how they differ.

Skills

On successful completion of this course, the candidate will be able to:

- Create and execute simple programs using a programming language, applying the concepts of program structure, data types, variables, expressions, and operators.
- Use conditional statements, loops, and other control structures to control the flow of a program.
- Can implement control structures such as if-else statements, loops, and switch-case statements to control the flow of a program.
- Can define and call functions in a program, including recursion and function overloading.
- Utilize arrays and pointers, including one-dimensional and multi-dimensional arrays, pointer variables, and dynamic memory allocation.

- Defines structures and unions, access structure members, and use nested structures to organize data within a program.
- Develop algorithms and write programs that utilize the concepts and techniques covered in the course.

- Debug and troubleshoot programming errors.

General competence

On successful completion of this course, the candidate:

- Applies their knowledge of programming to real-world applications.

- Can Create, access, and manipulate structures and unions to model complex data relationships, demonstrating the ability to apply these techniques to different problem domains and situations.

Course 3: Object-Oriented Programming

Academic content

The academic content of this course module focuses on the key concepts of object-oriented programming (OOP) in C++, which is essential for designing and developing complex, scalable, and maintainable software applications. The module starts with classes and objects, guiding candidates through the process of defining classes and objects, creating constructors and destructors, using access modifiers, and implementing member functions. This foundational knowledge enables learners to encapsulate data and behaviour within well-structured components. The course then moves on to inheritance, covering the concepts of deriving classes, understanding base and derived class relationships, and leveraging virtual functions for extensibility and flexibility in class hierarchies. Next, the module explores polymorphism, addressing function overloading, operator overloading, and virtual functions to enable greater abstraction and code reusability. The course also delves into object-oriented design, emphasizing the SOLID principles, which are critical for creating modular and maintainable software. Finally, the module introduces templates, teaching candidates how to implement function and class templates for generic programming, thus promoting code reusability and type safety. Overall, this module equips learners with a comprehensive understanding of OOP concepts in C++, allowing them to create robust, efficient, and maintainable software solutions.

Course relevance

The content presented in this course equips learners with the necessary knowledge and skills to create scalable, maintainable, and efficient software applications. OOP is a fundamental paradigm in C++ programming, and mastering its core principles, such as classes, objects, inheritance, polymorphism, and templates, is crucial for learners to succeed in the industry. By covering these essential topics, the module prepares candidates to handle complex software development challenges and utilize C++ features effectively to create well-structured, modular code. Additionally, the focus on object-oriented design principles, such as SOLID, ensures that learners are able to develop robust, adaptable software solutions that can easily evolve to meet changing requirements. The inclusion of templates further reinforces the importance of generic programming in C++ and promotes code reusability and type safety. Overall, the comprehensive understanding of OOP concepts gained through this module will significantly contribute to candidates' programming proficiency, preparing them for advanced topics and real-world projects throughout the bootcamp and beyond.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Understand the concepts of classes and objects in object-oriented programming.

- Learn to define classes and objects with appropriate constructors and destructors.
- Become familiar with access modifiers and their role in encapsulation.
- Gain proficiency in implementing member functions within classes.
- Comprehend the concept of inheritance and its applications in class derivation.
- Explore the relationships between base and derived classes.
- Master the use of virtual functions in inheritance hierarchies.
- Grasp the concept of polymorphism and its various implementations.
- Learn function overloading and operator overloading techniques.
- Deepen understanding of object-oriented design through SOLID principles.
- Gain knowledge of templates in C++ and their applications.
- Understands the difference between static and dynamic binding and their implications for C++ programs.
- Develop skills in creating and using function templates.
- Learn to design and implement class templates for generic programming.
- Understands the concept of namespaces and how they can be used to organize code.
- Understands the role of C++ in modern software development and the importance of good OOP design principles.

Skills

On successful completion of this course, the candidate will be able to:

- Define classes and objects in C++ and understand how they relate to each other.
- Define and implement class methods, including constructors, destructors, and copy constructors.
- Apply access modifiers to control the visibility and accessibility of class members.
- Use inheritance to create a hierarchy of classes with shared functionality and specialized behaviours.
- Use virtual functions to enable polymorphism and dynamic behaviour in class hierarchies.
- Implement function overloading and operator overloading in C++ programs.
- Use templates to create generic classes.
- Apply SOLID principles to create robust and maintainable object-oriented designs.
- Develop algorithms and write programs using OOP concepts covered in the course to solve complex problems.

General competence

On successful completion of this course, the candidate:

- Has gained a strong foundation in object-oriented programming principles and practices.
- Understands the ability to analyze, design, and implement software solutions using object-oriented techniques.
- Can gain improved code readability and organization by adhering to established object-oriented programming conventions.
- Can develop enhanced problem-solving skills through the application of object-oriented design patterns and practices.
- Understands how OOP concepts can be applied to real-world software development.

Course 4: Exceptions and I/O

Academic content

In this course, candidates explore the essential concepts of exception handling and file input/output (I/O) in C++ programming. They delve into exception handling techniques, including the use of try-catch blocks, throwing and catching exceptions, and managing multiple exceptions to ensure robust and fault-tolerant program execution. Furthermore, the module covers the fundamentals of file I/O, guiding candidates through the process of opening and closing files, reading and writing data to and from files, and working with both text and binary files. This academic content equips learners with the skills necessary to manage errors gracefully and handle data storage effectively in their C++ projects.

Course relevance

Teaching exception handling and I/O in this course ensures that candidates are provided with critical skills for building reliable and efficient software applications. Exception handling techniques are essential for creating robust programs that can handle unexpected errors and situations gracefully, without crashing or producing incorrect results. By learning how to handle exceptions properly, candidates can develop software that is more resilient and user-friendly. Furthermore, understanding file input/output operations is crucial for any programmer, as working with data stored in files is a common requirement in various application domains. The ability to read and write files, and handle different file formats, equips candidates with the practical skills needed to manage data effectively, which is a vital aspect of software development. Consequently, the inclusion of these topics in a C++ bootcamp course helps candidates become well-rounded and competent developers who can tackle a wide range of programming challenges.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Understands the concept of exception handling and its importance in creating robust and fault-tolerant applications.
- Learns to implement try-catch blocks for handling runtime errors and exceptions.
- Gains proficiency in throwing and catching exceptions for error reporting and recovery.
- Knows how to handle multiple exceptions using nested or sequential try-catch blocks.
- Develops a solid understanding of file input and output operations in C++.
- Learns to open and close files using appropriate functions and methods.
- Understands the differences between text and binary files and learns to handle each file type effectively.

Skills

On successful completion of this course, the candidate will be able to:

- Apply try-catch blocks to manage runtime errors and exceptions effectively.
- Write custom exception classes for more specific error handling.
- Find and catch multiple exceptions using nested or sequential try-catch blocks.
- Study file input and output operations in C++ programming and apply them in practical scenarios.
- Open and close files using appropriate functions and methods.
- Read from and write to files, handling various data types and formats.
- Distinguish between text and binary files and work with each file type accordingly.
- Apply proper exception handling techniques to improve error resilience and debugging skills in code.

General competence

On successful completion of this course, the candidate will have:

- The ability to identify and manage runtime errors and exceptions independently, ensuring robust and fault-tolerant applications.
- Capability to adapt exception handling techniques to various programming scenarios and effectively handle errors in different situations.
- Proficiency in utilizing file input and output operations to manage data storage, retrieval, and processing across diverse application contexts.
- Competence in working with different file formats, including text and binary files, and applying relevant techniques based on the file type.
- Improved problem-solving and debugging skills through the effective use of exception handling techniques in various programming challenges.
- Confidence in applying learned knowledge and skills to create maintainable, reliable, and efficient applications that handle errors gracefully.

Course 5: Standard Template Library (STL)

Academic content

In this course, candidates delve into advanced topics, focusing on the Standard Template Library (STL), smart pointers, and lambda expressions. The module introduces candidates to the key components of the STL, including containers, iterators, algorithms, and function objects, allowing them to efficiently manage and manipulate data structures using the rich set of built-in tools provided by the library.

Course relevance

Teaching the above academic content equips candidates with advanced skills that are vital for modern software development. By learning the intricacies of the Standard Template Library (STL), candidates gain proficiency in utilizing powerful and efficient data structures and algorithms that can greatly improve the performance of their code. This knowledge enables them to tackle complex programming challenges with ease, enhancing their overall competence as C++ developers.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Understands the purpose and components of the Standard Template Library (STL) in C++.
- Learns about various STL containers and their use cases in data management and storage.
- Gains proficiency in using iterators to traverse and manipulate data within STL containers.
- Develops an understanding of STL algorithms for efficient data manipulation and processing.
- Becomes familiar with function objects and their role in customizing STL algorithms.

Skills

On successful completion of this course, the candidate will be able to:

- Apply various STL containers for efficient data management and storage in diverse programming scenarios.
- Utilize iterators to traverse and manipulate data within STL containers effectively.
- Study and implement STL algorithms to solve complex data manipulation and processing tasks.
- Write custom function objects to tailor STL algorithms for specific use cases.

General competence

On successful completion of this course, the candidate will have:

- The ability to independently apply STL components, such as containers, iterators, algorithms, and function objects, in various programming situations to create efficient and maintainable code.

Course 6: Advanced C++ Techniques

Academic content

In this course, candidates delve into advanced topics, focusing on smart pointers, lambda expressions, concurrency, and connectivity. Learners explore smart pointers, such as unique_ptr, shared_ptr, and weak_ptr, which enable them to manage memory dynamically and safely, avoiding common pitfalls such as memory leaks and dangling pointers. Lastly, the module covers lambda expressions, teaching candidates how to define and use these concise, anonymous functions for flexible and expressive code. Candidates explore the concepts of multithreading and network programming, both of which are integral to building high-performance and scalable software systems. The module covers the fundamentals of multithreading, teaching candidates how to create threads, synchronize access to shared resources, and utilize mutexes to prevent data races and ensure the safe execution of concurrent code. This knowledge enables candidates to write efficient, parallelized programs that can take full advantage of modern multi-core processors. Additionally, the module delves into network programming in C++, focusing on the use of sockets to establish and manage network communication. By learning the ins and outs of sockets, candidates can create powerful networked applications, allowing them to communicate and exchange data seamlessly between different systems. The combination of these advanced topics equips candidates with essential skills to develop a wide range of software applications, from high-performance computing to distributed systems and beyond.

Course relevance

Teaching the above academic content equips candidates with advanced skills that are vital for modern software development. By learning the intricacies of the Standard Template Library (STL), candidates gain proficiency in utilizing powerful and efficient data structures and algorithms that can greatly improve the performance of their code. This knowledge enables them to tackle complex programming challenges with ease, enhancing their overall competence as C++ developers. Additionally, understanding smart pointers is essential for managing memory effectively, which is a crucial aspect of writing reliable and maintainable code in C++. Lastly, mastering lambda expressions allows candidates to write more expressive and flexible code, simplifying complex logic and facilitating code reuse. Together, these advanced topics ensure that candidates possess a comprehensive skillset, making them better prepared to tackle real-world software development projects and excel in their careers.

By learning multithreading, candidates can create applications that efficiently utilize system resources, leading to improved responsiveness and performance. This knowledge is crucial for developing high-performance software, such as video games, data processing tools, and other computationally intensive applications.

Furthermore, network programming is a vital skill for building distributed systems, which are becoming increasingly prevalent in the era of cloud computing and the Internet of Things (IoT). By mastering sockets and network communication in C++, candidates gain the ability to create robust and scalable networked applications, which are essential components of modern software systems. Overall, the inclusion of these advanced topics in a C++ bootcamp course equips candidates with a well-rounded skillset, enabling them to excel in various aspects of software development and tackle real-world programming challenges with confidence.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Understands the concept of smart pointers and their benefits in memory management.
- Learns to utilize unique_ptr, shared_ptr, and weak_ptr for efficient and safe memory management.
- Grasps the concept of lambda expressions and their use cases in C++ programming.
- Gains proficiency in defining and using lambda expressions for inline and anonymous functions.
- Understands the concept of multithreading and its benefits in concurrent programming.
- Learns to create and manage threads in C++ for efficient task execution and parallelism.
- Develops an understanding of synchronization techniques for coordinating and controlling thread execution.
- Gains proficiency in using mutexes to protect shared resources and prevent race conditions in multithreaded applications.
- Grasps the fundamentals of network programming and its applications in client-server systems.
- Becomes familiar with sockets as a means of establishing network communication in C++.
- Acquires the knowledge to implement network communication using sockets and related functions in C++ programming.

Skills

On successful completion of this course, the candidate will be able to:

- Apply smart pointers, such as unique_ptr, shared_ptr, and weak_ptr, for improved memory management.
- Find appropriate smart pointer types based on the ownership and lifetime requirements of objects.
- Study and implement lambda expressions to create inline and anonymous functions.
- Write and use lambda expressions for various purposes, such as customizing algorithms, creating short functions, or simplifying code.
- Apply multithreading concepts to create concurrent applications that efficiently execute tasks in parallel.
- Write code to create and manage threads in C++ for improved performance and resource utilization.
- Implement synchronization techniques to coordinate and control thread execution in complex scenarios.
- Use mutexes to protect shared resources and prevent race conditions in multithreaded applications.
- Study the fundamentals of network programming and apply them in client-server systems.
- Find appropriate socket types and functions to establish network communication in C++.
- Implement network communication using sockets and related functions in C++ programming for various network applications.

General competence

On successful completion of this course, the candidate will have:

- Proficiency in utilizing smart pointers for effective memory management, improving application safety and performance across diverse contexts.
- Competence in using lambda expressions to simplify code, create inline functions, and customize algorithms, adapting to different programming challenges with ease.
- The ability to independently design and implement concurrent applications using multithreading techniques for improved performance and resource utilization in diverse programming situations.
- Proficiency in employing synchronization methods and mutexes to ensure thread safety and prevent race conditions in multithreaded applications.
- Competence in applying network programming concepts, such as sockets and network communication, to create reliable and efficient client-server systems across various contexts.

- Enhanced problem-solving skills through the effective use of multithreading and network programming techniques in addressing diverse programming challenges.
- Confidence in applying learned knowledge and skills to a wide range of programming scenarios, creating adaptable, safe, and robust software applications that leverage the power of multithreading and network communication.

Course 7: Capstone Project

Academic content

The capstone project serves as a culminating experience that synthesizes the knowledge and skills acquired throughout the program. In this module, candidates work individually on a case project chosen from a variety of options, allowing them to tailor their project to their interests and career goals. The capstone project emphasizes the application of C++ programming concepts, techniques, and best practices to real-world scenarios. Candidates are challenged to demonstrate their proficiency in areas such as object-oriented programming, data structures, algorithms, memory management, and more. By tackling a complex, hands-on project, candidates gain valuable experience in problem-solving, debugging, code optimization, and project management. This module not only reinforces the academic content learned in previous courses but also prepares candidates for their future careers by showcasing their ability to independently develop, implement, and maintain C++ applications.

Course relevance

The capstone project provides candidates with the opportunity to integrate and apply the various concepts and techniques learned throughout the course in a practical, real-world context. By working on a comprehensive, hands-on project, candidates can demonstrate their mastery of C++ programming and problem-solving skills, which are crucial for success in the software development industry. This course, therefore, not only strengthens the candidates' understanding of the programming language but also equips them with the practical experience and confidence necessary to succeed in their careers as C++ developers.

Learning outcomes

Knowledge

On successful completion of this course, the candidate:

- Learns how to apply the principles and techniques learned throughout the course to design, implement, and debug a real-world C++ project.
- Has the ability to demonstrate a comprehensive understanding of C++ programming concepts, such as object-oriented design, standard libraries, data structures, and algorithms.
- Learns how to manage a small software development project, including setting objectives, establishing timelines, and allocating resources.

Skills

On successful completion of this course, the candidate will be able to:

- Effectively utilize problem-solving skills to tackle complex programming challenges and develop efficient solutions.
- Develop a project portfolio that showcases their proficiency in C++ programming, which can be presented to potential employers or clients.
- Enhance their C++ programming skills by implementing real-world projects, solidifying their understanding of the language's syntax, constructs, and libraries.
- Identify, diagnose, and resolve programming errors, as well as to create and perform tests to ensure the reliability and functionality of their code.

- Write clear, well-structured, and maintainable code, as well as to create comprehensive documentation that explains the purpose and functionality of their projects.

General competence

On successful completion of this course, the candidate will have:

- Reflect on the learning process and identify areas for further growth and improvement in their programming skills.
- Independent problem-solving: Candidates will develop the ability to approach complex programming tasks independently, analyze requirements, and devise appropriate solutions using their knowledge and skills in C++.
- Critical thinking: Candidates will learn to critically evaluate different approaches, algorithms, and design patterns, enabling them to select the most efficient and effective solutions for their projects.
- Project management: By working on a small real-world project, candidates will acquire experience in planning, organizing, and managing software development projects, ensuring that they meet deadlines and deliver high-quality outcomes.
- Adaptability: Candidates will enhance their ability to adapt to new programming challenges and technologies, demonstrating their capacity for continuous learning and growth in the ever-evolving field of software development.
- Creativity and innovation: The capstone project encourages candidates to explore novel ideas and solutions, fostering a spirit of creativity and innovation that can be applied to future projects and professional endeavors.
- Professionalism: Through the capstone project, candidates will develop a sense of responsibility and professionalism, demonstrating their commitment to delivering high-quality software and adhering to industry best practices.