Laboratorio de Programación

2016 -2

Juan Camilo Rada[1]

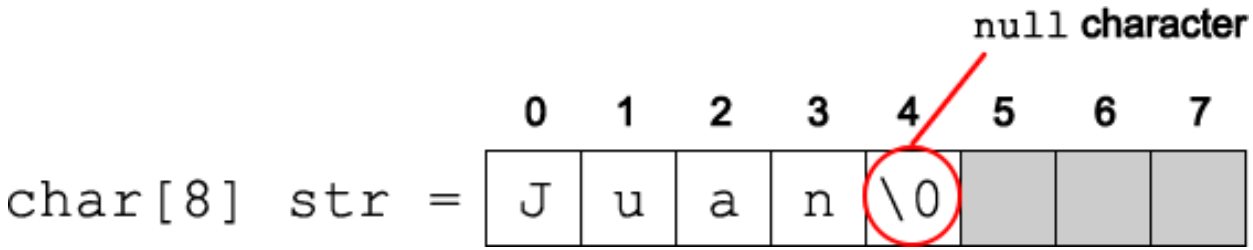
[1]Departamento de Electronica y ciencias de la computación

Septiembre de 2016

Outline

Outline

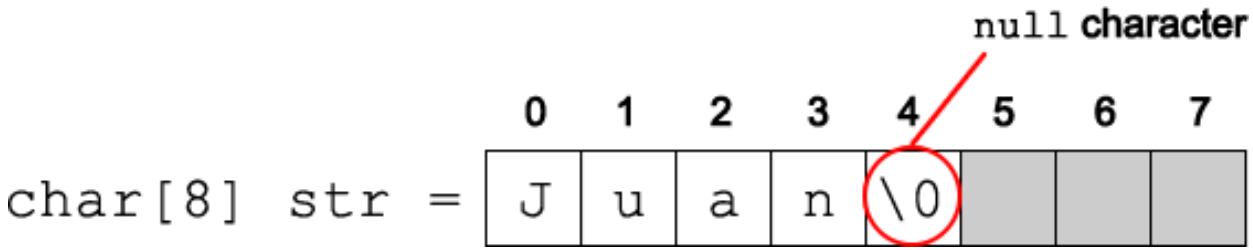
Arrays of Characters



Char array

- An string is represented in c by an array of chars

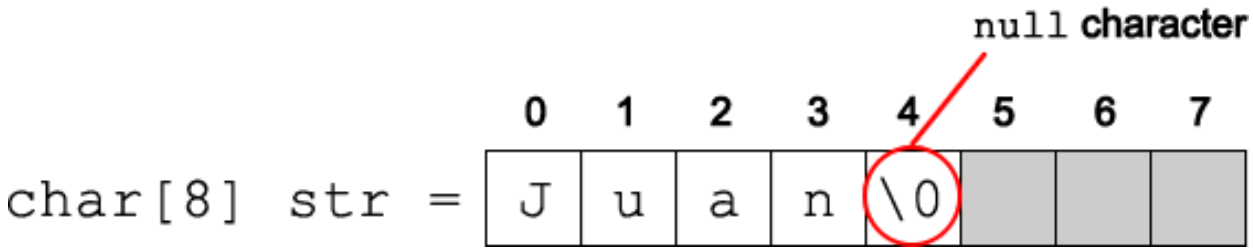
Arrays of Characters



Char array

- An string is represented in c by an array of chars
- We can store a short string into a larger char array

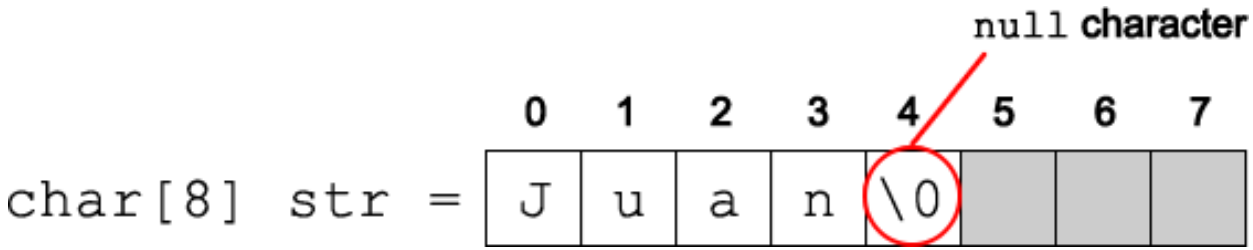
Arrays of Characters



Char array

- An string is represented in c by an array of chars
- We can store a short string into a larger char array
- The end of the string is represented by the null character \0

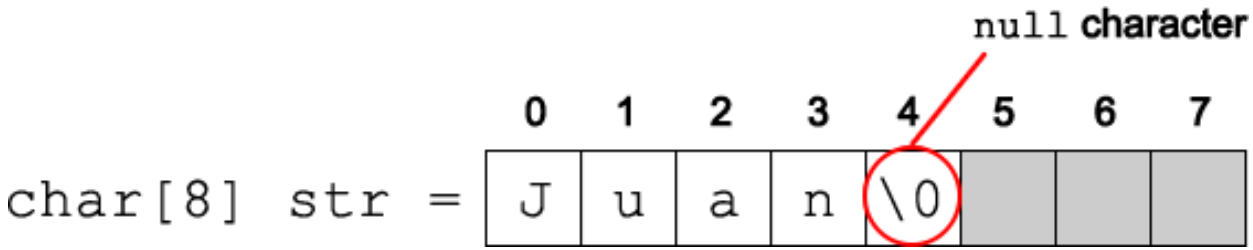
Arrays of Characters



Char array

- An string is represented in c by an array of chars
- We can store a short string into a larger char array
- The end of the string is represented by the null character \0
- Simple representation: `char str[] = "Juan"`

Arrays of Characters

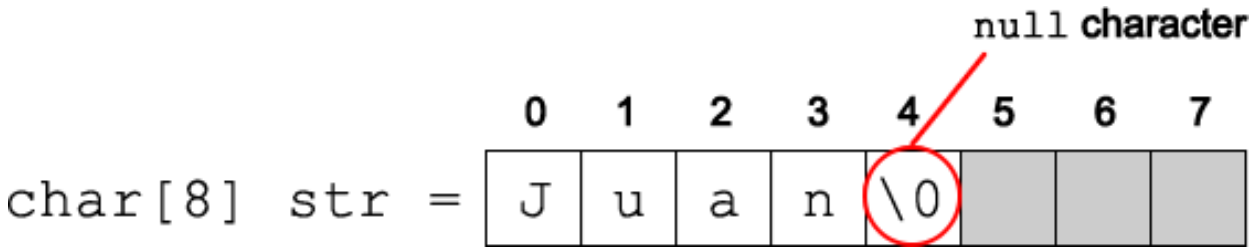


Operation

When we are talking about of strings, lots of operations come to mind

- Compare two strings

Arrays of Characters

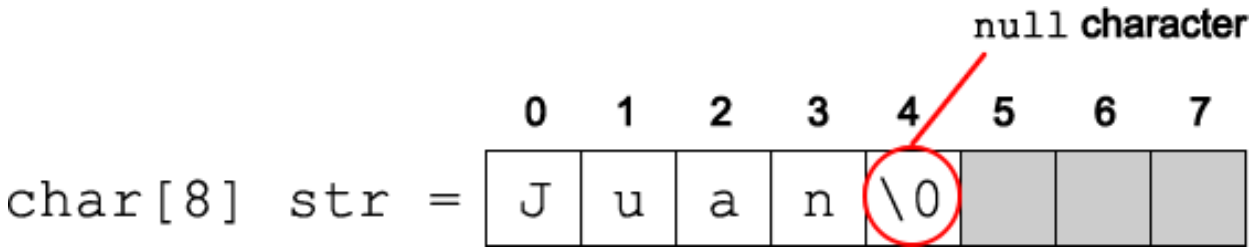


Operation

When we are talking about of strings, lots of operations come to mind

- Compare two strings
- Concatenate two strings

Arrays of Characters

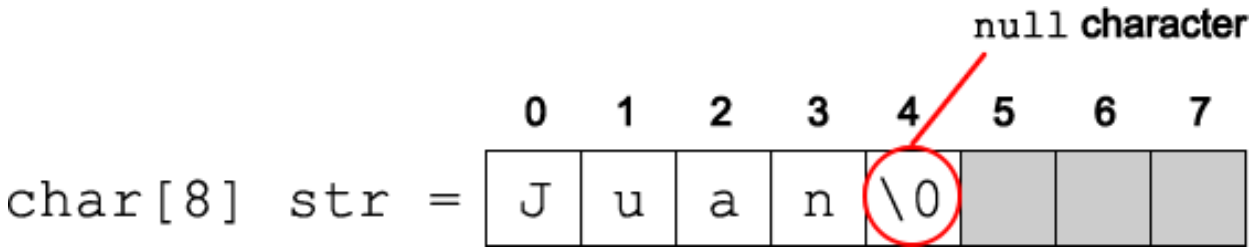


Operation

When we are talking about of strings, lots of operations come to mind

- Compare two strings
- Concatenate two strings
- Add characters

Arrays of Characters

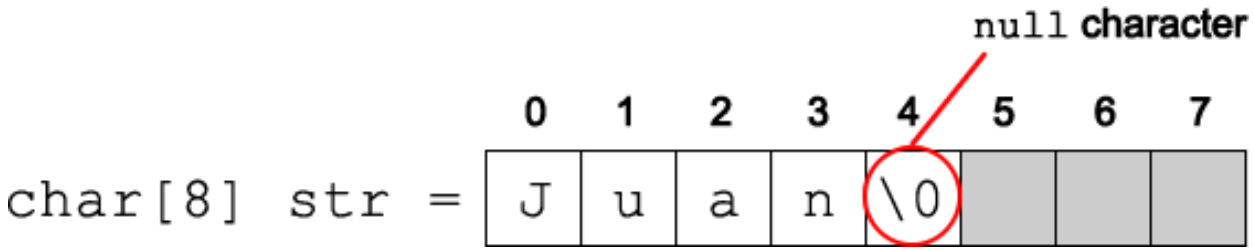


Operation

When we are talking about of strings, lots of operations come to mind

- Compare two strings
- Concatenate two strings
- Add characters
- Remove characters

Arrays of Characters



Operation

When we are talking about of strings, lots of operations come to mind

- Compare two strings
- Concatenate two strings
- Add characters
- Remove characters
- Search characters or strings into a string

Output of the code

C

```c
#include <stdio.h>

int main()
{
    char array1[] = "hola";
    char array2[] = "hola";
    if (array1 == array2)
        printf("ok\n");
    else
        printf("No\n");

    return 0;
}
```

Outline

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying
- Concatenation

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying
- Concatenation
- Comparison

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying
- Concatenation
- Comparison
- Searching

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying
- Concatenation
- Comparison
- Searching
- Determine the size of an string

string.h

The string.h library

C provide a library to ease the manipulation of strings. You can find functions to tackle the most common operations on strings:

- Copying
- Concatenation
- Comparison
- Searching
- Determine the size of an string

We have to include the library `#include<string.h>`

string.h - Copying

strcpy

```
strcpy(char* destination, char* source)
```

- Copy all the string from the source to the destination string

string.h - Copying

strcpy

`strcpy(char* destination, char* source)`

- Copy all the string from the `source` to the `destination` string

strncpy

`strncpy(char* destination, char* source, int num)`

- Copy the first `num` of chars of the `source` to the `destination` string
- The `null` character is not added so, It has to be added manually, otherwise the string can not be printed!

string.h - Copying

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {

    char str[] = "Hallo ";
    char str2[50];
    char str3[50];

    strcpy(str2, str);
    strncpy(str3, str, 7);
    str3[7] = '\0';

    printf ("Str2 = %s \nStr3 = %s", str2, str3);

}
```

string.h - Concatenation

strcat

```
strcat(char* destination, char* source)
```

- Appends a copy of the source string to the destination string

string.h - Concatenation

strcat

```
strcat(char* destination, char* source)
```

- Appends a copy of the source string to the destination string

strncat

```
strncat(char* destination, char* source, int num)
```

- Appends a copy of the first num of chars of the source to the destination string

string.h - Concatenation

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {
        char str[50] = "Ich";
        char str2[] = "habe ";
        char str3[] = "frage und antwort";

        strcat(str, " ");
        strcat(str, str2);
        strcat(str, "eine ");
        strncat(str, str3, 5);

        printf ("Str = %s", str);
}
```

string.h - Comparison

strcmp

```
int strcmp (char* str1, char* str2 )
```

- Compares the string str1 to the string str2
- Return 0 if both strings are equals.
- Return -1 if the first char of difference has a lower value in str1 than in str2
- Return 1 if the first char of difference has a higher value in str1 than in str2

string.h - Comparison

Test the code

```
int diff = strcmp("apple", "apples");
printf ("Diff = %d\n", diff);

diff = strcmp("apple", "apple");
printf ("Diff = %d\n", diff);

diff = strcmp("apples", "apple");
printf ("Diff = %d\n", diff);

diff = strcmp("apples", "applet");
printf ("Diff = %d\n", diff);

diff = strcmp(" ", "");
printf ("Diff = %d\n", diff);

diff = strcmp("", " ");
printf ("Diff = %d\n", diff);
```

string.h - Comparison

strncmp

```
int strncmp (char* str1, char* str2, int num )
```

- Compares the first num characters of the string str1 to the first num characters of the string str2
- Return 0 if both strings are equals.
- Return -1 if the first char of difference has a lower value in str1 than in str2
- Return 1 if the first char of difference has a higher value in str1 than in str2

string.h - Comparison

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {
    char names[][30] =
        {"Daniel", "Diana", "Paula", "Damian"};

    for(int i=0; i<4; i++){
        if(strncmp(names[i], "Da...", 2) == 0){
            printf(
                "%s starts with \"Da\" \n",
                names[i]);
        }
    }
}
```

string.h - Searching

strchr

```
char* strchr (char* str, int char )
```

- Points the first occurrence of the input char in the str

strcspn

```
int strcspn (char* str1, char* str2 )
```

- Search in str1 for the first appearance of any of the characters in str2
- Return the number of chars counted before the character is found

string.h - Searching

strpbrk

```
char* strpbrk (char* str1, char* str2 )
```

- Search in str1 for the first appearance of any of the characters in str2
- Return a pointer to the first occurrence in str1 of any character in str2

strstr

```
char* strstr (char* str1, char* str2 )
```

- Search an occurrence of str2 in str1
- Return a pointer to str1, where str2 has been located

string.h - Searching

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {

  char str[] = "Hello world, I am alive";
  char* pointer = strchr(str,'o');
  while (pointer != NULL)
  {
    printf ("found at %i\n",pointer-str+1);
    pointer=strchr(pointer+1,'o');
  }

  int position = strcspn (str,",");
  printf ("Comma found at %i\n", position);
}
```

string.h - Searching

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {

  char str[] = "Hello world, I am alive";
  char* pointer = strstr (str,"world");
  *pointer = 'P';

  printf ("%s\n", str);
}
```

string.h - Searching

strtok

```
char* strtok (char* str, char* delimiters )
```

- Splits the input string into tokens, separated by the occurrence of any of the delimiters

string.h - Searching

Test the code

```
#include <stdio.h>
#include <string.h>

int main() {

  char str[] = "Hello world, I-am alive.now";
  char delimiters[] = " ,.-";
  char* pointer = strtok (str, delimiters);

  while (pointer != NULL)
  {
    printf ("%s\n",pointer);
    pointer = strtok (NULL, delimiters);
  }
}
```

string.h - Size

strlen

```
int strtok (char* str)
```

- Returns the length of the input string

Test the code

```
char str[512] = "Hello world, I am alive";
printf("Size = %i", strlen(str));
```

string.h - Unsafe functions

```
char selectedOption[5];
printf("please enter an option: ");
scanf("%s", selectedOption); // option_five
printf("%s\n", selectedOption)
```

string.h - safe functions

```
char selectedOption[5];
printf("please enter an option: ");
scanf("%4s", selectedOption); // option_five
printf("%s\n", selectedOption)
```

string.h - Unsafe functions

- strcpy -> strncpy
- strcat -> strncat
- strcmp -> strncmp
- strchr -> strchrn

String in others languages

- Java
- C#

Outline

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

The problem

- Tables: Information arranged in *rows* and *columns*

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

The problem

- Tables: Information arranged in *rows* and *columns*
- Multiple arrays representing every row:

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

The problem

- Tables: Information arranged in *rows* and *columns*
- Multiple arrays representing every row:
  - `double[4] martin = {4.5, 4.0, 4.2, 4.3}`
  - `double[4] juana = {4.5, 4.0, 4.2, 4.3}`

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

The problem

- Tables: Information arranged in *rows* and *columns*
- Multiple arrays representing every row:
  - `double[4] martin = {4.5, 4.0, 4.2, 4.3}`
  - `double[4] juana = {4.5, 4.0, 4.2, 4.3}`
- What if we have hundreds or thousands of records?

Tables

Student	1st	2nd	3rd	4th
Martin	4.5	4.0	4.2	4.3
Juana	5.0	4.5	4.4	4.4
Beth	4.9	3.5	4.0	4.1
Frank	3.0	2.5	3.0	3.9
Caroline	2.9	3.5	3.4	3.1

The problem

- Tables: Information arranged in *rows* and *columns*
- Multiple arrays representing every row:
  - `double[4] martin = {4.5, 4.0, 4.2, 4.3}`
  - `double[4] juana = {4.5, 4.0, 4.2, 4.3}`
- What if we have hundreds or thousands of records?
- Do we need to create hundreds or thousands of arrays?

Outline

Matrix

4.5	4.0	4.2	4.3
5.0	4.5	4.4	4.4
4.9	3.5	4.0	4.1
3.0	2.5	3.0	3.9
2.9	3.5	3.4	3.1

Matrix

4.5	4.0	4.2	4.3
5.0	4.5	4.4	4.4
4.9	3.5	4.0	4.1
3.0	2.5	3.0	3.9
2.9	3.5	3.4	3.1

The solution

- Table: Information arranged in *rows* and *columns*

Matrix

4.5	4.0	4.2	4.3
5.0	4.5	4.4	4.4
4.9	3.5	4.0	4.1
3.0	2.5	3.0	3.9
2.9	3.5	3.4	3.1

The solution

- Table: Information arranged in *rows* and *columns*
- Tabular data can be represented using *Multidimensional Arrays*

Matrix

```
double matrix[5][4]=
```

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

`double matrix[5][4]=`

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

- Add a new subscript to the array notation

Matrix

`double matrix[5][4]=`

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

- Add a new subscript to the array notation
- First subscript represents the matrix rows

Matrix

`double matrix[5][4]=`

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

- Add a new subscript to the array notation
- First subscript represents the matrix rows
- Second subscript represents the matrix columns

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

- `double matrix[5][4]` $\rightarrow$ Declare a matrix with 5 *rows* and 4 *columns*

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

- `double matrix[5][4]` → Declare a matrix with 5 *rows* and 4 *columns*
- A matrix can be understood as an array which contains arrays

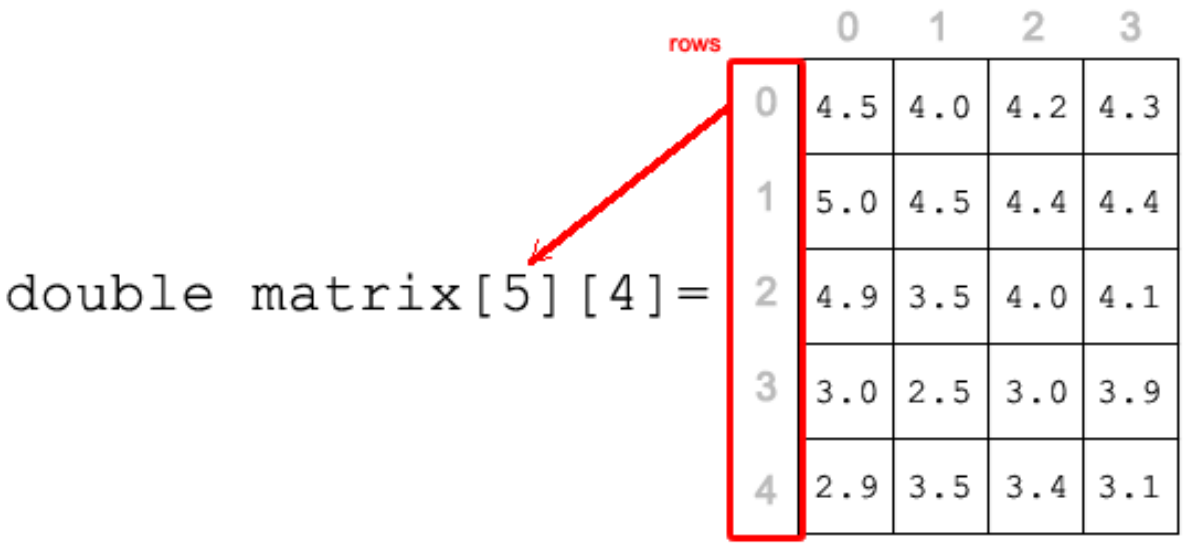
Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

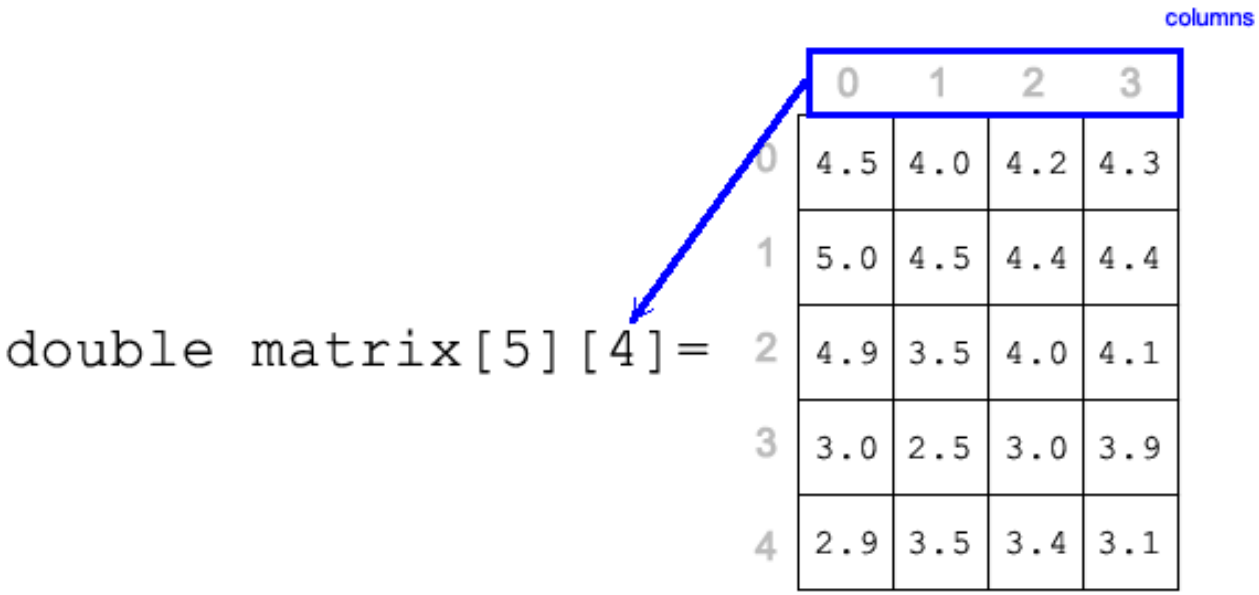
Matrix

- `double matrix[5][4]` → Declare a matrix with 5 *rows* and 4 *columns*
- A matrix can be understood as an array which contains arrays
- Every row is represented as an array

Matrix


rows
0 1 2 3
0 4.5 4.0 4.2 4.3
1 5.0 4.5 4.4 4.4
2 4.9 3.5 4.0 4.1
3 3.0 2.5 3.0 3.9
4 2.9 3.5 3.4 3.1
double matrix[5][4]=

Matrix


columns
0 1 2 3
0 4.5 4.0 4.2 4.3
1 5.0 4.5 4.4 4.4
2 4.9 3.5 4.0 4.1
3 3.0 2.5 3.0 3.9
4 2.9 3.5 3.4 3.1
double matrix[5][4]=

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

- `matrix[1][3]` $\rightarrow$ `4.4`

  Values are accessed with help of matrix subscripts

Matrix

```
double matrix[5][4] =
        {
                {4.5, 4.0, 4.2, 4.3},
                {5.0, 4.5, 4.4, 4.4},
                {4.9, 3.5, 4.0, 4.1},
                {3.0, 2.5, 3.0, 3.9},
                {2.9, 3.5, 3.4, 3.1}
        };
```

Matrix

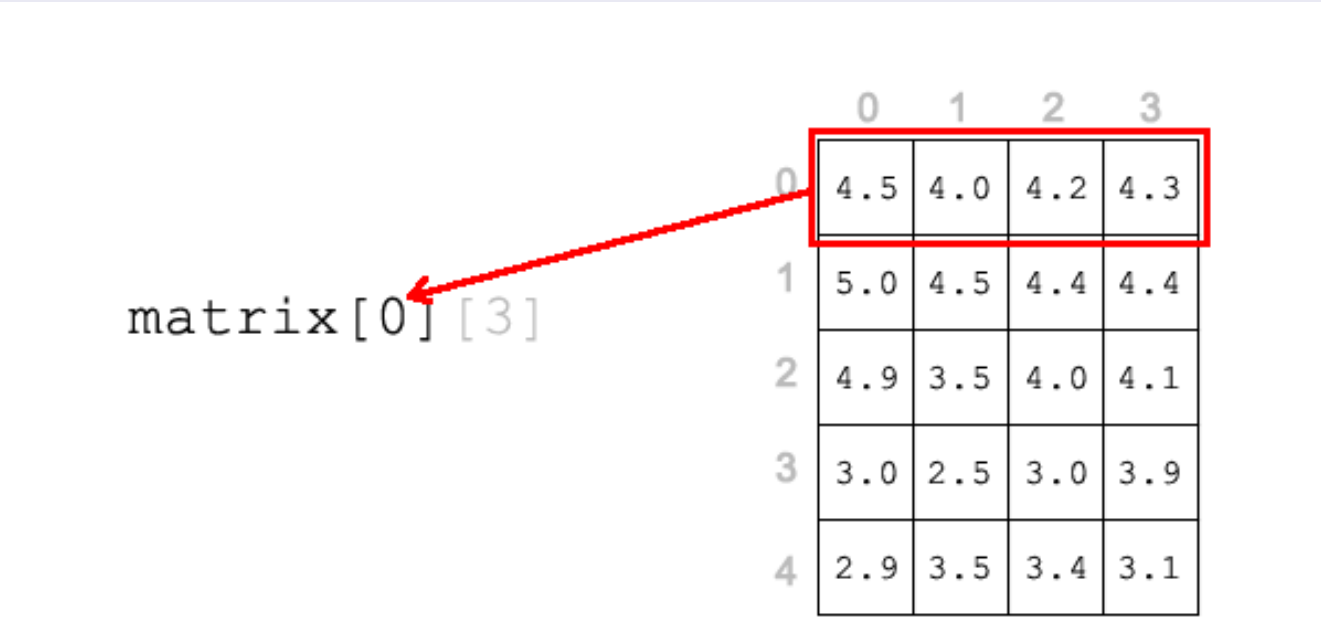
- `matrix[1][3]` $\rightarrow$ `4.4`
  Values are accessed with help of matrix subscripts
- `matrix[2][1] = 1.0`
  Values are set with help of matrix subscripts

Matrix

How it works?

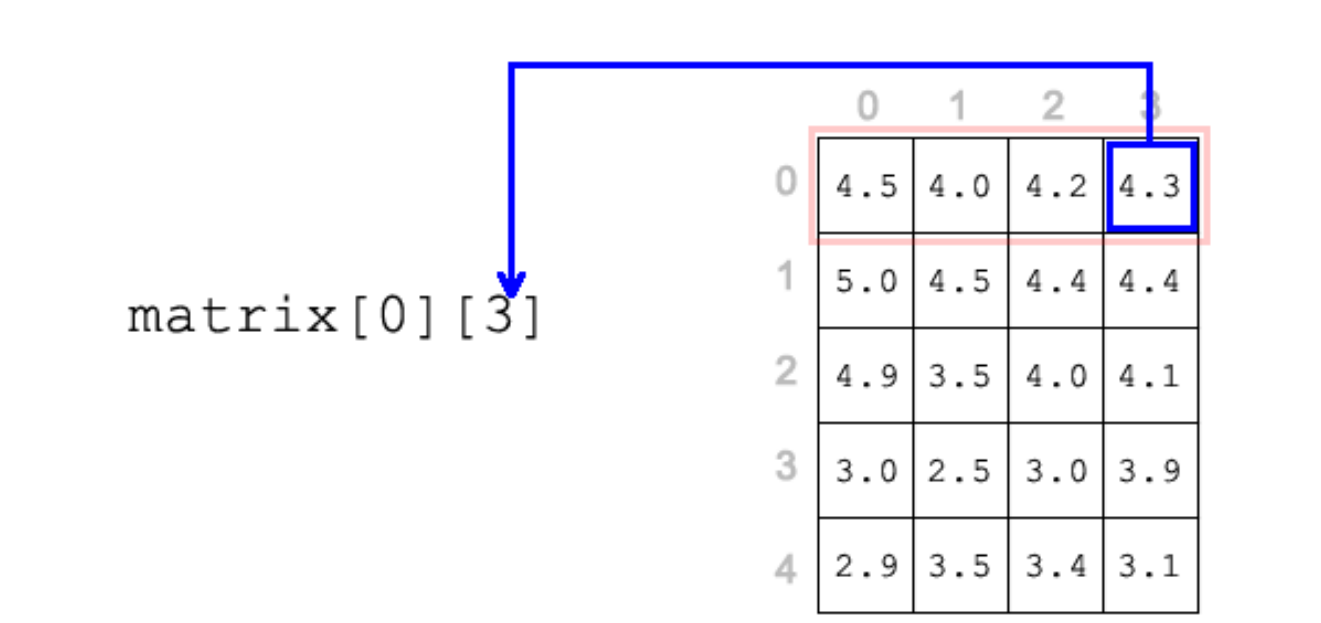
Matrix

Accessing a value



Matrix

Accessing a value



Matrix

Accessing a value

```
matrix[0][3]->4.3
```

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

Set a value

```
matrix[2][1]=1.0
```

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

Set a value

```
matrix[2][1]=1.0
```

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	3.5	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix

Set a value

```
matrix[2][1]=1.0
```

	0	1	2	3
0	4.5	4.0	4.2	4.3
1	5.0	4.5	4.4	4.4
2	4.9	1.0	4.0	4.1
3	3.0	2.5	3.0	3.9
4	2.9	3.5	3.4	3.1

Matrix - Iterate

```
int numberOfRows = 5;
int numberOfColumns = 4;

for (int row = 0; row < numberOfRows; row++)
{
        for (int column = 0; column < numberOfColumns; column++)
        {
                matrix[row][column];
        }
}
```

Matrix - Iterate

```
int numberOfRows = 5;
int numberOfColumns = 4;

for (int row = 0; row < numberOfRows; row++)
{
        for (int column = 0; column < numberOfColumns; column++)
        {
                matrix[row][column];
        }
}
```

Iterate over a Matrix

- We can iterate over all the element in the matrix to perform mass operations

Matrix - Iterate

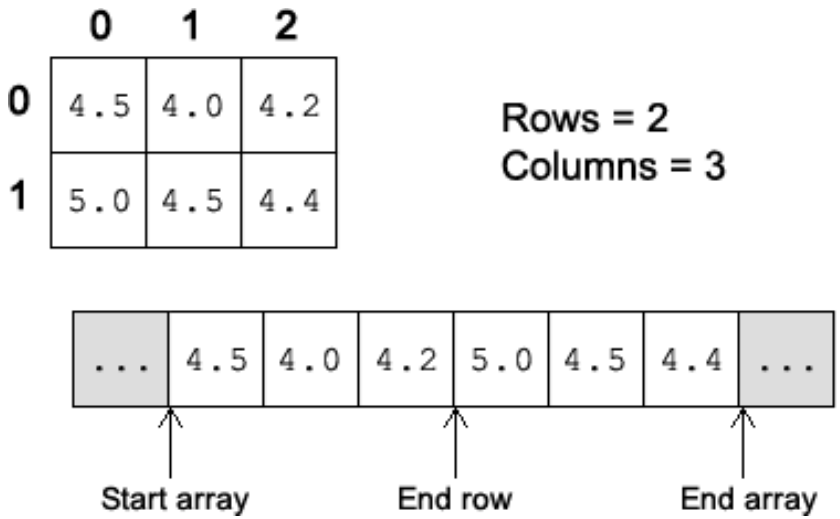
```
int numberOfRows = 5;
int numberOfColumns = 4;

for (int row = 0; row < numberOfRows; row++)
{
        for (int column = 0; column < numberOfColumns; column++)
        {
                matrix[row][column];
        }
}
```

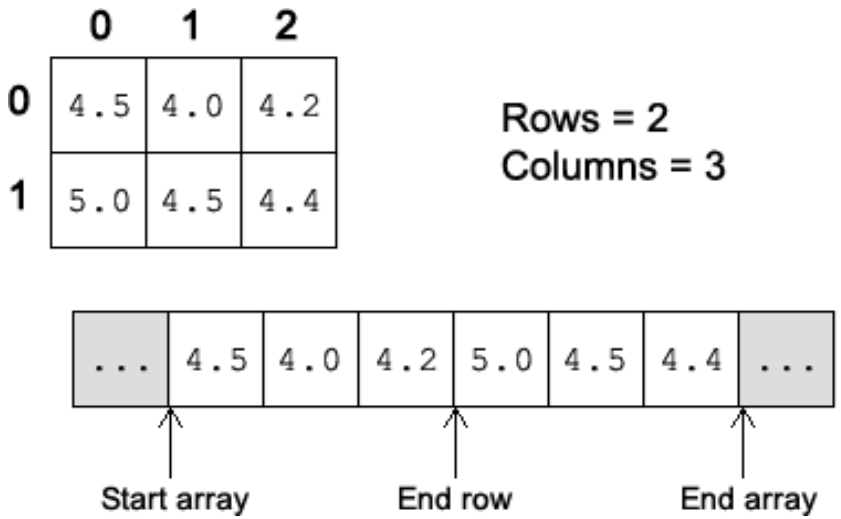
Iterate over a Matrix

- We can iterate over all the element in the matrix to perform mass operations
- Subscripts are modified in every iteration. That ensures that we can cover all the matrix positions

Matrix - Memory



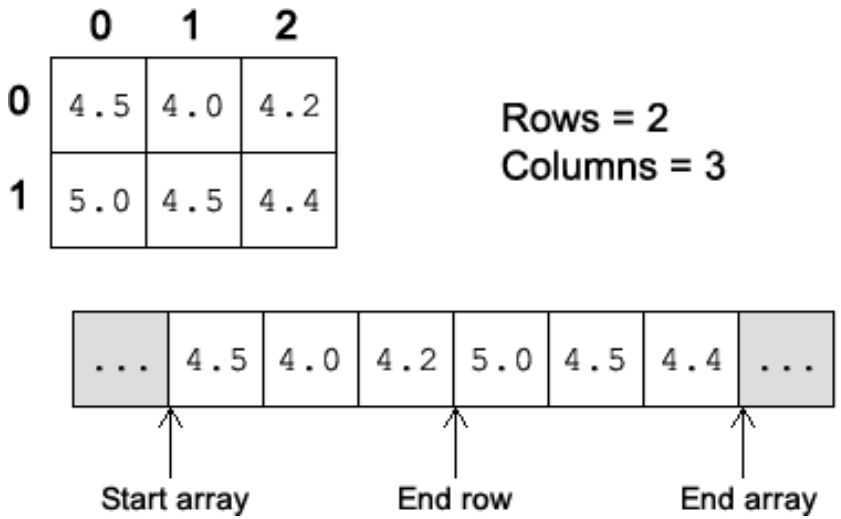
Matrix - Memory



How multidimensional arrays are stored

- Values are stored in memory over consecutive positions as a single dimension array

Matrix - Memory

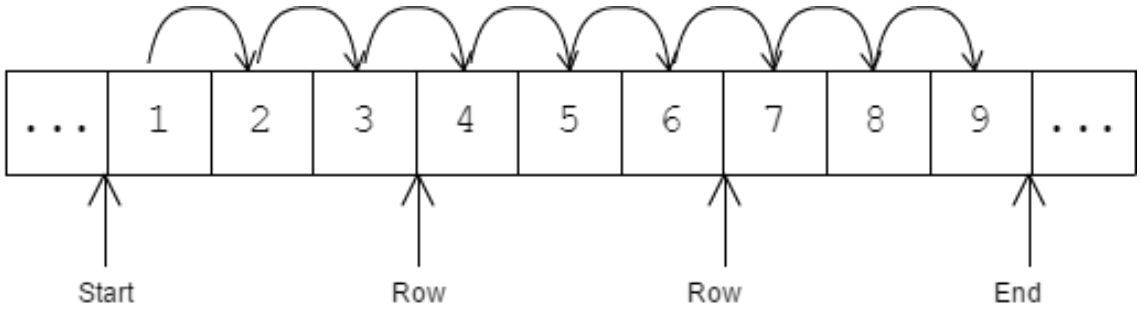


How multidimensional arrays are stored

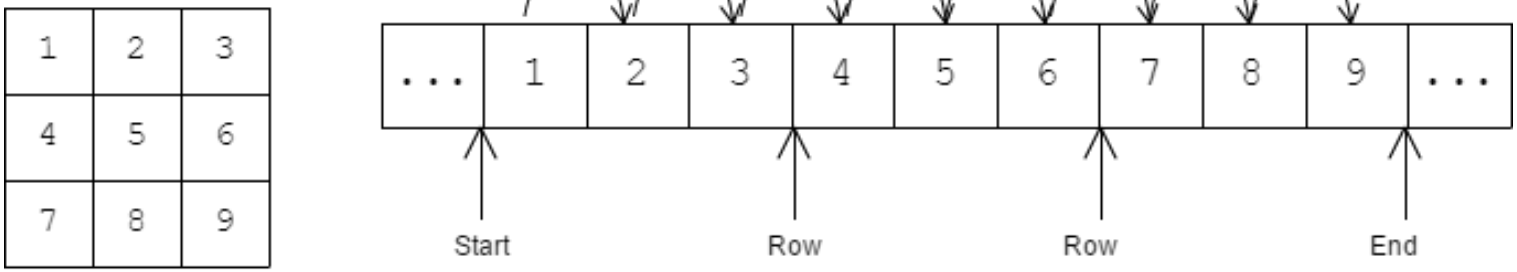
- Values are stored in memory over consecutive positions as a single dimension array
- Compiler has to know the number of columns of the array, to properly iterate over the memory allocations

Matrix - Memory

1	2	3
4	5	6
7	8	9



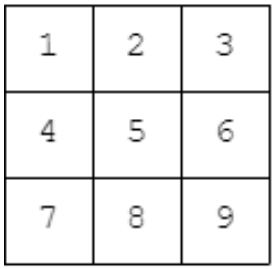
Matrix - Memory

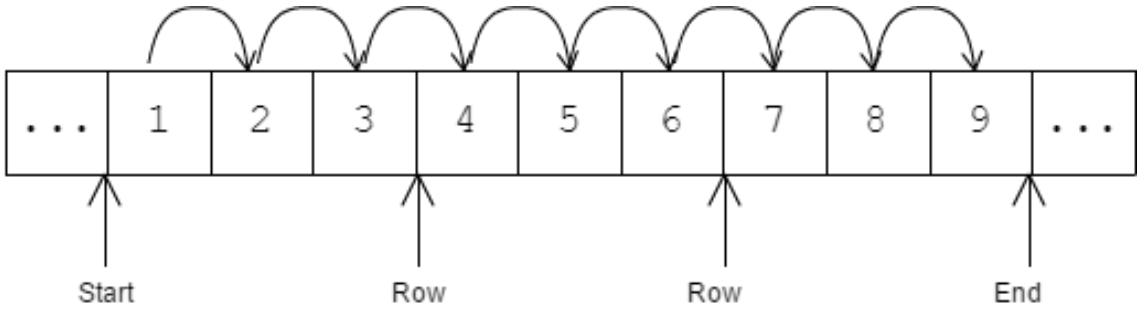


Why we have to iterate over rows - then columns

- Position of the row is calculated just one time per row

Matrix - Memory



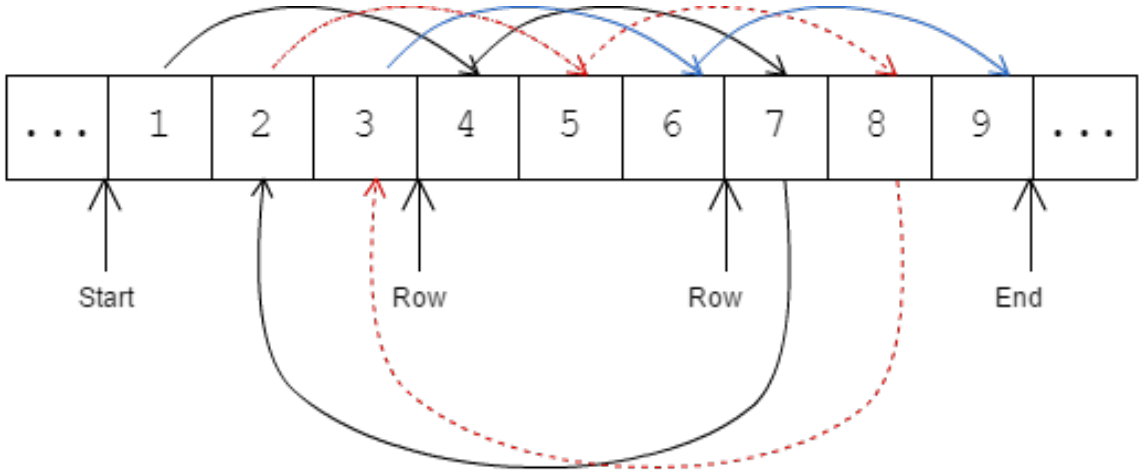


Why we have to iterate over rows - then columns

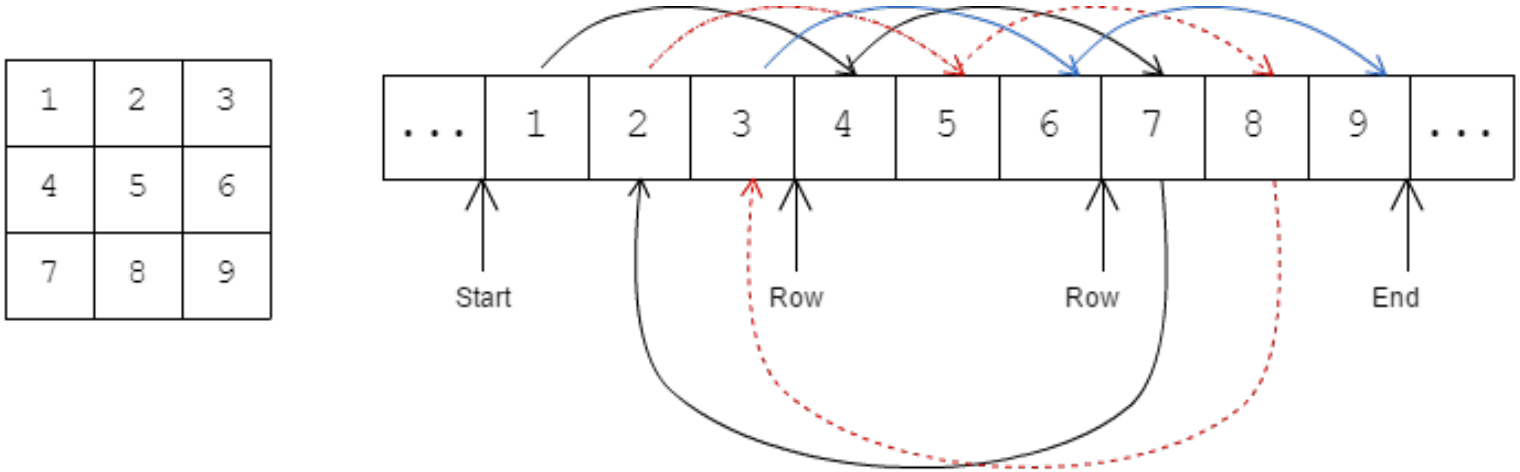
- Position of the row is calculated just one time per row
- Follow the order in which the matrix is stored in memory

Matrix - Memory

1	2	3
4	5	6
7	8	9



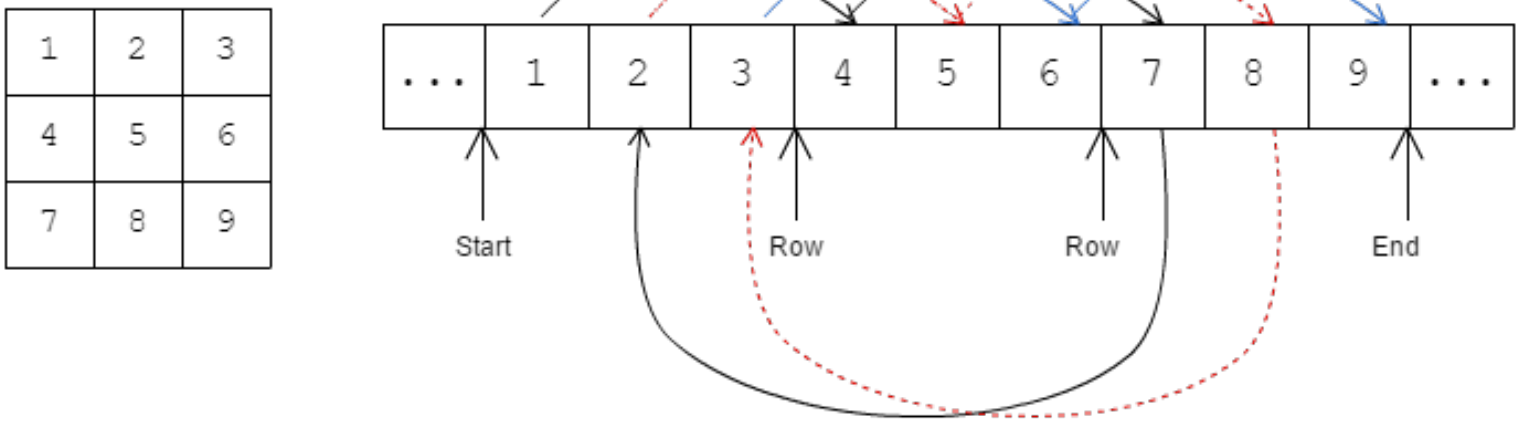
Matrix - Memory



Why we have to iterate over rows - then columns

- Iterating over columns - then rows, has to calculate the position of the row in every iteration

Matrix - Memory



Why we have to iterate over rows - then columns

- Iterating over columns - then rows, has to calculate the position of the row in every iteration
- Increase the complexity!

Matrix

Matrix as a parameter

- `void fun(double array[])`
  Simple way to pass an array as a parameter

Matrix

Matrix as a parameter

- `void fun(double array[])`
  Simple way to pass an array as a parameter
- `void fun(double matrix[][4])`
  Simple way to pass a multidimensional array as a parameter

Matrix

Matrix as a parameter

- `void fun(double array[])`
  Simple way to pass an array as a parameter
- `void fun(double matrix[][4])`
  Simple way to pass a multidimensional array as a parameter
- We MUST provide the number of columns of the matrix to be received

Matrix

Check the code - Write the output

```
int numberOfRows = 5;
int numberOfColumns = 4;

double matrix[numberOfRows][numberOfColumns] =
       {{4.5, 4.0, 4.2, 4.3},
       {5.0, 4.5, 4.4, 4.4},
       {4.9, 3.5, 4.0, 4.1},
       {3.0, 2.5, 3.0, 3.9},
       {2.9, 3.5, 3.4, 3.1}};
printf("%f\n", matrix[0][1]);
printf("%f\n", matrix[(int)matrix[3][0]][0]);
printf("%f\n", matrix[0][3] + matrix[3][0]);

for (int row = 0; row < numberOfRows; row++){
       for (int column = 0; column < numberOfColumns; column++){
              printf("%f ", matrix[row++][++column]);
       }
       printf("\n");
}
```

Matrix

Let's Code!

Code and test the next functions:

- `double avg(double matrix[][4], int rows, int cols)` Average function, which returns the average of all the elements in the matrix
- `double min(double matrix[][4], int rows, int cols)` Min function, which returns the min value of all the values in the matrix
- `void print(double matrix[][4], int rows, int cols)` Print function, which prints all the matrix values per row and col

Outline

Arrays and Pointers

Basics

- Arrays and pointer in C are highly related

Arrays and Pointers

Basics

- Arrays and pointer in C are highly related
- The identifier of an array can be interpreted as a *constant pointer* to the first position of the array

Arrays and Pointers

Basics

- Arrays and pointer in C are highly related
- The identifier of an array can be interpreted as a *constant pointer* to the first position of the array
- Let's try the next code:

```
int abc[] = {1,2,3,4,5,6};
printf("%d\n", *abc);
```

Arrays and Pointers

Basics

- Arrays and pointer in C are highly related
- The identifier of an array can be interpreted as a *constant pointer* to the first position of the array
- Let's try the next code:

```
int abc[] = {1,2,3,4,5,6};
printf("%d\n", *abc);
```

- But the pointing reference of the identifier can not be changed

Arrays and Pointers

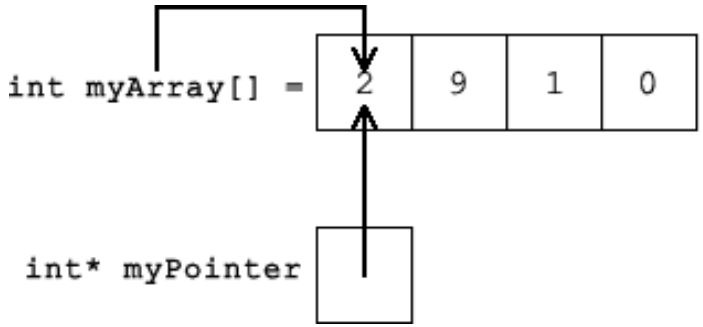
Basics

- Arrays and pointer in C are highly related
- The identifier of an array can be interpreted as a *constant pointer* to the first position of the array
- Let's try the next code:

```
int abc[] = {1,2,3,4,5,6};
printf("%d\n", *abc);
```

- But the pointing reference of the identifier can not be changed
    - We can use a pointer to iterate over an array!

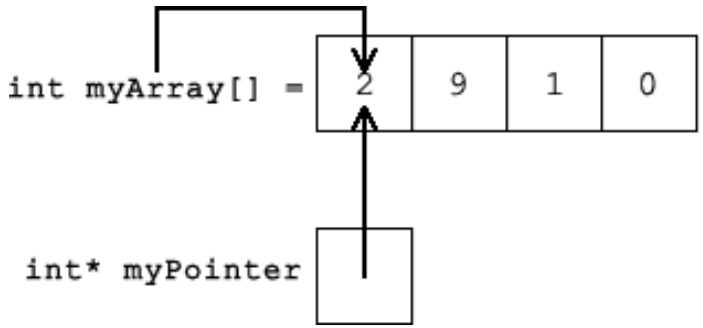
Arrays and Pointers



Basics

- In that way, we can treat the identifier of the array as a pointer to iterate over its element

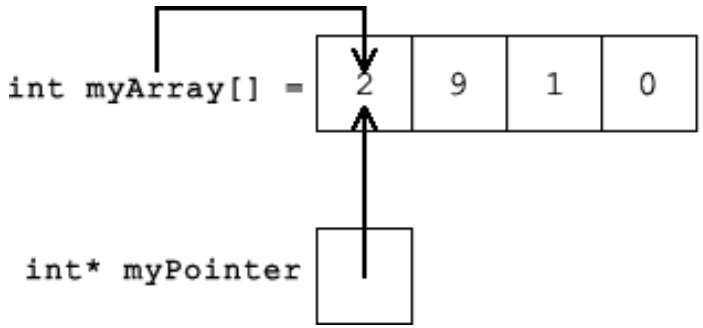
Arrays and Pointers



Basics

- In that way, we can treat the identifier of the array as a pointer to iterate over its element
- `myPointer = myArray`
  The pointer `myPointer` will point to the first position of the array

Arrays and Pointers

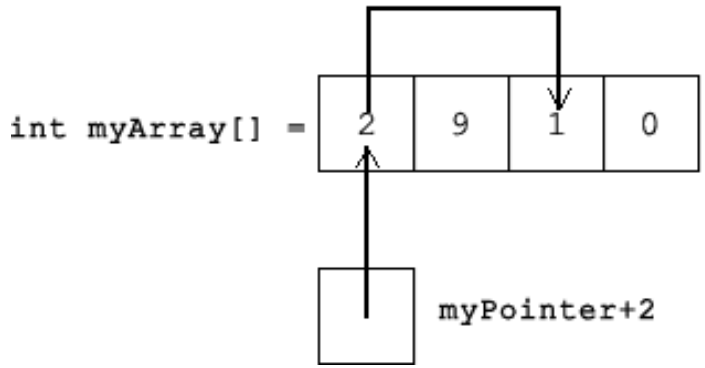


Basics

- In that way, we can treat the identifier of the array as a pointer to iterate over its element
- `myPointer = myArray`
  The pointer `myPointer` will point to the first position of the array
- `*myPointer`
  Will retrieve the element pointed by the pointer

Outline

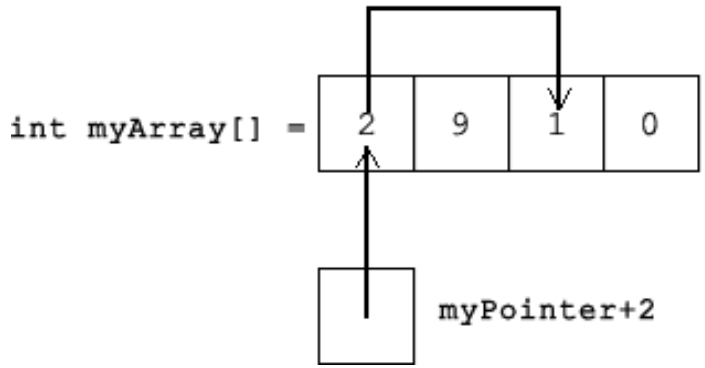
Arrays and Pointers - Iterating



Operations

- `myPointer+2`
  Will get the memory position two spaces right to the addressed one by `myPointer`

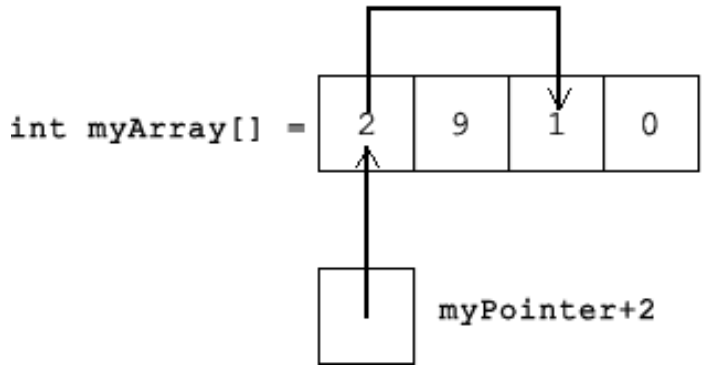
Arrays and Pointers - Iterating



Operations

- `myPointer+2`
  Will get the memory position two spaces right to the addressed one by `myPointer`
- `*(myPointer+2)`
  Will retrieved the value in that memory position

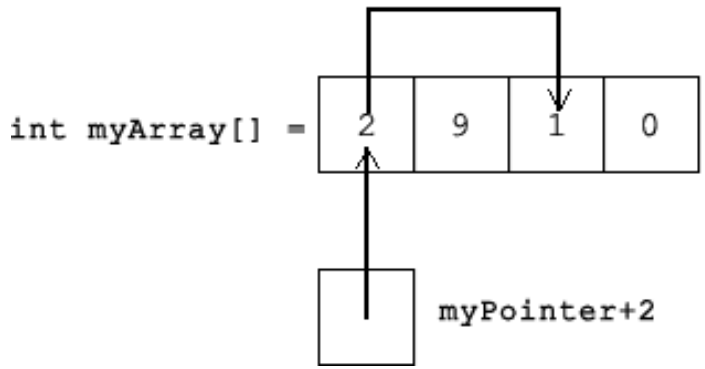
Arrays and Pointers - Iterating



Operations

- `myPointer+2`
  Will get the memory position two spaces right to the addressed one by `myPointer`
- `*(myPointer+2)`
  Will retrieved the value in that memory position
- The pointer won't be displaced and is still pointing to the first position of the array!

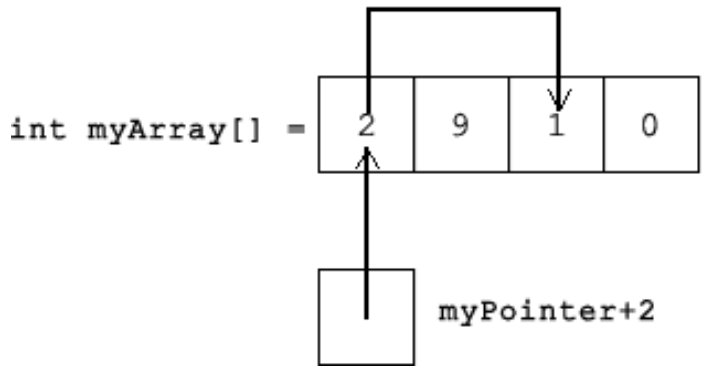
Arrays and Pointers - Iterating



Operations

- `myArray+2` and `myPointer+2` are equivalent taking into account that `myArray` is ALSO a pointer!. Both points to the same memory position

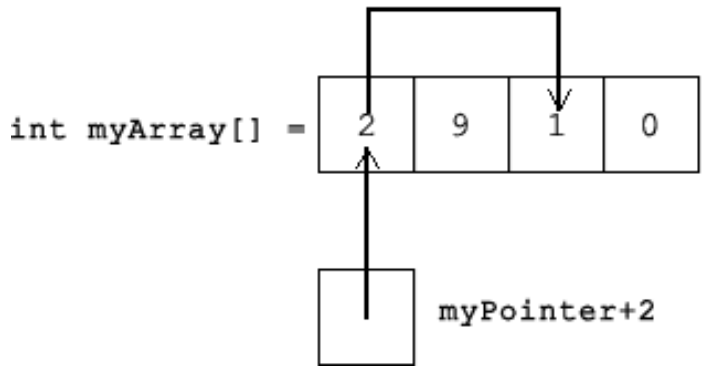
Arrays and Pointers - Iterating



Operations

- `myArray+2` and `myPointer+2` are equivalent taking into account that `myArray` is ALSO a pointer!. Both points to the same memory position
- `myArray[2]`, `*(myArray+2)`, `*(myPointer+2)` are equivalent! Will retrieved the value in that memory position

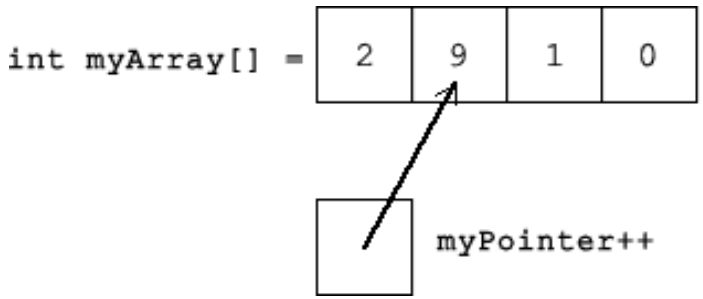
Arrays and Pointers - Iterating



Operations

- `myArray+2` and `myPointer+2` are equivalent taking into account that `myArray` is ALSO a pointer!. Both points to the same memory position
- `myArray[2]`, `*(myArray+2)`, `*(myPointer+2)` are equivalent! Will retrieved the value in that memory position

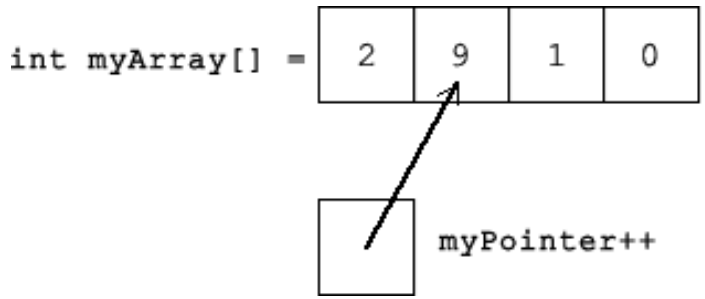
Arrays and Pointers - Iterating



Operations

- `myPointer++`
  Will DISPLACE the pointer to point the next memory address

Arrays and Pointers - Iterating



Operations

- `myPointer++`
  Will DISPLACE the pointer to point the next memory address
- At this point, `*myPointer` and `*myArray` are not equivalent because both are pointing to different positions!
- `*myPointer` is equivalent to `myArray[1]` because the pointer has been displaced one right position

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = myArray;

for (int i = 0; i < size; i++)
{
        printf("%i ", *pointer++);
}

printf("Pointer = %d", *pointer);
```

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = myArray;

for (int i = 0; i < size; i++)
{
        printf("%i ", *pointer++);
}

printf("Pointer = %d", *pointer);
```

- BE CAREFUL!. The pointer can move freely through the memory
- That's DANGEROUS and may cause big damages in the program execution

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = myArray;

for (int i = 0; i < size; i++)
{
        printf("%i ", *(pointer + i));
}

printf("Pointer = %d", *pointer);
```

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = myArray;

for (int i = 0; i < size; i++)
{
        printf("%i ", *(pointer + i));
}

printf("Pointer = %d", *pointer);
```

- What can we say about the pointer?

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = &myArray[2];

for (int i = 0; i < size; i++)
{
        printf("%i ", *pointer++);
}

printf("Pointer = %d", *pointer);
```

Arrays and Pointers

Test the code

```
int size = 5;
int myArray[] = {9, 10, 11, 20, 1};

int* pointer = &myArray[2];

for (int i = 0; i < size; i++)
{
        printf("%i ", *pointer++);
}

printf("Pointer = %d", *pointer);
```

- Are there Syntactical problems?
- What's wrong in the implementation?

Outline

Arrays and Pointers - Matrix

- The same way, we can use a pointer to iterate over a Matrix

Arrays and Pointers - Matrix

- The same way, we can use a pointer to iterate over a Matrix
- We can displace the pointer over the memory allocation where the matrix has been stored!

Arrays and Pointers - Matrix

- The same way, we can use a pointer to iterate over a Matrix
- We can displace the pointer over the memory allocation where the matrix has been stored!
- There is to ways to displace a pointer over a matrix

Arrays and Pointers - Matrix

- The same way, we can use a pointer to iterate over a Matrix
- We can displace the pointer over the memory allocation where the matrix has been stored!
- There is to ways to displace a pointer over a matrix
  - Displacing the pointer just 1 memory position per iteration

Arrays and Pointers - Matrix

- The same way, we can use a pointer to iterate over a Matrix
- We can displace the pointer over the memory allocation where the matrix has been stored!
- There is to ways to displace a pointer over a matrix
  - Displacing the pointer just 1 memory position per iteration
  - Displacing the pointer every time we move to another row

Arrays and Pointers - Matrix

Test the code - Displacing per iteration

```
double matrix[5][4] =
{
        {4.5, 4.0, 4.2, 4.3},
        {5.0, 4.5, 4.4, 4.4},
        {4.9, 3.5, 4.0, 4.1},
        {3.0, 2.5, 3.0, 3.9},
        {2.9, 3.5, 3.4, 3.1}
};

double *pointer = matrix;

for(int i=0; i<20; i++)
{
        printf("%f ", *pointer++);
}
```

Arrays and Pointers - Matrix

Test the code - Displacing per row

```
double matrix[5][4] = {
        {4.5, 4.0, 4.2, 4.3},
        {5.0, 4.5, 4.4, 4.4},
        {4.9, 3.5, 4.0, 4.1},
        {3.0, 2.5, 3.0, 3.9},
        {2.9, 3.5, 3.4, 3.1}
};
double *pointer;

for(int i=0; i<5; i++){
        pointer = &matrix[i];
        for(int j=0; j<4; j++)
                printf("%f ", *(pointer+j));
}
```

Outline

Basics

- Real life programs uses input arguments to parametrise its executions

Basics

- Real life programs uses input arguments to parametrise its executions
- E.g: `gcc main.c`
  The `gcc` program receives the name of the file to be compiled

Basics

- Real life programs uses input arguments to parametrise its executions
- E.g: `gcc main.c`
  The `gcc` program receives the name of the file to be compiled
- `ping www.google.com`
  The `ping` program receives the address of the host to ping

Basics

- Real life programs uses input arguments to parametrise its executions
- E.g: `gcc main.c`
  The `gcc` program receives the name of the file to be compiled
- `ping www.google.com`
  The `ping` program receives the address of the host to ping
- `copy myFile.doc folder\myCopiedFile.doc`
  The `copy` program receives the origin path and the destination path as parameters

Outline

Parameters in the main function

```
int main(int argc, char **argv)
```

Composition

- `int argc`
  Number of arguments - It is provided by the operating system during the execution

Parameters in the main function

```
int main(int argc, char **argv)
```

Composition

- `int argc`
  Number of arguments - It is provided by the operating system during the execution
- `char **argv`
  Pointer to an array of pointers pointing the argument values

Parameters in the main function

```
int main(int argc, char **argv)
```

Composition

- `int argc`
  Number of arguments - It is provided by the operating system during the execution
- `char **argv`
  Pointer to an array of pointers pointing the argument values
- All the arguments are interpreted as strings

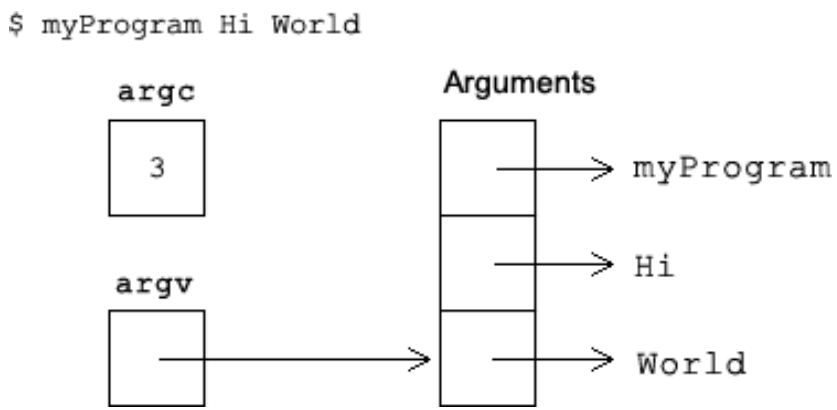
Parameters in the main function

```
int main(int argc, char **argv)
```

Composition

- `int argc`
  Number of arguments - It is provided by the operating system during the execution
- `char **argv`
  Pointer to an array of pointers pointing the argument values
- All the arguments are interpreted as strings
- The first argument of every program (`argv[0]` is the name of the executable file

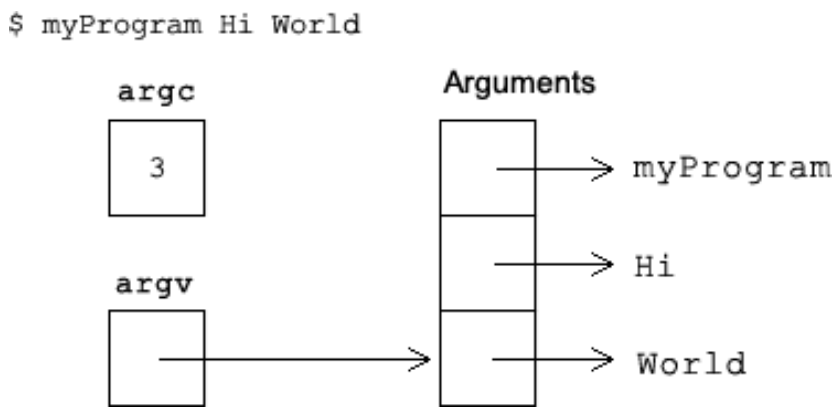
Parameters in the main function



Operations

- Don't be scared of double pointers!. We can access to a value like a simple array
  E.g: `argv[0]` $\rightarrow$ `myProgram`

Parameters in the main function



Operations

- Don't be scared of double pointers!. We can access to a value like a simple array
  E.g: `argv[0]` → `myProgram`
- Also iterating over the arguments array with help of the pointer
  E.g: `*(argv + 1)` → `Hi`
- ... or displacing the pointer → `*(argv++)`

Parameters in the main function

Test the code!

```
#include <stdio.h>

int main(int argc, char **argv)
{
    while(argc--)
        printf("%s\n", argv[argc]);
}
```

- Compile and run the program with multiple parameters

Parameters in the main function

Test the code!

```
#include <stdio.h>

int main(int argc, char **argv)
{
    while(argc--)
        printf("%s\n", argv[argc]);
}
```

- Compile and run the program with multiple parameters
- E.g: `program.exe Hi world im ok`

Parameters in the main function

Now you have a basic knowledge about multidimensional arrays, pointers, strings manipulation and the arguments of the main function!.

Parameters in the main function

Let's code!. We want to build a Dictionary.

- The program should receive a word in spanish in lowercase as an argument, and the language in which the word should be translated
- E.g: `dictionary.exe casa german`
- The translation should be printed. Print an error message if the word is not found or if the translation language is not supported
- Store 15 words in a matrix with their respective translations in English, Spanish, German and Italian
- The program can also receive the `-help` parameter. When it is invoked, it will print the supported languages and all the available words.
- E.g: `dictionary.exe -help`