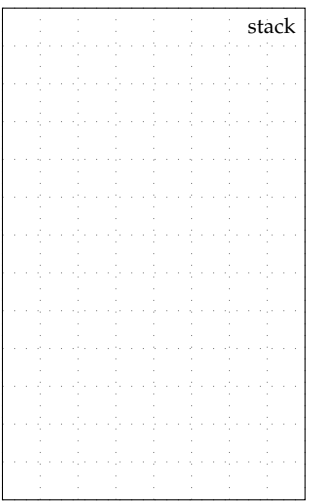
ECE 2400 Computer Systems Programming
Fall 2017

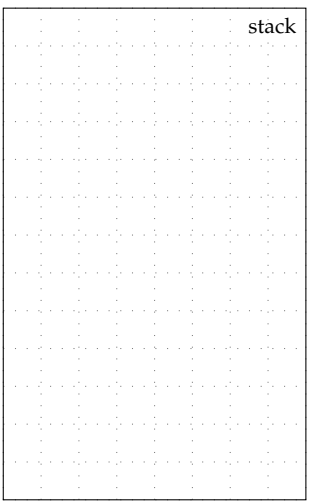
Topic 5: C Arrays

School of Electrical and Computer Engineering
Cornell University

revision: 2017-09-20-12-24

- We would 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 are 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`

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

- `size` can be `const` or non-`const`, but `const` is much more efficient

```
const int a_size = 4; // size is a const variable
int   a[a_size];      // array of four ints

int b_size = 4;       // size is a non-const variable
int   b[b_size];      // array of four ints
```

- Can initialize an array with struct-like initialization syntax

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

- Cannot assign to an array, only to array elements

```
int a[4] = { 10, 11, 12, 13 }; // array of four ints
int b[4];                      // array of four ints
b = a;                         // illegal!
```

- The subscript operator (`[i]`) evaluates to the value of element `i`
- The subscript operator is used for reading/writing elements
- The subscript operator uses zero-based indexing

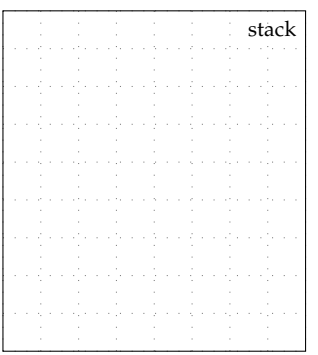
```
int a[4];       // array of four ints
a[0] = 13;      // assign 13 to element 0
int b = a[0];   // initialize b with value of element 0
```

- Out-of-bounds access is perfectly legal and dangerous!

```
int a[4];       // array of four ints
a[4] = 13;      // out-of-bounds write!
int b = a[5];   // out-of-bounds read!
```

Example declaring, initializing, RHS and LHS subscripting

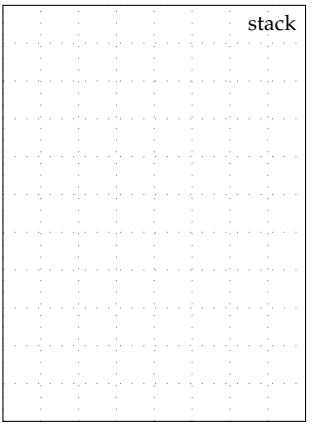
```
int a[4] = { 10, 11, 12, 13 };
int b = a[0] + a[1];
a[2] = b;
a[3] = a[0];
```



Relationship between arrays and pointers

- Assume we declare an array `int a[4]`
- Type of the expression `a` is an "array of four `int`s"
- Expression `a` can *act* like a pointer to first element in the array
- Similarly, a pointer can sometimes *act* like an array
- Definitely one of the more inconsistent aspects of C syntax

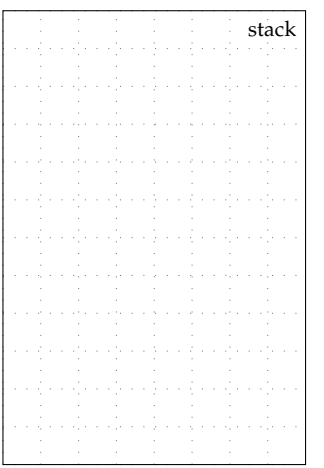
```
1 int  a[4]  = { 10, 11, 12, 13 };
2 int* a_ptr = a;
3 int  b     = a_ptr[0] + a_ptr[1];
4 a_ptr[2]   = b;
5 a_ptr[3]   = a[0];
```



2. Iterating Over Arrays

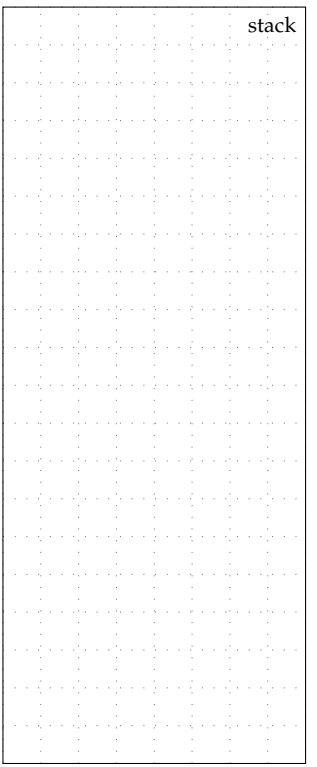
- We primarily work with arrays by iterating over their elements
- Example of calculating average of an array of `int`s

```
1 int a[] = { 10, 20, 30, 40 };
2 int sum = 0;
3 for ( int i = 0; i < 4; i++ )
4   sum += a[i];
5 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

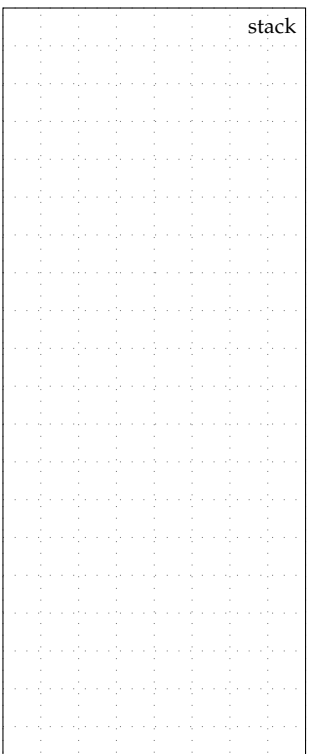
```
1  #include <stdlib.h>
2  #include <assert.h>
3
4  int a[] = { 0, 13, 0, 15, 17 };
5
6  size_t num_nonzeros = 0;
7  for ( size_t i=0; i<5; i++ )
8    if ( a[i] != 0 )
9      num_nonzeros++;
10
11 int b[num_nonzeros];
12
13 size_t j = 0;
14 for ( size_t i=0; i<5; i++ ) {
15   if ( a[i] != 0 ) {
16     b[j] = a[i];
17     j++;
18   }
19 }
20
21 assert( num_nonzeros == j );
```



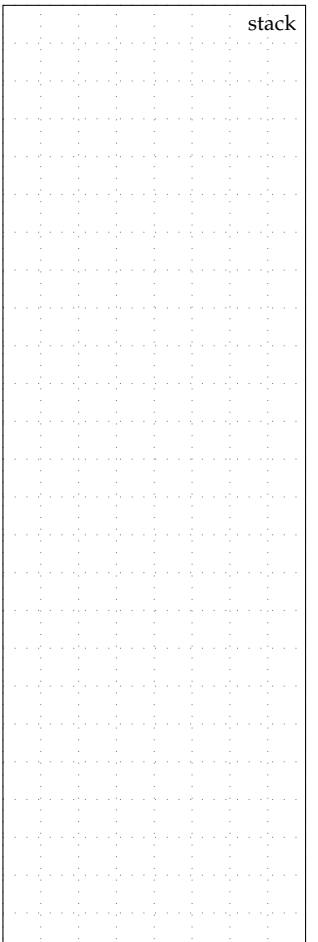
3. Arrays as Function Parameters

- Passing an array as a function parameter is like passing a pointer
- Arrays are *always* passed by reference
- Must pass the size along with the actual array

```
1  int avg( int x[], size_t size )
2  {
3    int sum = 0;
4    for ( size_t i=0; i<size; i++ )
5      sum += x[i];
6    return sum / size;
7  }
8
9  int main( void )
10 {
11   int a[] = { 10, 20, 30, 40 };
12   int b = avg( a, 4 );
13   return 0;
14 }
```



```
1  void vvadd( int dest[],
2              int src0[],
3              int src1[],
4              size_t size )
5  {
6    for ( size_t i=0; i<size; i++ )
7      dest[i] = src0[i] + src1[i];
8  }
9
10 int main( void )
11 {
12   int a[] = { 1, 2, 3 };
13   int b[] = { 5, 6, 7 };
14   int c[] = { 0, 0, 0 };
15   vvadd( c, a, b, 3 );
16   return 0;
17 }
```

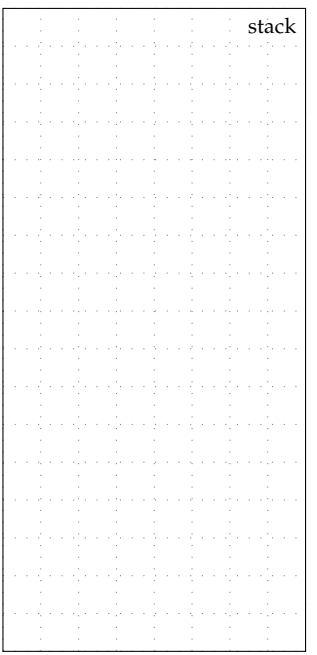


4. Multi-Dimensional Arrays

- Concept of a 1D sequence can be extended to higher dimensions
- A 2D array is just an 1D array of arrays
- Multi-dimensional arrays are stored in row-major order

```
int matrix[3][3] =
{
  { 1, 2, 3 },
  { 4, 5, 6 },
  { 7, 8, 9 },
};

int row = 2;
int col = 1;
int a = matrix[row][col];
matrix[row][col] = 13;
```



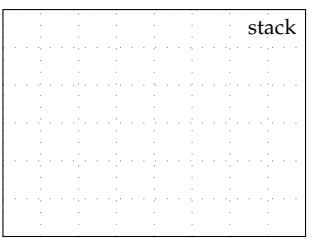
Classic matrix multiplication

```
void mmmult( int dest[][3],
             int src0[][3],
             int src1[][3],
             size_t size )
{
  for ( int i = 0; i < size; ++i )
    for ( int j = 0; j < size; ++j )
      for ( int k = 0; k < size; ++k )
        c[i][j] += a[i][k] * b[k][j];
}
```

5. Strings

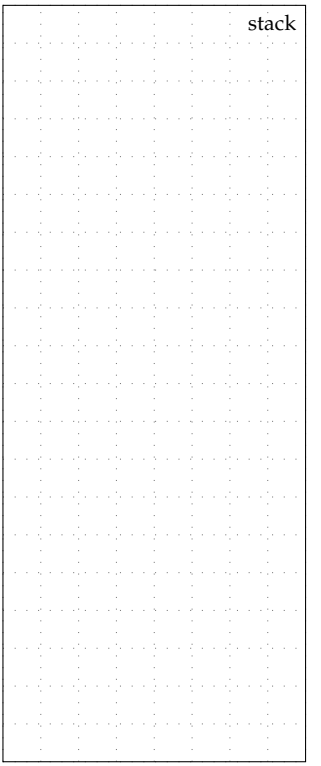
- Strings are just arrays of `chars`
- The length of a string is indicated in a special way
- The null terminator character (`\0`) indicates the end of string
- New syntax using double quotes for string literals (`""`)

```
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

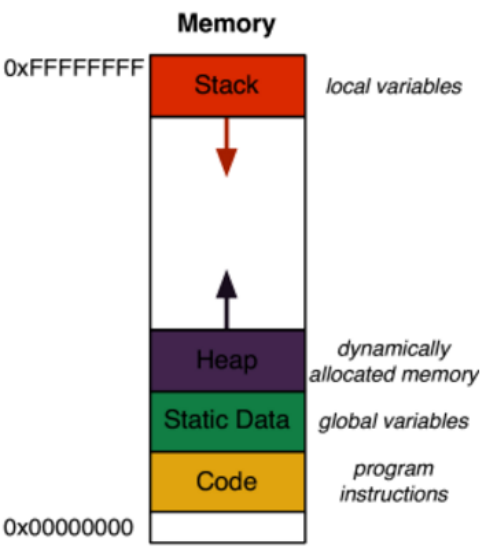
```
1  int strlen( char str[] )
2  {
3    int i = 0;
4    while ( str[i] != '\0' )
5      i++;
6    return i;
7  }
8
9  int main( void )
10 {
11   char a[] = "ece2400";
12   int b = strlen( a );
13   return 0;
14 }
```



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

int main( void )
{
  char a[] = "ece2400";
  int b = strcmp( a, "ece2300" );
  return 0;
}
```

- Constant strings are stored in the static data section of machine memory



stack

static data