- generic programming - algorithms written with generic types that are going to be specified later
  - reuse code

instantiation = generating an actual function from template function
template = skeleton
  - when instantiating a template → compiler creates a new class with the given template

- container = class designed to hold and organise multiple instances of another type
  - sequence containers: vector, dequeue, list
  - associative containers: set, multiset, map, multimap
  - derived containers: stack, queue, priority-queue

- iterator = object that can traverse (iterate over) a container class without the user having to know how the container is implemented

- stream = sequence of bytes flowing in/out of a program
  - abstraction for receiving/sending data
  - intermediate between programs and actual IO devices
  - input stream: hold input from data producer
  - output stream: hold output for particular data consumer
  - ~ serial: data must be sent/received one at a time/serial fashion
    random access not possible

insertion operator << : writing operations on a stream
  - operand from left → object from ostream class

extraction operator >> : reading on a stream
  - operand left → istream class object

manipulators = functions designed to be used in conjuction with insertion and extraction operators on stream objects
  - affect the way elements are displayed/read ⇒ formatting parameters

files i/o = DS stored on disk device
  - we must connect a stream to the file on the disk
    - opening the file
  - EOF

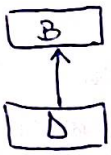
buffer = memory block that acts as an intermediary between stream and destination
  - buffer written to the disk → flushing

- polymorphism: ← respond in different ways to the same message
  virtual function: the most derived function that exists between the base and derived class
  derived function = match ⇔ same signature and return type
  override: specifier for the function in the derived class that is overriding
  ~ destructor → virtual (base class)
  ! constructors can't be virtual
* early (static) binding: directly associate identifier name with machine address
* late (dynamic) binding: pointers to functions, virtual functions] reference and pointers
- virtual table = special form of late binding
  - each virtual function → entry in virtual table
    function pointer that points to the most-derived function accessible by that class
    slows
    "=0" makes a function called
  - compiler adds: hidden pointer to base class: vptr
  - pointer to virtual table (vptr) is added to base class (and inherited later)
- object slicing ↔ assign derived object to base object
- pure virtual function → has no body
  ↳ placeholder meant to be redefined by derived classes
- abstract class ┌ has ≥ 1 pure virtual functions
  virtual table: nullptr ├ can't be instantiated
  └ derived classes must define virtual functions, otherwise they become abstract base class
- interface class – no variables, only pure virtual functions
- errors: throw: signals an exception
  try: marks instruction block that might raise problems
  catch: code to handle error
  noexcept → destructors
- lambda expression = anonymous function inside other function
  [capture closure] (parameters) { function body ; }
  ↓ gives access to variables available in the surrounding scope
  value =; reference &
  ↓ list all variables we want to access from within the lambda

- reference = alias for a variable
  same memory address as the original variable
- oop features: abstraction: separate an object specifications from its implementation
  polymorphism: an object can be of several types
  inheritance: hierarchy of "is a" relationships → reuse of code
  encapsulation: binds together data and functions → keeps safe from outside interference and misuse
- access modifiers: public: fields/methods → any objects of any class
  protected: within the class / child or derived classes
  private: within the class and (friend functions, classes)

	public	protected	private
class	yes	yes	yes
derived class	yes	yes	no
client code	yes	no	no

- this = pointer to current instance
- static functions: class_name:: characteristic of a class, doesn't depend on objects
  can access other static data members / functions + functions outside class
  don't have access to THIS
- overloading (functions) – same name, diff parameters
- rule of three: destructor, copy constructor, assignment operator
- inheritance: general → specific
  new class: all feature of old class, + feature of its own



overriden membra (defined in B, D) → programming by difference
↓
in defining derived classes we only specify what's different

- friend elements: private and protected
  not transitive, and not corresponded
  public: private → hidden
  protected → protected
  public → public
  protected: private → hidden
  protected → protected
  public → protected
  private: private → hidden
  protected → hidden
  public → hidden

~ delegation: derived member delegates part of its duty to the base class

~ ferial: if we don't want to inherit from a class

NOT INHERITED:
- default contructor
- parameter contructor
- copy contructor
- destructor
- assignment operator

⇓

INVOCATION → constructor of derived class INVOKES the contructor of the base class

- destructor derived class, destructor base class

I II

pointers - value = memory location of another variable
& take the address of variable
* get value at memory address pointed to

pass by value → makes a copy
pass by address → with pointers
void pointer = no associated data type
malloc + calloc

const int *p
const data ↑ can't change
changeable pointer ↑ can point to diff const value

int * const p
changeable data ↑ can change
const pointer ↑ can't point to diff mem loc

const int * const p
const data const pointer

- array = pointer to the first element of the array
- function pointer void (* func_pointer)(int)

stack → continuous block of memory
LIFO; data associated with one function call (local); limited size
heap → can be controlled dynamically
no size restriction (global)
uninitialized data → global + static variables initialized to 0 or don't have explicit initialization
initialized data → global + static initialized variables

STACK
- very fast access
- don't have to de-allocate
- memory won't become fragmented (CPU manages efficiently)
- local variables
- limit on stack size
- variables can't be resized

HEAP
- slower access
- must manage memory
- memory may become fragmented
- variables accessed globally
- no limit on memory size
- variables can be resized (realloc)

- modular programming ← module: collection of functions, variables that implement a well defined functionality
  - abstraction = essential features, ignores details
  - cohesion = single responsibility, functions related
  - layered architecture ← clarity, separation
    reusability, flexibility, independence

- L-value = what's on the left of an assignment expression
  have assigned memory addresses
  R-value = everything that's not an L-value
  expression scope = they die at the end of the expression
  reference: && → extend lifespan of object
- move constructor, move assignment } the argument for construction/assignment is an r-value
  move ownership of the resources from one object to another
  disable copy constructor '= delete'
  ↓
  steal resources held by the argument
  rule of 5 ← copy constructor
  copy assignment operator
  copy destructor
  copy move constructor
  copy assignment operator
- RAII ← Resource Acquisition Is Initialization
  resource is tied to the lifetime of objects which acquires it
  ↓
  memory location, database connection
  file, network socket
  └ constructor acquires, destructor releases
  ~ avoid resource leaks, exception-safe code
  → automatic management for different kinds of resources
  → lifetime of objects allocated on the stack - managed by constructor
- smart pointers = composition class designed to manage dinamically allocated memory
  std::unique_ptr → any dynamically allocated object not shared
  std::shared_ptr → multiple smart pointers co-owning a resource; cyclic dependencies
  std::weak_ptr → solve cyclic dependency; not considered owner
- program = code stored on disk/non-volatile memory
  process = program loaded into memory along with all the resources
  └ each process has separate memory address space (run independently)
  register → data storage locations part of CPU
  program counter / instruction pointer → keeps track of current location
  thread = unit of execution within a process
  shares memory and resources
  each have own stack; share heap
  CPU → uses scheduling, giving the illusion of parallel execution
  parallelism = simultaneous execution
  concurrency = interleaving of processes in time to give the appearance of simultaneous execution

- synchronization — process synchronization : multiple processes are to join up / handshake at a certain point in order to reach an agreement
  data synchronization : multiple copies of a dataset in coherence with one another
- mutual exclusion — prevent multiple threads from simultaneously accessing shared resources

QT

- event loop = ∞ loop works in background and handles events incoming from OS
- QApplication → initializes the app with user's desktop settings
  event handling

signals — emitted when a particular event occurs
slot — function called in response to a particular signal) connected