Memory Management (Part 1)

Devin J. Pohly <djpohly@cse.psu.edu>

Memory management

- Most of the data in a nontrivial system/program will not be stored in static variables or on the stack
  - Allocated manually by the program
  - Customizable data lifetime
- Learning to allocate, manage, and deallocate memory is key in becoming a good systems programmer.

Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

Box and arrow diagrams

boxarrow.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

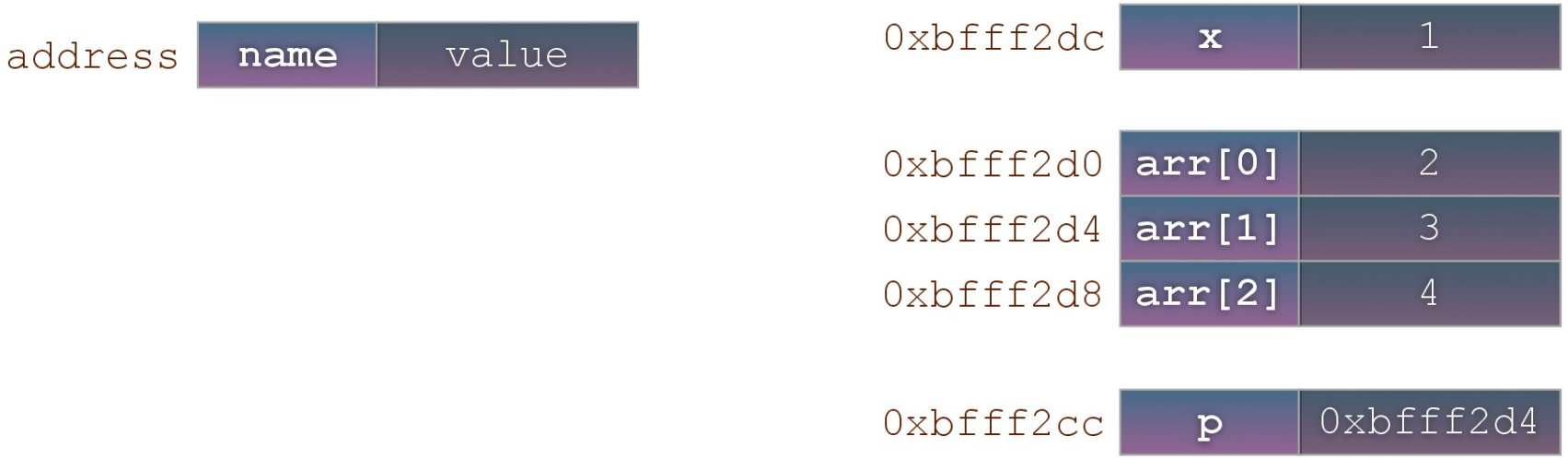
address	name	value
&x	x	1
&arr[0]	arr[0]	2
&arr[1]	arr[1]	3
&arr[2]	arr[2]	4
&p	p	&arr[1]

Box and arrow diagrams

boxarrow.c

```c
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```



Box and arrow diagrams

boxarrow.c

```c
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

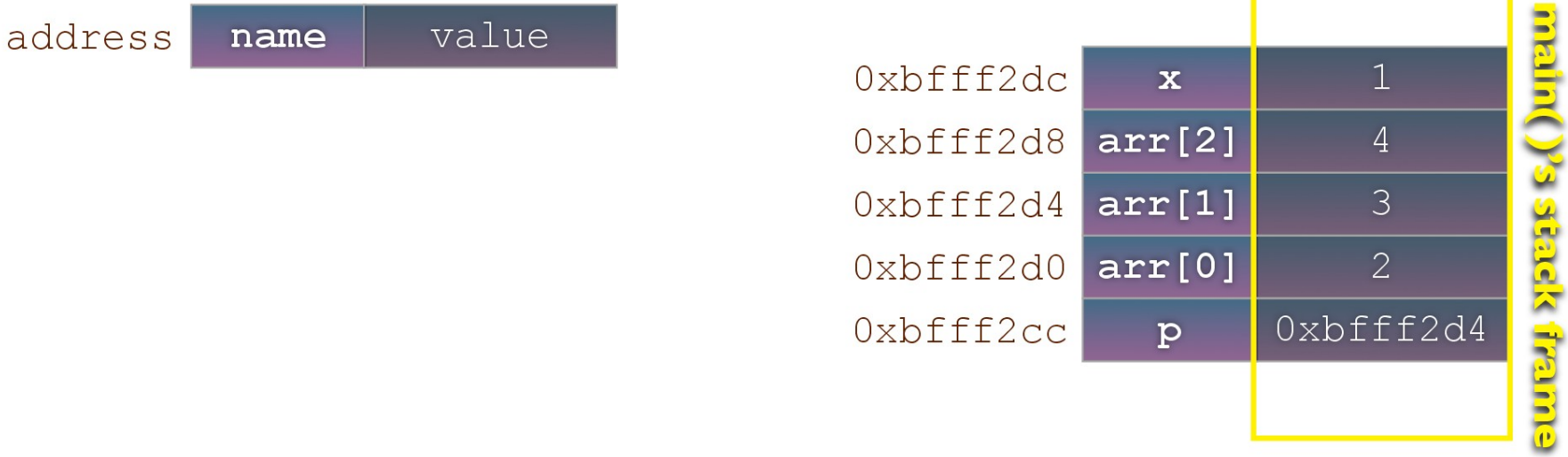
address	name	value
0xbfff2dc	x	1
0xbfff2d8	arr[2]	4
0xbfff2d4	arr[1]	3
0xbfff2d0	arr[0]	2
0xbfff2cc	p	0xbfff2d4

Box and arrow diagrams

boxarrow.c

```c
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

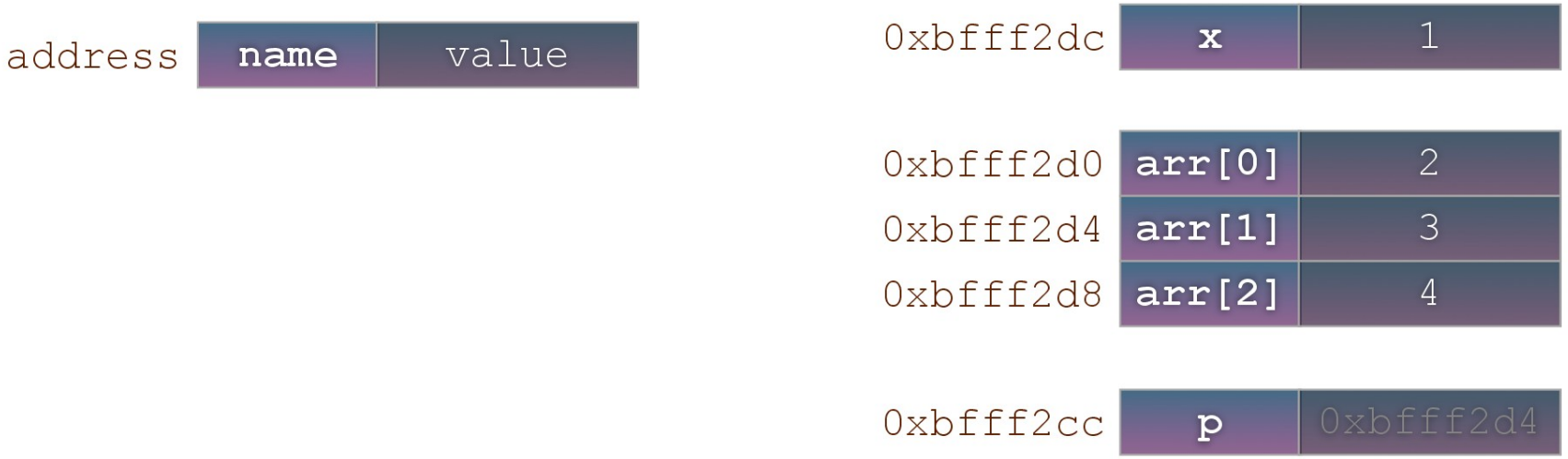


Box and arrow diagrams

boxarrow.c

```c
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

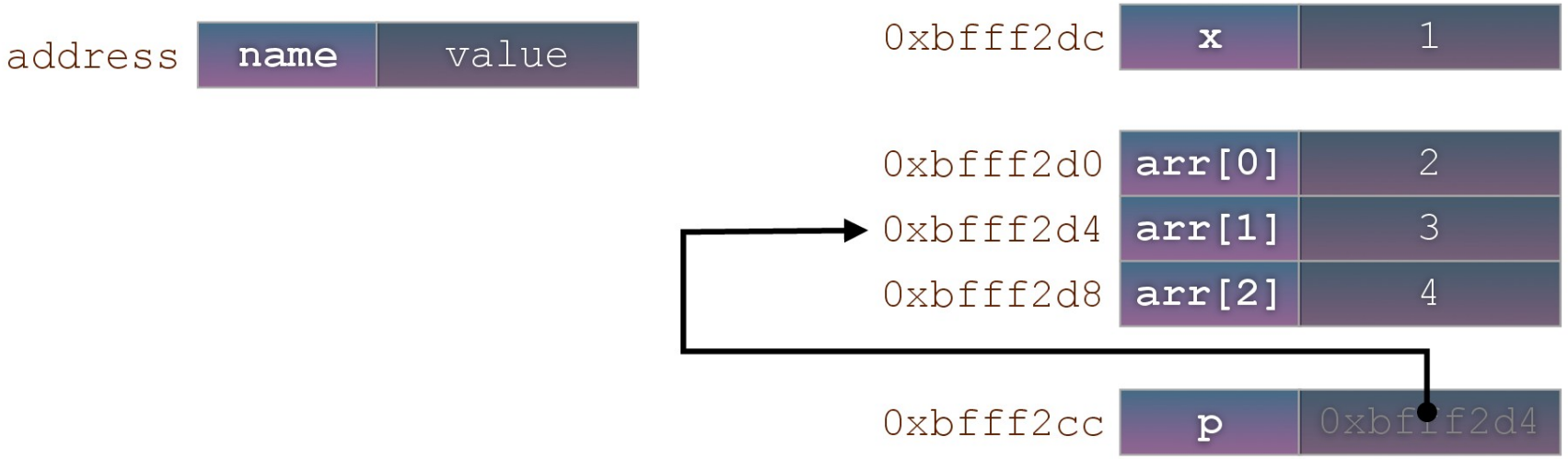


Box and arrow diagrams

boxarrow.c

```c
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p; arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p; arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  return 0;
}
```

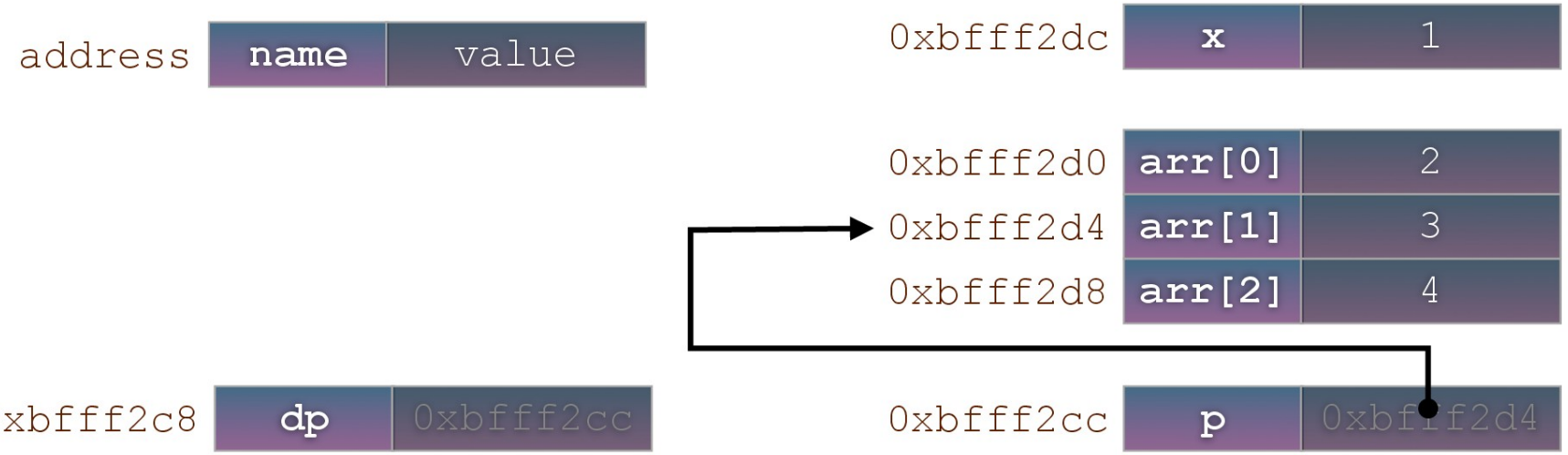


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

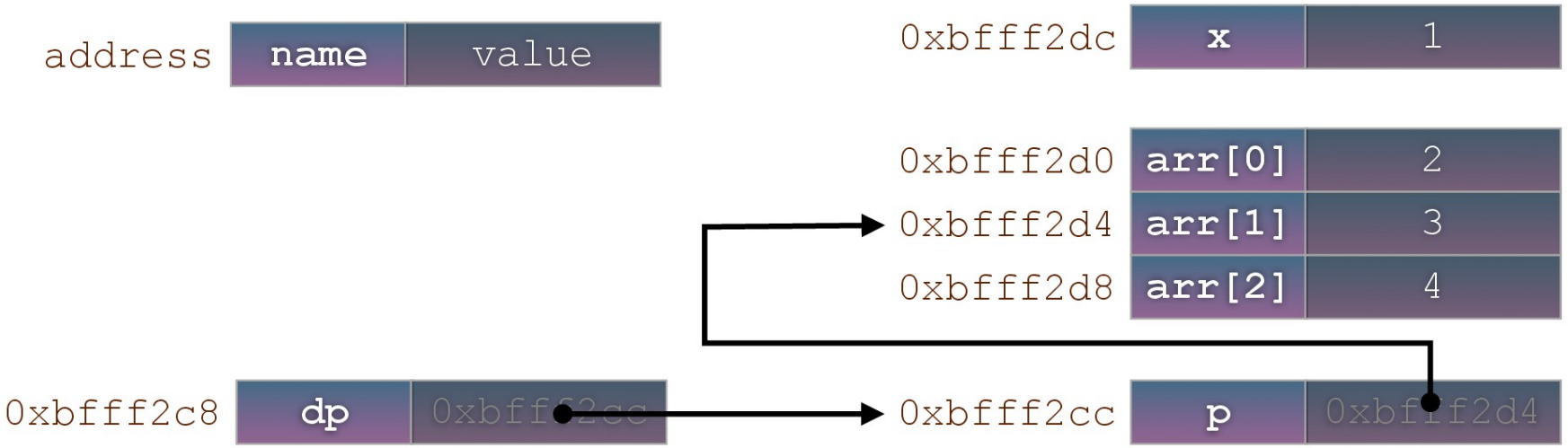


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

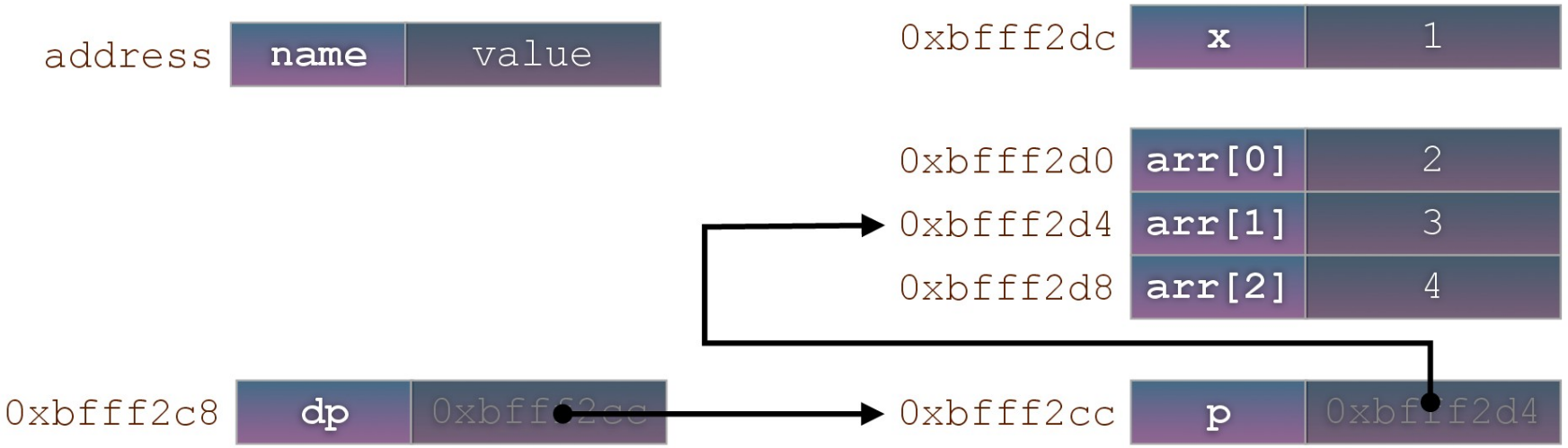


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

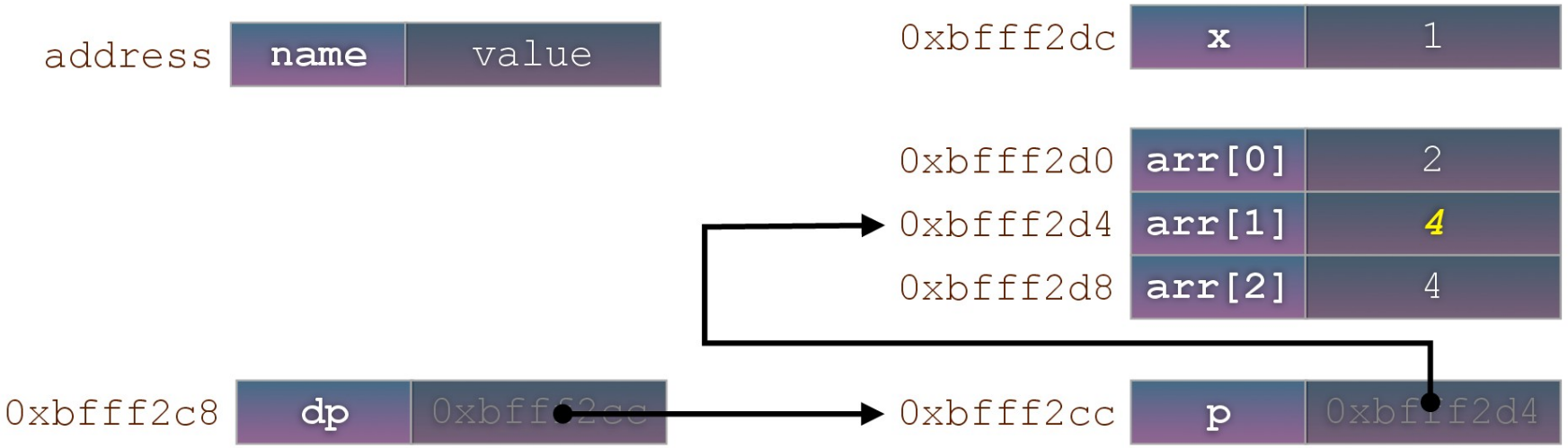


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

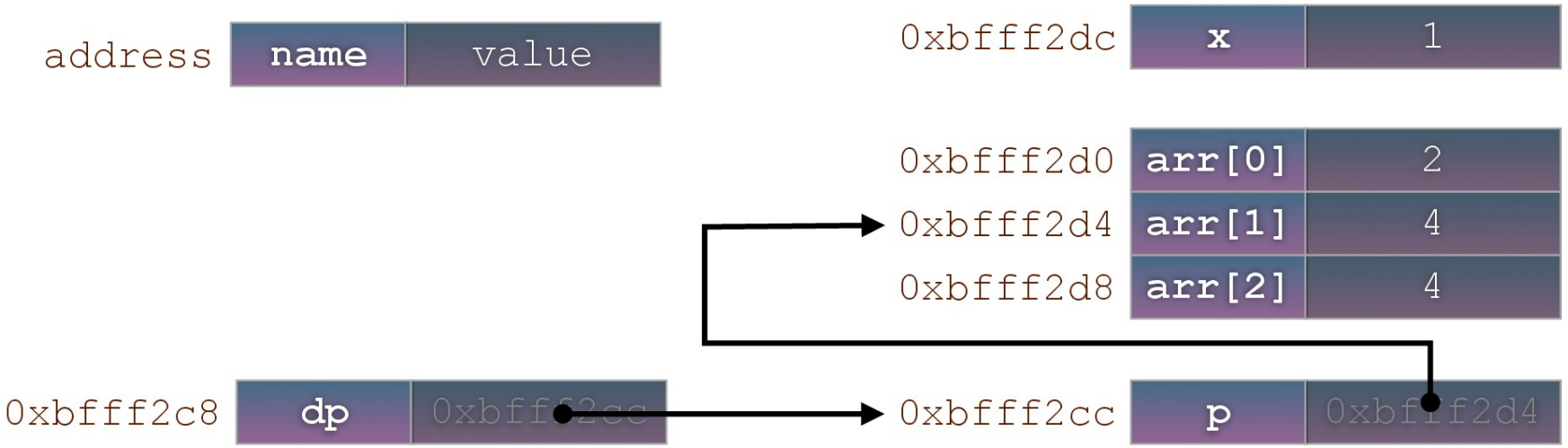


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

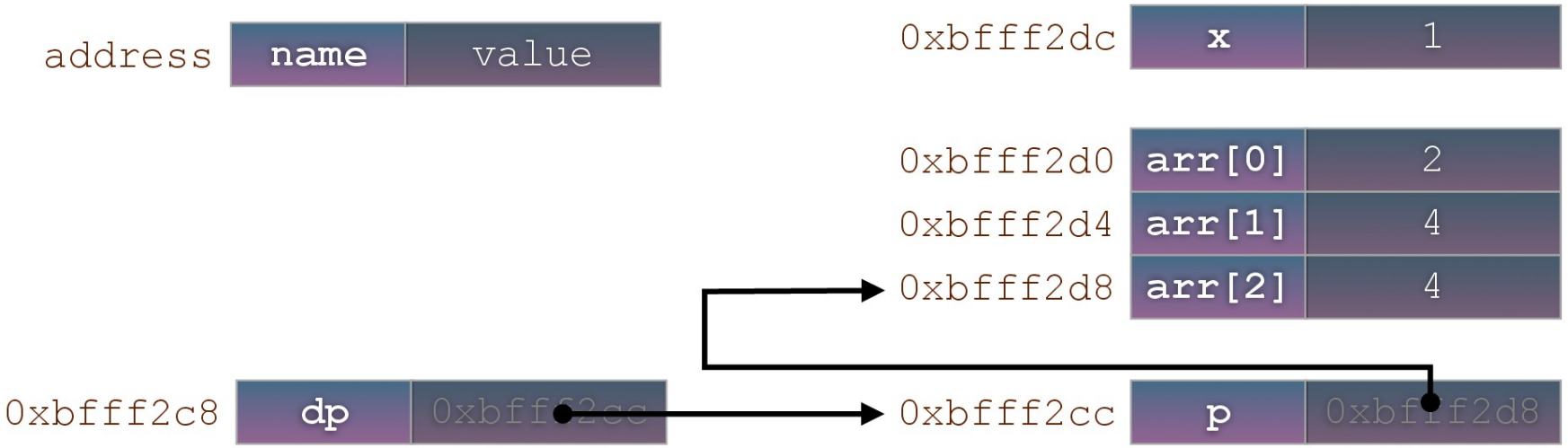


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

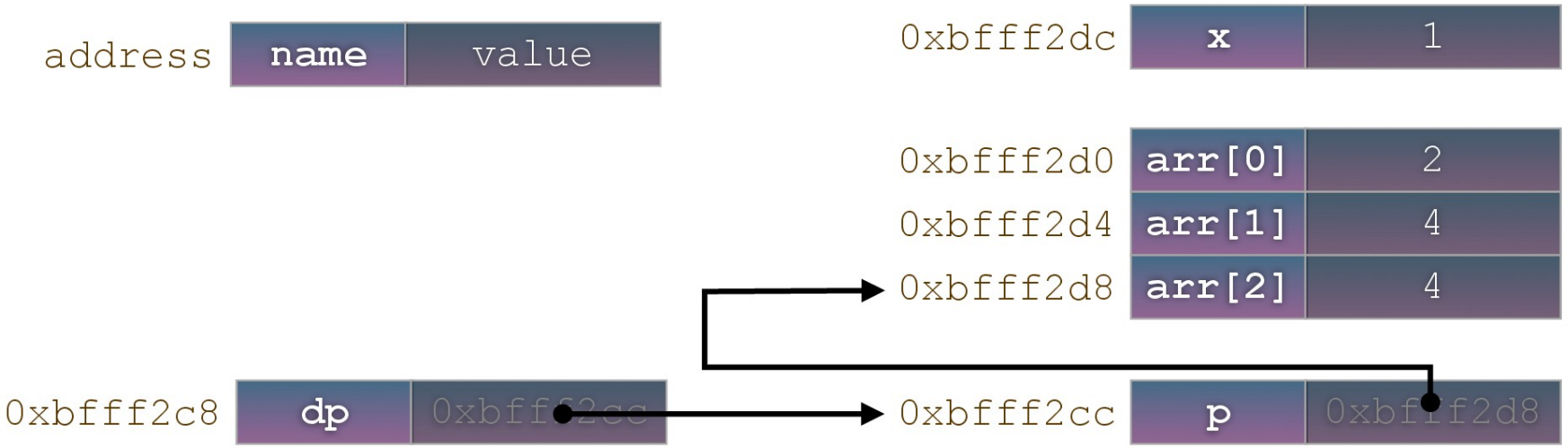


Box and arrow diagrams

boxarrow2.c

```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```

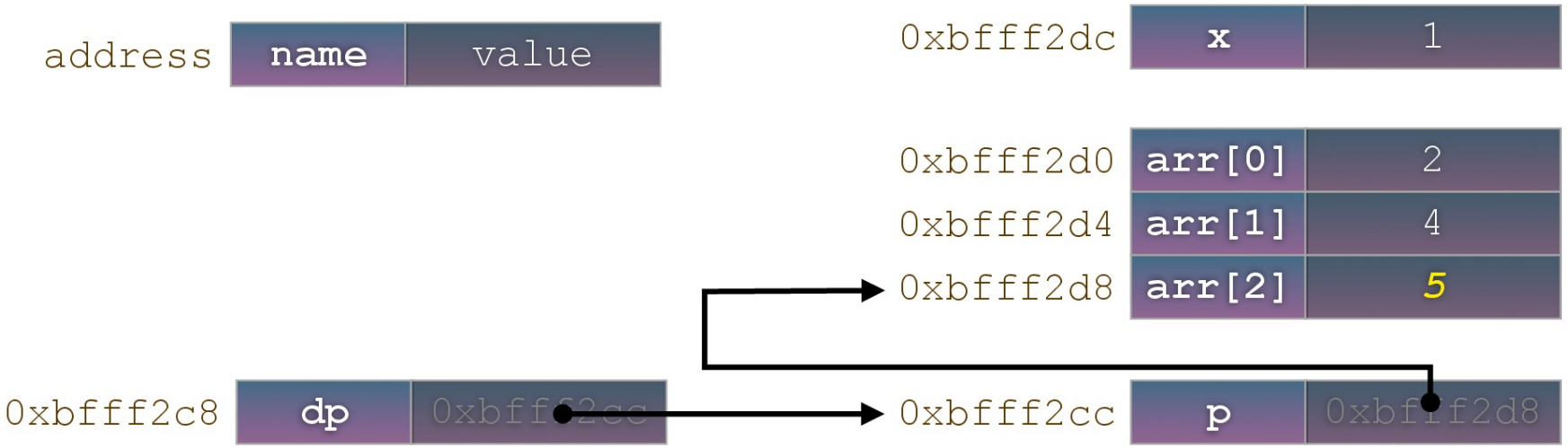


Box and arrow diagrams

boxarrow2.c

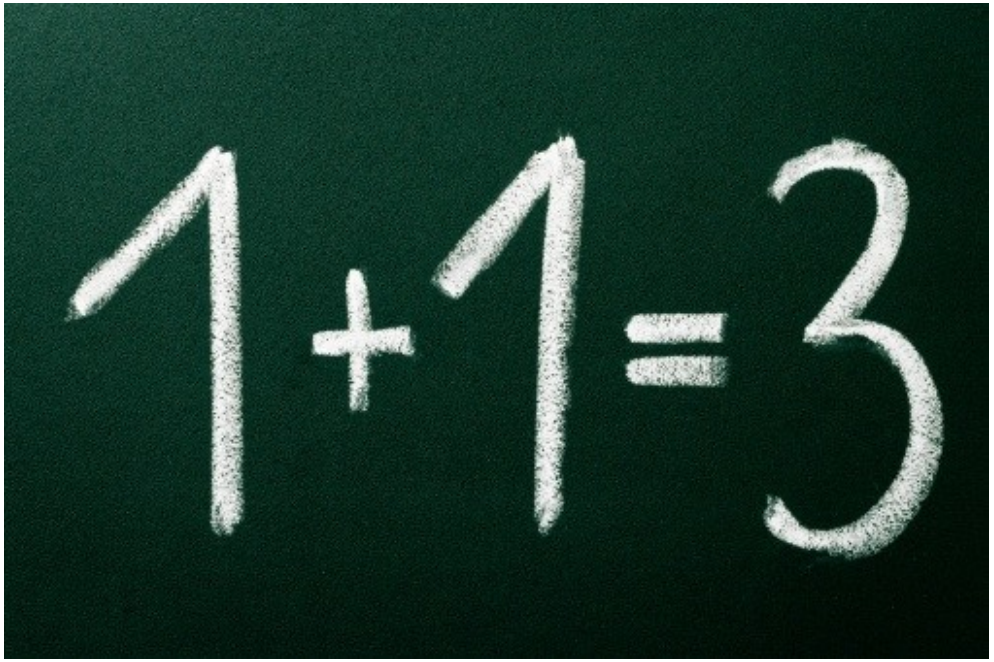
```
int main(int argc, char **argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];
  int **dp = &p;

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;
  return 0;
}
```



Pointer arithmetic redux

- Pointers are typed
  - `int *pi;`
  - `char *pch;`
  - Pointer arithmetic obeys these types
    - Adding 1 to a pointer adds the `sizeof` that type
    - Just like iterating over an array
- Exception: `void *`



Pointer arithmetic example

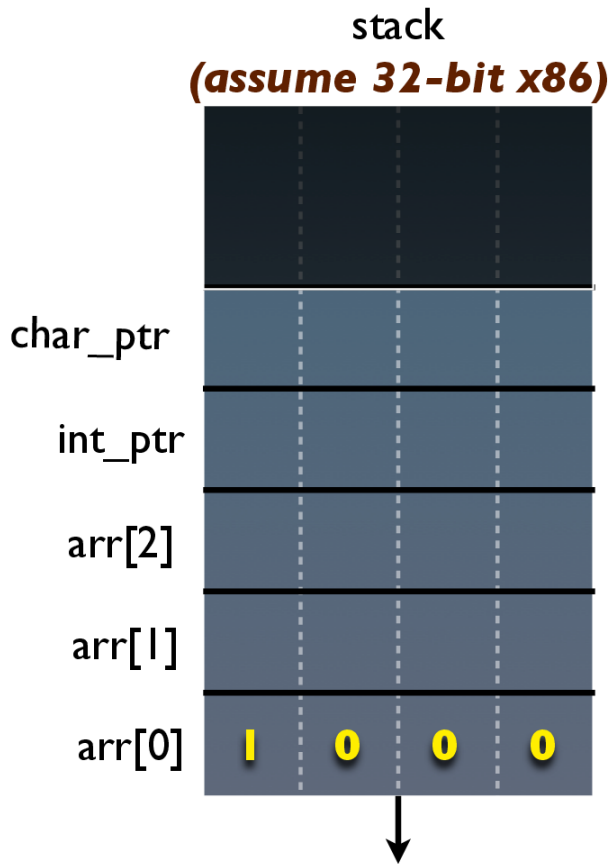
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```



(x86 is little endian)

pointerarithmetic.c

Pointer arithmetic example

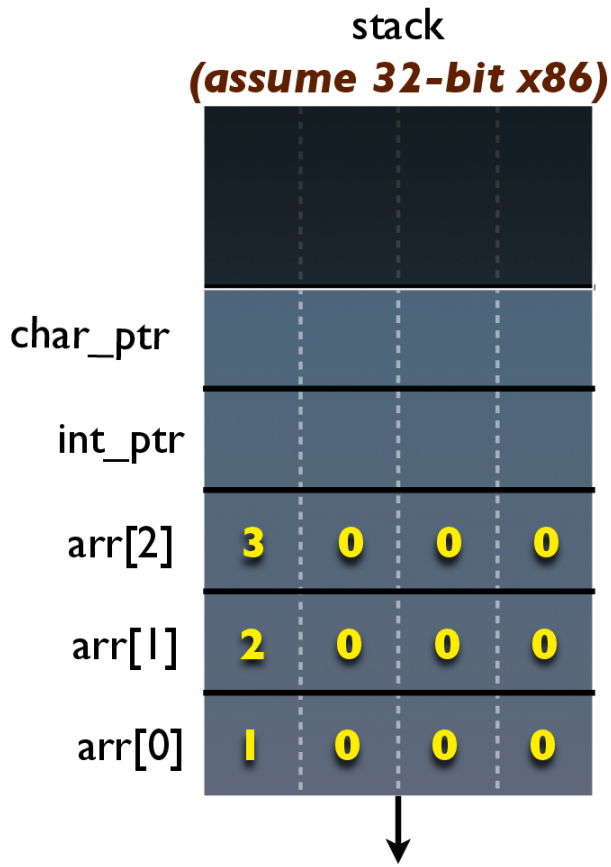
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```



pointerarithmetic.c

Pointer arithmetic example

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```

pointerarithmetic.c

Pointer arithmetic example

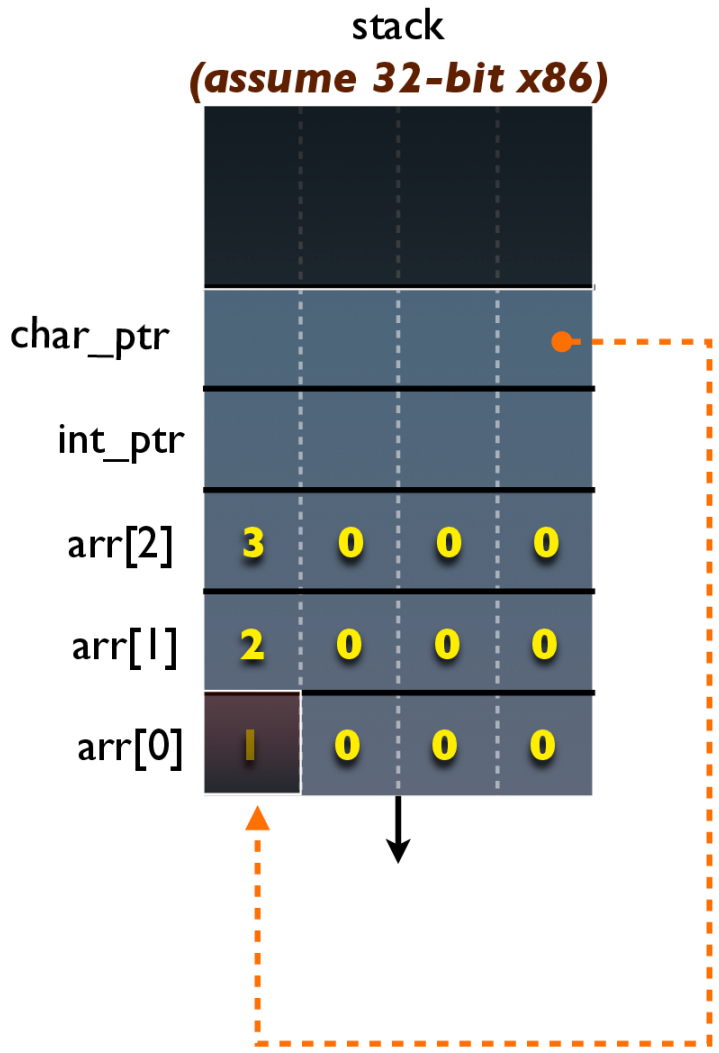
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```



pointerarithmetic.c

Pointer arithmetic example

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```


```
int_ptr: 0xbffff2ac;  *int_ptr: 1
```

pointerarithmetic.c

Pointer arithmetic example

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```


```
int_ptr: 0xbffff2ac;  *int_ptr: 1
```

pointerarithmetic.c

Pointer arithmetic example

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```


```
int_ptr: 0xbffff2ac;  *int_ptr: 1
int_ptr: 0xbffff2b0;  *int_ptr: 2
```

pointerarithmetic.c

Pointer arithmetic example

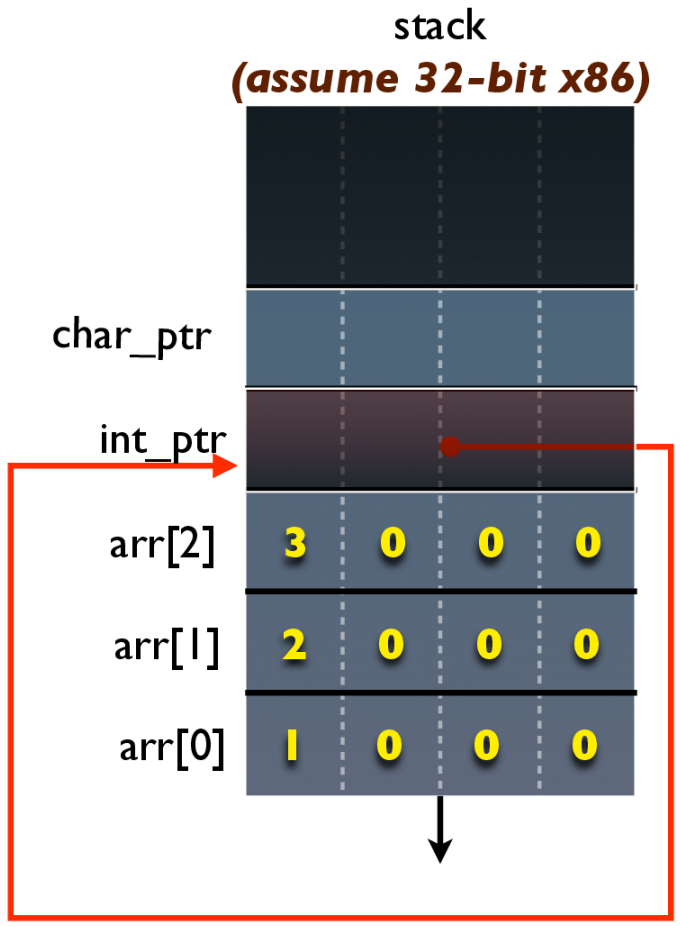
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```




```
int_ptr: 0xbffff2ac;  *int_ptr: 1
int_ptr: 0xbffff2b0;  *int_ptr: 2
```

pointerarithmetic.c

Pointer arithmetic example

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```


```
int_ptr: 0xbffff2ac;  *int_ptr: 1
int_ptr: 0xbffff2b0;  *int_ptr: 2
```

pointerarithmetic.c

Pointer arithmetic example

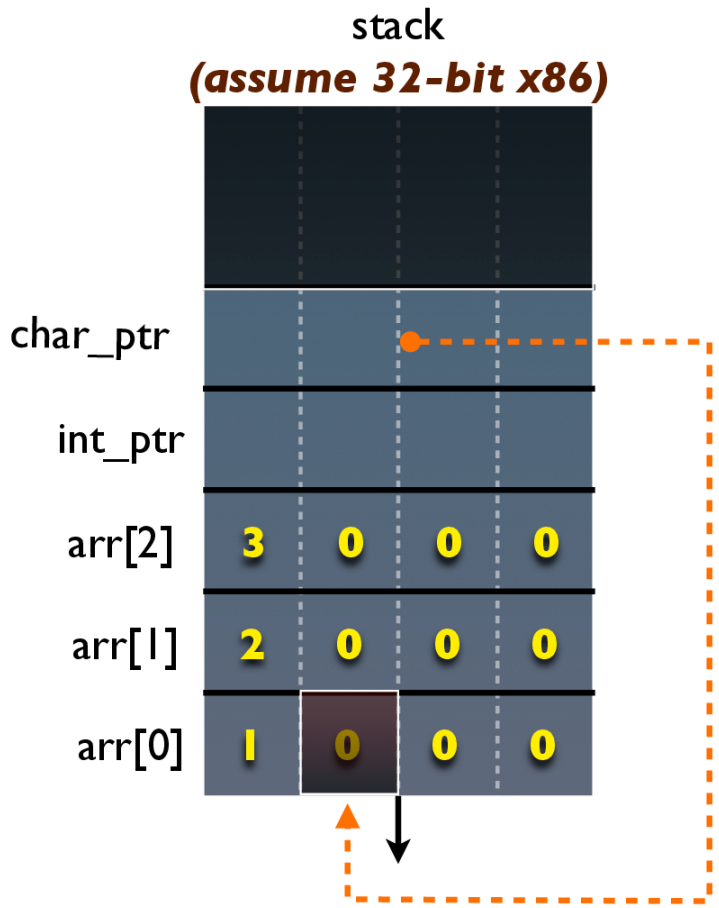
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```




```
char_ptr: 0xbffff2ac;  *char_ptr: 1
```

pointerarithmetic.c

Pointer arithmetic example

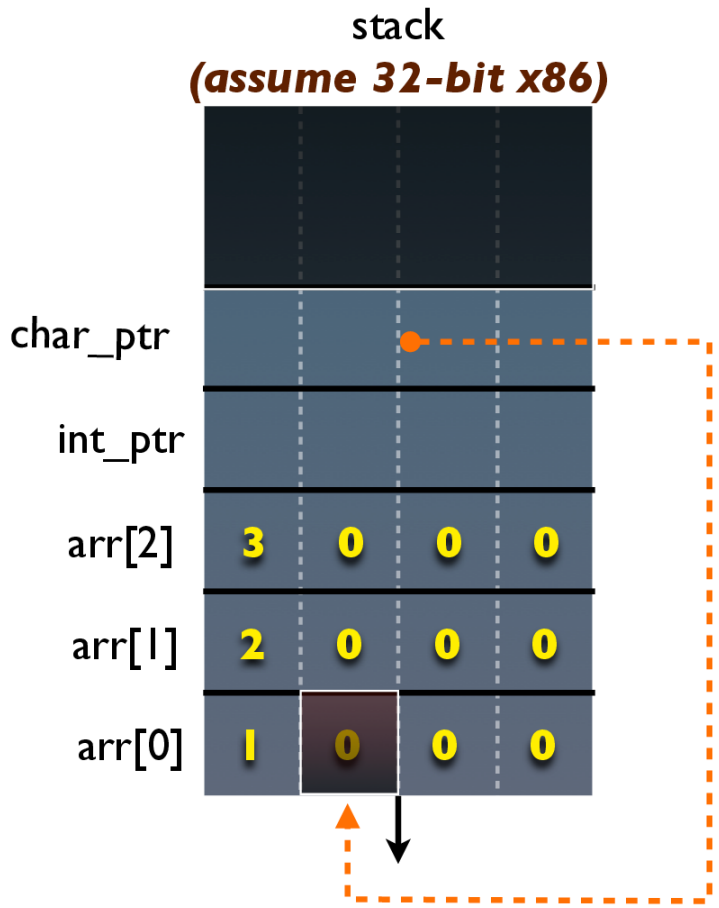
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```




```
char_ptr: 0xbffff2ac;  *char_ptr: 1
```

pointerarithmetic.c

Pointer arithmetic example

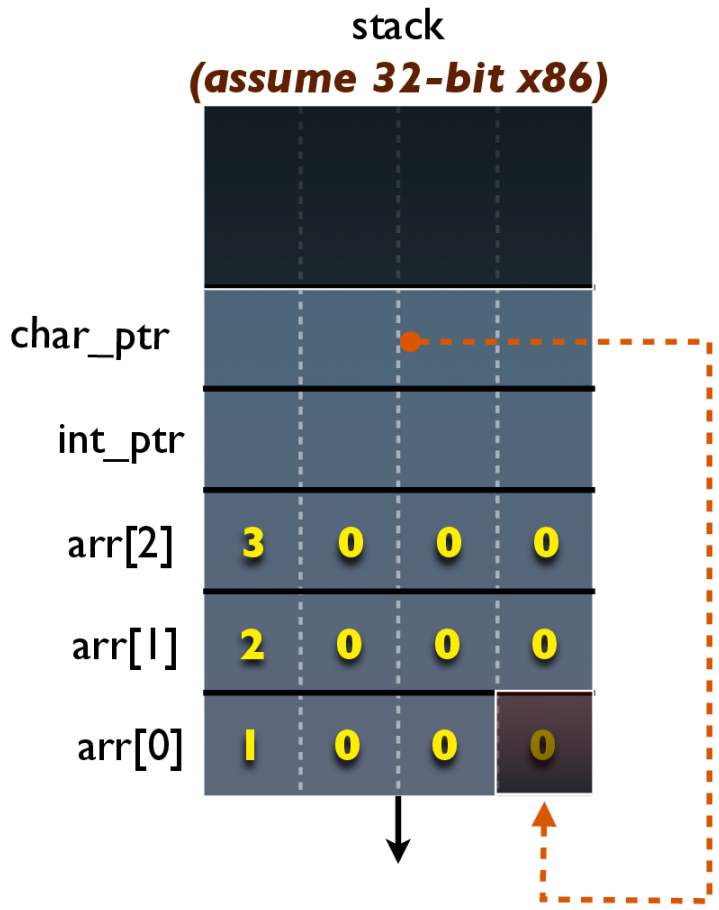
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```




```
char_ptr: 0xbffff2ac;  *char_ptr: 1
char_ptr: 0xbffff2ad;  *char_ptr: 0
```

pointerarithmetic.c

Pointer arithmetic example

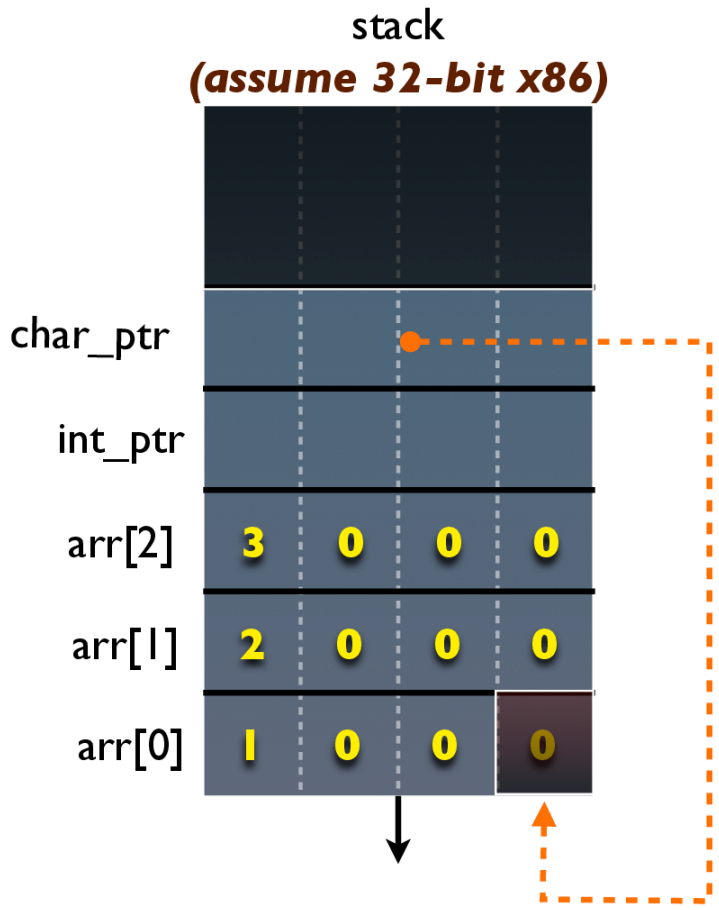
```c
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;

  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n",
         int_ptr, *int_ptr);
  int_ptr += 2;  // uh oh
  print("int_ptr: %p;   *int_ptr: %d\n",
        int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n",
         char_ptr, *char_ptr);

  return 0;
}
```




```
char_ptr: 0xbffff2ac;  *char_ptr: 1
char_ptr: 0xbffff2ad;  *char_ptr: 0
```

pointerarithmetic.c

Pointer to a pointer

- Question: what's the difference between a `char *` and a `char **`?
- Exercise: draw and update the box-and-arrow diagram for this program as it executes

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    char hi[] = "hello";
    char *p, **p2p;

    p = hi;   // What does this mean?
    p2p = &p;

    printf("%c %c\n", *p, **p2p);
    printf("%p %p %p\n", p, *p2p, hi);
    p++;
    printf("%c %c\n", *p, **p2p);
    printf("%p %p %p\n", p, *p2p, hi);
    *dp += 2;
    printf("%c %c\n", *p, **p2p);
    printf("%p %p %p\n", p, *p2p, hi);
    return 0;
}
```

Buffers

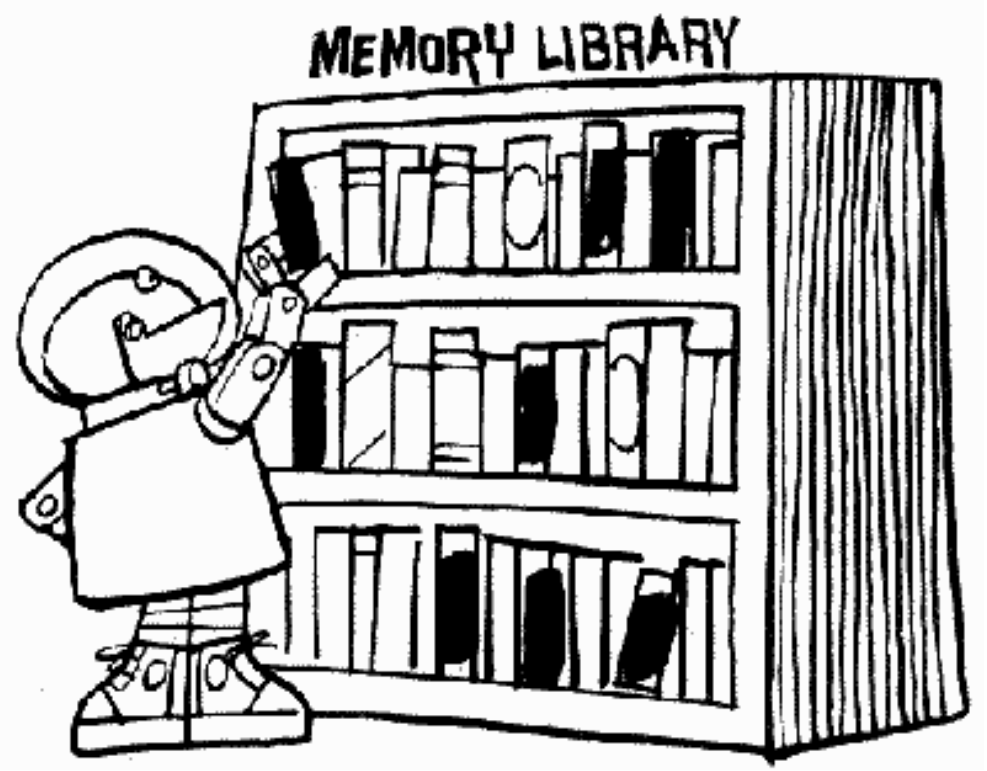
- We use the term "buffer" quite a lot but haven't yet defined it.
  - One way to think of it is just a memory region that has some use
  - Typically referenced and maintained through a pointer:

```
char *buffer;
char buf[256];
```

- Definition: a buffer is a temporary holding place

Memory library functions

- Found in `<string.h>`
  - Makes sense if you think about it
  - Copying memory
  - Setting memory
  - Comparing memory
- Found in `<stdlib.h>`
  - Allocating memory
  - Freeing memory
- Others in `<unistd.h>`



Copying memory

```
char buf1[] = {0, 1, 2, 3};
char buf2[] = {0, 0, 0, 0};

printf("Before\n");
for (i = 0; i < 4; i++)
    printf("buf1[i]: %d, buf2[i]: %d\n",
           buf1[i], buf2[i]);

// Copy buf1 into buf2
memcpy(buf2, buf1, 4);

printf("After\n");
for (i = 0; i < 4; i++)
    printf("buf1[i]: %d, buf2[i]: %d\n",
           buf1[i], buf2[i]);
```

```
Before
buf1[i]: 0, buf2[i]: 0
buf1[i]: 1, buf2[i]: 0
buf1[i]: 2, buf2[i]: 0
buf1[i]: 3, buf2[i]: 0
After
buf1[i]: 0, buf2[i]: 0
buf1[i]: 1, buf2[i]: 1
buf1[i]: 2, buf2[i]: 2
buf1[i]: 3, buf2[i]: 3
```

- `memcpy` copies one memory region to another:

```
memcpy(dest, src, n);
```

  - Copy n bytes into "destination" buffer from "source" buffer
    - Mnemonic: order is the same as "dest = src"
  - The size must be specified (unlike `strcpy`, there is no terminator)

Also copying memory

- `memmove` also copies one memory region to another:

```
memmove(dest, src, n);
```

- Difference: `memmove` works if memory regions overlap; `memcpy` has undefined behavior in this case
  - But `memcpy` can be faster if you know they don't overlap

Setting memory

```
char buf1[] = {0, 1, 2, 3};
char buf2[] = {0, 0, 0, 0};

printf("Before\n");
for (i = 0; i < 4; i++)
    printf("buf1[i]: %d, buf2[i]: %d\n",
           buf1[i], buf2[i]);

memset(buf1, 0, 4);
memset(buf2, 7, 4);

printf("After\n");
for (i = 0; i < 4; i++)
    printf("buf1[i]: %d, buf2[i]: %d\n",
           buf1[i], buf2[i]);
```

```
Before
buf1[i]: 0, buf2[i]: 0
buf1[i]: 1, buf2[i]: 0
buf1[i]: 2, buf2[i]: 0
buf1[i]: 3, buf2[i]: 0
After
buf1[i]: 0, buf2[i]: 7
buf1[i]: 0, buf2[i]: 7
buf1[i]: 0, buf2[i]: 7
buf1[i]: 0, buf2[i]: 7
```

- `memset` fills a memory region with one byte:

```
memset(buf, c, n);
```

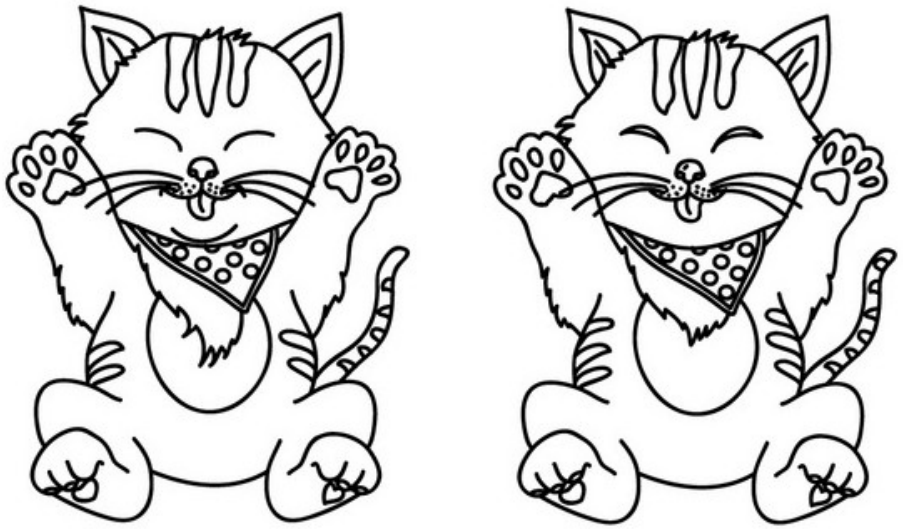
  - Writes n copies of byte c into buf
  - Useful for zeroing newly allocated memory
    - More on allocating memory next time...

Comparing memory

- `memcmp` compares the first n bytes of two buffers:

```
memcmp(buf1, buf2, n);
```

- Return value interpretation is the same as `strcmp`:
  - Negative if `buf1` is less than `buf2`
  - 0 if `buf1` is equal to `buf2`
  - Positive if `buf1` is greater than `buf2`



memcmp example

```
int i, j, x;
char cmps[4][2] = {
        {0, 0}, {1, 0},
        {0, 1}, {9, 0}};

printf("Before\n");
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        x = memcmp(cmps[i], cmps[j], 2);
        printf("compare %d%d with %d%d = %d\n",
                cmps[i][0], cmps[i][1],
                cmps[j][0], cmps[j][1], x);
    }
}
```

```
compare 00 with 00 = 0
compare 00 with 10 = -1
compare 00 with 01 = -256
compare 00 with 90 = -9
compare 10 with 00 = 1
compare 10 with 10 = 0
compare 10 with 01 = 1
compare 10 with 90 = -8
compare 01 with 00 = 256
compare 01 with 10 = -1
compare 01 with 01 = 0
compare 01 with 90 = -9
compare 90 with 00 = 9
compare 90 with 10 = 8
compare 90 with 01 = 9
compare 90 with 90 = 0
```

Midterm 1