

Pointer, Functions, Performance

CMPT 125

Mo Chen

SFU Computing Science

6/1/2020

Lecture 3

Today:

- Data vs Pointers
- Functions in C
- Performance Measurements of Code

Data vs Pointers (Review)

- Besides its data and its type, a variable needs a memory location to place the data.
 - the variable's address (a number)
 - each variable has a distinct address, i.e., they may not overlap
- The C language allows programs to store and manipulate these addresses
 - called a *pointer*

Pointer Operations in C

```
int main ( ) {  
    int area = 25;  
    int * pArea = &area;  
  
    printf("area = %d\n", area);  
    printf("pArea = %ld\n", pArea);  
    printf("pArea = 0x%lx\n", pArea);  
}
```

Output:

```
area = 25  
pArea = 140734562585432  
pArea = 0x7fff519c4b58
```

- a “*” in front of the var name means *pointer*
- the “&” operator means “*address of*”
 - saw before when using `scanf ("%d", &var);`

Pointer Operations in C

```
int main ( ) {  
    int area = 0;  
    int * pArea = &area;  
  
    *pArea = 25;  
    printf("area = %d\n", area);  
    *pArea = *pArea + 50;  
    printf("area = %d\n", *pArea);  
}
```

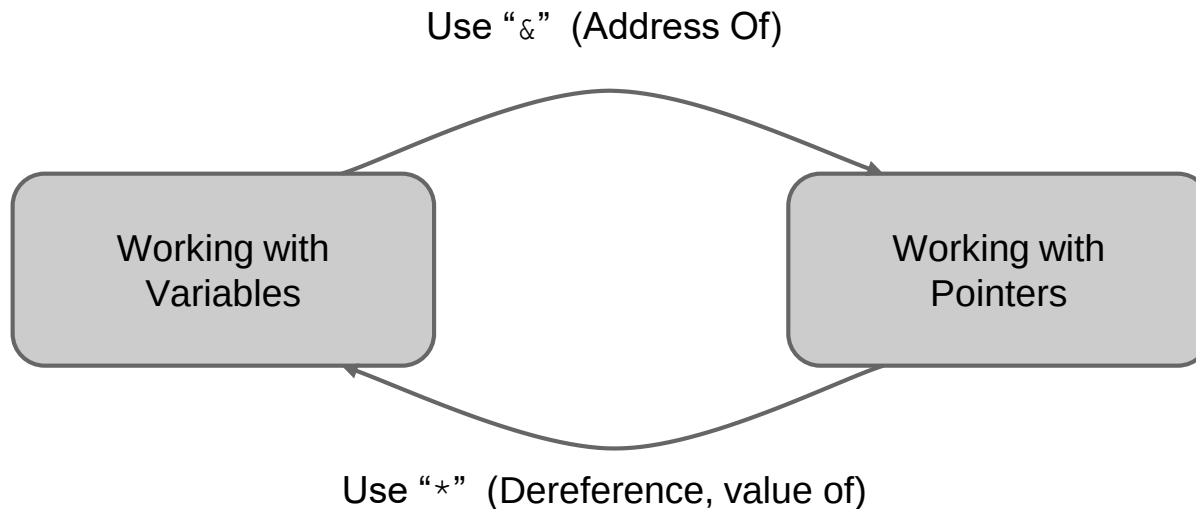
Output:

```
area = 25  
area = 75
```

- the “*” operator means *dereference*, or “*value of*”
 - use / modify the data where the pointer points

Pointer Operations - Recap

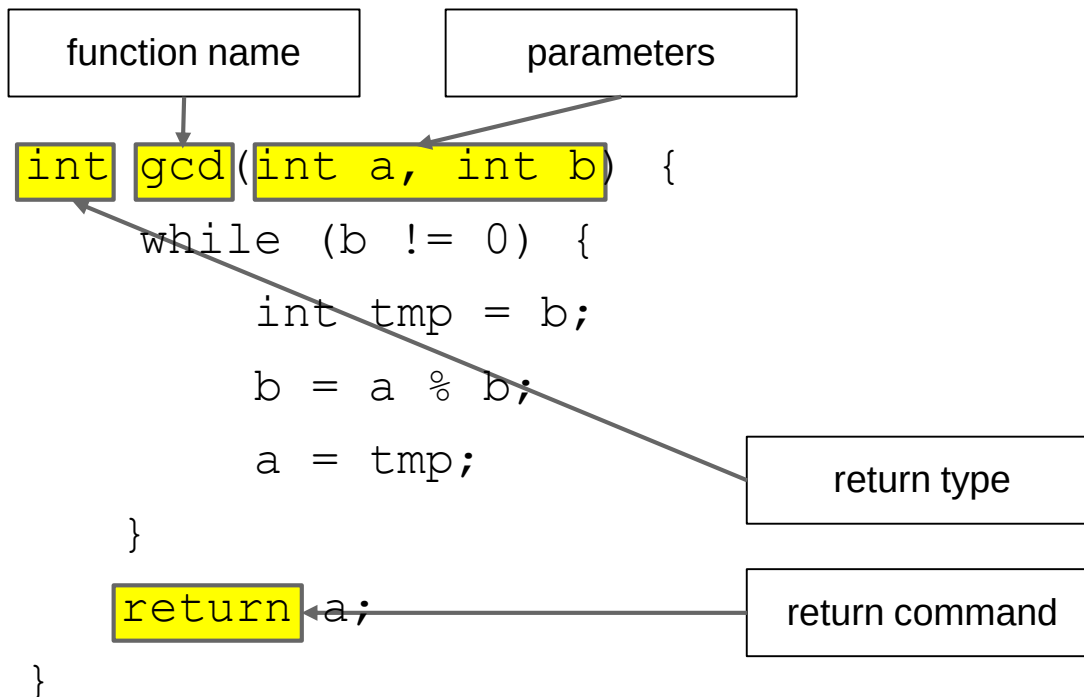
- Remember the difference between:
 - the data (variable)
 - the address (pointer)



Q. How are these operators related to each other?

Functions

- Define functions outside of main program
 - `main( )` is itself a function!
- Anatomy of a function:



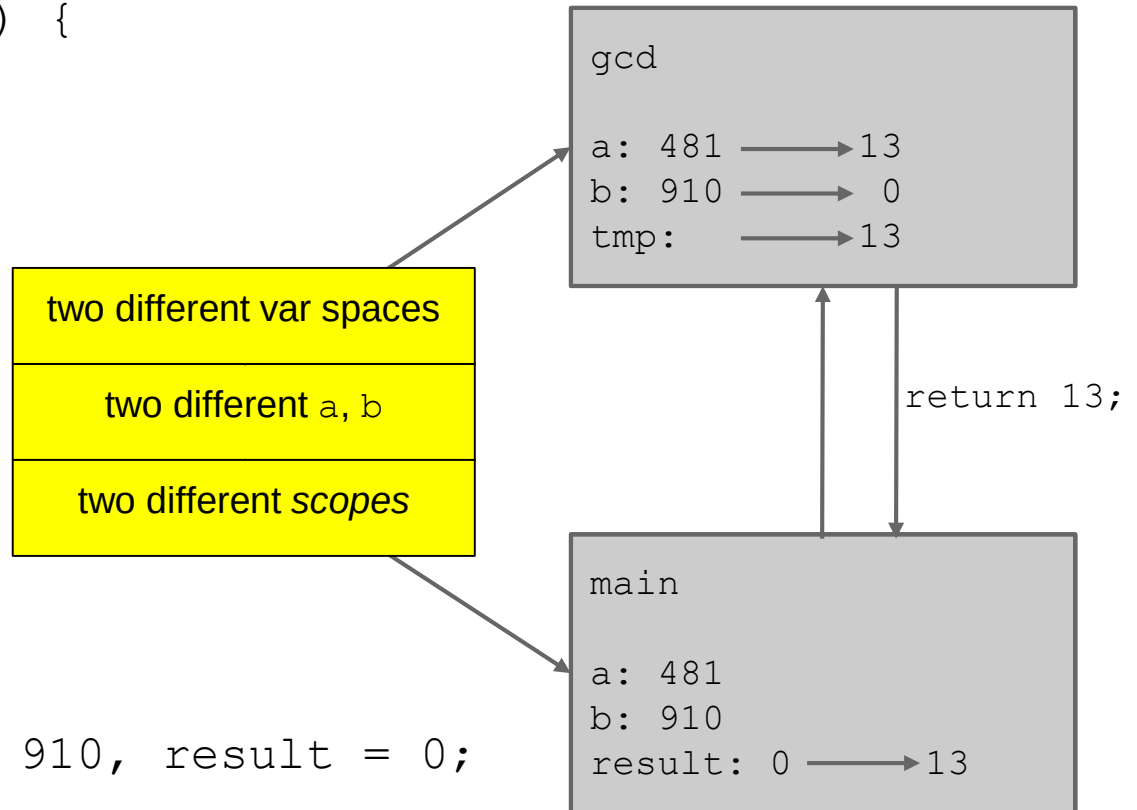
Pass By Value

- All functions in C pass parameters by value
 - call the subroutine, and it gets its own copy
 - each copy within its own *scope*
 - avoids *side-effects*: calling a function should not (unexpectedly) modify its parameters
- All functions in Python pass parameters by reference
 - side-effects only if the data is mutable
- Java is a mix

Experiment

```
int gcd(int a, int b) {  
    while (b != 0) {  
        int tmp = b;  
        b = a % b;  
        a = tmp;  
    }  
    return a;  
}
```

```
int main ( ) {  
    int a = 481, b = 910, result = 0;  
    result = gcd(a, b);  
    printf("gcd(%d,%d) = %d\n", a, b, result);  
}
```



output: gcd(481,910) = 13

Pointers as Parameters

To modify variables outside of scope, pass a pointer to that variable

```
void swap(int a, int b) {  
    int tmp = a;  
    a = b;  
    b = tmp;  
    return;  
}
```

```
int main ( ) {  
    int a = 5, b = 12;  
    swap(a, b);  
}
```

This won't change the values of `a`, `b` in the `main` routine. Only locally.

```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
    return;  
}
```

```
int main ( ) {  
    int a = 5, b = 12;  
    swap(&a, &b);  
}
```

Pass pointers to the integers instead, and use `*a` and `*b` (dereference) to access their values.

Functions - Summary

- Functions in C have similar syntax and operation to functions in Python
- Exceptions:
 - must define the types of all parameters
 - must define the type of return value
 - all parameters are pass by value
- Pass a pointer to modify a caller's variable

Any questions?

How Good is Your Code?

- Several measures of “good”-ness:
- Is it . . . :
 - correct? (bug-free)
 - reliable?
 - efficient?
 - affordable?
 - maintainable?
 - easy to use?

How Good is Your Algorithm?

- Efficiency is the primary focus
- Computers consume 2 major resources:
 - time
 - space (as in memory)
- Lately, time has become the most precious
 - memory is fairly cheap
 - memory is usually not a constraint

Performance Measurement

Two Options:

1. Time the code when it runs on a variety of inputs
 - plot graphs + predict behaviour
 - hardware dependent
2. Count the number of operations (steps) your algorithm performs
 - plot graphs OR derive functions OR . . .
 - . . . use the big-O estimate
 - hardware independent