National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Lecture 4-2: C Memory Management

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

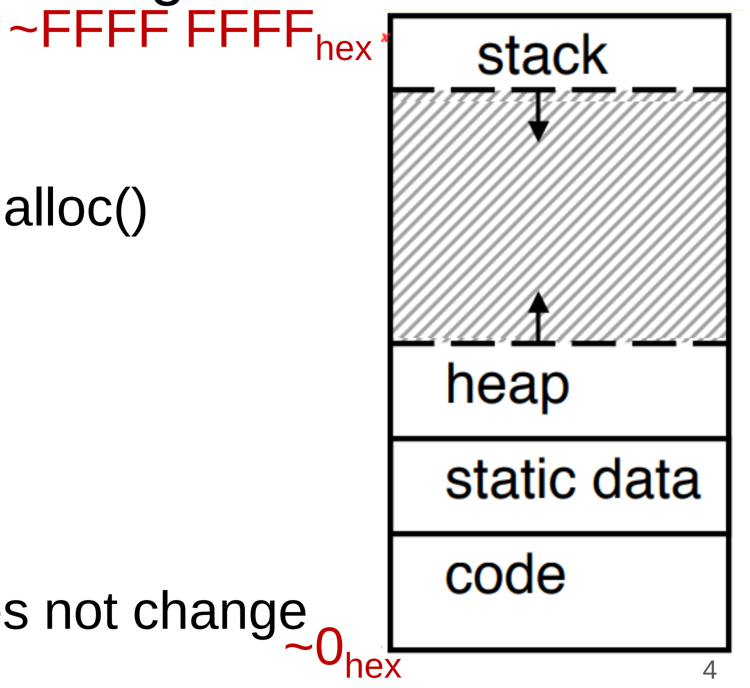
- Slides were developed in the reference with
  - CS 61C at UC Berkeley
    - https://inst.eecs.berkeley.edu/~cs61c/sp23/
  - CS 252 at UC Berkeley
    - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
  - CSCE 513 at University of South Carolina
    - https://passlab.github.io/CSCE513/

Outline

- C Memory Management

C Memory Management(1/3)

- A program's address space contains 4 regions
  - **Stack:**
    - Local variables, grows downward
  - **Heap:**
    - Space requested for pointers via malloc()
    - Resizes dynamically
    - Grows upward
  - **Static data:**
    - Variables declared outsize main
    - Does not grow or shrink
  - **Code**
    - Load when the program starts, does not change



C Memory Management(2/3)

- Variable declaration does allocate memory
  - If declare outside a procedure, allocated in static storage
  - If declare inside procedure allocated on the stack and free when procedure returns
    - main() is a procedure

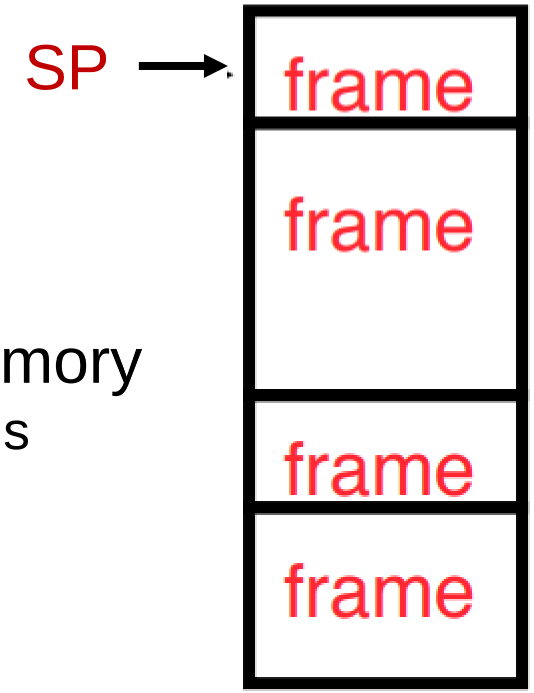
```
int myGlobal;
main() {
   int myTemp;
}
```

C Memory Management(3/3)

- C has 3 pools of memory
  - **Static storage:** global variable storage
    - basically permanent, entire program run
  - **The stack:** local variable storage
    - Parameters, return address
    - Stack frame in C
  - **The Heap:** dynamic storage
    - Malloc()
    - Data lives until deallocated by the programmer
  - C requires knowing where objects are in memory otherwise things don’t work as expected

The Stack (1/1)

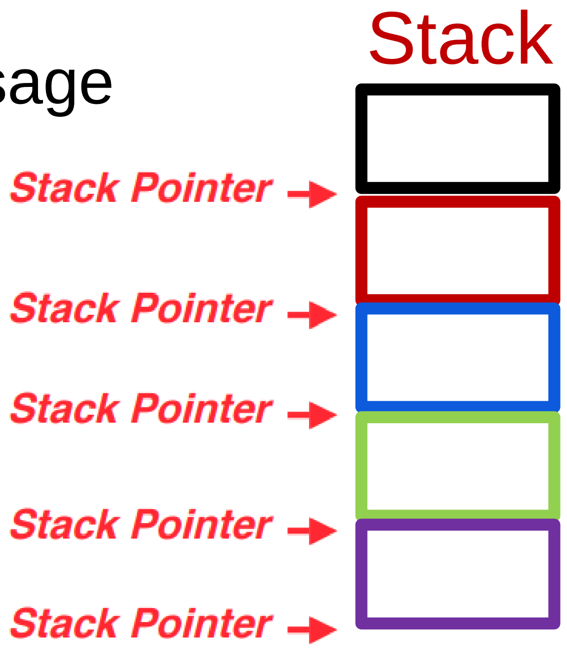
- Stack frame includes:
  - Return address
  - Parameters
  - Space for other local variables
- Stack frames contiguous blocks of memory
  - Stack pointer tells where top stack frame is
- When procedure ends
  - Stack frame is tossed off the stack
  - Free memory for future stack frame



The Stack (2/2)

- Last in, first out (LIFO) memory usage

```
main ()
{ a(0);
}
  void a (int m)
  { b(1);
  }
    void b (int n)
    { c(2);
    }
      void c (int o)
      { d(3);
      }
       void d (int p)
       {
       }
```



The Heap (Dynamic Memory)

- Large pool of memory, **not** allocated in contiguous order
- In C, specify the number of **bytes** of memory explicitly to allocate item

```
int *ptr;
ptr = (int *) malloc(sizeof(int));
/* malloc returns type (void *),
so need to cast to right type */
```

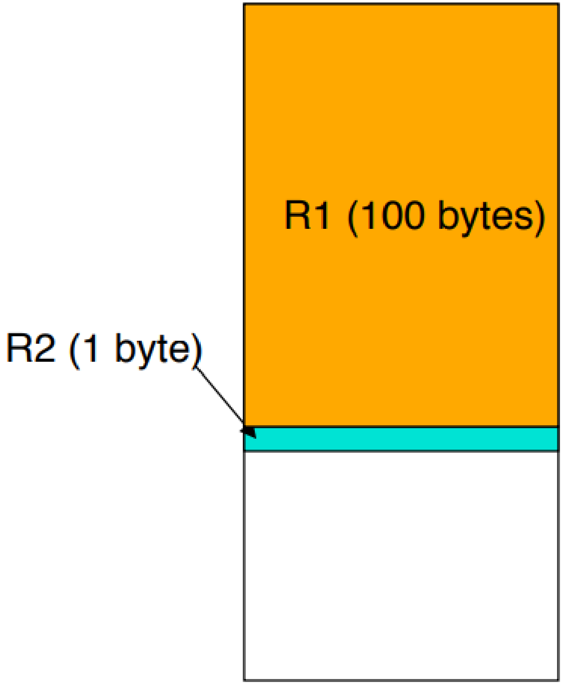
  - malloc (): Allocates raw, uninitialized memory from heap

So far …

- How do we manage memory?
  - **Code, Static storage are easy:**
    - They never grow or shrink
  - **Stack space is also easy:**
    - Stack frames are created
    - Destroyed in last-in, first-out (LIFO) order
  - **Managing the heap is tricky:**
    - Memory can be allocated/deallocated at any time

Heap Management (1/2)

- An example
  - Request R1 for 100 bytes
  - Request R2 for 1 byte
  - Memory from R1 is freed
  - Request R3 for 50 bytes



Heap Management (2/2)

- An example
  - Request R1 for 100 bytes
  - Request R2 for 1 byte
  - Memory from R1 is freed
  - Request R3 for 50 bytes

Where should we place the R3?