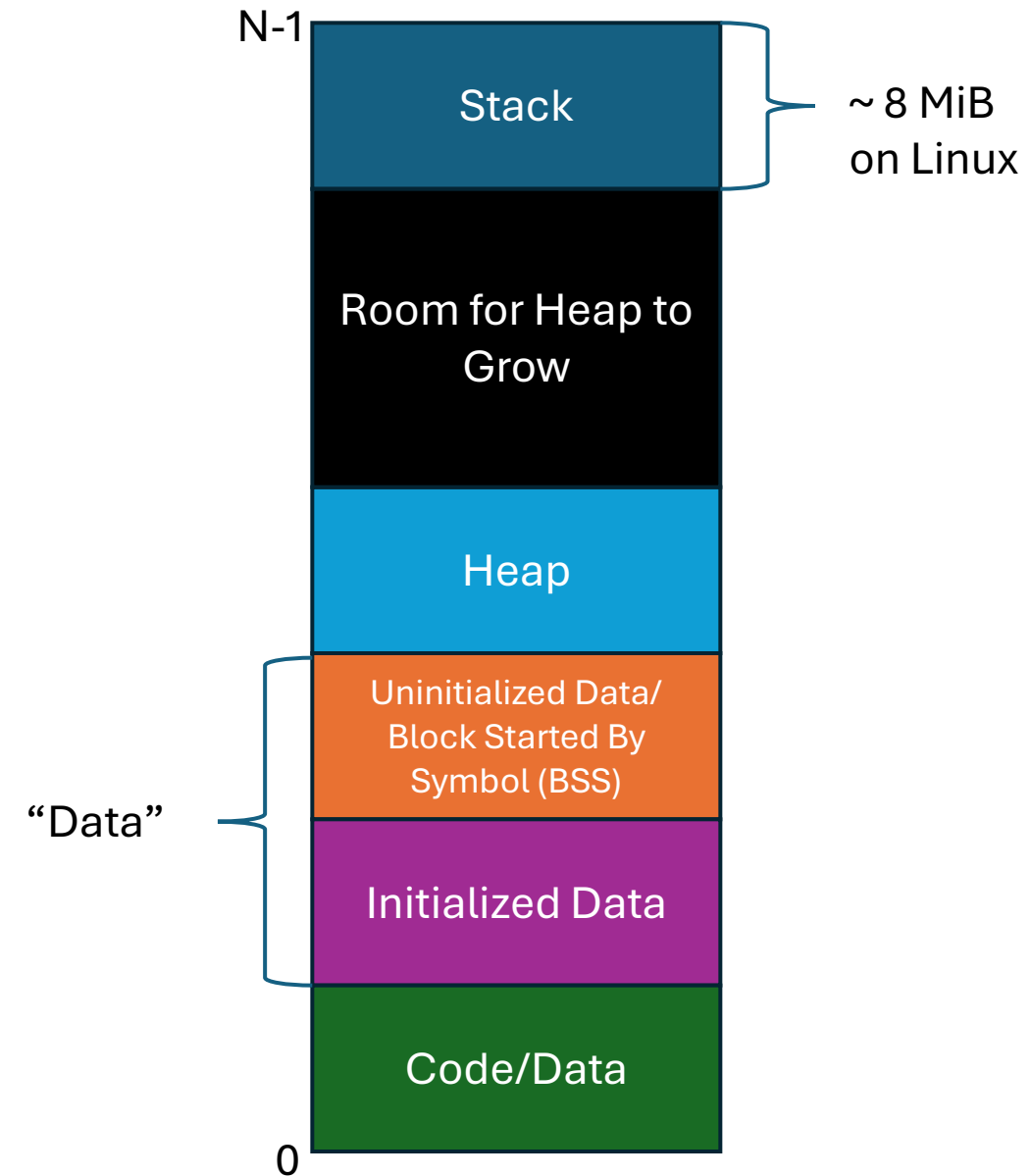


C Memory Layout

- The stack consists of “stack frames” or “activation records”. These contain local variables, and other details needed to execute a function
- The heap is for “Dynamic Storage”. We haven’t talked about that yet, but we will get there in week 7
- Uninitialized Data consists of global variables that haven’t been assigned a value
- Initialized Data consists of global variables that have been assigned a value
- The code segment holds the compiled code that is actually being executed



Pointer basics

- `int *x; // declare a pointer`
- Pointers store memory addresses, not the type specified
- `*x // “Dereference” X – access what is stored at the address stored in the pointer`
- `int y; // normal int`
- `&y // “Address of” y – get the memory address of y;`
- If X is a pointer and y is a normal int:
`x = &y; // X now “points” to y – its value is now y’s address`

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    swap_with_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

main()
x: 312341@ 0001

Every function
called exists
as a "Stack
Frame"

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    swap_with_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

```
main()  
x: 312341@ 0001  
y: 571123 @ 0002
```

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    swap_with_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

main()
x: 5 @ 0001
y: 571123 @ 0002

Addresses
don't change
when we
adjust a value

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    swap_with_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

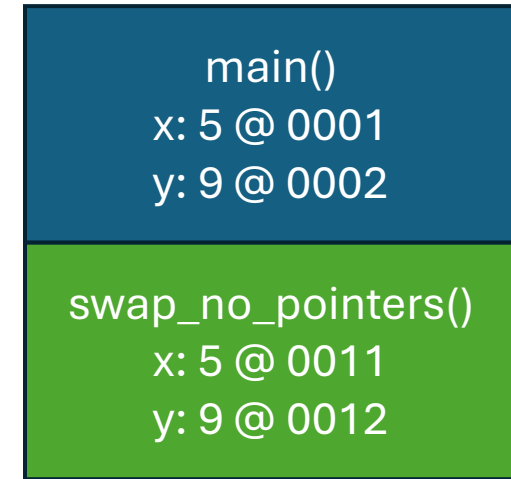
```
main()  
x: 5 @ 0001  
y: 9 @ 0002
```

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```



Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

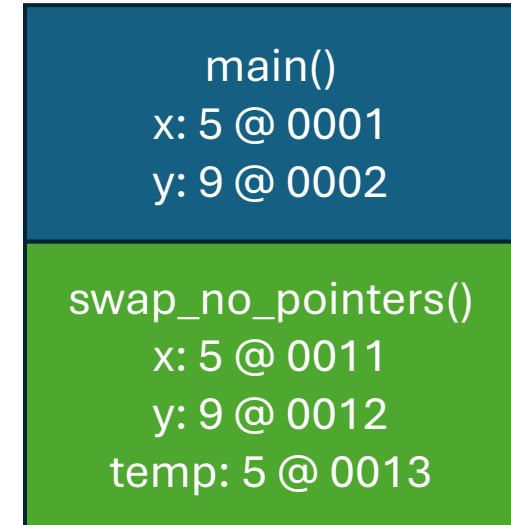
main() x: 5 @ 0001 y: 9 @ 0002
swap_no_pointers() x: 5 @ 0011 y: 9 @ 0012 temp: 23123 @ 0013

Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

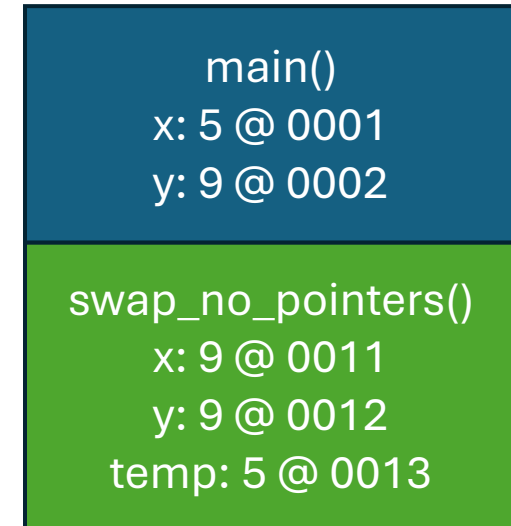


Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

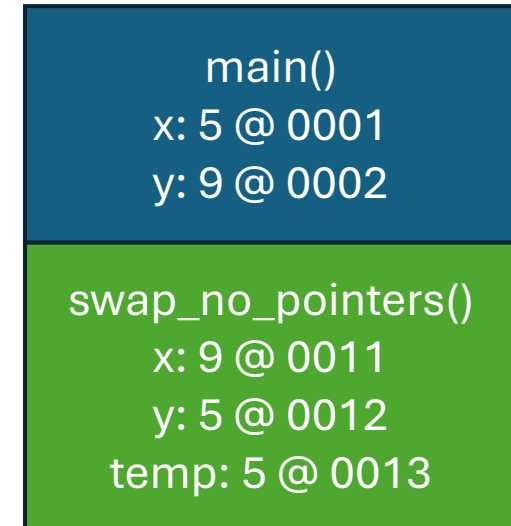


Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

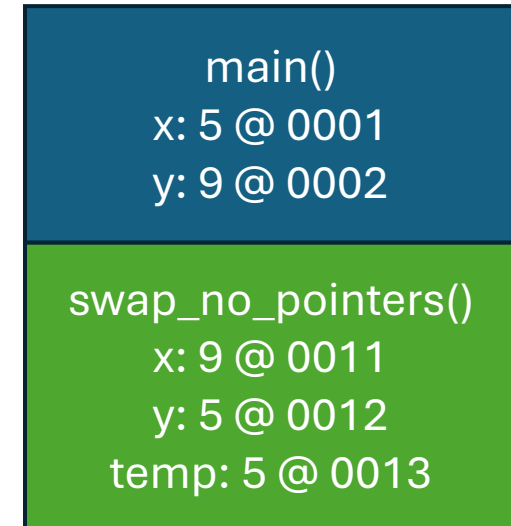


Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```



Inside the Stack

Pointer_fun.c (without prints)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_non_pointer(x,y);  
    return 0;  
}
```

```
void swap_no_pointers(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
    return;  
}
```

When we leave a function via a "return", the stack frame is deleted, and those variables are removed from memory

main()
x: 5 @ 0001
y: 9 @ 0002

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {
```

```
    int x;
```

```
    int y;
```

```
    x = 5;
```

```
    y = 9;
```

```
    swap_pointers(&x, &y);
```

```
    return 0;
```

```
}
```

```
void swap_pointers(int *x, int *y) {
```

```
    int temp;
```

```
    temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
    return;
```

```
}
```

```
main()
x: 312341@ 0001
```

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

```
main()  
x: 312341@ 0001  
y: 571123 @ 0002
```

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

```
main()  
  x: 5 @ 0001  
  y: 571123 @ 0002
```


Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

```
main()  
x: 5 @ 0001  
y: 9 @ 0002
```

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

main() x: 5 @ 0001 y: 9 @ 0002
swap_pointers() x: 0001 @ 0011 y: 0002 @ 0012

Since we want to use the same x & y in the function we call, we use &x and &y to send their addresses instead.

The pointers in the swap_pointers parameters can then store the address and reference the same x & y at 0001 and 0002

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

main() x: 5 @ 0001 y: 9 @ 0002
swap_pointers() x: 0001 @ 0011 y: 0002 @ 0012 temp: 134512 @ 0013

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

main()
x: 5 @ 0001
y: 9 @ 0002
swap_pointers()
x: 0001 @ 0011
y: 0002 @ 0012
temp: 5 @ 0013

Since we dereference x, we set temp to 5 – the number at address 0001 – instead of 0001 itself

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

Since we dereference both x and y, we edit the value at 0001 to be the value at 0002

main()	
x:	9 @ 0001
y:	9 @ 0002
swap_pointers()	
x:	0001 @ 0011
y:	0002 @ 0012
temp:	5 @ 0013

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

Similarly, when we dereference y and set it equal to temp, we set the value at 0002 to be equal to the temp variable (5)

main()	
x:	9 @ 0001
y:	5 @ 0002

swap_pointers()	
x:	0001 @ 0011
y:	0002 @ 0012
temp:	5 @ 0013

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}  
  
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

If we called another function here instead of returning, another new frame will be added.

Always we add a frame when a function is called, and delete that frame once we hit the “return” from that function

main()	
x: 9 @ 0001	
y: 5 @ 0002	

swap_pointers()	
x: 0001 @ 0011	
y: 0002 @ 0012	
temp: 5 @ 0013	

Inside the Stack

Pointer_fun.c (Now with pointers)

```
int main(int argc, char **argv) {  
    int x;  
    int y;  
    x = 5;  
    y = 9;  
    swap_pointers(&x, &y);  
    return 0;  
}
```

```
void swap_pointers(int *x, int *y) {  
    int temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
    return;  
}
```

main()
x: 9 @ 0001
y: 5 @ 0002

Again, since we hit a return, we delete the stack frame of the function we return from