CS 354 - Machine Organization & Programming
Tuesday, September 10, 2019

Waitlisted? Complete the form at: https://forms.gle/CRvL1oR8i9Bymvyo6

Jim Skrentny, 5379 CS, skrentny@cs.wisc.edu
Course website: https://canvas.wisc.edu/courses/154937

Project p1 (3%): DUE at 10 pm on Monday, September 23rd

Exam Conflicts: report any by this Friday using form at: https://forms.gle/6TwXssFmUCh7o8GS8
TA lab consulting & PM drop-in hours: are scheduled, see links on course front page
Linux Workshop: tonight 5:30 pm 1240 CS repeated on Friday 5:30 pm 1240 CS

Last Time

- Course Info and Coursework
- Java vs. C
- Coding in C Remotely
  - Get Connected to CS
  - Edit your Source
  - Compile/Run/Debug

Today

- C Program Structure
- C Logical Control Flow
- Recall Variables
- Meet Pointers

Next Time

- Pointers: Arguments and 1D Arrays
- **Read:**
  - K&R Ch. 5.1: Pointers and Addresses
  - K&R Ch. 5.2: Pointers and Function Arguments
  - K&R Ch. 5.3: Pointers and Arrays
  - K&R Ch. 5.4: Address Arithmetic
- **See:** Piazza post for web alternatives to K&R readings

C Program Structure

- *Variables and functions must be declared before they're used.*

  - What is output by the following code?

3 functions.

declaration

```
#include <stdio.h>

int bing(int x) {
   x = x + 3;
   printf("bing %d\n", x);
   return x - 1;
}

int bang(int x) {
   x = x + 2;
   x = bing(x);
   printf("BanG %d\n", x);
   return x - 2;
}

int main(void) {
   int x = 1;
   bang(x);
   printf("BOOM %d\n", x);

   return 0;
}
```

parameter

argument.

okay to ignore return value.

main : bottom

Passing Arguments

argument: data to be shared with func

parameter: VAR where func stores arg

connection

pass-by-value: a copy of the arg is passed in to its corresponding parameter.

Return Value

return-by-value: a copy of the return val is passed back replacing the func call

C Logical Control Flow

Sequencing

Execution
starts in main()
Flows from top to bottom
Does one statement after another
statement seperator ";"
statment block { }

Selection

→ Which value(s) means true? ~~**true**~~ ~~42~~ ~~-17~~ ~~0~~

C has no booleen type.
True is non-zero val, False is zero.

if - else

Like Java

→ What is output by this code when `money` is 11, -11, 0? got money.

? assignment non-zero?

```
if (money = 0)        printf("you're broke\n");
else if (money < 0)   printf("you're in debt\n");
else                  printf("you've got money\n");
```

↙ assignment. and return false.

To compare, use ==

To avoid, use 0 == money.

→ What is output by this code when it's 2/14? 11/31?

nothing

```
if (10 == month)
  if (31 == day)
    printf("Happy Halloween!\n");
else
  printf("It's not October.\n");
```

"dangling else" problem
by default, the else clause pairs with nearest unpaired if.

switch

Like Java, but you can not swith on strings.

C Logical Control Flow (cont.)

Repetition

Like Java.

```
int i = 0;
while (i < 11) {
   printf("%i\n", i);
   i++;
}
```

```
for (int j = 0; j < 11; j++) {
   printf("%i\n", j);
}
```

For loop scope is same as java if compile with gcc -std=gcc99

```
int k = 0;
do {
   printf("%i\n", k);
   k++;
} while (k < 11);
```

Recall Variables

What? A *scalar variable* is AKA primitive.
a unit storage whose contents can change

→ Draw a basic memory diagram for the variable in the following code:

```
void someFunction(){
   int i = 44;
```

i | 44 |

Aspects of a Variable

identifier: its name, i, associated with the var's ADDR.

value: its data, 44, stored in var's mem

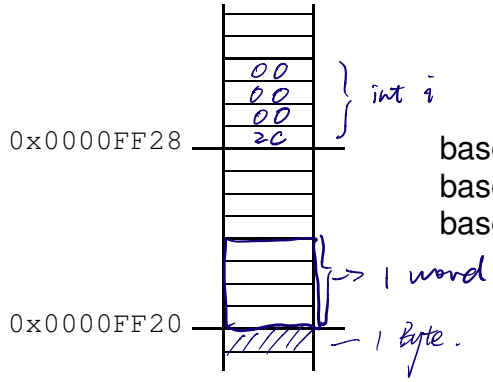
type: its kind of Data, int, how to interpret bit pattern in var's mem.

address: its starting location in mem. 0x0000FF28. (start)

size: its number of bytes of mem, int is 4 bytes.
= 1 word

Linear Memory Diagram

A linear memory diagram is A view of mem as a sequence of bytes.



	most significant							least
base 10	44							
base 2	0000 0000	0000 0000	0000 0000	0010 1100				
base 16	0 0	0 0	0 0	2 C				

1 byte

byte addressability: EACH ADDR identifies one byte.

endianess: Byte ordering of var's val when it has a size > 1 byte.

little endian: least significant side is at lowest ADDR.

big endian: Opposite

Meet Pointers

What? A *pointer* variable is

- A unit of storage who contents can change and is a mem ADDR.
- Like java references but requires more syntax and knowledge to use correctly.

Why?

- For indirect access to mem later: Function Pointer
- For indirect access to function – much later.
- Because they're commonly used in C libs and OS (operating system) code
- For access to machine mem-mapped hardware.

How?

→ Consider the following code:

```
void someFunction(){
   int i = 44;

   int *ptr = NULL;
```

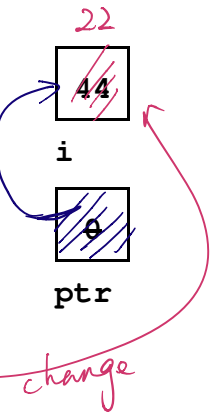
pointer variable →

ptr = &i;
(ADDR)

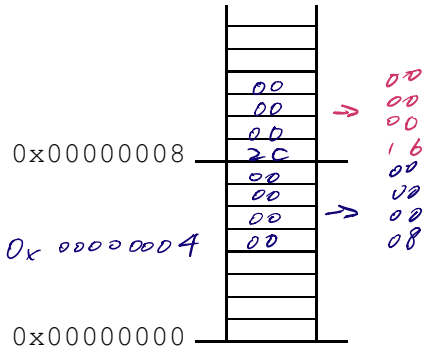
*ptr = 22;

change

Basic Diag.



Linear Diag.



→ What is ptr's initial value? NULL 0x 0000 0000 | Address? 0x00000004 | Type? int* | Size? 4 bytes (1 word)

pointer: does pointing

pointee: is pointed at

& *address of*: Return the address of its operand

***** *dereferencing*: Accesses a pointer var's pointee.