CSc 352

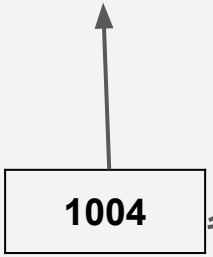
C Programming
Pointers

Benjamin Dicken

Pointers

A pointer is a numeric value representing the address of a location memory

```
char x = 50;
char * xp = &x;
```

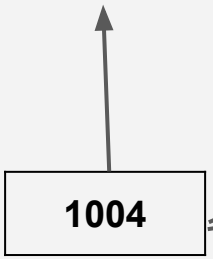


Memory Address	Sequential Memory
....	....
1001	0
1002	0
1003	0
1004	50
....	....
1008	1004
....	....
1016	0

Pointers

A pointer is a numeric value representing the address of a location memory

```
char x = 50;
char * xp = &x;
```



Memory Address	Sequential Memory
....	....
1001	105
1002	17
1003	32
1004	50
....	....
1008	1004
....	....
1016	10

Pointers

A pointer is a numeric value representing the address of a location memory

```
char x = 50;
char * xp = &x;
printf("%d\n", x);
printf("%p\n", &x);
printf("%p\n", xp);
```

Memory Address	Sequential Memory
....	....
1001	105
1002	17
1003	32
1004	50
....	....
1008	1004
....	....
1016	10

C Pointer

- An address to a location memory
- Access to the numeric value
- Can do *math* with that number
- For static / hard-coded values: compiler assigns pointer
- For dynamic values: malloc assigns values

Java / Py Reference

A reference is sorta like a pointer, however:

- No math on the pointer
- Less control over pointer address, dereference, etc.

Pointers

Addresses	Memory
. . . .	. . . .
0x100000000001	0
0x100000000002	0
. . . .	0
0x10000000000A	0
. . . .	0
0x100000000013	0
0x100000000014	0
. . . .	. . . .
. . . .	. . . .

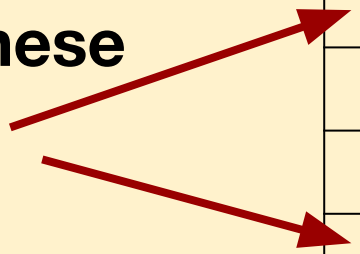
Activity

Pointers

How big are these numbers?

(8 bits? 16 bits? 32 bits? 48 bits? 64 bits?)

Why is it that size?

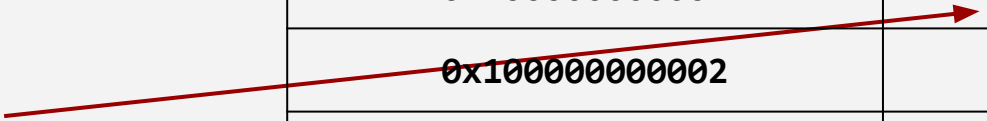


Addresses	Memory
. . . .	. . . .
0x100000000001	0
0x100000000002	0
. . . .	0
0x10000000000A	0
. . . .	0
0x100000000013	0
0x100000000014	0
. . . .	. . . .
. . . .	. . . .

https://en.wikipedia.org/wiki/X86-64#Canonical_form_addresses

Pointers

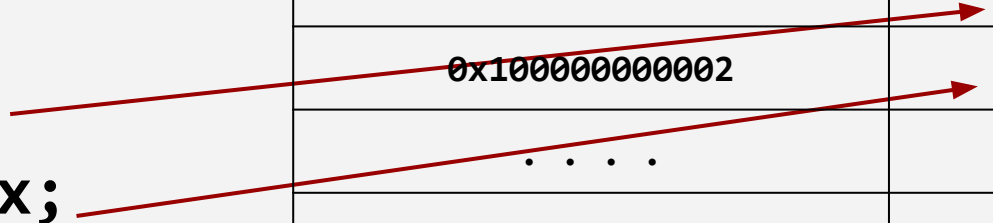
```
char x = 50;
```



Addresses	Memory
. . . .	. . . .
0x100000000001	50
0x100000000002	0
. . . .	0
0x10000000000A	0
. . . .	0
0x100000000013	0
0x100000000014	0
. . . .	. . . .
. . . .	. . . .

Pointers

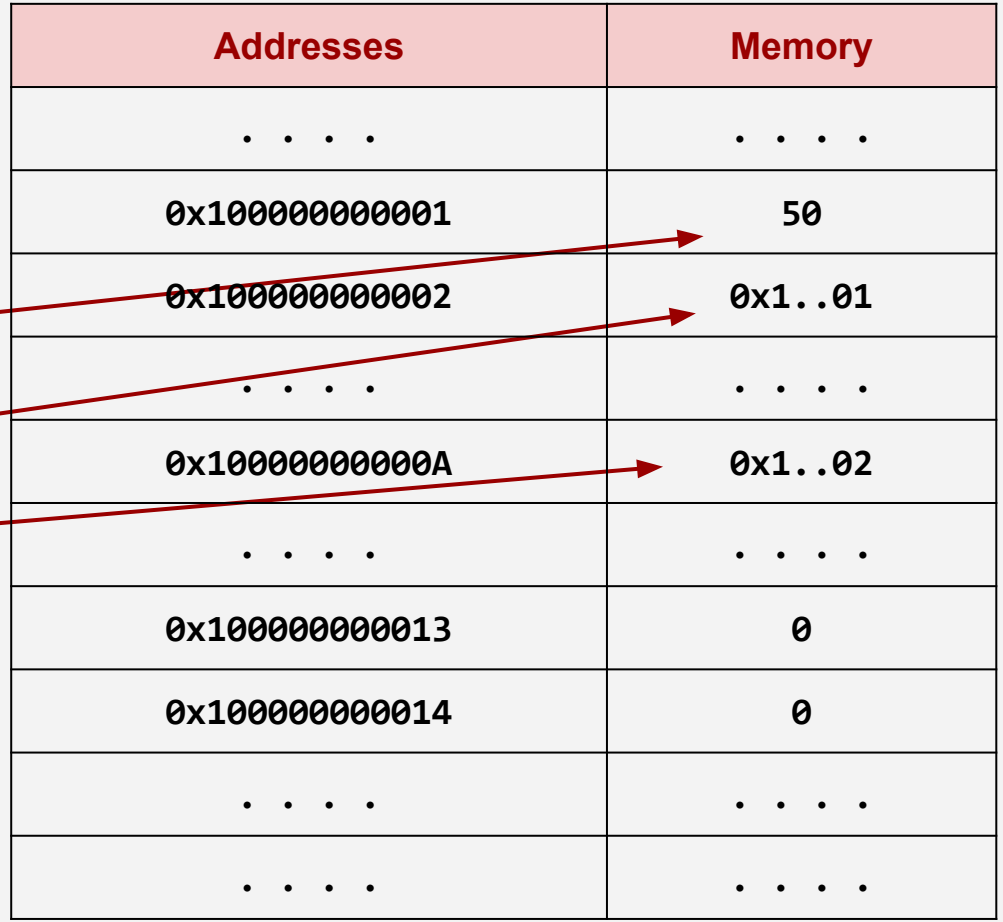
```
char x = 50;
char * xp = &x;
```



Addresses	Memory
. . . .	. . . .
0x100000000001	50
0x100000000002	0x1..01
. . . .	. . . .
0x10000000000A	0
. . . .	0
0x100000000013	0
0x100000000014	0
. . . .	. . . .
. . . .	. . . .

Pointers

```
char x = 50;
char * xp = &x;
char ** xpp = &xp;
```



Addresses	Memory
. . . .	. . . .
0x100000000001	50
0x100000000002	0x1..01
. . . .	. . . .
0x10000000000A	0x1..02
. . . .	. . . .
0x100000000013	0
0x100000000014	0
. . . .	. . . .
. . . .	. . . .

Pointer-related operators (unary, prefix)

***** dereferences a pointer (gives the values that the pointer points to)

If **x** is a pointer to an int, then ***x** is the int itself

One is the opposite of the other

& gets the address of a value

If **x** is an integer, **&x** is the address of that integer in memory

`*(&p)` is equivalent to **`p`**

What will print?

```c
#include <stdio.h>

int main() {
   int x = 50;
   int * z = &x;
   printf("%d\n", *z);
   return 0;
}
```

What will print?

```c
#include <stdio.h>

int main() {
   int x = 50;
   int ** z = &(&x);
   printf("%d\n", **z);
   printf("%d\n", x);
   return 0;
}
```

What will print?

```
int* something(
    int a, int * b) {
  int c = 40;
  a = 20;
  *b = 30;
  int * d = &c;
  return d;
}
```

```
int main() {
  int x = 100;
  int y = 200;
  int * z = something(y, &y);
  printf("%d\n", x);
  printf("%d\n", y);
  printf("%d\n", *z);
  return 0;
}
```

What will print?

```c
#include <stdio.h>
void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}
int * something() {
  int c = 100;
  int * d = &c;
  return d;
}
```

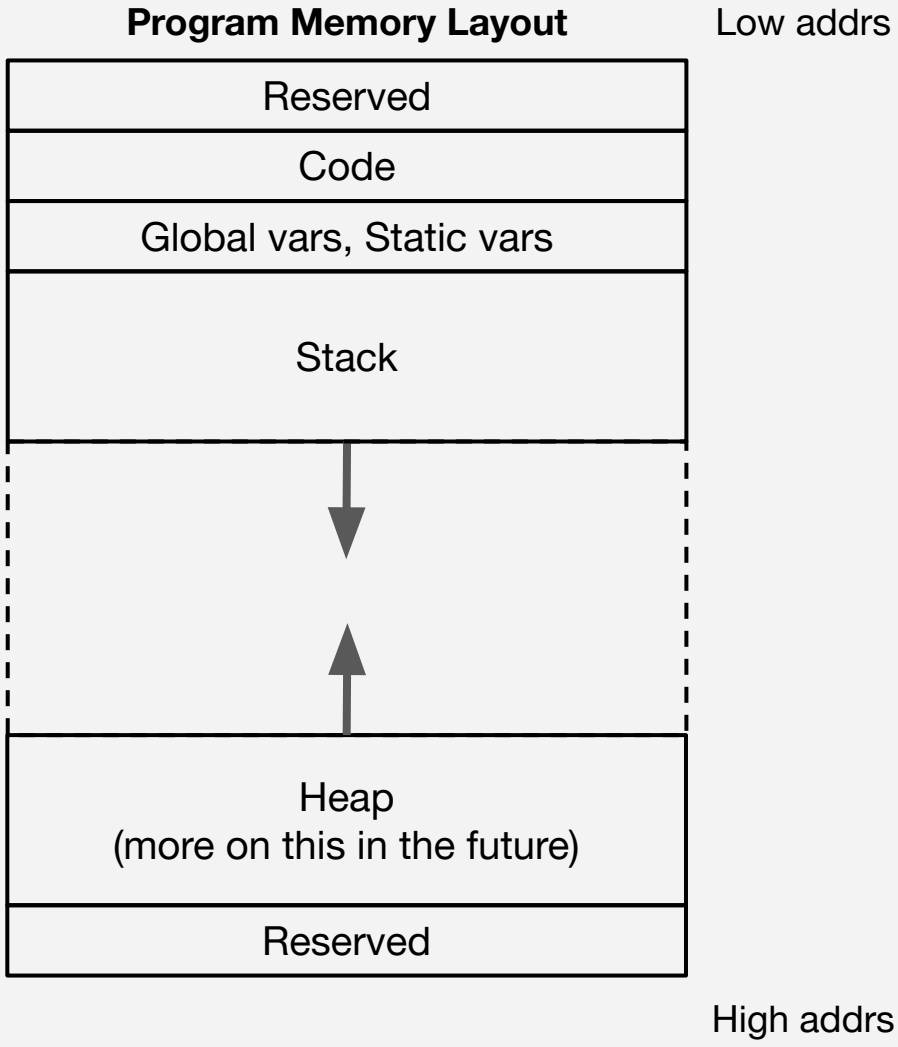
```c
int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

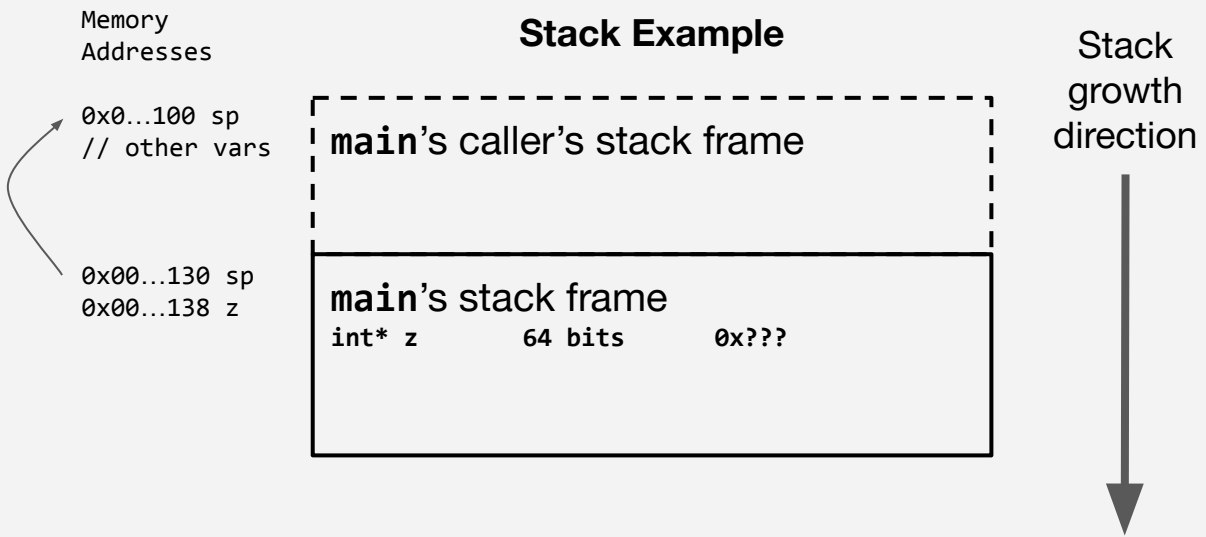


```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

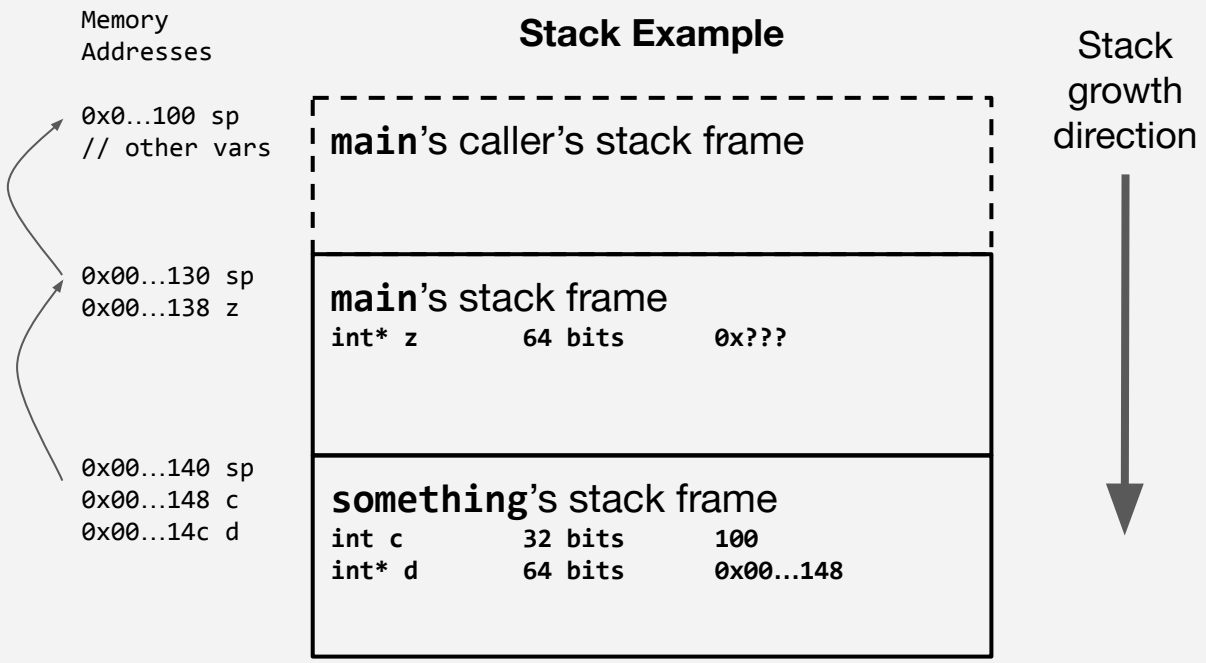


```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

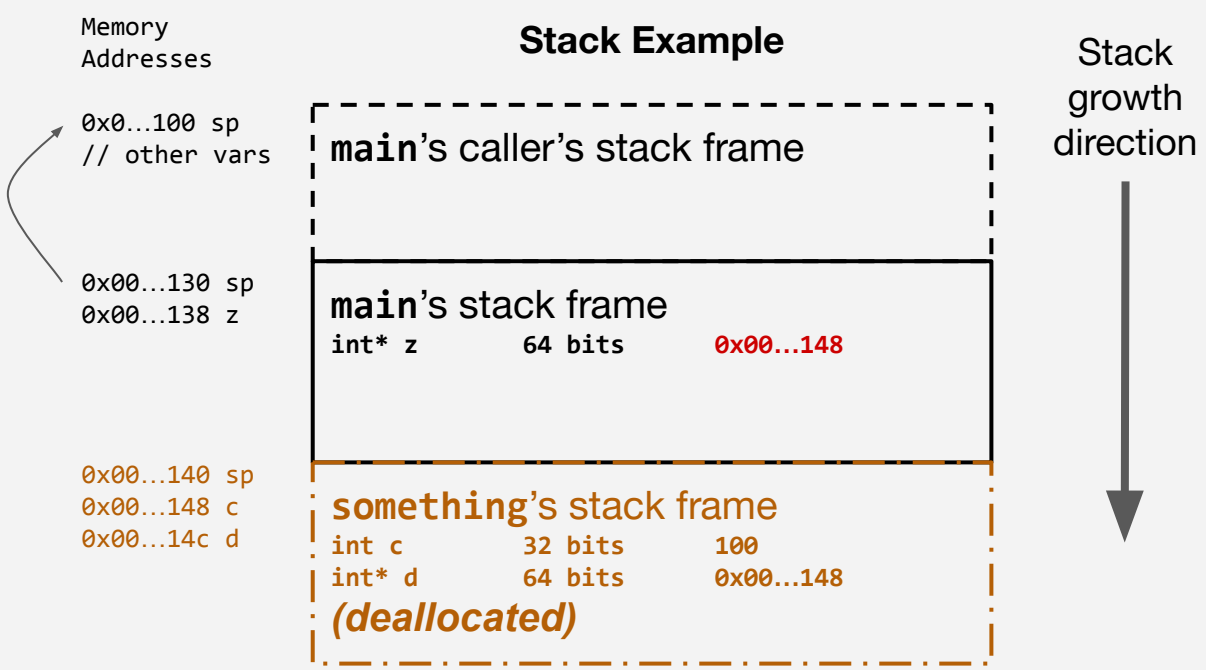


```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

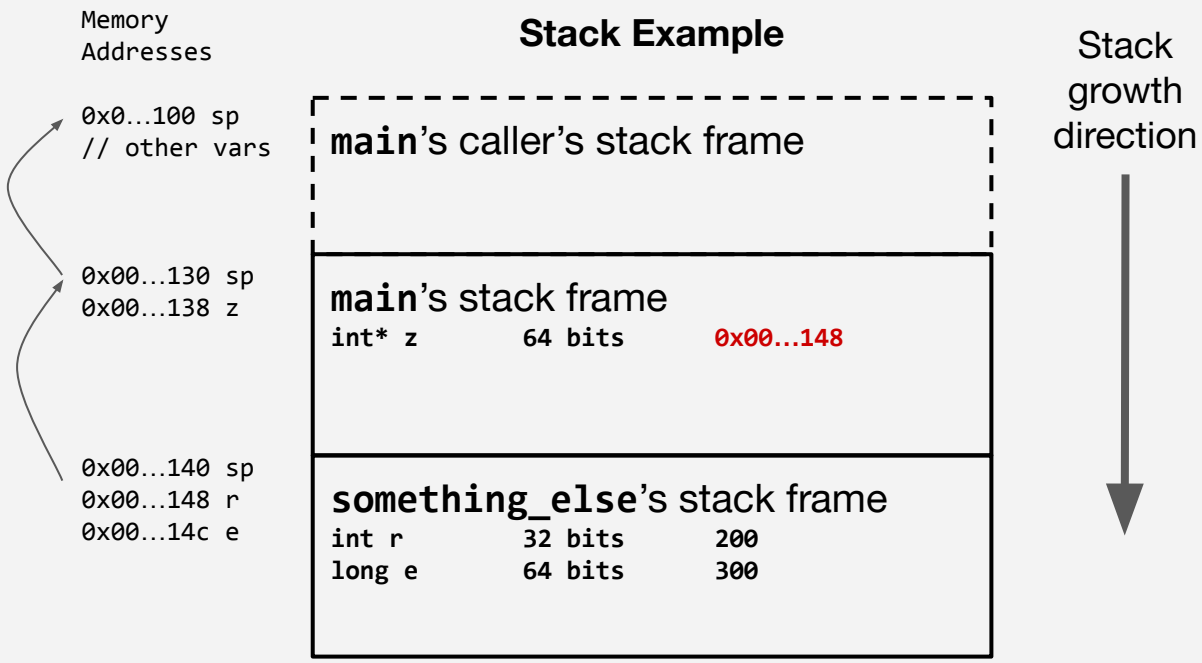


```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```

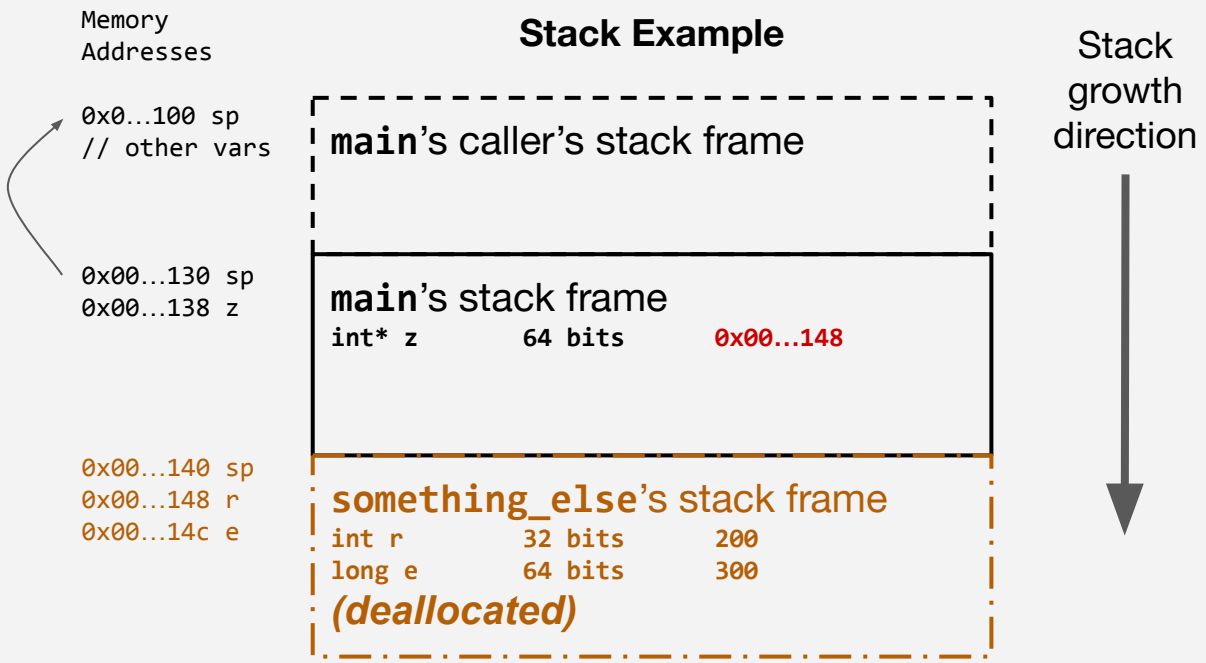


```c
#include <stdio.h>

void something_else() {
  int r = 200;
  long e = 300;
  printf("%ld, %ld\n", e, r);
}

int * something() {
  int c = 100;
  int * d = &c;
  return d;
}

int main() {
  int * z = something();
  something_else();
  printf("%d\n", *z);
  return 0;
}
```



```
char[] upper(char[] text) {
  for(int i = 0; text[i] != '\0'; i++) {
    if (text[i] >= 'a' && text[i] <= 'z') {
      text[i] = text[i] - 32;
    }
  }
  return text;
}

int main(int argc, char** argv) {
  char input[50];
  printf("Give me a word:\n");
  scanf("%s", input);
  printf("The word: %s\n", input);
  input = upper(input);
  printf("The word with upper case letters: %s\n", input);
  return 0;
}
```

What is wrong?

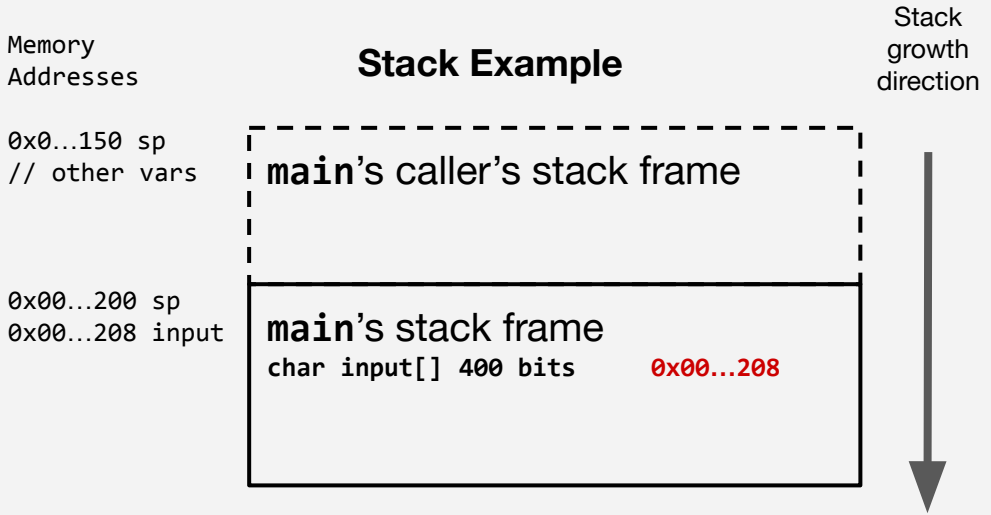
```c
void upper(char* text) {
  for(int i = 0; text[i] != '\0'; i++) {
    if (text[i] >= 'a' && text[i] <= 'z') {
      text[i] = text[i] - 32;
    }
  }
}

int main(int argc, char** argv) {
  char input[50];
  printf("Give me a word:\n");
  scanf("%s", input);
  printf("The word: %s\n", input);
  upper(input);
  printf("The word with upper case letters: %s\n", input);
  return 0;
}
```

What about this instead?

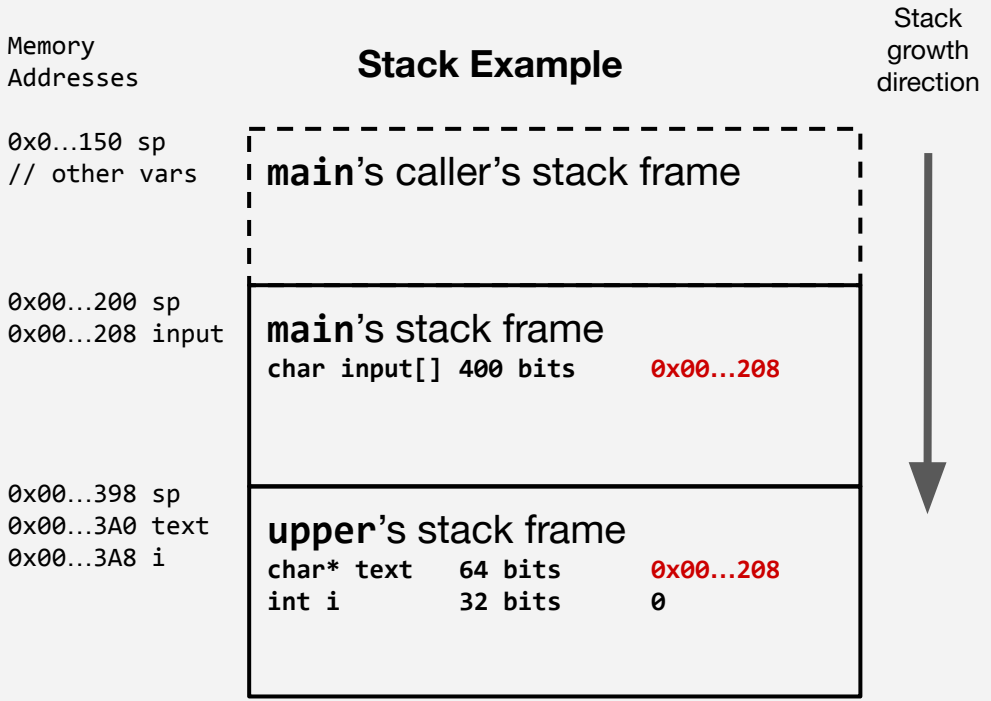
```c
void upper(char* text) {
  for(int i = 0; text[i] != '\0'; i++) {
    if (text[i] >= 'a' && text[i] <= 'z') {
      text[i] = text[i] - 32;
    }
  }
}

int main(int argc, char** argv) {
  char input[50];
  printf("Give me a word:\n");
  scanf("%s", input);
  printf("The word: %s\n", input);
  upper(input);
  printf("The word with upper case letters: %s\n", input);
  return 0;
}
```



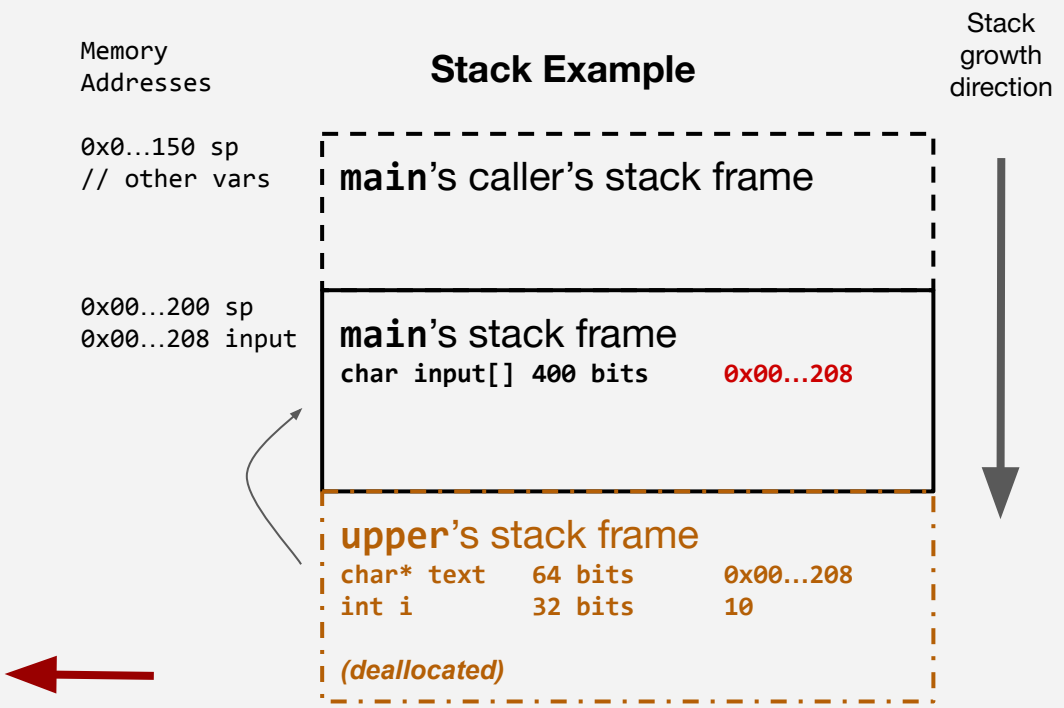
```c
void upper(char* text) {
  for(int i = 0; text[i] != '\0'; i++) {
    if (text[i] >= 'a' && text[i] <= 'z') {
      text[i] = text[i] - 32;
    }
  }
}

int main(int argc, char** argv) {
  char input[50];
  printf("Give me a word:\n");
  scanf("%s", input);
  printf("The word: %s\n", input);
  upper(input);
  printf("The word with upper case letters: %s\n", input);
  return 0;
}
```



```c
void upper(char* text) {
  for(int i = 0; text[i] != '\0'; i++) {
    if (text[i] >= 'a' && text[i] <= 'z') {
      text[i] = text[i] - 32;
    }
  }
}

int main(int argc, char** argv) {
  char input[50];
  printf("Give me a word:\n");
  scanf("%s", input);
  printf("The word: %s\n", input);
  upper(input);
  printf("The word with upper case letters: %s\n", input);
  return 0;
}
```



Uninitialized and Dangling Pointers

Uninitialized Pointer: A pointer that does not get assigned a value

- What happens when you look up a “random” address?

Dangling Pointer: Points to a location that is no longer valid

- Think: Points to a value that *was* on the stack but has been deallocated
- Think: Points to dynamically-allocated memory that has been freed

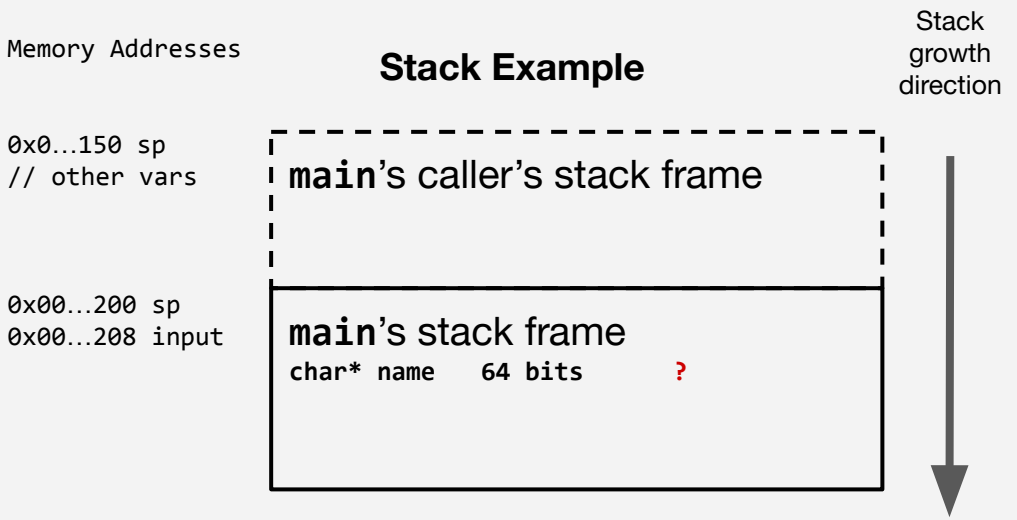
What do you think of this code?

```
char * get_name(char* prompt) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  return buffer;
}

int main() {
  char* name = get_name("Enter your name:\n");
  printf("Your name is %s\n", name);
  return 0;
}
```

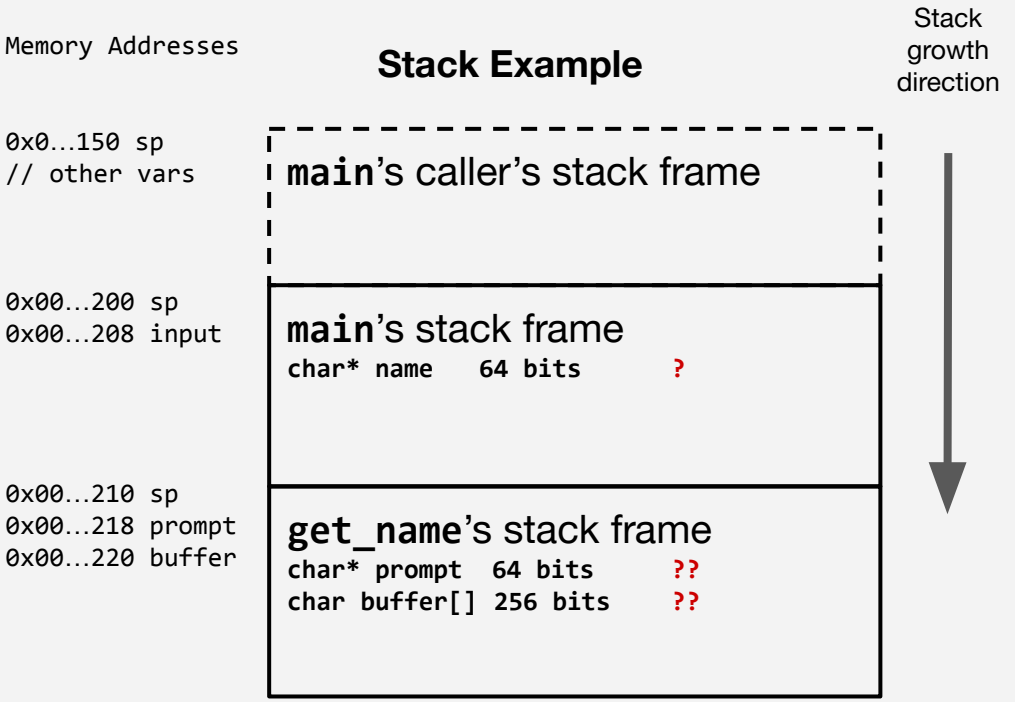
```c
char * get_name(char* prompt) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  return buffer;
}

int main() {
  char* name = get_name("Enter your name:\n");
  printf("Your name is %s\n", name);
  return 0;
}
```



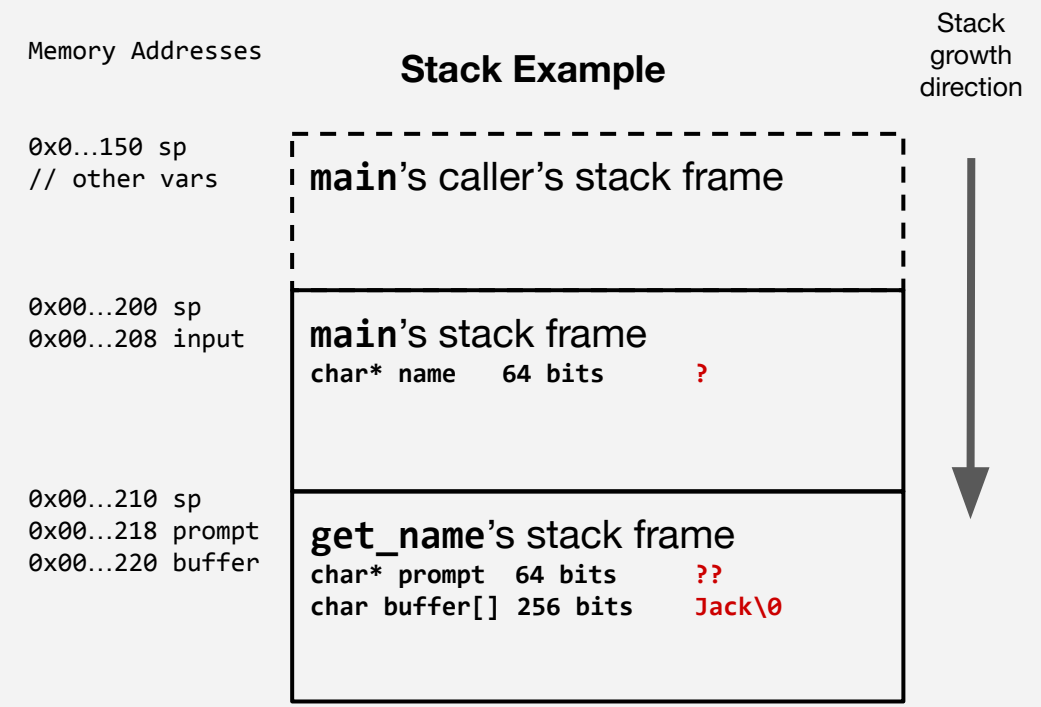
```c
char * get_name(char* prompt) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  return buffer;
}

int main() {
  char* name = get_name("Enter your name:\n");
  printf("Your name is %s\n", name);
  return 0;
}
```



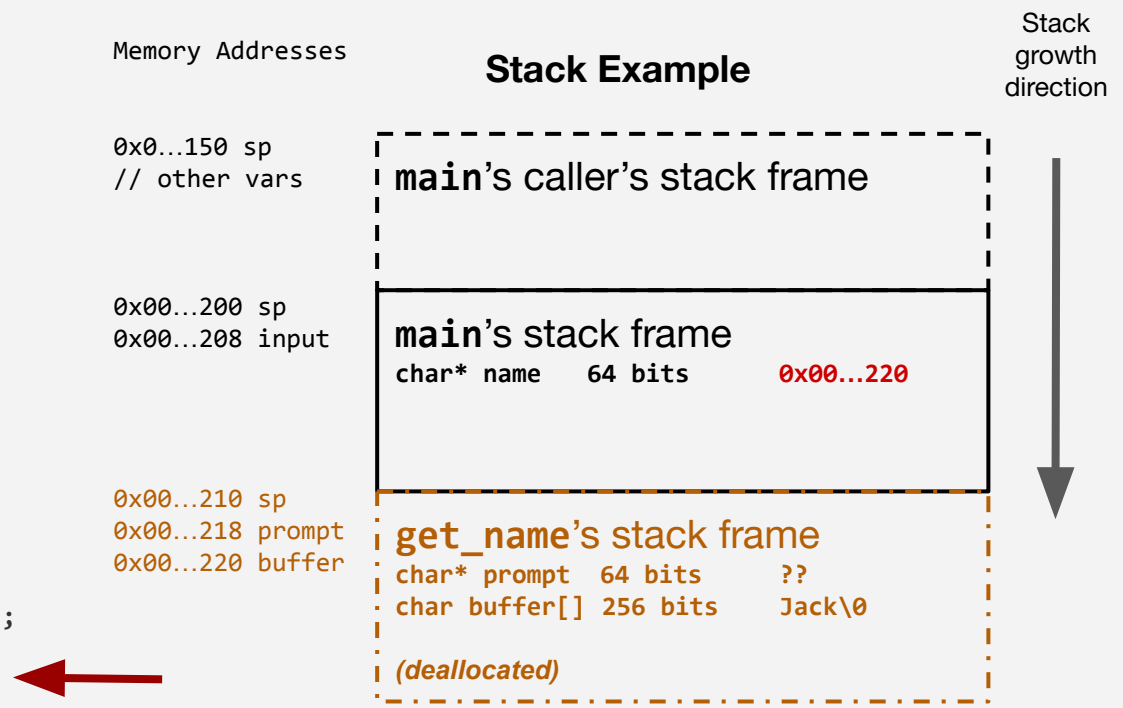
```c
char * get_name(char* prompt) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  return buffer;
}

int main() {
  char* name = get_name("Enter your name:\n");
  printf("Your name is %s\n", name);
  return 0;
}
```



```c
char * get_name(char* prompt) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  return buffer;
}

int main() {
  char* name = get_name("Enter your name:\n");
  printf("Your name is %s\n", name);
  return 0;
}
```



What do you think of this code?

```c
void get_name(char* prompt, char** name) {
  char buffer[32];
  printf("%s", prompt);
  scanf("%31s", buffer);
  *name = buffer;
}

int main() {
  char* name;
  get_name("Enter your name:\n", &name);
  printf("Your name is %s\n", name);
  return 0;
}
```

What do you think of this code?

```c
void get_name(char* prompt, char* name) {
  printf("%s", prompt);
  scanf("%31s", name);
}

int main() {
  char name[32];
  get_name("Enter your name:\n", name);
  printf("Your name is %s\n", name);
  return 0;
}
```

What do you think of this code?

```
int* get_id_number(char* prompt) {
  int id_number;
  int* id_number_address = &id_number;
  printf("%s", prompt);
  scanf("%d", &id_number);
  return id_number_address;
}

int main() {
  int* id = get_id_number("Enter your ID number:\n");
  printf("Your ID number is %d\n", *id);
  return 0;
}
```

L-values and R-values

- An L-value is a ***location*** and can be used on the left-hand side of an equals sign
  - Arithmetic-type variables, array elements, pointers
  - Also structs (later)
- An R-value is something that does not actually have a stored location in memory
  - Return value from function, a math expression, etc

Plus Plus

x++ yields x and increments x sometime before or at the completion of the statement it is within.

++x yields (x+1) and increments x sometime before or at the completion of the statement it is within.

```
int x = 1;
int y = 2;
int r1 = 0;
int r2 = 0;
```

What is valid and what is not valid?

```
r1 = x++;                    // A
r2 = (x++)++;                // B
x + y  = x + y;              // C
*(&x) = (++y) + (r1++);      // D
*(&x) = (++y) + (x++);       // E
*(&x + y) = 10;              // F
x++ = y++;                   // G
```

```
int x = 1;
int y = 2;
int* xp = &x;
int* yp = &y;

*(++xp) = 30;  // A
y = (*xp)++;   // B
y = *(xp++);   // C
y = *((&x)++); // D
```

What is valid and what is not valid?