Pointer and Memory (1)

Song Liu (song.liu@bristol.ac.uk)
GA 18, Fry Building,
Microsoft Teams (search "song liu").

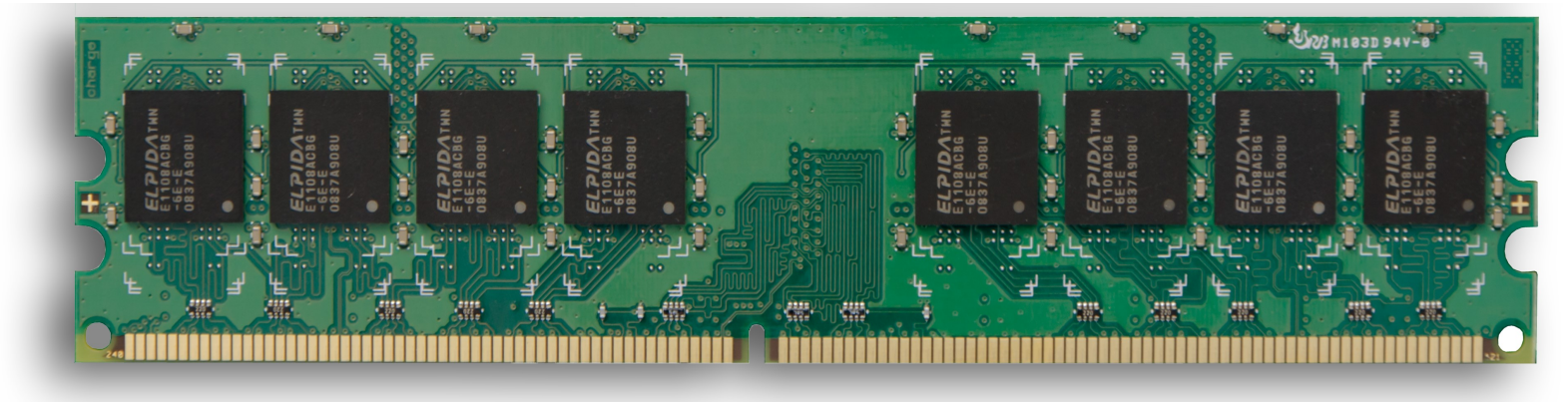
Highlight

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

- 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.

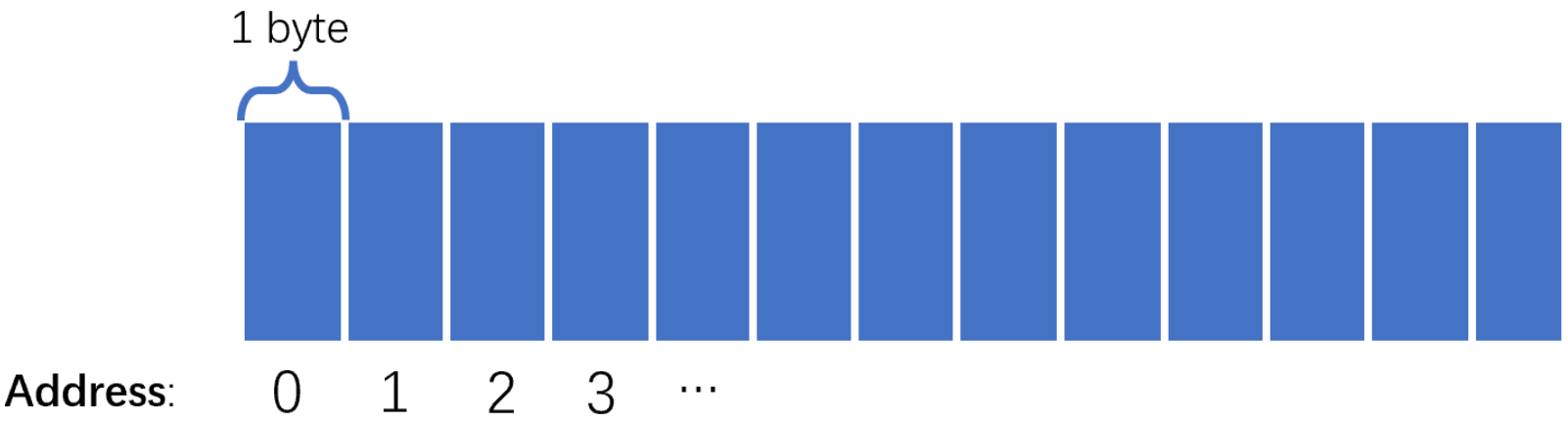


Physical Memory and Virtual Memory

- 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" that is allocated to it.

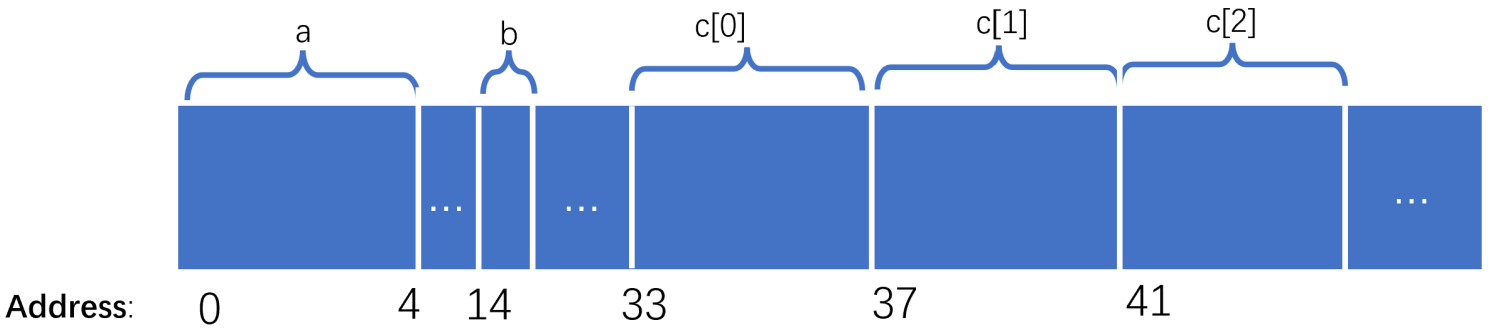
Physical Memory and Virtual Memory

- **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**.
- You can refer to a byte using its index. The index is called **the address** of the byte.



Multi-byte values

- 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.
- Different elements in an array will also be stored consecutively in the virtual memory.
- Example: `int a; char b; int c[3];`

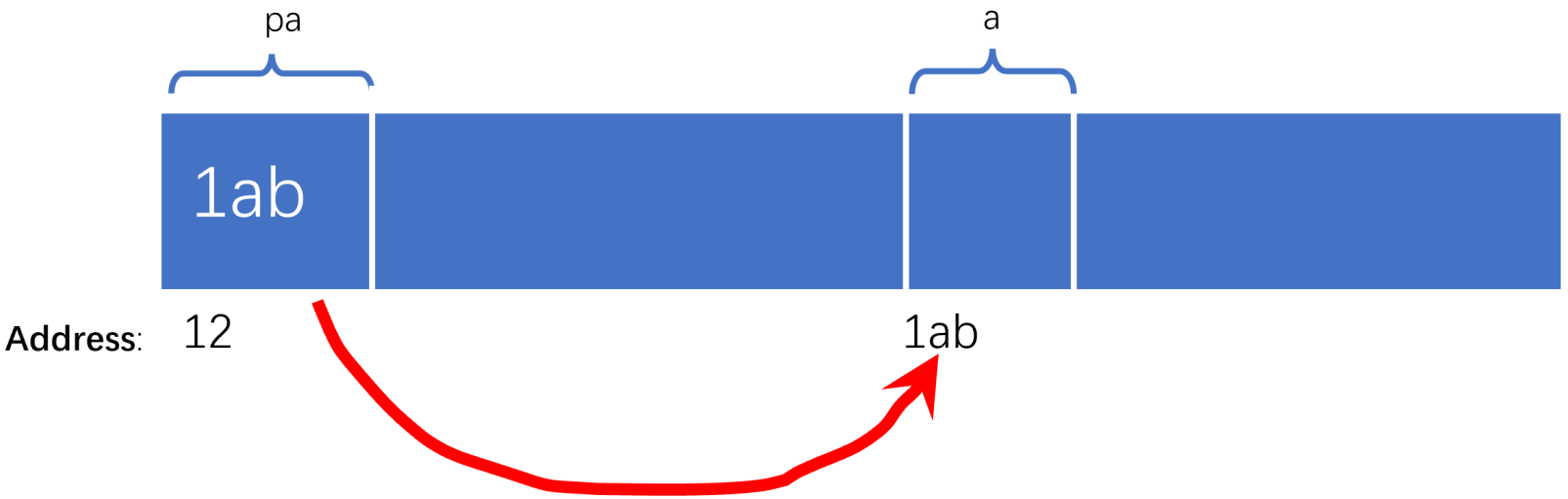


Pointer

- Pointer is the address of a variable stored in virtual memory. Using a pointer we can access the content stored in that memory location.
    - Like index card to a book in the library.
- Pointer itself is a variable in C, and should be declared before use.
    - Syntax: `data_type *var_name;`
    - For example, `int *pa;` declares a `int` pointer and `double *pb;` declares a `double` pointer.

Pointer

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



Pointer

- `size_of` operator returns the size of a data type. For example, the value of the expression `size_of(int)` is 4.
- `size_of(int *)` returns the size of a int pointer.

```c
#include <stdio.h>
void main(){
    printf("%d bytes.\n", sizeof(int *));
    // prints out "8 bytes."
    // The pointer type "int *" occupies
    // 8 bytes of 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!!!
- We will talk about a "fix" to this problem later.

Pointer Operators

- `&` takes the memory address of a variable.

```c
#include <stdio.h>
void main(){
    //initialize the pointer to be the address of a.
    int a = 1; int *pa = &a;
    printf("My address is %p.\n", pa);
    // displays "My address is 0000000a6f3ffa0c."
}
```

- `*` takes the value from a certain memory address.

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

`NULL` Pointer

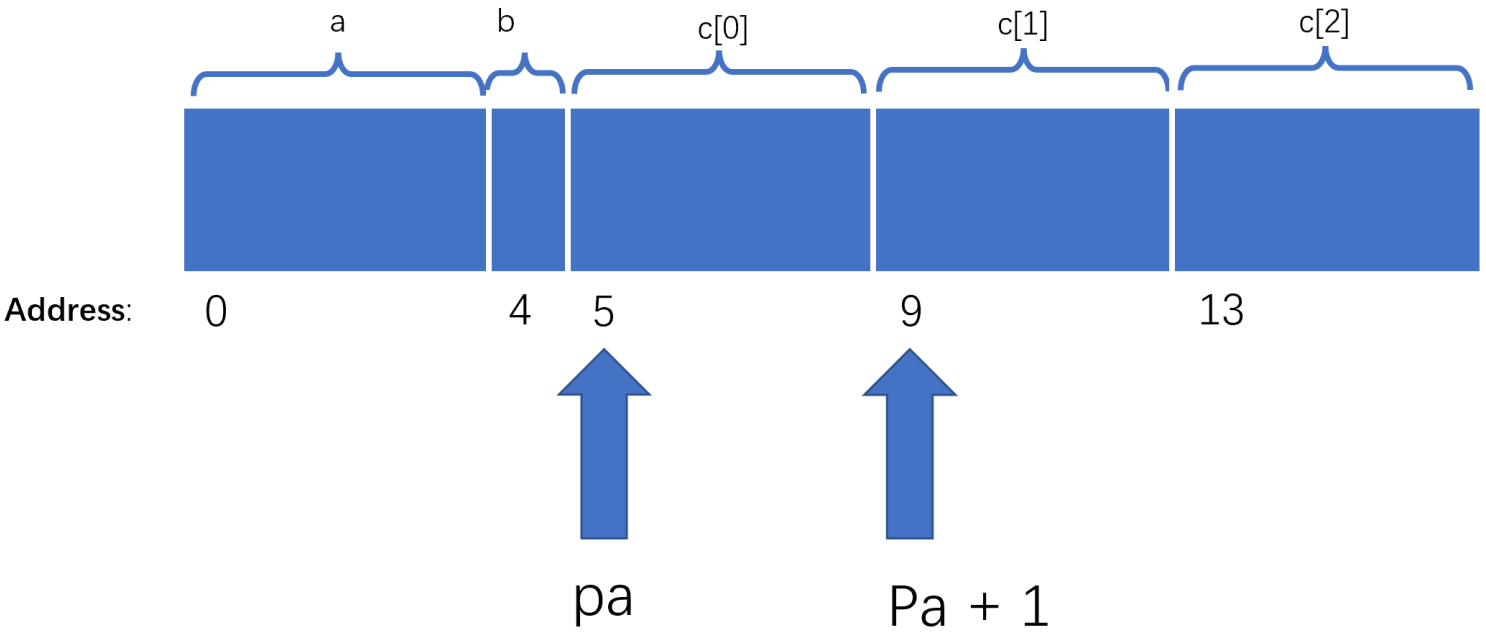
- It is dangerous to use an uninitialized pointer:
  - You may overwrite important information at some random memory address.
- If you do not know how to initialize a pointer, the convention is to initialize it as a `NULL` pointer.

  `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 `x` bytes where `x = sizeof(data_type)`.



Pointer arithmetic

```c
#include <stdio.h>
void main(){
    //declare a pointer pointing to the first element
    // of an array
    int a[3] = {2,3,4}; int *pa = &a[0];

    printf("%d\n", *pa);
    printf("%d\n", *(pa+1));
    // prints 2, 3.
}
```

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+k` points at the `k+1` -th element of the array.

In fact, the array name itself is a pointer which points at the first element in the array!

```c
#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.
    //You can use pa and a interchangeably!!
}
```

Pointer and Array

`a[k]` is equivalent to `*(a+k)` is equivalent to `pa[k]`.

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

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

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**.
    - `calc_length(3, 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.

Pass by Reference, Revisited

You can declare a function whose input argument is a pointer, then call it using an array variable name.

```
#include <stdio.h>
int sum(int len, int *pa){
// same as int sum(int len, int a[len])
    int s = 0;
    for(int i=0; i<len; i++){
        s += pa[i];
    }
    return s;
}
void main(){
    int a[3] = {2,3,4};
    printf("%d\n", sum(3, a));
    // prints 9.
}
```

Pass by Reference, Revisited

You can also write to an array using the pointer argument

```
#include <stdio.h>
void zero(int len, int *pa){
    for(int i=0; i<len; i++){
        pa[i] = 0;
    }
}
void main(){
    int a[3] = {2,3,4};
    zero(3, a);
    printf("%d %d %d\n", a[0], a[1], a[2]);
    // prints 0 0 0.
}
```

Return a Pointer

You may be tempted to return a pointer from a function, like this:

```
int *give_me_a_pointer(){
    int a = 2; int *pa = &a;
    return pa;
}
```

This won't work!

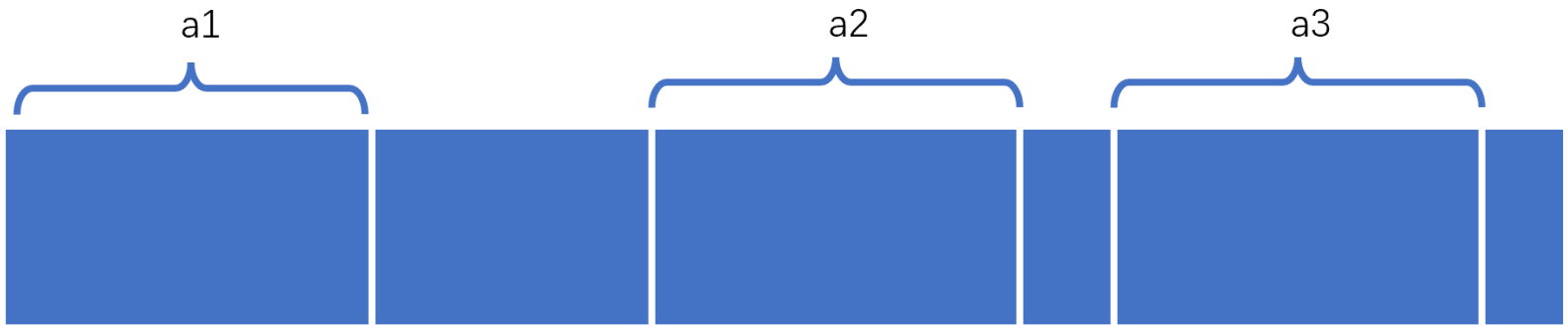
- `a` is a **local variable**, and the memory it occupies will be freed once the function is returned.
- After that, there is no way to know what value is stored at `a`'s old memory location.
- Therefore, `pa` only points to a memory address of some random value.

Return a Pointer

- However, you can return a pointer from a function in some special cases. We will talk about this in the next lecture.

Array of Pointers

- Consider arrays, `a1` , `a2` ... `aK` .
- They are scattered at different locations of memory.
- Can we write a loop and iterate over all the arrays?
    - No, you cannot. They are not stored at continuos memory addresses.



Array of Pointers

- You can create an array to store all the beginning addresses of arrays.
- Loop over the addresses in this array of pointers.

```
#include<stdio.h>
void main(){
int a1[] = {1,2,3}, a2[] = {4,5,6}, a3[] = {7,8,9};
int *aa[] = {a1, a2, a3};

for(int i = 0; i<3; i++){
    printf("%d %d %d\n", aa[i][0],aa[i][1],aa[i][2]);
}
}
// prints:
//1 2 3
//4 5 6
//7 8 9
```

Array of Pointers

`int *aa[] = {a1, a2, a3};` : Creating an array that stores pointers to the first elements of `a1, a2, a3` .

`aa[i][j]` : The `j` -th element of the `i` -th array.

Array of Pointers

- When a program is starting, OS sends the program command line arguments.
  `song$: /home/song/program.o "Hello" "World!"`
- You can read command line arguments sent to `main` from an array of pointers. Each pointer points to the initial `char` variable of an argument string.
  - The path to the program is the `0` th argument.
- In the above example, the array of argument strings is defined as:

```
char *args[] = {"/home/song/program.out", "Hello", "world!"};
```

Array of Pointers

- A program reads all the input arguments:

```
#include<stdio.h>
void main(int nargs, char *args[]){
for(int i = 0; i<nargs; i++>){
    printf("Argument %d: %s\n", i, args[i]);
}
}
//./lab3 1 2 3
Argument 0: /user/song/lab3
Argument 1: 1
Argument 2: 2
Argument 3: 3
```

Homework 1

Write a function which takes a `double` array (and its length) as inputs, output the **index** of the minimum value in the array.

Your function should look like

```
int find_minimum(int len, double array[len])
```

For example,

```
double vec[] = {0.2, 0.5, .3,.001,.1};
printf( "%d\n", find_minimum(5, array));
//prints out 3
```

- Assume there is no duplicate values in the array.
- Assume all elements in the array are in between [0, 1].

Homework 2 (Submit)

Write a function which takes a `double` array (and its length) as inputs, output the 3 smallest values in the array.

- Assume there is no duplicate values in the array.
- Assume all elements in the array are in between [0, 1].

Your function should look like

```
void find_bottom3(int len, double array[len], double bottom[3])
```

After the execution of `find_bottom3`, `bottom` array stores the three smallest values in the array.

Use the skeleton code/test cases provided in your lab pack.

Homework 3 (Submit)

- Declare three `int` variables `a1` and `a2` in the `main` function. Initialize them with random values.
- Write a function, so that after you calling it from the `main` function, `a1` will store the smaller value, `a2` will store the bigger value.
- You should NOT swap values of variables in the `main` function. You must call your function which swaps values of `a1` and `a2` for you.
- Hint: Consider passing by reference.

Homework 3 (Submit)

- Hint: If `pa` is a pointer to `a`, then `*pa = 1;` will replace the value stored in `a` with `1`. For example:

```c
#include <stdio.h>
void main(){
    int a = -9999;
    int *pa = &a;
    *pa = 1;
    printf("%d \n", a); // prints out "1"
}
```

Homework 3 (Submit)

Example code:

```c
#include <stdio.h>

// write your function here
//...

void main(){
    int a1 = 3, a2 = 2;

    //call your function here
    //you cannot swap values of a1 and a2 here.
    //you must call a function to swap values for you.

    printf("%d %d\n", a1, a2);
    // should display "2 3"
}
```