CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 9 – Arrays and Strings
About hw3 and hw4

- HW3 and **HW4** 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
  - We will look at how to use a debugger next lecture

- 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:

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

![Stack diagram]

Stack (one possible arrangement)

- 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!
int c[3] = { 1 , 2 , 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

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

13  1  42  3  54  64  6  7  8
Example 2

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

<table>
<thead>
<tr>
<th></th>
<th>c[0]</th>
<th></th>
<th>c[8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c[0]</th>
<th></th>
<th>c[8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Example 3

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

```
c   c+1   c+3
   c
0    1    2    3    4    5    6    7    8
```

```
c   c+1   c+3
   c
10   11   12   13   14   15   16   17   18
```
Example 4

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

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

```
c
  0  1  2  3  4  5  6  7  8
```
Examples

See `array.c` to experiment with examples 1 through 4.
// To pass an array to a function
// Indicate name without brackets
// Typically, want to pass size as well
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:

```c
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

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>

```
int c[2][3];
int i, j;

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

```c
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

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

```c
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
// Compare s1 and s2
int comparison = strcmp(s1,s2);
// Can also use strncmp
```
Array of Pointers

```c
```

![Diagram showing the array of pointers and the associated characters.](image)
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)