Xi'an Jiaotong-Liverpool University
西交利物浦大學

CPT104 - Operating Systems Concepts

Lab 3

Functions, Recursions

Functions

- main, printf and scanf are functions. How do we write our own functions?
  - As an example, write a function sum that sums three integers!

```c
#include <stdio.h>
int sum(int a, int b, int c) {
    return a+b+c;
}
int main() {
    int a, b, c, result;
    scanf("%d%d%d", &a, &b, &c);
    result = sum(a, b, c);
    printf("%d\n", result);
    return 0;
}
```

Similar to methods you learned
a function has a return type, a name, a parameter list, a body that contains the definition/implementation

```
return-type function-name(argument declarations)
{
    declarations and statements
}
```

return statement might be required.

```
return expression;
```

refer to book 4.1

Copy paste, compile in Codecast, run step-by-step using Step Into

sum function must be declared **before** the main function that calls it

Parameters and Arguments

- The values in arguments in main are passed into the parameters of sum

Variables

```
sum(1, 2, 3)
  int a = 1 0
  int b = 2 0
  int c = 3 0
main()
```

Source

```c
#include <stdio.h>

int sum(int a, int b, int c) {
    return a+b+c;
}

int main() {
    int a, b, c, result;
    scanf("%d%d%d", &a, &b, &c);
    result = sum(a, b, c);
    printf("%d\n", result);
    return 0;
}
```

Terminal

```
1 2 3
```

parameters

arguments

next, the computed value is returned to main,
in this case, stored in result

Prototype and Definition of a Function (1)

- What happened if we define the sum function after the main function?
  - The compiler will complain, because at this line, it does not know sum yet

```c
#include <stdio.h>
int main() {
    int a, b, c, result;
    scanf("%d%d%d", &a, &b, &c);
    result = sum(a, b, c);
    printf("%d\n", result);
    return 0;
}
int sum(int a, int b, int c) {
    return a+b+c;
}
```

Prototype and Definition of a Function (2)

- You can tell C that the function will be defined below, by using function prototypes
  - It tells C the name, the return types, how many parameter and their types

```c
#include <stdio.h>
int sum(int a, int b, int c);
int main() {
    int a, b, c, result;
    scanf("%d%d%d", &a, &b, &c);
    result = sum(a, b, c);
    printf("%d\n", result);
    return 0;
}
int sum(int a, int b, int c) {
    return a+b+c;
}
```

terminate the prototype with **a semicolon**

you can also write the prototype without parameter names, just the types:
`int sum(int, int, int);`

let's use this !

you can then write the **function definition** below

always use prototypes, especially when there are *many* functions that call *other* functions

Recursion

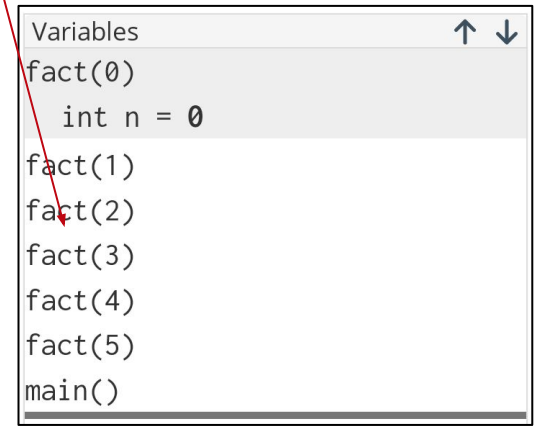
- Recall that to solve a problem recursively, you define:
  - Base Case: the simplest case that can be immediately solved
  - Recursive Step: call the same function on input of smaller size, and then do some computation on it
- For example, factorial fact(n) is defined as n x (n-1) x … x 3 x 2 x 1; and fact(0) is 1
  - We can compute factorial recursively by fact(n) = n x fact(n-1)
- Thus, for the factorial problem:
  - Base Case: when n is 0, simply return 1
  - Recursive Step: we call fact on input of smaller size, that is, fact(n-1), and then multiply the recursive call result with n

Recursive Factorial

- Write a factorial function recursively, with main function read the input integer ≥ 0, call that function, and then display the result

```c
#include <stdio.h>
int fact(int);
int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", fact(n));
    return 0;
}
int fact(int n) {
     // base case
     if (n == 0) {
         return 1;
     }
     // recursive step
     return n * fact(n-1);
}
```

try running in Codecast step-by-step using Step Into to view the recursive calls in action



Thank you for your attention !

- In this lab, you have learned:
  - Function
    - Function prototype
    - Function definition
    - Parameters, arguments
  - Recursion
    - Base case
    - Recursive steps

- **For more information:**

✓ refer to book chapter 4, 4.1-4.6, 4.10