

CS 211: Intro to Computer Architecture

6.2: C Data Representation IV: Derived Types Cont.

Minesh Patel

Spring 2025 – Thursday 27 February

Policy on Disrespect

Respect your instructional staff.

- This includes **all instructors, TAs, and tutors**, including during recitations and office hours, and at the CSL
- We **will not tolerate** disruptions to a healthy learning environment
- Continued disruptions may be **reported to any or all** of the following and may result in being denied help in recitations and office hours
 - RUPD
 - Residents Life
 - The Dean of Students

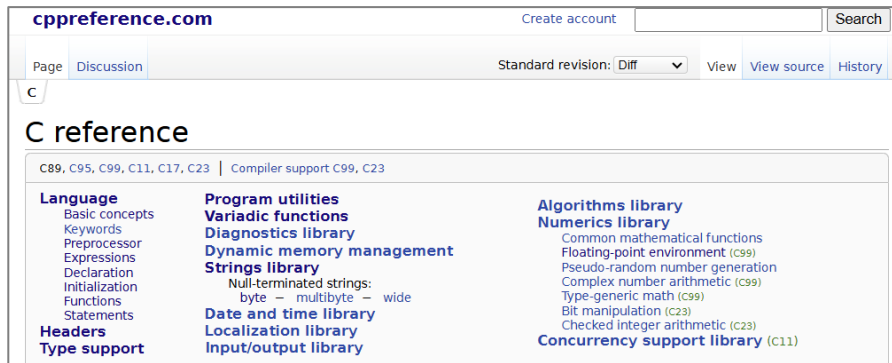
Announcements

- PA2 survey will be on Canvas
- Coming up next week:
 - **PA3** and **WA4**
 - Exam information (TBA)
 - ODS: please remember to contact ODS for special accommodations for the exam

Reference Material

- Today's lecture partially draws inspiration from:
 - [CS 61C @ UC Berkeley](#) (Prof. Dan Garcia)

And Various C and Linux Reference Materials



cppreference.com

Create account Search

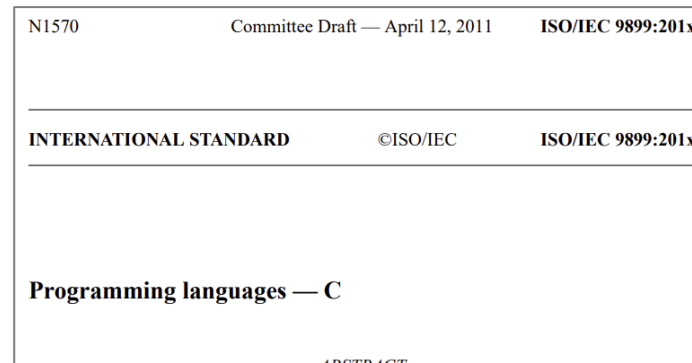
Page Discussion Standard revision: Diff View View source History

C

C reference

C89, C95, C99, C11, C17, C23 | Compiler support C99, C23

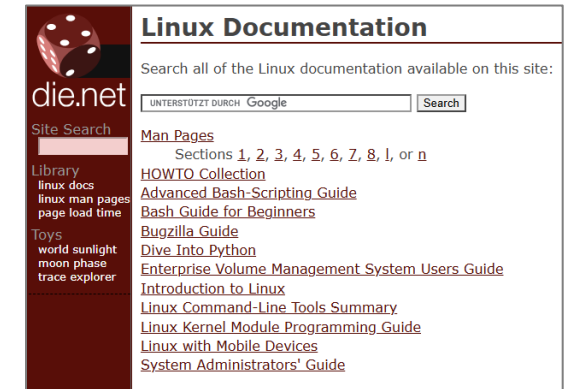
Language <ul style="list-style-type: none">Basic conceptsKeywordsPreprocessorExpressionsDeclarationInitializationFunctionsStatements	Program utilities <ul style="list-style-type: none">Variadic functionsDiagnostics libraryDynamic memory managementStrings library<ul style="list-style-type: none">Null-terminated strings:<ul style="list-style-type: none">byte – multibyte – wideDate and time libraryLocalization libraryInput/output library	Algorithms library <ul style="list-style-type: none">Numerics library<ul style="list-style-type: none">Common mathematical functionsFloating-point environment (C99)Pseudo-random number generationComplex number arithmetic (C99)Type-generic math (C99)Bit manipulation (C23)Checked integer arithmetic (C23)Concurrency support library (C11)
--	---	--



N1570 Committee Draft — April 12, 2011 ISO/IEC 9899:201x

INTERNATIONAL STANDARD ©ISO/IEC **ISO/IEC 9899:201x**

Programming languages — C



die.net

Linux Documentation

Search all of the Linux documentation available on this site:

UNTERSTÜTZT DURCH Google Search

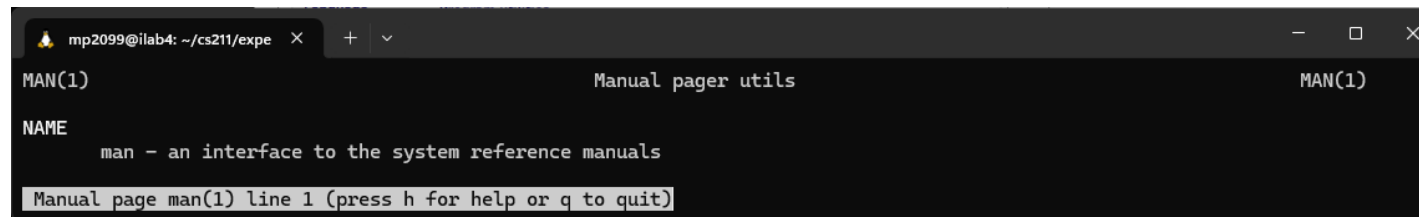
Site Search

Man Pages
Sections 1, 2, 3, 4, 5, 6, 7, 8, 1, or n

Library
linux docs
linux man pages
page load time

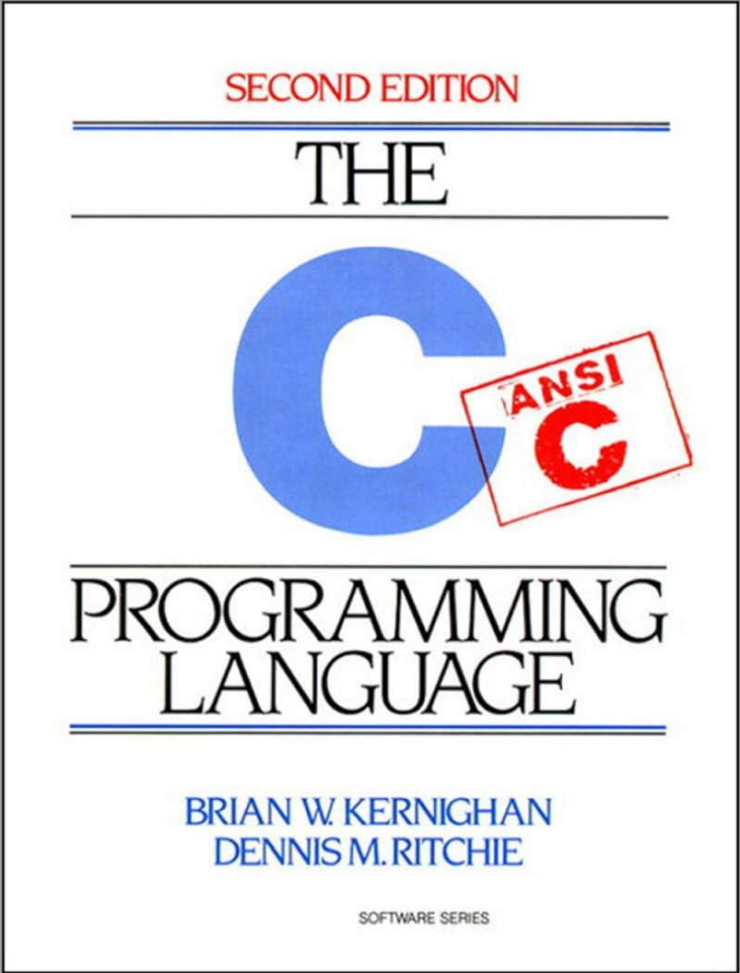
Toys
world sunlight
moon phase
trace explorer

- [HOWTO Collection](#)
- [Advanced Bash-Scripting Guide](#)
- [Bash Guide for Beginners](#)
- [Bugzilla Guide](#)
- [Dive Into Python](#)
- [Enterprise Volume Management System Users Guide](#)
- [Introduction to Linux](#)
- [Linux Command-Line Tools Summary](#)
- [Linux Kernel Module Programming Guide](#)
- [Linux with Mobile Devices](#)
- [System Administrators' Guide](#)

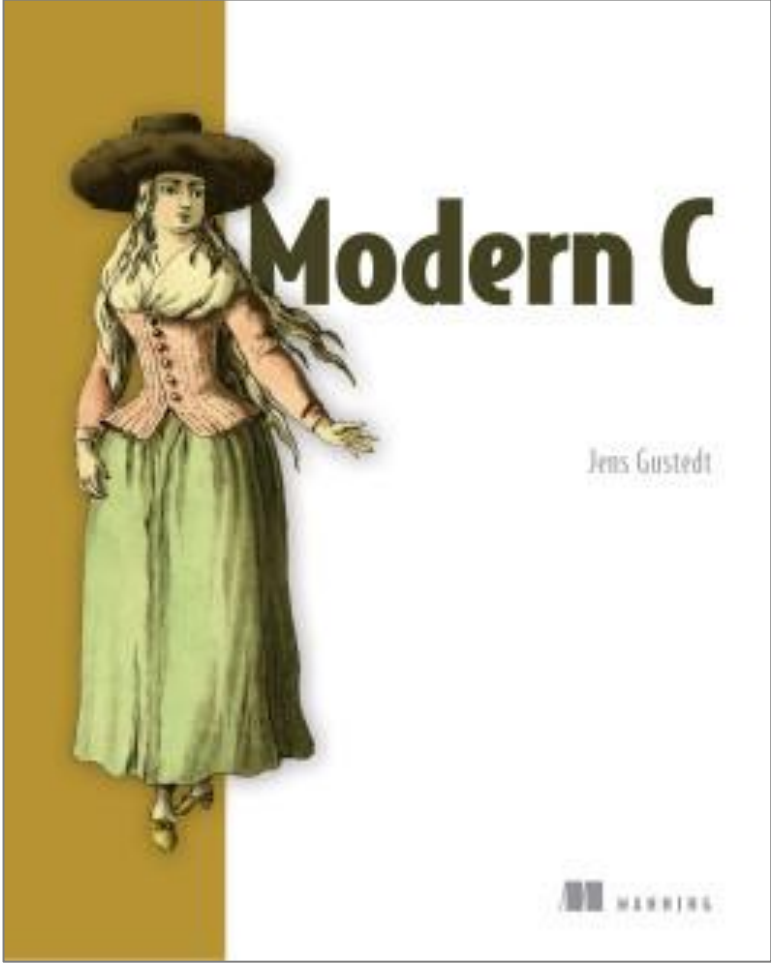


```
mp2099@ilab4: ~/cs211/expe x + v
MAN(1) Manual pager utils MAN(1)
NAME
    man - an interface to the system reference manuals
Manual page man(1) line 1 (press h for help or q to quit)
```

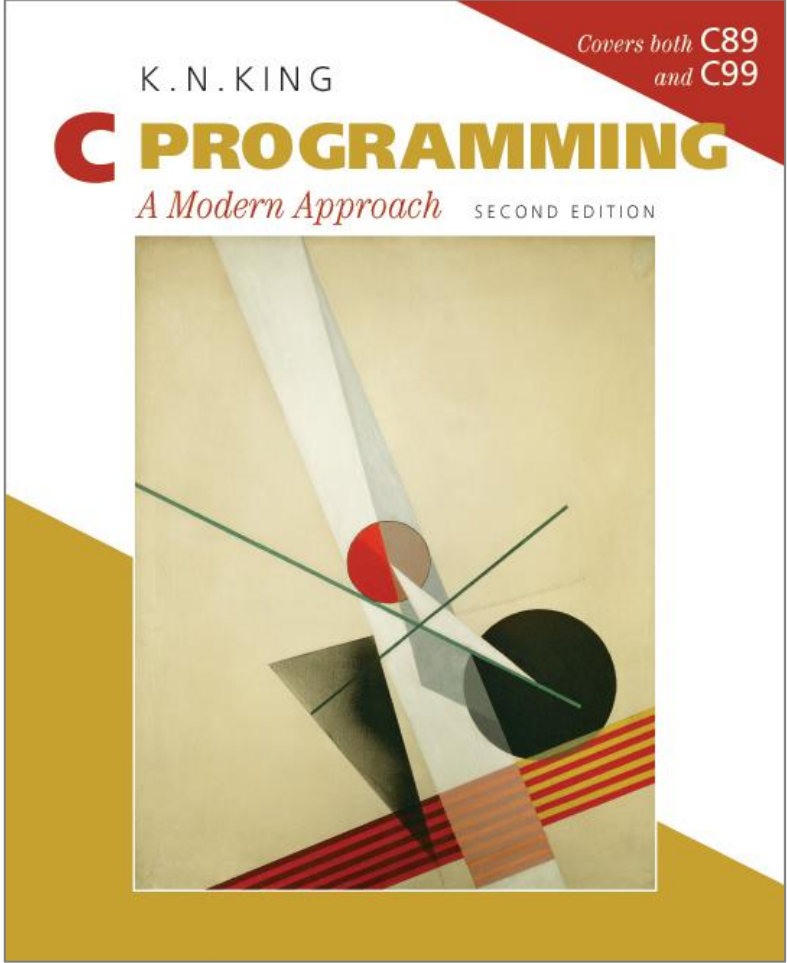
Standards are Great, But...



https://m.media-amazon.com/images/I/51EyaJeebHL._SL1056_.jpg



<https://www.manning.com/books/modern-c>



<http://knking.com/books/c2/cover.html>

Agenda

- void
- basic types
 - char
 - signed integers
 - unsigned integers
 - floating-point
- enumerated types
- **derived types**
 - structures
 - pointers
 - **arrays**
 - unions
 - functions

• **Arrays and Strings**

- Array nuances

• Unions

Pointer Fun with

B **i** **n** **k** **y**



by Nick Parlante

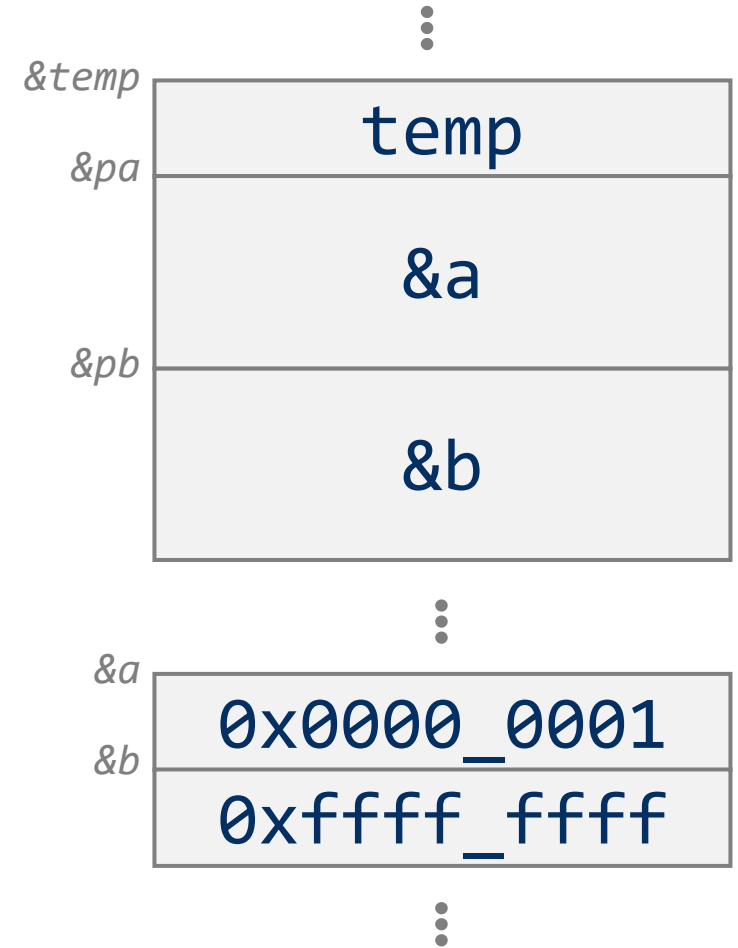
This is document 104 in the Stanford CS
Education Library — please see
cslibrary.stanford.edu
for this video, its associated documents,
and other free educational materials.

Copyright © 1999 Nick Parlante. See copyright
panel for redistribution terms.
Carpe Post Meridiem!

Recap: Pointer Example

```
void swap(int *pa, int *pb)
{
    int temp = *pa;
    *pa = *pb;
    *pb = temp;
}

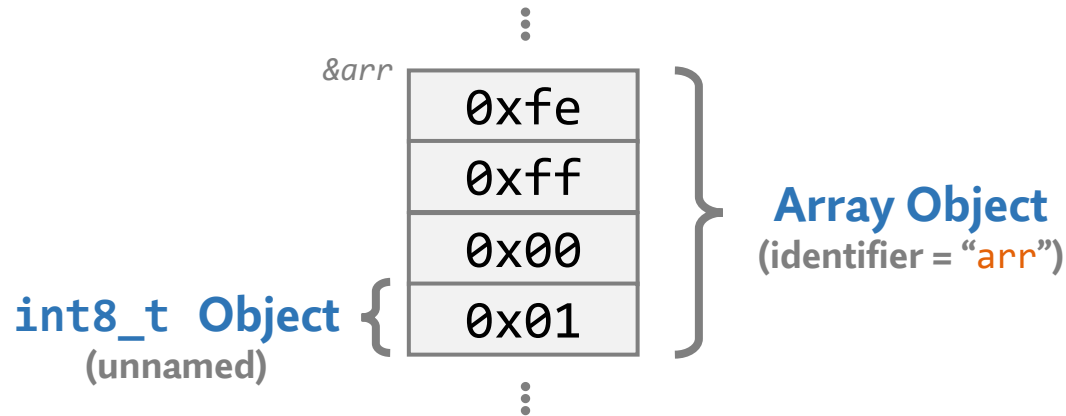
void func(void)
{
    int a = 1, b = -1;
    swap(&a, &b); // swaps a and b
}
```



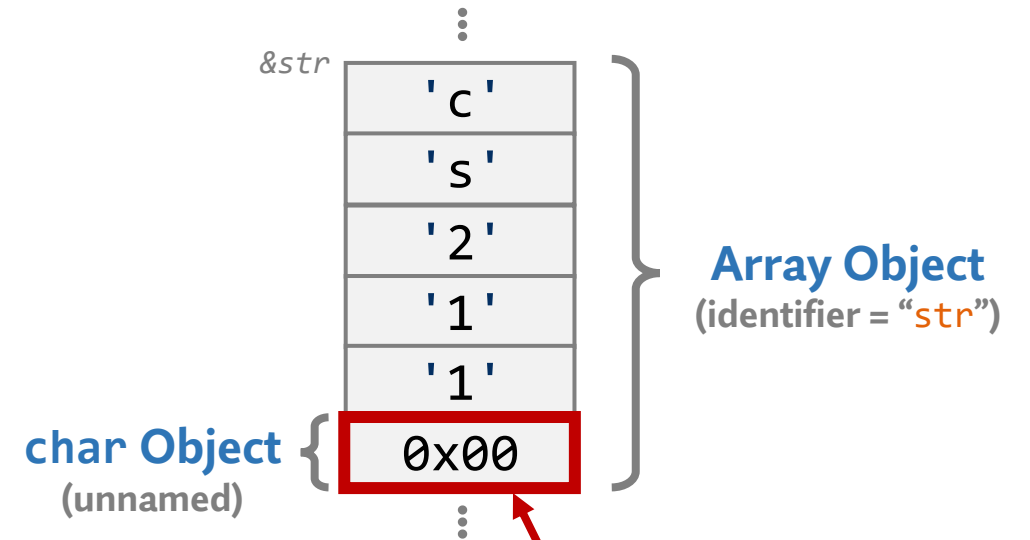
Recap: Arrays and Strings

- **Arrays objects** are a list of objects in memory

```
int8_t arr[4] = {-2, -1, 0, 1};
```



```
char str[] = "cs211";
```



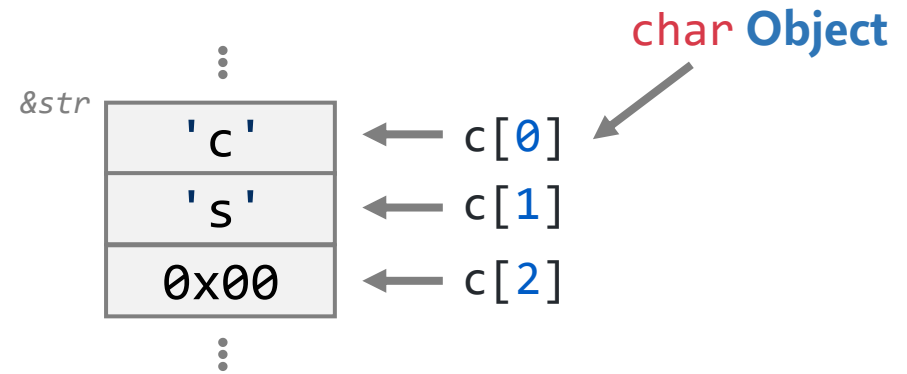
"null terminator" for C strings
(C library functions expect this)

Indexing C Arrays

- Array indexing is very straightforward

```
void func(void)
{
    char c[3] = {'c', 's', '\0'};
    printf("%c%c", c[0], c[1]);
    printf("%s", c);
}
```

Special format specifier for C strings
(expects the null termination)



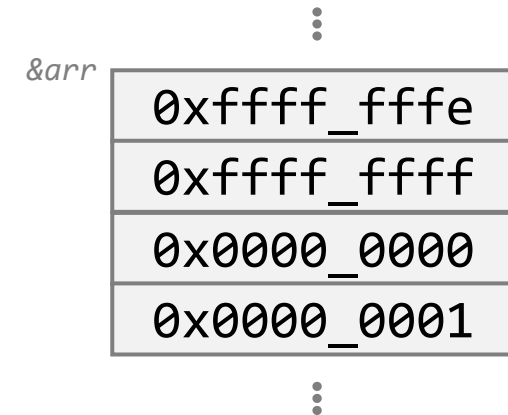
Size vs. Length of an Array

- **Size:** total bytes in memory
- **Length:** number of elements

```
void func(void)
{
    int32_t arr[4] = {-2, -1, 0, 1};

    int size = sizeof(arr); // 16 (bytes)

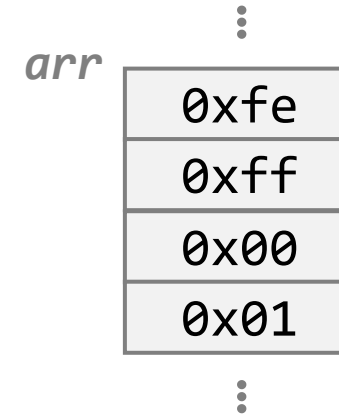
    int length = sizeof(arr) / sizeof(arr[0]);
                // 16 (bytes) / 4 (bytes) = 4 elements
}
```



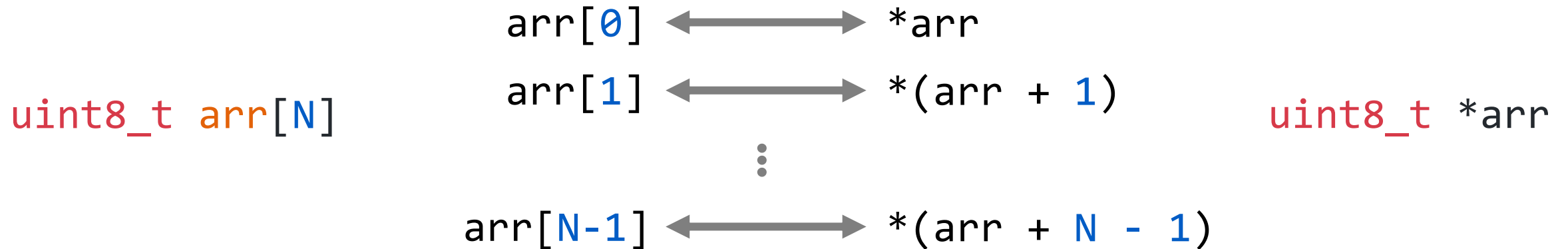
Under the Hood, Indexing is Just Pointers

- **Array identifiers** act like **pointers** (to the first element)

```
uint8_t arr[4] = {0, 1, 2, 3};  
uint8_t a0 = *arr;           // == arr[0]  
uint8_t a1 = *(arr + 1);    // == arr[1]  
...
```



- **Element access** is just shorthand for **pointer arithmetic**



C Strings

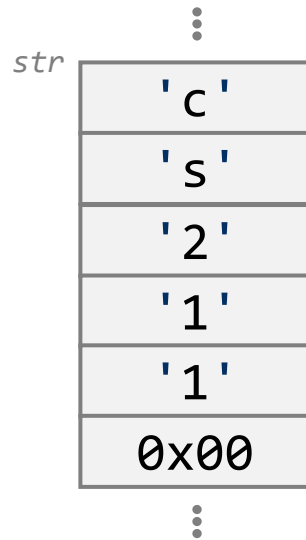
C Strings (have null termination)

```
char *str = "cs211";
```

```
char str[6] = "cs211";
```

```
char str[] = "cs211";
```

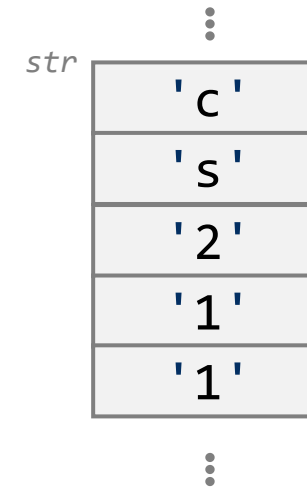
```
char str[] = {'c', 's', '2', '1', '1', '\0'};
```



```
printf("%s", str);
```

NOT a C String

```
char str[] = {'c', 's', '2', '1', '1'};
```



```
printf("%s", str);
```

Pointer Arithmetic

Element access is just shorthand for **pointer arithmetic**

$$p[n] \Leftrightarrow *(p + n)$$

- Adding/subtracting **integers** operates on units of **1**

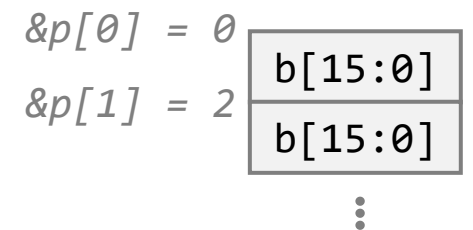
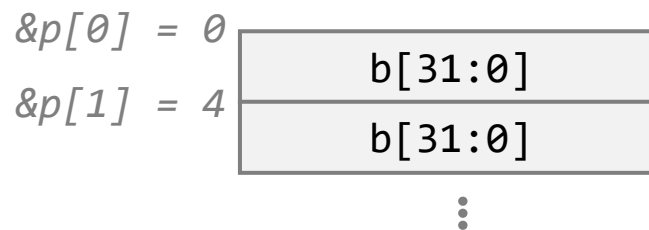
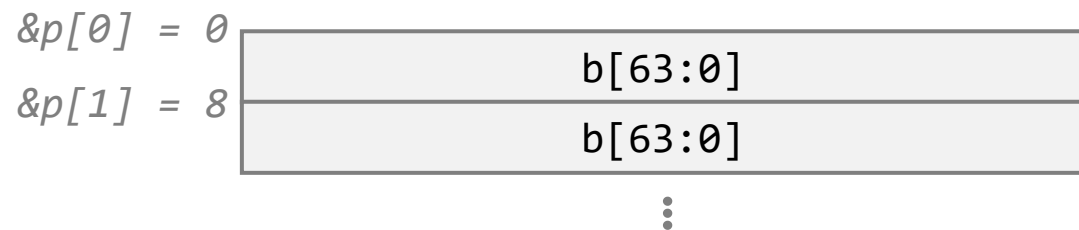
```
uint64_t u = 0;  
u = u + 1; // u = 1
```

- Adding/subtracting **pointers** operates on units of "**sizeof(pointee)**"

```
uint64_t *p = 0;  
p = p + 1; // p = 8
```

```
uint32_t *p = 0;  
p = p + 1; // p = 4
```

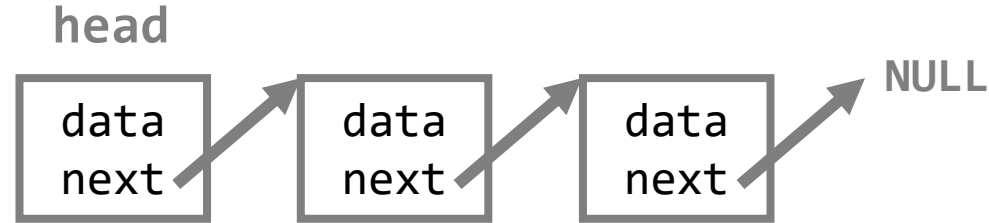
```
uint16_t *p = 0;  
p = p + 1; // p = 2
```



Example: Linked List with Pointer Arithmetic

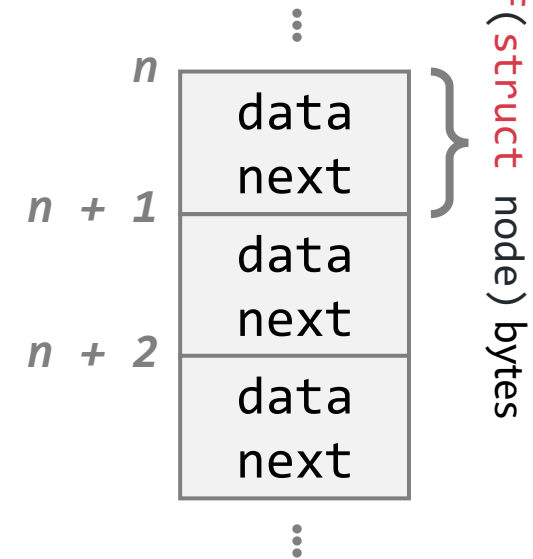
Singly-Linked List

```
struct node
{
    uint8_t data;
    struct node *next;
}
```



```
#define N 100
struct node n[N]; // array of statically-allocated nodes

for(int i = 0; i < N; i++)
{
    struct node *this = n + i; // &n[i]
    this->data = (uint8_t)i;
    this->next = (i == N - 1) ? NULL : (n + i + 1); // &n[i + 1]
}
```



Example: "argv"

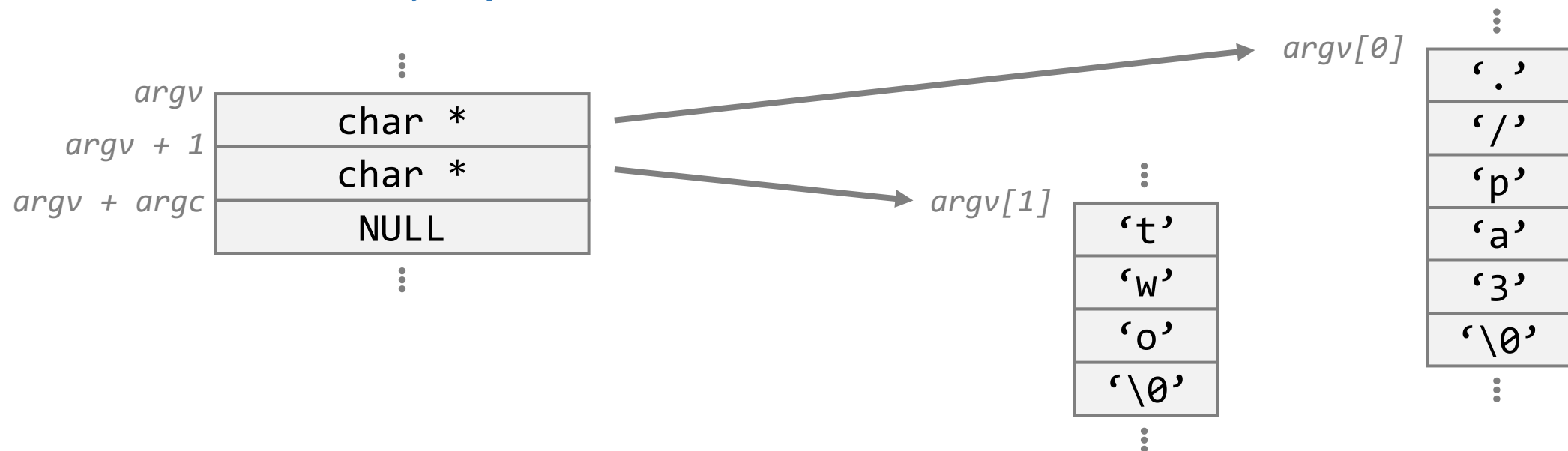
```
netid@ilab:~/cs211$ ./pa3 two
```

```
int main(int argc, char *argv[])  
{  
    // code  
}
```

sizeof(argv) won't work
(compiler doesn't know the length)

argv is an **array of pointers**

argv[n] are **C strings**

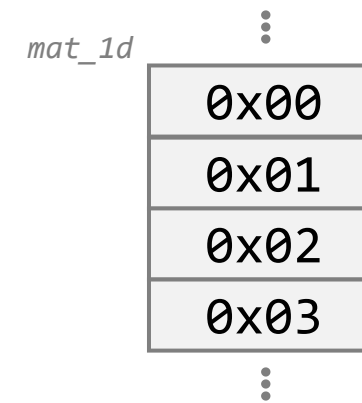
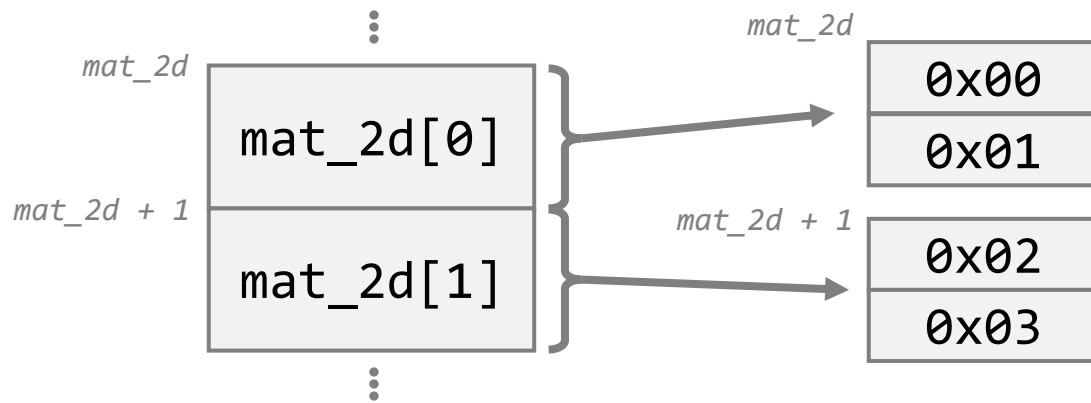


Multidimensional Arrays

- The compiler “does the right thing” for multidimensional arrays

```
uint8_t mat_2d[][2] = {{0, 1}, {2, 3}};  
// sizeof(mat2d) == 4
```

```
uint8_t mat_1d[] = {0, 1, 2, 3};  
// sizeof(mat1d) == 4
```



Example: Flattened Matrix

```
#define N 100
#define M 100

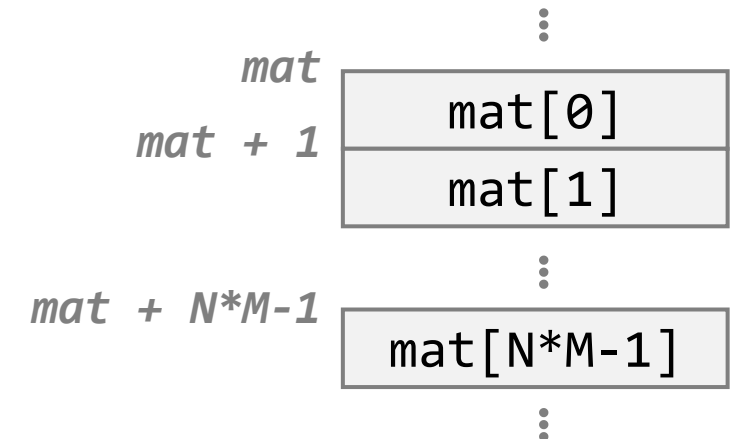
int print_mat(uint32_t *mat)
{
    for(int n = 0; n < N; n++) // rows
        for(int m = 0; m < M; m++) // cols
            printf(mat[m + n * M]); // *(mat + m + n * M )
}

int main(int argc, char *argv[])
{
    uint32_t mat[N * M] = {0}; // statically-allocated array
    print_mat(mat);
}
```

Logical Organization

0	1	2	...	M-1
M	M+1	M+2	...	2M-1
2M	2M+1	2M+2	...	3M-1
⋮	⋮	⋮	⋮	⋮
(N-1)*M	(N-1)*M + 1	(N-1)*M + 2	...	NM - 1

Physical Organization



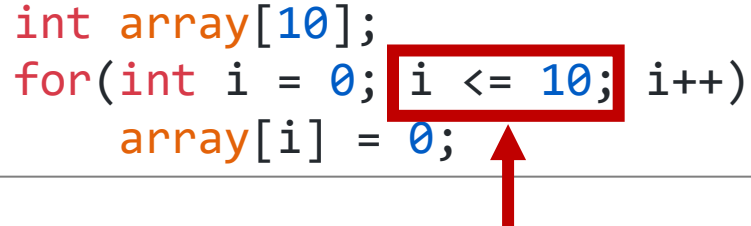
Agenda

- void
- basic types
 - char
 - signed integers
 - unsigned integers
 - floating-point
- enumerated types
- **derived types**
 - structures
 - **pointers**
 - arrays
 - unions
 - functions

- **Pointers**
 - Basics (continued)
 - Why pointers?
 - Pointer nuances
- **Arrays and Strings**
 - **Array nuances**

Caution: Array Off-By-One Errors

```
int array[10];  
for(int i = 0; i <= 10; i++)  
    array[i] = 0;
```



Array “out of bounds” access

- C does NOT check for you!
- Undefined behavior: can be hard to debug

- Avoid magic numbers – error prone and hard to read/change

```
#define LEN 10  
  
int array[LEN];  
for(int i = 0; i < LEN; i++)  
    array[i] = 0;
```

CVEs on Arrays

```
ec 28 89 54 24 1c 48 8974 24 10 [ 18.881766] RSP: 002b:00007ffcdd00fad8 EFLAGS: 00000246 ORIG_RAX: 000000000000002e [ 18.882149] RAX: ffffffffda RBX: 00007ffcdd010db8 RCX: 00000000044a957 [ 18.882507] RDX: 0000000000000000 RSI: 00007ffcdd00fb70 RDI: 0000000000000003 [ 18.885037] RBP: 00007ffcdd010bc0 R08: 000000000703c770 R09: 000000000703c7c0 [ 18.887203] R10: 0000000000000080 R11: 0000000000000246 R12: 0000000000000001 [ 18.888026] R13: 00007ffcdd010da8 R14: 00000000004ca7d0 R15: 0000000000000001 [ 18.888395] </TASK> [ 18.888610] ---[ end trace ]---
```

[CVE-2025-21680](#)

In the **Linux kernel** the following vulnerability has been resolved: pktgen: Avoid out-of-bounds access in get_imix_entries Passing a sufficient amount of imix entries leads to invalid access to the pkt_dev->imix_entries array because of the incorrect boundary check. UBSAN: array-index-out-of-bounds in net/core/pktgen.c:874:24 index 20 is out of range for type 'imix_pkt [20]' CPU: 2 PID: 1210 Comm: bash Not tainted 6.10.0-rc1 #121 Hardware name: QEMU Standard PC (i440FX + PIIX, 1996) Call Trace: <TASK> dump_stack_lvl lib/dump_stack.c:117 __ubsan_handle_out_of_bounds lib/ubsan.c:429 get_imix_entries net/core/pktgen.c:874 pktgen_if_write net/core/pktgen.c:1063 pde_write fs/proc/inode.c:334 proc_reg_write fs/proc/inode.c:346 vfs_write fs/read_write.c:593 ksys_write fs/read_write.c:644 do_syscall_64 arch/x86/entry/common.c:83 entry_SYSCALL_64_after_hwframe arch/x86/entry_64.S:130 Found by Linux Verification Center (linuxtesting.org) with SVACE. [fp: allow to fill the array completely; minor changelog cleanup]

[CVE-2025-21643](#)

In the **Linux kernel** the following vulnerability has been resolved: netfs: Fix kernel async DIO Netfslib needs to be able to handle kernel-initiated asynchronous DIO that is supplied with a bio_vec[] array. Currently, because of the async flag, this gets passed to netfs_extract_user_iter() which throws a warning and fails because it only handles IOVEC and UBUF iterators. This can be triggered through a combination of cifs and a loopback blockdev with something like: mount //my/cifs/share /foo dd if=/dev/zero of=/foo/m0 bs=4K count=1K losetup --sector-size 4096 --direct-io=on /dev/loop2046 /foo/m0 echo hello >/dev/loop2046 This causes the following to appear in syslog: WARNING: CPU: 2 PID: 109 at fs/netfs/iterator.c:50 netfs_extract_user_iter+0x170/0x250 [netfs] and the write to fail. Fix this by removing the check in netfs_unbuffered_write_iter_locked() that causes async kernel DIO writes to be handled as userspace writes. Note that this change relies on the kernel caller maintaining the existence of the bio_vec array (or kvec[] or folio_queue) until the op is complete.

[CVE-2024-9143](#)

Issue summary: Use of the low-level GF(2^m) elliptic curve APIs with untrusted explicit values for the field polynomial can lead to out-of-bounds memory reads or writes. Impact summary: Out of bound memory writes can lead to an application crash or even a possibility of a remote code execution, however, in all the protocols involving Elliptic Curve Cryptography that we're aware of, either only "named curves" are supported, or, if explicit curve parameters are supported, they specify an X9.62 encoding of binary (GF(2^m)) curves that can't represent problematic input values. Thus the likelihood of existence of a vulnerable application is low. In particular, the X9.62 encoding is used for ECC keys in X.509 certificates, so problematic inputs cannot occur in the context of processing X.509 certificates. Any problematic use-cases would have to be using an "exotic" curve encoding. The affected APIs include: EC_GROUP_new_curve_GF2m(), EC_GROUP_new_from_params(), and various supporting BN_GF2m_*() functions. Applications working with "exotic" explicit binary (GF(2^m)) curve parameters, that make it possible to represent invalid field polynomials with a zero constant term, via the above or similar APIs, may terminate abruptly as a result of reading or writing outside of array bounds. Remote code execution cannot easily be ruled out. The FIPS modules in 3.3, 3.2, 3.1 and 3.0 are not affected by this issue.

[CVE-2024-8408](#)

A vulnerability was found in **Linksys WRT54G 4.21.5**. It has been rated as critical. Affected by this issue is the function validate_services_port of the file /apply.cgi of the component POST Parameter Handler. The manipulation of the argument services_array leads to stack-based buffer overflow. The attack may be launched remotely. The exploit has been disclosed to the public and may be used. The vendor was contacted early about this disclosure but did not respond in any way.

[CVE-2024-7721](#)

The HTML5 Video Player – mp4 Video Player Plugin and Block **plugin for WordPress** is vulnerable to unauthorized modification of data due to a missing capability check on the 'save_password' function in all versions up to, and including, 2.5.34. This makes it possible for authenticated attackers, with Subscriber-level access and above, to set any options that are not explicitly checked as false to an array, including enabling user registration if it has been disabled.

[CVE-2024-6606](#)

Clipboard code failed to check the index on an array access. This could have led to an out-of-bounds read. This vulnerability affects **Firefox** < 128 and Thunderbird < 128.

[CVE-2024-5991](#)

In function MatchDomainName(), input param str is treated as a NULL terminated string despite being user provided and unchecked. Specifically, the function X509_check_host() takes in a pointer and length to check against, with no requirements that it be NULL terminated. If a caller was attempting to do a name check on a non-NULL terminated buffer, the code would read beyond the bounds of the input array until it found a NULL terminator. This issue affects **wolfSSL** through 5.7.0.

[CVE-2024-58015](#)

In the **Linux kernel** the following vulnerability has been resolved: wifi: ath12k: Fix for out-of bound access error Selfgen stats are placed in a buffer using print_array_to_buf_index() function. Array length parameter passed to the function is too big, resulting in possible out-of bound memory error. Decreasing buffer size by one fixes faulty upper bound of passed array. Discovered in coverity scan, CID 1600742 and CID 1600758

[CVE-2024-58000](#)

In the **Linux kernel** the following vulnerability has been resolved: io_uring: prevent reg-wait speculations With *ENTER_EXT_ARG_REG instead of passing a user pointer with arguments for the waiting loop the user can specify an offset into a pre-mapped region of memory, in which case the [offset, offset + sizeof(io_uring_reg_wait)) will be interpreted as the argument. As we address a kernel array using a user given index, it'd be a subject to speculation type of exploits. Use array_index_nospec() to prevent that. Make sure to pass not the full region size but truncate by the maximum offset allowed considering the structure size.

[CVE-2024-57996](#)

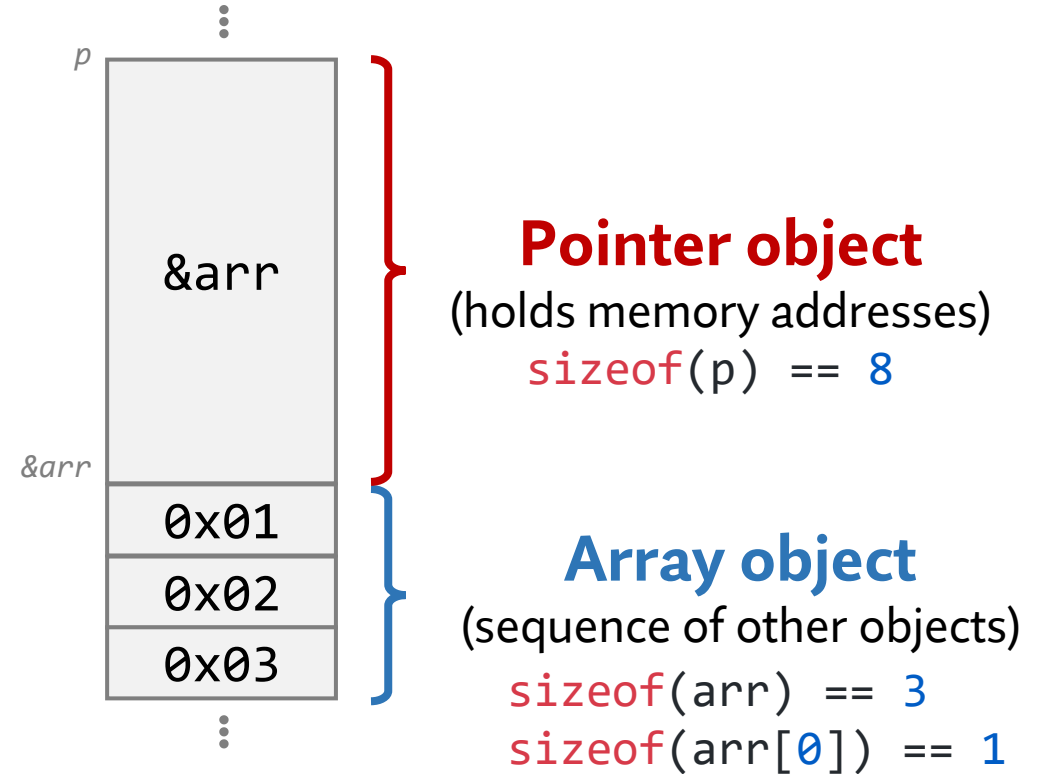
In the **Linux kernel** the following vulnerability has been resolved: net_sched: sch_sfq: don't allow 1 packet limit The current implementation does not work correctly with a limit of 1. iproute2 actually checks for this and this patch adds the check in kernel as well. This fixes the following syzkaller reported crash: UBSAN: array-index-out-of-bounds in net/sched/sch_sfa.c:210:6 index 65535 is out of range for type 'struct sfa_head[128]' CPU: 0 PID: 2569 Comm: svz-executor101 Not tainted 5.10.0-smp-DEV #1 Hardware name:

Array vs. Pointer Identifiers

Array identifiers refer to
one array object

```
uint8_t arr[3] = {1, 2, 3};  
uint8_t *p = &arr;
```

Pointer identifiers refer to
one pointer object



Pointer Arithmetic on Arrays

```
uint8_t arr[3] = {1, 2, 3};  
uint8_t *p = &arr;
```

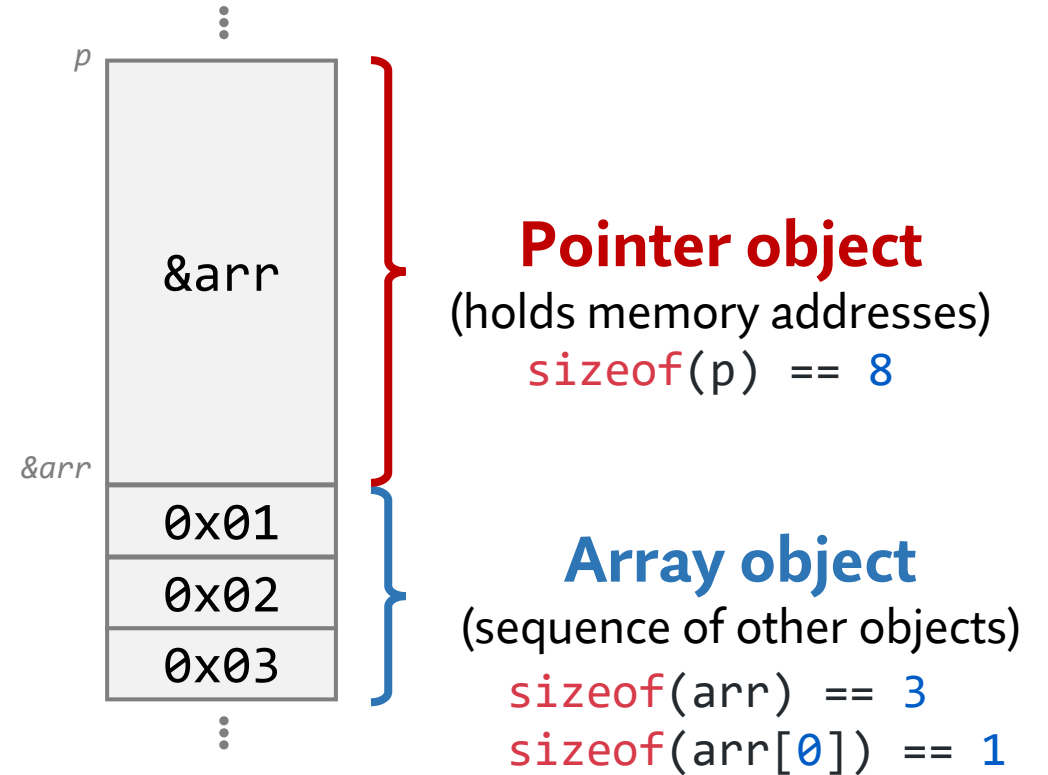
Array Objects are NOT Pointers

```
arr++; // illegal
```

Arrays and Pointers-to-Arrays Are Different

```
uint8_t *p = arr;  
p++; // p = &arr[0]
```

```
uint8_t (*pa)[3] = &arr;  
pa++; // pa = &arr[4] (OOB)
```



Demotion to Pointers

- Array objects are **passed as pointers** when passed to a function

```
void func(int *arr);  
void func(int arr[]);
```

 } Equivalent

```
int main(int argc, char *argv[]);  
int main(int argc, char **argv);
```

 } Equivalent

- Function arguments lose the array type information
 - Can no longer use `sizeof(arr)`
 - Need to explicitly pass the array length or use a sentinel to mark the end

Agenda

- void
- basic types
 - char
 - signed integers
 - unsigned integers
 - floating-point
- enumerated types
- **derived types**
 - structures
 - pointers
 - arrays
 - **unions**
 - functions

• **Arrays and Strings**

- Array nuances

• **Unions**

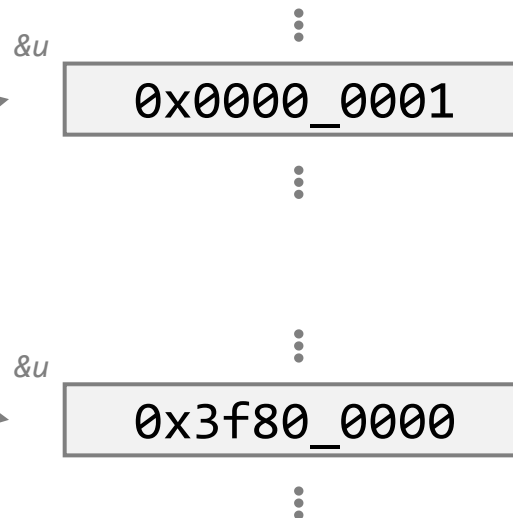
Union Types

A union type describes an overlapping nonempty set of member objects

ISO Standard 9899:2011 6.2.5.20

```
union my_union  
{  
    uint32_t i;  
    float f;  
};
```

```
union my_union u;  
u.i = 1;  
u.f = 1;  
// ...
```



Same memory location
(updates overwrite each other)

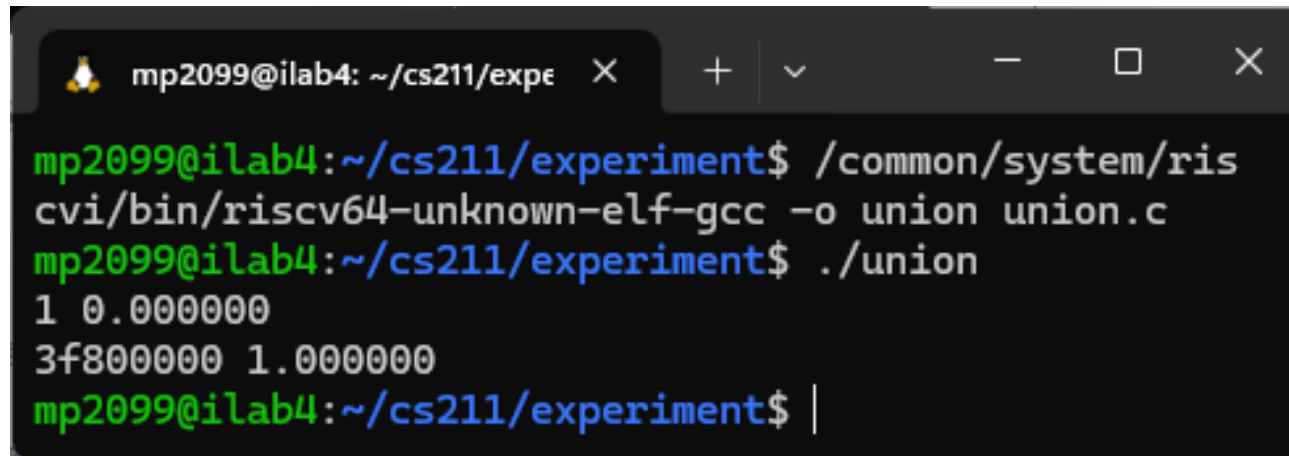
Union Example Code

```
union my_union
{
    uint32_t i;
    float f;
};
```

```
union my_union u;
// sizeof(u) == 4

u.i = 1;
printf(“%X %f”, u.i, u.f);

u.f = 1;
printf(“%X %f”, u.i, u.f);
```



```
mp2099@ilab4: ~/cs211/expe x + v - □ x
mp2099@ilab4:~/cs211/experiment$ /common/system/riscv/bin/riscv64-unknown-elf-gcc -o union union.c
mp2099@ilab4:~/cs211/experiment$ ./union
1 0.000000
3f800000 1.000000
mp2099@ilab4:~/cs211/experiment$ |
```

Unions vs. Pointers

- You can do the same thing by **type casting pointers**

```
union my_union
{
    uint32_t i;
    float f;
};
```

```
union my_union u;

uint32_t i = 1;
printf(“%f”, *(float *)&i);

float f = 1;
printf(“%X”, *(uint32_t *)&f);
```

- However: unions **guarantee** that the object can represent the type
- Unions are generally **safer** than type casting pointers

[Demo] Examining Memory in GDB

Terminal Emulator 1:

```
netid@ilab4:~/cs211/experiment$ /common/system/riscv64i/grun pointer
```

same ilab machine! same directory!

Terminal Emulator 2:

```
netid@ilab4:~/cs211/experiment$ /common/system/riscv64i/gdb pointer
```

- **start** or **run** – start the session
- **break <location>** - set a breakpoint
 - **b main** – break at the main function
 - **b main.c:11** – break there, for example
- **p <variable>** - print a variable
 - **p/x <variable>** – print in hex
 - **p/t <variable>** – print in binary
- **x <address>** - print memory at the address
 - **x/10b <address>** – print 10 bytes
 - **x/10b <variable>** - get the address from a variable
- **layout <command>** - change the GDB layout

```
mp2099@ilab4: ~/cs211/expe x + v
mp2099@ilab4:~/cs211/experiment$ /common/system/riscv64i/grun pointer
cd /common/home/mp2099/cs211/experiment
['/usr/bin/qemu-riscv64', '-g', '/common/home/mp2099/grun.sock2', '/common/home/mp2099/cs211/experiment/pointer']
! @ 0x40008002a7

mp2099@ilab4: ~/cs211/expe x + v
pointer.c
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(int argc, char *argv[])
5 {
6     char c = '!'; // ascii 0x21
7     char *p = &c; // address of c
8
9     printf("%c @ %p\n", c, p);
10    return EXIT_SUCCESS;
11 }

remote Thread 1.3921658 (src) In: main L10 PC: 0x10210
(gdb) p/x c
$3 = 0x21
(gdb) p/x p
$4 = 0x40008002a7
(gdb) x/10b p
0x40008002a7: 0x21 0xa7 0x02 0x80 0x00 0x40 0x00 0x00
0x40008002af: 0x00 0x00
(gdb) x/10b 0x40008002a7
0x40008002a7: 0x21 0xa7 0x02 0x80 0x00 0x40 0x00 0x00
0x40008002af: 0x00 0x00
(gdb) |
```

Next Week

- Functions
- Memory layout of a C program
- Dynamic memory allocation

CS 211: Intro to Computer Architecture

6.2: C Data Representation IV: Derived Types Cont.

Minesh Patel

Spring 2025 – Thursday 27 February