

CAS CS 210: Computer Systems

C Primer
Fall 2007



Adapted from CMU's 15213 C Primer



Outline

- ◆ Overview comparison of C and Java
- ◆ Good evening
- ◆ Arrays
- ◆ Pointers
- ◆ Dynamic memory



What we will cover

- ◆ A crash course in the basics of C
- ◆ You should read the K&R C book for lots more details

Like Java, like C

- ◆ Operators same as Java:
 - Arithmetic
 - `i = i+1; i++; i--; i *= 2;`
 - `+, -, *, /, %`
 - Relational and Logical
 - `<, >, <=, >=, ==, !=`
 - `&&, ||, &, |, !`
- ◆ Syntax same as in Java:
 - `if () { } else { }`
 - `while () { }`
 - `do { } while ();`
 - `for(i=1; i <= 100; i++) { }`
 - `switch () {case 1: ... }`
 - `continue; break;`

Simple Data Types

datatype	size	values
char	1	-128 to 127
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647
float	4	$3.4 \times 10^{\pm 38}$ (7 digits)
double	8	$1.7 \times 10^{\pm 308}$ (15 digits long)

Java programmer gotchas (1)

```
{  
  int i  
  for(i = 0; i < 10; i++)  
  ...
```

NOT

```
{  
  for(int i = 0; i < 10; i++)  
  ...
```



Java programmer gotchas (2)

- ◆ Uninitialized variables
 - Instance variables (and array elements) initialized to 0, null, or false
- ◆ NOT guaranteed in C
 - May be initialized to “garbage”



Java programmer gotchas (3)

- ◆ Error handling
 - NO exceptions in C
 - Must look at return values

“Good evening”

```
#include <stdio.h>
int main()
{
    /* print a greeting */
    printf("Good evening!\n");
    return 0;
}
```

```
$ ./goodevening
Good evening!
$
```

Breaking down the code

- ◆ **#include <stdio.h>**
 - Include the contents of the file `stdio.h`
 - Case sensitive – lower case only
 - No semicolon at the end of line
- ◆ **int main()**
 - The OS calls this function when the program starts running
- ◆ **printf(format_string, arg1, ...)**
 - Prints out a string, specified by the format string and the arguments

format_string

- ◆ Composed of ordinary characters (not %)
 - Copied unchanged into the output
- ◆ Conversion specifications (start with %)
 - Fetches one or more arguments
 - For example
 - `char` `%c`
 - `char*` `%s`
 - `int` `%d`
 - `float` `%f`
- ◆ For more details: `man 3 printf`

Arrays

- ◆ **char foo[80];**
 - An array of 80 characters
 - **sizeof(foo)**
= $80 \times \text{sizeof(char)}$
= $80 \times 1 = 80$ bytes
- ◆ **int bar[40];**
 - An array of 40 integers
 - **sizeof(bar)**
= $40 \times \text{sizeof(int)}$
= $40 \times 4 = 160$ bytes



Pointers



- ◆ Pointers are variables that hold an address in memory
- ◆ That address contains another variable

Memory layout and addresses

```
int x = 5, y = 10;  
float f = 12.5, g = 9.8;  
char c = 'c', d = 'd';
```

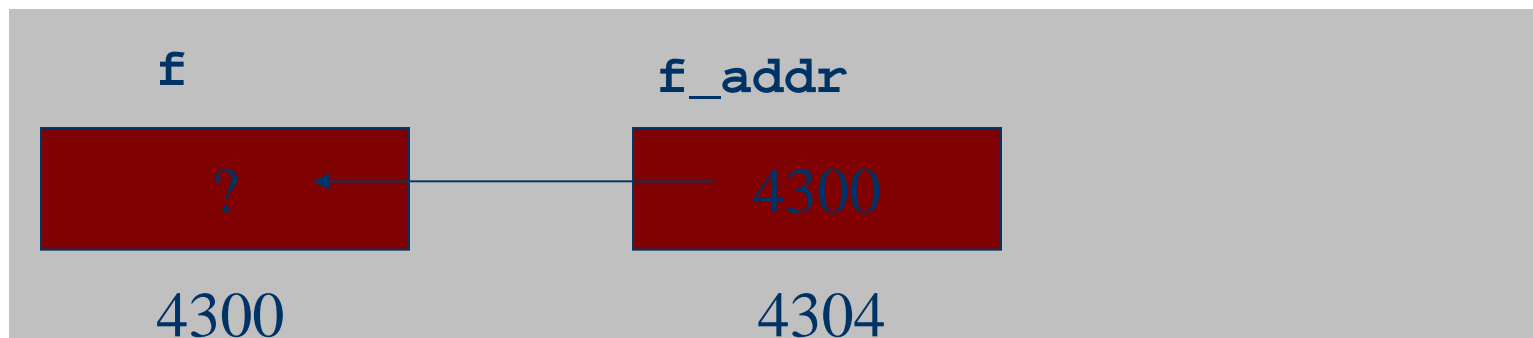


Using Pointers (1)

```
float f;           /* data variable */  
float *f_addr;    /* pointer variable */  
    f              f_addr          any float
```

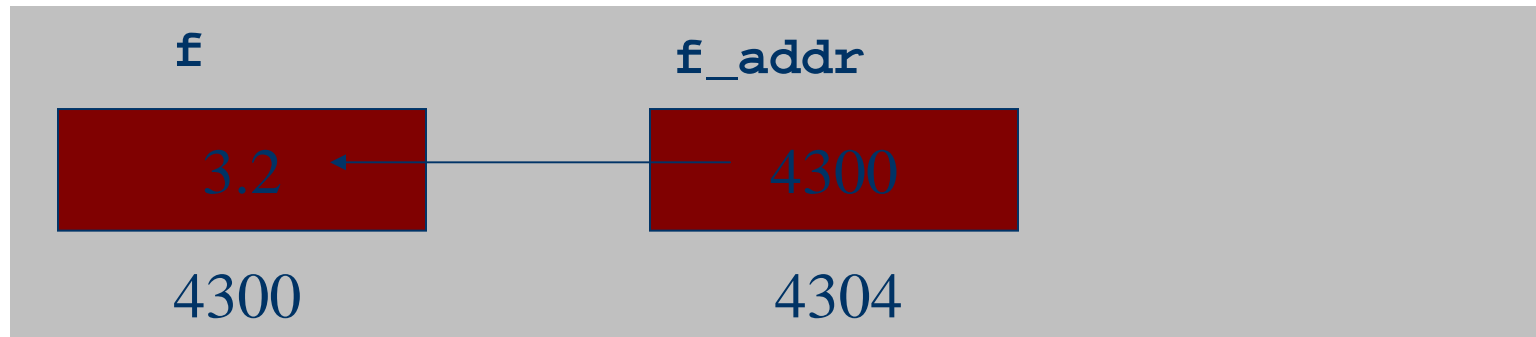


```
f_addr = &f;     /* & = address operator */
```

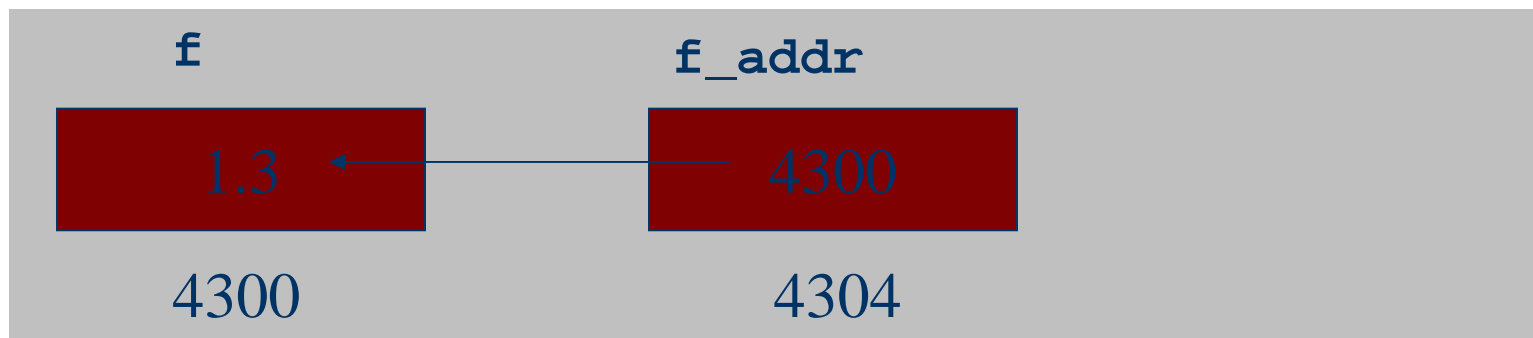


Pointers made easy (2)

```
*f_addr = 3.2; /* indirection operator */
```



```
float g = *f_addr; /* indirection: g is now 3.2 */  
f = 1.3;          /* but g is still 3.2 */
```





Function Parameters

- ◆ Function arguments are passed “by value”
- ◆ What is “pass by value”?
 - The called function is given a copy of the arguments
- ◆ What does this imply?
 - The called function can NOT alter a variable in the caller function, but its private copy
- ◆ Three examples

Example 1: swap_1

```
void swap_1(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Q: Let $x=3$, $y=4$,
after `swap_1(x,y)`;
 $x=?$ $y=?$

~~A1: $x=4$; $y=3$;~~

A2: $x=3$; $y=4$;

Example 2: swap_2

```
void swap_2(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Q: Let $x=3$, $y=4$,
after
`swap_2(&x,&y);`
 $x=?$ $y=?$

~~A1: $x=3$; $y=4$;~~

A2: $x=4$; $y=3$;

Example 3: scanf

```
#include <stdio.h>

int main()
{
    int x;
    scanf("%d\n", &x);
    printf("%d\n", x);
}
```

Q: Why using pointers in scanf?

A: We need to assign the value to x.

Dynamic Memory

- ◆ Java manages memory for you, C does NOT
 - C requires the programmer to *explicitly* allocate and deallocate memory
 - Unknown amounts of memory can be allocated dynamically during run-time with **malloc()** and deallocated using **free()**



Not like Java

- ◆ No **new**
- ◆ No garbage collection
- ◆ You ask for n bytes
 - Not a high-level request such as “I’d like an instance of class **String**”

malloc

- ◆ Allocates memory in the heap
 - Lives between function invocations
- ◆ Example
 - Allocate an integer
 - `int* iptr = (int*) malloc(sizeof(int));`
 - Allocate a structure
 - `struct name* nameptr = (struct name*) malloc(sizeof(struct name));`

free

- ◆ Deallocates memory in heap
- ◆ Pass in a pointer that was returned by **malloc**
- ◆ Example
 - `int* iptr = (int*) malloc(sizeof(int)); free(iptr);`
- ◆ Caveat: do NOT free the same memory block twice!