

# Pointer and Memory

Song Liu ([song.liu@bristol.ac.uk](mailto:song.liu@bristol.ac.uk))

GA 18, Fry Building,

Microsoft Teams (search "song liu").

# Previously

- What is Array?
  - How to
    - create an array? `int a[4];` or `int a[]={1,2,3};`
    - access an array? `a[0] = 1;`
  - How an array is stored in the computer memory?
    - consecutive blocks of memory
- How to pass an array as an input argument of a function?
  - Passing by reference
  - Not, passing by value

# Lab 3, Row Major Matrix

- Print a row major matrix  $A$  stored in an array.

```
int A[6] = {1,2,3,4,5,6};
for(int i = 0; i < 3; i++){
    for(int j = 0; j < 2; j++){
        // printf("*");
        // what should I put here?
    }
    printf("\n");
}
```

- What is the array element index for  $A_{k,l}$ ?
- (1,1) -> 0
- (1,2) -> 1
- (2,1) -> 2
- See a pattern?

# Today's Agenda

- **What is a pointer?**
- How to declare and initialize a pointer?
- What are "address of" and "dereference" operators?
- Array and Pointer

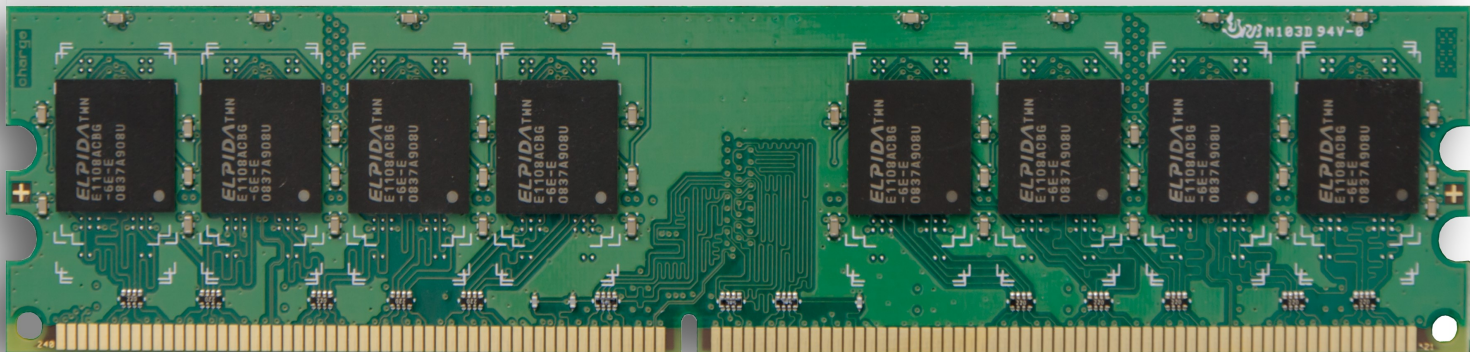
# A Double-Edged Sword

C is different from many other programming languages, it gives you the ability to directly read/write memory that is allocated to your program.

- Make good use of this feature, your code can run much more efficiently than programming in other languages.
- Abuse this feature, your code can become buggy, unreadable and unpredictable.

# Physical Memory and Virtual Memory

- Modern computers use **Random Access Memory (RAM)** to temporarily store information being used by the CPU. RAM is called the "Physical Memory" of a computer.
  - It stores machine code of programs and their data in small "cells" that are made of **MOSFET**.

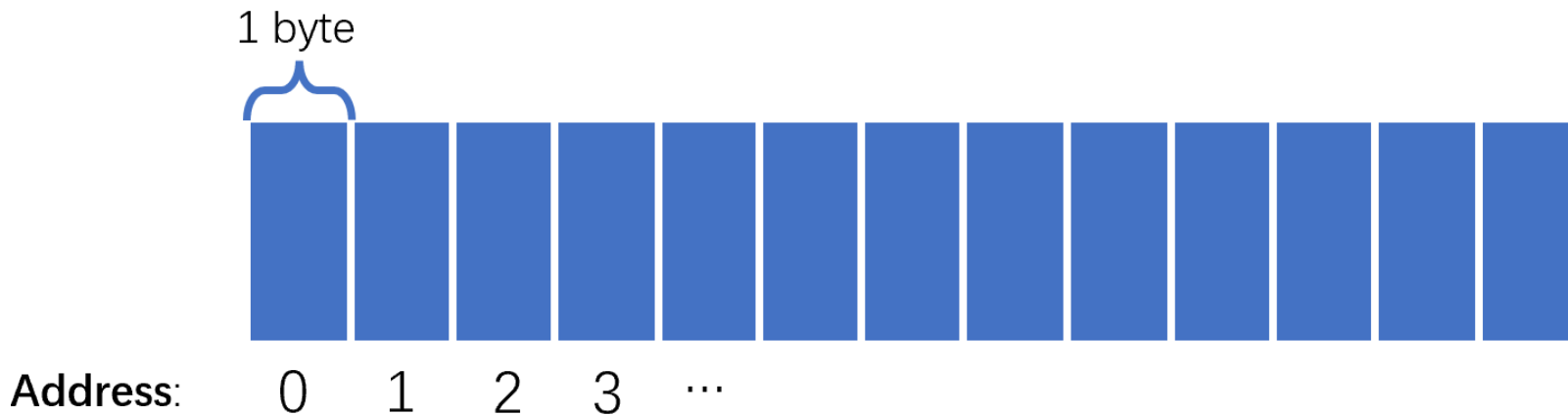


# Physical Memory and Virtual Memory

- Physical memory is a precious computational resource thus should be carefully rationed and managed. Poor management of physical memory would lead to errors and severe performance degradation over time.
- Luckily, OS manages physical memory for you so you do not have to.
  - Physical memory is not visible to your program for security reasons anyway.
- Your program can only see "**Virtual Memory**".

# Physical Memory and Virtual Memory

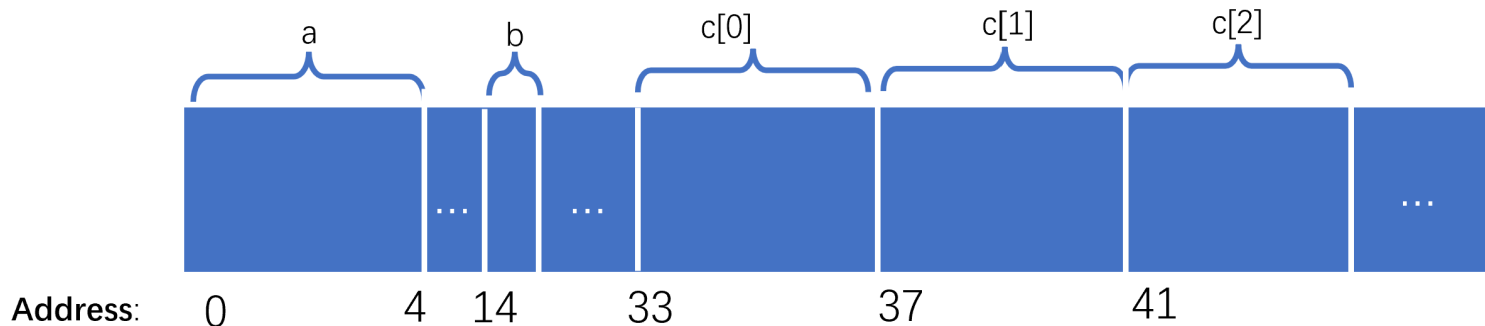
- **Virtual Memory** is an **abstraction** of the physical memory, made available to your program by the OS.
- You can think of it as a huge library shelf with many small slots next to each other. Each slot is a smallest memory unit, usually a **byte** (8 binary bits).
- You can refer to a byte using its index. The index is called **the address** of the byte.





# Multi-byte Variables

- However, we know some of the data types in C occupies more than one byte. For example, `int` occupies 4 bytes.
- These variables will occupy consecutive bytes in virtual memory.
- Elements in an array will also be stored consecutively in the virtual memory.
- Example: `int a; char b; int c[3];`



# Pointer

- Pointer is a variable that **stores the address of another variable in the virtual** memory.
- Using a pointer we can access the content stored in that memory location.
  - Like index card to a book in the library.

# Today's Agenda

- What is a pointer?
- **How to declare and initialize a pointer?**
- What are "address of" and "dereference" operators?
- Array and Pointer

# Declare and Initialize Pointer

- A pointer itself is a variable in C, thus should be declared before use.
  - Syntax: `data_type *var_name;`
  - For example,
    - `int *pa;` declares a `int` pointer
    - `double *pb;` declares a `double` pointer.
    - `int *pa = &a;` initialize `pa` with the address of `a`.
  - `&` is the "address of" operator. `&a` gives you the virtual memory address of `a`.

# & Operator

```
#include <stdio.h>
void main(){
    int a = 0;
    printf("The address of a is %p.\n", &a);
    // displays "The address of a
    // is <some memory address>".
}
```

- Use `%p` to print out a pointer.

## \* Operator

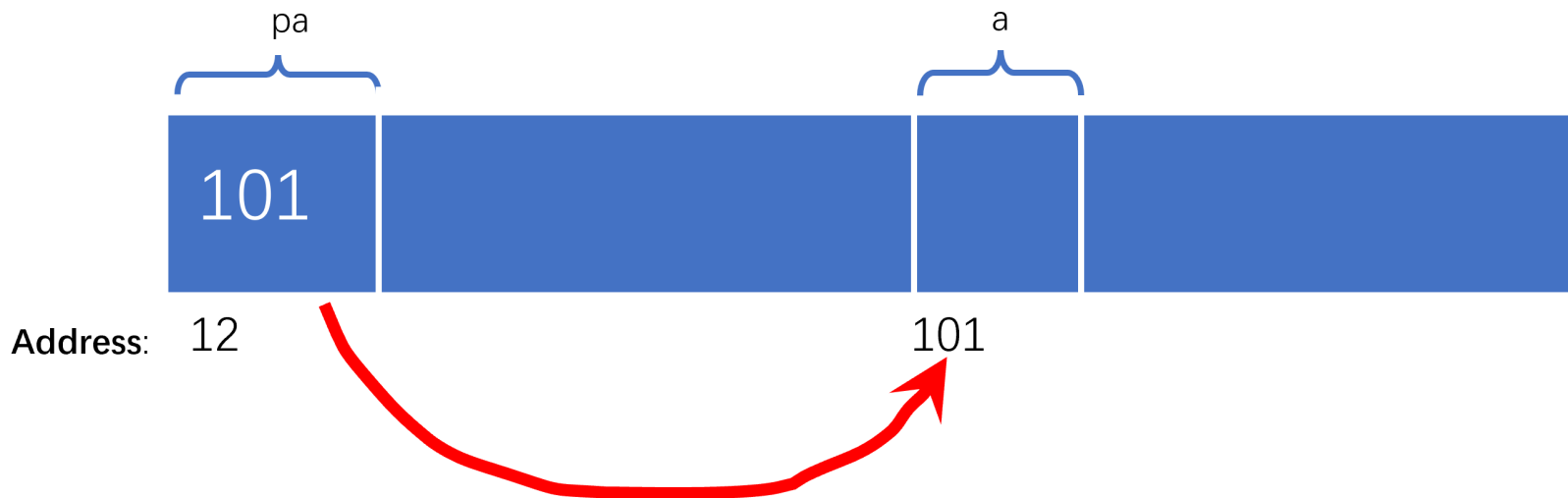
- \* is the opposite of &, called dereference operator.
- \* takes the value from a certain memory address.

```
#include <stdio.h>
void main(){
    //initialize the pointer to be the address of a.
    int a = 1;
    int *pa = &a;
    int b = *pa;
    printf("My value is %d.\n", b);
    // displays "My value is 1."
}
```

# Memory Diagram of Pointer

```
int a = 0;  
int *pa = &a;
```

- `pa` is an `int` pointer pointing to an `int` variable `a`.
- `pa` itself is also a variable and is stored in the memory.



# Pointer

- Without any initialization or assignment, the pointer points to a random memory location.

```
#include <stdio.h>
void main(){
    int *pa; //BAD!
    printf("I point to %p.\n", pa);
    // displays "I point to <some random memory space>".
}
```

- Trying to access memory in such random location will result in unpredictable behaviors!!!



# NULL Pointer

- It is dangerous to use an uninitialized pointer:
  - You may overwrite important information at some random memory address.
- If you do not know how to initialize a pointer, the convention is to initialize it as a NULL pointer.

```
int *pa = NULL;
```

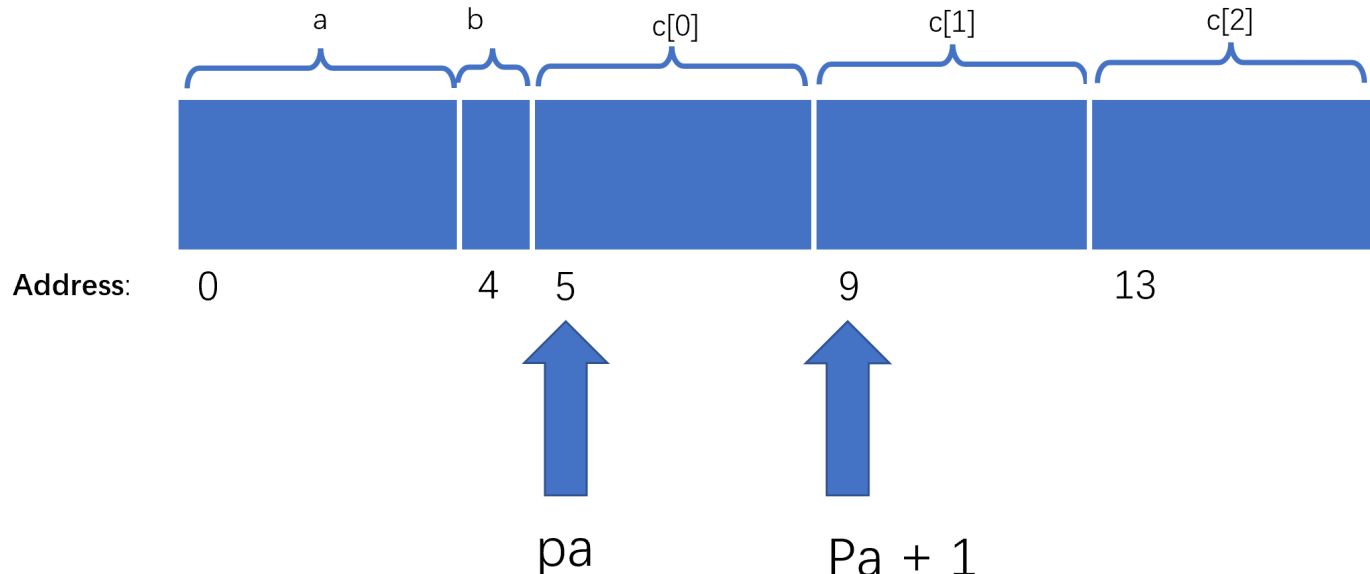
- NULL is a [preprocessor macro](#) and has a value zero.

```
printf("%d\n", NULL); // prints "0".
```

# Pointer arithmetic

- Compiler knows the type of variable a pointer points to.
- Adding "1" to a pointer would move the pointer by `x` bytes where `x` is the variable size in bytes.
- Consider the following memory layout for

○ `int a; char b; int c[] = {1,2,3};`



# Pointer arithmetic

```
#include <stdio.h>
void main(){
    int a = 0;
    int *pa = &a;
    int *pa_plus_one = pa + 1;

    printf("pa is %p.\n", pa);
    printf("pa + 1 is %p.\n", pa_plus_one);

    //pa is 000000d7785ffbfc.
    //pa + 1 is 000000d7785ffc00.
}
```

The outputs have a difference of 4.

# Today's Agenda

- What is a pointer?
- How to declare and initialize a pointer?
- What are "address of" and "dereference" operators?
- **Array and Pointer**

# Pointer and Array

Pointers are suitable for manipulating array variables.

- Arrays occupies contiguous bytes of memory.
- If `pa` points at the first element of an array, `pa+1` points at the 2nd element of the array...

```
#include <stdio.h>
void main(){
    int a[3] = {2,3,4}; int *pa = &a[0];

    printf("%p\n", pa); // the pointer points to the first element
    printf("%p\n", pa+1); // the pointer + 1

    printf("\n");

    printf("%p\n", &a[0]); // the address of the first element
    printf("%p\n", &a[1]); // the address of the 2nd element

    //the print outs are the same!
}
```

# Pointer and Array

- If `pa + 1` points to the `k+1` element in the array, then `*(pa+k)` is equivalent to `a[k]` ?
- YES!!

```
#include <stdio.h>
void main(){
    int a[3] = {2,3,4}; int *pa = &a[0];

    printf("%d\n", a[2]);
    printf("%d\n", *(pa+2));
    //prints out, 4, 4.
}
```

# Pointer and Array

In fact, in C, array name is a pointer pointing to the first element of the array!!

```
#include <stdio.h>
void main(){
    int a[3] = {2,3,4};
    int *pa = &a[0];
    printf("%p\n", a);
    printf("%p\n", pa);
    //prints out
    // 000000c0db3ffbec,
    // 000000c0db3ffbec, the same thing.
}
```

# Pass by Reference, Revisited

- Do you remember we mentioned that when arrays are passed as input arguments of a function, they are passed by reference rather than by value?
- When passing an array as an input argument, **the array name represents a pointer to the first element to the array so it is actually this pointer get passed to the function, rather than the array elements themselves.**
  - `swap(a)` , where `a` is the array name, AND a pointer to its first element.
  - This is why, arrays are passed by reference, not by value.



# Conclusions

- A pointer is a variable stores the address of another variable.
- A pointer needs to be declared before use.
- `&` "address of" operator takes the address from a variable.
- `*` "dereference" operators takes the content from an address.
- You can use a pointer to access an array.
  - The array name and the pointer pointing to the first element of the array is interchangeable.

# Homework 1.

1. Download the lab file and put them into your labpack.
2. Read `operators.c` and pay attention to the usage of `&` and `*`.
3. Complete the TODO tasks specified in the file.

## Homework 2.

1. Read `pointers_and_arrays_2.c` and run the code.
2. Explain why the code prints out such an outcome.
3. Complete the TODO tasks specified in the file.

## Homework 3.

1. Open `swap.c` .
2. Write a function `swap` , that takes two `int` **pointer** inputs.
3. `swap` function swaps the contents of variables which `pa` and `pb` point to.

## Homework 4.

1. Open `domainname.c` .
2. Read and run the code first. Answer questions in the code.
3. Write a function `getdomain` which extracts the domain name in an email address.
  - i. Domain name in email is the string after `@` .

You can leave the lab after you finish. However, please find TA/me to check your code before leaving!