

Function overloading (ad hoc polymorphism)

Michael Nowak

Texas A&M University

Acknowledgement: Lecture slides based on those created by Bjarne Stroustrup for use with his textbook

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**
- ▶ Function **overloading** is also known as ad hoc polymorphism

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**
- ▶ Function overloading is also known as ad hoc polymorphism
 - ▶ Polymorphism comes from the Greek word *poly* meaning “many, much” and *morphē* meaning “form, shape”; a **polymorphic function** provides different implementations depending on the type of argument(s) to which it is applied

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**
- ▶ Function overloading is also known as ad hoc polymorphism
 - ▶ Polymorphism comes from the Greek word *poly* meaning “many, much” and *morphē* meaning “form, shape”; a **polymorphic function** provides different implementations depending on the type of argument(s) to which it is applied

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**
- ▶ **Function overloading** is also known as **ad hoc polymorphism**
 - ▶ **Polymorphism** comes from the Greek word *poly* meaning “many, much” and *morphē* meaning “form, shape”; a **polymorphic function** provides different implementations depending on the type of argument(s) to which it is applied
 - ▶ **Ad hoc** refers to notion that the **overloaded functions** have been defined explicitly for distinct parameter configurations

Introduction

- ▶ Functions that have the same name but different parameter lists *and* appear in the same scope are **overloaded**
- ▶ **Function overloading** is also known as **ad hoc polymorphism**
 - ▶ **Polymorphism** comes from the Greek word *poly* meaning “many, much” and *morphē* meaning “form, shape”; a **polymorphic function** provides different implementations depending on the type of argument(s) to which it is applied
 - ▶ **Ad hoc** refers to notion that the **overloaded functions** have been defined explicitly for distinct parameter configurations
 - ▶ This type of polymorphism is not a fundamental feature of the type system

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Motivation

- ▶ Eliminates the need to define different names for functions that perform the same general action but on different parameter types

Motivation

- ▶ Eliminates the need to define different names for functions that perform the same general action but on different parameter types
- ▶ Instead of providing different names, we can use the same name and let the compiler figure out which function to call based on the types arguments in a call

Motivation

- ▶ Eliminates the need to define different names for functions that perform the same general action but on different parameter types
- ▶ Instead of providing different names, we can use the same name and let the compiler figure out which function to call based on the types arguments in a call
- ▶ For instance, there is only one name for addition, yet it can be used to add values of the arithmetic types

Motivation

- ▶ Eliminates the need to define different names for functions that perform the same general action but on different parameter types
- ▶ Instead of providing different names, we can use the same name and let the compiler figure out which function to call based on the types arguments in a call
- ▶ For instance, there is only one name for addition, yet it can be used to add values of the arithmetic types
 - ▶ When a name is semantically significant, the convenience of overloading becomes practically essential

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Defining overloaded functions

- ▶ When we overload functions, we are creating multiple functions that have the:
 - ▶ Same name
 - ▶ Different parameter configurations
 - ▶ Number of parameters
 - ▶ Types of parameters
 - ▶ Order for parameter types
- ▶ C++ forbids functions that differ only in return type; this would introduce ambiguity as to which function is to be called

Defining overloaded functions

- Using function overloading we could declare a collection of functions that convert the `double` and `char` arguments to a `std::string`:

```
string to_str(double);  
string to_str(char);
```

Defining overloaded functions

- Using function overloading we could declare a collection of functions that convert the `double` and `char` arguments to a `std::string`:

```
string to_str(double);  
string to_str(char);
```

- We could define these functions as:

```
string to_str(double d)  
{  
    stringstream ss;  
    ss << d;  
    return ss.str();  
}
```

```
string to_str(char c)  
{  
    stringstream ss;  
    ss << c;  
    return ss.str();  
}
```

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Calling an overloaded function

- **Overload resolution** is the process by which the compiler determines which specific function is called from a set of overloaded functions

Calling an overloaded function

- ▶ **Overload resolution** is the process by which the compiler determines which specific function is called from a set of overloaded functions
- ▶ The compiler determines this by comparing the arguments against the parameters of each function in the set of overloaded functions

Calling an overloaded function

- For now, lets consider the following outcomes:

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match
 - ▶ A match through a promotion: `char` to `int`, `float` to `double`, etc.

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match
 - ▶ A match through a promotion: `char` to `int`, `float` to `double`, etc.
 - ▶ A match using standard conversions: `int` to `double`, `double` to `int`, etc.

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match
 - ▶ A match through a promotion: `char` to `int`, `float` to `double`, etc.
 - ▶ A match using standard conversions: `int` to `double`, `double` to `int`, etc.
 - ▶ ...

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match
 - ▶ A match through a promotion: `char` to `int`, `float` to `double`, etc.
 - ▶ A match using standard conversions: `int` to `double`, `double` to `int`, etc.
 - ▶ ...
 - ▶ There is no function with parameters that are a best match (exact match or compatible with) the arguments; compiler will report there was **no match**

Calling an overloaded function

- ▶ For now, let's consider the following outcomes:
 - ▶ The compiler finds exactly one function whose parameter(s) is(are) a **best match** for the actual argument(s):
 - ▶ An exact match
 - ▶ A match through a promotion: `char` to `int`, `float` to `double`, etc.
 - ▶ A match using standard conversions: `int` to `double`, `double` to `int`, etc.
 - ▶ ...
 - ▶ There is no function with parameters that are a best match (exact match or compatible with) the arguments; compiler will report there was **no match**
 - ▶ There is more than one function that matches and amongst the matches, there isn't a best match; the compiler will report an **ambiguous call**

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

Overloading guidance

- ▶ You should use `function overloading` when a name is semantically significant amongst different data types
- ▶ Otherwise, you should probably construct functions that are identified by different names

Overview

Introduction

Motivation

Defining overloaded functions

Calling an overloaded function

Overloading guidance

References

References

- ▶ Lippman, B., Lajoie, Josee, & Moo, B. E. (2016). *C++ primer* (5th ed.). Addison-Wesley.
- ▶ Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.