Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

Outline

Arrays

- An array is **not a class** (as in Java)
- An *array* is a **contiguous** area of memory allocated to **several variables** of the **same type**
- An array is declared by
  `<type> <identifier>[<size>]`;
  it has size `sizeof(<identifier>) = sizeof(<type>)*<size>`
- Example:
  `int v[10];`
  declares the variable `v` as an array of `10` `int` variables.
- Elements are `v[0]`, `v[1]`, ..., `v[<size>-1]` and are stored contiguosly in memory
- C does NOT check array boundaries! WARNING: `v[-1]` is syntactically correct

address	content	variable
...	...	...
0080F8	...	...
0080FC	v[-1]	outside v!!!
008100	v[0]	
008104	v[1]	
...	...	v
008124	v[9]	
008128	v[10]	outside v!!!
00812C	...	...

Arrays: length

- The length of an array is **not** saved in the data structure
  - **Do not ever try** to invoke the “method” `length()` with an “array object”
  - “methods” and “objects” **do not exist** in C
- The programmer must record the length of the array in some way
  - by storing a special character terminating the useful content (such as in strings, which are terminated by the byte `0`)
  - by recording the length in another (additional) variable
- Still, the following constant expression is useful to compute the length of an array

```
int v[10], len;  /* declaring an array v of a given length */

len = sizeof(v)/sizeof(v[0]);
```

Strings: arrays of non-zero bytes terminated by 0

- The String “object” or “class” **does not exist** in C
  - (again, “object” and “class” **do not exist** at all in C)
- In C, the term “string” is used to denote
  1. an array of `char`, as declared by:

     `char s[100];`
  2. the bytes of such an array are interpreted as ASCII codes of characters
  3. the byte 0 is written in `s` after the last character, to terminate the string
- Constant strings are enclosed by **double quotes** `"`

```
"constant"   /* valid constant string. Allocates 9 bytes */
'wrong'      /* string are NOT enclosed by single quotes ' */
'A'          /* ASCII code of 'A'  */
"A"          /* 2-bytes string: 'A', 0 */
```

- A string may be printed by the %s placeholder of the `printf` as in

```
printf("The string s is \"%s\"\n", s);
```

Initialization of arrays

- Arrays may be initialized by a sequence of values enclosed within { and }

1. The size of the array may be unspecified and determined by the length of the initialization, as follows

   ```
   int v[] = {1,2,3};
   ```

2. The following declaration+intialization

   ```
   char v1[] = {'C', 'i', 'a', 'o', 0};
   char v2[] = "Ciao";
   ```

   are equivalent and create an array of **5 bytes** (NOT 4 bytes) (strings are arrays of characters terminated by 0)

3. If the size is specified, as in

   ```
   int v[10] = {3, -1, 4};
   ```

   then all following elements are set equal to zero. Hence,

   ```
   int v[100] = {0};
   ```

   is a convenient way to initialize all elements of the vector to zero.

Strings in memory, converting string into integers

- A strings is stored as an array (sequence) of characters, terminated by the null character (0)

```
char v[] = "258";
```

address	(hex)	
7FFF0040671A8108	32='2'	v
7FFF0040671A8109	35='5'	
7FFF0040671A810A	38='8'	
7FFF0040671A810B	00	

```
int n = 258; /* =256+2 */
```

address	(hex)	
7FFF0040671A8108	02	n
7FFF0040671A8109	01	
7FFF0040671A810A	00	
7FFF0040671A810B	00	

- Converting a string into an integer

```
#include <stdlib.h>

int a;

a = strtol(s, NULL, 10);/* 10 is the base of conversion */
```

  - stores the value represented by the string s in the integer variable a
  - the second parameter is for advanced users

Strings in memory, converting string into floating point

- Converting a string into a floating point number

```
#include <stdlib.h>

double a;

a = atof("123.45");  /* same as a = 123.45 */
```

  - stores the value represented by the string s in the floating point variable a

Strings: manipulation by including `string.h`

- By including the library `#include <string.h>` some useful function strings to manipulate strings may be used
  1. The following function returns the number of bytes in `s` before the terminating byte 0

     ```
     strlen(s);
     ```

  2. to append string `src` to string `dest`

     ```
     strcat(dest, src);
     ```

     - ★ `dest` **must** be allocated at least `strlen(dest)+strlen(src)+1`
     - ★ otherwise (quoting from `man strcat`): "If `dest` is not large enough, program behavior is unpredictable; **buffer overruns are a favorite avenue for attacking secure programs**."

  3. to append **up to** `n` bytes of `src` to string `dest`

     ```
     strncat(dest, src, n);
     ```

     - ★ if no 0 byte terminating `scr` among the first `n` bytes, only first `n` bytes are concatenated
     - ★ it prevents the user to write arbitrary-long data

Reading input from the keyboard: `fgets()`

- the function `fgets(...)` reads a string of characters
- `#include <stdio.h>` must be added on top to use it
- Syntax

```
char s[80];

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

  - `s[80]` is a pre-allocated array of characters (**string** of characters)
  - reads a string from `stdin` (**standard input**)
  - store the string up to `sizeof(s)-1` characters into `s`. The string **cannot** be sizeof(s) long because the terminating zero must be stored too
  - the string is read until EOF (end-of-file, `Ctrl+D`) or newline
  - if "Enter" is pressed, then the ASCII code of "new line" (=10) is also stored in `s`
- `man fgets`

`test-read.c`, try with input from file