

# CS441 4 Recitation 1

## Introduction and C++ basics

---

08/29/2025

Alicia Yang , Shouxu Lin

## About TA -- Alicia

- Senior PhD student in CS
- Advised by Prof. Birman on distributed systems
- Research focus: cluster scheduling, ML serving systems
- Office hours:
  - After recitation from Friday 4:10PM
  - By appointment, email: [yy354@cornell.edu](mailto:yy354@cornell.edu)

# About TA -- Shouxu

- Senior PhD student in CS
- Advised by Prof. Birman on ML systems
- Research focus: cluster scheduling, ML serving systems
- Office hours:
  - After recitation from Friday 4:10PM
  - By appointment, email: [sl3343@cornell.edu](mailto:sl3343@cornell.edu)

# Logistics

- TA Help Session: C++ Coding Environment Setup
  - Session 1: 7:00 PM – 8:00 PM, Thursday, 08/28 (led by Nam Anh Dang)  
Location: Uris G01
  - Session 2: 7:15 PM - 8:15 PM, Tuesday, 09/03 (led by Haadi Khan and Ryan Wu)  
Location: Phillips 101

HW1 will be released on Monday

**Ed discussion** announcement

The writeup and starter code are on **Canvas**

Submission to **Gradescope**

# | Logistics

- We will be using
- Log in with your Cornell email at <http://pollev.com/>

# Overview

- Recitation introduction and logistics
- C++ primitive types
- C++ standard library, e.g. I/O, container
- A note about AI

# Recitation objectives



Learn how to write good systems programs in C++

(demo <https://github.com/aliciayuting/CS4414Demo.git> )



Poll Everywhere during recitations ( login via <http://pollev.com/> )



Assignment introductions and explanations



Exam preparation and reviews

# CPP Reference

<https://en.cppreference.com/w/>

## Example

Demonstrates how to inform a program about where to find its input and where to write its results.

A possible invocation: `./convert table_in.dat table_out.dat`

Run this code

```
#include <cstdlib>
#include <iomanip>
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "argc == " << argc << '\n';

    for (int ndx{}; ndx != argc; ++ndx)
        std::cout << "argv[" << ndx << "] == " << std::quoted(argv[ndx]) << '\n';
    std::cout << "argv[" << argc << "] == "
        << static_cast<void*>(argv[argc]) << '\n';

    /* ... */

    return argc == 3 ? EXIT_SUCCESS : EXIT_FAILURE; // optional return value
}
```

Possible output:

```
argc == 3
argv[0] == "./convert"
argv[1] == "table_in.dat"
argv[2] == "table_out.dat"
argv[3] == 0
```

# | What is C++?

A federation of related languages, with four primary sublanguages

- ➔ • **C:** C++ is based on C, while offering approaches superior to C. Blocks, statements, processor, built-in data types, arrays, pointers, etc., all come from C
- **Object-Oriented C++:** “C with Classes”, classes including constructor, destructors, inheritance, virtual functions, etc.
- **Template C++:** generic programming language. Gives a template, define rules and pattern of computation, to be used across different classes.
- **STL(standard template library):** a special template library with conventions regarding containers, iterators, algorithms, and function objects

# C++ Built-in Types

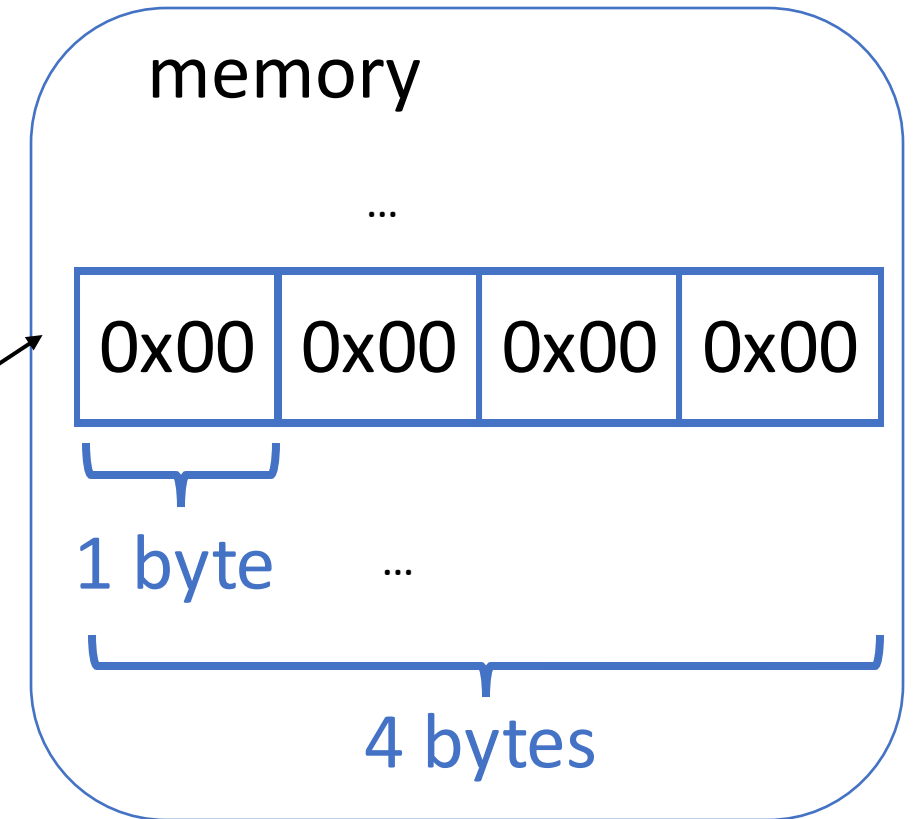
---

Brief refresher, we will have more detailed C introduction to revisit CS3410 in TA-led session

# & Address



```
int32_t x = 0;
```



0x00 is a pair of hex number  
(0x is the prefix, 00 is hex digits)

# & Address

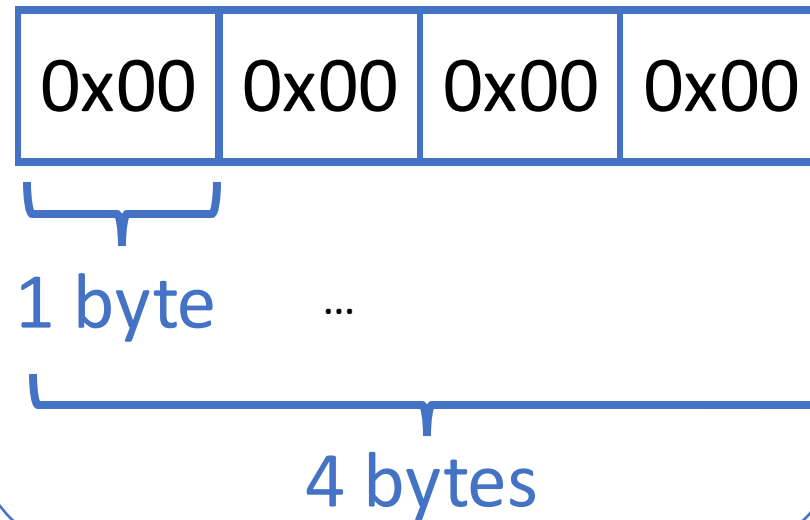


Where does x live  
in memory  
exactly?

int32\_t x =  
0;



memory



# & | Address



- Can obtain the address (represented in hex) with the **&** operator

```
int32_t x = 0;
```

```
std::cout << &x << std::endl;
```

```
// prints to the address of x  
for example, 0x7ffd55bdaa4
```

\*

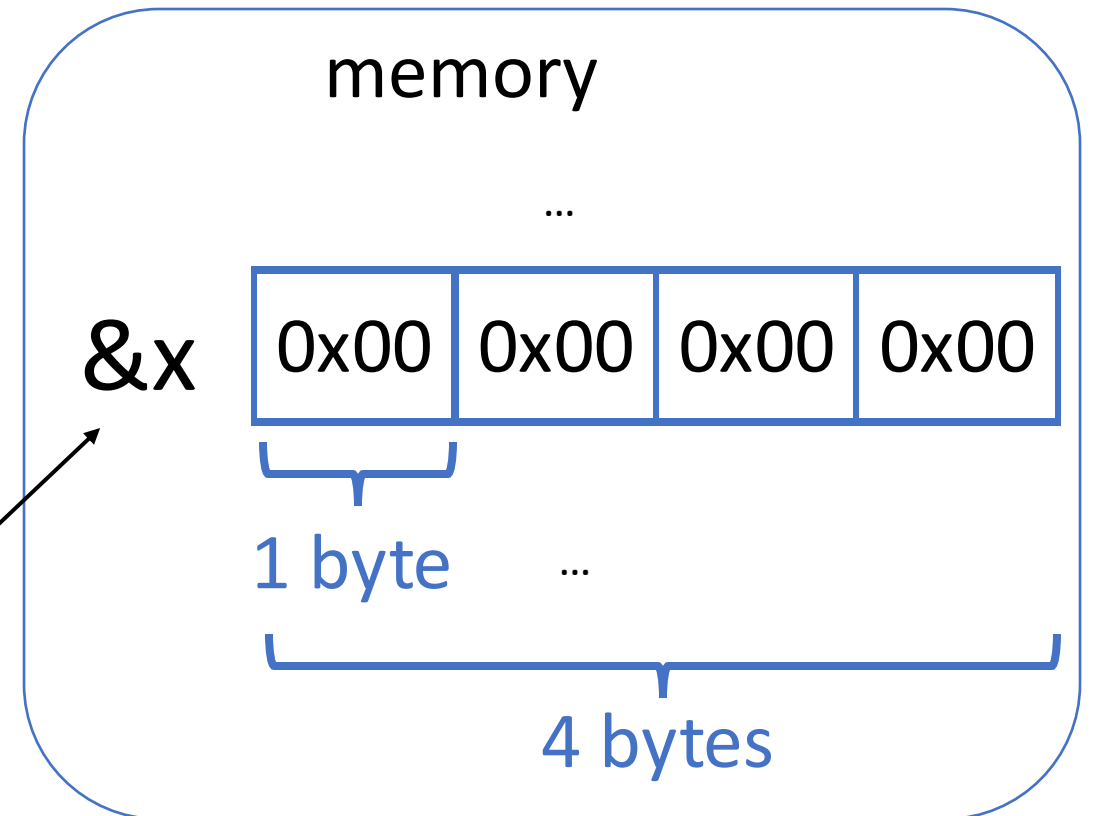
# Pointers

- A pointer is a variable that stores a memory address.

```
int32_t x = 0;
```

```
int32_t* px;
```

```
px = &x;
```



\*

## Pointers

- A pointer is a variable that stores a memory address.
- A pointer is declared just like a variable but with \* **after the type**

```
int32_t* px;
```



A pointer that could point to an integer

## \* | Pointers

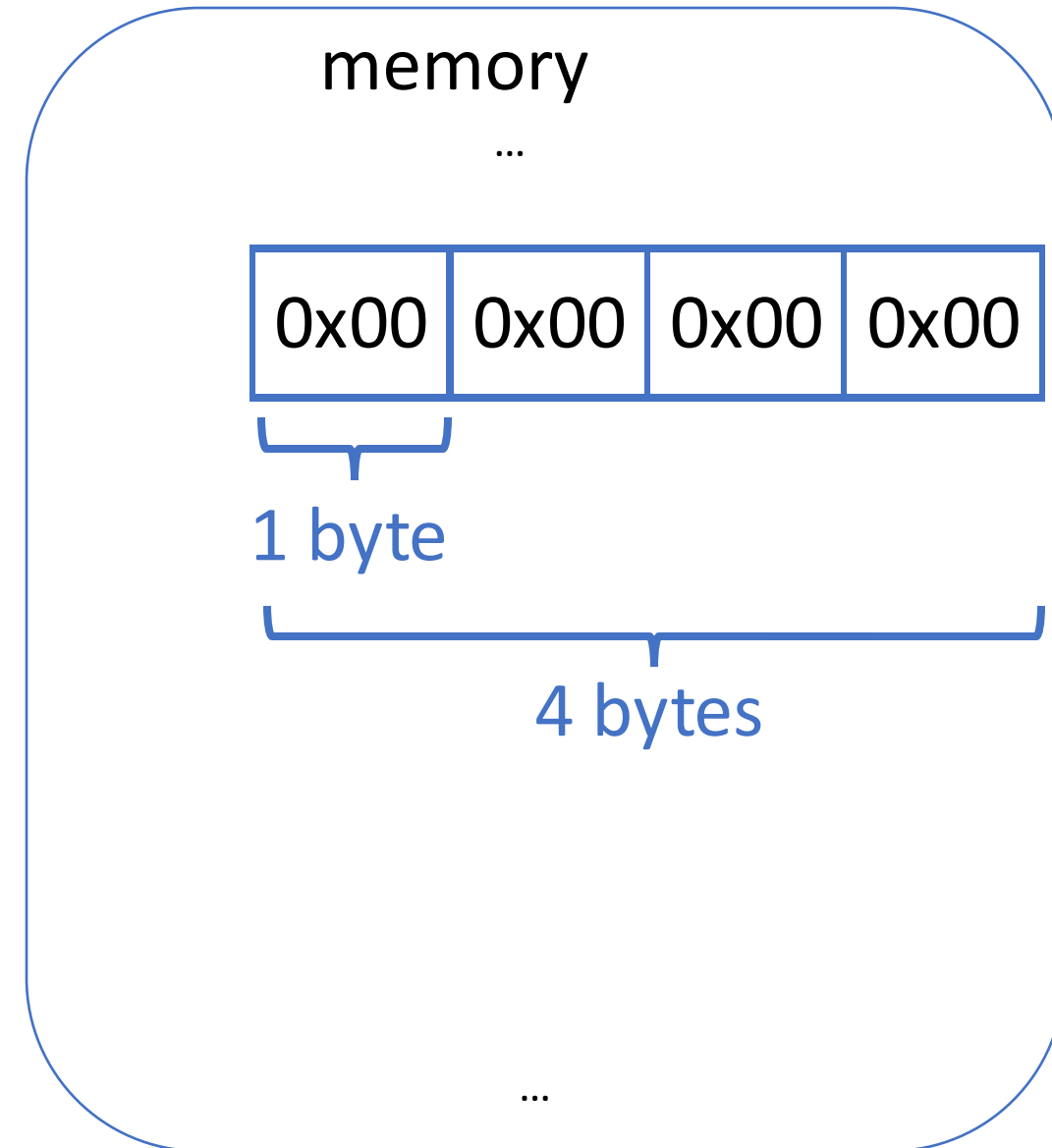
A pointer is a variable that stores a memory address.

```
int32_t x = 0;
```

```
int32_t* px;
```

```
px = &x;
```

// e.g. 0x7ffd39809084





## Pointers

- On **the same type of machines**, all pointers have **the same size**
  - e.g. sizes of `float*`, `int32_t*`, `char*`, `void*`, ... are the same on the same machine.
- Across **different machine architectures**, pointers' sizes may **differ**
  - 4 bytes on 32-bit machine
  - 8 bytes on 64-bit machine

\*

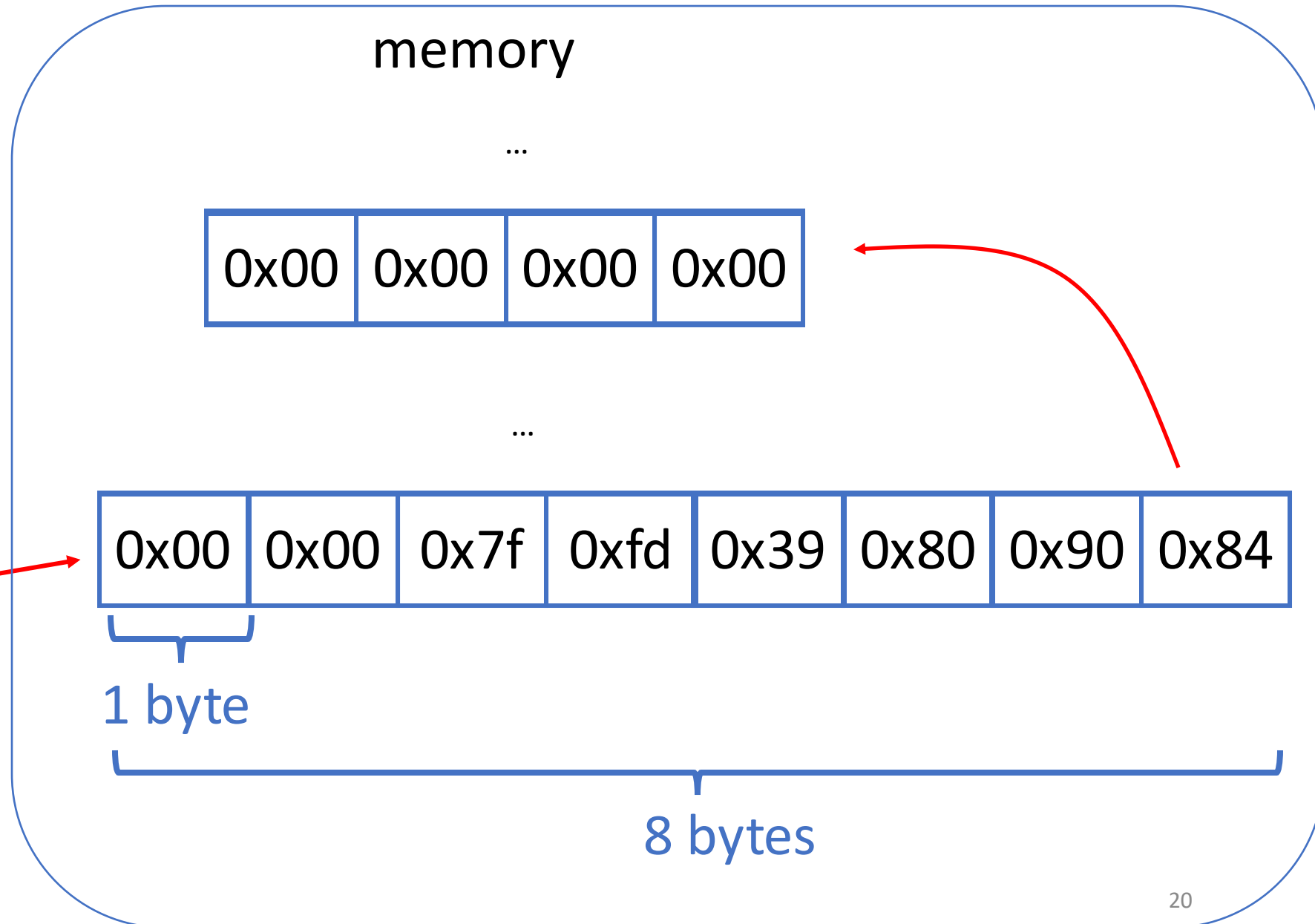
## Pointers

```
int32_t x = 0;
```

```
int32_t* px;
```

```
px = &x;
```

```
// e.g. 0x7ffd39809084
```



\*

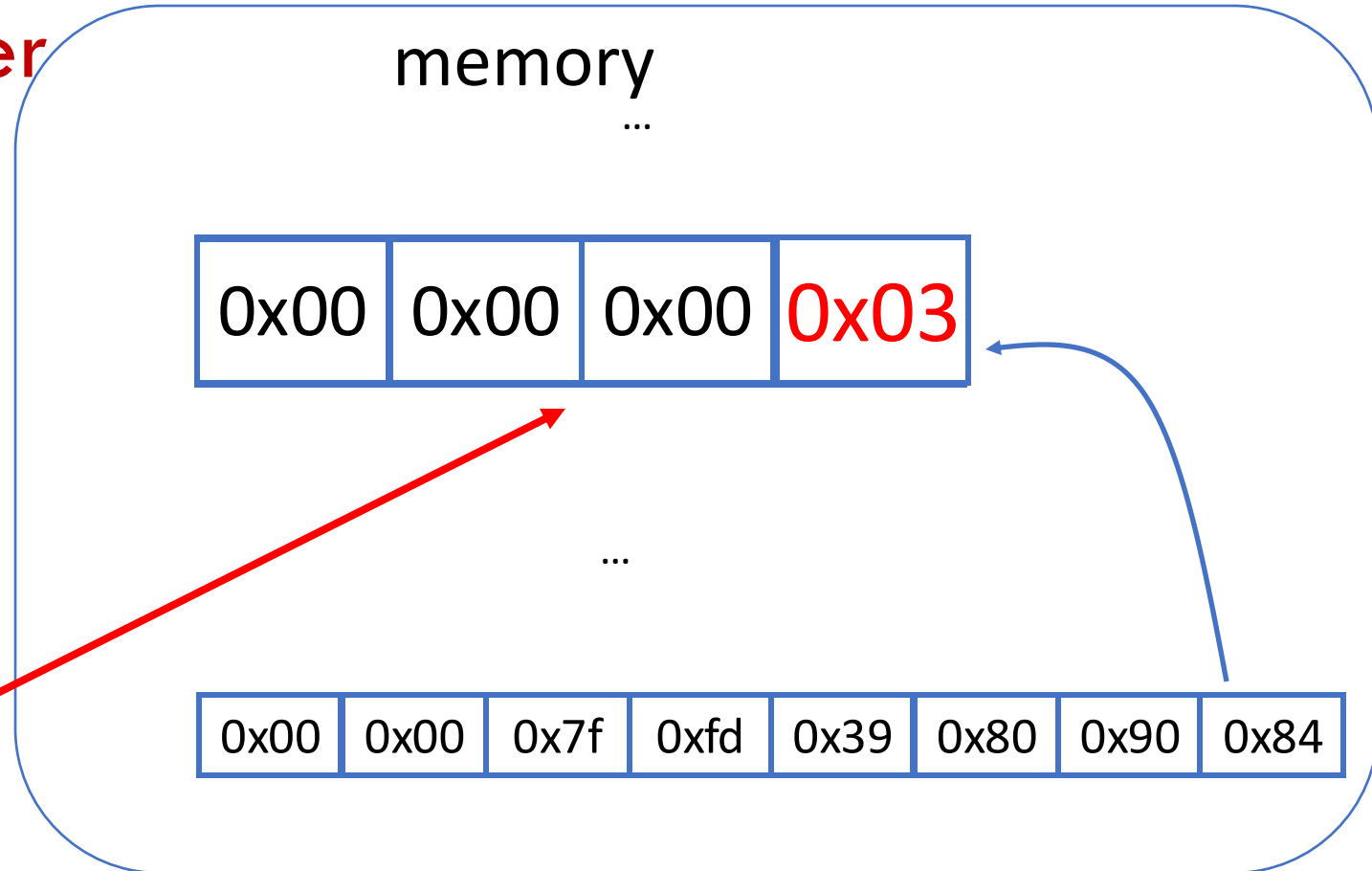
## Dereference a pointer

```
int32_t x = 0;
```

```
int32_t* px;
```

```
px = &x;
```

```
*px = 3;
```



# & Reference

Can I use a different name for object x?



• `int32_t x = 0;`



`int32_t& ref_x = x;`

memory

...

|      |      |      |      |
|------|------|------|------|
| 0x00 | 0x00 | 0x00 | 0x00 |
|------|------|------|------|

1 byte

4 bytes

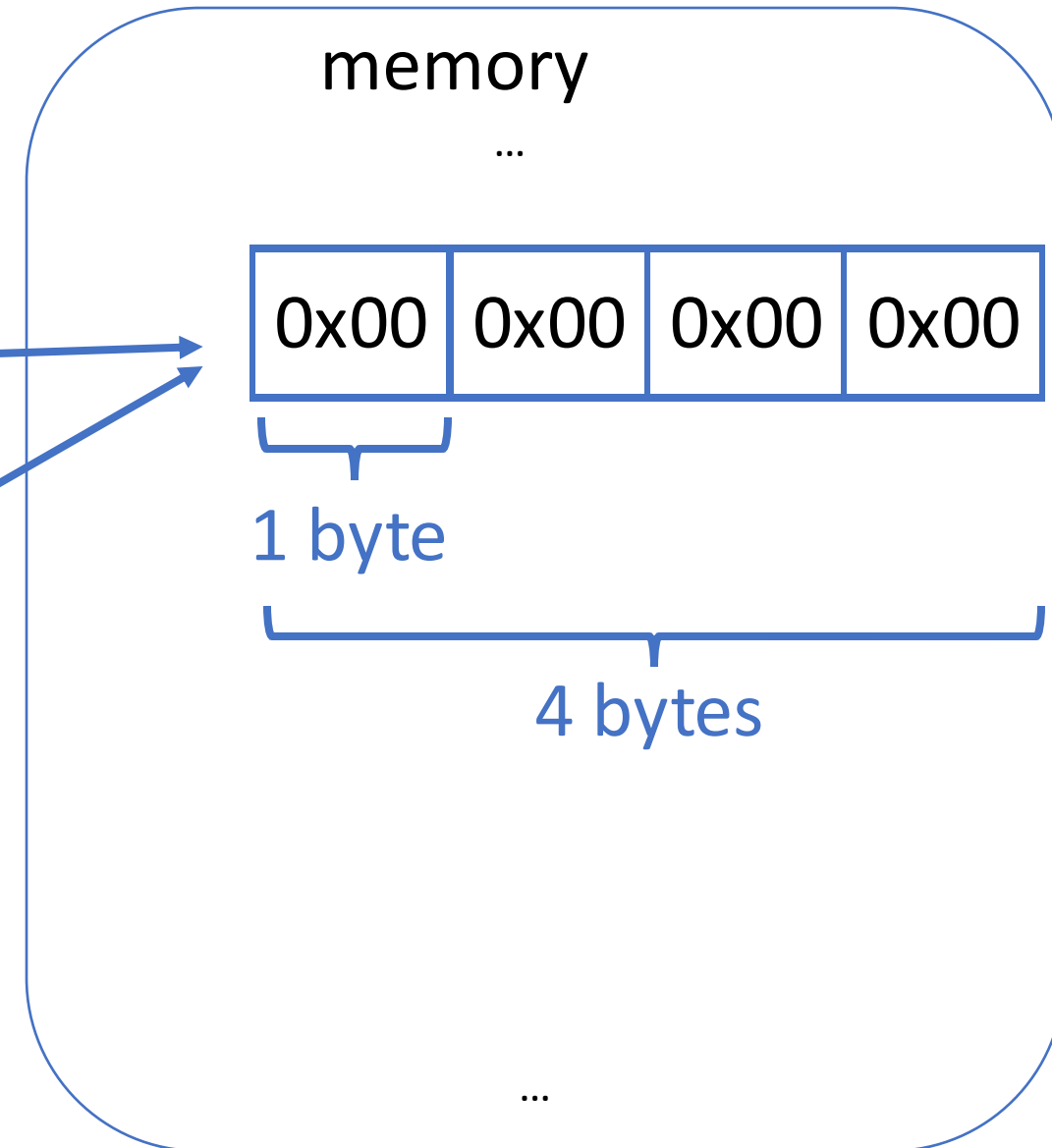
...

# & Reference

an **alias** to an **existing** variable

```
int32_t x = 0;
```

```
int32_t& ref_x = x;
```



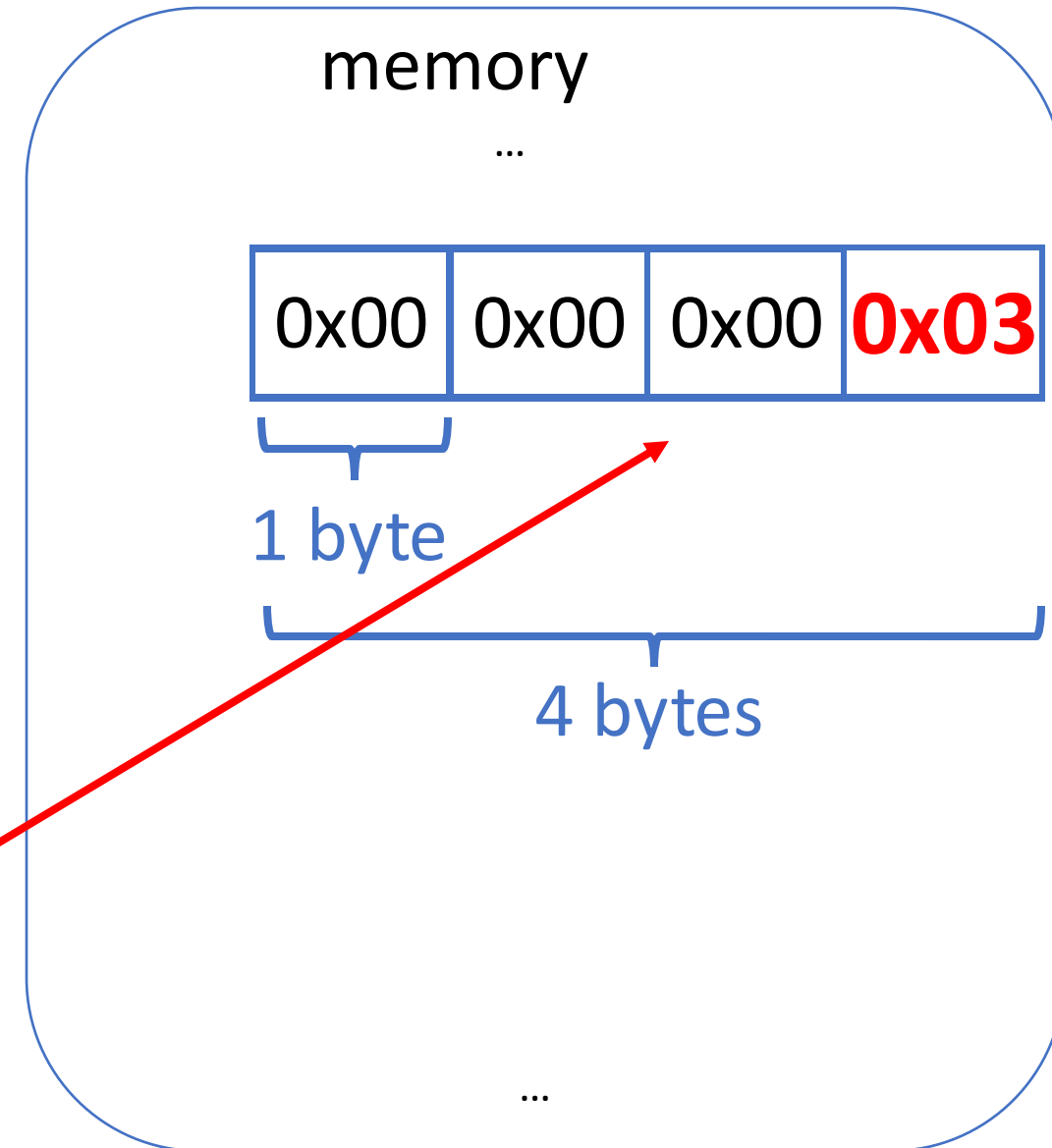
# & Reference

an **alias** to an **existing** variable

```
int32_t x = 0;
```

```
int32_t& ref_x = x;
```

```
ref_x = 3;
```



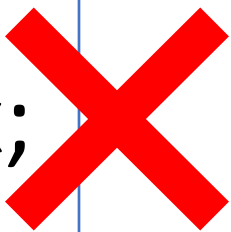
## & Reference

an **alias** to an **existing** variable


- Cannot be NULL
- Must be initialized at time of creation

```
int32_t x = 0;
int32_t& ref_x;
ref_x = x;
```

Compile error!



```
int32_t x = 0;
int32_t& ref_x = x;
```



```
int x = 0;
```

```
int y = 8;
```

```
int& ref = x;
```

```
ref = y;
```

```
ref = 3;
```

Now, what is x?  
What is y?

# & Reference

A reference is an **alias**(alternative name) to an **existing** variable

- **Permanently** bound to a **single** storage location, and **cannot** later be **rebound**

```
int x = 0;
```

```
int y = 8;
```

```
int& ref = x;
```

```
ref = y;
```

```
// initialize ref to reference variable x
```

```
// assign the value in y to ref
```

## Some easily confused notations

In a declaration,  
prefix with

In an expression,  
prefix with

`int a = 3;`

`*` = “pointer to”

`int* b = &a;`

`&` = “address of”

`&` = “reference to”

`int& c = a;`

`int d = *b;`

`*` = “contents of”

## Which program will error?

Option A:

```
int main() {  
    int a = 3;  
    int* p = &a;  
    *p = 7;  
    return 0;  
}
```

Option B:

```
int main() {  
    int* p;  
    *p = 7;  
    return 0;  
}
```

Option C:

```
int main() {  
    int* p = malloc(sizeof(int));  
    *p = 7;  
    free (p);  
    return 0;  
}
```

# Which program will error?

Option A:

```
int main() {  
    int a = 3;  
    //pointer to a  
    int* p = &a;  
    //assign contents of p (= 7)  
    *p = 7;  
    return 0;  
}
```

Option B:

```
int main() {  
    int* p;  
    // dereferencing an  
    // invalid address ->  
    // undefined behavior  
    // (often segfault)  
    *p = 7;  
    return 0;  
}
```

Option C:

```
int main() {  
    // allocate space for one int on the  
    // heap  
    int* p = malloc(sizeof(int));  
    // store 7 into that int  
    *p = 7;  
    // free memory  
    free (p);  
    return 0;  
}
```

# | What is C++?

A federation of related languages, with four primary sublanguages

- **C:** C++ is based on C, while offering approaches superior to C. Blocks, statements, processor, built-in data types, arrays, pointers, etc., all come from C
- ➔ • **STL(standard template library):** a special template library with conventions regarding containers, iterators, algorithms, and function objects
- **Object-Oriented C++:** “C with Classes”, classes including constructor, destructors, inheritance, virtual functions, etc.
- **Template C++:** generic programming language. Gives a template, define rules and pattern of computation, to be used across different classed.

# HelloWorld.cpp example

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello world!" << std::endl;  
    return 0;  
}
```

Program starting point  
Every C++ program must have  
exactly one main() function.

## Helloworld.cpp example

**#include <iostream>**

Instruct the compiler to include the declaration of the standard stream I/O facilities in iostream

```
int main() {  
    std::cout << "Hello world!" << std::endl;  
    return 0;  
}
```

## Helloworld.cpp example

```
#include <iostream>
```

std:: (standard library)  
specifies that the name cout to be found  
in the standard library namespace

```
int main() {
```

```
    std::cout << "Hello world!" << std::endl;
```

```
    return 0;
```

```
}
```

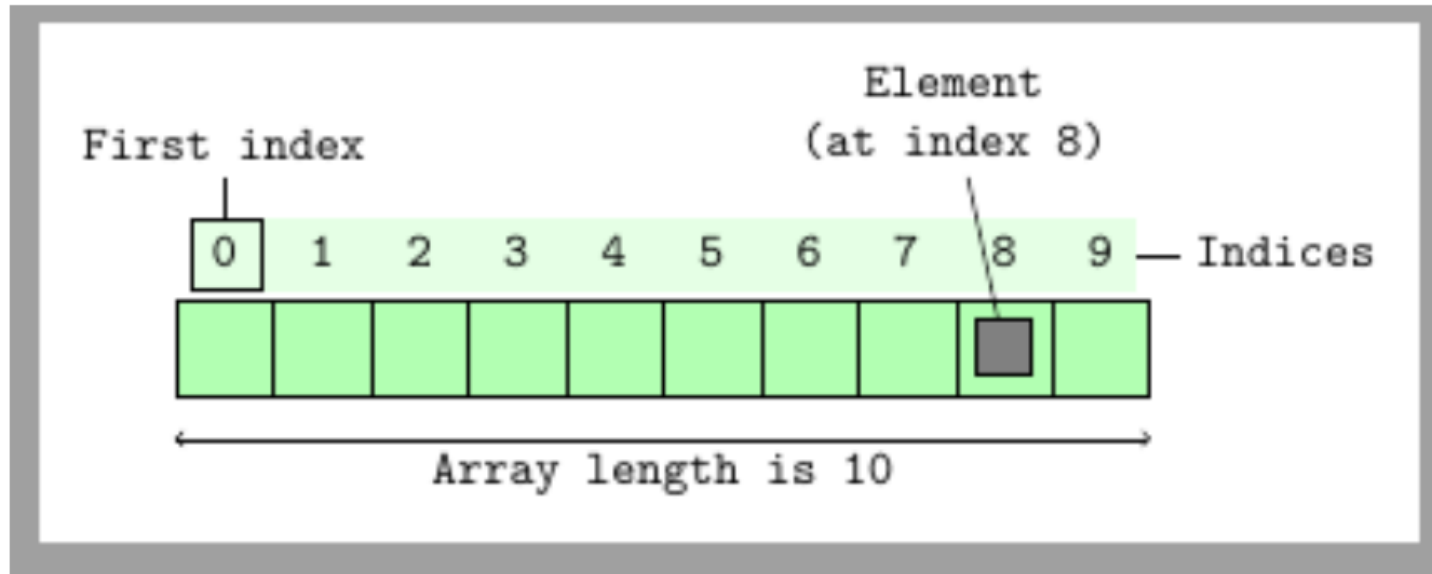
Operator << , writes its second argument to its first.  
(write "Hello world" to  
the standard output stream std::cout)

# C++ Containers

---



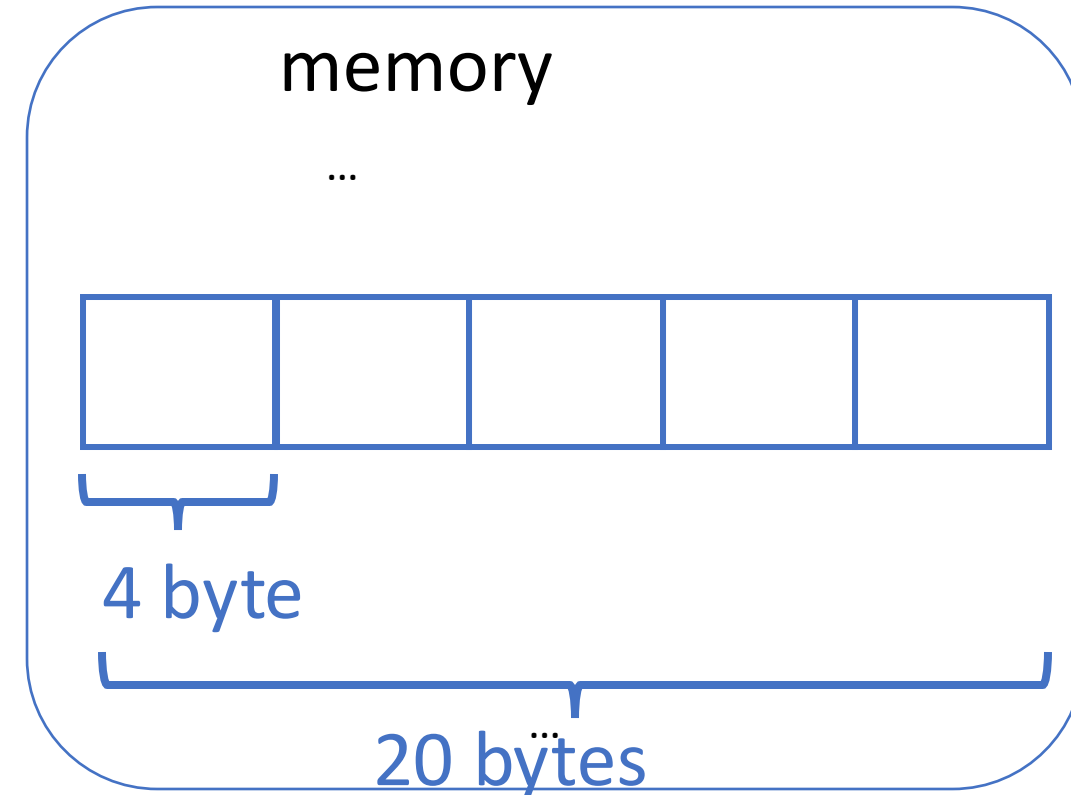
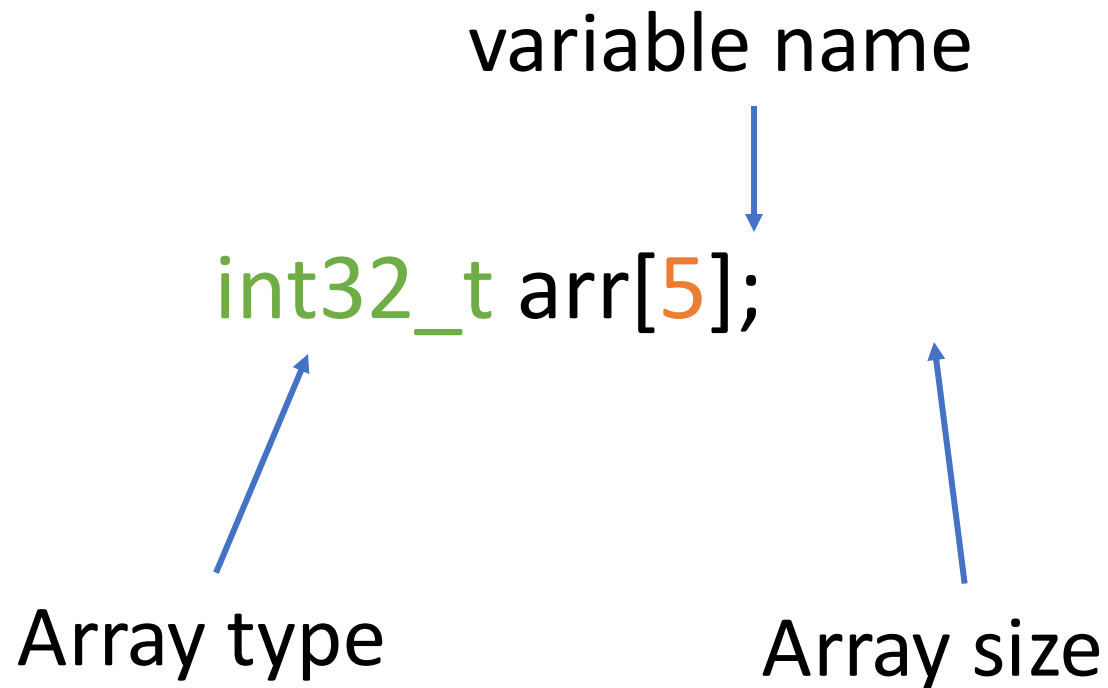
# C-style fixed-size Array



- Arrays must be declared by type and size
- The size must be fixed at compile-time
- Stores elements contiguously (in continuous memory locations)
- Elements are accessed starting with position 0 (0-based indexing)
- $O(1)$  access given the index of the element

# C-style fixed-size Array

- **Contiguously** allocated sequence of objects with the **same type**
- The array **size never changes** during the array lifetime.



## C-style fixed-size array -- Initialization

```
int32_t arr[5]={1,2,3,4,5};
```

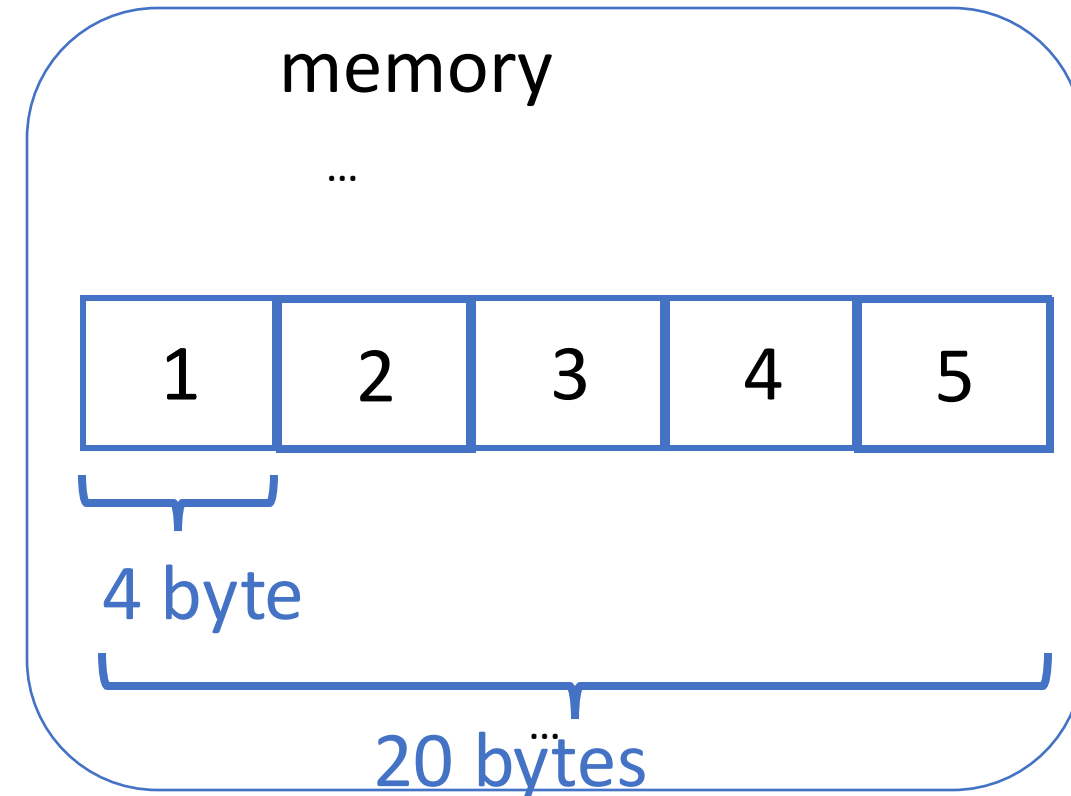
```
// declares int[5] initialized to {1,2,3,4,5};
```

```
int32_t arr[]={1,2,3,4,5};
```

```
// compiler could deduce the size of array is 5,  
and initialized to {1,2,3,4,5};
```

# C-style fixed-size array -- Indexing

```
int32_t arr[5];  
arr[0] = 1;  
arr[1] = 2;  
arr[2] = 3;  
arr[3] = 4;  
arr[4] = 5;
```



# C-style array pointer conversion and arithmetic

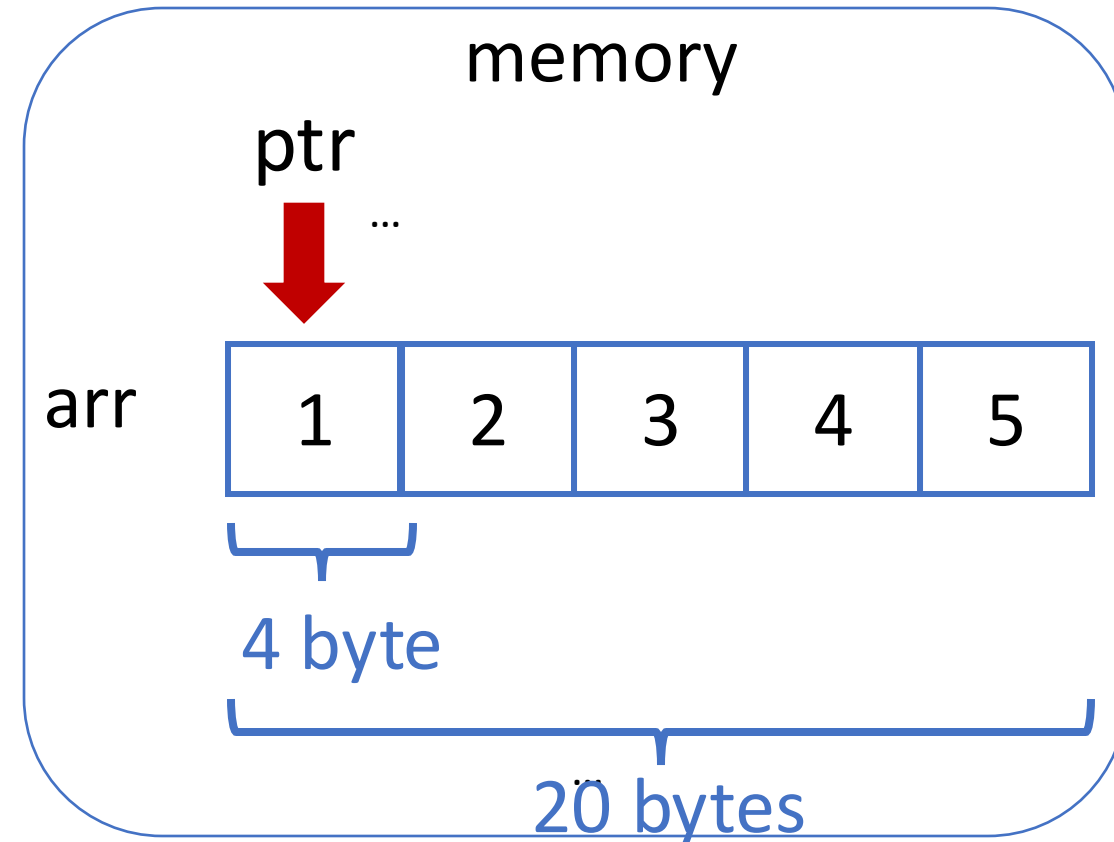
```
int32_t arr[5]={1,2,3,4,5};
```

```
int32_t* ptr = arr;
```

```
// ptr points to the address of arr[0]
```

```
for (int i=0; i<5; i++){  
    std::cout << *ptr << ",";  
    ptr++;  
}
```

```
// uint32_t pointer incremented by its  
type size
```

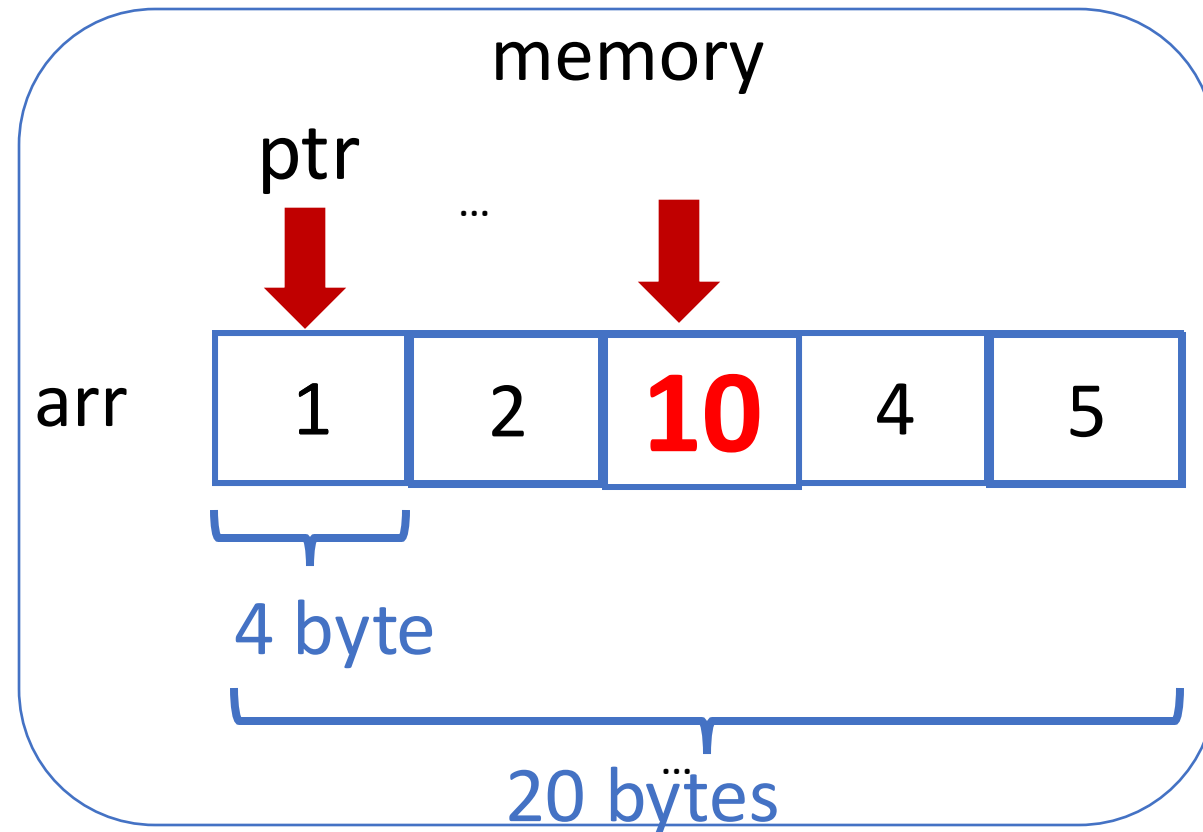


# C-style array pointer arithmetic

```
int32_t arr[5]={1,2,3,4,5};
```

```
int32_t* ptr = arr;
```

```
*(ptr + 2) = 10;
```



# C++ Container

## Standard Template Library

- Collection of classes and functions for general purpose use
- Provides container types (list, vector, map, ...), pair, tuple, string, thread and many other functionalities
- Available in the std namespace

# C++ Container

- A Container is an object used to **store other objects** and take care of the **management of the memory** of the objects it contains.
- Containers include many commonly used structure:
  - `std::array`,
  - `std::vector`,
  - `std::queues`,
  - `std::map`,
  - `std::set`,
  - ...

# C-style array (fixed-size array)

- C-style array is a block of memory that can be interpreted as an array

```
int a[10];
```

// declare **a** as an **array object** that consist of 10 contiguous allocated objects of type int

```
int a[3] = {1 , 3, 6} ;
```

// assignment of objects in array



`std::array<T, N>`

---a container that holds fixed size arrays

- Has the same semantics as a C-style array, but implemented by standard template library
- To use this container, include it at the beginning of the file

`#include <array>`

- T and N are template parameters: T is the type of the array, and N defines the number of elements
  - E.g., `std::array<char, 10>`, `std::array<int, 3>`

`std::array<T, N>`

---a container that holds fixed size arrays

- Has the same semantics as C++ arrays offered by standard template library
- To use this container

Why use `std::array`  
offered by C++  
Standard Template  
Library(std)?

`#include <array>`

- T and N are template parameters: T is the type of the array, and N is the number of elements

g., `std::array<char, 10>`, `std::array<int, 3>`



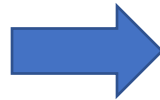
# C-style array vs. `std::array<T, N>`

- C-style array
  - No bound check when accessing element using operator[]
    - Undefined result if access `a[20]` if `a` is an array with size 3
- Array-to-pointer decay
  - E.g., When pass a C-style array as **a value** to a function it decays to **a pointer** of the first element in the array, losing the size information.

# C-style array vs. `std::array<T, N>`

- C-style array characteristics
  - No bound check when accessing element using operator[]
  - Array-to-pointer decay

```
void print_array(int arr[]){  
    size_t arr_size = sizeof(arr) / sizeof(int);  
    for(int i = 0; i < arr_size; ++ i){  
        std::cout << arr[i] << std::endl;  
    }  
}
```



```
void print_array(int * arr){  
    size_t arr_size = sizeof(arr) / sizeof(int);  
    for(int i = 0; i < arr_size; ++ i){  
        std::cout << arr[i] << std::endl;  
    }  
}
```

```
yy354@en-ci-cisugcl14:~/CS4414Demo/recitation2$ g++ -fstack-protector-all array_example.cpp -o arr  
array_example.cpp: In function 'void print_arr(int*)':  
array_example.cpp:11:34: warning: 'sizeof' on array function parameter 'arr' will return size of 'int*' [-Wsizeof-array-argument]  
   11 |         size_t arr_size = sizeof(arr) / sizeof(int);  
      |                             ^  
array_example.cpp:10:20: note: declared here  
   10 | void print_arr(int arr[]){  
      |
```

# C-style array vs. `std::array<T, N>`

`Std::array<T>` has more functions of standard container, makes it easier to use

```
std::array<int, 3> a = {1, 2, 3};
```

- `size()` : get the size of the array

```
std::cout << a.size() << std::endl;
```

- `at()` / operator `[]` : access specified element with bounds checking

```
std::cout << a.at(2) << std::endl;
```

- Use iterator to access container elements

```
for(auto it = a.begin(); it < a.end(); ++it )  
{....}
```

- More functionalities: <https://en.cppreference.com/w/cpp/container/array>

# `std::vector<T>`

- T is a template parameter
- `std::vector<int>` is a vector of integers, `std::vector<char>` is a vector of characters
- Same as `std::array`, T can be a class or other C++ container
  - E.g., `std::vector<Rectangle>`,

`std::vector<std::map<int, std::string>>...`

# `std::vector<T>`

- T is a template parameter

- `std::vector<int>` for a vector of integers  
`std::vector<char>` for a vector of characters

Why do I want to use `std::vector<T>` ?

- Same as `std::array`, T can be a class or other C++ container

e.g., `std::vector<Rectangle>`,

`std::vector<std::map<int, std::string>>...`



## `std::vector<T>` - A dynamically-sized array

- Main problem: How to support adding elements efficiently?
- Concept of size vs. capacity

# `std::vector<T>` - under the hood memory structure

```
void foo(){
```

```
→ std::vector<int> vect= {1,2,3};  
}
```

```
int main(){
```

```
    foo();
```

```
    .....
```

```
}
```

main()

foo()

capacity

size

foo()

std::vector<int>  
vect

int 1

int 2

int 3

...

Stack

Heap

data

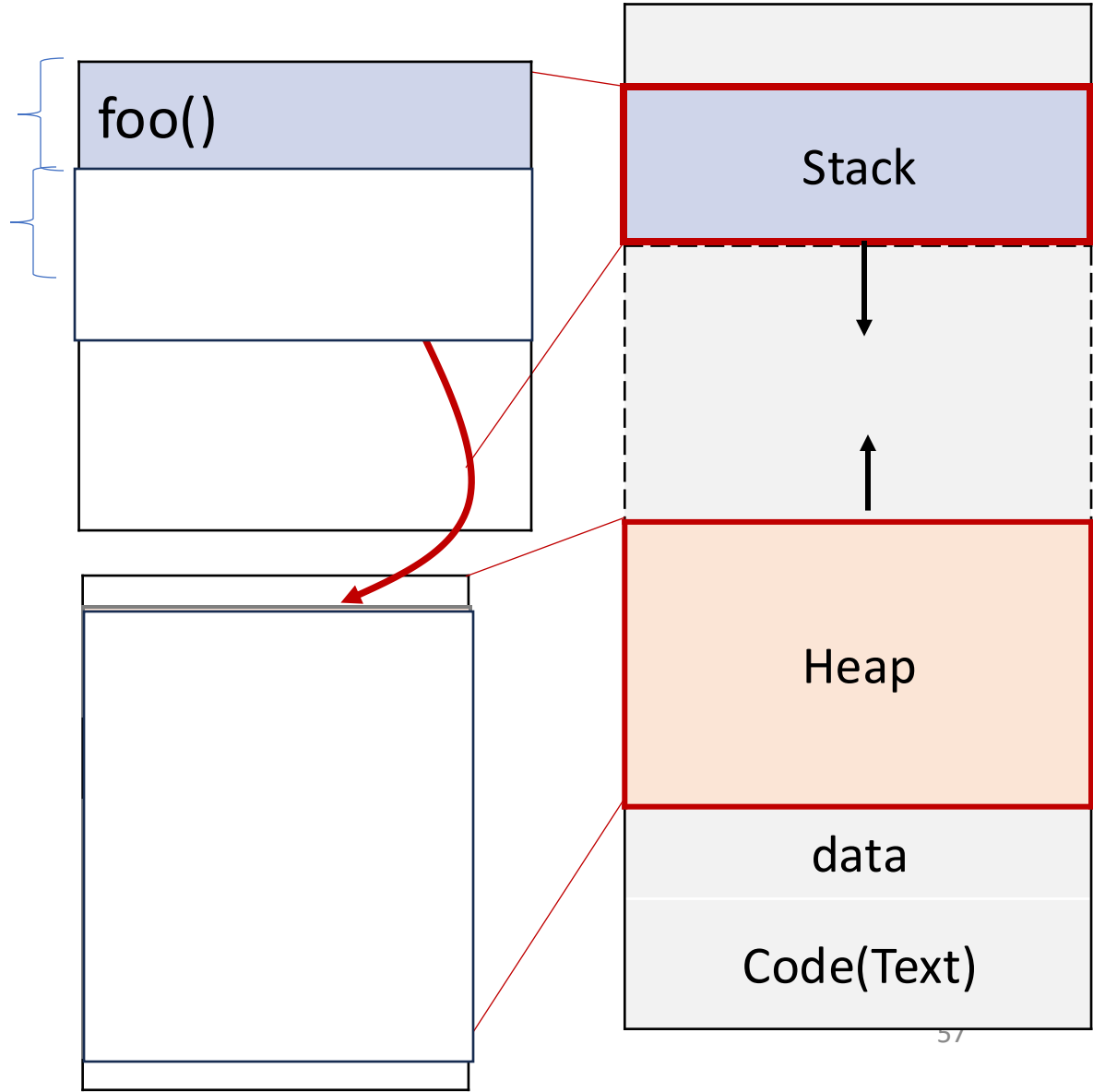
Code(Text)

# `std::vector<T>` - under the hood memory structure

```
void foo(){  
    std::vector<int> vect= {1,2,3};  
}
```

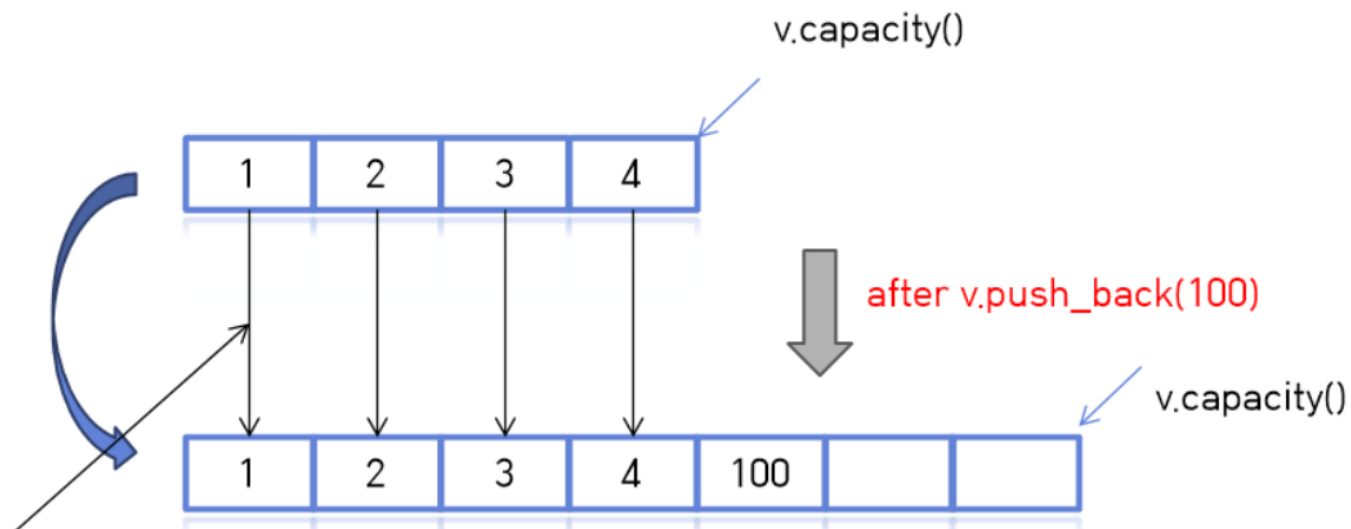
```
int main(){  
    foo();  
    .....  
}
```

main() {  
 foo()  
}



# `std::vector<T>` - A dynamic-sized array

- Main problem: How to support adding elements efficiently?
- Concept of size vs. capacity
- Reallocates elements when capacity is exceeded



# `std::vector<T>` - functionalities

- Element access: operator [], at, front, back, data
- Iterators: begin, end, rbegin, rend
- Capacity: size, capacity, reserve
- Modifiers: emplace, push\_back, erase, resize

# Building reliable and efficient systems

---



# System programming in the era of LLMs



**Andrej Karpathy**  @karpathy · Feb 2



There's a new kind of **coding** I call "**vibe coding**", where you fully give in to the vibes, embrace exponentials, and forget that the **code** even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper

[Show more](#)

 1.3K

 5.2K

 30K

 5.1M



<https://karpathy.ai>

# Ways you can use LLMs

- **As Learning Tools**
  - Reinforce course concepts through continuous querying
  - Ask for examples and verify by running them
  - Use tools like ChatGPT's study mode

## Ways you can use LLMs

- **To Understand the Codebase**

- Trace the call stack and function dependencies
- Use tools like Cursor to navigate and analyze large codebases

One needs fundamental system knowledge to use, understand and generate reliable code with LLM

Why are we taking 4414 now that we have?




## Ways you can use LLMs


- **Acting as a Project Manager to the LLM**
  - Clearly provide **context** about the problem
  - Ask **precise, well-scoped** questions
  - Apply **knowledge learnt in class** to query LLM
  - Check every line, **verify** its output
  - Generating smaller blocks of code at a time and making sure you understand it is better than generating an entire file or program















# Example. Writing efficient code

## Example. DNA Phylogenetic Tree

by Jeffrey Qian. (The top1 winning solution on leaderboard in 2024 Fall)

|   |  |
|---|--|
|  | IAAYTYAIIITAITTYAYTAIYYITITITAYIIAIAIAIYYAT      |
| Sprats  | AAYIYITATAITTTATIYAATIIYAYAYIAIYAIITAAAATIAAIIIT |

 *Green segments are scored. The distance algorithm is run recursively on the red segments. Yellow (matching) segments are discarded. The total distance is the sum of the green score and the recursive red score.*

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| Hairy Rock Snot   | Hallucigenia  | Jelly Belly   | Larval Treenymph  | Leaping Lizard  | Long-Snouted Squirk   | Munkles Mouse   |
|  |  |  |  |  |  |  |
| Nocturnal Mourningbird  | Nocturnal Plexum  | Paradise Rockfish   | Biscuit   | Pink Ziffer   | Poticle   | Pompous SnarkS  |
|  |  |  |  |  |  |  |

```
G15=ITYTYTAYIITAIYYITYAYTTAIITIAIIYTIAAIYIATTAYTIIITAYYAYYTTAYIATYTYTIIYTAAIIATYYTAIYYTYAYIIIAIYIYAAYIAI
AIYYATIIY
G16=IAAYIYIAATIIITTAIATITTYTIAAYAYTTAAIIIIYVIAIAIAAITIATAAAYYTYTYVAATYYIAIAITVAYITYAIYAIITYAAAYTIAAAYAY
YAAIATAAIYY
G17=IYYAATAATAITTTTAAIYYTIIAAYIYITATAITTTATIYAATIAIYAIIAAATIAAIIITAAAYTTYATATYATAYAYIIYATYTIIAAAYYATIIYTIY
YAYAYYIAAIY
G18=IIAATIIITTAIATITTYTIAAYAYYTTAAIIIIYVIAIAAAAYIIYVYVYIAAAITATAAAYYAAATYYIAAYITYAIYAIITYAAAYTIAAAYAYYAA
ITAYIYIATAAIYY
G19=IYYAATAATAITTTTAAIYYTIIAAYIYITATAITTTATIYAATIIYAYAYIAIYAIIAAATIAAIIITAAAYTTYATATYATAYAYIIYATYTIIAAAYYAA
TIYTIYAYAYYIAAIY
G20=IYYAATAITTTAYYATAITTTTAAIYYTIIAAYIYITATAITTTATIYAATIAIYAIIAAATIAAIIITAAAYTTYATATYATAYAYIIYATYTIIAAAYYAA
TIYTIYAYAYYIAAIY

S0=Armored Snapper: Genes [0, 1, 2, 3, 12, 13, 17]
S1=Asian Boxing Lobster: Genes [5, 6, 9, 10, 14, 16, 19]
S2=Ballards Hooting Crane: Genes [4, 7, 8, 11, 15, 18, 20]
```

# Example. Score computation

The right is a very naive implementation of the logic. One that generative AI might give you from a good starter prompt.

```
11 int simple_score_slow(const std::string s1, const std::string s2) {
12     int s1_count[4] = { 0, 0, 0, 0 }; // Index 0 = A, 1 = I, 2 = T, 3 = Y
13     int s2_count[4] = { 0, 0, 0, 0 }; // Index 0 = A, 1 = I, 2 = T, 3 = Y
14
15     for(int i = 0; i < s1.size(); i++) {
16         if(s1[i] == 'A') {
17             s1_count[0]++;
18         } else if(s1[i] == 'I') {
19             s1_count[1]++;
20         } else if(s1[i] == 'T') {
21             s1_count[2]++;
22         } else if(s1[i] == 'Y') {
23             s1_count[3]++;
24         }
25     }
26     for(int i = 0; i < s2.size(); i++) {
27         if(s2[i] == 'A') {
28             s2_count[0]++;
29         } else if(s2[i] == 'I') {
30             s2_count[1]++;
31         } else if(s2[i] == 'T') {
32             s2_count[2]++;
33         } else if(s2[i] == 'Y') {
34             s2_count[3]++;
35         }
36     }
37
38     int score = 0;
39     for(int i = 0; i < 4; i++) {
40         score += std::min(s1_count[i], s2_count[i]) * COMMON_COST;
41         score += std::max(s1_count[i], s2_count[i]) * DIFF_NUM_COST;
42     }
43     return score;
44 }
```

## Example. Score computation

What's wrong with it? Let's ask Claude

- Claude says the if-else chain causes branch predictions and proposes:
  - Direct mappings from character to array index
  - Switches statements
  - Using a hashmap
- This is where good fundamentals comes in → direct mappings is the fastest
  - Switch statements: to avoid branching costs, switch statements usually will “hash” the input and use that value as an index into a jump table [source](#). However, if the compiler is smart enough, it would perform similar optimizations with if-else statements.
  - Hashmaps are expected  $O(1)$  but the hash algorithm on characters is probably more expensive than a direct mapping.

| Benchmark                     | Time        | CPU         | Iterations | bytes_per_second |
|-------------------------------|-------------|-------------|------------|------------------|
| BM_SimpleScoreSlow/100        | 0.409 us    | 0.409 us    | 1732107    | 282.334/s        |
| BM_SimpleScoreSlow/1000       | 3.54 us     | 3.54 us     | 194825     | 2.83213Ki/s      |
| BM_SimpleScoreSlow/10000      | 94.6 us     | 94.5 us     | 6840       | 30.2006Ki/s      |
| BM_SimpleScoreSlow/100000     | 1122 us     | 1121 us     | 614        | 283.857Ki/s      |
| BM_SimpleScoreSlow/1000000    | 11310 us    | 11302 us    | 61         | 2.76666Mi/s      |
| BM_SimpleScoreSlow/10000000   | 115211 us   | 115059 us   | 6          | 27.6286Mi/s      |
| BM_SimpleScoreSlow/100000000  | 1151250 us  | 1150901 us  | 1          | 165.727Mi/s      |
| BM_SimpleScoreSlow/1000000000 | 13324972 us | 12594618 us | 1          | 151.442Mi/s      |

# Jeffrey spotted Claude missed something.....

What did Claude miss? It's missed simplest change!

```
int simple_score_slow(std::string s1, std::string s2) {
```

- Should replace with `const std::string& s1` or `const std::string_view& s1`
- We're not mutating the string inside the function, so there's no reason to pass by copy
- Can either pass by reference, or use `std::string_view` which is a non-owning read only view into a character buffer
- No reason to use one or the other in the assignment, but in practice, experience tells me to use `std::string_view` because it avoids a heap allocation in this case below. Again, **know the fundamentals really well**

```
46 void example(const std::string& s1) {...}  
47 example("this char* will be coerced into an std::string which is allocated on the heap")  
48
```

## What did that simple change get us?

| Benchmark                     | Time        | CPU         | Iterations | bytes_per_second |
|-------------------------------|-------------|-------------|------------|------------------|
| BM_SimpleScoreSlow/100        | 0.409 us    | 0.409 us    | 1732107    | 282.334/s        |
| BM_SimpleScoreSlow/1000       | 3.54 us     | 3.54 us     | 194825     | 2.83213Ki/s      |
| BM_SimpleScoreSlow/10000      | 94.6 us     | 94.5 us     | 6840       | 30.2006Ki/s      |
| BM_SimpleScoreSlow/100000     | 1122 us     | 1121 us     | 614        | 283.857Ki/s      |
| BM_SimpleScoreSlow/1000000    | 11310 us    | 11302 us    | 61         | 2.76666Mi/s      |
| BM_SimpleScoreSlow/10000000   | 115211 us   | 115059 us   | 6          | 27.6286Mi/s      |
| BM_SimpleScoreSlow/100000000  | 1151250 us  | 1150901 us  | 1          | 165.727Mi/s      |
| BM_SimpleScoreSlow/1000000000 | 13324972 us | 12594618 us | 1          | 151.442Mi/s      |

| Benchmark                    | Time      | CPU       | Iterations | bytes_per_second |
|------------------------------|-----------|-----------|------------|------------------|
| BM_SimpleScoreSlow/100       | 0.337 us  | 0.337 us  | 2094410    | 283.737/s        |
| BM_SimpleScoreSlow/1000      | 3.26 us   | 3.25 us   | 216431     | 2.77246Ki/s      |
| BM_SimpleScoreSlow/10000     | 34.0 us   | 33.8 us   | 21138      | 27.337Ki/s       |
| BM_SimpleScoreSlow/100000    | 345 us    | 343 us    | 2075       | 274.415Ki/s      |
| BM_SimpleScoreSlow/1000000   | 3389 us   | 3381 us   | 207        | 2.72548Mi/s      |
| BM_SimpleScoreSlow/10000000  | 34313 us  | 34160 us  | 21         | 26.5885Mi/s      |
| BM_SimpleScoreSlow/100000000 | 345099 us | 343878 us | 2          | 277.329Mi/s      |



# Let's implement what Claude told us now

- Jeffrey made everything compile time
- Used range based for loops
- Pragma gcc unroll

```
47 inline int simple_score(const std::string_view& s1, const std::string_view& s2) {
48     // static ensures the initial setup is made once
49     static constexpr std::uint8_t lookup[128] = {
50         ['A'] = 0,
51         ['I'] = 1,
52         ['T'] = 2,
53         ['Y'] = 3
54     };
55
56     int s1_count[4] = { 0, 0, 0, 0 }; // Index 0 = A, 1 = I, 2 = T, 3 = Y
57     int s2_count[4] = { 0, 0, 0, 0 }; // Index 0 = A, 1 = I, 2 = T, 3 = Y
58
59     for(auto& o : s1) {
60         s1_count[lookup[static_cast<std::size_t>(o)]]++;
61     }
62
63     for(auto& o : s2) {
64         s2_count[lookup[static_cast<std::size_t>(o)]]++;
65     }
66
67     int score = 0;
68     #pragma GCC unroll 4
69     for(std::size_t i = 0; i < 4; i++) {
70         int smaller = std::min(s1_count[i], s2_count[i]);
71         int larger = std::max(s1_count[i], s2_count[i]);
72
73         // from testing this does not give any more performance than
74         // pure branching
75         // probably because multiplications are so expensive on the
76         // hardware level
77         score += smaller * COMMON_COST;
78         score += ONLY_IN_ONE_COST * larger * (smaller == 0);
79         score += (larger - smaller) * DIFF_NUM_COST * (smaller != 0);
80     }
81
82     return score;
83 }
```

## Jeffrey's winning solution

| Benchmark                     | Time        | CPU         | Iterations | bytes_per_second |
|-------------------------------|-------------|-------------|------------|------------------|
| BM_SimpleScoreSlow/100        | 0.409 us    | 0.409 us    | 1732107    | 282.334/s        |
| BM_SimpleScoreSlow/1000       | 3.54 us     | 3.54 us     | 194825     | 2.83213Ki/s      |
| BM_SimpleScoreSlow/10000      | 94.6 us     | 94.5 us     | 6840       | 30.2006Ki/s      |
| BM_SimpleScoreSlow/100000     | 1122 us     | 1121 us     | 614        | 283.857Ki/s      |
| BM_SimpleScoreSlow/1000000    | 11310 us    | 11302 us    | 61         | 2.76666Mi/s      |
| BM_SimpleScoreSlow/10000000   | 115211 us   | 115059 us   | 6          | 27.6286Mi/s      |
| BM_SimpleScoreSlow/100000000  | 1151250 us  | 1150901 us  | 1          | 165.727Mi/s      |
| BM_SimpleScoreSlow/1000000000 | 13324972 us | 12594618 us | 1          | 151.442Mi/s      |

| Benchmark                 | Time      | CPU       | Iterations | bytes_per_second |
|---------------------------|-----------|-----------|------------|------------------|
| <u>BM_SimpleScore/100</u> | 0.158 us  | 0.158 us  | 4293293    | 295.287/s        |
| BM_SimpleScore/1000       | 1.63 us   | 1.63 us   | 432852     | 2.76878Ki/s      |
| BM_SimpleScore/10000      | 16.3 us   | 16.3 us   | 42766      | 27.9611Ki/s      |
| BM_SimpleScore/100000     | 165 us    | 165 us    | 4261       | 278.432Ki/s      |
| BM_SimpleScore/1000000    | 1646 us   | 1645 us   | 422        | 2.74734Mi/s      |
| BM_SimpleScore/10000000   | 16663 us  | 16649 us  | 42         | 27.2772Mi/s      |
| BM_SimpleScore/100000000  | 176594 us | 172440 us | 4          | 276.524Mi/s      |

# Situations to be careful when vibe coding

- Bloated files and codebases
- Always verify. Even though hallucinations have improved with each passing model release, it isn't 100% reliable
  - Even 99% accuracy over 100 code changes equals a 63% chance of a mistake
  - If you don't have the baseline knowledge to catch the mistakes, you may ship faulty code

# Situations to be careful when vibe coding

In 3 months, this hallucinated package got over 30k authentic downloads!

```
1 SELECT file.project, COUNT(*) AS num_downloads,
2 FROM `bigquery-public-data.pypi.file_downloads`
3 WHERE
4   file.project in ('huggingface-cli', 'blabladsa123')
5   AND DATE(timestamp)
6   BETWEEN DATE_TRUNC(DATE_SUB(CURRENT_DATE(), INTERVAL 3 MONTH), MONTH)
7   AND CURRENT_DATE()
8 GROUP BY file.project
```

Query results

| Row | project         | num_downloads |
|-----|-----------------|---------------|
| 1   | huggingface-cli | 31177         |
| 2   | blabladsa123    | 696           |

- <https://www.lasso.security/blog/ai-package-hallucinations>

## Rules

- If your assignment submissions include code that was entirely generated by AI, you must disclose this, and you will receive a maximum of a 70/100
- You *are* allowed to use AI as a learning tool

## Advice

- If you choose to use LLMs to assist you in the HWs, make sure that you do so in such a way where you are still learning the material thoroughly
- By the end of this class, you should feel confident that you could go back and complete similar HWs without AI

# References

## Vibe Coding

- How I use LLMs, Andrej Karpathy, <https://www.youtube.com/watch?v=EWvNQjAaOHw>
- Vibe Coding in prod by Claude, [https://www.youtube.com/watch?v=fHWFF\\_pnqDk](https://www.youtube.com/watch?v=fHWFF_pnqDk)

## C++

- A Tour of C++, Bjarne Stroustrup, 2<sup>nd</sup> edition
- Effective C++: 55 specific ways to improve your programs and designs, Scott Meyers, 3<sup>rd</sup> edition
- Large Scale C++, Process and Architecture, John Lakos, Volume 1
- GDB documentation: <https://www.sourceware.org/gdb/>
- <https://www.geeksforgeeks.org/gdb-step-by-step-introduction/>
- GDB quickstart tutorial: <https://web.eecs.umich.edu/~sugih/pointers/gdbQS.html>
- How does gdb work? <https://www.aosabook.org/en/gdb.html>