

Arrays

Song Liu (song.liu@bristol.ac.uk)

GA 18, Fry Building,

Microsoft Teams (search "song liu").

Previously

- Time Complexity
 - The number of elementary computing cycles
 - Number of for loop iterations (`prime1` , `prime2` , `prime3`)
- Recursion
 - Function calls it self
 - Deposit example
- Stack Memory Layout
 - The Ice cream analogy
 - The function being called is placed on the top of the stack.

Today's Agenda

- **What is Array?**
 - How to
 - create an array?
 - access an array?
- How an array is stored in the computer memory?
- Compilation Error vs. Runtime Error.
- How to handle matrix algebra using array?
- How to pass an array as an input argument of a function?
 - Passing by value vs Passing by reference

Vector Calculation

- Vectors are a sequence of numbers.
 - $\mathbf{a} = [1, 2, 3]$ is a three-dimensional vector.
- Vector calculations: addition, subtraction, dot product, etc.

$$\begin{array}{c} + \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array} \\ \\ = \\ \begin{array}{|c|c|c|} \hline 3 & 5 & 7 \\ \hline \end{array} \end{array}$$

Vector Addition

$$\begin{array}{c} \cdot \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array} \\ \\ = \\ \begin{array}{|c|} \hline 20 \\ \hline \end{array} \end{array}$$

Dot Product

Vector Calculation

- Consider the following program tries to compute the three dimensional vector addition
 - $c = a + b, a = [1, 2, 3], b = [2, 3, 4]$.

```
#include <stdio.h>
void main(){
    double a1=1.0, a2=2.0, a3=3.0;
    double b1=2.0, b2=3.0, b3=4.0;
    double c1, c2, c3;
    c1 = a1+b1; c2 = a2 + b2; c3= a3 + b3;
    ...
}
```

- This program is dumb.
- What if you have a 100-dimension vector?
 - `c1 = a1 + b1; ... c100 = a100 + b100; ?`

Vector Calculation

To automate vector calculations, we need two things:

- A way to represent an element in vector via an integer index.
 - e.g. `a[i]` represents the `i`-th element of vector `a`.
- A loop to perform calculations on all elements in a vector

Imagine we can do something like:

(pseudo code)

```
For i from 1 to 100  
    c[i] = a[i] + b[i]
```

This program is much more scalable and understandable.

Array

- Array is a fundamental **data structure** in C programming language that **stores a sequence of elements**.
- You can declare an array using the syntax:

- `data_type variable_name[array_size];` .

- ```
// declares an int array with 100 elements.
int a[100];
```

- `array_size` can **NOT** be a variable.

- ```
int c = 100;  
int b[c]; // compilation error!
```

- `array_size` must be determined **compilation time**.

Accessing Array

- i-th element in an array is referred to as `array_name[i]` .
- The first element in the array `a` is `a[0]` .
- The second element in the array `a` is `a[1]` .
- and so on. This is called **zero-based indexing**.
- e.g., `a[2] = 5;` assigns 5 to the third element of `a` .
- You can access elements using a loop:

```
int a[10];
for(int i = 0; i < 10; i = i+1){
    a[i] = 123; // assigning the i+1-th element
}
for(int i = 0; i < 10; i = i+1){
    printf("%d ", a[i]); // print the i+1-th element
}
```


Initialize Array

- Array declaration only reserves memory space for the array. The array will not be initialized automatically.
- If you do not initialize an array yourself, **it will contain rubbish value**, similar how variable declaration is handled in C.
- You can initialize an array using the syntax:
 - `data_type variable_name[] = {elements};` .
 - `int a[] = {1,2,3};`
 - No need to specify the `array_size` .

Vector Addition, Revisited

```
#include <stdio.h>
void main(){
    //declare and initialize array a and b.
    double a[] = {1.0, 2.0, 3.0},
           b[] = {2.0, 3.0, 4.0};
    double c[3];
    //addition
    for(int i = 0; i < 3; i=i+1){
        c[i] = a[i] + b[i];
    }
    //display each element in the array c
    for(int i = 0; i < 3; i=i+1){
        printf("%f\n", c[i]);
    }
}
```

Vector Addition, Revisited

- What is the time complexity of adding two d -element array?

Be Careful about the Length!

- What will happen if I try to access the element that does not exist?

```
#include <stdio.h>
void main(){
    //declare and initialize array a and b.
    double a[] = {1.0, 2.0, 3.0};

    printf("%d\n", a[5]);
    // It is clearly wrong but compiles!
    // ...
}
```

Be Careful about the Length!

- C compiler will **not check** if the index has exceeded the array length.
- If you try to access an array element that is clearly out of bound,
 - No error will be raised during the compilation stage.
 - However, your program will have undefined behavior (possibly a runtime error).
- There is no Out-of-Bound checking in C, unlike Python or R.

Compilation Error vs. Runtime Error

- Errors happen during compilation are called **compilation errors**.
- For example, you forget to add a semicolon after a statement:

```
printf("hello world!")
```

- The compiler will raise an error saying:

```
vecadd.c:11:26: error: expected ';' before '}' token
```

- The compiler will refuse to generate executable code if compilation error happens.
 - This kind of error is easy to identify and fix.

Compilation Error vs. Runtime Error

- Errors during the runtime are called **Runtime errors**.
- Your code compiles and generates an executable file.
- However, your program behaves **unexpectedly**.
- Examples of Runtime Error:
 - Division by zero
 - Loop that never ends (infinite loop)
 - Out of memory (stack overflow)
 - Index out of bounds (undefined behavior)
 - many more
- Much harder to spot and fix! Be careful about it!

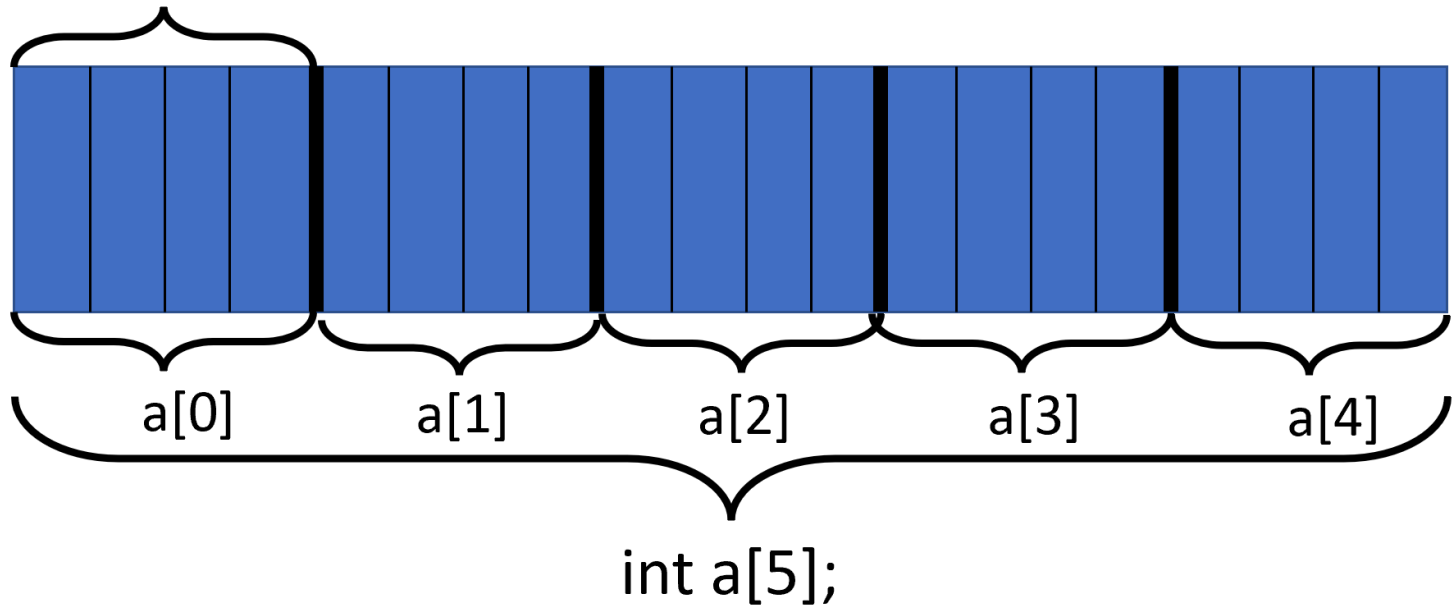
Good Habits Matters!

- Having good coding habit reduces runtime errors significantly.

Array's Memory Layout

- Array is stored in a **contiguous** section memory.

int = 4 bytes

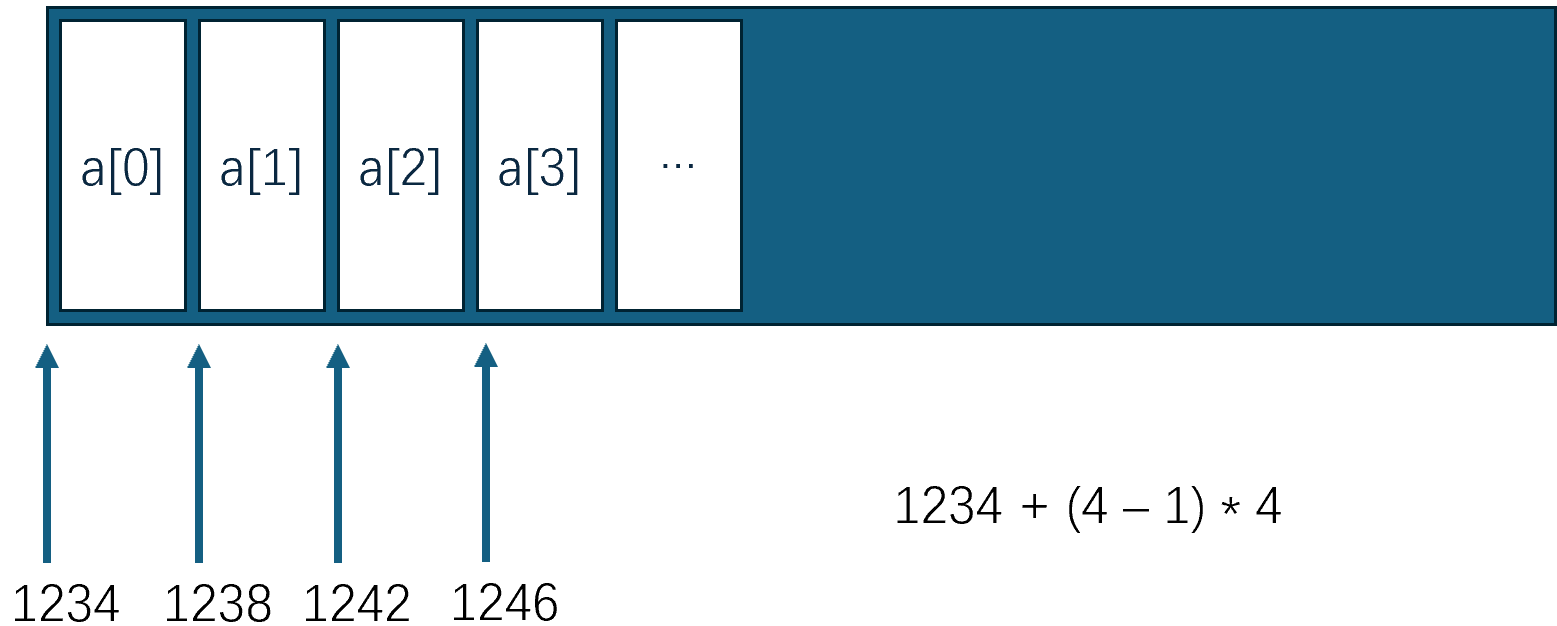


- Recall, an integer variable takes up 4 bytes of memory.

Array's Memory Layout

- Suppose I have an array of 10 elements, stored at memory address 1234.
 - Assume each element takes up 4 bytes of memory.
 - What is the memory address of the 4th element of the array?
- Memory address is the starting place where data is stored.
 - If I have 10 bytes of memory, and integer variable `a` is stored from the 6th byte to the 9th byte, then the memory address of `a` is 6.

Array's Memory Layout



Increment (Good to know!)

- Since we are going to write `i = i + 1` a lot,
- Use `i++` as a shorthand for `i = i + 1`.

```
for(int i = 0; i < 3; i++){  
    printf("%f\n", c[i]);  
}  
// is the same as  
for(int i = 0; i < 3; i = i + 1){  
    printf("%f\n", c[i]);  
}
```

- Similarly, `i += k` is short for `i = i + k`.

```
for(int i = 1; i < 10; i += 2){  
    printf("%d\n", i);  
} //prints out 1,3,5,7,9
```

Today's Agenda

- What is Array?
 - How to
 - create an array?
 - access an array?
- How an array is stored in the computer memory?
- Compilation Error vs. Runtime Error.
- **How to handle matrix algebra using array?**
- How to pass an array as an input argument of a function?
 - Passing by value vs Passing by reference

How to handle matrix algebra using array?

How to store and operate on matrices by using arrays?

Row Major and Column Major Order

- Row Major and Column Major Order are two methods storing a matrix in an array.
- Matrix is a "2D object", you need to flatten it before storing it in a sequential container (such as an array).
- Using zero-based indexing (indices i, j starts from 0),
- Row-major order stores a matrix as

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \end{bmatrix} \implies [A_{00}, A_{01}, A_{02}, A_{10}, A_{11}, A_{12}].$$

- Row major order means $A_{i,j}$ is the $i * \text{ncol} + j$ -th element in the array.

Row Major and Column Major Order

- Column-major order stores a matrix as

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \end{bmatrix} \implies [A_{00}, A_{10}, A_{01}, A_{11}, A_{02}, A_{12}].$$

- Column Major order means $A_{i,j}$ is ?-th element in the array

Row Major and Column Major Order

- In the exam and future CWs, I will say something like:
 - "an array `a` stores a matrix $A \in \mathbb{N}^{m \times n}$, in row-major order. "
 - You should know what I mean by that!

Printing a Matrix

The code below prints out a matrix stored in array A in row major order.

```
int A[] = {1, 2, 3, 4, 5, 6};

for (int i = 0; i < 2; i = i + 1){ // for row i
    for (int j = 0; j < 3; j = j + 1){ // for column j
        printf("%d ", A[i*3 + j]); // print A_ij
    }
    printf("\n");
}
```

It prints out

```
1 2 3
4 5 6
```

Time complexity

What is the time complexity of printing a matrix?

Today's Agenda

- What is Array?
 - How to
 - create an array?
 - access an array?
- How an array is stored in the computer memory?
- Compilation Error vs. Runtime Error.
- How to handle matrix algebra using array?
- **How to pass an array as an input argument of a function?**
 - Passing by value vs Passing by reference

Array as Input Argument

- You can pass array as input variables of a function.
- Simply write `datatype variable_name[]` inside the parenthesis following the function name.
- `array_size` is not needed.

Dot Product

- Consider a function `dot` computes the dot product between two vectors a, b : $a \cdot b = \sum_i a_i b_i$.

```
//compute dot product between a and b.  
//a and b are two input arrays  
double dot(double a[], double b[]){  
    double s = 0;  
    for(int i = 0; i < 3; i++){  
        s += a[i]*b[i];  
    }  
    return s;  
}
```

- What if you do not know the size of `a` and `b`?

Dot Product

- Pass another input argument, specifying the array length.

```
//compute dot product between a and b.  
//a and b are two input arrays  
//len is the length of both a and b.  
double dot(double a[], double b[], int len){  
    double s = 0;  
    for(int i = 0; i < len; i++){  
        s += a[i]*b[i];  
    }  
    return s;  
}
```

Pass by Value

When you pass an input argument to a function, you are passing by value: The program will copy the value to the input variable.

```
#include <stdio.h>
void hack(double s){
    s = 100; //assignment to the input variable!
}
void main(){
    double score = 40;
    hack(score);
    printf("%f %f\n", score);
    //display 40
}
```

The value of `score` is copied to the input argument `s`, thus operations on `s` has no effect on `score`

Pass by Reference

- However, comparing to ordinary variables, the array occupies a much bigger memory space, thus pass by value can be expensive.
- In C, array is passed by reference.
 - If callee changes the array, caller's array will also be changed.

Pass by Reference, Example

```
//add all elements in an array by 1
void hack(double s[]){
    for(int i = 0; i < 4; i++){
        s[i] = 100;
    }
}

void main(){
    double scores[] = {40.0, 41.0, 46.0, 48.0};
    hack(scores);
    printf("%f %f %f %f\n", scores[0], scores[1], scores[2], scores[3]);
    //display 100 100 100 100, NOT 40.0, 41.0, 46.0, 48.0!!
}
```

Return an Array

- Array cannot be returned by a function.
- However, since a function can make changes to caller's array, you can pass an empty array as input argument, and the function stores results in that array.

```
//compute a+b and store the result in c
void add(double a[], double b[], double c[], int len){
    for(int i = 0; i < len; i++){
        c[i] = a[i] + b[i];
    }
}

void main(){
    double a[] = {1.0, 2.0}, b[] = {2.0, 3.0};
    double c[2];
    add(a,b,c,2);
    printf("%f %f\n", c[0], c[1]);
    //display 3 5
```

String = `char` Array!

In C, a string is simply a `char` array that ends with `0`.

For example,

```
char a[] = {'A', 'B', 'C', 0}; // Notice the 0 at the end
char b[] = "ABC";
printf("%s\n", a);
printf("%s\n", b);
printf("%s\n", "ABC");
//all prints out "ABC"
```

Homework 5.1

- Download homework files and read code in the lecture,
 - `vecadd.c` : Vector Addition
 - `vecdot.c` : Vector Dot Product
 - `passingbyval.c` : Passing by Value Example
 - `passingbyref.c` : Passing by Reference Example
- Make sure you understand how to
 - Declare array and Initialize array
 - Read/Write elements of an array
 - Pass array as input to a function and
 - What is pass by value/reference.

Homework 5.2, Find Max (submit)

1. Create a new C file (file->new file... save as `max.c`)
2. In the `main` function, create an integer array and initialize it with a sequence of integers `[2,1,3,4,3]` .
 - Use the initialization syntax introduced in the lecture.
3. Write a function called `max` taking two inputs:
 - The input array
 - The length of the input array.
4. `max` returns the maximum value in the input array.
5. For example, given input `[1,2,2]` , `max` should return 2.
6. Test `max` function in `main` using the array you just created at the 2nd step and print out the maximum value.

- Hint: write pseudo code on a paper first and talk to your coursemates/TA if you are not sure where to start.

Homework 5.3, Swap (submit)

1. Write a new function **in the same file** called `swap`.
2. `swap` takes an array with length 2 as input.
3. `swap` does not return anything.
4. After `swap` function is called, the elements in the input array will be swapped.
 - If the input array is `[1,2]`, it becomes `[2,1]` after `swap` is called.
5. Test your `swap` function in `main` with the array `[1987,10]` and print out the array after swap.

Homework 5.4, Printing Matrix (submit)

1. Start a new file, called `printmat.c` .
2. In your `main` function, create an integer array `A` and initialize it with value `[1,2,3,4,5,6]` .
3. Suppose `A` is a 3 by 2 matrix stored in a row major order.

Homework 5.4, Printing Matrix (submit)

4. Write a function `print` taking 3 inputs.
 - an array storing a matrix in row major order.
 - the number of rows of the matrix.
 - the number of columns of the matrix.
5. `print` prints out the input matrix in a proper format.
6. In the `main` function, test your `print` function with the matrix `A` you have created in the second step.