

Function argument passing

Michael Nowak

Texas A&M University

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

Notes

Overview

- Introduction
- Passing arguments by value
- Passing arguments by reference
 - Using references to avoid copies
 - Using references to “return” multiple values
- Passing arguments by constant reference
- Aside: reading declarations
- Guidance for passing arguments
- Functions that return a value
 - How values are returned
 - Never return a reference or pointer to a local variable
- References

Notes

Overview

- Introduction
- Passing arguments by value
- Passing arguments by reference
 - Using references to avoid copies
 - Using references to “return” multiple values
- Passing arguments by constant reference
- Aside: reading declarations
- Guidance for passing arguments
- Functions that return a value
 - How values are returned
 - Never return a reference or pointer to a local variable
- References

Notes

Introduction

- ▶ Each time a function is called, its parameters are defined and initialized by the arguments passed in the function call
- ▶ The semantics of argument passing are identical to the semantics of initialization
 - ▶ Parameter initialization/binding works the same way as variable/reference initialization
 - ▶ When the parameter is an object, we say that the argument is passed by **value** because a value is copied into that object for initialization
 - ▶ When a parameter is a reference, we say that the argument is passed by **reference** because the parameter binds to the object being passed; the parameter becomes an **alias** for the object to which it is bound
 - ▶ Argument types are checked during compilation and implicit type conversions take place when necessary

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

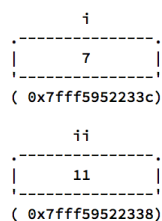
References

Notes

Passing arguments by value

- ▶ When we initialize a variable, the value of the initializer is copied into the object that the name refers; consider that:

```
int i = 7;
int ii = i; // ii value is a copy of the value in i
i = 11 // value stored in i is 11; value in ii is left
unchanged
```



Notes

Passing arguments by value

- ▶ The semantics of argument passing are identical to the semantics of initialization
- ▶ Passing an argument by value works the exact same way as initializing a variable with some value
 - ▶ Changes that the function makes to the parameter will *never* be reflected in the object (i.e., argument) used to initialize that parameter: we're merely working with a copy of the argument

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

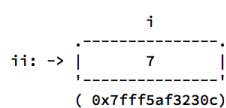
References

Notes

Passing arguments by reference

- ▶ When defining a reference to an object, we bind that reference to its initializer

```
int i = 7;  
int &ii = i; // ii a reference to i  
ii = 11 // value stored in i is now 11
```



Notes

Passing arguments by reference

- ▶ The semantics of argument passing are identical to the semantics of initialization
- ▶ Passing an argument by reference works the exact same way as binding a reference to a named object for which it is initialized
 - ▶ Changes that the function “makes” on a reference parameter will *always* be reflected in the object bound to that reference
 - ▶ The reference parameter is simply another name for the object for which it is initialized

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Using references to avoid copies

- ▶ Some objects in our program can become very large
 - ▶ Passing such objects by reference avoids the overhead of copying very large arguments
- ▶ Some objects (such as the IO types) cannot be copied
 - ▶ Passing such objects by reference allows our functions to operate on objects that cannot be copied

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Using references to “return” multiple values

- ▶ A function can only return a single object (value)
- ▶ Sometimes, we would like to “return” multiple objects to the caller
 - ▶ We could create a user-defined type that contains those objects
 - ▶ An easier solution is to pass-by-reference additional arguments that can be used to house the values that we'd like to return

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Passing arguments by constant reference

- ▶ Passing arguments by “plain vanilla” references should generally be avoided (unless you have reason to use them)
 - ▶ They can lead to obscure bugs when you forget which arguments can be changed
- ▶ You should use `reference` to `const` when you are passing large objects and would like the benefits of pass-by-reference, but do not need to modify the arguments
 - ▶ We can use the non-type specifier `const` in a declaration to tell the compiler that the object being referenced should be constant (and thus not be changed):

```
const int &i = j;
```

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Aside: reading declarations

- ▶ It is often helpful to read declarations from the right-to-left to figure out what we've got going on in a declaration
- ▶ I'll frequently declare a constant reference by writing the alternative (but equivalent) declaration `int const &i = j`;
- ▶ Reading from right-to-left, it is certain that *i* is a *reference to a constant int*
- ▶ In the following code, what is `pr`?

```
int i = 7;  
int const*p = &i;  
int const*&pr = p;
```

 - ▶ What happens if we write the statement `*pr = 11`; in our program?

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Guidance for passing arguments

- ▶ Use pass-by-value for small objects, such as the primitive built-in types
 - ▶ Using pass-by-reference to a small object is a bit slower than copying that object
 - ▶ If you think of a reference as a constant pointer (value cannot be changed) that automatically dereferences itself when it is used, the reason becomes clear
 - ▶ Such indirection would require us to look-up the address of the object being referenced and then look up the object residing at that address
- ▶ Use pass-by-const-reference for large objects, such as vectors
- ▶ Use pass-by-reference *only when you have to*
 - ▶ Favor returning a result rather than modifying an object through a reference argument

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

Functions that return a value

- ▶ Every function with a return type other than `void` must return a value
- ▶ The value that the function returns must match the return type or be compatible with the return type

Notes

Overview

Introduction

Passing arguments by value

Passing arguments by reference

Using references to avoid copies

Using references to “return” multiple values

Passing arguments by constant reference

Aside: reading declarations

Guidance for passing arguments

Functions that return a value

How values are returned

Never return a reference or pointer to a local variable

References

Notes

How values are returned

- ▶ The value to be returned is used to initialize a temporary variable at the call site
 - ▶ The type of this variable is that of the return type
- ▶ This temporary is the result of the function call
- ▶ The semantics used for initializing the temporary are identical to the semantics of initialization

Notes

Overview

- Introduction
- Passing arguments by value
- Passing arguments by reference
 - Using references to avoid copies
 - Using references to “return” multiple values
- Passing arguments by constant reference
- Aside: reading declarations
- Guidance for passing arguments
- Functions that return a value**
 - How values are returned
 - Never return a reference or pointer to a local variable**
- References

Notes

Never return a reference or pointer to a local variable

- Why not?
- Think about this... with respect to an automatic variable's lifetime...

Notes

Overview

- Introduction
- Passing arguments by value
- Passing arguments by reference
 - Using references to avoid copies
 - Using references to “return” multiple values
- Passing arguments by constant reference
- Aside: reading declarations
- Guidance for passing arguments
- Functions that return a value
 - How values are returned
 - Never return a reference or pointer to a local variable
- References

Notes

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.

Notes

Notes

Notes
