CSE 31
Computer Organization

Lecture 8 – C Memory Management (cont.)

Announcements

- Labs
  - Lab 3 due this week (**with 7 days grace period** after due date)
    - Demo is REQUIRED to receive full credit
  - Lab 4 out this week
    - Due at 11:59pm on the same day of your next lab (with 7 days grace period after due date)
    - You must demo your submission to your TA within 14 days from posting of lab
    - Demo is REQUIRED to receive full credit
- Reading assignments
  - Reading 01 (zyBooks 1.1 – 1.5) due **tonight**, 13-FEB and Reading 02 (zyBooks 2.1 – 2.9) due 20-FEB
    - Complete **Participation Activities** in each section to receive grade towards Participation
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses
- Homework assignment
  - Homework 01 (zyBooks 1.1 – 1.5) due 20-FEB
    - Complete **Challenge Activities** in each section to receive grade towards Homework
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

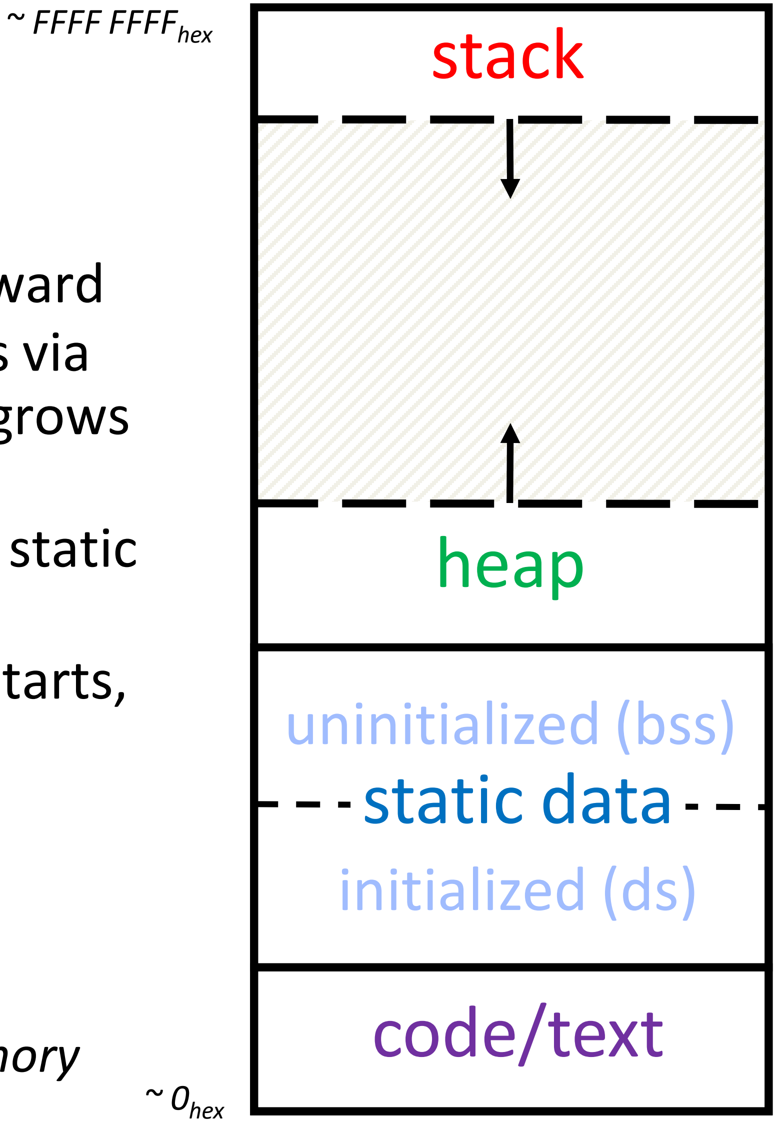
Announcements

- Project 01
  - Due 17-MAR
  - Can work in teams of 2 students
    - Each team member must identify teammate in “Comments…” text-box at the submission page
    - If working in teams, each student must submit code (can be the same as teammate) and demo individually
    - Grade can vary among teammates depending on demo
  - Demo required for project grade
    - No partial credit for submission without demo
  - No grace period
    - Must complete submission and demo by due date.

Normal C Memory Management (review)

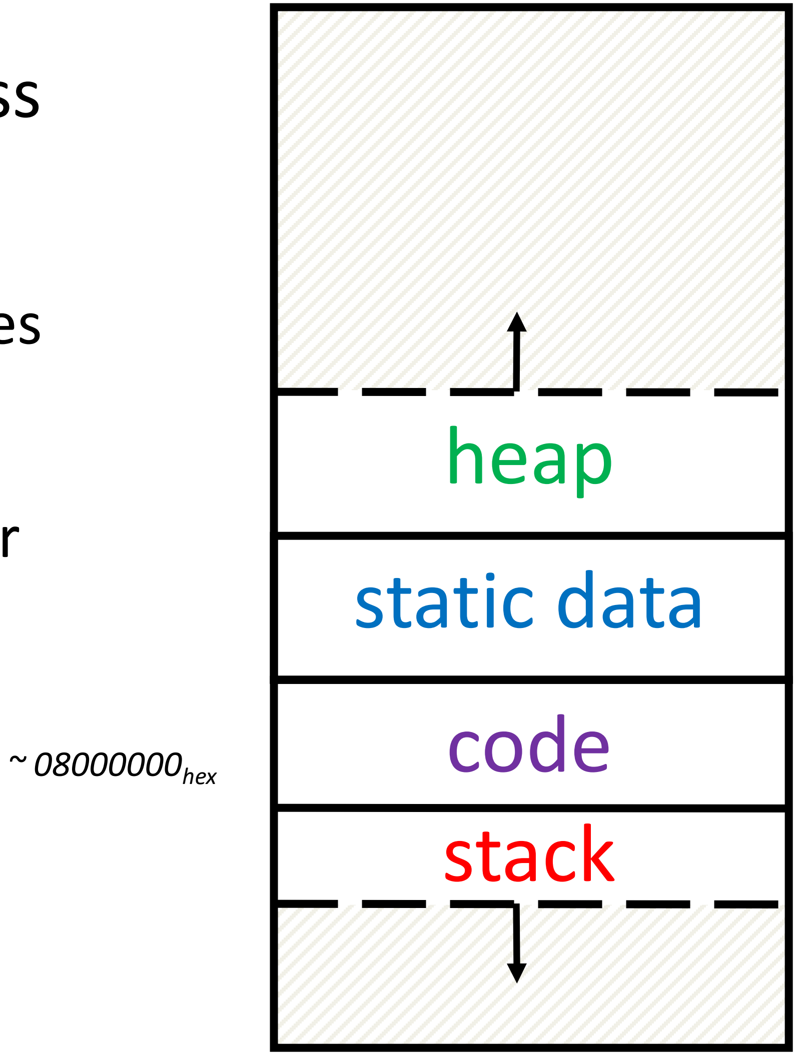
- A program's address space contains 4 regions:
  - stack: local variables, grows downward
  - heap: space requested for pointers via `malloc()` ; resizes dynamically, grows upward
  - static data: Initialized/uninitialized static and global variables
  - code/text: loaded when program starts, does not change

For now, OS somehow prevents accesses between stack and heap (gray hash lines). Wait for virtual memory



Intel 80x86 C Memory Management

- A C program’s 80x86 address space :
  - heap: space requested for pointers via `malloc()`; resizes dynamically, grows upward
  - static data: variables declared outside main, does not grow or shrink
  - code: loaded when program starts, does not change
  - stack: local variables, grows downward



Where are variables allocated?

- If declared outside of any function or using the static keyword
  - allocated in "static" storage

- If declared inside of a function
  - allocated in the "stack" (unless declared as static)
  - freed when a function returns.
    - » That's why the scope is within the function

- Note: `main()` is a function!

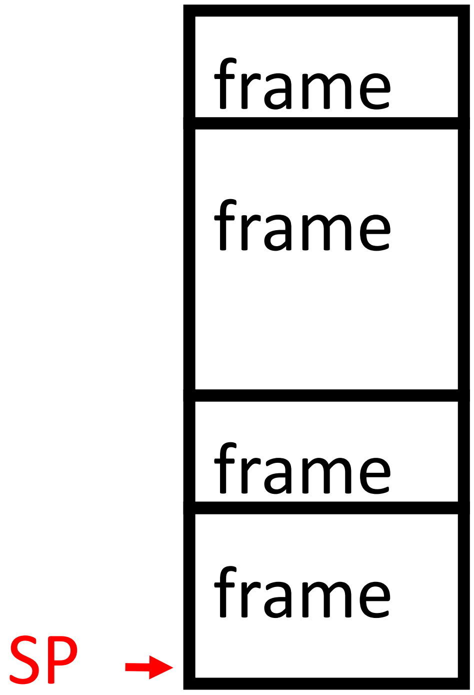
```
int myGlobal;
main() {
   static char myStatic;
   double myTemp;
}
```

Stack frames

- Stack frame includes storage for:
  - Return “instruction” address
  - Parameters (input arguments)
  - Space for other local variables

- Stack frames:
  - contiguous blocks of memory for a function
  - stack pointer tells where top stack frame is

- When a function ends, stack frame is “popped off” the stack; frees memory for future stack frames



Stack

- Last In, First Out (LIFO) data structure

stack

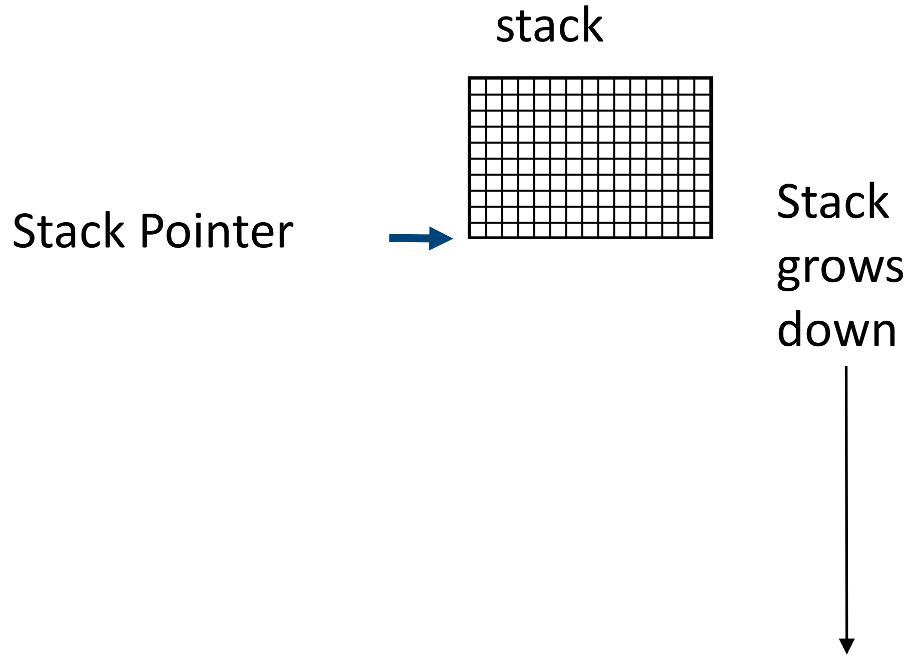
```
main (){
 a(0);
}
void a (int m) {
   b(1);
 }
void b (int n){
  c(2);
 }
void c (int o){
  d(3);
 }
void d (int p) {
}
```

Stack grows down

Stack

- Last In, First Out (LIFO) data structure

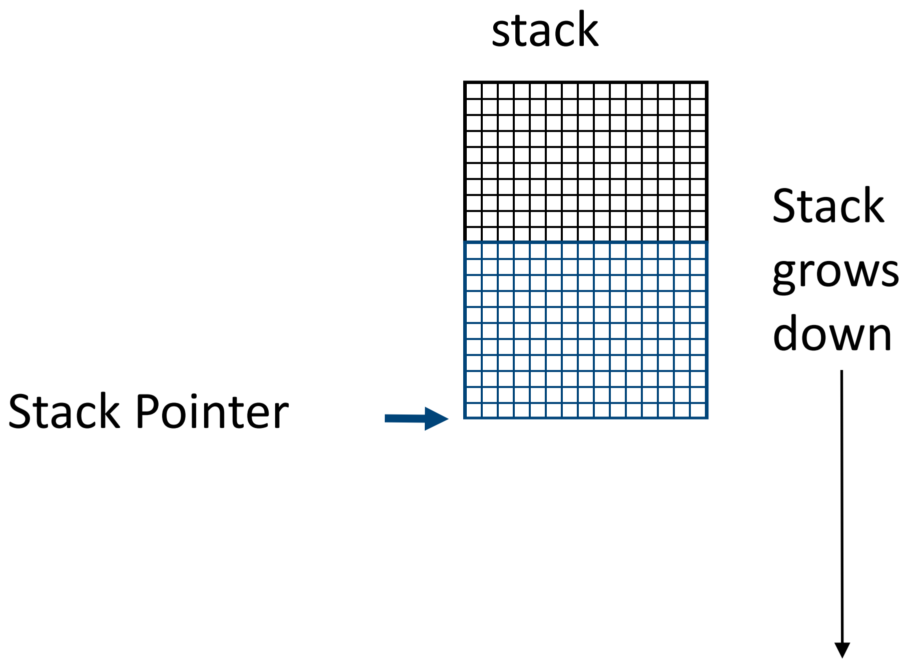
```
main (){
 a(0);
}
void a (int m) {
   b(1);
}
void b (int n){
  c(2);
}
void c (int o){
  d(3);
}
void d (int p) {
}
```



Stack

- Last In, First Out (LIFO) data structure

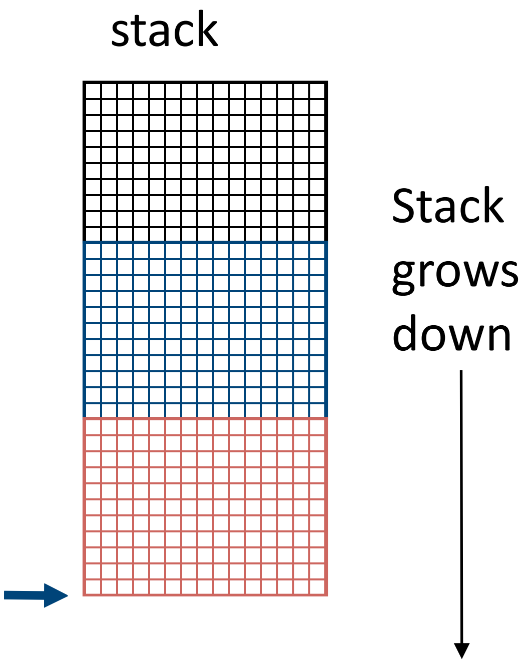
```
main (){
 a(0);
}
 void a (int m) {
    b(1);
  }
  void b (int n){
   c(2);
  }
  void c (int o){
   d(3);
  }
 void d (int p) {
 }
```



Stack

- Last In, First Out (LIFO) data structure

```
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) {
}
```

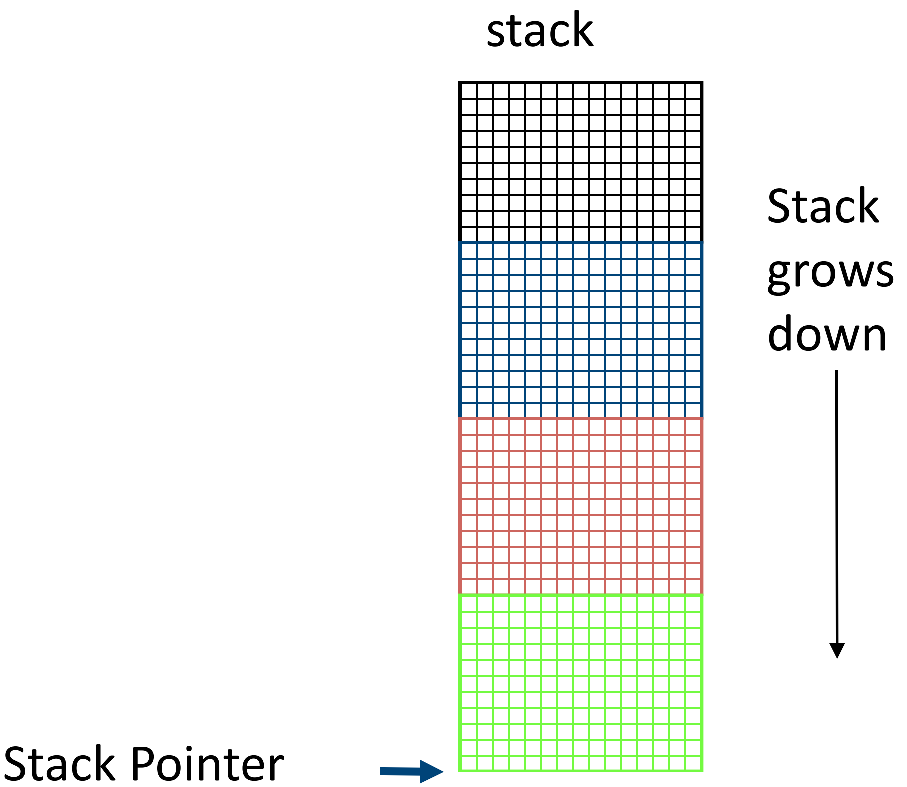


Stack Pointer

Stack

- Last In, First Out (LIFO) data structure

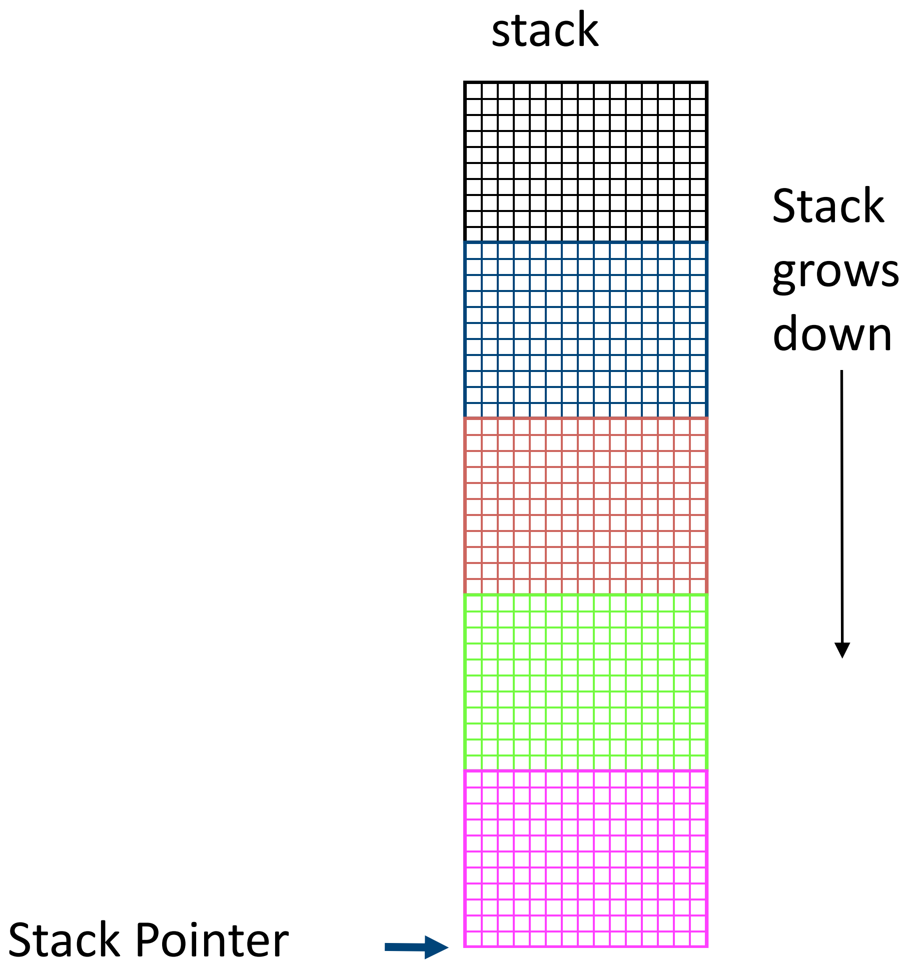
```
main (){
 a(0);
}
void a (int m) {
   b(1);
}
void b (int n){
  c(2);
}
void c (int o){
  d(3);
}
void d (int p) {
}
```



Stack

- Last In, First Out (LIFO) data structure

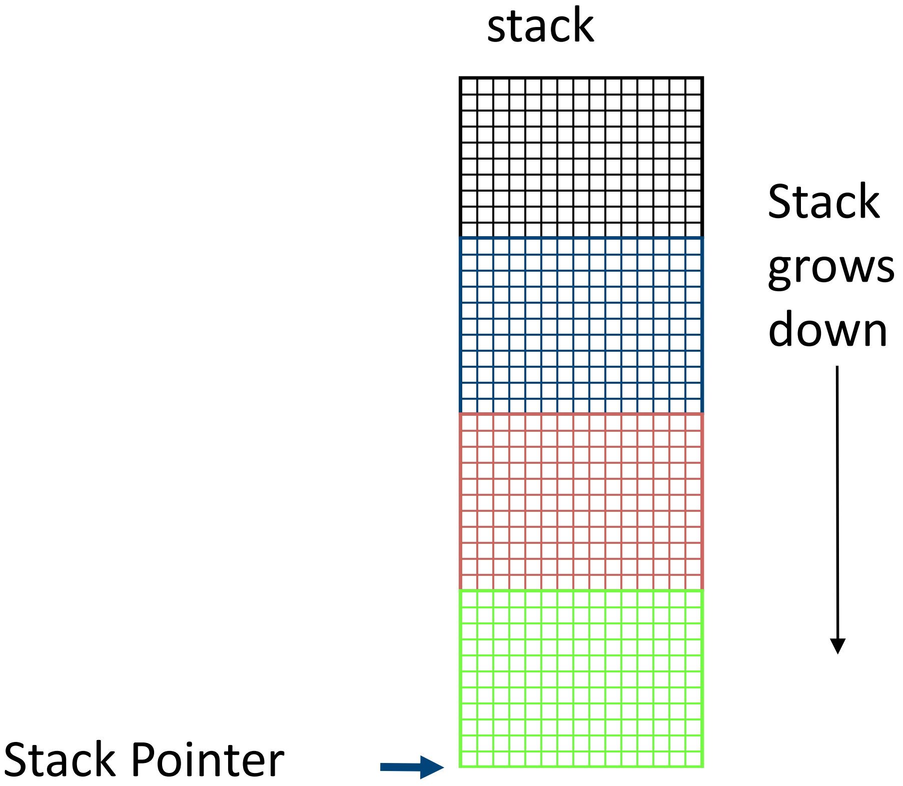
```
main (){
 a(0);
}
void a (int m) {
   b(1);
 }
void b (int n){
  c(2);
 }
void c (int o){
  d(3);
 }
void d (int p) {
}
```



Stack

- Last In, First Out (LIFO) data structure

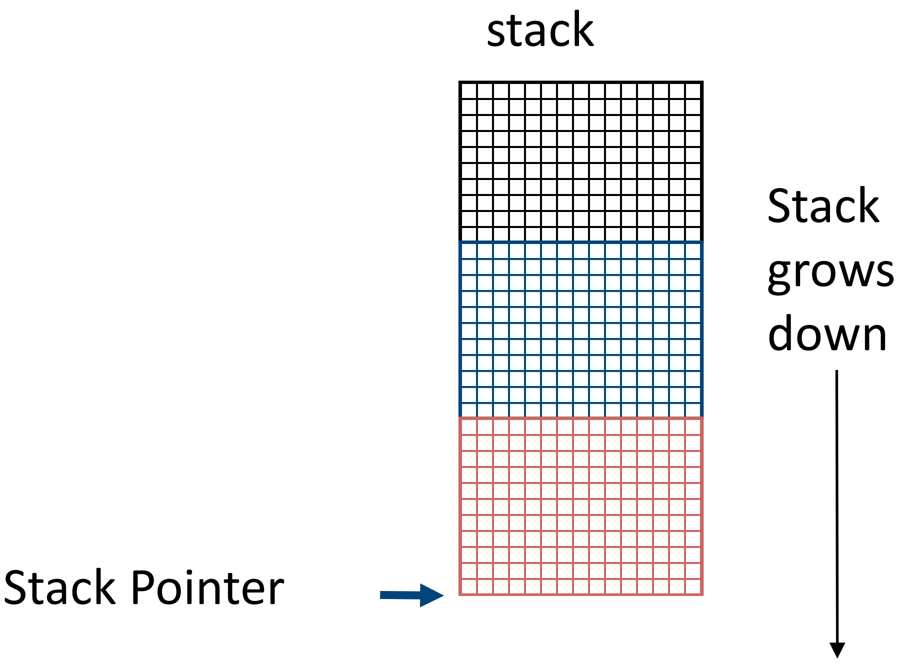
```
main (){
 a(0);
}
void a (int m) {
   b(1);
 }
void b (int n){
  c(2);
 }
void c (int o){
  d(3);
 }
void d (int p) {
}
```



Stack

- Last In, First Out (LIFO) data structure

```
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) {
}
```



stack

Stack grows down

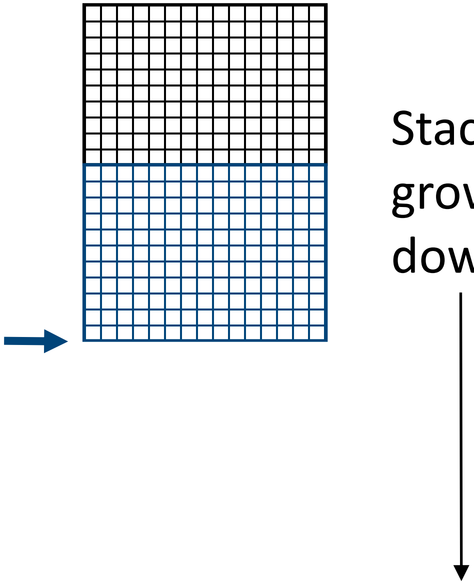
Stack Pointer

Stack

- Last In, First Out (LIFO) data structure

```
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) {
}
```

stack



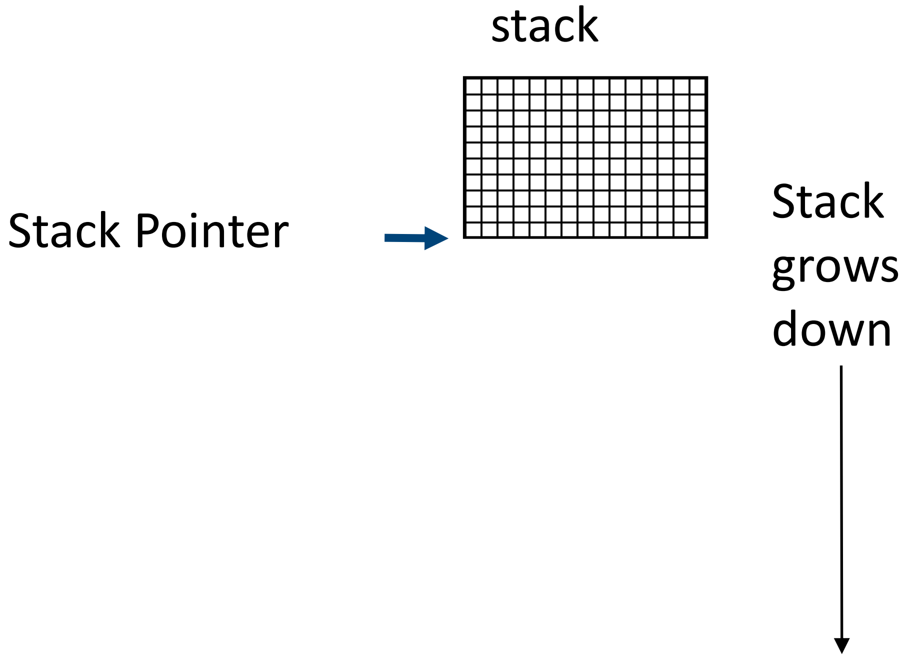
Stack Pointer

Stack grows down

Stack

- Last In, First Out (LIFO) data structure

```
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) {
}
```



Stack

- Last In, First Out (LIFO) data structure

stack

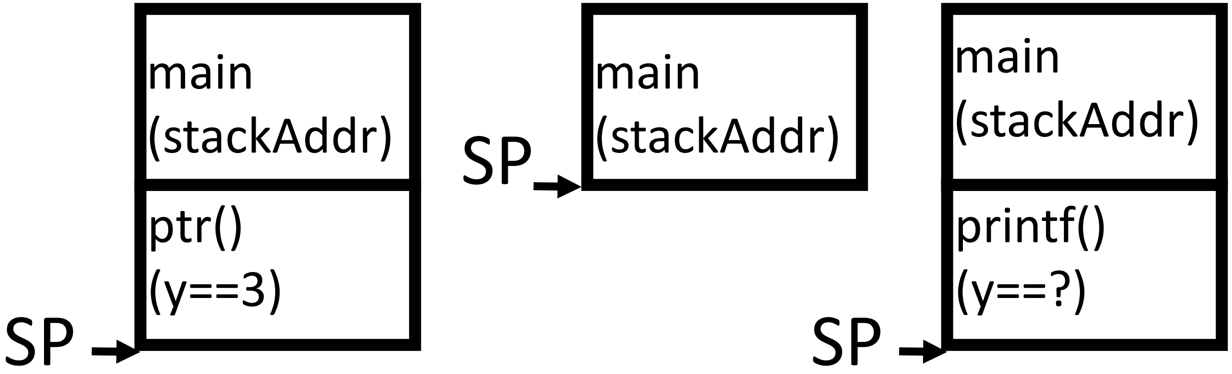
```
main (){
 a(0);
}
void a (int m) {
   b(1);
 }
 void b (int n){
  c(2);
 }
 void c (int o){
  d(3);
 }
void d (int p) {
}
```

Stack grows down

Who cares about stack management?

- Pointers in C allow access to deallocated memory, leading to hard-to-find bugs !

```
int *ptr () {
    int y;
    y = 3;
    return &y;
}
```



```
int main () {
        int *stackAddr, content;
        stackAddr = ptr();
        content = *stackAddr;
        printf("%d", content); /* 3 */
        return 0;
}
```