Type safety

Michael Nowak

Texas A&M University

Overview

Basic terminology

Strongly typed languages

Strong typing vs. weak typing

Static vs. dynamic type checking

Type Safety

References

Overview

Basic terminology

Type Defines a set of possible values and a set of operations for an object

Basic terminology

Type Defines a set of possible values and a set of operations for an object

Type checking The activity of ensuring that the operands of an operator are of compatible types

Basic terminology

Type Defines a set of possible values and a set of operations for an object

Type checking The activity of ensuring that the operands of an operator are of compatible types

Compatible types Is one that is either legal for the operator or is allowed under language rules to be implicitly converted by compiler-generated code to a legal type

Basic terminology

Type Defines a set of possible values and a set of operations for an object

Type checking The activity of ensuring that the operands of an operator are of compatible types

Compatible types Is one that is either legal for the operator or is allowed under language rules to be implicitly converted by compiler-generated code to a legal type

Type error The application of an operator to an operand of an inappropriate type

Overview

`Strongly typed` languages

- A programming language is `strongly typed` if type errors are always detected

Strongly typed languages

- A programming language is `strongly typed` if type errors are always detected
- This requires that the types of all operands can be determined, either at compile-time or run-time

Strongly typed languages

- A programming language is `strongly typed` if type errors are always detected
- This requires that the types of all operands can be determined, either at compile-time or run-time
- The importance of strong typing is in its ability to detect all misuses of variables that result in type errors

Strongly typed languages

- A programming language is `strongly typed` if type errors are always detected
- This requires that the types of all operands can be determined, either at compile-time or run-time
- The importance of strong typing is in its ability to detect all misuses of variables that result in type errors
- A strongly typed language also allows the detection, at run-time, of uses of the incorrect type values in variables that can store values of more than one type

Strongly typed languages

- A programming language is `strongly typed` if type errors are always detected
- This requires that the types of all operands can be determined, either at compile-time or run-time
- The importance of strong typing is in its ability to detect all misuses of variables that result in type errors
- A strongly typed language also allows the detection, at run-time, of uses of the incorrect type values in variables that can store values of more than one type
- C++ is not strongly typed: it includes union types, which are not type checked

Overview

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)
  - With that said, even languages that allow you to break the type system frequently discourage it

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)
  - With that said, even languages that allow you to break the type system frequently discourage it
- Different languages might be more liberal with what is "allowed" by the type system

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)
  - With that said, even languages that allow you to break the type system frequently discourage it
- Different languages might be more liberal with what is "allowed" by the type system
  - An arithmetic operator with one floating-point operand and one integer operand is legal in C++

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)
  - With that said, even languages that allow you to break the type system frequently discourage it
- Different languages might be more liberal with what is "allowed" by the type system
  - An arithmetic operator with one floating-point operand and one integer operand is legal in C++
  - The value of the integer operand is `coerced` to floating-point, and a floating-point operation takes place

Strong typing vs. weak typing

- From a certain perspective, languages are either strongly typed or they are weakly typed
  - Either a language allows you to break the type system (`weakly typed`) or it doesn't (`strongly typed`)
  - With that said, even languages that allow you to break the type system frequently discourage it
- Different languages might be more liberal with what is "allowed" by the type system
  - An arithmetic operator with one floating-point operand and one integer operand is legal in C++
  - The value of the integer operand is `coerced` to floating-point, and a floating-point operation takes place
  - Such `coercion` results in the loss of one of the benefits of strong typing – error detection

- A lot of effort has gone into making the recent standards of C++ language strongly typed
  - You should use the subset of the C++ language that is strongly typed (unless, of course, you have reason not to)
  - With that said, C++ has a great deal of coercion that you must be aware the implications of

Overview

Static vs. dynamic type checking

- `Static type checking` is done at compile-time by the compile
- `Dynamic type checking` is done at run-time by the run-time system
- "*Proponents of statically type checked languages argue that they provide a certain level of safety in that the compiler does a formal proof that the types in the program are correct*" (M. C. Lewis)
- "*Proponents of dynamically typed languages argue that the type systems associated with static type checking are overly restrictive and add a burden to the programming process*" (M. C. Lewis)

Overview

Type safety

- A program (or part of one) is type safe when objects are used only according to the rules specified for their type
- Complete type safety is ideal; however, the C++ compiler does not guarantee complete type safety
  - C++ is a predominantly statically typed language that includes some dynamic typing as well
  - This is because C++ includes some constructs that can't be statically checked
- We should avoid type safety violations when possible; when this is not practical, it is our responsibility to provide the relevant checks

Overview

References

- Lewis, M. C. *Strong Typing vs. Weak Typing vs. Static Typing vs. Dynamic Typing*. http://dynamicsofprogramming.blogspot.com/2014/05/strong-typing-vs-weak-typing-vs-static.html
- Sebesta, R. W. (2016). *Concepts of programming languages* (11th ed.). Pearson Education.
- Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.