CSE 220: Systems Programming
Variables, Strings, and Loops

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo
This week’s lab will be due **February 9 at 10:00 PM**. (That’s before lab 01! …but not much.)

**TA office hours** start this week.

Remember that many of you are new to the command prompt! Read **everything** for all assignments and labs.

Check Piazza frequently.
Types

C is a typed language.

Every variable has a type, and is declared.

Every value assigned to a variable must match that type.

The compiler will automatically convert between some types.

Valid:

```c
int x = 0;
float y = 0;
x = 37;
y = x;
```

Invalid:

```c
int x = 0;
x = "Hello, world!";
```
Some Types

There are many types; for now, consider:

- **int**: Integers of a convenient size for the computer (32-bit for us)
- **char**: Characters (typically 8-bit integers)
- **double**: Double-precision floating-point numbers

There are also array types.

Array types are declared with square brackets: `[]`:

- **char a[]**: An array of characters. Often used for C strings.
- **int scores[200]**: An array of exactly 200 integers.
A Warning

These slides attempt to be precise, but simplify some things.

Usually this is because the details:
- are unnecessarily confusing, or
- require knowledge you are not expected to have.

If something here conflicts with the standard or the compiler, the standard or compiler wins.

I will try to mark imprecise statements with a pilcrow: ¶.
Declaring Variables

Variables are declared by stating their type and name.

```c
int x;          /* x is an integer */
double d;      /* d is a floating-point double */
```

Variables retain their type while they are in scope.

Various modifiers can be applied to variables. In particular, `const` declares the variable is a constant.
Scope

Variables in C have **scope**.

A variable cannot be used **out of scope**.

Variables declared **outside of any block** (```{}```):
- are normally **global**: they can be accessed by **any code**
- are **file-local** with the modifier **static**: they can be accessed by any code in this file

Variables declared **in a block**:
- Come into scope **where declared**
- are valid until the ```}```
Arrays

C arrays are a series of contiguous memory locations. (This will become important later.)

Arrays are declared with []. The size is between [].

Arrays can have three “sizes”, depending on what’s in the []:

- **Unknown size**: Nothing is specified
- **Constant size**: A constant expression is specified
- **Variable size**: A run-time computed expression is specified
Array Sizes

Array sizes specify how many elements are in the array.

```c
int x[32];
int matrix[32][16];
```

C does not remember the array’s size. ¶

This means that illegal accesses aren’t caught.

```c
int x[4];
x[10234] = 0; /* Whoops. */
```
Unknown Array Sizes

Unknown size arrays are limited in use.

They often appear as arguments to functions (as in `main()`).

An array of unknown size cannot be declared normally.

Sizes are required for multidimensional arrays.
(Except for the “outermost” dimension, but only sometimes.)

```c
void func(int matrix[][3][2]);
```
Array Indexing

C array indexes start at 0.

An array of size 10 contains elements 0 through 9.

Arrays can be dereferenced with []:

```c
int array[10];
int i = 7;

array[4] = 0;
array[i] = 0;
array[i + 1] = 0;
```
Static Initializers

An array can be initialized all at once at declaration.

\[
\text{int } \text{array}[10] = \{0, 3, 5, 0, 0, 1, 0, 0, 2, 0\};
\]

This is called a static initializer.

Static initializers can only be used at declaration.

\[
\text{int } \text{array}[3];
\]

\[
\text{array} = \{1, 3, 5\}; \quad /* \text{syntax error} */
\]
C Strings

C strings are just arrays.

Strings, like arrays are not associated with a length. (You have to count the characters to know how long they are.)

A C string consists of:

- the characters in the string, followed by
- a zero byte (the ASCII NUL character) (NUL terminator).

The zero byte is idiomatically written '0'.
**Quoted Strings**

**Quoted strings** automatically build such arrays.

```c
char str[] = "Hello";
/* str = { 'H', 'e', 'l', 'l', 'o', '\0' } */
```

A quoted string may be assigned to an **array only at declaration**.

After declaration, quoted strings must be **copied into arrays**:

```c
char str[32];
strncpy(str, 32, "Hello");  /* See man 3 strncpy */
```
String Functions

There are many string functions in the C library. Most of them are defined in `<string.h>`.

Some useful examples:

- `strlen()`: Compute the length of a string by counting bytes
- `strncpy()`: Copy a string until its NUL character
- `strncat()`: Concatenate one string to another
- `strstr()`: Search for one string inside another
Strings as Pointers

The idiomatic string type is `char *`. Arrays and pointers are closely related, we’ll discuss this later.

```c
char *str = "Hello, CSE 220";
char array[] = "Another string";
char *otherstr = array;
```
Character Constants

C strings can be in many encodings, but C code is in ASCII. ASCII contains Latin characters, numbers, and punctuation.

An ASCII character can be converted to an integer with \texttt{\textasciitilde}.  

\begin{verbatim}
char c = 'A';   /* 65 */
int i = 'B';   /* 66 */
\end{verbatim}

Each byte of a string can be assigned in this fashion.

\begin{verbatim}
char str[] = "emacs";
/* Give it the respect it deserves */
str[0] = 'E';
\end{verbatim}
The for Loop

The C for loop is its most versatile loop.

It allows looping over almost anything.

```
for (initialization; condition; increment) {
    body;
}
```

It translates to a more traditional while loop (with caveats):

```
initialization;
while (condition) {
    body;
    increment;
}
```
It is **idiomatic** to loop over strings:

```c
for (int i = 0; str[i] != '\0'; i++) {
    /* use str[i] */
}
```

Note that the string length is never directly computed!
Looping over Arrays

For other arrays, other conditions must be specified:

```c
int array[ARRAYSZ];

for (int i = 0; i < ARRAYSZ; i++) {
    /* Use array[i] */
}
```

Remember that you must somehow know the size of the array.
Controlling the Loop

Two keywords control loop execution:
- break
- continue

The continue statement will immediately:
- Execute the increment statement
- Start the body over at the top

The break statement will immediately end the loop.
We will develop `strlen()` together.
Summary

- C is a **typed language**
- Every variable has a type
- Variable values must match the type
- Variables have **scope**, and cannot be used outside that scope
- Arrays are **contiguous memory locations**
- Array syntax uses `[]`
- C strings are arrays of characters
- Every C string is **terminated with a zero byte**
- For loop syntax
- For loops are very flexible
Next Time …

- More about integers
Optional Readings

Copyright 2019 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.