

# CSE 303

## Concepts and Tools for Software Development

Magdalena Balazinska  
Winter 2007  
Lecture 9 – Arrays and Strings

# About hw3 and hw4

- Assignments 3 and 4 are the **most difficult** assignments this quarter
- Programming in C takes longer than programming in Java because **debugging is more difficult**
  - Debugging is an important skill to acquire
  - The only way to learn is really to spend the time
- Please start early and plan to spend time debugging
- **Always write as little code as possible and test often**

# Where We Are

- **Previous two lectures**
  - Introduction to C and pointers
- **Today**
  - Arrays
  - Strings
  - Command line arguments

# Arrays in C

- An array is a “group of memory locations related by the fact that they all have the same name and the same type”

- Example:

```
int i;  
int c[3];  
int j=23;  
for (i=0; i<3; i++) {  
    c[i]=0;  
}
```

Stack  
(one possible arrangement)

i	3
c[2]	0
c[1]	0
c[0]	0
j	23

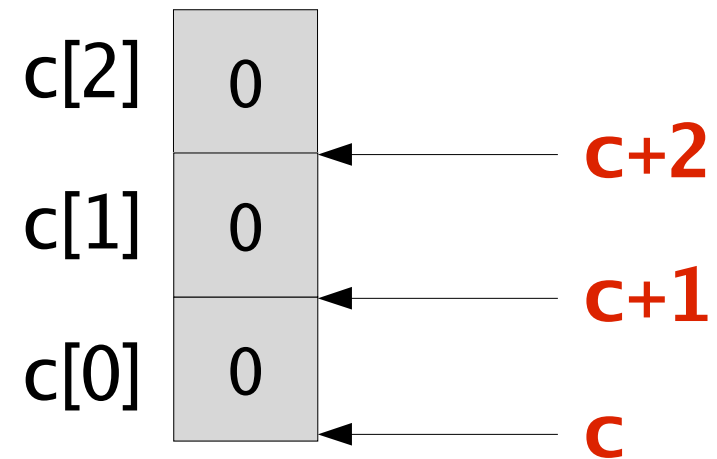
↑  
Increasing  
addresses

# Arrays in C

- **Elements of an array**
  - Are a set of ordered data items
  - Occupy contiguous memory locations
- **Checking array bounds**
  - The compiler does not check array bounds
  - There are no runtime checks either
  - The program must explicitly remember the array size and must check bounds
  - Array out-of-bounds errors can often go undetected for a long time!

# Pointer Arithmetics

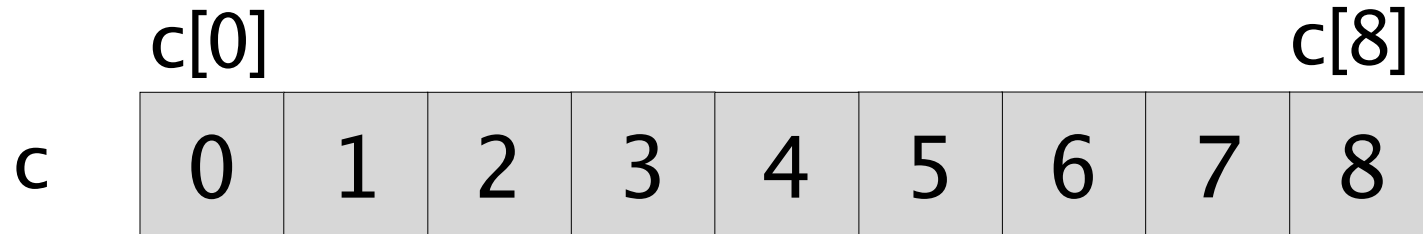
```
int c[3];  
for (i=0; i<3; i++) {  
    printf( "%d\n", c[i] );  
    printf( "%d\n", *(c+i) );  
}
```



Array name corresponds to *address* of start of array

Example: `simple-array.c`

# Example 1



```
c[0] = 13;
```

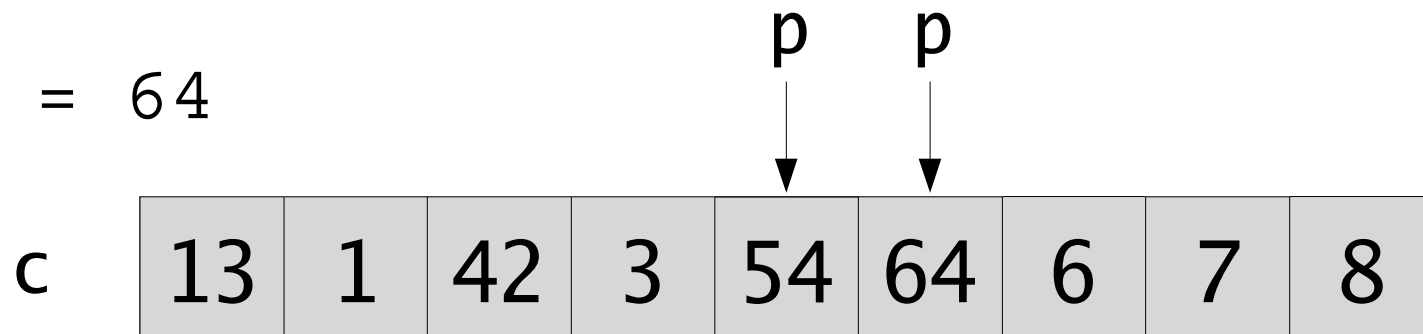
```
c[2] = 42;
```

```
int *p = &c[4];
```

```
*p = 54;
```

```
p++;
```

```
*p = 64
```



# Example 2

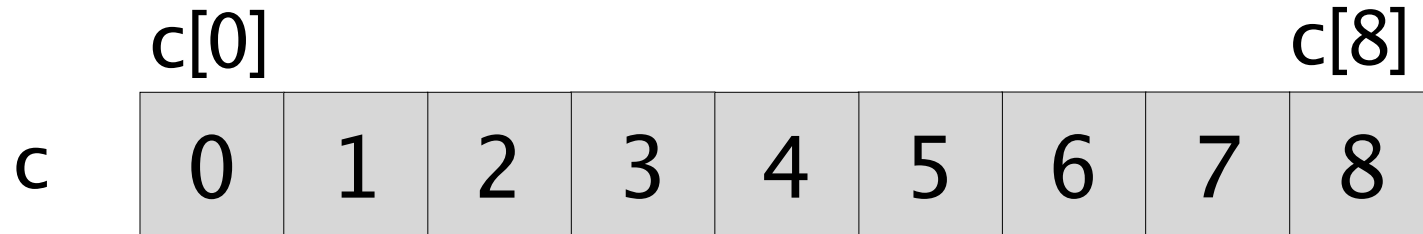
	c[0]							c[8]	
c	0	1	2	3	4	5	6	7	8

```
int i;  
for (i = 0; i <= 8; i++ ) {  
    c[i] = c[i] + 10;  
}
```

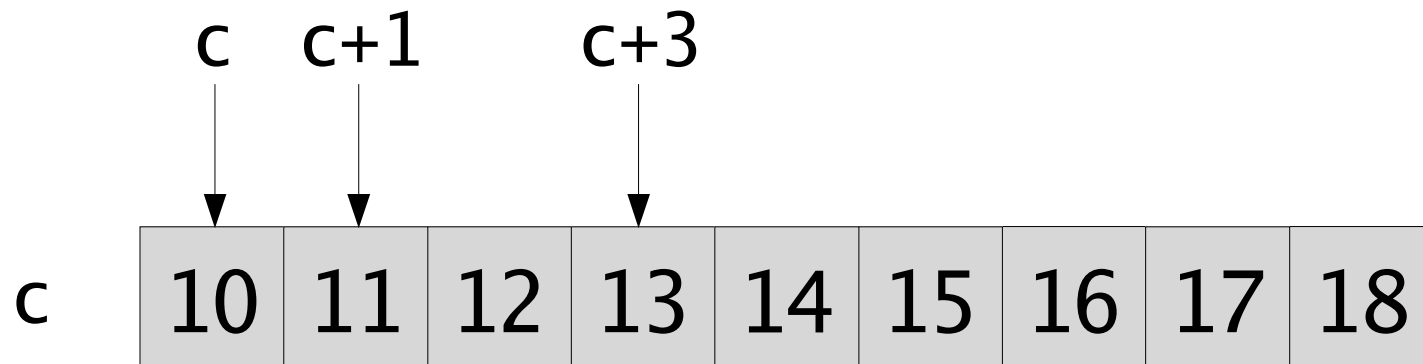
c	10	11	12	13	14	15	16	17	18
---	----	----	----	----	----	----	----	----	----



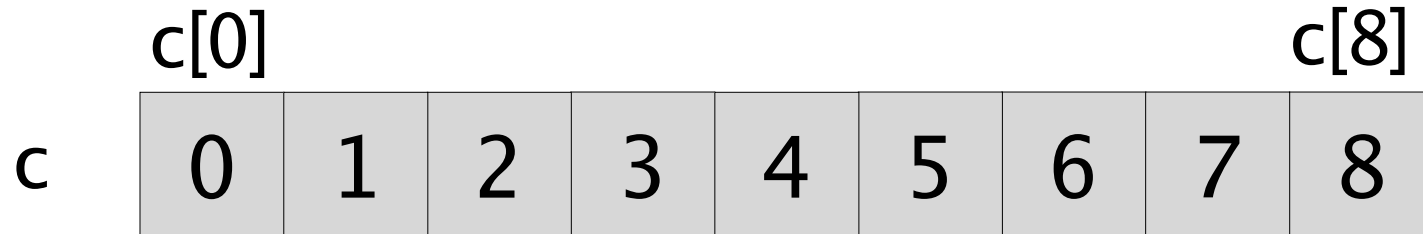
# Example 3



```
int i;  
for (i = 0; i <= 8; i++ ) {  
    *(c+i) = *(c+i) + 10;  
}
```



# Example 4



```
int *p;
```

```
for (p = c; p <= c+8; p++ ) {
```

```
    *p = *p + 10;
```

```
}
```



# Examples

See `array.c` to experiment with examples 1 through 4

# Passing Arrays to Functions

```
// To pass array to function
// Indicate name without brackets
modify(c, size);

// Function definition is then
void modify(int c[], int size) {
    // Modification visible to caller
    c[i] = 3;
}
```

# Passing Arrays to Functions

Because the array name is the address of the beginning of the array, the following is also allowed:


```
void modify(int *c, int size) {  
    // Modification visible to caller  
    c[i] = 3;  
}
```

Also see `array.c` for simple examples

# Multi-Dimensional Arrays

Rows Columns

```
int c[2][3];
```



```
int i, j;
```

```
for ( i = 0; i < 2; i++ ) {  
    for ( j = 0; j < 3; j++ ) {  
        c[i][j] = 0;  
    }  
}
```

# Passing Multi-Dimensional Arrays to Functions

```
void modify(int rows, int cols,  
           int c[][cols]) {  
    c[2][3] = 3;  
}
```

- Compiler needs to find address of element given subscripts
- So compiler needs to know **nb columns per row**

Example: `multi-array.c`

# Strings

- A string is an **array of characters** plus a special string termination character called the **null character**
- **Null character**
  - Denoted with `'\0'`
  - Character with ASCII value 0
- Size of array must include space for `'\0'`
- We can do same operations as on array!
- Common bug: **overwrite `'\0'`**



# Declaring and Initializing Strings

```
int max_length = 20;
```

```
char str[max_length];
```

```
// Copy the string "Hello world" into str
```

```
// We must make sure that str has enough room
```

```
strncpy(str, "Hello world", max_length);
```

```
printf("str is %s", str);
```

# Declaring and Initializing Strings

```
char str[] = "Hello world";  
printf("str is %s", str);  
// Will print: Hello world
```

```
char str[20] = "Hello world";  
printf("str is %s", str);  
// Will print: Hello world
```

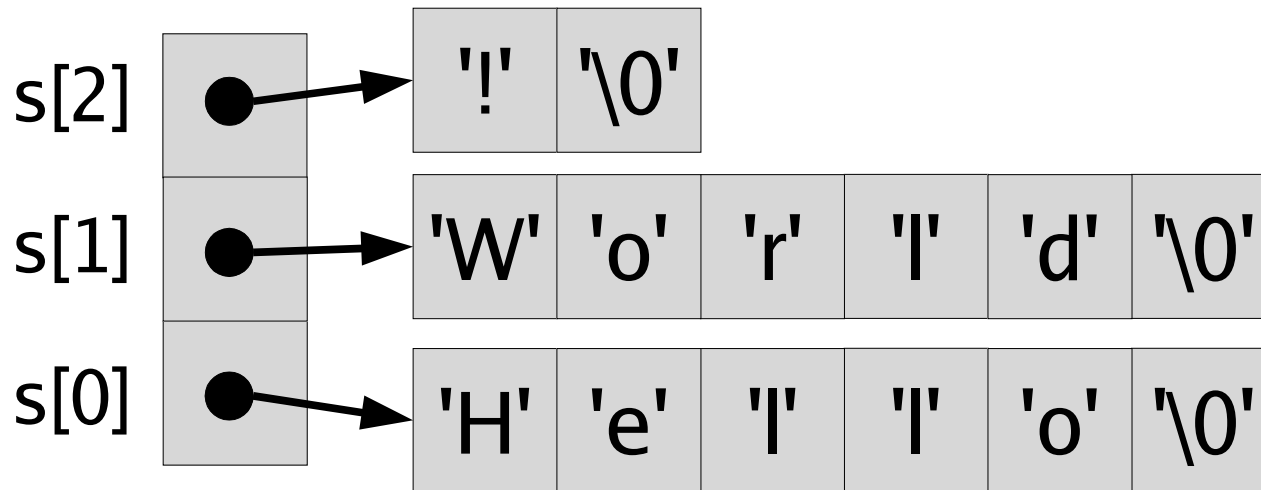
# Standard C Library (string.h)

Various utility functions to operate on strings (p. 470)

```
char s1[20] = "blue ";
char s2[] = "gray";
// Append s2 to s1
// We must make sure s1 has enough room
strcat(s1,s2);
// Better to use strncat (see hw3)
// Compare s1 and s2
int comparison = strcmp(s1,s2);
// Can also use strncmp
```

# Array of Pointers

```
char* s[3] = { "Hello", "World", "!" };
```



# Command-Line Arguments

```
int main (int argc, char** argv) {  
    printf("Prog name: %s", argv[0]);  
    int i;  
    for (i = 1; i < argc; i++ ) {  
        printf("Next arg is %s", argv[i]);  
    }  
}  
  
// Can also use  
int main (int argc, char* argv[]) {  
}
```

# Readings

## Programming in C

- Chapter 7 “Working with Arrays”
- Chapter 8, Section “Functions and Arrays” (pp 137-152)
- Chapter 10 “Character Strings”
- Chapter 11 “Pointers”
  - Section on “Pointers and Arrays” (pp 259-273)