Overloading functions, static variables in functions

Comp Sci 1570 Introduction to C++

Missouri S&T | Computer Science

Outline

Overloading functions

- C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading
- Two different functions can have the same name if at least of of their parameters are different, either because they have a different number of parameters, or because any of their parameters are of a different type.
- You cannot overload function declarations that differ only by return type.
- When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions.
- The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Outline

Why and how

- Function overloading is usually used to enhance the readability of the program.
- If you have to perform one single operation but with different number or types of arguments, then you can simply overload the function.
- Ways to overload a function:
  - By changing number of arguments.
  - By having different types of argument.

Outline

Examples

- See the code!

Outline

Return type does not count as uniqueness

- Function's return type is NOT considered when overloading functions.
- Consider the case where you want to write a function that returns a random number, but you need a version that will return an int, and another version that will return a double.
- You might be tempted to do this:

```
int getRandomValue();
double getRandomValue();
```

- Don't.

Outline

How function calls match overloaded functions

Making a call to an overloaded function results in one of three possible outcomes:

1. A match is found. The call is resolved to a particular overloaded function.
2. No match is found. The arguments can not be matched to any overloaded function.
3. An ambiguous match is found. The arguments matched more than one overloaded function.

Which function to call?

When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:

1. First, C++ tries to find an exact match. This is the case where the actual argument exactly matches the parameter type of one of the overloaded functions.

Which function to call?

When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:

2. Second, if no exact match is found, C++ tries to find a match through promotion:
   - Char, unsigned char, and short is promoted to an int.
   - Unsigned short can be promoted to int or unsigned int, depending on the size of an int
   - Float is promoted to double
   - Enum (not covered yet) is promoted to int

   .

Which function to call?

When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:

3. Third, if no promotion is possible, C++ tries to find a match through standard conversion. Standard conversions include:
    - Any numeric type will match any other numeric type, including unsigned (e.g. int to float)
    - Enum will match the formal type of a numeric type (e.g. enum to float)
    - Zero will match a pointer type (not covered yet) and numeric type (e.g. 0 to char*, or 0 to float)
    - A pointer will match a void pointer

Which function to call?

When an overloaded function is called, C++ goes through the following process to determine which version of the function will be called:

4. Finally, C++ tries to find a match through user-defined conversion. Although we have not covered classes yet, classes can define conversions to other types that can be implicitly applied to objects of that class.

Outline

Ambiguity

If every overloaded function has to have unique parameters, how is it possible that a call could result in more than one match? Because all standard conversions are considered equal, and all user-defined conversions are considered equal, if a function call matches multiple candidates via standard conversion or user-defined conversion, an ambiguous match will result. For example:

```
void print(unsigned int value);
void print(float value);

print('a');
print(0);
print(3.14159);
```

Ambiguity

```
void print(unsigned int value);
void print(float value);
print('a');
print(0);
print(3.14159);
```

- In the case of print('a'), C++ can not find an exact match. It tries promoting 'a' to an int, but there is no print(int) either. Using a standard conversion, it can convert 'a' to both an unsigned int and a floating point value. Because all standard conversions are considered equal, this is an ambiguous match.
- print(0) is similar. 0 is an int, and there is no print(int). It matches both calls via standard conversion.
- print(3.14159) might be a surprising, as you might assume it matches print(float). But remember that all literal floating point values are doubles unless they have the 'f' suffix. 3.14159 is a double, and there is no print(double). It matches both calls via standard conversion.

Outline

Static variables

- When a function returns, the local variables (including parameters) go out of scope and are deallocated.
- This is true for all “normal” variables, those declared local to the function, and all the parameters.
- Static variables in a function persist after the function has terminated.
- Here is the general form of the syntax:

```
void f()
{
  static variable_type variable_name;
}
```

Static variables

- A static variable in a function is particular to that function.
- That is, you can only access the variable in that function.
- Because of this, you could have a static variable in 5 functions, each with the same name.
- There are simple rules governing static variable that you will need to keep in mind.
    - A static variable declaration is only executed once, the first time the function is executed.
    - A static variable is initialized only once (since this is part of the declaration process) and will be initialized to 0 unless the programmer designates otherwise.
    - Subsequent invocations of the function in which a static variable resides will retain the last value of that variable.

Outline

Examples

Check out the code!

Outline

Static duration

- Using the static keyword on local variables changes them from automatic duration to static duration (also called fixed duration).
- A static duration variable (also called a "static variable") is one that retains its value even after the scope in which it has been created has been exited!
- Static duration variables are only created (and initialized) once, and then they are persisted throughout the life of the program.

Outline

Why?

- Local static variables were inherited from the C programming language, but their need has diminished with the introduction of objects in C++.
- Functions with static variables provide a way to implement executable code with persistent state.
- Objects provide a more natural and more flexible way to achieve the same effect.