GCST

2.3 PRODUCING ROBUST PROGRAMS

CONCISE NOTES

GCSE OCR

GCST

2.3.1 DEFENSIVE DESIGN

A **robust program** is one that functions correctly even if the input data, intended use or situation changes. **Defensive design** and testing help to make programs robust. Defensive design means thinking ahead about problems that could occur. It involves the following areas.

Anticipating misuse is thinking about ways that users could cause the program to fail. This could be either deliberate or accidental on the user's behalf. These potential problems can then be dealt with. Authentication is checking the identity of a user. This could be done by using a username and password, through possession of a unique electronic key or through biometrics.

Input validation is checking whether data matches certain rules as it is input. These rules can check that data is sensible but can never show that the data is correct. For example, ST13 7TY would be a sensible postcode and meets the defined rules, but it is not possible for a computer to easily decide whether it is actually that user's postcode.

Maintainability is ensuring that a program is as easy to understand and modify as possible for other future programmers. It involves the following:

- Use of **subprograms** splitting programs down into multiple subprograms reduces the need to copy and paste code.
- Variables and subprograms should use sensible **naming conventions**.
- **Indentation** is the use of the tab or space key to align code in a program. Indenting is used to highlight the structure of code and show which code belongs inside a particular block.
- **Commenting** allows programmers to add notes to their program to explain what sections of code do or how they work.

2.3.2 TESTING

The purpose of **testing** is to ensure that a program functions as expected and meets all requirements. Testing should be destructive and aim to find errors, not just show that the program works in one situation.

Iterative testing is testing during the development of a program. Each module is thoroughly tested as it is completed. This type of testing is repeated for future modules.

Final/terminal testing is completed near to the end of program development. It is testing the program as a whole for functionality

A **syntax error** is one that breaks the grammatical rules of the programming language. Examples include misspelling a keyword, missing a bracket or using a keyword in the wrong way. Syntax errors will stop the program from running when encountered.

A **logic error** is one that causes the program to produce an unexpected or incorrect output but will not stop the program from running.

Test data should be chosen so that the system as a whole can be tested destructively, checking for errors wherever they may occur.

- **Normal test data** is data of the correct type that would typically be expected from a user who is correctly using the system. This should be accepted by the program without causing errors.
- **Boundary test data** is test data that is of the correct type but is on the very edge of being valid. Boundary test data should be accepted by the program without causing errors.
- **Invalid test data** is test data that is of the correct type but outside the accepted limits. Invalid test data should be rejected by the program.
- Erroneous test data is test data that is of the incorrect type and should be rejected by the system. For example, if a program expected numeric input, a string would be erroneous input.

A test plan lists all of the tests that will be carried out, the expected result and the actual result in each case.

If you found this useful, drop a follow to help me out!

THANK YOU!

GCST

GCSECOMPUTERSCIENCETUTOR.COM