CSCII 2212: Intermediate Programming / C
Parts of Chapter 11, 12, and 13

Alice E. Fischer

September 10 and 12, 2013

Basic Types in C
Pictures of C Types
Pointers
Pointer Arithmetic, L- and R- values.
Using *

Types in C

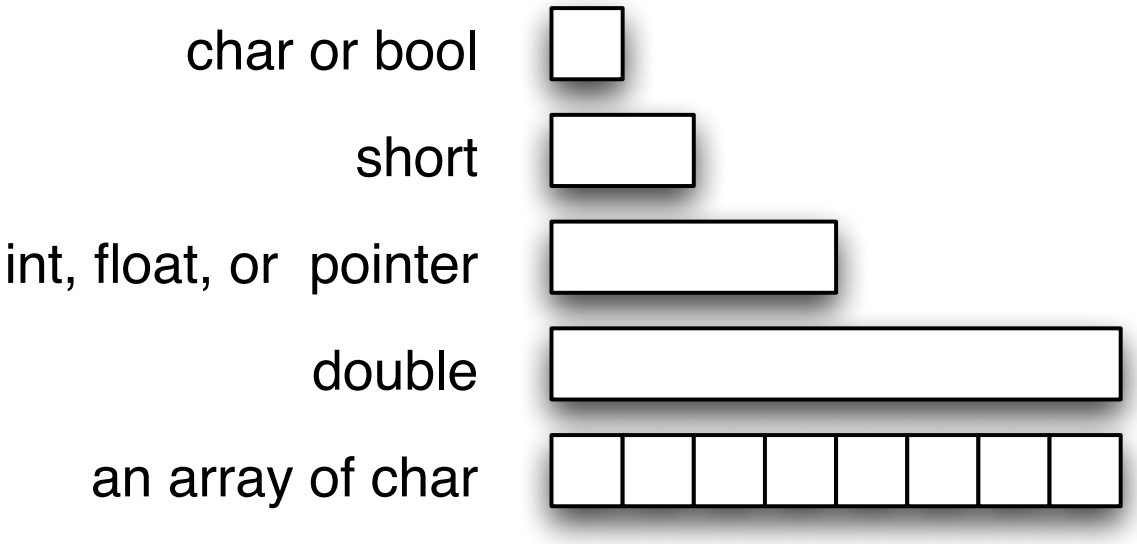
C has many built-in types. These include:

- int and unsigned int
- short and unsigned short
- long and unsigned long
- char, signed char, and unsigned char
- bool (since C-99)
- float, double, and long double

Chapter 7 lists the range of values that can be stored in each type and gives the limits of precision for the floating types.

Pictures of the Basic Types.

In the diagrams, the size of the box is proportional to the number of bytes needed to store a value.



Pointers.

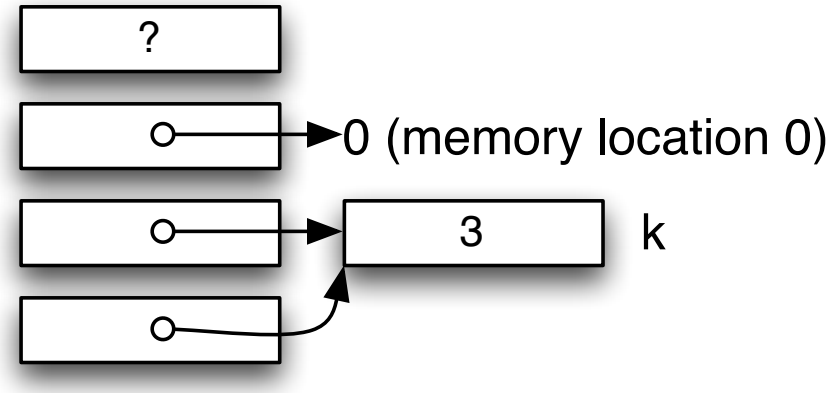
```
int k = 3;
int* p;
int* q = NULL;
int* pi = &k;
```

A pointer, uninitialized p

A NULL pointer q

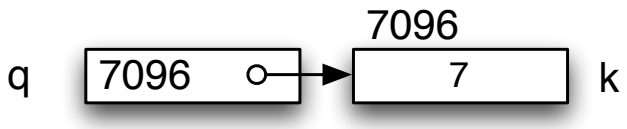
A pointer to an integer pi

After executing p = pi; p



Pointer arithmetic, L- and R-values.

```
int k = 7;
int* q = &k;
```

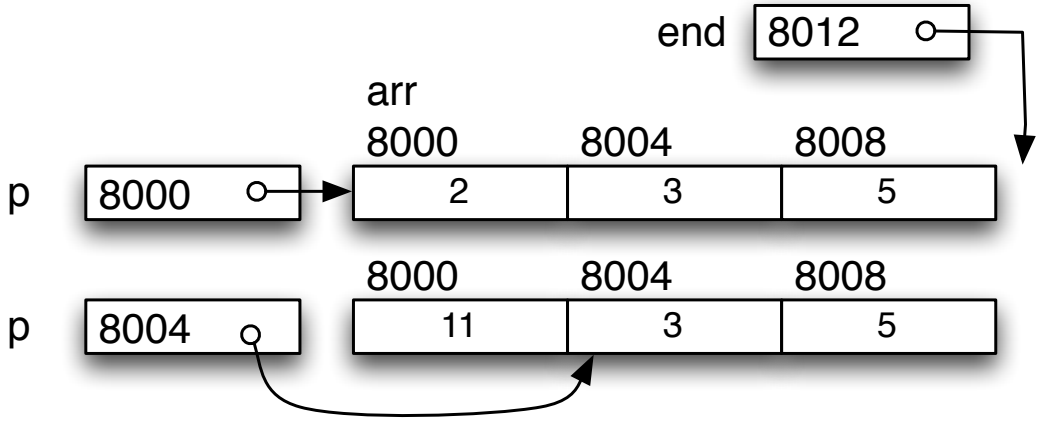


The L-value of k is 7096. Its R-value is 7.

The R-value of q is 7096.

***q is the same as k.**

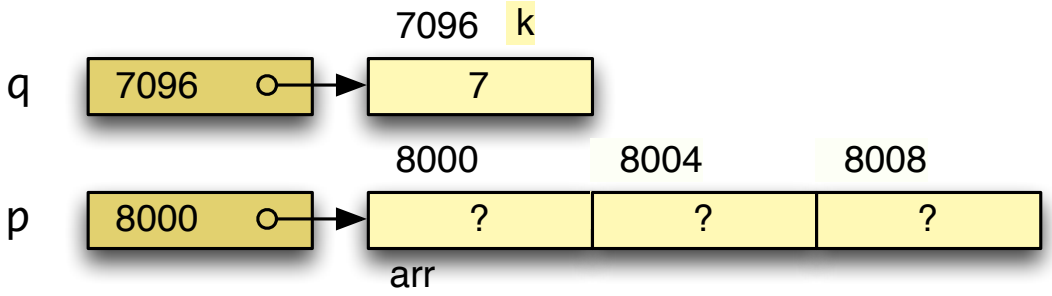
```
int arr[3] = {2, 3, 5}
int* p = arr;
int* end = arr + 3;
```



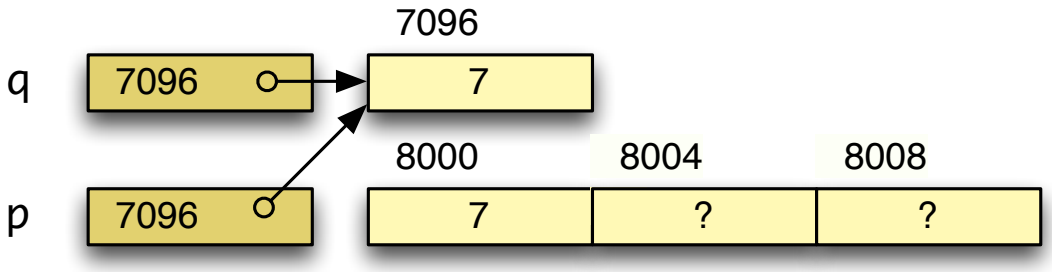
After ++p

Using *.

```
int k = 7;
int arr[3];
int* p = arr;
int* q = &k;
```




```
*p = *q;
p  =  q;
```



- If you write p = , you will change the contents of p.
- If you write *p = you will change the value of whatever p points at.
- The star level on both sides of the = must be the same.

Strings

What is a Literal String (review)
Pictures of Strings
Examples

What is a string?

The word string applies to objects of three different types in C:

- A literal string is zero or more characters enclosed in quotes. This kind of string has type `const char*` .
- A string is a null terminated array of characters. This kind of string has type `char[]` .
- Or a pointer to a char that is part of a null-terminated array of chars. This kind of string has type `char*` .
- The C compiler treats the last two types the same way in most situations.

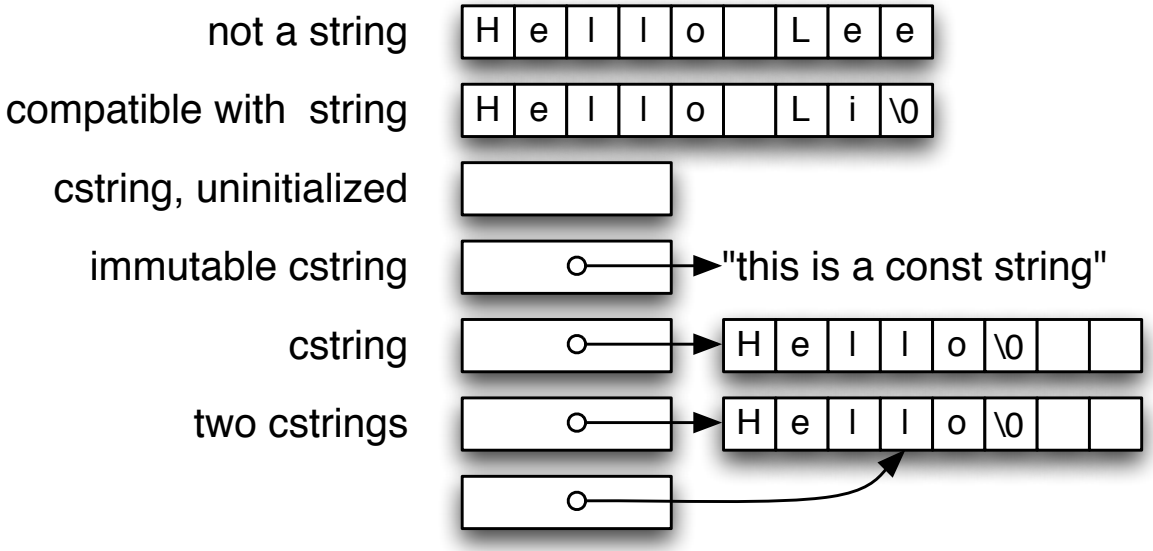
About Literal Strings.

- Literal strings are defined by writing them in your code.
- They are not variables and cannot be modified – they are `immutable.`
- The compiler assigns read-only storage for literal strings and makes them available for you. You can output these strings or point at them.

About String Variables.

- If a string is stored in an array, it might or might not fill up the array.
- The `\0` character marks the end of the string. After that, the contents of the array are garbage.
- You can modify the contents of the array by using assignment and/or the string functions.
- When `scanf()` inputs a string, it will ALWAYS end in `\0` .
- However, if the string is longer than the array that stores it, it will overwrite the value of some other variable. This is called `walking on memory`.

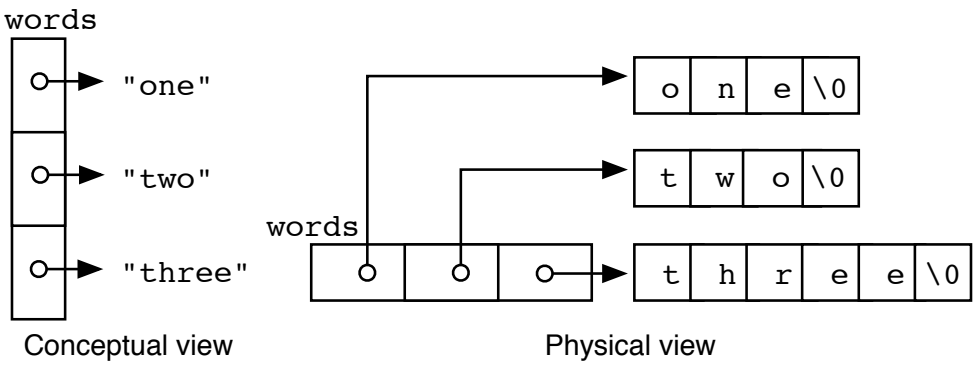
Pictures of Strings.



An Array of Strings

Arrays of strings are useful in many application. This data structure (often called a ragged array) has two parts:

- A backbone: an array of pointers.
- Several attachments of type char* or cstring, possibly const.



This is a 2-dimensional data structure: you can subscript the backbone, then you can subscript the strings attached to it.

How do you store a string?

- A string has two parts: a pointer and an array of characters.
- Memory must be declared or created dynamically for both.
- To declare a cstring variable (the pointer part) use type `char*` or use type `cstring` from the tools library.
- To declare the array part, create an array of chars that is long enough. This array will hold up to 9 chars and a null terminator: `char ary[10];`
- We declare an array to be 1 longer than the longest possible contents because there must be space at the end of the data for a null terminator character, \0

Remember: a string can occupy many bytes of storage, and a pointer is just 4 bytes. So a `char*` cannot store a string.

How do you declare and create a string?

- You can declare an array to store a string WITHOUT initialization, like `ary` above. For this kind of declaration, you must give the maximum string length +1 in the square brackets.
- You can initialize a string in the declaration
  ```
  char greeting[] = "Hello";
  ```
- You can also give BOTH the array length AND an initializer
  ```
  char name[16] = "A. Fischer";
  ```
- When you choose the array length, consider the longest thing you intend to store in it. That is often longer than the initial value for the string.

How do you use a string?

- A cstring variable can point at one literal string, then later, be changed to point at another.
- It can point at either the beginning or the middle of a character array or literal string.
- To print a string, use `printf( "%s", myStringName );`
- To input a string that does not have internal spaces, use `scanf( "%ns", myArrayName );` The number `n` is the length-1 of the array you will be reading the string into.
- Inputting a string with internal spaces is more complex and will be handled later.

Examples

```
char* fname = "Alice";   // Point at string literal.
char letters[20] = "Waltz";
string lname = letters;  // Point at letters[0].
char* fun = &fname[2];   // Point at fname[2].

printf("My name was %s %c. %s\n", fname, 'E', lname);
lname = "Fischer";
printf("Now my name is %s %c. %s ", fname,'E',lname);
printf("\nI like to skate on %s.\n", fun);
```

Output:

```
My name was Alice E. Waltz
Now my name is Alice E. Fischer
I like to skate on ice.
```

The String Library

Basic String Operations
Copying a String
String Comparisons
Searching a String
An Array of Strings
A 2-D array of chars

String Operations

```
char word[10] = "Hi"
char* st = word;
```

There are two ways to change a string: change the pointer, or change one of the chars it points at.

- Operate on the pointer part of the string:
  - `if (st == "Hi")` Does st point at the string literal? (No.)
  - `sizeof` The number of bytes in st (4)
  - `st = "Harmony"` Make st point at a different word.
- Operate on the series of characters:
  - `strcmp(word, "Joy");` // Do the words have same letters?
  - `strlen(word);` // The # of letters before the \0. (3)
  - `strcpy(word, "Joy");` // Copy "Joy" letters to word array.

Basic String Operations.

- `size_t strlen( const char* s );`
  Returns the number of characters in the string `s`, excluding the null character on the end.
- Use subscript and = to modify individual chars in the middle of a string.
- Subscript can be used for an array of chars, and also for a pointer to an array of chars.
- When subscripting a pointer, the subscripts are relative to the slot the pointer points at.
- If two string pointers point into the same array of chars, they can both be used to modify the string stored there.

Copying a String

- `char* strcpy( char* dest, const char* src );`
  Copies the string `src` into the array `dest`. We assume that `dest` has space for the string.
- `char* strncpy(char* to,const char* src, size_t n)`
  Copies exactly `n` characters from `src` into `to`. If fewer than `n` characters are in `src`, null characters are appended until exactly `n` have been written.
- `char* strcat( char* dest, const char* src );`
  Appends the string `src` to the end of the string `dest`, overwriting its null terminator. It is a serious error is `dest` does not have space for the combined string.
- `char* strncat(char* to, const char* src,size_t n)`
  Same as `strcat()` except that it stops after copying `n` characters, then writes a null terminator.

String Comparisons.

```
typedef char* cstring;
cstring s1, s2;
```

- `s1 == s2` asks if the pointers store the same memory address.
- To compare the chars that s1 and s2 point at, use `strcmp()`.
  `int strcmp( const char* p, const char* q );`
  Compares string `p` to string `q` and returns a negative value if `p` is lexicographically less than `q`, 0 if they are equal, or a positive value if `p` is greater than `q`.
- `int strncmp(const char* p,const char* q,size_t n)`
  Same as `strcmp()` but returns after comparing at most `n` characters. It will return sooner if a null character happens sooner.

Searching a String

- `char* strchr( const char* s, int ch );`
  Searches the string `s` for the first (leftmost) occurrence of the character `ch`. Returns a pointer to that occurrence if it exists; otherwise returns `NULL`.
- `char* strrchr( const char* s, int ch );`
  Searches the string `s` for the last (rightmost) occurrence of the character `ch`. Returns a pointer to that occurrence if it exists; otherwise returns `NULL`.
- `char* strstr( const char* s, const char* sub );`
  Searches the string `s` for the first (leftmost) occurrence of the substring `sub`. Returns a pointer to the first character of that occurrence if it exists; otherwise returns `NULL`.

Some String Tools

The tools library contains five functions that process strings.

These functions are called from `banner()`, but could also be used separately.

- `cstring today( char date[] );`
- `cstring oclock( char hour[] );`
- `void when( char date[], char hour[] );`

These functions provide a convenient way to do a common job.

- `char menu_c(cstring title, int n, cstring menu[])`
- `int  menu_i(cstring title, int n, cstring menu[])`

Example: Composing a Form Letter

Techniques to learn from this example:

- Password validation with `strcmp()`
- Parallel arrays of strings.
- Menu processing.
- Create one string out of many using `stlen()` and `strncpy()`
- Parsing a string using `strchr()` and `isspace()`
- Getting the gender-word right using a string variable.
- Using post-increment during string processing.
- The `? :` operator.