### Topic 5: C Arrays

School of Electrical and Computer Engineering  
Cornell University  
revision: 2019-09-20-09-43

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Basics</td>
<td>3</td>
</tr>
<tr>
<td>Iterating Over Arrays</td>
<td>7</td>
</tr>
<tr>
<td>Arrays as Function Parameters</td>
<td>9</td>
</tr>
<tr>
<td>Strings</td>
<td>10</td>
</tr>
</tbody>
</table>

Sections marked with a star (★) are not covered in lecture but are instead covered in the online lecture notes. Students are responsible for all material covered in lecture and in the online lecture notes. Material from the online lecture notes will definitely be assessed in the prelim and final exam.

Copyright © 2019 Christopher Batten. All rights reserved. This handout was prepared by Prof. Christopher Batten at Cornell University for ECE 2400 / ENGRD 2140 Computer Systems Programming. Download and use of this handout is permitted for individual educational non-commercial purposes only. Redistribution either in part or in whole via both commercial or non-commercial means requires written permission.
• In C, we would also like to be able to store a sequence of values all of the same type and then perform operations on this sequence

• We already saw how to implement a sequence of values using a chain of nodes; each node is a struct with a value and a next pointer

• Arrays are an alternative approach where the sequence of values is directly mapped into a linear sequence of variables
1. Array Basics

- Arrays require introducing **new types and new operators**
- Every type $T$ has a corresponding array type
- $T$ name[size] declares an array of size elements each of type $T$

```c
1 int a[4]; // array of four ints
2 char b[4]; // array of four chars
3 float c[4]; // array of four floats
```

- size should be a constant expression (e.g., literal)
- Technically a const variable is not a constant expression

```c
1 const int a_size = 4;
2 int a[a_size]; // technically avoid this
```

- Can initialize an array with struct-like initialization syntax

```c
1 int a[] = { 10, 11, 12, 13 };
```

- Cannot assign to an array

```c
1 int a[] = { 10, 11, 12, 13 }; // array of four ints
2 int b[4]; // array of four ints
3 b = a; // illegal!
```
Relationship between arrays and pointers

- Assume we declare an array `int a[4]`
- Type of the expression `a` is an “array of four ints”
- Expression `a` can act like a pointer to first element in the array
- Can use **pointer arithmetic** to access elements in an array
- The following expressions evaluate to pointers to each element
  - `a` pointer to element 0
  - `a+1` pointer to element 1
  - `a+2` pointer to element 2
  - `a+3` pointer to element 3

Example declaring, initializing, accessing an array

```c
int a[] = { 10, 11, 12, 13 };
int* a_ptr0 = a;
int* a_ptr1 = a+1;
int b = *a_ptr0 + *a_ptr1;
int c = *(a+2) + *(a+3);
*a = 20;
*(a+1) = 21;
*(a+2) = 22;
*(a+3) = 23;
```
1. Array Basics

Draw a state diagram corresponding to the execution of this program

```c
int a[] = { 10, 11, 12, 13 };
int* a_ptr0 = a;
int* a_ptr1 = a+1;
int* a_ptr2 = a+2;
int* a_ptr3 = a+3;

*a = *a_ptr3;
*(a+1) = *a_ptr2;
*(a+2) = *a_ptr1;
*(a+3) = *a_ptr0;

int* a_ptr4 = a+4;
int* a_ptrX = a-1;

int b = *(a_ptr4) + *(a_ptrX);
```
1. Array Basics

Subscript syntactic sugar

- The **subscript** operator \( a[i] \) is syntactic sugar for \( *(a+i) \)
- A pointer can *act* like an array
- Can use subscript operator to access elements via pointer

Example declaring, initializing, accessing an array

```c
int a[] = { 10, 11, 12, 13 };  // stack
int b = a[2] + a[3];

a[0] = 20;
a[1] = 21;
a[2] = 22;
a[3] = 23;

int* a_ptr0 = &(a[0]);
int* a_ptr1 = &(a[1]);
int c = a_ptr0[3] + a_ptr1[1];
```
2. Iterating Over Arrays

- We primarily work with arrays by iterating over their elements
- Example of calculating average of an array of ints

```c
int a[] = { 10, 20, 30, 40 }; int sum = 0; for ( int i = 0; i < 4; i++ ) sum += a[i]; int avg = sum / 4;
```

- Similar code except using pointer arithmetic

```c
int a[] = { 10, 20, 30, 40 }; int sum = 0; for ( int i = 0; i < 4; i++ ) sum += *(a+i); int avg = sum / 4;
```

```c
int a[] = { 10, 20, 30, 40 }; int* end = &a[4]; int* curr = a; int sum = 0; while ( curr != end ) {
    sum += *curr; curr++;
} int avg = sum / 4;
```
• size_t is a typedef for a type suitable for subscripting
• size_t is defined in stdlib.h
• Prefer size_t over int since size_t cannot be negative
• Example of collecting non-zero values from input array

```c
#include <stdlib.h>
#include <assert.h>

int a[] = { 0, 13, 0, 15 };
int b[4];

size_t j = 0;
for ( size_t i=0; i<4; i++ ) {
  if ( a[i] != 0 ) {
    b[j] = a[i];
    j++;
  }
}
```
3. Arrays as Function Parameters

- Arrays are always passed by pointer
- Must pass the size along with the actual array

```c
int avg( int* x, size_t n )
{
    int sum = 0;
    for ( size_t i=0; i<n; i++ )
        sum += x[i];
    return sum / n;
}

int main( void )
{
    int a[] = { 10, 20, 30, 40 };
    int b = avg( a, 4 );
    return 0;
}
```

- Arrays are always passed by pointer
- ... even with the following syntax

```c
int avg( int x[], size_t n )
{
    int sum = 0;
    for ( size_t i=0; i<n; i++ )
        sum += x[i];
    return sum / n;
}
```

- Prefer using int* x for parameters
- It makes it obvious arrays are always passed by pointer
4. Strings

- Strings are just arrays of chars
- The length of a string is indicated in a special way
- The null terminator character (\0) indicates the end of string
- New syntax using double quotes for string literals (""")

```c
1  char str[] = { 'e', 'c', 'e', '\0' };  
2  char str[] = "ece";
```

- C standard library provides many string manipulation functions
- These functions are declared in the string.h header
  - `strlen` : calculate length of a string
  - `strcmp` : compare two strings
  - `strcpy` : copy one string to another string
  - `atoi`  : convert a string into an integer
Draw a state diagram corresponding to the execution of this program

```c
int strlen( char str[] )
{
    int i = 0;
    while ( str[i] != '\0' )
        i++;
    return i;
}

int main( void )
{
    char a[] = "ece2400";
    int b = strlen( a );
    return 0;
}
```