

Strings

Lecture Notes

Overview

- A *string* is a collection of ordered characters
 - Some languages support strings as a native type, others use arrays of characters
 - Strings are sequences of characters under some *encoding* (ASCII, Unicode)
1. Static & dynamic strings
 - a. String literals
 - b. Declaration & initialization
 2. String operations/library functions
 - a. Assignment
 - b. Printing
 - c. Substrings
 - d. Concatenation
 - e. Comparisons
 - f. String length
 3. Misc
 - a. Input
 - b. Arrays of Strings
 - c. Tokenizing
 - d. Character Tests (alpha, upper/lower, space, etc.) Conversions (number/string)
 4. Pitfalls
 - a. Null vs empty string

Strings in C

1. Overview
 - Strings in C are *null-terminated* arrays of `char` elements
 - Bookkeeping: size of arrays is not maintained, neither is the length of strings
 - Instead: the end of a string is indicated by the special character: `'\0'` (zero, null)
 - Null terminator can appear anywhere in the array (string is effectively cut short)
 - Without null terminator: many functions will fail (continue to scan, bleeding into memory that is not part of the string)
2. Static declarations
 - `char message[] = "Hello World!";`
 - size is one more than the number of characters (to accommodate null terminator)

- Manually change contents of a string:

```
message[0] = 'h';
message[6] = 'w';
message[11] = '\0';
```

-

3. Dynamic strings

- Exactly the same as any dynamic array!
- When allocating space, need to allocate bytes + 1 for the null terminator.
- Examples:

```
char *msg = NULL;
msg = (char *) malloc(sizeof(char) * (n+1));
```

4. String/character operations

a. Libraries

- `string.h`
- `ctype.h` (`isalpha`, `isdigit`, `islower`, `toupper`, `isspace`, etc.)

b. Assignment

- Can only use the assignment operator in a declaration, *not* to assign values:
`message = "goodbye world!"; //not allowed`
- Only elements in an array can be set with the assignment operator
- Can use the `strcpy` (string copy) function to copy contents of one string into another:

```
char *strcpy(char *dest, const char *src)
strcpy(destinationStr, sourceStr);
strcpy(destinationStr, "goodbye world!");
```

- Pitfall: destination must be big enough to hold the source!
- Alternative: if we only want to copy part of the string:
`char *strncpy(char *dest, const char *src, size_t n)`
- Copies from the first character, *n* bytes (characters)

c. Printing

- `printf` placeholder: `%s`
- Example:
`printf("message is %s \n", msg);`

d. Substrings

- To copy a substring: just need to start from another index!
`strncpy(foo, &message[6], 6); //foo is now "world"`
- If null terminator is in the first *n* bytes, copied, otherwise it is our responsibility

e. Concatenation

- Concatenation is an operation whereby two strings are linked together
- `strcat`
- `strncat`
- Both concatenate the second string to the first
- The first string must be large enough to hold both

- f. Comparisons
 - Character comparisons can use regular numeric comparison operators (<, >, <=, >=) since chars have integer values (ASCII table)
 - Comparing strings: lexicographic ordering (not alphabetic; see: <http://www.codinghorror.com/blog/2007/12/sorting-for-humans-natural-sort-order.html>)
 - General comparison contract: a comparator/comparison function returns:
 - Something negative if a < b
 - Zero if a is equal to b
 - Something positive if a > b
 - int strcmp(a, b)
 - int strncmp(a, b, n)
 - g. String length
 - int strlen(a)
 - Iterating over characters in a string
5. Misc
- a. Input
 - Most techniques are dangerous (buffer overflows)
 - fgets is safe, but buffer processing may be necessary
 - b. Arrays of Strings
 - 2D array of chars; same rules apply as with other multidimensional arrays
 - c. Tokenizing
 - Lots of data may be separated by some *delimiter* (commas, tabs, whitespace)
 - Common to *split* a string along some delimiter into *tokens* and process each token.
 - C: `char *strtok(char *str, const char *delimiter)`
 - First call: string to be tokenized along some delimiter
 - Subsequent calls: use NULL instead of str to get the next token (use the same delimiter, optionally a different one)
 - Careful: `strtok` modifies the given string (it uses it as a buffer)
6. Pitfalls
- a. NULL is not the same thing as ""
 - b. Memory management & null terminator (C only)
 - Some string functions take care of null terminator for us, others don't: RTM (Read the Manual!)

Strings in Java

1. Representation: String class (could do character arrays, but not recommended)
 - a. Immutability
 - b. String s: s is a reference to a string

- c. Creating new strings: `new String("foo")`
- 2. String operations
 - a. Java Documentation: <http://docs.oracle.com/javase/6/docs/api/java/lang/String.html>
 - b. Assignment
`String s = "Hello World";`
`String t = s;`
 - c. Concatenation:
 - Use the `+` operator (creates a new string)
 - Operator is overloaded: can mix types!
 - Under the hood: Polymorphism magic!
 1. Code is replaced with `StringBuilder` calls
 2. Object code is wrapped in `String.valueOf` calls
 - d. Substrings
 - `s.substring(int)`
 - `s.substring(int, int)`
 - e. Comparisons
 - `s == t`: compares memory addresses!
 - `s.compareTo(String)`
 - `s.compareToIgnoreCase(String)`
 - f. String length
 - `s.length()`
 - g. Others
 - i. `contains`
 - ii. `replace`
 - iii. `split`
- 3. Character class:
 - a. <http://docs.oracle.com/javase/6/docs/api/java/lang/Character.html>
 - b. `isSpace`, `isDigit`, etc.
- 4. Misc
 - a. `StringBuilder` Class
 - Mutable version of a `String`
 - <http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html>
 - <http://docs.oracle.com/javase/tutorial/java/data/buffers.html>
 - `append`, `insert`, `replace`

Exercises

1. Write a function to copy a string that also dynamically allocates new memory for it.
2. Write a function to determine if a given string is a *palindrome*. A palindrome is a string that is the same forward and backward.
3. Write a function to convert all characters in a string to lower case

4. Write a function to return a new string that is the substring of a given string; the function should take, as part of its input a beginning and an ending index
5. Write a function to reverse the contents of a string
6. Write a function to return a reversed copy of the contents of a string
7. Write a function to replace certain characters with other characters
8. Write a function to remove certain specified characters
9. Write a function to return a copy of a string with certain characters removed/replaced
10. Write a function to remove all whitespace from a string (and/or to return a copy of the new string)
11. Write a function and/or program to detect whether or not a string contains repeated words (such as "the the")
12. Write a program/function to compute (and sort) a suffix array. A suffix array of a string is a sorted array of all of its suffixes.
13. Write a function to replace all space characters with an underscore
14. Write a function to "double space" a string: replace all end-line characters with two end-line characters
15. Implement a true split function for C: it should return an array of strings split along a given delimiter
16. Write a function to replace any single numeric character (surrounded by spaces) to its English word (but leaves other instances of numbers alone)
17. Write a function to create an acronym from a given string: it should take the first letter of each word and capitalize them (International Business Machines -> IBM)
18. Write a function to sort a collection of strings