

Arrays

Topics

- Arrays
 - Array Parameters
 - Typical Array Operations
- main – closer look

Arrays

- We typically encounter groups similar items
 - Eggs in an egg carton
 - Apartments in an apartment building
- It is easier to manage the group rather than each individual alone
- Each object in the set is the same
- The overall set has a size

Arrays

- An array is a collection of values of all identical type
- Logically (between the programmer and the outside world) these values are also related to each other.
- The collection has a variable name
- Each item in the collection has a subscript that defines its position within the collection

Example

- An array for assignment 1 grades: will be float, and each entry represents someone's grade of assignment 1
- The first entry represents the grade of the first student in the roster.
- The compiler will let you assign, for example, the second entry to the balance of someone's bank account – they are all float numbers. This is a logical error on your part.

Array Declaration

- Syntax:
`type arrayname[size] ;`
- `type` referred to as the data type for all array elements
- `arrayname` is the variable name for the entire collection
- `size` is the number of elements allowed in the collection
 - indexes from 0 to `size-1`

Arrays

- **Example:**

- `int grades[5];`

Arrays

- **Example:**

- `int grades[5];`

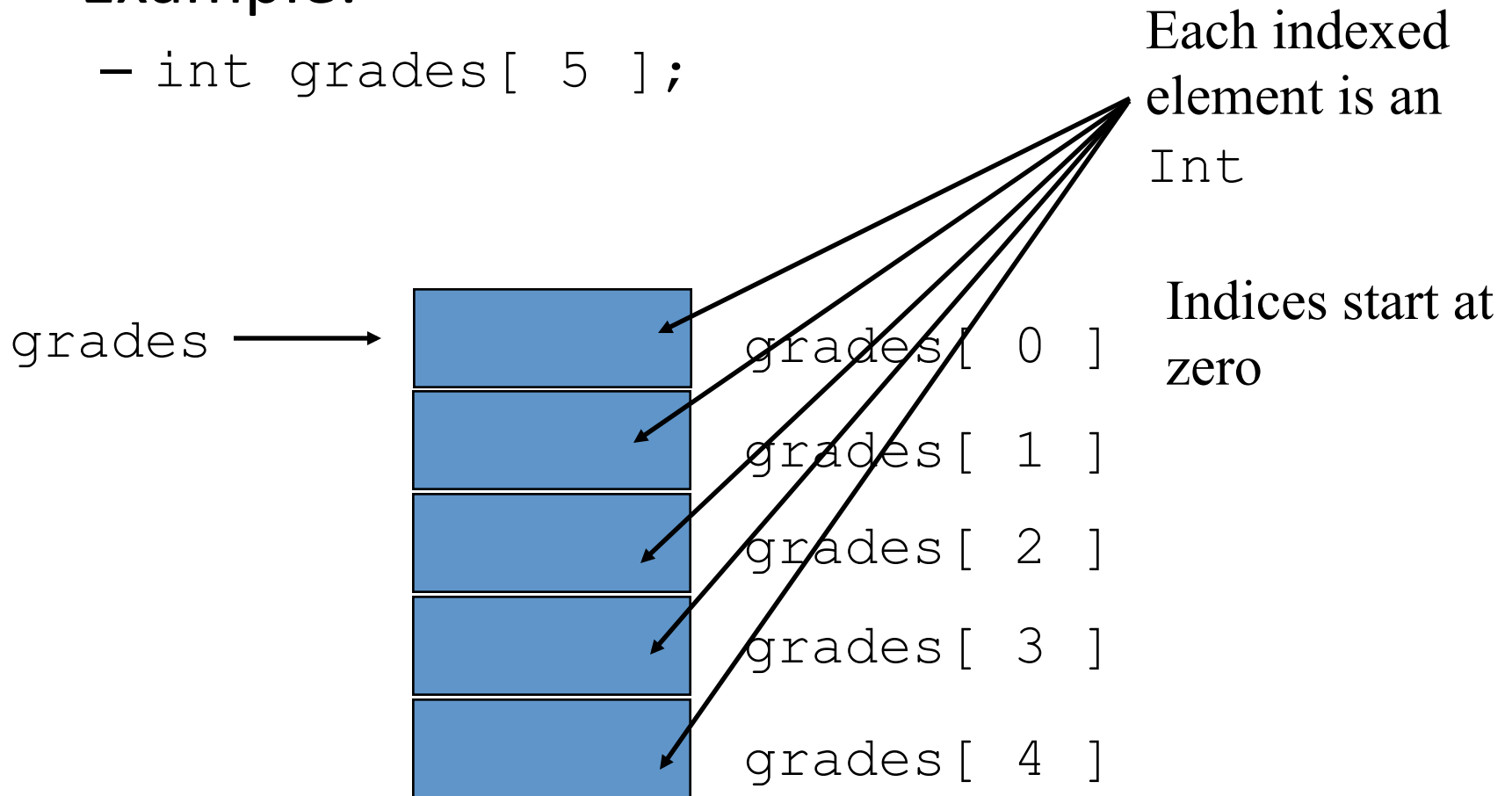
Here we have the variable name `grades`, and the size of the array is 5. The first element of `grades` is located at location 0, the second element at location 1, ... and the 5th element is at location (in programming we call it index) 4.

0 to 4 is 5 elements.

Arrays

- Example:

```
– int grades[ 5 ];
```



Exercise

- `int x[7]`
 - How many elements are in `x`?
 - What is the data type of `x[2]`?
 - What is the index of the last element in `x`?
 - What is the index of the first element in `x`?

Exercise - Answers

- `int x[7]`
 - How many elements are in `x`? 7
 - What is the data type of `x[2]`? `int`, as is the data type of every element in `x`
 - What is the index of the last element in `x`? 6
 - What is the index of the first element in `x`? 0
- We pay special attention to the address of the last element. Why? We'll learn in a few slides

Arrays

- Arrays are an ordered list

`grades[1]` precedes `grades[10]`

`grades[10]` precedes `grades[11]`

- Arrays are stored contiguously in one block
- Each index is a literal in its own right, e.g. `grades[n]`, `n` can be a variable calculated through a math expression. This is where the fun is.
- `[]` Is used to declare and access arrays

Arrays

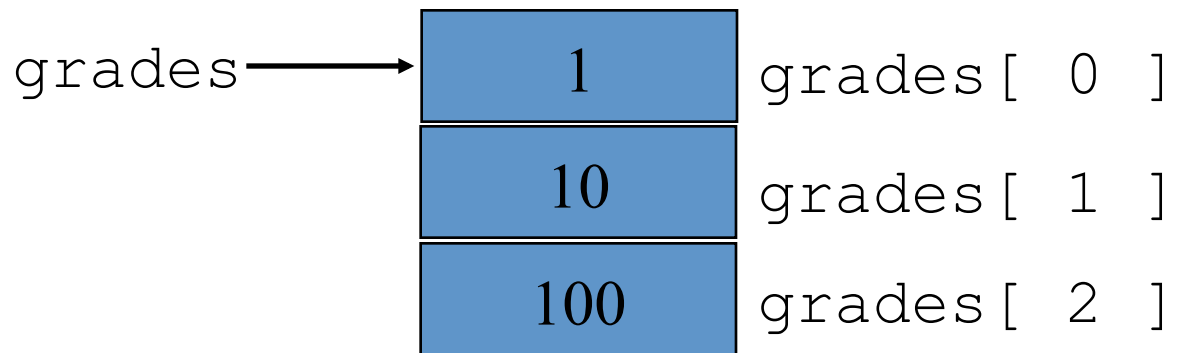
- Example:

```
int grades[3]; //declare
grades[0] = 1; // assign first element to 1
grades[1] = 10; // assign second to 10
grades[2] = 100; // assign third to 100
```

Arrays

- Example:

```
int grades[3];  
grades[0] = 1;  
grades[1] = 10;  
grades[2] = 100;
```

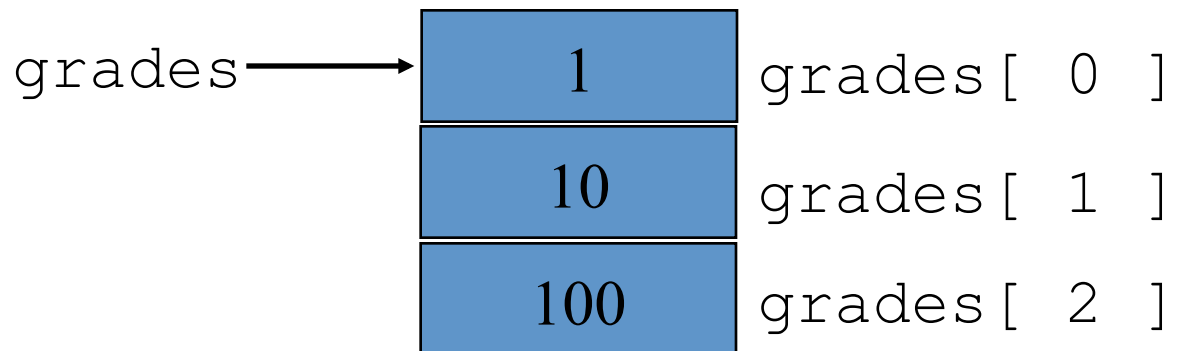


Arrays

- Example:

```
int grades[3];  
grades[0] = 1;  
grades[1] = 10;  
grades[2] = 100;
```

Don't confuse declaration
syntax with element access



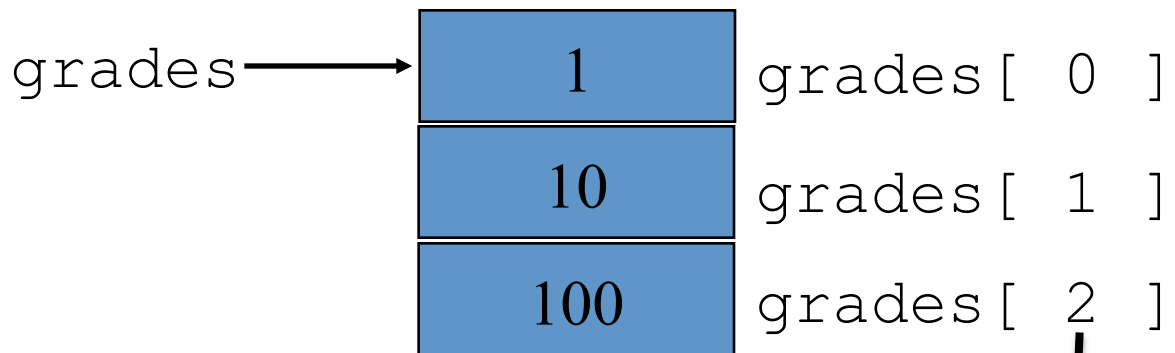
Arrays

- Example:

```
int grades[3];  
grades[0] = 1;  
grades[1] = 10;  
grades[2] = 100;
```

Don't confuse declaration
syntax with element access

0, 1 and 2 are
the indices.
Each of those
is a literal



Array Initialization


- Like other variables, arrays can be initialized when they are declared
- Generally, it's a good idea to define constants for array size

```
const int SIZE=3;  
int grades[SIZE];  
grades[0] = 1;  
grades[1] = 10;  
grades[2] = 100;
```

Array Initialization

- Alternative ways to initialize:

```
const int SIZE=3;    const int SIZE=3;
int grades[SIZE];    int grades[SIZE]={1,10,100};
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```



Array Iteration

- `for` loops are often used with arrays
 - In the loop, Array index (`i` below) should not be a fixed constant

```
const int SIZE=3;
int a[SIZE]={1,10,100}; //initialize here
                        // helps in testing

for (int i=0; i<SIZE; i++) {
    printf( "a[%d] = %d\n", i, a[i] );
}
```

Important Considerations

- Don't exceed array bounds. It is a fatal run-time error to do so...
- Typically, bounds errors come on the last iteration going over the edge
- You are forewarned!

Arrays Demo!

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

```
void sort( int a[], int size );
void dump( int a[], int size );
```

```
int main( ) {
```

/* An array is a whole set of like values. When you declare an array, you must use square brackets and provide a fixed, constant size. You cannot use a variable to declare its size. All of the elements of the array are the same; they will be int

```
*/
    int array[ 10 ];
    int amount;
    int data;
    int i;

    printf( "How many values do you want to enter? (<10,please)" );
    scanf( "%d", &amount );

    /* Arrays lead to lots of looping as you walk the set of values.
    */
    for ( i = 0; i < amount; i++ ) {
        printf( "Enter a value: " );
        scanf( "%d", &data );
        array[ i ] = data;
    }
    /*
```

You can pass a whole array to a function. When you do, you must also send a companion parameter which states how big the array is, because the function has no other way of knowing unless you supply

it.

```
    */
    sort( array, amount );
    dump( array, amount );
    return( 0 );
}

void sort( int a[], int size ) {
    int i, j, temp;
    for ( i = 0; i < size; i++ ) {
        for ( j = 0; j < size; j++ ) {
            //Get one pair of values ordered correctly...
            if ( a[i] < a[j] ) {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
}

void dump( int a[], int size ) {
    int i;
    for ( i = 0; i < size; i++ ) {
        printf( "a[ %d ] = %d\n", i, a[i] );
    }
}
```

Summarizing Arrays Demo!

- Arrays let you work with groups of data
- Carefully track array size!
- Don't reinvent the wheel – use code from the demo for future with adjustments as needed

Arrays As Function Arguments

- Like any other *value*, array **elements** can be passed to functions

```
void print_value( int i ); //some function

const int SIZE=3;
int a[SIZE]={1,10,100}; //array

for (int i=0; i<SIZE; i++) {
    print_value( a[ i ] );
//a[i] is an int and can be passed to print_value
}
```

Arrays As Function Arguments

- The whole array can also be a parameter in a function
- Arrays are passed to functions as array arguments
- If a function changes element value, these changes will be seen by the caller

Arrays As Function Parameters

- **Parameter Syntax:** *type name[]*
 - *Need [] to stress its an array*
- **Argument Syntax:** *name*
 - *Prototype knows name is an array*
 - *Hence do not use []*

Arrays As Function Parameters

- Parameter Syntax: *type name[]*
- Argument Syntax: *name*



```
void fill_up( int items[], int length );
```

```
const int SIZE=3;  
int a[SIZE]={1,10,100};
```

```
fill_up( a, SIZE );
```

Arrays As Function Parameters

- Parameter Syntax: *type name[]*
- Argument Syntax: *name*

```
void fill_up( int items[], int length );
```

```
const int SIZE=3;  
int a[SIZE]={1,10,100};
```

```
fill_up( a, SIZE );
```



No Index Value

Observations

- Since the array parameter definition lacks array size value, it is always A good idea to pass the size of the array as an extra argument

Observations

- Since the array parameter definition lacks array size value, it is a good idea to pass the size of the array as an extra argument

```
void fill_up( int items[], int length );
```

```
int a[5], b[10];
```

```
fill_up( a, 5 );
```

```
fill_up( b, 10 );
```

Observations

- Since the array parameter definition lacks array size value, it is always a good idea to pass the size of the array as an extra argument


```
void fill_up( int items[], int length );
```

```
int a[5], b[10];
```

```
fill_up( a, 5 );
```

```
fill_up( b, 10 );
```

Function can be called
with arrays of various
sizes



Observations

- When arrays are passed to functions, elements changed by the function are visible to the caller
- This is kinda pass-by-reference
 - No copies of the individual elements are made
 - Changes to any elements will be permanent, hence seen by the caller
 - We will cover pass by reference in the next unit

`const` Array Arguments

- If you know the function will not change the array values, use `const` modifier

`const` Array Arguments

- If you know the function will not change the array values, use `const` modifier

```
void print(const int items[],int length);
```

Typical Array Operations

- Searching
- Sorting

Revisiting Arrays Demo!

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

```
void sort( int a[], int size );
```

```
void dump( int a[], int size );
```

```
int main( ) {
```

```
/* An array is a whole set of like values. When you declare an array, you must use square brackets and provide a fixed, constant size. You cannot use a variable to declare its size. All of the elements of the array are the same; they will be int
```

```
*/
    int array[ 10 ];
    int amount;
    int data;
    int i;
```

```
    printf( "How many values do you want to enter? (<10,please)" );
    scanf( "%d", &amount );
```

```
/* Arrays lead to lots of looping as you walk the set of values.
```

```
*/
    for (i = 0; i < amount; i++) {
        printf( "Enter a value: " );
        scanf( "%d", &data );
        array[ i ] = data;
    }
```

```
/*
```

You can pass a whole array to a function. When you do, you must also send a companion parameter which states how big the array is, because the function has no other way of knowing unless you supply

it.

```
*/
```

```
sort( array, amount );
```

```
dump( array, amount );
```

```
return( 0 );
```

```
}
```

```
void sort( int a[], int size ) {
```

```
    int i, j, temp;
```

```
    for (i = 0; i < size; i++) {
```

```
        for (j = 0; j < size; j++) {
```

```
            //Get one pair of values ordered correctly...
```

```
            if (a[i] < a[j]) {
```

```
                temp = a[i];
```

```
                a[i] = a[j];
```

```
                a[j] = temp;
```

```
            }
```

```
        }
```

```
    }
```

```
void dump( int a[], int size ) {
```

```
    int i;
```

```
    for (i = 0; i < size; i++) {
```

```
        printf( "a[ %d ] = %d\n", i, a[i] );
```

```
    }
```

```
}
```

Arrays Summary

- Arrays
 - Array Parameters
 - Typical Array Operations
- Read the book section 24

Important Reminders

- When declaring an initializing arrays:
 - Size may be omitted
 - Size, if provided may not have to match the number of elements initialized
 - If you provide a size, but initialize less elements, the rest will default to 0 or 0.0
- C will go past an array element's last without throwing a runtime exception; hence you could be reading or working with garbage
- In fact C will go before arrays first element without throwing a runtime exception (see p. 182)
- Array index must be int

main – closer look...

`main ()` – Officially

- Officially, `main ()` can accept parameters:

```
int main( int argc, char * arg[])
```

 - You know `argc` is an `int`
 - But what is `char * arg[]`?
 - It's an array of type `char`
 - We will cover `char` and strings in a later unit

main Arguments

- Although many of our examples up until now have avoided it, you can pass data back and forth to your `main ()` function
 - Data returned from the `main ()` function is passed back to the operating system
 - Data sent to the `main ()` function comes from the operating system

Returning From `main ()`

- Officially, `main ()` is supposed to return `int`

```
int main( )
```

- By convention, a return value of zero indicates “everything is OK”
- Any other value returned would be interpreted by the operating system to indicate some kind of error
 - This an FYI

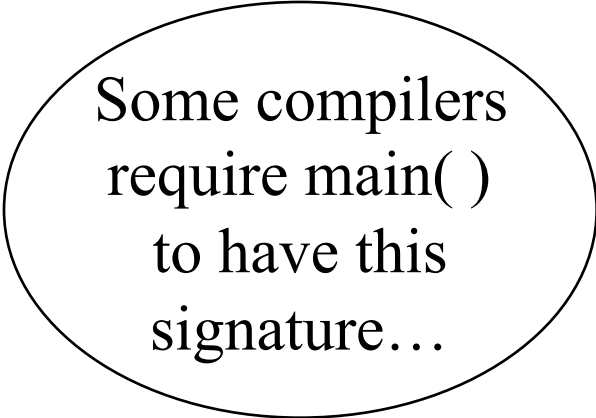
FYI- Passing Data To `main ()`

- Officially, `main ()` can accept parameters
- These parameters come from the command-line which is used to call your program
- `main ()` gets passed an array of `cstring` known as `argv`
 - These are text from the actual command line broken up into space-separated words
- `main ()` gets an `int` known as `argc`
 - This value tells you how big the array of `argv` is

Passing Data To `main ()`

- Officially, `main ()` Can Accept Parameters

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



Some compilers
require `main()`
to have this
signature...

Passing Data To `main ()`

- Officially, `main ()` Can Accept Parameters
 - `int main(int argc, char * argv[])`

Some compilers
require `main()`
To have this
signature...

Visual studio is
fairly sloppy on
this point...

FYI - Background

- Whether you start your program in visual studio or by double-clicking in windows explorer or by typing into A DOS box, you always get the chance to send command-line arguments

FYI - Examples

- For A Program Named `foo.exe`:
DOS> `foo param1 param2`

Examples

- For A Program Named `foo.exe`:
DOS> `foo param1 param2`

`argc`

`argv`

Examples

- For A Program Named `foo.exe`:
DOS> `foo param1 param2`

`argc` 3

`argv`

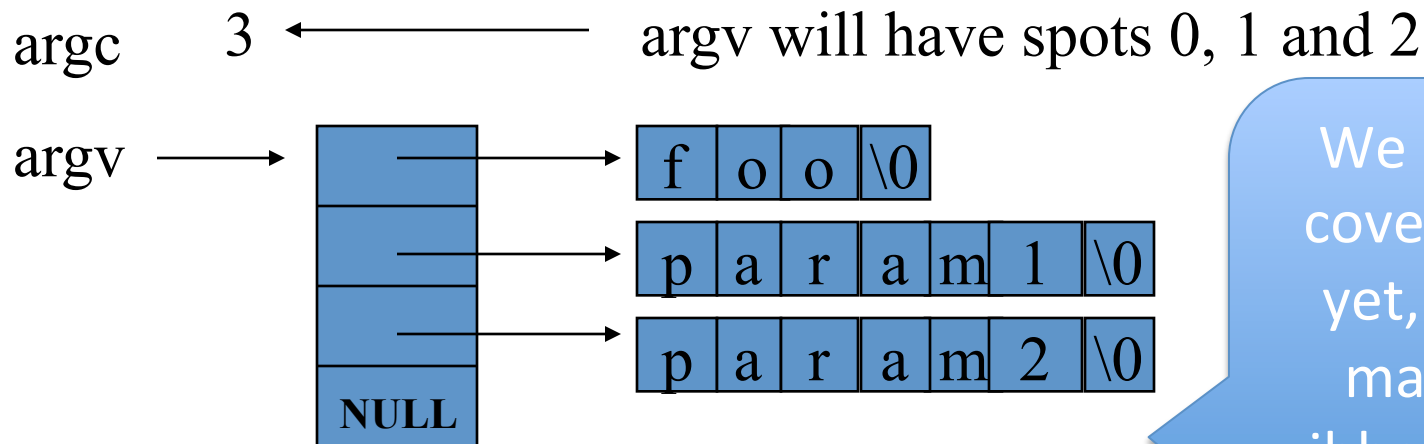
Examples

- For A Program Named `foo.exe`:
`DOS> foo param1 param2`

`argc` 3 ← `argv` will have spots 0, 1 and 2
`argv`

Examples

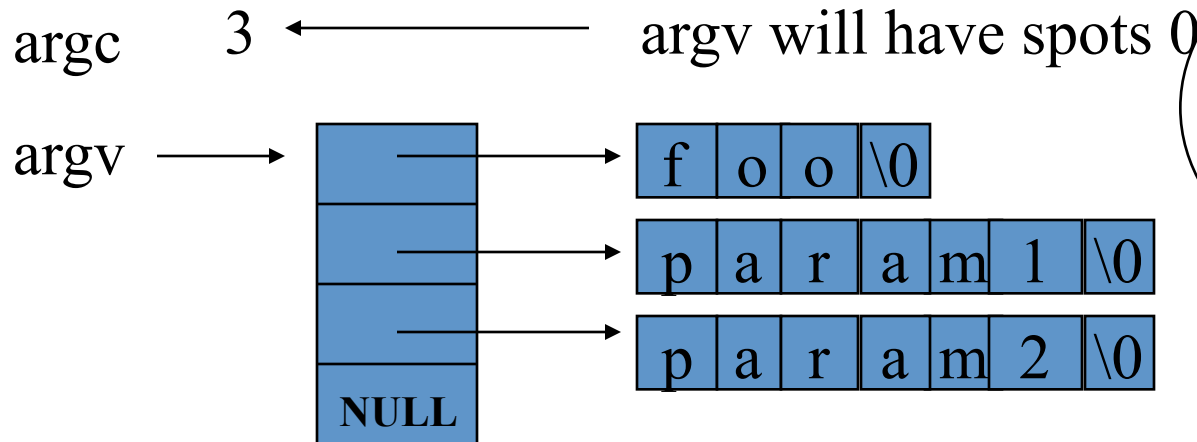
- For A Program Named `foo.exe`:
`DOS> foo param1 param2`



We did not cover arrays yet, so this may look gibberish. You may ignore it

Examples

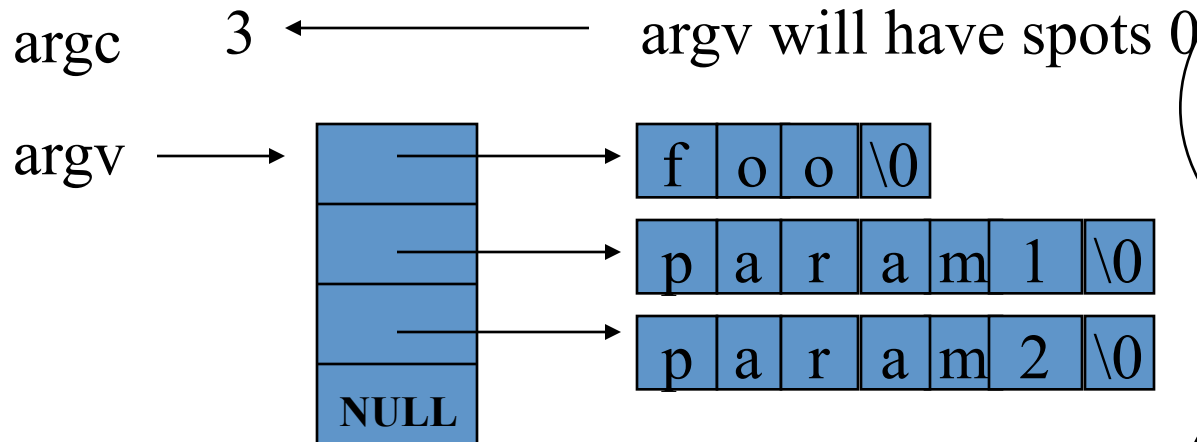
- For A Program Named `foo.exe`:
`DOS> foo param1 param2`



The program name
is always the
first argv value...

Examples

- For A Program Named `foo.exe`:
`DOS> foo param1 param2`



The program name
is always the
first argv value...

As a result, you
will never have an
empty argv...

FYI Demo

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

```
int main( int argc, char ** argv ) {  
    int counter;  
    printf( "Here are your command line arguments...\n" );  
    printf( "When you run from inside Visual Studio,\n" );  
    printf( "there won't be any arguments besides the\n" );  
    printf( "program name, unless you change your project\n" );  
    printf( "settings, as shown in the powerpoint slides for this unit...\n" );  
    /* The command line arguments are passed as array of c-string */  
    for (counter = 0; counter < argc; counter++) {  
        printf( "%d. %s\n", counter, argv[counter] );  
    }  
    getchar();  
    return( 0 );  
}
```

FYI - Setting Command-Line Arguments

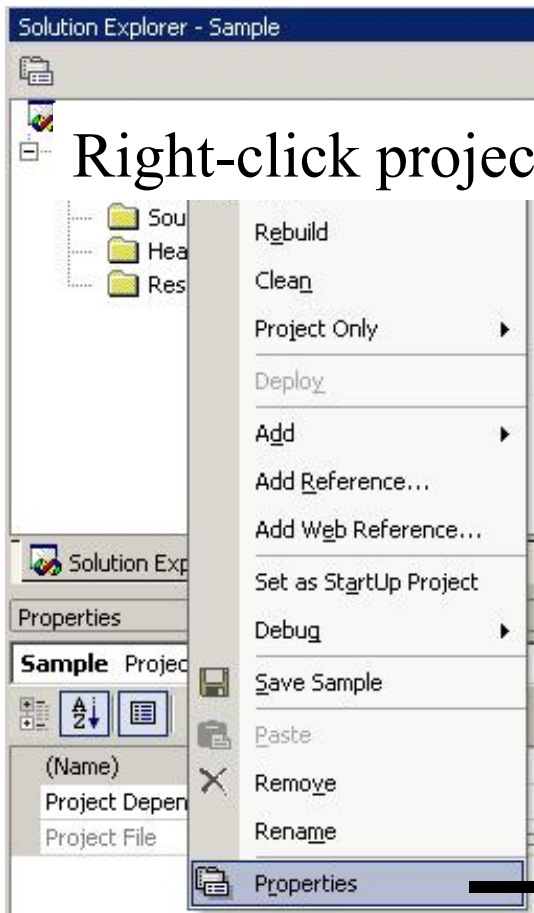
- When working with the DOS> prompt, you type in the command-line arguments yourself

Setting Command-Line Arguments - FYI

- When working with visual studio, whether from debug or release versions, you can coax visual studio into supplying command-line arguments to your programs

Setting Command-Line Arguments - FYI

- Visual Studio .NET



Right-click project folder, then Properties

FYI

