# **Chapter 1**

|  |  |
| --- | --- |
| affordability | Whether something is affordable, within price range. |
| analysis | Thinking about the goal of the software, what the user wants and needs and can afford, and how much reliability is needed |
| blackboard | Design slate. |
| CAD/CAM | Computer-aided design / computer-aided manufacture |
| communication |  |
| correctness | That the program does what it is supposed to do. |
| customer | Who will order or buy the software. |
| design | How the software is structured. |
| feedback | Reactions from users on the usage of the software. |
| GUI | Graphical user interface |
| ideals | Striving for correctness, reliability, well-designed, affordable, and maintainable. |
| implementation | Programming + testing |
| programmer | Person who programs |
| programming | ‘Express the solution to the problem (the design) in code. Write the code in a way that meets all constraints (time, space, money, reliability, and so on). Make sure that the code is correct and maintainable.’ |
| software | The programs that tell a computational device how to behave |
| stereotype |  |
| testing | Testing of all use-cases of a program |
| user | Person who will use the software |

# **Chapter 2**

|  |  |
| --- | --- |
| // | Comment marker |
| << | Output operator |
| C++ | Programming language |
| comment | Comment explaining what the code is supposed to be doing, for the benefit of the programmer who reads the code |
| compiler | Turns the source code into machine code |
| compile-time error | Error during compiling |
| cout | [see-out] Character output stream |
| executable | Program which can be run on a computer |
| function | Small part of code that performs some operation, possibly with in- and/or output. |
| header | ‘a file containing declarations used to share interfaces between parts of a program’ |
| IDE | Interactive development environment |
| #include | Directive that tells the compiler to include certain files |
| library | Code that can be accessed using declarations found in an #included file. |
| linker | Links separate parts of machine code into executable. |
| main() | Function where the program starts executing. |
| object code | The compiled source code that the machine can read. Also: machine code |
| output | What is written to the screen (or possibly files). |
| program | Instructions for computer/executable. |
| source code | The C++ code source files that the programmer writes. |
| statement | Line of code that specifies an action is not an #include directive. |

# **Chapter 3**

|  |  |
| --- | --- |
| assignment | Setting a variable to hold a certain value. |
| cin | [see-in] character input stream |
| concatenation | Combining strings into larger string |
| conversion | Changing value into another type |
| declaration | Statement that gives a name to an object. |
| decrement | Reduce value. |
| definition | A declaration that sets aside memory for an object. |
| increment | Add to value. |
| initialization | Declare object and give value to the object. |
| name | Identifier of object. |
| narrowing | Transforming a datatype with a certain range into one with a smaller range. |
| object | Some memory that holds a value of a given type. |
| operation | Action |
| operator | Action specifier |
| type | Defines a set of possible values and a set of operations for an object. |
| type safety | Whether objects are used only according to the rules for their type and only if they have been initialized. |
| value | What the variable refers to. |
| variable | A named object. |
| assignment | Setting a variable to hold a certain value. |

# **Chapter 4**

|  |  |
| --- | --- |
| abstraction | Hide implementation details behind a convenient and general interface |
| begin() | Function of vector object pointing to the first element. |
| computation | Calculation |
| conditional statement | Statement that is executed if a certain condition is met |
| declaration | Declaring variable name for object |
| definition |  |
| divide and conquer | Take large problem and divide it into several little ones. |
| else | Other branch of if-statement, what to do if if-condition is not met |
| end() | Function of vector object pointing to the first element. |
| expression |  |
| for-statement | for (int i = 0; i<100; ++i) |
| range-for-statement | for (int x : temps) : x is an element of temps vector |
| function | Small part of code that computes something (can have in- and output) |
| if-statement | If (condition is met) { do stuff; } |
| increment | Add to value |
| input | What goes into a function |
| iteration | One cycle of multiple |
| loop | Repeated action |
| lvalue | Value of left hand of the expression |
| member function | Function of class / object |
| output | What comes out of the function/ is printed to console. |
| push\_back() | Function of vector to add element to the back of the vector |
| repetition |  |
| rvalue | Value of right hand side expression |
| selection | Choose among possible actions |
| size() | Function of vector class to get the size of the vector |
| sort() | Function of vector class to sort the vector |
| statement |  |
| switch-statement | switch (unit) {  case ‘x’:  # do stuff  case ‘y’:  # do other stuff |
| vector | (extendable) array/ list of objects |
| while-statement | while (condition is met){ do stuff } |

# **Chapter 5**

|  |  |
| --- | --- |
| argument error | Error in type of argument |
| assertion | Test if assumption on variable is correct |
| catch | Block of code to execute when error is thrown in try {} block |
| compile-time error | Error that is found by the compiler during compiling. |
| container |  |
| debugging | ‘The act of searching for and removing errors from a program; usually far less systematic than testing.’ |
| error | ‘a mismatch between reasonable expectations of program behavior (often expressed as a requirement or a users’ guide) and what a program actually does.’ |
| exception | Unexpected occurrence in running the program that you (hopefully) handle. |
| invariant | ‘A rule for what constitutes a valid value´ |
| link-time error | Error during linking all the code to an .exe file. |
| logic error | Error when code results in wrong answer. |
| post-condition | Conditions that should hold at the end of the function being called. |
| pre-condition | Conditions that should hold at the start of the functions being called (concerning the inputs). |
| range error | Trying to access an element that cannot be reached (might not be defined). |
| requirement | 1. a description of the desired behavior of a program or part of a program; 2. a description of the assumptions a function or template makes of its arguments |
| run-time error | Error that happens during running of the program. |
| syntax error | Mistake in writing of source code. |
| testing | Trying out the software rigorously with different use cases and inputs. |
| throw | Recognize an error being made and handing that status to calling program. |
| type error | Error in type of variable supplied. |

# **Chapter 6**

|  |  |
| --- | --- |
| analysis | Thinking about problem and how to handle it. |
| class | Type of object |
| class member | Part of class |
| data member | Attribute of class |
| design | Idea of how to implement. |
| divide by zero |  |
| grammar | Rules for processing input. |
| implementation | The code of how the task is performed. |
| interface | Accessible functions and attributes of the class. |
| member function | Function of the class. |
| parser | Program that reads in a stream of tokens according to a grammar. |
| private | Not accessible for users of the class. |
| prototype | Initial not yet fully tested version |
| pseudo code | Words describing what the code should do |
| public | Accessible to users of the class |
| syntax analyzer | Other word for parser |
| token | Item that represents an operand or operation |
| use case | Possible case in which the program could be used. |

# **Chapter 7**

|  |  |
| --- | --- |
| code layout | Layout of the code |
| commenting | Text for programmers to describe what the code does for non-obvious things |
| error handling | How the program deals with errors during execution |
| feature creep | Trying to |
| maintenance | Keeping the code correct, and pretty. |
| recovery | Returning to ‘normal’ state after an error has occurred. |
| revision history | The history of changes to code. |
| scaffolding | Include statements, the main() function structure. |
| symbolic constant | Variable with name and constant literal value |
| testing | Trying out the code. |

# **Chapter 8**

|  |  |
| --- | --- |
| activation record | Data structure containing a copy of all its parameters and local variables. |
| argument | Parameter |
| argument passing | Giving the parameter to function. |
| call stack | Stack of activation records |
| class scope | Scope of the class |
| const | Keyword for constant variable |
| constexpr | Keyword for constant compile-time evaluated variable |
| declaration | Introduction of variable |
| definition | Assigning a value to a variable |
| extern | Keyword for external variable not defined in current code |
| forward declaration | Declaration so that something else can refer to it, even though the definition is not in the code above the call to the function. |
| function | Modular code that does something, possibly with and/or output |
| function definition | The body of the function, what the code does |
| global scope | The total scope |
| header file | File with function declarations |
| initializer | The code that sets the initial value |
| local scope | Scope between {} |
| namespace | Set of functions/classes organized |
| namespace scope | Scope of the namespace |
| nested block | Nested {} |
| Parameter | Object |
| pass-by-const-reference | Const <type>& <name> |
| pass-by-reference | <type>& <name> |
| pass-by-value | <type> <name> |
| Recursion | Calling within calling |
| Return | To give a value to the code that calls a function |
| return value | Output of function |
| Scope | ‘Area’ in the code where a variable is defined |
| statement scope | Scope for example a for or while loop |
| technicalities | Syntax rules and such |
| undeclared identifier | Error when for example a namespace or function has not been declared |
| using declaration | “using std::string” |
| using directive | “using namespace std;” |

# **Chapter 9**

|  |  |
| --- | --- |
| built-in types | Standard types |
| class | Collection of data members and member functions as logical unit. |
| const | Constant |
| constructor | Function for creation of class object |
| destructor | Function for destruction of class object |
| enum | Enumeration function to link labels to number value |
| enumeration | An enum (an enumeration) is a very simple user-defined type, specifying its set of values (its enumerators) as symbolic constants. |
| enumerator | Set of values in enumeration |
| helper function | Function that implements a function for a class, but is defined outside the class |
| implementation | (1) the act of writing and testing code; (2) the code that implements a program. |
| in-class initializer | An initializer for a class member specified as part of the member declaration |
| inlining | Declaring a function within a class (in-line) |
| interface | The public functions of a class |
| invariant | The rule for what values are valid for data members. |
| representation | How the data is represented |
| struct | Structure of data with default public members |
| structure | How data is organized |
| user-defined types | Types defined by a or the user |
| valid state | Allowed states for objects/variables |

# **Chapter 10**

|  |  |
| --- | --- |
| bad() | Unexpected and serious state |
| buffer | Temporary storage space between istream or ostream and “somewhere” |
| clear() | Function to take the stream out of the fail() state into the good() state |
| close() | Function to close in or output file |
| device driver | Intermediate between in/output device and in/output library |
| eof() | End of file state |
| fail() | Unexpected state |
| file | Stored data |
| good() | Actions succeeded state |
| ifstream | Input stream from file |
| input device | Any device that can receive input |
| input operator | >> |
| iostream | In or output stream |
| istream | Input stream |
| ofstream | Output stream from file |
| open() | Function to open in or output file |
| ostream | Output stream |
| output device | Any device that can give output |
| output operator | << |
| stream state | State of the istream or ostream |
| structured file | File with formatted data |
| terminator | Character to end input reading with |
| unget() | Put back character into stream |

# **Chapter 11**

|  |  |
| --- | --- |
| binary | Using bitwise representation |
| character classification | Type of character: digit/alphanumeric/uppercase/space etc. |
| decimal | Base 10 (0 1 2 3 4 5 6 7 8 9) |
| defaultfloat | Default float representation |
| file positioning | Setting the position to read/write from in a file |
| fixed | Having the number of digits fixed in output |
| hexadecimal | Using 16 base (0 1 2 3 4 5 6 7 8 9 a b c d e f) |
| irregularity | Opposite of regularity |
| line-oriented input | Using getline() function for |
| manipulator | A term that is used to change the behavior of a stream |
| nonstandard separator | Non-white space separator |
| noshowbase | Refrain from showing the base |
| octal | Base 8 (0 1 2 3 4 5 6 7) |
| output formatting | Format of the output |
| regularity | “treating all in-memory objects uniformly, treating all input sources equivalently, and imposing a single standard on the way to represent objects entering and exiting the system” |
| scientific | Setting the output to scientific representations |
| setprecision() | Function that sets the precision for displaying value output |
| showbase | Displays a prefix for a value signifying the base of the value |

# **Chapter 14**

|  |  |
| --- | --- |
| abstract class | Class that cannot be instantiated |
| access control | Determining who can call/access functions and members with public, protected and private keywords |
| base class | Parent class or superclass |
| derived class | Child class or subclass |
| dispatch | The function is called determined at run time based on the type of the object used |
| encapsulation | “protecting something meant to be private (e.g., implementation details) from unauthorized access” |
| inheritance | Having classes being types of other (super)classes, with more or other members and or implementations of functions |
| mutability | The possibility to change (member) values of a class |
| object layout | The layout of an object (how the members are stored in memory) |
| object-oriented override | Having child/derived classes have their own version of a function declared/defined in the parent/base class |
| polymorphism | Having different subclasses have different function implementationss |
| private | Only accessible by the members of the class in which it is declared |
| protected | Only accessible by the members of the class in which it is declared and members of classes derived from that |
| public | Accessible by all functions |
| pure virtual function | Virtual function defined as “=0” |
| subclass | Derived class or child class |
| superclass | Base class or parent class |
| virtual function | “the ability to define a function in a base class and have a function of the same name and type in a derived class called when a user calls the base class function” |
| virtual function call | Call to virtual function |
| virtual function table | References to the function to call for the object when function could be overriding a virtual function |

# **Chapter 15**

|  |  |
| --- | --- |
| approximation | Often easier way to compute the actual analytical value |
| default argument | Standard value of argument used when no other input is given |
| function |  |
| lambda | From “lambda expression” for an inline function like:  [](double x) { return cos(x)+slope(x); } |
| scaling |  |
| screen layout |  |

# **Chapter 17**

|  |  |
| --- | --- |
| address |  |
| address of: & |  |
| allocation |  |
| cast |  |
| container |  |
| contents of: \* |  |
| deallocation a |  |
| delete |  |
| delete[] |  |
| dereference |  |
| destructor |  |
| free store |  |
| link |  |
| list |  |
| member access: –> |  |
| member destructor |  |
| memory |  |
| memory leak |  |
| new |  |
| null pointer |  |
| nullptr |  |
| pointer |  |
| range |  |
| resource leak |  |
| subscripting |  |
| subscript: [ ] |  |
| this |  |
| type conversion |  |
| virtual destructor |  |
| void\* |  |

# **Chapter 18**

|  |  |
| --- | --- |
| array |  |
| array initialization |  |
| copy assignment |  |
| copy constructor |  |
| deep copy |  |
| default constructor |  |
| essential operations |  |
| explicit constructor |  |
| move assignment |  |
| move construction |  |
| palindrome |  |
| shallow copy |  |

# **Chapter 19**

|  |  |
| --- | --- |
| #define |  |
| at() |  |
| basic guarantee |  |
| exception |  |
| guarantees |  |
| handle |  |
| instantiation |  |
| macro |  |
| owner |  |
| push\_back() |  |
| RAII |  |
| resize() |  |
| resource |  |
| re-throw |  |
| self-assignment |  |
| shared\_ptr |  |
| specialization |  |
| strong guarantee |  |
| template |  |
| template parameter |  |
| this |  |
| throw; |  |
| unique\_ptr |  |