Refresher Module CS231

Introduction to C

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Pointer arithmetic

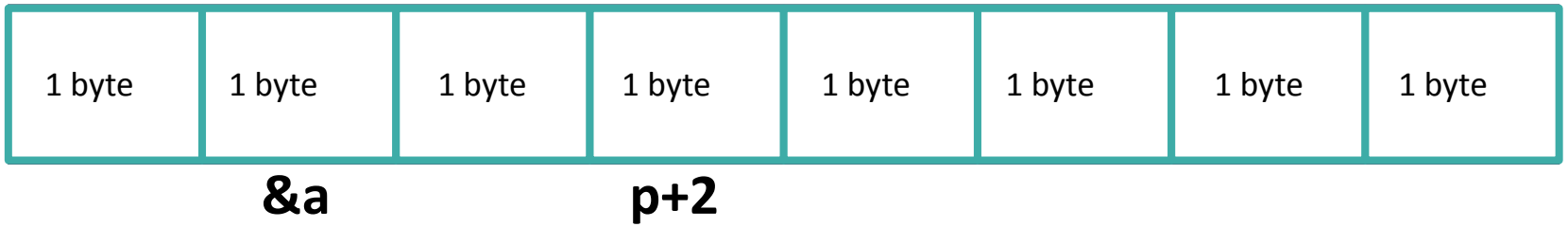
datatype *p = &a;

p+**n** = &a + **n***`sizeof`(**datatype**)

We are traversing in **blocks of memory** whenever address arithmetic is used.

Pointer arithmetic (concrete example)

```
char *p = &a;
p+2 = &a + 2*sizeof(char)
    = &a + 2*1
    = &a + 2
```



Pointer arithmetic

```
datatype *p = &a;
```

```
p+n = &a + n*sizeof(datatype)
```

Pointer arithmetic

```
datatype *p = &a;
```

p+**n** = &a + **n***`sizeof`(**datatype**)

Application: Arrays!

Arrays

```
datatype name[size];
```

Declaration

```
name =  { el1, el2,..., elk};
```

Definition

Arrays are stored in **contiguous** memory locations!

Arrays

```
datatype name[size];
```

Declaration

```
name =  { el1, el2,..., elk};
```

Wrong Definition!

Assignment in arrays is not allowed after declaration!

Arrays are stored in **contiguous** memory locations!

Arrays

```
datatype name[size];
```

Declaration

```
name =  { el1, el2,..., elk};
```

Wrong Definition!

Assignment in arrays is not allowed after declaration!

```
name[j] =  elj;
```

Piecewise assignment is allowed

Arrays are stored in **contiguous** memory locations!

Arrays

```
datatype name[size] =  { el1, el2,..., elk};
```

```
datatype name[] = { el1, el2,..., elk};
```

Arrays

```
datatype name[size] =  { el1, el2,..., elk};
```

What is the difference between these declarations and definitions?

```
datatype name[] = { el1, el2,..., elk};
```

Declaration of n-dimensional Arrays

Arrays are stored in **contiguous** memory locations!

```
datatype name[s1][s2]....[sn];
```

```
datatype name[s1 * s2 *...* sn];
```

These are equivalent declarations

Array storage in memory

1. The elements of an array can be stored in **column-major layout** or **row-major layout**.

2. For an array stored in column-major layout, the elements of the columns are contiguous in memory.

3. In row-major layout, the elements of the rows are contiguous. **This is what C uses.**

Array storage in memory

Row major

A(i,j) element is at **A[j + i * n_columns]**

Array storage in memory

Row major

`A(i,j)` element is at **`A[ j + i * n_columns]`**

Col major

`A(i,j)` element is at **`A[i + j * n_rows]`**

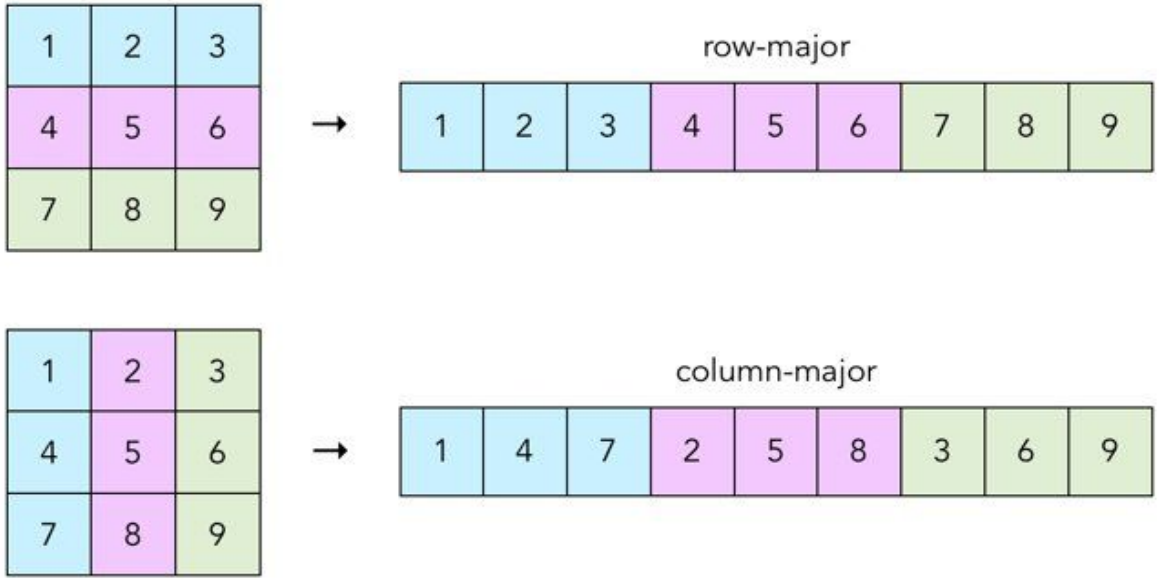
Array storage in memory

Row major

`A(i,j)` element is at **`A[ j + i * n_columns]`**

Col major

`A(i,j)` element is at **`A[i + j * n_rows]`**



https://craftofcoding.wordpress.com/2017/02/03/column-major-vs-row-major-arrays-does-it-matter/

Array storage in memory

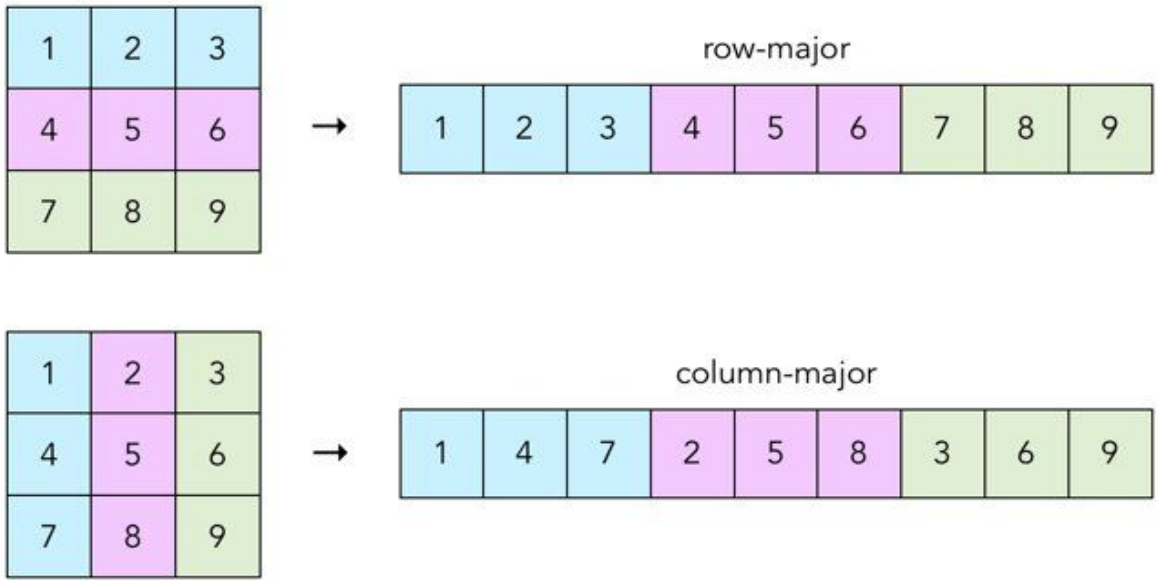
Row major

A(i,j) element is at **A[j + i * n_columns]**

Does it matter?

Col major

A(i,j) element is at **A[i + j * n_rows]**



https://craftofcoding.wordpress.com/2017/02/03/column-major-vs-row-major-arrays-does-it-matter/

Array storage in memory

Row major

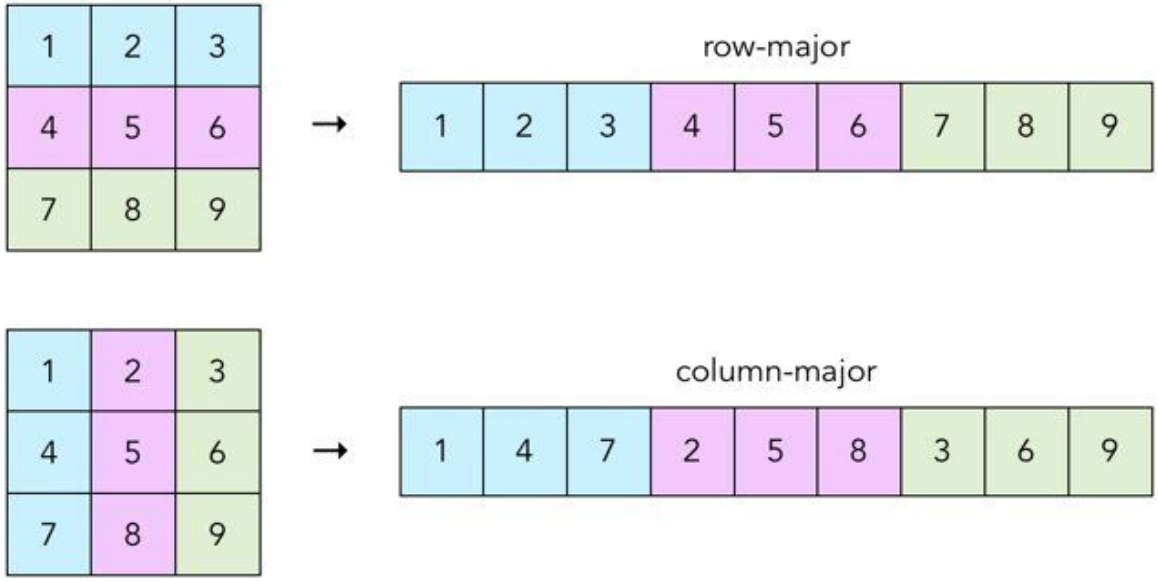
`A(i,j)` element is at `A[ j + i * n_columns]`

Col major

`A(i,j)` element is at `A[i + j * n_rows]`

Does it matter?

OS concept: Caching



https://craftofcoding.wordpress.com/2017/02/03/column-major-vs-row-major-arrays-does-it-matter/

Array, pointers and memory

```
int arr[10]={1,2,3};
```

1. Here space has been reserved for 10 integers at contiguous locations in the memory.

Array, pointers and memory

```
int arr[10]={1,2,3};
```

1. Here space has been reserved for 10 integers at contiguous locations in the memory.

2. The name of the array points to the first byte of the address of the first array location.

Array, pointers and memory

```
int arr[10]={1,2,3};
```

1. Here space has been reserved for 10 integers at contiguous locations in the memory.

2. The name of the array points to the first byte of the address of the first array location.

For chars, referencing **arr** is the same as saying **&arr[0]**.

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

How does pointer arithmetic work with arrays?

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

How does pointer arithmetic work with arrays?

The following 3 notations are equivalent

`*(pa+i)` **`arr[i]`** **`*(arr+i)`**

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

Difference between pointers and arrays

```
*(pa+i)   arr[i]   *(arr+i)
```

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

Difference between pointers and arrays

Warning!

```
*(pa+i)    arr[i]    *(arr+i)
```

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

Difference between pointers and arrays

Warning! What happens in the previous example if **`i>10`**?

```
*(pa+i)    arr[i]    *(arr+i)
```

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

Difference between pointers and arrays

Warning! What happens in the previous example if **`i>10`**? The **red options** throw error.

`*(pa+i)` `arr[i]` `*(arr+i)`

Array, pointers and memory

```
int arr[10]={1,2,3};
int *pa=arr;
```

Difference between po

Warning! What happe
example if **`i>10`**? The

```
*(pa+i)    arr[i]
```

Strings in C

1-dimensional character arrays which terminate with the null symbol.

They have special semantics (meaning) with respect to the C-compiler compared to normal character arrays.

Strings in C

1-dimensional character arrays which terminate with the null symbol.

They have special semantics (meaning) with respect to the C-compiler compared to normal character arrays.

All C-strings are char arrays, but not all char arrays are C-strings.

Strings have dedicated special library functions such as
`strlen(), strcpy(), strcmp(), strcat()`

Strings in C

1-dimensional character arrays which terminate with the null symbol.

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

char str[50] = "abcd";

char str[5] = "abcde";

char str[] = {'a', 'b', 'c', 'd', '\0'};

char str[]; str = "abcd";

char str[5] = {'a', 'b', 'c', 'd', '\0'};

char str[1,2] = {'a','b',\0'};
```

Which of these are valid strings?

Strings in C

1-dimensional character arrays which terminate with the null symbol.

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

char str[50] = "abcd";

char str[5] = "abcde";

char str[] = {'a', 'b', 'c', 'd', '\0'};

char str[]; str = "abcd";

char str[5] = {'a', 'b', 'c', 'd', '\0'};

char str[1,2] = {'a','b',\0'};
```

Which of these are valid strings?

Strings in C

1-dimensional character arrays which terminate with the null symbol.

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

char str[50] = "abcd";

char str[5] = "abcde";

char str[] = {'a', 'b', 'c', 'd', '\0'};

char str[]; str = "abcd";

char str[5] = {'a', 'b', 'c', 'd', '\0'};

char str[1,2] = {'a','b',\0'};
```

Which of these are valid strings?

This string declaration is allowed in C!!

Strings in C

1-dimensional character arrays which terminate with the null symbol.

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

char str[50] = "abcd";

char str[5] = "abcde";

char str[]

char str[];

char str[5]

char str[1,
```

Which of these are valid strings?

This string declaration is allowed in C!!

Read more:

- https://stackoverflow.com/questions/3828307/defining-a-string-with-no-null-terminating-char-0-at-the-end
- https://stackoverflow.com/questions/14884434/where-can-i-assume-null-characters-will-be-added-automatically-in-c/14884549#14884549

Passing Arrays to Functions

```
#include <stdio.h>

int func1(float array[4]) {
  float value = 0.3;
  //do something
  return value;
}

int main() {
  float result, arr[] = {23.4, 55, 22.6};
  result = func1(?);
  printf("Result = %.2f", result);
  return 0;
}
```

Pass the float array to func1() and store the return value in result

Passing Arrays to Functions

```
#include <stdio.h>

int func1(float array[4]) {
  float value = 0.3;
  //do something
  return value;
}

int main() {
  float result, arr[] = {23.4, 55, 22.6};
  result = func1(?);
  printf("Result = %.2f", result);
  return 0;
}
```

Pass the float array to func1() and store the return value in result

Passing Arrays to Functions

```
#include <stdio.h>

int func1(float array[4]) {
  float value = 0.3;
  //do something
  return value;
}

int main() {
  float result, arr[] = {23.4, 55, 22.6};
   result = func1(arr);
  printf("Result = %.2f", result);
  return 0;
}
```

Pass the float array to func1() and store the return value in result

Passing Arrays to Functions

```
#include <stdio.h>

int func2(??) {
  float value = 0.3;
  //do something
  return value;
}

int main() {
  float result, arr[2][2] = {23.4, 55, 22.6, 10};
  result = func2(arr);
  printf("Result = %.2f", result);
  return 0;
}
```

Pass the float array to func2() and store the return value in result

Passing Arrays to Functions

```
#include <stdio.h>

int func2(float array[2][2]) {
  float value = 0.3;
  //do something
  return value;
}

int main() {
  float result, arr[2][2] = {23.4, 55, 22.6, 10};
  result = func2(arr);
  printf("Result = %.2f", result);
  return 0;
}
```

Pass the float array to func2() and store the return value in result

Passing Arrays to Functions

1. `int func(int array[s1][s2][s3]);`
2. `int func(int array[][s2][s3]);`
3. `int func(int array[][][s3]);`
4. `int func(int array[][][]);`

Which of these is a valid function declaration?

Passing Arrays to Functions

1. `int func(int array[s1][s2][s3]);`
2. `int func(int array[][s2][s3]);`
3. `int func(int array[][][s3]);`
4. `int func(int array[][][]);`

Which of these is a valid function declaration?

Passing Arrays to Functions

1. `int func(int array[s1][s2][s3]);`
2. `int func(int array[][s2][s3]);`
3. `int func(int array[][][s3]);`
4. `int func(int array[][][]);`

The number of columns should always be specified!

Which of these is a valid function declaration?

Passing Arrays to Functions

1. `int func(int array[s1][s2][s3]);`
2. `int func(int array[][s2][s3]);`
3. `int func(int array[][][s3]);`
4. `int func(int array[][][]);`

The number of columns should always be specified!

The answer is related to the fact that C uses Row-Major storage.

Passing Arrays to Functions

```
1. int func(int array[s1][s2][s3]);

2. int func(int array[][s2][s3]);

3. int func(int array[][][s3]);

4. int func(int array[][][]);
```

The number of columns should always be specified!

The answer is related to the fact that C uses Row-Major storage.

Read More:

1. https://stackoverflow.com/questions/68047223/why-is-it-mandatory-to-specify-number-of-columns-while-declaring-initializing-mu
2. https://stackoverflow.com/questions/35289985/why-is-dimension-range-of-higher-dimensions-in-multi-dimensional-array-required

Take Home lab 1

Write your version of the strlen() function

- Using a for-loop.
- Using a while-loop.

Learning

Look up what a string in C is.

Look up what EOF is.

Take Home lab 2

Write a function to pass a 2*3 array and

1. Count the number of odd elements.
2. Change the last element of every row to zero.
3. Print the changed array back in main.

Learning

Passing arrays to functions.