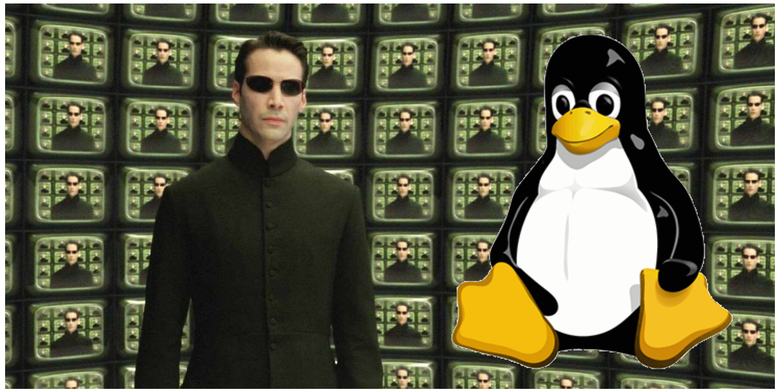
11.06.20 19:18

CSCI 125
Introduction to Computer Science and Programming II
Lecture 4: Function



Jetic Gū

2020 Summer Semester (S2)

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. How functions work
    2. Basic techniques with function
    3. Implementing an algorithm written in pseudocode

P0
Review

Data Types

- Primary √
  - Integers, Characters, Boolean √
  - Floating point √
  - Void √
- Derived
  - Function, Array√, Pointer√, Reference√
- User Defined
  - Struct, Class, Enumerate, Typedef

Review

How Functions Work

From beginning, to execution, to `return`

Summary

Function

- A group of statements (instructions) that is given a name, and can be reused easily by calling that name
- The following are all functions
  - `int main() {}`
  - `scanf`, `printf`: defined in `cstdio` (C `stdio.h`)
    // `cin`, `cout`: NOT functions, these are `objects`[1]
- Distinction: parenthesis (special cases: operators, etc.)
  - Parentheses are what differentiates functions to other declarations or statements

Concept

1. http://www.cplusplus.com/doc/tutorial/basic_io/

Function Declaration

```
type functionName (parameters) {
    statements
}
```

- Type: a function is always typed. This refers to the `return` value type.
- parameters: just like variable declarations (but with initialisation, more on this later)
- statements: just like other subroutines defined by curly brackets

Concept

1. http://www.cplusplus.com/doc/tutorial/basic_io/

Function Call

- Function call without parameters

```
printCheese();
```

- Function call with parameters

```
printCheeseNTimes(n);
```

- Function call and catching return values

```
a = power(2, n);
```

Concept

What happens when you call a function?

```
int cheese(int i) {
    stuff;
    return p;
}
a = cheese(n);
```

- In runtime, you can consider this equivalent to the righthand example code

```
int return;
// subroutine now
{
    int i=n;
    stuff;
    return=p;
}
a = return;
// then delete return
```

Concept

Think

- Why do you have to use & (address-of) for variables in `scanf`?
- Why don’t you need to use & (address-of) for variables in `printf`?

Defined Functions?

- Some call it predefined
  - Defined elsewhere. Can be your own library (`mylib.hpp`), can be C++ library, can be some other library that you install with APT
- User-Defined Functions
  - You wrote it here.

Two Types of Functions

- Value-returning

```
DataType name(param...) {stuff...}
```

  - **Must** include `return` statement
  - Once a function returns a value, it exits the subroutine

- Void function

```
void name(param...) {stuff...}
```

  - Doesn't return any value, so no need for `return` statement

Concept

Basic techniques with function

How to make your life easier

Summary

P2
Techniques

When should you put your stuff into functions?

- When it makes logical sense!
  - When the same set of operations are being **executed multiple times** in multiple places
  - When it’s more intuitive to do **recursive calls**
  - When you have a huge chunk of code **doing just one thing**
- This is just a recommendation. Personal preferences are also important.

Concept

Wait! Before you implement it!

- Check if there are C++ Standard libraries that does it for you already
  e.g. power function `pow()` comes from `cmath` library

- Check if there are third-party libraries that does it for you already
  e.g. support for audio format flac comes from xiph.org

- Check if you've written it in the past
  e.g. I wrote a very efficient linear algebra library for a few of my projects during undergrad

- Google is your friend, or you can use `man` command

Wait! Before you implement it!

```
$ man math
```

```
MATH(3)                   BSD Library Functions Manual                   MATH(3)

NAME
     math -- mathematical library functions

SYNOPSIS
     #include <math.h>

DESCRIPTION
     The header file math.h provides function prototypes and macros for work-
     ing with floating point values.

     Each math.h function is provided in three variants: single, double and
     extended precision.  The single and double precision variants operate on
     IEEE-754 single and double precision values, which correspond to the C
     types float and double, respectively.

     On Intel Macs, the C type long double corresponds to 80-bit IEEE-754 dou-
     ble extended precision.  On iOS devices using ARM processors, long double
     is mapped to double, as there is no hardware-supported wider type.

     Details of the floating point formats can be found via "man float".

:
```

Technical

Use Functions from Other Libraries

- Import the library
  - `#include <cmath>`
- Check the definition and description (e.g. on Google)
  - `double pow(double base, double exponent);`
- Use it, fill-in the parameters in/with the right order/type
  - `a = pow(2, 12);`
  - `a = pow(base=2, exponent=12); // this does NOT work`
    `// C/C++ does not support named parameters`

Concept

P2
Techniques

Use Functions from Other Libraries

- Some library will require extra parameters when you compile
- e.g. GNU `readline` library
  - Compile with
    ```
    g++ code.cpp -L/usr/local/lib -I/usr/local/include
    -lreadl
    ```
  - For this course, this won't happen

Concept

P2
Techniques

Tip 1: A function can call itself

```
int a() {
    ...doing stuff...;
    ..... = a();
    ...doing stuff...;
}
```

- This is OK, and is called a **recursive** function, common in divide-and-conquer
- BUT! Be careful with infinite loops: a recursive call that never ends

Tips

Tip 2: Declared before Calling

```
a();
void a() {
    ...;
}
```

- This won't work!

```
void a() {
    ...;
}
a();
```

- This works

Tip 3: Calling each other

```
int a() {...b()...}
int b() {...a()...}
```

- This won't work!
  - Function `b` is not declared when function `a` is being compiled
- How to solve it?

```
int a();
int b();
int a() {...b()...}
int b() {...a()...}
```

- This works!
- Declare first
  this is called a **function prototype**,
  followed by semicolon

Tip 4: Array stuff

```
void sort(int a[1000]) {...}
int a[n];
for (int i=0;i<n;i++) cin >> a[i];
sort(a);
```

- Don’t do this!
  - This is wasting memory space, since you are declaring a large array
- This is better

```
void sort(int a[]) {...}
void sort(int* a) {...}
```

Tip 5: Changing a param

```
void increase(int n) {...}

increase(a);
```

- `a` will not be changed!
  - Upon executing the function, a new variable `n` is getting created, you are changing that value.
- This is better

```
void increase(int& n) {...}

void increase(int * n) {...}    // call with increase(&a)
```

Tip 6: Terminate

```
int foo(int n) {
    ...
    if (stuff) return XXX;
    ...
    return YYY;
}
```

- Sometimes you may be able to get the answer quicker, then you can just return the value and exit the subroutine. Any return statement will terminate the subroutine.

P3
Implement

Implementing an algorithm written in pseudocode

Using none other than: functions!

Summary

P3
Implement

Palindrome Number (P10)

- A nonnegative integer is a palindrome if it reads forward and backward in the same way
  e.g.: 3, 545, 9778779

Technical

P3
Implement

Palindrome Number (P10)

- Recursive design
  - Get the most significant digit and least significant digit, compare them
    - if the same, proceed to the next
    - if not, return false

Technical

P3
Implement

Palindrome Number (P10)

- Pseudocode (recursive):

```
function isPalin(num):
    if (num<10) return true;
    first = leastSigDigit(num);
    last = mostSigDigit(num);
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
```

- Implementation
  - Translate line-by-line to programming language

Technical

P3
Implement

Palindrome Number (P10)

```
int isPalin(int num) {
    if (num<10) return true;
    first = leastSigDigit(num);
    last = mostSigDigit(num);
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
}
```

Technical

P3
Implement

Palindrome Number (P10)

```
int isPalin(int num) {
    if (num<10) return true;
    first = leastSigDigit(num);
    last = mostSigDigit(num);
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
}
```

Technical

P3
Implement

Palindrome Number (P10)

```
int isPalin(int num) {
    if (num<10) return true;
    first = num % 10;
    last = mostSigDigit(num);
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
}
```

Technical

P3
Implement

Palindrome Number (P10)

```
int isPalin(int num) {
    if (num<10) return true;
    first = num % 10;
    last = num;
    while (last>=10)
        last /= 10;
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
}
```

Technical

P3
Implement

Palindrome Number (P10)

```
int isPalin(int num) {
    if (num<10) return true;
    first = num % 10;
    last = num;
    while (last>10)
        last /= 10;
    if (first != last) return false;
    return isPalin(num with first/last digits removed)
}
```

Technical