Lecture #4 | C/C++: function, scope

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Today's Topic

- Functions (definition & declaration)

- Header files

- Scope

- Array

- C string

Functions

- What is function in C/C++?

  - A reusable sequence of statement(s) designed to a particular job

- Why define your own function?

  - Readability: `sqrt(5)` is clearer than copy-pasting in an algorithm to compute the square root

  - Maintainability: To change the algorithm, just change the function (vs changing it everywhere you ever used it)

  - Code reuse: Lets other people use algorithms you've implemented

- `main()` is called (or invoked) after initialization of non-local objects, i.e., the entry point of program execution

Function definition and declaration

- Function definition

  return_type function_name(parameters) {

      statement;

  }

- *parameters*

  void

  [*data_type1 param1*[*, data_type2 param2*[*, ...*]]]

- Function declaration: parameter names can be omitted

  return_type function_name(parameters);

Example: function

```cpp
int raise_to_power(int base, int exponent)
{
    int result = 1;
    for (int i = 0; i < exponent; ++i)
        result *= base;
    return result;
}

int main(void)
{
    cout << "3^4 is " << raise_to_power(3, 4) << endl;
    return 0;
}
```

Function overload

- When two or more different declarations are specified for a single name in the same scope, that name is said to be *overloaded* (only in C++*)

- Overloaded functions should have different parameters, i.e., number/type of parameters; functions with different return types cannot be overloaded

```cpp
void print(int arg) {
    cout << "int value:" << arg << endl;
}

void print(double arg) {
    cout << "double value:" << arg << endl;
}

int main(void) {
    print(1);
    print(1.0);
}
```
* C11 supports similar function using _Generic()

Example: function declaration

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

int cube(int x) {
    return x * square(x);
}
int square(int x) {
    return x * x;
}
int main() {
    cout << "2^3" << cube(2) << endl;
}
```

```
$ g++ func_cube_square.cpp
func_cube_square.cpp:5:16:
error: use of undeclared
identifier 'square'
    return x * square(x);
                ^
1 error generated.
```

Example: function declaration (cont.)

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

int cube(int x) {
    return x * square(x);
}
int square(int x) {
    return x * x;
}
int main() {
    cout << "2^3" << cube(2);
    cout << endl;
}
```

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

int square(int x);
int cube(int x) {
    return x * square(x);
}
int square(int x) {
    return x * x;
}
int main() {
    cout << "2^3" << cube(2);
    cout << endl;
}
```

Header files: mostly function and class declarations*

```
/* func_cube_square.h
 * with function prototypes
 */
int cube(int);

int square(int);
```

- Only data types of return value and parameters matter
- But it is recommended to provide "meaningful" parameters names

```cpp
// func_cube_square.cpp
#include <iostream>
#include "func_cube_square.h"
using namespace std;

int cube(int x) {
    return x * square(x);
}
int square(int x) {
    return x * x;
}
int main(int) {
    cout << "2^3" << cube(2) << endl;
}
```

* or prototypes

Scope

- Scope: a portion of program text that a particular name (e.g., variable, function) is valid

- Global variable: end of a file (or whole files when used with `extern`)
  - Using global variables is **<u>DISCOURAGED!!!</u>**
  - Initialized when a program begins
  - Destroyed when a program exits

- Local variable
  - Valid until the end of block or function
  - Allocated (and initialized) when a block/function starts
  - Destroyed when a block/function exits

Scope: global variable

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

int n_count = 6;
void func()
{
    n_count++;
}
int main()
{
    cout << "n_count=" << n_count << endl;
    func(); func();
    cout << "n_count=" << n_count << endl;
}
```

- Global variable can be accessed from everywhere

- If no assignment is provided, global variables are initialized as the basic value of the type, e.g.,
  – char/int/long/…: `0`
  – double/float/…: `0.0`
  – string: null string

Scope: local variable

- Local variable can be accessed within only its local scope
- Local variable is destroyed when a block/function exits, thus the value in local variable is NOT preserved for the next loop/invocation
- Note: in python, the scope of a local variable range from its definition to the end of the function

```cpp
int n_calls;
int pow(int base, int exponent)
{
    n_calls++;
    int result = 1;
    for (int i = 0; i < exponent; i++)
        result *= base;
    return result;
}
int max(int num1, int num2)
{
    n_calls++;
    int result = num1 > num2 ? num1 : num2;
    return result;
}
int main()
{
    int result = max(pow(2,10), pow(10, 3));
    cout << result << " " << n_count << endl;
}
```

Scope: local variable (cont.)

```cpp
int main(void)
{
    int result = 1024;
    {
        int result = 42;
        cout << "result=" << result << endl;
    }
    cout << "result=" << result << endl;
    for (int i = 0; i < 2; i++)
    {
        int result = 0;
        result++;
        cout << "result=" << result << endl;
    }
}
```

Scope: recursion

```cpp
int factorial(int n)
{
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}

int main()
{
    cout << "3!=" << factorial(3) << endl;
}
```

- Whenever a function is called, a new scope is generated

- The same rule holds even though a function calls itself (recursion)

- This is the case for most programming languages (C, C++, python, Java, …)

Namespace

- An optionally-named declarative region

- The name of a namespace can be used to access entities declared in that namespace, i.e., the members of the namespace.

- The definition of a namespace can be split over several parts of one or more translation units (i.e., different files)

```cpp
// Standard iostream objects
namespace std
{
  extern istream cin;
  extern ostream cout;
  ...
} // namespace
```

```cpp
// user_program.cpp
#include <iostream>
using namespace std;

void main(void) {
    cout << "Hello, world!" << endl;
}
```

Reading List

- Learn C++

  – Chapter 6: 1, 2, 3, 6

- What good is `static_cast`?

  – http://www.stroustrup.com/bs_faq2.html#static-cast

ANY QUESTIONS?