# ****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.**

# ****Chapter 10****

1. **When dealing with input and output, how is the variety of devices dealt with in most modern computers?  
   Variety is large: in/output devices – device drivers – in/output library takes care of in and output of program**
2. **What, fundamentally, does an istream do?  
   An istream reads in character sequences from “somewhere” (for example a console, a file, the main memory or another computer) and turns those into various types of certain values.**
3. **What, fundamentally, does an ostream do?  
   Turns values of various types into character sequences which are sent to “somewhere”.**
4. **What, fundamentally, is a file?  
   Stored data.**
5. **What is a file format?  
   The structure of a file.**
6. **Name four different types of devices that can require I/O for a program.**
7. **What are the four steps for reading a file?  
   have filename, open, read, close**
8. **What are the four steps for writing a file?  
   have filename, open, write, close**
9. **Name and define the four stream states.**

**good() – operations succeeded; eof() – end of input “end of file”; fail() – something unexpected happened; bad() – something unexpected and serious happened**

1. **Discuss how the following input problems can be resolved:   
   a. The user typing an out-of-range value: prompting for a new value/ skipping to in-range value  
   b. Getting no value (end of file): throw error, return  
   c. The user typing something of the wrong type: prompting for a new value/ skipping to in-range value**
2. **In what way is input usually harder than output?  
   You need to know the format of the input to make sense of it.**
3. **In what way is output usually harder than input?  
   You might get write errors.**
4. **Why do we (often) want to separate input and output from computation?  
   Because then your computation does not depend on the input/output format and you can make your code modular and easy to use with a variety of input and output data.**
5. **What are the two most common uses of the istream member function clear()?  
   Setting the state of the istream to good()**
6. **What are the usual function declarations for << and >> for a user-defined type X?  
   ostream operator<<(ostream& os, X& x)  
   istream operator>>(istream& is, X& x)**

# ****Chapter 11****

1. **Why is I/O tricky for a programmer?  
   Because the specs for how in- and output is formatted can vary a lot.**
2. **What does the notation << hex do?  
   Makes all the subsequent values being printed in hexadecimal notation.**
3. **What are hexadecimal numbers used for in computer science? Why?  
   Hexadecimal numbers can be represented in 8 bytes (so 64 bits).**
4. **Name some of the options you may want to implement for formatting integer output.**
5. **What is a manipulator?  
   A terms such as hex, dec,**
6. **What is the prefix for decimal? For octal? For hexadecimal?  
   decimal: no prefix, octal: 0, hexadecimal: 0x**
7. **What is the default output format for floating-point values?  
   Using six digits.**
8. **What is a field?  
   A string in which you can output a value with at least a certain set width. If the value is larger than the set field width, the complete will still be outputted (rather than truncating the value).**
9. **Explain what setprecision() and setw() do.  
   Sets the number of digits for outputting values. Setw sets the width of**
10. **What is the purpose of file open modes?  
    To specify how or where you want to write/ modify / read from the file.**
11. **Which of the following manipulators does not “stick”: hex, scientific, setprecision(), showbase, setw?  
    Only setw does not stick.**
12. **What is the difference between character I/O and binary I/O?  
    Binary uses fixed byte sizes to represent character, where the size of a representation**
13. **Give an example of when it would probably be beneficial to use a binary file instead of a text file.  
    Binary notation is more compact.**
14. **Give two examples where a stringstream can be useful.  
    An stringstream can be useful:  
    - for formatting output for a system that requires a simple string argument, such as a GUI system  
    - when we want to separate actual I/O from processing**
15. **What is a file position?  
    The place where you are reading/writing from**
16. **What happens if you position a file position beyond the end of file?  
    Depends on the operating system, but probably and exception.**
17. **When would you prefer line-oriented input to type-specific input?  
    When you don’t know how many arguments you will get and want to process all in one batch.**
18. **What does isalnum(c) do?  
    It checks if char is alphanumeric, and returns true if so, false otherwise.**

# ****Chapter 14****

1. **What is an application domain?**
2. **What are ideals for naming?  
   Not too long, and clear**
3. **What can we name?  
   Functions and members**
4. **What services does a Shape offer?**
5. **How does an abstract class differ from a class that is not abstract?  
   It cannot be instantiated**
6. **How can you make a class abstract?  
   By making the constructor abstract**
7. **What is controlled by access control?  
   Access to functions and members**
8. **What good can it do to make a data member private?  
   Make sure the data member is not incorrectly edited**
9. **What is a virtual function and how does it differ from a non-virtual function?  
   A function that can be overridden by a child class object, this is not the case for non-virtual functions**
10. **What is a base class?  
    The superclass from which subclasses can be derived**
11. **What makes a class derived?  
    class <child> : <parent>**
12. **What do we mean by object layout?  
    The object data members and how they are saved in memory**
13. **What can you do to make a class easier to test?  
    Modularity, clear coding / commenting**
14. **What is an inheritance diagram?  
    The diagram showing the interconnection of class types**
15. **What is the difference between a protected member and a private one?  
    Private members can only be accessed from the class within they are defined.**
16. **What members of a class can be accessed from a class derived from it?  
    The protected and public members**
17. **How does a pure virtual function differ from other virtual functions?  
    A pure virtual function is defined as “=0”**
18. **Why would you make a member function virtual?  
    Add virtual before function name**
19. **Why would you make a virtual member function pure?  
    Define function body as =0**
20. What does overriding mean? **Having a different function implementation for child class than for parent class**
21. How does interface inheritance differ from implementation inheritance? **Interface inheritance: child object can be accepted as in/output arguments instead of base class objects  
    Implementation inheritance: child objects can use data and member functions of parent objects**
22. What is object-oriented programming? **Using superclasses and subclasses etc.**

# ****Chapter 17****

1. **Why do we need data structures with varying numbers of elements?  
   So you can add/remove elements to/from an existing data structure**
2. **What four kinds of storage do we have for a typical program?  
   Code/ text storage (memory for the code)  
   Static data (for global variables defined)  
   Stack/ automatic storage: for arguments and local variables  
   Free store/ the heap: the rest of the computer**
3. **What is the free store? What other name is commonly used for it? What operators support it?  
   Storage in the computer memory (permanent); the heap; the new operator allocates memory**
4. **What is a dereference operator and why do we need one?  
   It (\*) is the operator that points to the first element of the data structure the pointer points to. So for and array p, p[0] = \*p**
5. **What is an address? How are memory addresses manipulated in C++?  
   The location in memory of an object. With the “address of” operator & and the “contents of” operator \*.**
6. **What information about a pointed-to object does a pointer have? What useful information does it lack?  
   The location; the size of the pointed-to object**
7. **What can a pointer point to?  
   Any type of object.**
8. **What is a leak?  
   If you keep allocating memory to an object but do not dereference it after it is not useful anymore.**
9. **What is a resource?  
   Resources include: memory, locks, file handles, thread handles, sockets, windows, etc.**
10. **How can we initialize a pointer?  
    for example: int x = 17; int\* p = &x;**
11. **What is a null pointer? When do we need to use one?  
    A pointer that doesn’t point to an address. You would need one to dereference a pointer.**
12. **When do we need a pointer (instead of a reference or a named object)?  
    When you want access to the memory address instead of the value.**
13. **What is a destructor? When do we want one?  
    The function that is automatically called when the object disappears from the scope. You need one if you allocate memory on the free store for the construction of the object.**
14. **When do we want a virtual destructor?  
    When subclasses might allocate memory in the free store.**
15. **How are destructors for members called?  
    When the object goes out of scope.**
16. **What is a cast? When do we need to use one?  
    Converting from one type to another. For instance if the range is too small for a certain type.**
17. **How do we access a member of a class through a pointer?  
    (\*pointer).member-name**
18. **What is a doubly-linked list?  
    A list where you can find both the predecessor and successor given a link in the list.**
19. **What is <this> and when do we need to use it?  
    The pointer that points to the entire object, you need it if a vector’s member functions need to refer to that object.**

# ****Chapter 18****

1. **What does “Caveat emptor!” mean?  
   “Beware”**
2. **What is the default meaning of copying for class objects?  
   Copying all the data members, so also pointers.**
3. **When is the default meaning of copying of class objects appropriate? When is it inappropriate?  
   When you do not plan to use both objects in the**
4. **What is a copy constructor?  
   Constructor for copying the data of an object into a new object.**
5. **What is a copy assignment?  
   Constructor for assigning the data of an object into a new object.**
6. **What is the difference between copy assignment and copy initialization?  
   Copy initialization both creates and assigns the new object.**
7. **What is shallow copy? What is deep copy?  
   Copying pointer references, both are referring to the same memory address.**
8. **How does the copy of a vector compare to its source?**
9. **What are the five “essential operations” for a class?  
   copy constructor, copy assignment, move constructor, move assignment, destructor**
10. **What is an explicit constructor? Where would you prefer one over the (default) alternative?  
    A constructor that prevents implicit conversion from the argument type to the constructor’s type (accomplished by declaring the constructor explicit for a constructor taking a single argument).**
11. **What operations may be invoked implicitly for a class object?  
    Desctructor. Copy assignment, Move assignment?**
12. **What is an array?  
    A list of objects stored after another in memory.**
13. **How do you copy an array?  
    std::copy(array\_to\_copy.begin(), array.to.copy.end(), array\_to\_copy\_to)**
14. **How do you initialize an array?  
    for instance: array<T, 3> {1,2,3};**
15. **When should you prefer a pointer argument over a reference argument? Why?  
    A pointer can receive a null parameter, so if there is a chance that you need to pass that, use pointer.**
16. **What is a C-style string?  
    String ending with “\0”**
17. **What is a palindrome?  
    words that are the same if they are spelled backwards: parterretrap – moorddroom - tattarrattat**

# ****Chapter 19****

1. **Why would we want to change the size of a vector?  
   Because you might not know in advance how much space you need.**
2. **Why would we want to have different element types for different vectors?  
   Because it is useful to have be able to have lists of different element (memory also plays a role)**
3. **Why don’t we just always define a vector with a large enough size for all eventualities?  
   Not superefficient, limits the amount of vectors you can have. Also maybe not even possible to account for all eventualities.**
4. **How much spare space do we allocate for a new vector?**
5. **When must we copy vector elements to a new location?  
   When the elements do not fit anymore in the total space allocated.**
6. **Which vector operations can change the size of a vector after construction?  
   v.resize(x) / v.push\_back(val) / v = v2;**
7. **What is the value of a vector after a copy?  
   The value of the vector copied.**
8. **Which two operations define copy for vector?  
   Copy and swap.**
9. **What is the default meaning of copy for class objects?  
   Copying all data members?**
10. **What is a template?  
    A generalizable parameterized type.**
11. **What are the two most useful types of template arguments?**
12. **What is generic programming?  
    Accounting for multiple types.**
13. **How does generic programming differ from object-oriented programming?**
14. **How does array differ from vector?  
    Array size is fixed at compile time.**
15. **How does array differ from the built-in array?**
16. **How does resize() differ from reserve()?  
    Resize sets the value of the additional elements to the default, reserve only allocates space that can eventually be filled, but is not yet.**
17. **What is a resource? Define and give examples.**
18. **What is a resource leak?  
    Keeping resources open/ occupied, and not releasing them.**
19. **What is RAII? What problem does it address?  
    Resource acquisition is initialization. Memory leaks/ forgetting to delete pointers.**
20. **What is unique\_ptr good for?  
    A unique pointer will delete itself if it disappears out of scope, so you can use it to return from a function.**

# ****Chapter 20****

1. **Why does code written by different people look different? Give examples.**
2. **What are simple questions we ask of data?**
3. **What are a few different ways of storing data?**
4. **What basic operations can we do to a collection of data items?**
5. **What are some ideals for the way we store our data?**
6. **What is an STL sequence?**
7. **What is an STL iterator? What operations does it support?**
8. **How do you move an iterator to the next element?**
9. **How do you move an iterator to the previous element?**
10. **What happens if you try to move an iterator past the end of a sequence?**
11. **What kinds of iterators can you move to the previous element?**
12. **Why is it useful to separate data from algorithms?**
13. **What is the STL?**
14. **What is a linked list? How does it fundamentally differ from a vector?**
15. **What is a link (in a linked list)?**
16. **What does insert() do? What does erase() do?**
17. **How do you know if a sequence is empty?**
18. **What operations does an iterator for a list provide?**
19. **How do you iterate over a container using the STL?**
20. **When would you use a string rather than a vector?**
21. **When would you use a list rather than a vector?**
22. **What is a container?**
23. **What should begin() and end() do for a container?**
24. **What containers does the STL provide?**
25. **What is an iterator category? What kinds of iterators does the STL offer?**
26. **What operations are provided by a random-access iterator, but not a bidirectional iterator?**

# ****Chapter 21****

1. **What are examples of useful STL algorithms?**
2. **What does find() do? Give at least five examples.**
3. **What does count\_if() do?**
4. **What does sort(b,e) use as its sorting criterion?**
5. **How does an STL algorithm take a container as an input argument?**
6. **How does an STL algorithm take a container as an output argument?**
7. **How does an STL algorithm usually indicate “not found” or “failure”?**
8. **What is a function object?**
9. **In which ways does a function object differ from a function?**
10. **What is a predicate?**
11. **What does accumulate() do?**
12. **What does inner\_product() do?**
13. **What is an associative container? Give at least three examples.**
14. **Is list an associative container? Why not?**
15. **What is the basic ordering property of binary tree?**
16. **What (roughly) does it mean for a tree to be balanced?**
17. **How much space per element does a map take up?**
18. **How much space per element does a vector take up?**
19. **Why would anyone use an unordered\_map when an (ordered) map is available?**
20. **How does a set differ from a map?**
21. **How does a multimap differ from a map?**
22. **Why use a copy() algorithm when we could “just write a simple loop”?**
23. **What is a binary search?**