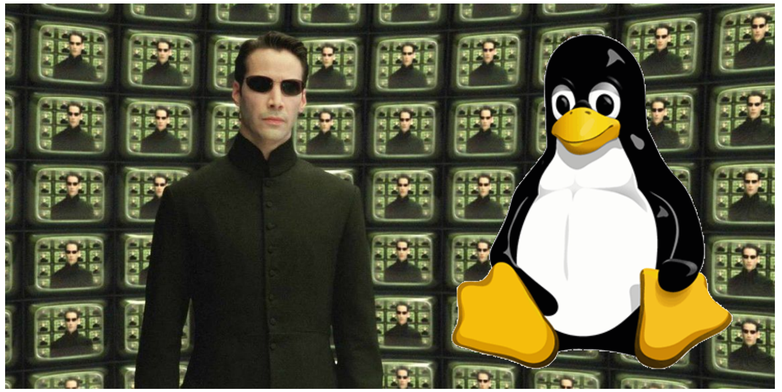
22.06.20 17:36

CSCI 125
Introduction to Computer Science and Programming II
Lecture 5: Char Array and String



Jetic Gū

2020 Summer Semester (S2)

Announcement

- Assignment 2 has been posted last night, due date 30 June 2020
- Lab 2 submission has begun (I forgot to turn it on before Sunday)
  - Due date is still 28 June 2020
- Practice Problem p18 test cases are still in the making (Tuesday at the latest)
  - Turned out a lot harder to design test cases than provide solution
  - Due date for all practice problems (p009-p018) is 28 June 2020

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. Character Arrays
    2. Strings

P0
Review

Data Types

- Primary √
  - Integers, Characters, Boolean √
  - Floating point √
  - Void √
- Derived
  - Function√, Array√, Pointer√, Reference√
- User Defined
  - Struct, Class, Enumerate, Typedef

`char` is simple enough. what causes problems is **`char`** arrays

`string` is a class! You won't find this in C!

Review

Character Arrays

Character Arrays

- Syntax

```
char str[100];
```

- The standard `string` data type in C
- You create an array of char, and you will have a string
- Fixed length: cannot exceed

Concept

Character Arrays

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]
'c'	'h'	'e'	'c'	'k'	0	e'

- Storage: like standard array
- Must include a termination element: signals the end of string
  - Otherwise when you print it, your program will keep accessing memory regions until it finds a 0 value, which will likely cause **segmentation fault**
- In this example, it will be printed as "check"

Input and Output using `cstdio`

- Input Output: `cstdio` library (or in C, `stdio.h`)

```
char str[100];

scanf("%s", str); // no address-of op since its array

printf("%s", str);
```

- Single char symbol in formatting: `"%c"`
- Char array (string) symbol in formatting: `"%s"`
- Although `scanf` works in most cases, it is NOT recommended

Concept

Input and Output using `cstdio`

- Read in a single string, separated by whitespace

```
scanf("%s", str);    // input "Hello World"
printf("%s", str);   // outputs "Hello"
```

- `%s` operates similarly to how `scanf` handles int input: it stops reading upon seeing a whitespace

- Demo 1 here

Input and Output using `cstdio`

- When the storage space is not enough for the string

```
char str[3];

scanf("%s", str);    // input "cheeseburger"

printf("%s", str);   // outputs "cheeseburger"
```

- `scanf` doesn't check of memory space! This way **unallocated memory** will be used!

- Dangerous! Not recommended to use `scanf` for char arrays, we have **alternatives**.

Input and Output using `cstdio`

- Read in a single character

```
char str[3];

scanf("%s", str);    // input "cheeseburger"

printf("%s", str);   // outputs "cheeseburger"
```

- `scanf` doesn't check of memory space! This way **unallocated memory** will be used!

- Dangerous!

Input and Output using `cstdio`

- Correct way of reading in Char Array

```
fgets(str, sizeof(str), stdin);
```

  - `stdin`: standard input stream, by default it is Keyboard input
  - `sizeof`: returns the declared size of the array
  - **returns true** if reading was successful

- This will keep reading until **newline** `'\n'` is detected (reads whitespace as well!) or storage is used up
- Safe: ensures only reading in characters within declared storage space
- Demo 3 here

Input and Output using `cstdio`

- Correct way of reading in Char Array

  `fgets(str, sizeof(str), stdin);`

  - `stdin`: standard input stream, by default it is Keyboard input
  - `sizeof`: returns the declared size of the array
  - **returns true** if reading was successful

```
#include <cstdio>

using namespace std;

int main() {

    char str[10000];

    if (fgets(str, sizeof(str), stdin))

        printf("\n%s\n", str);

    else {

        // handle error

    }

    return 0;

}
```

Input and Output using `cstdio`

- Reading in characters one at a time

```
char ch;

ch = getchar();
```

- Common usage

```
while ((ch = getchar()) != '\n' && ch != EOF) {... do stuff}
```

  - `EOF`: end of file symbol, signals end of input (control-d if stdin, or an actual end of file)

- This can be very useful as well.

Utilities

- Some useful utilities for Character strings in `cstring` (In C the header is `string.h`)
  - `strcpy`: Copy string
    e.g.: `strcpy (tgtStr, srcStr);  // source str can be const`
  - `strcat`: Concatenate strings
    e.g.: `strcat (tgtStr, "Hello ");  // "Hello " + tgtStr`
  - `strcmp`: Compare of two strings, outputs 0 if equal
    e.g.: `if (strcmp(str1, str2) == 0) printf("Equal!\n");`

1. http://www.cplusplus.com/reference/cstring/

More

- For more information on Char Arrays (especially **do**s and **don't**s), here's a useful reference online from CMU's Software Engineering Institute
  - From CMU: https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator

1. http://www.cplusplus.com/reference/cstring/

First taste of C++ class: `string`

String Class

- It is a class, so not supported in C (C++ Only)
- Internal Storage: uses Char Arrays
  - If you look at the memory regions' representation, it is almost identical to C char arrays
- Easier management, the class methods handles everything for you

Concept

String Class

- Header: must include `string`

```
#include <string>
```

- Syntax

```
std::string str;

// uses std namespace
```

- Maximum length: 4,294,967,295 (2^{32}-1)

Concept

String Class

- The statement:

```
string name = "William Jacob";
```

declares name to be a string variable and also initialises name to `"William Jacob"`

  - The first character, 'W', is in position 0
  - The second character, 'i', is in position 1

String Class

- Binary operator `+` and the array subscript operator `[]`, have been defined for the data type `string`
  - `+` performs the string concatenation operation
- Example:

```
str1 = "Sunny";
str2 =  str1 + " Day";
```

- stores `"Sunny Day"` into `str2`

Input and Output

- Assuming you are using `std` namespace

```
cin >> str;  // reads until whitespace

cout << str;
```

- `endl` is also a `string` instance, which for us can be considered equivalent to `"\n"`

Concept

Input and Output

- Extraction operator
  - Skips any leading whitespace characters and reading stops at a whitespace character
- The function `getline`

```
getline(cin, str);
```

  - Reads until end of the current line

String Memory Structure

- Building blocks
  - Char Array, stored as Char pointer
    Remember, Arrays are Pointers! (Demo 4, 5)
  - Size information: the memory allocated for this instance
  - Iterator: provides the [ ] array access mechanism
- Core functions
  - The class manages memory for you!

String Memory Structure

- Creation: storage space for short strings
  - Empty string: same space
- Extra long strings: dynamically allocate space for you
  - You wouldn’t notice this though, it’s designed so neatly!

Additional `string` Operations

- `length`
- `substr`
- `swap`
- There's tons more, for more reference consult the C++ official doc
  - http://www.cplusplus.com/reference/string/string/

Concept

`length` Function

- Returns the number of characters currently in the string
- Syntax:

```
str.length();
```

- `length` returns an `unsigned int`
- The value returned can be stored in an integer variable

Concept

`substr` Function

- Returns a particular substring of a string
- Syntax:

```
str.substr(expr1, expr2);
```

  - `expr1` and `expr2` are expressions evaluating to unsigned integers
  - `expr1` specifies a position within the string (starting position of the substring)
  - `expr2` specifies the length of the substring to be returned

Concept

`swap` Function

- Interchanges contents of two string variables
- Syntax:

```
str1.swap(str2);
```

- Suppose you have the following statements:

```
string str1 = "Warm";
string str2 = "Cold";
```

- After `str1.swap(str2);` executes,
  the value of `str1` is `"Cold"` and the value of `str2` is `"Warm"`

Concept