

Function basics

Michael Nowak

Texas A&M University

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

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

- ▶ This names f as a function that takes a number, which we give the name x , and it has the value of that number squared

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

- ▶ This names f as a function that takes a number, which we give the name x , and it has the value of that number squared
- ▶ When the function is written, the parameter, x , has no value

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

- ▶ This names f as a function that takes a number, which we give the name x , and it has the value of that number squared
- ▶ When the function is written, the parameter, x , has no value
- ▶ The value of x is specified when we use the function

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

- ▶ This names f as a function that takes a number, which we give the name x , and it has the value of that number squared
- ▶ When the function is written, the parameter, x , has no value
- ▶ The value of x is specified when we use the function
- ▶ $f(3)$ specifies 3 as the argument to the function f and x takes on the value of 3 during our calculations

Function refresher

- ▶ The simple example of a function from math class might look like

$$f(x) = x^2$$

- ▶ This names f as a function that takes a number, which we give the name x , and it has the value of that number squared
- ▶ When the function is written, the parameter, x , has no value
- ▶ The value of x is specified when we use the function
- ▶ $f(3)$ specifies 3 as the argument to the function f and x takes on the value of 3 during our calculations
- ▶ We easily calculate $f(3) = 9$

Function refresher

- We can have functions with more than one variable, such as

$$f(x, y) = x^2 + y^2$$

Function refresher

- We can have functions with more than one variable, such as

$$f(x, y) = x^2 + y^2$$

- We provide two values as arguments when we'd like to call this function; they are used for x and y

Function refresher

- ▶ We can have functions with more than one variable, such as

$$f(x, y) = x^2 + y^2$$

- ▶ We provide two values as arguments when we'd like to call this function; they are used for x and y
- ▶ $f(2, 3) = 4 + 9 = 13$, the x -value is 2 and the y -value is 3

Function refresher

- ▶ We can have functions with more than one variable, such as

$$f(x, y) = x^2 + y^2$$

- ▶ We provide two values as arguments when we'd like to call this function; they are used for x and y
- ▶ $f(2, 3) = 4 + 9 = 13$, the x -value is 2 and the y -value is 3
- ▶ The order of the arguments matches the order of the parameters

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Function declarations

- We can **declare** a function by writing a **declarator** of the form `f(args)`, where `f` is the **name** being introduced and `args` is a list of zero or more **parameters**, for example:

Function declarations

- We can **declare** a function by writing a **declarator** of the form `f(args)`, where `f` is the **name** being introduced and `args` is a list of zero or more **parameters**, for example:

```
double mult(double, double);
```


Function declarations

- ▶ We can **declare** a function by writing a **declarator** of the form **f(args)**, where **f** is the **name** being introduced and **args** is a list of zero or more **parameters**, for example:

```
double mult(double, double);
```

- ▶ The **base type** specifies the **return type** of the **function**
 - ▶ We can specify **void** as a “pseudo return type” for functions that do not return a value

Function declarations

- ▶ We can **declare** a function by writing a **declarator** of the form `f(args)`, where `f` is the **name** being introduced and `args` is a list of zero or more **parameters**, for example:

```
double mult(double, double);
```

- ▶ The **base type** specifies the **return type** of the **function**
 - ▶ We can specify `void` as a “pseudo return type” for functions that do not return a value
- ▶ The function's parameters are specified in a comma-separated list enclosed in parentheses
 - ▶ A function's parameter list can be left empty but can not be omitted (still need the parentheses)

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Function definitions

- ▶ A `declaration` that fully specifies the entity being `declared` is called a `definition`

Function definitions

- ▶ A **declaration** that fully specifies the entity being **declared** is called a **definition**
- ▶ We write a function **definition** by writing a function **declaration** with a statement block, i.e., a **function body**, that specifies a sequence of statements that the function will perform when called

```
double mult(double x, double y)
{
    return x*y;
}
```

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Declarations and definitions

- ▶ In your C++ programs, you can't define something twice
 - ▶ A `definition` says what something is
- ▶ However, you can declare something multiple times
 - ▶ A `declaration` says how something can be used

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used
- ▶ Often we want the **definition** “elsewhere”
 - ▶ Later in a file
 - ▶ In another file

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used
- ▶ Often we want the **definition** “elsewhere”
 - ▶ Later in a file
 - ▶ In another file
- ▶ **Declaration** are used to specify interfaces code

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used
- ▶ Often we want the **definition** “elsewhere”
 - ▶ Later in a file
 - ▶ In another file
- ▶ **Declaration** are used to specify interfaces code
 - ▶ Declarations are frequently introduced into a program through “headers”

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used
- ▶ Often we want the **definition** “elsewhere”
 - ▶ Later in a file
 - ▶ In another file
- ▶ **Declaration** are used to specify interfaces code
 - ▶ Declarations are frequently introduced into a program through “headers”
 - ▶ A “header” is a file containing declarations, giving you access to functions, types, etc. for use in your programs

Why both declarations and definitions?

- ▶ To refer to something, we need (only) its **declaration**
 - ▶ A declaration introduces a name to the compiler and how that name can be used
- ▶ Often we want the **definition** “elsewhere”
 - ▶ Later in a file
 - ▶ In another file
- ▶ **Declaration** are used to specify interfaces code
 - ▶ Declarations are frequently introduced into a program through “headers”
 - ▶ A “header” is a file containing declarations, giving you access to functions, types, etc. for use in your programs
- ▶ **Definitions** can be provided in different translation units
 - ▶ The object code containing those definitions will eventually be linked together with other object code by the **linker** in formulation of an **executable**

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Write function declarations in a header file

- For your functions, write the declarations in some header file (`descriptive_name.h`)

Write function declarations in a header file

- ▶ For your functions, write the declarations in some header file (`descriptive_name.h`)
- ▶ What happens if `descriptive_name.h` `#includes` `other_descriptive_name.h` which then `#includes` `descriptive_name.h` in the same translation unit?
 - ▶ We really want our header files to be included exactly once per translation unit

Write function declarations in a header file

- ▶ For your functions, write the declarations in some header file (`descriptive_name.h`)
- ▶ What happens if `descriptive_name.h` `#includes` `other_descriptive_name.h` which then `#includes` `descriptive_name.h` in the same translation unit?
 - ▶ We really want our header files to be included exactly once per translation unit
- ▶ We thus use `header guards` to ensure that if a file has already been included, we do not include it again;
`descriptive_name.h` would have the following structure:

```
#ifndef DESCRIPTIVE_NAME_H
#define DESCRIPTIVE_NAME_H
/* function declarations */
#endif
```

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Write function definitions in a source file

- ▶ After writing the header file for `descriptive_name.h`, we would write a corresponding source file `descriptive_name.cpp` which would define the functions previously declared

Write function definitions in a source file

- ▶ After writing the header file for `descriptive_name.h`, we would write a corresponding source file `descriptive_name.cpp` which would define the functions previously declared
- ▶ We conventionally `#include` the corresponding header `descriptive_name.h` in `descriptive_name.cpp` to ensure that each declaration has a definition with the same **function signature** and **return type**
 - ▶ If things didn't match up, we'd likely see a compiler or linker error

Write function definitions in a source file

- ▶ After writing the header file for `descriptive_name.h`, we would write a corresponding source file `descriptive_name.cpp` which would define the functions previously declared
- ▶ We conventionally `#include` the corresponding header `descriptive_name.h` in `descriptive_name.cpp` to ensure that each declaration has a definition with the same **function signature** and **return type**
 - ▶ If things didn't match up, we'd likely see a compiler or linker error
- ▶ `descriptive_name.cpp` will have the following structure:

```
#include "descriptive_name.h"  
  
/* function definitions */
```

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Using functions defined in other source files

- We can use our functions in say `main.cpp` by `#include "descriptive_name.h"`

Using functions defined in other source files

- ▶ We can use our functions in say `main.cpp` by `#include "descriptive_name.h"`
- ▶ Just make sure that when you compile, you remember to compile both `main.cpp` and `descriptive_name.cpp`:

```
g++ -std=c++14 main.cpp descriptive_name.cpp
```

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

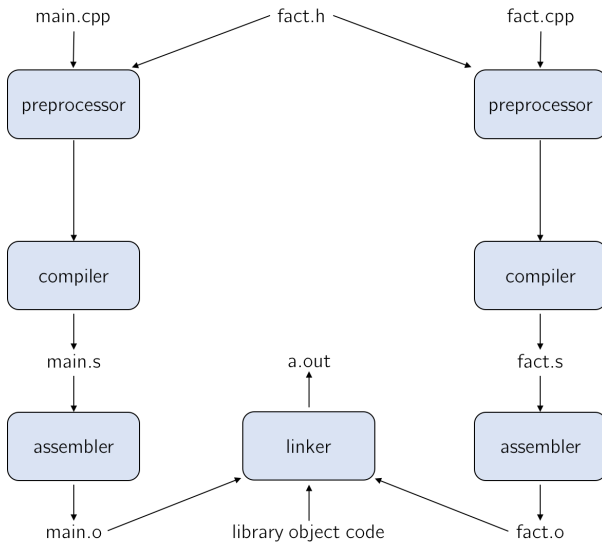
Writing a function

Calling a function

Why functions?

References

Separate compilation



Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Executing a function

- ▶ We execute a function through the `call` operator, which is a pair of parentheses

Executing a function

- ▶ We execute a function through the `call operator`, which is a pair of parentheses
- ▶ The `call operator` takes an expression that is a function

Executing a function

- ▶ We execute a function through the `call operator`, which is a pair of parentheses
- ▶ The `call operator` takes an expression that is a function
- ▶ We provide a comma-separated list of `arguments` inside the parentheses

Executing a function

- ▶ We execute a function through the `call operator`, which is a pair of parentheses
- ▶ The `call operator` takes an expression that is a function
- ▶ We provide a comma-separated list of `arguments` inside the parentheses
- ▶ The `arguments` are used to initialize the function's `parameters`

Executing a function

- ▶ We execute a function through the `call operator`, which is a pair of parentheses
- ▶ The `call operator` takes an expression that is a function
- ▶ We provide a comma-separated list of `arguments` inside the parentheses
- ▶ The `arguments` are used to initialize the function's `parameters`
- ▶ The type of a call expression is the return type of the function

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Writing a function

- We would like to write a function that calculates the `factorial` of a number

Writing a function

- ▶ We would like to write a function that calculates the `factorial` of a number
- ▶ The factorial function in mathematics, $n!$ means to multiply each integer together from 1 up to n , that is:

Writing a function

- ▶ We would like to write a function that calculates the `factorial` of a number
- ▶ The factorial function in mathematics, $n!$ means to multiply each integer together from 1 up to n , that is:

$$n! = \prod_{i=1}^n i$$

which can be rewritten informally as

$$n! = n \times n - 1 \times \dots \times 2 \times 1$$

Writing a function

- ▶ We would like to write a function that calculates the `factorial` of a number
- ▶ The factorial function in mathematics, $n!$ means to multiply each integer together from 1 up to n , that is:

$$n! = \prod_{i=1}^n i$$

which can be rewritten informally as

$$n! = n \times n - 1 \times \dots \times 2 \times 1$$

- ▶ For example, $3!$ is thus calculated as:

$$3! = 3 \times 2 \times 1$$

Writing a function

- We can write an iterative algorithm in C++ to solve for the factorial of some value `int val` as:

Writing a function

- We can write an iterative algorithm in C++ to solve for the factorial of some value `int val` as:

```
int res = 1;
while(val > 1) {
    res *= val;
    val -= 1;
}
```


Writing a function

- We can `encapsulate` our iterative solution in a function

Writing a function

- ▶ We can `encapsulate` our iterative solution in a function
- ▶ We could define such a `function` as follows:

Writing a function

- ▶ We can **encapsulate** our iterative solution in a function
- ▶ We could define such a **function** as follows:

```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```

Writing a function

- ▶ We can **encapsulate** our iterative solution in a function
- ▶ We could define such a **function** as follows:

```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```

- ▶ This **function** is named **fact**

Writing a function

- ▶ We can **encapsulate** our iterative solution in a function
- ▶ We could define such a **function** as follows:

```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```

- ▶ This **function** is named **fact**
- ▶ It takes an **int** **parameter** identified by **val**

Writing a function

- ▶ We can **encapsulate** our iterative solution in a function
- ▶ We could define such a **function** as follows:

```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```

- ▶ This **function** is named **fact**
- ▶ It takes an **int** **parameter** identified by **val**
- ▶ The **function** **body** calculates the factorial of the argument used to initialize the parameter **val**

Writing a function

- ▶ We can **encapsulate** our iterative solution in a function
- ▶ We could define such a **function** as follows:

```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```

- ▶ This **function** is named **fact**
- ▶ It takes an **int** parameter identified by **val**
- ▶ The **function** body calculates the factorial of the argument used to initialize the parameter **val**
- ▶ The **return** statement ends the execution of **fact**, returning **res** to the caller

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ As long as our function has been **declared** before its name is used, the source file containing the function call will compile to object code

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ As long as our function has been **declared** before its name is used, the source file containing the function call will compile to object code
- ▶ As long as our function has been **defined** in some source file being compiled, the linker will be able to join together the object code into an executable

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- The function call to `fact()` does two things

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created
 - ▶ That variable is initialized by the argument in the function call, 5

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created
 - ▶ That variable is initialized by the argument in the function call, `5`
- ▶ It transfers control to that function

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created
 - ▶ That variable is initialized by the argument in the function call, `5`
- ▶ It transfers control to that function
 - ▶ Execution of the calling function is suspended and execution of the called function begins

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created
 - ▶ That variable is initialized by the argument in the function call, `5`
- ▶ It transfers control to that function
 - ▶ Execution of the calling function is suspended and execution of the called function begins
 - ▶ Execution of the function ends when the thread of execution passes over the `return` statement

Calling a function

```
int main()
{
    int j = fact(5);
    /* do something */
    return 0;
}
```

- ▶ The function call to `fact()` does two things
- ▶ It initializes the function's parameters from the corresponding arguments
 - ▶ An `int` variable named `val` is created
 - ▶ That variable is initialized by the argument in the function call, `5`
- ▶ It transfers control to that function
 - ▶ Execution of the calling function is suspended and execution of the called function begins
 - ▶ Execution of the function ends when the thread of execution passes over the `return` statement
 - ▶ At this point, a value (if applicable) is returned as control is transferred back to the calling function

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”
- ▶ Match our understanding of the problem domain
 - ▶ Name logical operations
 - ▶ A function should do one thing well

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”
- ▶ Match our understanding of the problem domain
 - ▶ Name logical operations
 - ▶ A function should do one thing well
- ▶ Functions make the program easier to read

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”
- ▶ Match our understanding of the problem domain
 - ▶ Name logical operations
 - ▶ A function should do one thing well
- ▶ Functions make the program easier to read
- ▶ A function can be useful in many places in a program

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”
- ▶ Match our understanding of the problem domain
 - ▶ Name logical operations
 - ▶ A function should do one thing well
- ▶ Functions make the program easier to read
- ▶ A function can be useful in many places in a program
- ▶ Ease testing, distribution of labor, and maintenance

Why functions?

- ▶ Chop a program into manageable pieces
 - ▶ “divide and conquer”
- ▶ Match our understanding of the problem domain
 - ▶ Name logical operations
 - ▶ A function should do one thing well
- ▶ Functions make the program easier to read
- ▶ A function can be useful in many places in a program
- ▶ Ease testing, distribution of labor, and maintenance
- ▶ Keep functions small
 - ▶ Easier to understand, specify, and debug

Overview

Function refresher

Function declarations

Function definitions

Declarations and definitions

Why both declarations and definitions?

Write function declarations in a header file

Write function definitions in a source file

Using functions defined in other source files

Separate compilation

Executing a function

Writing a function

Calling a function

Why functions?

References

References

- ▶ Lewis, M. C. (2015). *Introduction to the art of programming using Scala*. CRC Press.
- ▶ 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.