Lecture 4

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Functions – C++ Primer Chapter 6

Functions: Basic Definitions

function prototype: Synonym for function declaration.

Name, return type, and parameter list of a function.

To call a function, its prototype must have been declared before the point of call.

call operator: The operator that causes a function to be executed.

Pair of parentheses and takes two operands: Name of the function to call and a (possibly empty) comma-separated list of arguments to pass to the function.

Parameters vs arguments!

Header Files

Header: Mechanism for making class definitions and other declarations available in multiple source files.

Headers are for declarations, not definitions!

header guard: Preprocessor variable defined to prevent a header from being included more than once in a single source file.

Functions

- Functions must specify a return type
- C++ is a statically typed language, arguments of every call are checked during compilation

C++


```
inline double multiply(double a, double b)
{
   return a*b;
}
```

```
template<typename Number>
  Number multiply(Number a, Number b)
{
   return a*b;
}
```

Argument Passing

- Each parameter is created anew on each call to the function
- Value used to initialize a parameter is the corresponding argument passed in the call
- If parameter is a nonreference type, then argument is copied
- If parameter is a reference, then it is just another name for argument

Nonreference Parameters

- Nonreference parameters represent local copies of the corresponding argument
- Changes made to the parameter are made to the local copy
- Once the function terminates, these local values are gone
  - Example: int fct(int i);
- Pointer parameters
  - Example: int fct(int* i);
- Const parameters
  - Example: int fct(int const i);

Reference Parameters

- Copying an argument is not suitable for every situation
  - We want the function to change the value of the argument
  - We want to pass a large object as an argument
- Reference parameters
  - Example: int fct(int& i);
- Array parameters
  - Example: int fct(int*);
  - Equivalent to: int fct(int[]); int fct(int [10]);
  - Array dimensions are ignored and size is not checked!
  - Passing by reference: int fct(int (&arr)[10]); (here size is checked!)

Argument Passing

- Command line options
  - Example: int main(int argc, char *argv[]) {}
- Functions with varying parameters (old style!)
  - Example: void fct(parm_list, …);
  - In C: printf
- initializer_lists parameter
  - Example: void msg (initializer_lists<string> il) {}

Return Statement

- Functions with no return value
  - Example: void fct() { return; }
- Functions that return a value
  - The value returned by a function is used to initialize a temporary object created at the point the call was made
  - Never return a reference/pointer to a local object
  - Reference returns are lvalues
  - List initialization the return value (C++11)
- Recursion: function calls itself again

Function Declarations

- Function prototypes provide the interface between programmer and user
- Source file that defines the function should include the header that declares the function
- Default arguments
  - Either specified in function definition or declaration (not both!)
  - Example: int fct(int i = 1);

Object Lifetime

Names have scope, objects have lifetime!

object lifetime: Every object has an associated lifetime.

- Objects defined inside a block exist from when their definition is encountered until the end of the block.
- Local static objects and global objects defined outside any function are created during program startup and destroyed when main function ends.
- Dynamically created objects created through a new expression exist until the memory in which they were created is freed through delete.

Local Objects I

automatic objects: Objects local to a function.

- Automatic objects are created and initialized anew on each call and destroyed at the end of the block in which they are defined.
- They no longer exist once the function terminates.
- Examples: Parameters

temporary (object): Unnamed object automatically created by the compiler when evaluating an expression.

- A temporary persists until the end of largest expression that encloses the expression for which it was created.
- Example: i+j in expression int res = i + j + k

Local Objects II

local static objects:

- Guaranteed to be initialized no later than first time that program execution passes through object‘s definition
- Not destroyed until program terminates
- Example: size_t count() { static size_t ctr = 0; return ++ctr; }

Inline Functions

- **inline function**: Function that is expanded at the point of call, if possible.
- Inline functions avoid normal function-calling overhead by replacing the call by the function's code.
- Inline specification is only a request to compiler
- Inline functions should be defined in header files
  - In order to expand the code the compiler must have access to function declaration

Class Member Functions

- Member function may access private members of its class
- **this pointer**: Implicit parameter of a member function.
  - this points to object on which the function is invoked.
  - It is a pointer to the class type.
  - In a const member function the pointer is a pointer to const.
- **const member function**: Function that is member of a class and that may be called for const objects of that type.
  - const member functions may not change the data members of the object on which they operate.
  - Example: int MyClass::fct() const {}

Overloaded Functions (std 13)

- **overloaded function**: Function having the same name as at least one other function
  - Overloaded functions must differ in the number or type of their parameters
- Main function may not be overloaded
- When not to overload: keep function names and operator behavior intuitive!

Overload Resolution I

- **function matching (overload resolution)** : Compiler process by which a call to an overloaded function is resolved
  - Arguments used in the call are compared to the parameter list of each overloaded function
  - In C++ name lookup happens before type checking at compile-time

Overload Resolution II

- Steps in overload resolution:
  1. Candidate functions
  2. Determine viable functions (default arguments are treated the same way as other arguments)
  3. Find best match, if any

Overload Resolution III

candidate functions: Set of functions that are considered when resolving function call.

Candidate functions are all functions with the name used in the call for which a declaration is in scope at time of call.

viable functions:Subset of overloaded functions that could match a given call.

Viable functions have the same number of parameters as arguments to the call and each argument type can potentially be converted to corresponding parameter type.

Overload Resolution IV (std p. 316)

ambiguous call: Compile-time error that results when there is not a single best match for a call to an overloaded function.

best match: Single function from a set of overloaded functions that has the best match for the arguments of a given call.

Argument-type Conversions I

- Argument-type conversions in descending order:
  1. Exact match: argument and parameter type are the same
  2. Promotion: integral types like char, short are converted to int
  3. Standard conversions: like double to int
  4. Class type conversions

Argument-type Conversions II

- Arguments should not need casts when calling overloaded functions
- Whether a parameter is const only matters when the parameter is a reference or pointer

Pointers to Functions I

- Parentheses around function name are necessary
  - Example: bool (*pf)(const string&, const string&);
- Use typedefs to simplify pointer definitions
  - Example: typedef bool (*ptrfct)(const string&, const string&);
- Function pointer may be initialized and assigned only by a function (pointer) that has the same type or by a zero-valued constant expression
  - Example: ptrfct pf1 = 0;

Pointers to Functions II

- We can define a parameter as a function type
  - Example: void fct (const string&, bool (*) (const string&));
  - Equivalent: void fct (const string&, bool (const string&));
- A function return type must be a pointer to function, it cannot be a function
  - Example: int (*ff(int)) (int*, int);
  - Equivalent: typedef int (*PF) (int*, int); PF ff(int);

Lambda expressions (std 5.1.5)

- Callable unit of code
- A lambda is somewhat like a unnamed, inline function
- Syntax:
  - [capture list] (parameter list) -> return type { function body }
  - return type, parameter list, and function body are same as for ordinary functions
  - capture list is an (often empty) list of local variables defined in the enclosing function
  - capture list and function body are obligatory
  - Example: auto f = [] { return 42; }