**1.** What do we mean by “Programming is understanding”?

Every stage of program development requires strong understanding - from the analysis and design till the implementation of program. We need to understand what we are doing and why we do it like that in order to receive a result that we need to get to solve the problem.

**2.** The chapter details the creation of a calculator program. Write a short analysis of what the calculator should be able to do.

A calculator program needs to perform simple calculations such as addition, substraction, multiplying and dividing of floating-point numbers.

**3.** How do you break a problem up into smaller manageable parts?

By using the libraries and other useful tools we can save a lot of time. Also, we should look up for a parts of a solution that can be separately described.

**4.** Why is creating a small, limited version of a program a good idea?

By creating a prototype we can bring out the problems in our understanding, and change the details of problem to make it manageable. Also we can simply start writing a new protype, because this is just a first version. It helps not to grow a lot of mess.

**5.** Why is feature creep a bad idea?

Feature creep can significantly increase development time. It is better to build a stable version of program with essential features and then start to add new one.

**6.** What are the three main phases of software development?

Analysis of the problem and searching for a solution to it

Design of overall structure of the program and deciding which parts will have the program and how they will be connected to each other.

Implementing - writing a code, debugging and testing it.

**7.** What is a “use case”?

A specific use of a program performed to show its abilities and purpose

**8.** What is the purpose of testing?

The main purpose of testing is to confirm that our program outputs what we need to solve the problem when the input of appropriate type is provided.

**9.** According to the outline in the chapter, describe the difference between a Term, an Expression, a Number, and a Primary.

A Primary can be a ‘(‘ or ‘)’ symbol and an integer. Its purpose is to parse all the numbers and call expression for calculations in parentheses.

All Numbers are being parsed by Primary. If it is a number, it is being brought back to stream and then the whole number from the stream is being parsed.

A Term performs calculations with \* and / symbols.

An Expression performs + and - calculations and outputs a result of all calculations.

**10.** In the chapter, an input was broken down into its component Terms, Expressions, Primarys, and Numbers. Do this for (17+4)/(5–1).

(17+4)(5-1) = Term

(17+4), (5-1) = Primarys

17+4, 5-1 = Expressions

17, 4, 5, 1 = Numbers

**11.** Why does the program not have a function called number()?

Because primary() processes not only numbers but also parentheses, and it wouldn’t be correct to call its number();

**12.** What is a token?

Token is a sequence of characters that we consider as a unit such as a number or an operator.

**13.** What is a grammar? A grammar rule?

Grammar is a set of rules for syntax of the program. A grammar rule is a set of statements that must be true during the execution of program

**14.** What is a class? What do we use classes for?

A user-defined type, that contains member function, member data and member types.

**15.** How can we provide a default value for a member of a class?

We should initialize a member of class with the default value in curly brackets.

**16.** In the expression function, why is the default for the switch-statement to “put back” the token?

If expression() will encounter not a ‘-’ or ‘+’ symbol it won’t be processed but it will be thrown out of token and term() or primary() won’t get it. That’s why we put it back to token stream in order other functions can use it.

**17.** What is “look-ahead”?

To look one step forward. For example to look at the next symbol while working with the current one.

**18.** What does putback() do and why is it useful?

The tokens, that term() and expression() don’t use are being thrown out without actual processing by functions that processes such tokens. That’s why we need to bring the token back to input stream.

**19.** Why is the remainder (modulus) operation, %, difficult to implement in the term()?

Because we use float-point numbers as an input. If we want to implement % we need to check both operands whether they are integers or not.

**20.** What do we use the two data members of the Token class for?

The first is used to hold kind of value. If it is an operator or parentheses than we simply hold this symbol. If it is a number the kind will be ‘8’ as we decided by ourselves.

The second member of Token is used to hold a value.

**21.** Why do we (sometimes) split a class’s members into private and public members?

By splitting a class’s members we separate facilities that are available for users from facilities that they shouldn’t be able to use.

**22.** What happens in the Token\_stream class when there is a token in the buffer and the get() function is called?

The current token in the buffer is being returned by get(), buffer becomes available to hold a new token.

**23.** Why were the ';' and 'q' characters added to the switch-statement in the get() function of the Token\_stream class?

If we didn’t add them, the statement in default section would be executed and the program would output ‘Bad Token’

**24.** When should we start testing our program?

After finishing debugging.

**25.** What is a “user-defined type”? Why would we want one?

There are thousand of user-defined type possible. However C++ can’t hold all of them, that’s why C++ supports design and use of our own types.

**26.** What is the interface to a C++ “user-defined type”?

Interface describes how member function or member data should be used.

**27.** Why do we want to rely on libraries of code?

To save time by using already tested facilities.