CS 2113
Software Engineering

LAB 3
Strings and File IO

Modified by Bo Mei

Professor Tim Wood - The George Washington University

char[]'s (AKA strings)

- C doesn't natively support a string data type
  - Instead, must use arrays of chars
- But, things are a bit tricky!

```
char name[12]; // reserve 12 bytes
name = "Dracula"; // try to set string
```

ERROR!

- You can not simply assign one string to another
  - Need to access array elements one at a time
  - Or use string functions

char[] on the stack

- Defining an array reserves space on the stack
  - 1 byte per char

```
char name[12];
int a;
int b;
```

	int main()
10000	name[0]
10001	name[1]
10002	name[2]
...	...
10011	name[11]
10012	a
10016	b

char[] on the stack

- Defining an array reserves space on the stack
  - 1 byte per char
  - Can set each element of array

```
char name[12];
int a;
int b;

name[0] = 'D';
name[1] = 'r';
```

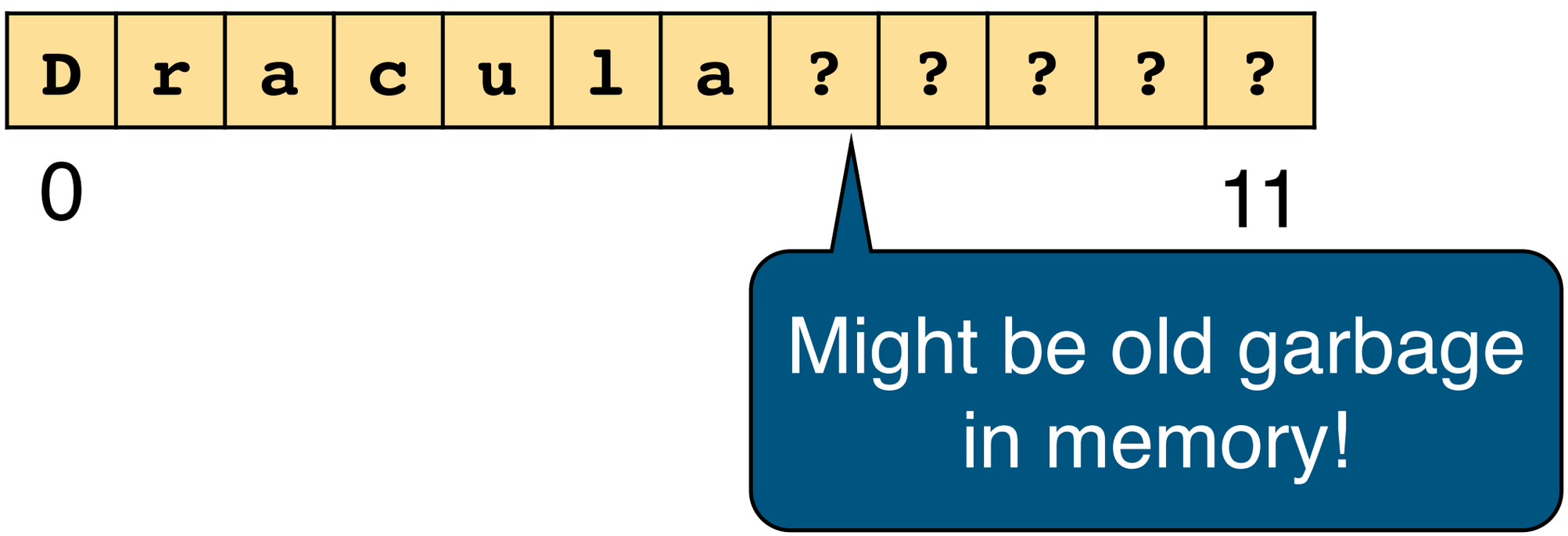
	int main()
10000	name[0] = D
10001	name[1] = r
10002	name[2] = ???
...	...
10011	name[11] = ???
10012	a = ???
10016	b = ???

Printing Strings

- printf can display strings using %s
  - `printf("My name is: %s\n", name);`

- What if the string doesn't use the full array???

name[12]

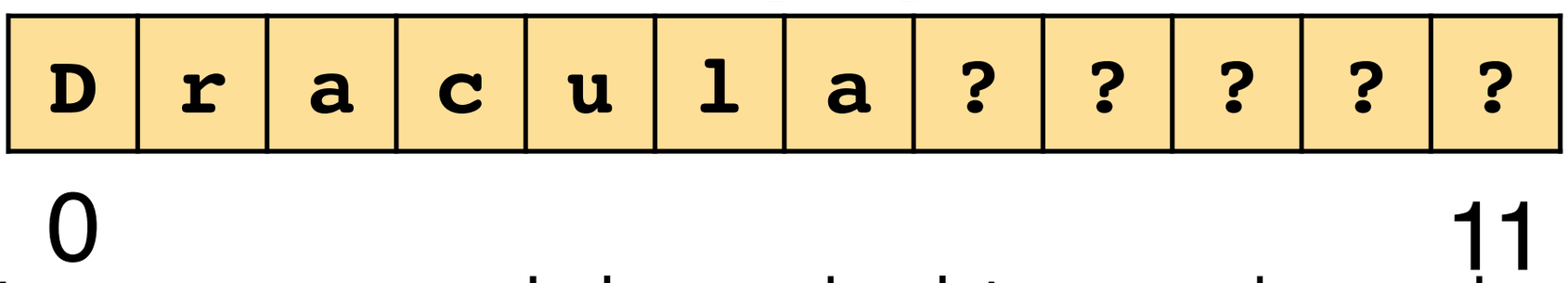


Printing Strings

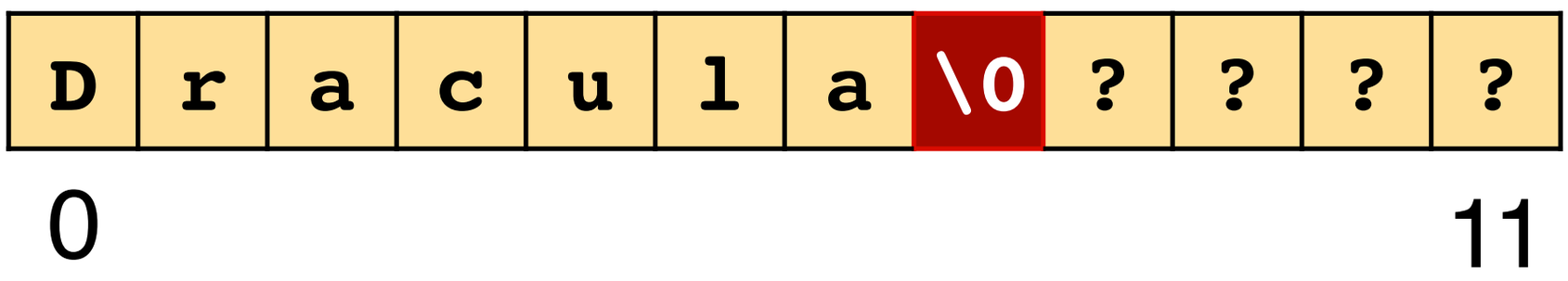
- printf can display strings using %s
  - `printf("My name is: %s\n", name);`

- What if the string doesn't use the full array???

name[12]



- C stores a special symbol to mark end of string



Buffer Overflows

- What happens in Java?

```
// bad Java code
int myArray[12];
myArray[99] = 666;
```

What happens in C?

```
// bad C code
int myArray[12];
myArray[99] = 666;
```

- Java tracks the size of an array... C does not
- Need to be extra careful with strings

```
// ??? C code
char myWord[5];
strcpy(myWord, "Hello");
```

Buffer Overflows

- What happens in Java?

```
// bad Java code
int myArray[12];
myArray[99] = 666;
```

What happens in C?

```
// bad C code
int myArray[12];
myArray[99] = 666;
```

- Java tracks the size of an array... C does not

- Need to be extra careful with strings
  - Need space for \0!

```
// BAD C code
char myWord[5];
strcpy(myWord, "Hello");
```

NO! Need 5 + \0

Arrays and Pointers

- Arrays and pointers are similar
  - Store data in different place
  - Use the same basic syntax to reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char *ptr = malloc(10);
    char  name[10];
    strcpy(name, "Jane");
    strcpy(ptr, "Joe");
    printf("ptr = %s\n", ptr);
    printf("name = %s\n", name);
    return 0;
}
```

	Read-Only Data
5000	J
5001	a
5002	n
5003	e
5004	\0
5005	J
5006	o
5007	e
5008	\0
5009	???
...	...
5014	???

	Heap
20000	J
20001	o
20002	e
20003	\0
20004	???
...	...
20009	???

	Stack
10000	ptr = 20000
10004	name[0] = J
10005	name[1] = a
10006	name[2] = n
10007	name[3] = e
10008	name[4] = \0
10009	name[5] = ???
	...

Things that don't work...

- Cannot copy a string with =

```
char a[] = "test"; // only works for declaration
char *b;
b = a; // makes b point to a, does not copy!
b[0] = 'r'; // affects both a and b
```

- Cannot test equality with ==

```
char *a = "test";
char *b = "test";
if(*a == *b) // only compares first char
if(a == b)   // only compares memory address
if(strcmp(a, b)==0) // returns 0 if equal
```

  - but you can compare individual characters with ==

str Functions

- http://www.cplusplus.com/reference/clibrary/cstring/
- Don't forget #include <string.h>

```
char * strcpy ( char * dest, const char * source );  copy from
source to dest
  be sure dest points to a region of memory with enough
  space for the string in source (including '\0' symbol!)
```

```
size_t strlen ( const char * str );  Find the number of
letters in a string
  Returns a "size_t" type (basically the same as unsigned int)
  Does not include the '\0' character!
```

```
int strcmp ( const char * str1, const char * str2 );  Compare
two strings
  Returns 0 if their characters are all identical
  Returns > 0 if str1 starts with "higher" character value
  Returns < 0 otherwise
```

A few things to remember

- String literals are stored in read-only data segment, so string literals are immutable!
- A string literal like “How are you?” represents the address of its first *char,* which is the address that stores ‘H’ in this case.

- Array variable is a special pointer, a constant. Always store the same address and can’t be reassigned. (The contents/elements of the array can certainly be changed.)
- Array variable stores the address of the first element.

A few things to remember

- Initializing a char array by using a string literal only happens when declaring the char array.

```
char str[] = {‘H’, ‘o’, ‘w’, ‘ ‘, ‘a’, ‘r’,
‘e’, ‘ ‘, ‘y’, ‘o’, ‘u’, ‘?’, ‘\0’};
```

is equivalent to

```
char str[] = “How are you?”;
```

- Otherwise, use `strcpy`

```
char str[100];
str = “How are you?”; // Wrong! Why?
strcpy(str, “How are you?”); // Correct. Why?
```

Try this…

- Write a function to calculate the length of a string:

```
#include <stdio.h>

int stringlength(char *str)
{
  // do not include the ending '\0'
}

int main()
{
  int l;
  char *text = "blah!";
  l = stringlength(text);
  printf("Length of %s is %d\n", text, l);
  return 0;
}
```

Don't use the strlen function!

stringlength.c

- Write a function to calculate the length of a string:

```
int stringlength(char *str){
    int len = 0;
    char c = *str; // first letter of string
    while(c != '\0') { // do not include the ending ‘\0’
        len++; // increment length counter
        str++; // go to next char in array
        c = *str;
    }
    return len;
}

int main()
{
    int l;
    char *text = "blah!";
    l = stringlength(text);
    printf("Length of %s is %d\n", text, l);
    return 0;
}
```

Opening Files

- You can **read** and **write** both **binary** and **text** files
  - Binary files = raw data. Must be read by computer, not a human
- C has a FILE data type

```
FILE *dataFile;
dataFile = fopen("data.txt", "w");
```

- Open a file with **`fopen(NAME, MODE)`**
  - NAME of the file to be opened, possibly with directory path
  - MODE to open with:

```
"w" - for writing (erase existing)
"r" - for reading
"a" - for appending
"b" - for a binary file.
ANSI C99 also allows:
"r+" - for reading and writing
"w+" - for reading and writing (erase existing)
"a+" - for reading and appending
```

Closing Files

- Don’t forget to close files after using the file.
- fclose(*file-variable*);

- C provides 3 pre-opened files

File	**Description**
stdin	Standard input (open for reading)
stdout	Standard output (open for writing)
stderr	Standard error (open for writing)

Reading from Text Files

- C provides basic read support
  - One character (**`fgetc`**) or line (**`fgets`**) at a time...

```
FILE * txtFile;
char line[200];
txtFile = fopen("mine.txt", "r");
fgets(line, sizeof(line), txtFile);
```

open for reading

string to fill

size of buffer

FILE to read, or *stdin*

  - fgets will read at most (size-1) letters until it reaches a new line (\n) or end of the file
- **`fgets`** returns:
  - A pointer to the data read in (i.e., the same as the first param)
  - or **`NULL`** if at end of file or there was an error

a C defined constant

read.c

- Write a function to read a file and print it line by line
  - Print the line number in front of each line

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE * txtFile;
    char line[200];
    int i=0;
    txtFile = fopen("read.c", "r");

    /*
       WHAT TO PUT HERE???
    */
}
```

read.c

- Write a function to read a file and print it line by line
  - Print the line number in front of each line

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE * txtFile;
    char line[200];
    int i=0;
    txtFile = fopen("read.c", "r");

    while(fgets(line, 200, txtFile) != NULL) {
        i++;
        printf("%d: %s", i, line);
    }
    fclose(txtFile);
}
```

Need to allocate space before reading

Use return value to check if at end of the file

Don't need an extra \n since it is kept as part of string

Don't forget to close the file.

Reading Numbers

- scanf
  - Like printf
  - But almost never works. Notorious for its poor end-of-line handling.
- Solution: combine fgets and sscanf (string scanf)
  - **fgets(line, sizeof(line), stdin);**
  - **sscanf(line, format, &variable1, &variable2 . . .);**
- Note
  - *format* is a string similar to the printf format string
  - ampersand (&) in front of the variable names

Example

```
#include <stdio.h>
char line[100];/* line of input data */
int  height;   /* the height of the triangle */
int  width;    /* the width of the triangle */
int  area;     /* area of the triangle (computed) */
int main() {
    printf("Enter width height? ");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %d", &width, &height);
    area = (width * height) / 2;
    printf("The area is %d\n", area);
    return (0);
}
```