# ****Chapter 1****

1. What is software?
2. Why is software important?
3. Where is software important?
4. What could go wrong if some software fails? List some examples.
5. Where does software play an important role? List some examples.
6. What are some jobs related to software development? List some.
7. What’s the difference between computer science and programming?
8. Where in the design, construction, and use of a ship is software used?
9. What is a server farm?
10. What kinds of queries do you ask online? List some.
11. What are some uses of software in science? List some.
12. What are some uses of software in medicine? List some.
13. What are some uses of software in entertainment? List some.
14. What general properties do we expect from good software?
15. What does a software developer look like?
16. What are the stages of software development?
17. Why can software development be difficult? List some reasons.
18. What are some uses of software that make your life easier?
19. What are some uses of software that make your life more difficult?

# ****Chapter 2****

1. What is the purpose of the “Hello, World!” program?  
   To print the sentence “Hello, World” out on the console.
2. Name the four parts of a function.  
   A return type, a name, a parameter list enclosed in parentheses and a function body enclosed in curly braces.
3. Name a function that must appear in every C++ program.  
   The main function.
4. In the “Hello, World!” program, what is the purpose of the line return 0;?  
   To return 0 if the function body has been successfully completed.
5. What is the purpose of the compiler?  
   To ‘compile’ the source code to machine code which the computer can run.
6. What is the purpose of the #include directive?  
   ‘It instructs the computer to make available (“to include”) facilities from the file after the #include directive.’
7. What does a .h suffix at the end of a file name signify in C++?  
   That the file is a header file.
8. What does the linker do for your program?  
   The linker links object code files (from libraries or local files) together into the executable program.
9. What is the difference between a source file and an object file?  
   Source files are the instruction files that contain the literal c++ code, object files are the compiled files which are platform dependent.
10. What is an IDE and what does it do for you?  
    IDE stands for interactive development environment, and it helps with programming by providing an environment in which to code, build and debug, possibly with syntax coloring etc.
11. If you understand everything in the textbook, why is it necessary to practice?  
    Because practice is not the same as the theory and programming needs practice.

# ****Chapter 3****

1. What is meant by the term *prompt*?  
   Waiting for user input.
2. Which operator do you use to read into a variable?  
   cin >>
3. If you want the user to input an integer value into your program for a variable named number, what are two lines of code you could write to ask the user to do it and to input the value into your program?  
    int number;  
    cout << “Please enter a number:\n”;  
    cin >> number;
4. What is \n called and what purpose does it serve?  
   \n is called a newline, it serves to move the cursor the next line.
5. What terminates input into a string?  
   A whitespace or newline
6. What terminates input into an integer?  
   A whitespace or newline
7. How would you write

cout << "Hello, ";  
cout << first\_name;  
cout << "!\n";

as a single line of code?  
cout << "Hello, " << first\_name << "!\n";

1. What is an object?  
   Some memory that holds a value of a given type.
2. What is a literal?  
   A representation of a value of a certain type (for instance integers, strings or Boolean values).
3. What kinds of literals are there?  
   Chars and strings
4. What is a variable?  
   A named object.
5. What are typical sizes for a char, an int, and a double?  
   char:1 byte , int: 4 bytes, double: 8 bytes.
6. What measures do we use for the size of small entities in memory, such as ints and strings?  
   Bytes (1 byte = 8 bits).
7. What is the difference between = and ==?  
   = is an assignment operator, == is a comparison operator
8. What is a definition?  
   A declaration that sets aside memory for an object.
9. What is an initialization and how does it differ from an assignment?  
   In an initialization you declare the variable; an assignment gives the variables value.
10. What is string concatenation and how do you make it work in C++?  
    Adding strings together to form a larger string, you use the ‘+’ operator for this.
11. Which of the following are legal names in C++? If a name is not legal, why not?  
    Legal ones : This\_litle, This\_little\_pig, This\_1\_is fine, \_this\_is\_ok, MiniMineMine, number  
    Not legal ones: 2\_For\_1\_special (starts with number), latest thing (contains space), the\_$12\_method (contains $, which is not a letter, digit or underscore), correct? (contains question mark)
12. Give five examples of legal names that you shouldn’t use because they are likely to cause confusion.  
    - single character variables unless in loop  
    - names with 1 or l where 1 and l look alike  
    - names starting with an underscore  
    - variable names xx1, xx2 etc
13. What are some good rules for choosing names?  
    Not too short but also not too long, clear what the variable holds, don’t start with underscore (might interfere with system variables)
14. What is type safety and why is it important?  
    Making sure that operations on variables are allowed, and will that the result of the operation is valid. This is important because you might encounter unexpected behavior/results otherwise.
15. Why can conversion from double to int be a bad thing?  
    Because a double can have larger values than an int.
16. Define a rule to help decide if a conversion from one type to another is safe or unsafe.  
    Check the value of the type to see if it fits in the range of the type to be converted to.

# ****Chapter 4****

1. What is a computation?   
   A calculation.
2. What do we mean by inputs and outputs to a computation? Give examples.   
   We mean what goes in and comes out of a computation.
3. What are the three requirements a programmer should keep in mind when expressing computations?  
   Correctness, simplicity, efficiency.
4. What does an expression do?   
   Compute a value from a number of operands.
5. What is the difference between a statement and an expression, as described in this chapter?   
   Statement does not have to compute something, whereas an expression does.
6. What is an lvalue? List the operators that require an lvalue. Why do these operators, and not the others, require lvalue?  
   lvalue is the left-hand side of an assignment. The assignment operator and the increment and decrement operators?
7. What is a constant expression?  
   A variable that cannot be changed.
8. What is a literal?  
   A literal value for an object.
9. What is a symbolic constant and why do we use them?  
   A constant variable, like pi or Avogadro’s constant.
10. What is a magic constant? Give examples.  
    A non-obvious literal in code.
11. What are some operators that we can use for integers and floating-point values?  
    = \* / == + - != > < >= <= >> << ++ --
12. What operators can be used on integers but not on floating-point numbers?  
    a % b (modulo)
13. What are some operators that can be used for strings?  
    = : assignment, + : concatenation, == : equals, != not equal, > / >= / < / <=
14. When would a programmer prefer a switch-statement to an if-statement?  
    When comparing against many constants,
15. What are some common problems with switch-statements?  
    Case labels need to be constants and distinct, and do not forget the “break;” statement.
16. What is the function of each part of the header line in a for-loop, and in what sequence are they executed?  
    for (int i=0; i < n; ++i) : initialization; stopping condition; incrementation
17. When should the for-loop be used and when should the while-loop be used?  
    for-loop : when the range is known, while-loop : when the range is not known
18. How do you print the numeric value of a char?  
    char(‘a’ + 1)
19. Describe what the line char foo(int x) means in a function definition.  
    Function “foo” takes as input argument an integer x, and outputs a char.
20. When should you define a separate function for part of a program? List reasons.  
    For conciseness, reusability etc.
21. What can you do to an int that you cannot do to a string?  
    division, multiplication, subtraction, increment, decrement, remainder
22. What can you do to a string that you cannot do to an int?  
    Concatenate
23. What is the index of the third element of a vector?  
    2
24. How do you write a for-loop that prints every element of a vector?  
    for (size\_t j = 0; j<N; ++j){ for (int x : v){  
     cout << v[j]; cout << x;  
    } }
25. What does vector<char> alphabet(26); do?  
    Initialize a vector of chars named alphabet with length 26.
26. Describe what push\_back() does to a vector.  
    v.push\_back(el) adds element el to vector v.
27. What do vector’s member functions begin(), end(), and size() do?  
    point to beginning element, end element and size of the vector.
28. What makes vector so popular/useful?  
    A vectors stores a list of data
29. How do you sort the elements of a vector?  
    sort(v.begin(), v.end()) where v is the vector

# ****Chapter 5****

1. Name four major types of errors and briefly define each one.  
   Compile-time errors: found by compiler, to do with syntax/types  
   Link-time errors: found by compiler when trying to make executable: libraries/functions not found/ exe still running  
   Run-time errors: errors during the running of the program, for instance, exceptions, hardware/software issues  
   Logic errors: mistakes in implementation / wrong result errors
2. What kinds of errors can we ignore in student programs?  
   Misbehaving hardware, system software errors
3. What guarantees should every completed project offer?  
   Should produce the desired results for all legal inputs, should give reasonable error messages for all illegal inputs.
4. List three approaches we can take to eliminate errors in programs and produce acceptable software.  
   Minimum of errors by debugging and testing, non-serious remaining errors.
5. Why do we hate debugging?  
   Because it is not so easy.
6. What is a syntax error? Give five examples.
7. What is a type error? Give five examples.
8. What is a linker error? Give three examples.
9. What is a logic error? Give three examples.
10. List four potential sources of program errors discussed in the text.  
    Poor specification, incomplete programs, unexpected arguments, unexpected input, unexpected state, logical errors.
11. How do you know if a result is plausible? What techniques do you have to answer such questions?  
    You can add post-conditions that include only the plausible values, you can test the function with inputs for which you know the answer.
12. Compare and contrast having the caller of a function handle a run-time error vs. the called function’s handling the runtime error.  
    It is easier for the called function to handle the error, since it knows how the function is implemented.
13. Why is using exceptions a better idea than returning an “error value”?  
    Recognizable “error value” may not exist, you need to check for the error value yourself, instead of an exception that is thrown, and try {} catch {} can be used to handle the exception.
14. How do you test if an input operation succeeded?  
    Check if the function returns the output for successful operation.
15. Describe the process of how exceptions are thrown and caught.
16. Why, with a vector called v, is v[v.size()] a range error? What would be the result of calling this?  
    Because the index starts at 0, the last element is v[v.size()-1]. An access violation exception will be thrown, since the specified element cannot be accessed.
17. Define pre-condition and post-condition; give an example (that is not the area() function from this chapter), preferably a computation that requires a loop.
18. When would you not test a pre-condition?  
    If it too complicated to check.
19. When would you not test a post-condition?  
    If based on the pre-conditions, the post-condition is automatically true.
20. What are the steps in debugging a program?  
    Compile, link, “get the program to do what it is supposed to do”.
21. Why does commenting help when debugging?  
    It should help in keeping the purpose of the different functions and overall code clear.
22. How does testing differ from debugging?  
    Using test cases to test the code in a systematic way, whereas in debugging you try to write the code so that it compiles and returns the correct results, in testing you try to find cases where the code fails.

# ****Chapter 6****

1. **What do we mean by “Programming is understanding”?  
   When you can program a task, you understand it, or when you understand how something is done, you can program it**
2. **The chapter details the creation of a calculator program. Write a short analysis of what the calculator should be able to do.  
   Take in inputs that represent calculations and output the correct answer, where calculations can include summation, subtraction, multiplication, division, modulo, factorial.**
3. **How do you break a problem up into smaller manageable parts?  
   By writing functions that take care of part of the problem, which call on each other.**
4. **Why is creating a small, limited version of a program a good idea?  
   Then you have a working basis which you can expand.**
5. **Why is feature creep a bad idea?  
   Because it is probably more difficult to implement an advanced feature right from the start, and it’s better to expand from working code. Also implementation-wise it might be more defined how you can best implement the fancy feature.**
6. **What are the three main phases of software development?  
   Analysis: figure out the problem and what is needed  
   Design: decide how you want to implement this  
   Implementation: writing the code, and also debugging and testing**
7. **What is a “use case”?  
   Examples of situations the code might be used in.**
8. **What is the purpose of testing?  
   To find if the outputs are correct, and in which cases the code fails (so that you can fix it).**
9. **According to the outline in the chapter, describe the difference between a Term, an Expression, a Number, and a Primary.  
   Expression : expression; expression + term, expression - term  
   Term : primary; term \* primary; term / primary  
   Primary : number; expression enclosed in brackets of any type (), {}, []  
   Number : (floating-point) number**
10. **In the chapter, an input was broken down into its component Terms, Expressions, Primarys, and Numbers. Do this for (17+4)/(5–1).**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | **Expression** |  |  |  |  |  |
|  |  |  |  |  | **↑** |  |  |  |  |  |
|  |  | **Expression** | **----------** | **----** | **|** | **------** | **----------** | **---┐** |  |  |
|  |  | **↑** |  |  | **|** |  |  | **|** |  |  |
|  |  | **Term** |  |  | **|** |  |  | **Term** |  |  |
|  | **┌-----** | **↑** | **-----┐** |  | **|** |  | **┌-----** | **--↑--** | **-----┐** |  |
|  | **Term** | **|** | **|** |  | **|** |  | **Term** | **|** | **|** |  |
|  | **↑** | **|** | **|** |  | **|** |  | **↑** | **|** | **|** |  |
|  | **Primary** | **|** | **Primary** |  | **|** |  | **Primary** | **|** | **Primary** |  |
|  | **↑** | **|** | **↑** |  | **|** |  | **↑** | **|** | **↑** |  |
|  | **Number** | **|** | **Number** |  | **|** |  | **Number** | **|** | **Number** |  |
|  | **↑** | **|** | **↑** |  | **|** |  | **↑** | **|** | **↑** |  |
| **(** | **17** | **+** | **4** | **)** | **/** | **(** | **5** | **-** | **1** | **)** |

1. **Why does the program not have a function called number()?  
   Because numbers are already evaluated in the get\_token() function.**
2. **What is a token?  
   A user-defined class that has a ‘kind’ attribute, and optionally a ‘value’ attribute.**
3. **What is a grammar? A grammar rule?  
   Grammar is that which defines the syntax of the inputs. Grammar rules then tell us what the order of calculation should be.**
4. **What is a class? What do we use classes for?  
   A (possibly) user-defined object, that can have certain attributes, functions etc. They are used to store information logically in objects.**
5. **How can we provide a default value for a member of a class?  
   Write it in the constructor of the object, so that it will be set to the default value during initialization.**
6. **In the expression function, why is the default for the switch-statement to “put back” the token?  
   Because if you do not use the token for the current expression, you should use it for the next.**
7. **What is “look-ahead”?  
   Using the next input to decide how the term evaluation should behave.**
8. **What does putback() do and why is it useful?  
   It puts a token back into the token stream and is useful because otherwise you couldn’t evaluate multiple expressions at the same time.**
9. **Why is the remainder (modulus) operation, %, difficult to implement in the term()?  
   Because the operation is not defined for floating numbers.**
10. **What do we use the two data members of the Token class for?  
    Kind and value**
11. **Why do we (sometimes) split a class’s members into private and public members?  
    Because public members are those that should be accessible from the outside, for users of the class, and private members contain what is necessary to implement the functions, but what the users need not know about/shouldn’t directly use.**
12. **What happens in the Token\_stream class when there is a token in the buffer and the get() function is called?  
    Then the Boolean full is set to ‘false’, and the buffer is returned.**
13. **Why were the ';' and 'q' characters added to the switch-statement in the get() function of the Token\_stream class?  
    Because these characters are also valid tokens (should lead to some action) and they should not lead to the ‘Bad token’ message.**
14. **When should we start testing our program?  
    During implementation, to see if the functions are really doing what they should.**
15. **What is a “user-defined type”? Why would we want one?  
    An object type that is defined in the source code by the programmer. It may be specialized for current code purposes, and therefore be useful.**
16. **What is the interface to a C++ “user-defined type”?  
    The things defined under public::**
17. **Why do we want to rely on libraries of code?  
    If others have implemented functions (and they work well and conveniently) using this code saves you as a programmer a lot of time (and debugging).**

# ****Chapter 7****

# ****Chapter 8****

1. **What is the difference between a declaration and a definition?  
   Definition adds value to the variable, declaration only specifies that the variable exists.**
2. **How do we syntactically distinguish between a function declaration and a function definition?  
   function(input\_args); versus function(input\_args){ /\* code \*/}**
3. **How do we syntactically distinguish between a variable declaration and a variable definition?  
   Object var; versus Object var = Object(init\_args);**
4. **Why can’t you use the functions in the calculator program from Chapter 6 without declaring them first?  
   Because the interface of the functions are then not yet linked, and are basically “unknown” in the code.**
5. **Is int a; a definition or just a declaration?  
   A definition, a now holds the default value for integers of 0.**
6. **Why is it a good idea to initialize variables as they are declared?  
   So you don’t run into runtime errors of variables without a value.**
7. **What can a function declaration consist of?  
   The return type, name of the function and input arguments. (You can also define the function body immediately.)**
8. **What good does indentation do?  
   Making the structure of the code clear, so that you can see the scope of variables much easier.**
9. **What are header files used for?  
   To declare the functions’ interfaces.**
10. **What is the scope of a declaration?  
    The “area” in the code where the values used in the declaration can also be accessed.**
11. **What kinds of scope are there? Give an example of each.  
    • The global scope: the area of text outside any other scope  
    • A namespace scope: a named scope nested in the global scope or in another namespace; see §8.7  
    • A class scope: the area of text within a class; see §9.2  
    • A local scope: between { . . . } braces of a block or in a function argument list  
    • A statement scope: e.g., in a for-statement**
12. **What is the difference between a class scope and local scope?  
    See 11.**
13. **Why should a programmer minimize the number of global variables?  
    To prevent global variables being accidentally changed without the programmer noticing.**
14. **What is the difference between pass-by-value and pass-by-reference?  
    The first passes a copy of the object, the other a reference to the object itself.**
15. **What is the difference between pass-by-reference and pass-by-const-reference?  
    Pass-by-const-reference does not allow modification of the object values.**
16. **What is a swap()?  
    Changing the values of two objects.**
17. **Would you ever define a function with a vector<double>-by-value parameter?  
    Sure. If the vector is not overly big.**
18. **Give an example of undefined order of evaluation. Why can undefined order of evaluation be a problem?  
    If you have global variables which are declared / initialized in multiple files. Undefined order of evaluations can gives unpredictable results.**
19. **What do x&&y and x||y, respectively, mean?  
    x&&y = true if both x and y are true (AND), x||y is true if either x or y is true (or both) (OR)**

**(&:** bitwise and, |: bitwise or, ^: bitwise exclusive or)

1. **Which of the following is standard-conforming C++: functions within functions, functions within classes, classes within classes, classes within functions?  
   Functions within classes.**
2. **What goes into an activation record?  
   The variables defined in the scope and implementation stuff.**
3. **What is a call stack and why do we need one?  
   The call stack is the stack of activation records, we need one to keep track of all the variables in the current scope.**
4. **What is the purpose of a namespace?  
   To uniquely identify functions or classes that might be named the same within their respective namespaces.**
5. **How does a namespace differ from a class?  
   A class organizes functions, data and types into a type, a namespace organizes classes, functions, data, and types into an identifiable and named part of a program without defining a type.**

# ****Chapter 9****

1. **What are the two parts of a class, as described in the chapter?  
   Data members and member functions.**
2. **What is the difference between the interface and the implementation in a class?  
   The interface is public and what the user of the class can access, the implementation is private, so inaccessible to the users except through the interface functions.**
3. **What are the limitations and problems of the original Date struct that is created in the chapter?  
   There are no checks on whether the values entered for int y, m, d should define an actual date that exists.**
4. **Why is a constructor used for the Date type instead of an init\_day() function?  
   There is no guarantee that init\_day() will be called before the Date object is referred to.**
5. **What is an invariant? Give examples.  
   The rule for what constitutes a valid value for an object / its data members.**
6. **When should functions be put in the class definition, and when should they be defined outside the class? Why?  
   “Rule of thumb: Don’t put member function bodies in the class declaration unless you know that you need the performance boost from inlining tiny functions. Large functions, say five or more lines of code, don’t benefit from inlining and make a class declaration harder to read.”**
7. **When should operator overloading be used in a program? Give a list of operators that you might want to overload (each with a reason).  
   Because the operators are not yet defined for user-defined type, for instance the == / != / ++ / -- operators.**
8. **Why should the public interface to a class be as small as possible?  
   For clarity and to reduce the amount of functions that need to be compiled when the class body is changed.**
9. **What does adding const to a member function do?  
   Ensure that the function can only be called by a constant of the class object.**
10. **Why are “helper functions” best placed outside the class definition?  
    To reduce the number of bugs that can live there and or corrupt data from the class.**