Programming in Modern C++

Module M00: Course Outline

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Table of Contents

Module Objectives

- Understanding course outline in terms of modules and tutorials
- Understanding the Critical Actions needed before the Course starts
  - Revise C
  - Revise basic Data Structures (array, stack, queue, priority queue)
  - Revise Algorithms (sorting and searching, matrix-vector, graph)
  - Install gcc and gdb and try out several C programs with it
  - Suggest tutorials if you feel the need

Course Objectives

- Learn to develop software using C++ (**C++98/03**)
  - Features of C++ over and above C
  - Object-Oriented Paradigm in C++
  - STL for extensive code reuse
- Learn to improve software development using modern C++ (**C++11**)
  - Features of **C++11** over and above **C++98/03**
  - Concurrent Programming in C++
  - Better quality and efficiency by **C++11**
- Cultivate skills to design, code, debug, and test software written in C++
- Attain strong employability with hands-on skills of software development

Week 00: Quick Recap by Self Study and Tool Setup

- This week is for getting ready for the course
- *Understanding Course Outline*
  - **Module 00**:
    - This module with week-wise plan of modules and tutorials
- *Quick Recap of C*: C language, Standard Library and Programming:
  - **Module QR1**:
    - *Data Types*; *Variables*; *Literals*; *Operators*; *Expressions*; *Statements*; and *Control Construct*
  - **Module QR2**:
    - *Containers and Pointers*; *Functions*; and *Input / Output*
- *Recap of Data Structures and Algorithms*: NPTEL Course Design and Analysis of Algorithms or any standard textbook
- *Install and try build tool*: MinGW - Minimalist GNU for Windows
  - Use GDB: The GNU Project Debugger for code debugging
  - Check How to install gdb in windows 10 to install minGW and gdb for Windows together

Course Outline

- The course comprises:
  - 60 **Modules** (5 modules / week for 12 weeks). These are numbered serially as `Mnn`
    - These cover the course syllabus
    - These are used in assignments and examinations
  - Supplementary **Quick Recap** modules to revise C language and related topics in Week 0. These are numbered serially as `QRn`
    - These may be used to recapitulate C programming, as needed
    - These are not directly part of the syllabus, but cover the prerequisites. So their understanding are critical for the main modules. Those who know, may skip
  - **Tutorials** to build skills in C / C++ programming. These are numbered serially as `Tnn`
    - Some tutorials are of *Complementary* nature. These talk about various aspects of program development, program building, programming practices, etc. that may help to develop software using C / C++
    - Remaining tutorials are of *Supplementary* nature. These talk about additional information about C / C++ like how to mix these language, what is their compatibility etc.
    - Tutorials are not part of the syllabus. These are included for developing all-round skills for those who desire so

Course Outline: Modules

Week	Topic	
Week 01	**Programming in C++ is Fun**: *Introduction & Overview*	C++98/03
Week 02	**C++ is Better C**: *Procedural Extensions of C*	
Week 03	**OOP in C++/1**: *Classes and Encapsulation*	
Week 04	**OOP in C++/2**: *Overloading, namespace, struct & union*	
Week 05	**Inheritance**: *ISA & HAS_A in C++*	
Week 06	**Polymorphism**: *Binding, VFT, Multiple Inheritance*	
Week 07	**Type Casting**: *C++ cast operators*	
Week 08	**Exceptions & Templates**: try-throw-catch; *Meta-programming*	
Week 09	**Streams & STL**: *IO, Containers, Algorithms*	
Week 10	**Modern C++**: *C++11 and beyond – better C++, basic features*	C++11
Week 11	**λ & Concurrency**: *λ functions; threads, async call & mutex*	
Week 12	**Move, Rvalue & Containers**: *Move semantics; Summarization*	

Course Outline: Tutorials

- **Tutorials** are complementary or supplementary:
  - *Complementary* Tutorials introduce new ideas and skill areas to complement the understanding of the C/C++ languages. These include:
    - How to build a C/C++ program and / or static and dynamic libraries?
    - How to automate build using `make` utility?
    - What tools may be used to design, develop, test, and manage C / C++ software?
    - How to reuse?
      - binary (static or dynamic library)
      - code (template and meta-programming)
      - design (desing pattern)
    - and more
  - *Supplementary* Tutorials provide additional information and insight to supplement the understanding of the C/C++ languages. These include:
    - How to mix C/C++ in a single program?
    - What is the compatibility of C/C++?
    - What are the coding styles to write good C/C++ programs?
    - and more

Week 01: Programming in C++ is Fun

[1] Course Overview

- *Know Your C/C++*: Evolution & Comparison; Why learn C/C++?; C/C++ Standards
- *Know Your Course*: Objectives; Pre-requisites; Outline - Modules, Tutorials; Evaluation; Text Books & References; Tools

[2] IO and Loops in C and C++

- *Hello World: Handling IO*
- *Add Two Numbers and Handling IO*
- *Square Root*: `math Library`
- *C and C++ Standard Library Headers &* `std`: Header Conventions
- *Sum of* `n` *Numbers: Variable Declaration*
- *Using Boolean in C and C++*

[3] Arrays and Strings

- *Arrays &* `vectors`: Array Impl. for fixed size / arbitrary sized array, `vector`s in C++
- *C-Style Strings &* `string` *type in C++*: Concatenation of strings; More string operations; `string.h`; `string` class

[4] Sorting and Searching

- *Sorting in C and C++*: Bubble Sort, Using Standard Library
- *Searching in C and C++*: Using Standard Library
- *STL: algorithm - The* `algorithm` *Library*

[5] Stack and Common Data Structures / Containers

- *Stack in C*: Common Applications of Stack in C, Reverse a String, Evaluate Postfix Expressions
- *Stack in C++*: Reverse a String, Evaluate Postfix Expressions
- *Data Structures / Containers in C++*: Containers in C++

Week 02: C++ is Better C

[6] Constants and Inline Functions

- *cv-qualifier:* `const` & `volatile`: Notion & Advantages of `const`; `const` & pointer - C-String; Notion of `volatile`
- `inline` *functions*: Macros with Params in C - Pitfalls of Macros; Notion of `inline` - Comparison of Macros and `inline` functions, Limitations of `inline` functions

[7] Reference & Pointer

- *Reference Variable*: Pitfalls in Reference
- *Call-by-Reference*: Simple C Program to swap; Simple C/C++ Program to swap two numbers; `const` Reference Parameter
- *Return-by-Reference*: Pitfalls of Return-by Reference
- *I/O Parameters of a Function*
- *Recommended Call and Return Mechanisms*
- *Difference between Reference and Pointer*

[8] Default Parameters & Function Overloading

- *Default Parameters*: Examples; Highlights; Restrictions on default parameters
- *Function Overloading*: Examples; Restrictions; Rules
- *Overload Resolution*: Exact Match; Promotion & Conversion; Examples; Ambiguity
- *How to overload Default Parameter*

[9] Operator Overloading

- *Operators and Functions*: Difference
- *Operator Functions in C++*
- *Operator Overloading*: Advantages and Disadvantages
- *Examples*: String: Concatenation; Enum: Changing the meaning of `operator+`
- *Operator Overloading Rules*
- *Operator Overloading Restrictions*

[10] Dynamic Memory Management

- *Dynamic Memory Management in C*: `malloc` & `free`
- *Dynamic Memory Management in C++*: `new` & `delete` operator; Dynamic Memory Allocation for Array; Placement `new`; Restrictions
- *Operator Overloading for Allocation and De-allocation*

Week 03: OOP in C++/1

[11] **Classes and Objects**

- *Classes*
- *Objects*
- *Data Members*: `Complex`; `Rectangle`; `Stack`
- *Member Functions*: `Complex`; `Rectangle`; `Stack`
- `this` *Pointer*
- *State of an Object*: `Complex`; `Rectangle`; `Stack`

[12] **Access Specifiers**

- *Access Specifiers*: Examples
- *Information Hiding*
- *Stack Example*: Stack (`public`) - Risky; Stack (`private`) - Safe; Interface and Implementation
- *Get-Set Idiom*
- *Encapsulation*
- *Class as a Data-type*

[13] **Constructors, Destructors & Object Lifetime**

- *Constructor*: Contrasting with Member Functions; Parameterized - Default Parameters; Overloaded
- *Destructor*: Contrasting with Member Functions
- *Default Constructor*
- *Object Lifetime*: Automatic; Static; Dynamic; Storage Class Specifiers

[14] **Copy Constructor and Copy Assignment Operator**

- *Object Lifetime Examples*: `String`; `Date`: Practice; `Rect`: Practice; `Name` & `Address`: Practice; `CreditCard`: Practice
- *Copy Constructor*: Call by Value; Signature; Data Members; Free Copy Constructor and Pitfalls
- *Copy Assignment Operator*: Copy Objects; Self-Copy; Signature; Free Assignment Operator
- *Comparison of Copy Constructor and Copy Assignment Operator*
- *Class as a Data-type*

[15] `const`-**ness**

- *Constant Objects*: Simple Example
- *Constant Member Functions*: Simple Example
- *Constant Data Members*: Simple Example; Credit Card Example: Putting it all together - `String`, `Date`, `Name`, `Address`, `CreditClass`
- `mutable` *Members*: Simple Example; `mutable` Guidelines

Week 04: OOP in C++/2

[16] `static` **Members**

- `static` *Data Member*: Example; Print Task; Order of Initialization
- `static` *Member Function*: Print Task; Count Objects
- *Comparison*
- *Singleton Class*

[17] `friend` **Function and** `friend` Class

- `friend` *Function*: Matrix-Vector Multiplication; Linked List
- `friend` *Class*: Linked List; Iterator
- *Properties of* `friend`
- *Comparison*

[18] **Overloading Operator for User-Defined Types: Part 1**

- *Operator Function*: Non-Member Function; Member Function; Operator Overloading Rules
- *Using Global Function*: `public` data members; `private` data members
- *Using Member Function*: `operator+`; `operator=`; Unary Operators

[19] **Overloading Operator for User-Defined Types: Part 2**

- *Issues in Operator Overloading*
- `operator+`
- `operator==`
- `operator<<`, `operator>>`
- *Guidelines for Operator Overloading*

[20] `namespace`

- `namespace` *Fundamental*
- `namespace` *Scenarios*
- `namespace` *Features*: Nested `namespace`; `using namespace`; Global `namespace`; `std namespace`; `namespace` are Open
- `namespace` vis-a-vis `class`
- *Lexical Scope*

Week 05: Inheritance

[21] **Inheritance: Part 1: Inheritance Semantics**

- *ISA Relationship*
- *Inheritance in C++*: Phones; Semantics

[22] **Inheritance: Part 2: Data Member & Member Function: Override & Overload**

- *Inheritance in C++*
- *Data Members*: Object Layout
- *Member Functions*: Overrides and Overloads
- *Comparison*

[23] **Inheritance: Part 3: Constructor & Destructor: Object Lifetime**

- *Inheritance in C++*
- `protected` *Access*: Streaming
- *Constructor & Destructor*
- *Object Lifetime*

[24] **Inheritance: Part 4: Phone Hierarchy**

- *ISA Hierarchy Design by Inheritance*
- *Helper Classes*
- *Hierarchy of Phones by Interfaces*
- *Interfaces & State Variables of Phones*: Landline Phone; Mobile Phone; Smart Phone
- *Refactoring*
- *Hierarchy Integration*: Extended Hierarchy of Phones

[25] **Inheritance: Part 5: `private` & `protected` Inheritance**

- *Inheritance in C++*
- `private` *Inheritance*: Uncopyable; HAS_A
- `protected` *Inheritance*
- *Visibility*
- *Examples*

Week 06: Polymorphism

[26] **Polymorphism: Part 1: Type Casting**

- *Type Casting*: Basic Notions; Comparison of Implicit and Explicit Casting; Built-in Type - Promotion & Demotion; Unrelated Classes; Inheritance Hierarchy - Upcast, Downcast

[27] **Polymorphism: Part 2: Static and Dynamic Binding**

- *Type Binding*: Type of an Object; Static and Dynamic Binding; Comparison of Static and Dynamic Binding; Static Binding; Dynamic Binding
- *Polymorphic Type*

[28] **Polymorphism: Part 3: Abstract Base Class**

- *Virtual Destructor*: Slicing;
- *Pure Virtual Function*
- *Abstract Base Class*: Shape Hierarchy - Pure Virtual Function with Body

[29] **Polymorphism: Part 4: Staff Salary Processing using C**

- *Binding: Exercise*: Exercise 1; Exercise 2
- *Staff Salary Processing*: C Solution - Engineer + Manager, Engineer + Manager + Director, Advantages and Disadvantages

[30] **Polymorphism: Part 5: Staff Salary Processing using C++**

- *Staff Salary Processing*: C Solution: Flat C Solution: Recap - Advantages and Disadvantages
- *Staff Salary Processing*: C++ Solution: Non-Polymorphic Hierarchy - Advantages and Disadvantages; Polymorphic Hierarchy - Advantages and Disadvantages; Polymorphic Hierarchy (Flexible) - Advantages and Disadvantages

Week 07: Type Casting

[31] **Virtual Function Table**

- *Staff Salary Processing*: New C Solution
- *Staff Salary Processing*: C++ Solution
- *C and C++ Solutions*: A Comparison
- *Virtual Function Pointer Table*

[32] **Type Casting & Cast Operators: Part 1**

- *Type Casting*: Upcast & Downcast
- *Cast Operators*: `const_cast`

[33] **Type Casting & Cast Operators: Part 2**

- *Cast Operators*: `static_cast` - Built-in Types, Class Hierarchy, Hierarchy Pitfall, Unrelated Classes; `reinterpret_cast`

[34] **Type Casting & Cast Operators: Part 3**

- *Cast Operators*: `dynamic_cast` - Pointers, References
- `typeid` *Operator*: Polymorphic Hierarchy; Non-Polymorphic Hierarchy; `bad_typeid`
- *Run-Time Type Information (RTTI)*

[35] **Multiple Inheritance**

- *Multiple Inheritance in C++*: Semantics; Data Members and Object Layout; Member Functions - Overrides and Overloads; Access Members of Base: `protected` Access; Constructor and Destructor; Object Lifetime
- *Diamond Problem*: Exercise
- *Design Choice*

Week 08: Exceptions and Templates

[36] **Exceptions (Error handling in C): Part 1**

- *Exception Fundamentals*: Types of Exceptions; Exception Stages
- *Error Handling in C*: C Language Features - Return Value and Parameters, Local `goto`; C Standard Library Support - Global Variables, Abnormal Termination, Conditional Termination, Non-Local `goto`, Signals; Shortcomings

[37] **Exceptions (Error handling in C++): Part 2**

- *Exceptions in C++*: `try`-`throw`-`catch`; Exception Scope (`try`); Exception Arguments (`catch`); Exception Matching; Exception Raise (`throw`); Advantages; `std::exception`

[38] **Template (Function Template): Part 1**

- *What is a Template?*
- *Function Template*: Definition; Instantiation; Template Argument Deduction; Example
- `typename`

[39] **Template (Class Template): Part 2**

- *What is a Template?*: Recap
- *Function Template*
- *Class Template*: Definition; Instantiation; Partial Template Instantiation & Default Template Parameters; Inheritance

[40] **Functors: Function Objects**

- *Callable Entities*
- *Function Pointers*: Replace Switch / IF Statements; Late Binding; Virtual Function; Callback - `qsort`; Issues
- *Functors in C++*: Basic Functor; Simple Example; Examples from STL - Function Pointer, Functor without state, Functor with state

Week 09: Streams and STL

[41] **Input-Output: File Handling in C**

- *Standard Library for I/O*
- *Files and Streams*: File Open / Close
- *Formatted I/O*: Output; Read
- *Unformatted I/O*
- *Direct IO*
- *File Positioning*

[42] **Input-Output: Streams in C++**

- *Features of C++ I/O*
- *Streams*
- *Stream Output*
- *Stream Input*
- *File I/O*
- *Type-safe I/O*
- *Unformatted I/O*
- *Stream Manipulators*
- *Stream States*: Format States; Error States
- *Standard I/O Library*

[43] **C++ Standard Library: Part 1 (Generic Programming)**

- *Standard Library*: C Standard Library; C++ Standard Library - `std`, Header Conventions
- *Generic Programming*: Common Tasks; Lifting Example; Algorithms-Iterators-Containers Model; Examples

[44] **C++ Standard Library: Part 2 (STL)**

- *The STL*: Policy Parameterization
- *Common Standard Library Components*: `vector`; `list`; `map`; `set`

[45] **C++ Standard Library: Part 3 (STL)**

- *Data Structures / Containers in C++*: Containers in C++
- `algorithm` *Component*: `copy`
- `numeric` *Component*: `accumulate`; `inner_product`
- `functional` *Component*

Week 10: Modern C++

[46] **C++11 and beyond: General Features: Part 1**

- *Major C++11 Features*
- `auto` & `decltype`: `auto`; `decltype`
- *Suffix Return Type*: `decltype(auto)`: C++14

[47] **C++11 and beyond: General Features: Part 2**

- *Initializer Lists*: `initializer_list`; Overload Resolution; {}-Initializers and `auto`
- *Uniform Initialization: Syntax and Semantics*: Syntax; Semantics
- *Range-for Statement*

[48] **C++11 and beyond: General Features: Part 3**

- `constexpr`: Evaluate constant expressions at compile-time
- `noexcept`: To prevent Exception Propagation
- `nullptr`: `null` Pointer Literal
- `inline namespace`: Efficient Version Mgmt.
- `static_assert`: Compile-time Assertions
- *User-defined Literals*: Closer to Built-in Types
- *Digit Separators and Binary Literals*
- *Raw String Literals*
- *Unicode Support*
- *Memory Alignment*
- *Attributes*

[49] **C++11 and beyond: General Features: Part 4**

- *Copying vs. Moving*: Return Value; Append Full Vector; Swap; Deep vs. Shallow Copy; Performance Test
- *Rvalue References and Move Semantics*: Rvalue References; Copy vs. Move - Lvalue vs. Rvalue, Vector
- *Implementing Move Semantics*

[50] **C++11 and beyond: General Features: Part 5**

- *Recap of Copy vs. Move and related Concepts*
- *Move Semantics: How to code?*: Simple Move Constructor and Assignment; Challenges; Solution
- `std::move`: Use; Implementation
- *Move Semantics Project*: `ResMgr` Class; `MyResource` Class; `MyClass` Class

Week 11: λ, Classes, and Templates

[51] **C++11 and beyond: General Features: Part 6**

- *Universal References*: Recap; `T&&` is Universal Reference; `auto` is Universal Reference; Rvalue vs. Universal References
- *Perfect Forwarding*: Type Safety; Practice Examples
- `std::forward`
- *Move is an Optimization of Copy*: Compiler Generated Move

[52] **C++11 and beyond: General Features: Part 7**

- *λ in* **C++11**, **C++14**, **C++17**, **C++20**: Syntax and Semantics; Closure Object - λs vs. Closures, First Class Object, Anatomy; Parameters; Capture - By Reference `[&]`, By Value `[=]`, Mutable, Restrictions, Practice Examples

[53] **C++11 and beyond: General Features: Part 8**

- *λ in C++*: Recap
- `std::function`: Examples
- *Generic λ in* **C++14**
- *Recursive λ in C++*: Practice Examples; Generic Recursive λ - Practice Examples
- *Generalized λ Captures*

[54] **C++11 and beyond: Class Features**

- `=default` / `=delete` *Functions*
- *Control of default* `move` *and* `copy`: Compiler Rules; User Guidelines
- *Delegating Constructors*
- *In-class Member Initializers*
- *Inheriting Constructors*
- *Override Controls*: `override`; `final`
- `explicit` *Conversion Operators*: `bool`

[55] **C++11 and beyond: Non-class Type & Template Features**

- *Other (non-class) Types*: `enum class` - Scope, Underlying Type, Forward-Declaration; Integer Types; Generalized unions; Generalized PODs
- *Templates*: Extern Templates; Template aliases; Variadic templates - Practice Examples; Local types as template arguments; Right-angle brackets (Nested Template Closer); Variable templates

Week 12: Move, Rvalue and STL Containers

[56] **C++11 and beyond: Resource Management by Smart Pointers: Part 1**

- *Raw Pointers*: Operations; Ownership Issue; Pointers vs. Reference
- *Smart Pointers*: Policies - Storage Policy, Ownership Policy

[57] **C++11 and beyond: Resource Management by Smart Pointers: Part 2**

- *Smart Pointers*: Recap; Ownership Policy; Conversion Policy; Null-test Policy
- *Resource Management*: `std::unique_ptr`; `std::shared_ptr`; `std::weak_ptr`; `std::auto_ptr`; Summary of Smart Pointer Operations; Binary Tree
- *Recommendations for Smart Pointers*

[58] **C++11 and beyond: Concurrency: Part 1**

- `thread` *Programming in C++*: `std::thread`; `std::bind`
- *Race Condition & Data Race*: Race Condition Example - Solution by Mutex, Solution by Atomic

[59] **C++11 and beyond: Concurrency: Part 2**

- *Threads*
- *Race Condition and Data Race*: Solution by Mutex; Solution by Lock; Solution by Atomic; Solution by Future; Solution by Async
- *Synchronization*: Thread Local
- *Self-Study*: Mutual Exclusion; Locks - Deadlock; Atomics; Sync: Condition Variables; Sync: Futures and Promises; Async; Practice Examples

[60] **Closing Comments**

- *Course Summary*
- *Modern C++ Features*: C++11 Features; C++14 Features; Deprecated Features
- *Key Take-back*: Prepare for Examination
- *Road Forward*

Tutorials

- **Tutorial 01**: How to build a C/C++ program?: Part 1: C Preprocessor (CPP)
- **Tutorial 02**: How to build a C/C++ program?: Part 2: Build Pipeline
- **Tutorial 03**: How to build a C/C++ program?: Part 3: `make` Utility
- **Tutorial 04**: How to build a C/C++ program?: Part 4: Static and Dynamic Library
- **Tutorial 05**: Mixing C and C++ Code: Part 1: Issues and Resolutions
- **Tutorial 06**: Mixing C and C++ Code: Part 2: Project Example
- **Tutorial 07**: How to design a UDT like built-in types?: Part 1: Fraction UDT
- **Tutorial 08**: How to design a UDT like built-in types?: Part 2: Int & Poly UDT
- **Tutorial 09**: How to design a UDT like built-in types?: Part 3: Updates and Mixes of UDTs
- **Tutorial 10**: How to optimize **C++11** programs using Rvalue and Move Semantics?
- **Tutorial 11**: Compatibility of C and C++: Part 1: Significant Features
- **Tutorial 12**: Compatibility of C and C++: Part 2: Summary

Module Summary

- We have discussed the course outline in terms of modules and tutorials
- Critical Actions before the Course starts
  - Revise C
  - Revise basic Data Structures (array, stack, queue, priority queue)
  - Revise Algorithms (sorting and searching, matrix-vector, graph)
  - Install gcc and gdb and try out several C programs with it
  - Suggest tutorials if you feel the need