APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 18
March 2, 2012

Today

- Multi-File Programs
- Strings

- Midterm

Multi-File Programs

Why?

- Modularity
- Clarity
- Logistics

Remember `-lm`?

- When using `math.h`, we needed to add `-lm` flag
- To use a function, we need two things:
  - Prototype (`.h` file, aka "header file")
  - Implementation (`.c` file)

.h Header File

Ignore these for now

```
#ifndef SUM_H
#define SUM_H

/* Returns the sum of 'x' and 'y'. */
int sum(int x, int y);

#endif
```

sum.h

.c Implementation File

- Don't need prototype (in the `.h` file)
- Uses `#include "sum.h"`, not `#include <sum.h>`

```
#include "sum.h"

int sum(int x, int y)
{
    return x + y;
}
```

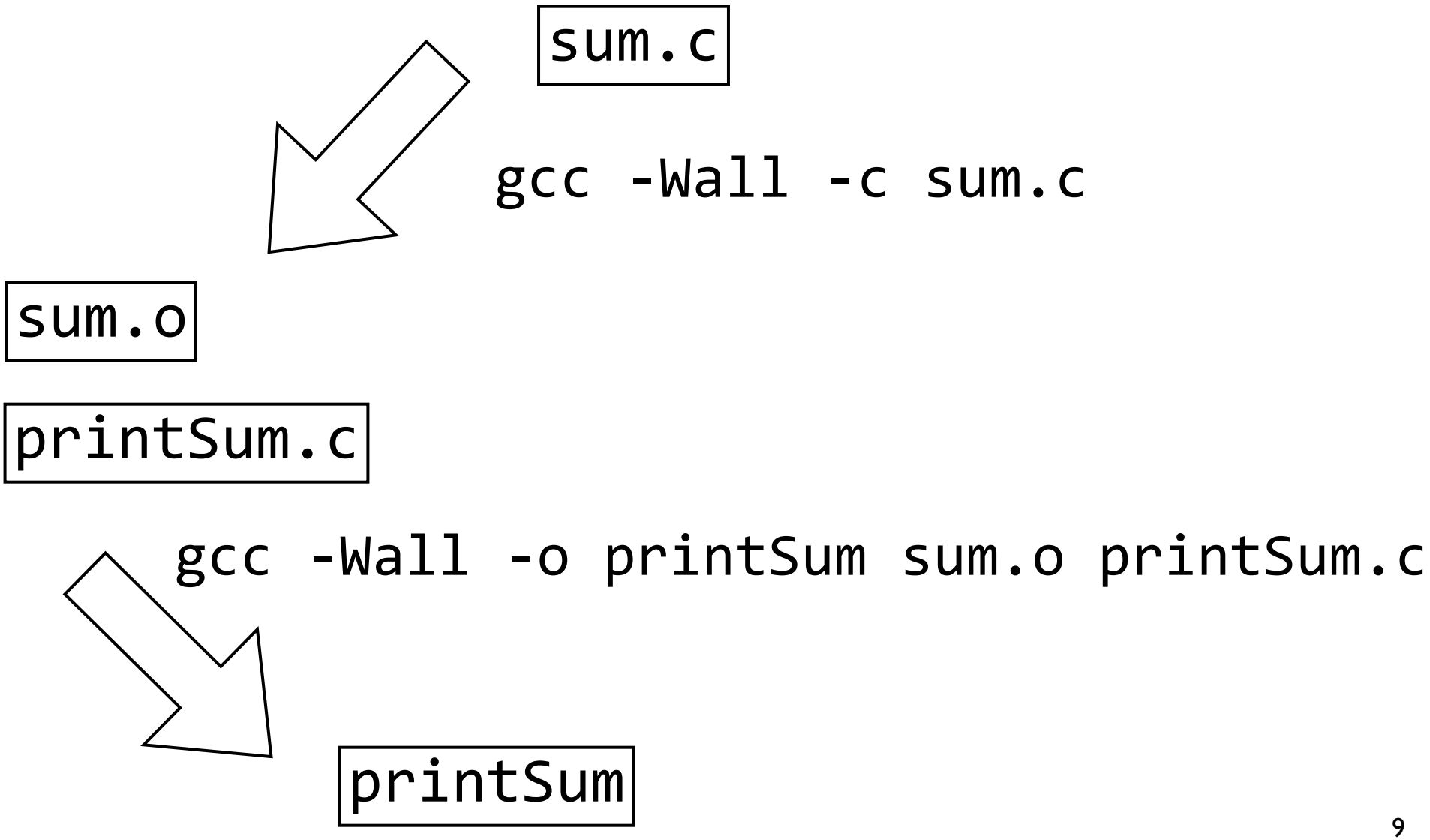
sum.c

Calling Code

```
#include <stdio.h>
#include "sum.h"

int main(void)
{
    int i = 2;
    int j = 3;

    int result = sum(i, j);
    printf("%d\n", result);

    return 0;
}
```

printSum.c

Compiling


sum.c
gcc -Wall -c sum.c
sum.o
printSum.c
gcc -Wall -o printSum sum.o printSum.c
printSum

`make`

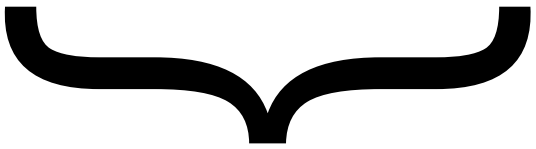
- Several tools to automate this process
- Most common UNIX tool is `make`
- A3+ have `Makefile` (see website)
- Not part of course, you're not being tested on it

Strings

Strings

- We've been using them since the first class

```
printf("Hello world!\n");

printf("This is a string.\n");
```



string literal

Literals

- An expression with a fixed value
- `int size = 12;`
- `double volume = 355.0;`

- `bool b = true;`
- `char c = 'j';`

Literals

- An expression with a fixed value
- `int size = 12;`
- `double volume = 355.0;`

- `bool b = true;`
- `char c = 'j';`
- `string s = "This is a string\n";`

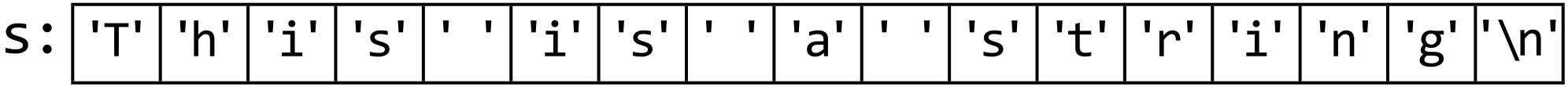
Error

Strings Aren't a Type

- In C, strings are just arrays of chars...

```
char s[] = "This is a string\n";
```

```
char s[] = {'T','h','i','s',' ','i','s',' ','a',
            ' ','s','t','r','i','n','g','\n'};
```



- ... with two differences

Not Quite Right

Differences

- Compiler knows about double quotes

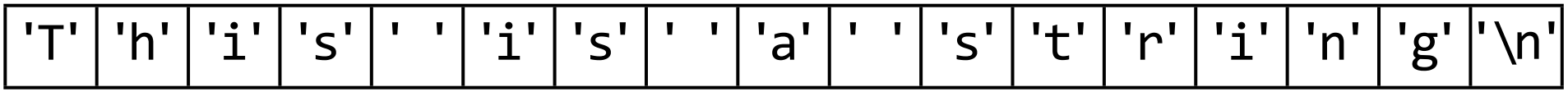
array of `char`	`{'X', 'Y', 'Z'}`
array of `char`	`"This is a string\n"`
array of `int`	`{1, 3, 4, 9}`
array of `double`	`{3.5, 2.2, 1.9}`
array of `bool`	`{false, true, false}`

- Strings must end with null character (`'\0'`)

```
char s[] = "This is a string\n";
```

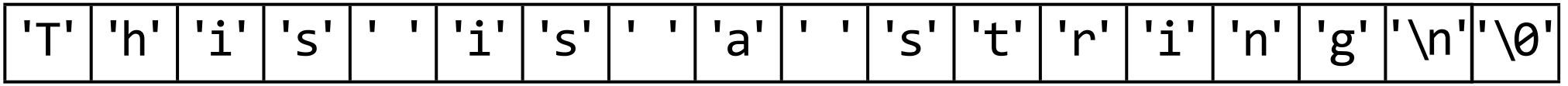
```
char s[] = {'T','h','i','s',' ','i','s',' ','a',
            ' ','s','t','r','i','n','g','\n'};
```

s:



```
char s[] = "This is a string\n";

char s[] = {'T','h','i','s',' ','i','s',' ','a',
            ' ','s','t','r','i','n','g','\n','\0'};
```

s:



Arrays vs. Strings

- A `char[]` is just a sequence of `chars`
- A string is a sequence of `chars` followed by `'\0'`

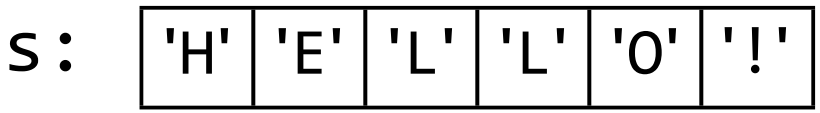
```
char validResponses[] = {'A', 'B', 'C', 'D'};

char lastName[] = {'D', 'E', 'B', 'E', 'R', '\0'};
```

- Every string is a `char[]`
- Not all `char[]` are strings

Arrays vs. Strings

- The status of a `char[]` can change

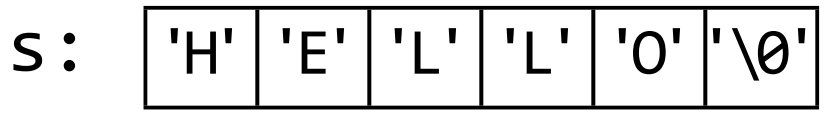
```
char s[] = {'H', 'E', 'L', 'L', 'O', '!'};
```



Arrays vs. Strings

- The status of a `char[]` can change

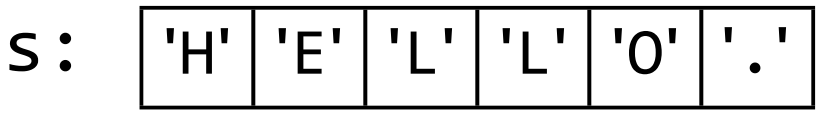
```
char s[] = {'H', 'E', 'L', 'L', 'O', '!'};
s[5] = '\0';
```



Arrays vs. Strings

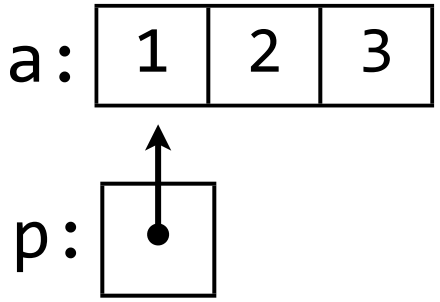
- The status of a `char[]` can change

```
char s[] = {'H', 'E', 'L', 'L', 'O', '!'};
s[5] = '\0';
s[5] = '.';
```

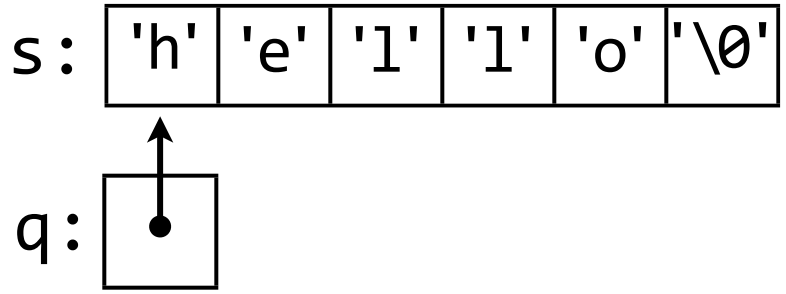


Arrays vs. Pointers

```
int a[] = {1, 2, 3};
int *p = a;
```

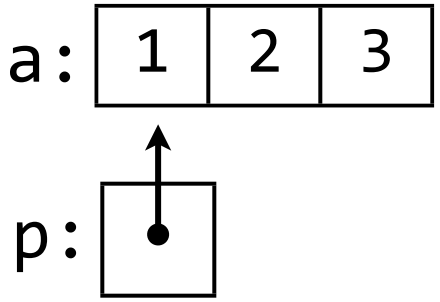



```
char s[] = "hello";
char *q = s;
```

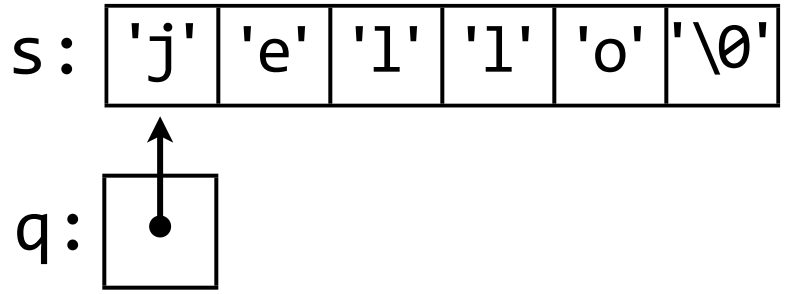


Arrays vs. Pointers

```
int a[] = {1, 2, 3};
int *p = a;
```

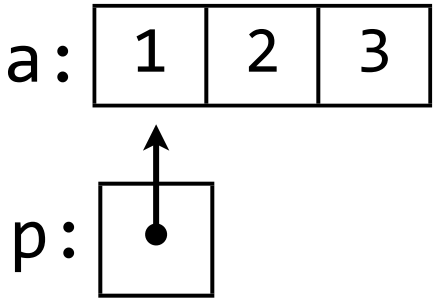



```
char s[] = "hello";
char *q = s;
*q = 'j';
```

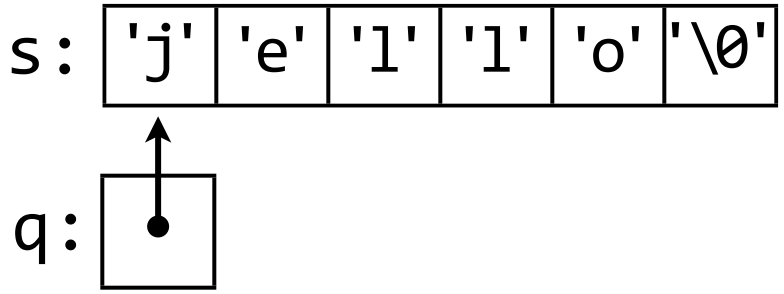


Arrays vs. Pointers

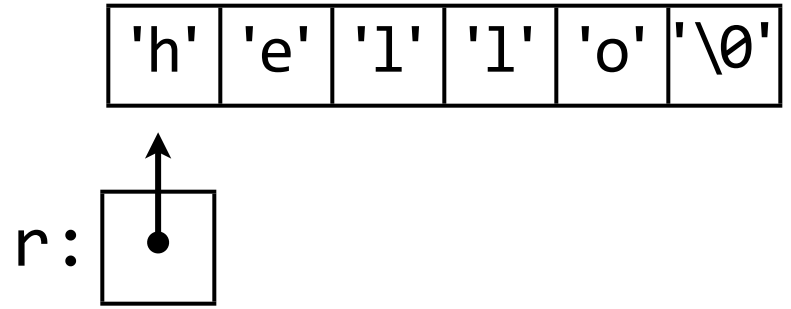
```
int a[] = {1, 2, 3};
int *p = a;
```




```
char s[] = "hello";
char *q = s;
*q = 'j';
```

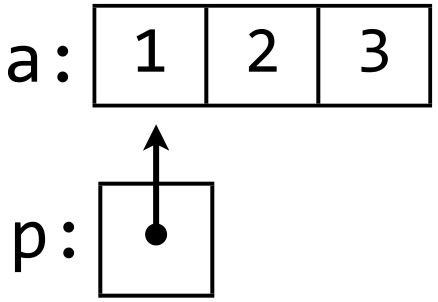



```
char *r = "hello";
```

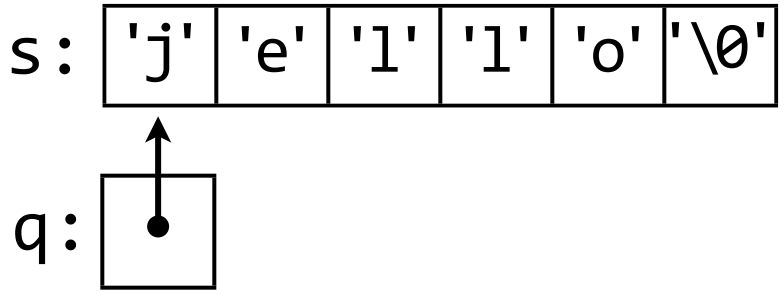


Arrays vs. Pointers

```
int a[] = {1, 2, 3};
int *p = a;
```

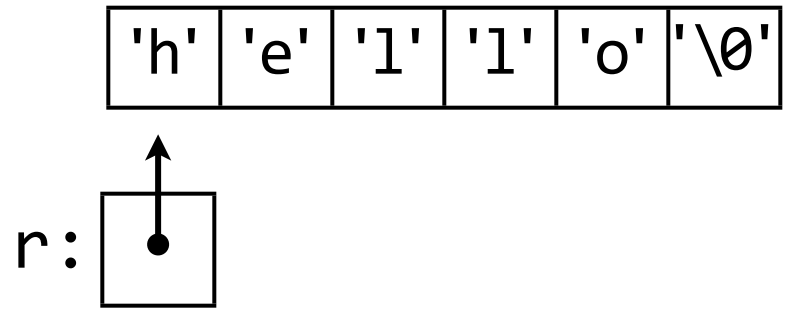



```
char s[] = "hello";
char *q = s;
*q = 'j';
```




```
char *r = "hello";
*r = 'j';
```

Wrong



Arrays vs. Pointers

- We'll use both styles:

```
char string[10];
char *p = string;

string[0] = 'H';
string[1] = 'i';
string[2] = '\0';

*p = 'P';
```

C String Issues

- Need to decide maximum string length in advance

```
#define STR_LEN 40
...
char s[STR_LEN + 1];
```


for the '\0'

Warning!

Shorter is Fine

```
#define STR_LEN 13
...
char name[STR_LEN + 1] = "Jonathan";
```

'J'	'o'	'n'	'a'	't'	'h'	'a'	'n'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

```
int a[5] = {1,2};
int a[5] = {1,2,0,0,0};
```

```
char name[STR_LEN + 1] = "Jonathan123456";
```

'J'	'o'	'n'	'a'	't'	'h'	'a'	'n'	'1'	'2'	'3'	'4'	'5'	'6'

Legal, but not a string

C String Issues

- Strings don't know how long they are
  - Need to search for the `'\0'`
- Arrays don't know how big they are
  - Don't know if a string will fit into a given `char[]`