Strong Typing

A strongly-typed programming language is one in which variable type is defined. (such as integer, character, hexadecimal, packed decimal, and so forth).

If we specify a particular type to our data, the compiler will consider the data as the specified type and no other type.

An example of arithmetic operation in strongly typed language that will generate an error:
int addition = 10 + “10”;
Here, a string type cannot be added with an integer type.

Strong typed language minimize errors while running the program because most of the errors will have been corrected before running the program.

Some examples of strongly typed languages are C and Java.

Viva Voce on Strong Typing

Q1. What is strong type language?
Ans. A strongly-typed programming language is one in which each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Q2. Is strong type language predefined part of language?
Ans. Yes.

Q3. In Strongly typed language errors are maximize or minimize?
Ans. Minimize.

Q4. In this Strongly typed language are slower / faster ?
Ans. It is faster.

Q5. Write some examples of strong typed language?
Ans. SQL, C , Java.

Q6. A language is strongly typed if there are compile-time or run-time checks for what?
Ans. it check the type constraint violations. If no checking is done, it is weakly typed.

Q7. What is the rule of strong typed language?
Ans. Generally, a strongly typed language has stricter typing rules at compile time, which implies that errors and exceptions are more likely to happen during compilation.

Q8. A language is strongly typed if the type of its data objects is fixed or not?
Ans. Yes, A language is strongly typed if the type of its data objects is fixed.

Q9. A language is weakly typed if the type of its data objects is changeable or not?
Ans. If the type of a datum can change.

Related Posts:

1. Sequence Control & Expression | PPL
2. PPL:Named Constants
3. Parse Tree | PPL | Prof. Jayesh Umre
4. Basic elements of Prolog
5. Loops | PPL | Prof. Jayesh Umre

6. Subprograms Parameter passing methods | PPL | Prof. Jayesh Umre
7. Programming Paradigms | PPL | Prof. Jayesh Umre
8. Subprograms Introduction | PPL | Prof. Jayesh Umre
9. Phases of Compiler | PPL | Prof. Jayesh Umre
10. Parse Tree | PPL
11. Influences on Language design | PPL | Prof. Jayesh Umre
12. Fundamentals of Subprograms | PPL | Prof. Jayesh Umre
13. Programming Paradigm
14. Influences on Language Design
15. Language Evaluation Criteria
16. OOP in C++ | PPL
17. OOP in C# | PPL
18. OOP in Java | PPL
19. PPL: Abstraction & Encapsulation
20. PPL: Semaphores
21. PPL: Introduction to 4GL
22. PPL: Variable Initialization
23. PPL: Conditional Statements
24. PPL: Array
25. PPL: Coroutines
26. PPL: Exception Handler in C++
27. PPL: OOP in PHP
28. PPL: Character Data Type
29. PPL: Exceptions
30. PPL: Heap based storage management
31. PPL: Primitive Data Type
32. PPL: Data types

33. Programming Environments | PPL
34. Virtual Machine | PPL
35. PPL: Local referencing environments
36. Generic Subprograms
37. Local referencing environments | PPL | Prof. Jayesh Umre
38. Generic Subprograms | PPL | Prof. Jayesh Umre
39. PPL: Java Threads
40. PPL: Loops
41. PPL: Exception Handling
42. PPL: C# Threads
43. Pointer & Reference Type | PPL
44. Scope and lifetime of variable
45. Design issues for functions
46. Parameter passing methods
47. Fundamentals of sub-programs
48. Subprograms
49. Design issues of subprogram
50. Garbage Collection
51. Issues in Language Translation
52. PPL Previous years solved papers
53. Type Checking | PPL | Prof. Jayesh Umre
54. PPL RGPV May 2018 solved paper discussion| Prof. Jayesh Umre
55. PPL Viva Voce
56. PPL RGPV June 2017 Solved paper | Prof. Jayesh Umre
57. Concurrency
58. Basic elements of Prolog
59. Introduction and overview of Logic programming

60. Application of Logic programming
61. PPL: Influences on Language Design
62. Language Evaluation Criteria PPL
63. PPL: Sequence Control & Expression
64. PPL: Programming Environments
65. PPL: Virtual Machine
66. PPL: Programming Paradigm
67. PPL: Pointer & Reference Type
68. try-catch block in C++