Ch 9: Functions

CSCI 330

Overview

- Function Basics
- Default Arguments
- Function Overloading
- Inline Functions
- Lambdas
- Recursion
- Name Mangling and extern "C"

Basics:

- Definition: A function is a reusable block of code that performs a specific task.
- General form (Syntax):

```
return_type function_name(parameter_list) {
    // body
    return value;
}
```

```
int add(int a, int b) {
    return a + b;
}
```

Function Declaration vs Function Defining

- Declaration (Prototype): Tells the compiler about a function's name, return type, and parameters.
- Definition: Provides the actual implementation.
- Example:

```
int add(int, int); // Declaration
int add(int a, int b) { return a + b; } // Definition
```

Scope and Lifetime

- Function parameters and local variables are scoped to the function body.
- They are destroyed when the function exits.
- Avoid using global variables unless necessary.

Default Arguments

- C++ allows default values for function parameters.

Syntax:

void log(std::string message, int level = 1);Call Examples:

Call Examples

log(”File not found”); //level =1

log(“Fatal error”, 5); //level = 5

Function Overloading

- You can define multiple functions with the same name but different parameter types

Example

```
in square(int x) {return x * x;}
double square(double x) {return x * x;}
```

Inline Functions

- inline suggest that function code be substituted at the point of call to reduce function call overhead
- Useful for small, performance-critical functions

Syntax:

```
inline int cube(int x) {return x * x * x;}
```

Lambdas (Anonymous Functions)

- Labdas are inline, anonymous functions introduced in C++11 (the book is using C++ 17)

  Syntax

  [capture_list] (parameter_list) -> return_type { //body};

Lambda Capture “Flavors”

Syntax	Maning	Typical use
[=]	Implicit by-value capture of all used outer vars	Thread safety, immutability
[&]	Implicitly by-reference capture of all used vars	Mutating shared state
[x]	Capture x by value	Preserve snapshot
[&x]	Capture x by reference	Avoid copies
[this]	Capture the current object pointer	Member lambdas
[=, &mut]	Mix: everything by value, except mut by ref	Fine control
[y = std::move(src)]	Init capture / move capture (C++ 14)	Transfer ownership

By-Values vs By-Reference

- Note: Value capture freezes the state; reference reflects changes

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 0;
    auto val = [=]() { return n; };    // captured by value
    auto ref = [&]() { return ++n; };  // captured by reference

    n = 42;
    cout << val() << ", " << ref() << endl; // Output: 0, 43
    return 0;
}
```

Explicit Captures

- Use when only certain variables should be mutable/live

Recursive Functions

- A function that calls itself (Tree traversal, factorial, Fibonacci, etc.)

Example:

```
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

extern “c” and Name Mangling

- C++ compilers mangle function names to support overloading (extern “C” is essential for C/C++ interoperability)
- Use extern “C” to prevent name mangling when linking with C code
- Example:

```
extern “C” void c_function();
```