Slides by Andrew DeOrio

EECS 280

Programming and Introductory Data Structures

Strings and IO

Outline

- Today, we will cover 3 topics:
- Strings
  - C-style strings
  - C++ strings
- Command line arguments
  - Argv and argc
- Stream input
  - cin and fstream

Review: where does an array end?

- How do we keep pointers inside their arrays?
  - Keep track of the length separately
  - Put a sentinel value at the end of the array

- When we keep track of the length separately, we can use *traversal by index* or *traversal by pointer*

Review: traversal by index and pointer

```
int const SIZE = 5;
int array[SIZE] = {1, 2, 3, 4, 5};
```

- Traversal by index

```
for(int i=0; i < SIZE; ++i){
  cout << array[i] << endl;
  cout << *(array + i) << endl; //same thing
}
```

- Traversal by pointer

```
for(int *i=array; i < array + SIZE; ++i){
  cout << *i << endl;
}
```

Review: where does an array end?

- How do we keep pointers inside their arrays?
  - Keep track of the length separately
  - Put a sentinel value at the end of the array

- C-strings are a special use of arrays

```
char str[] = "hello";
```

Review: C-style strings

- In the old days of the C language, strings were originally represented as just an array of characters

```
char str[] = "hello";
```

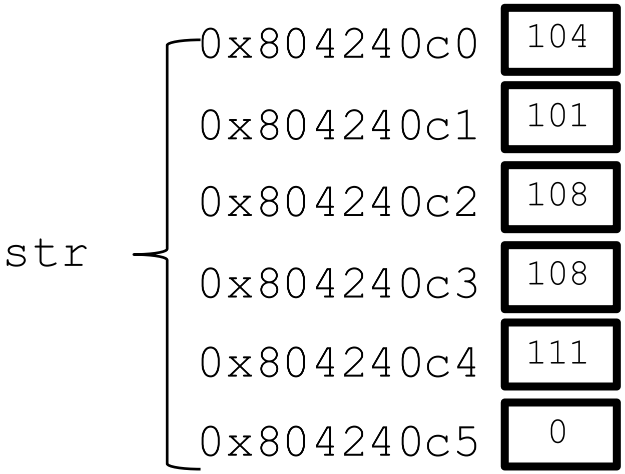
Compiler automatically puts `'\0'` at the end of string literals

- There is a **null character** at the end of every string
  - `'\0'` in code
  - ASCII value `0`
  - Acts as a **sentinel** to say “Whoa, the array stops here!”

Review: C-style strings

- In the old days of the C language, strings were originally represented as just an array of characters

```
char str[] = "hello";
```



str

0x804240c0	104
0x804240c1	101
0x804240c2	108
0x804240c3	108
0x804240c4	111
0x804240c5	0

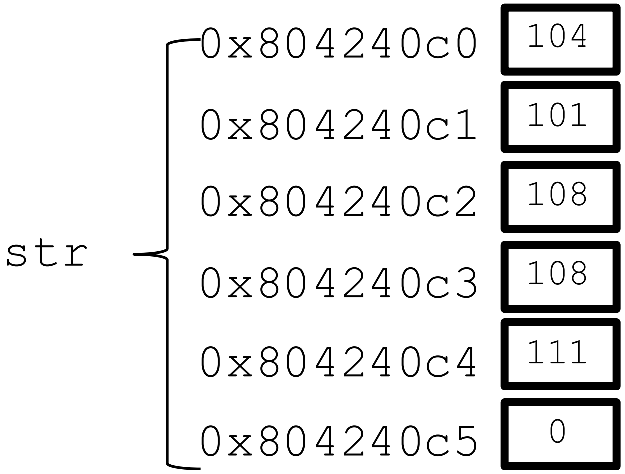
Review: C-style strings

```
char str[] = "hello";
```

- Why are the memory locations filled with numbers?
- `char` objects are really numbers under the hood (ASCII)

AKA `NULL`
AKA `false`

Symbol	Number
'\0'	0
…	
'e'	101
'f'	102
'g'	103
'h'	104
…	



`str`

0x804240c0	104
0x804240c1	101
0x804240c2	108
0x804240c3	108
0x804240c4	111
0x804240c5	0

Review: traversing a C-string

- Just keep going until we find the **sentinel**
  - When the current element has value `'\0'`

```
char str[6] = "hello";
int strlen(const char *str){
  const char *ptr = str;
  while(*ptr != '\0'){
    ++ptr;
  }
  return ptr - str;
}
```

Pointer starts at beginning of the array

Continue until sentinel is found

Increment **pointer**

Take difference to see how many steps we took. (Does not count `'\0'`.)

Review exercise: increment

Line 0:	**int x = 0;**			
Line 1:	x + 1;	x += 1;	++x;	x++;
new value of x?				

Line 0:	**int x=0, y=0;**			
Line 1:	y = x+1;	y = x+=1;	y = ++x;	y = x++;
new value of x?				
new value of y?				

```
void f(int i) { /*...*/ }
```

Line 0:	**int x = 0;**			
Line 1:	f(x + 1);	f(x += 1);	f(++x);	f(x++);
value of i				
new value of x?				

Solution

Line 0:	int x = 0;			
Line 1:	x + 1;	x += 1;	++x;	x++;
new value of x?	0	1	1	1

Line 0:	int x=0, y=0;			
Line 1:	y = x+1;	y = x+=1;	y = ++x;	y = x++;
new value of x?	0	1	1	1
new value of y?	1	1	1	0

```
void f(int i) { /*...*/ }
```

Line 0:	int x = 0;			
Line 1:	f(x + 1);	f(x += 1);	f(++x);	f(x++);
value of i	1	1	1	0
new value of x?	0	1	1	1

More on C-strings

- When you use a **string literal**, it has to be stored somewhere

```
"hello"
```

- If you declare an **array**, you are “specifying” where. It’s your array, so you can change it

```
char str[] = "hello";
```

- If you declare a **pointer**, the compiler puts it somewhere special, and you just get a pointer to it. You’re not allowed to change it

```
const char *str = "hello";
```

C-strings and cout

- We saw earlier you can’t print out arrays.

```
int array[] = {1,2,3,4};
cout << array << endl;
```

Turns into an `int*`
Prints an address, not 1,2,3,4

- But you can print out C-style strings

```
char str[] = "hello";
cout << str << endl;
```

Turns into a `char*`
Still prints out “hello”

- `cout` treats ALL `char*` as C-style strings
  - Starts printing characters until it finds a null character
  - Don’t try to print a `char*` not pointing into a c-style string!

What about C++ strings?

	C-Style Strings	C++ Strings
Library Header	`<cstring>`	`<string>`

Declaration	`char cstr[];`
	`char *cstr;`
		`string str;`

Length	`strlen(cstr);`	`str.length();`
Copy value	`strcpy(cstr1, cstr2);`	`str1 = str2;`
Indexing	`cout << cstr[i];`	`cout << str[i];`
Concatenate	`strcat(cstr1, cstr2);`	`str1 += str2`
Compare	`strcmp(cstr1, cstr2);`	`str1 == str2`

`string` to C-style string: `char *cstr = str.c_str();`

C-style string to `string`: `string str = string(cstr);`

Comparing Strings

- C++ strings
  - Just use `==, !=, <, <=, >, >=`
- C-style strings
  - Don't use built-in operators
    These will just compare addresses
  - Instead, use the `strcmp` function
  - `strcmp(A,B)` returns:

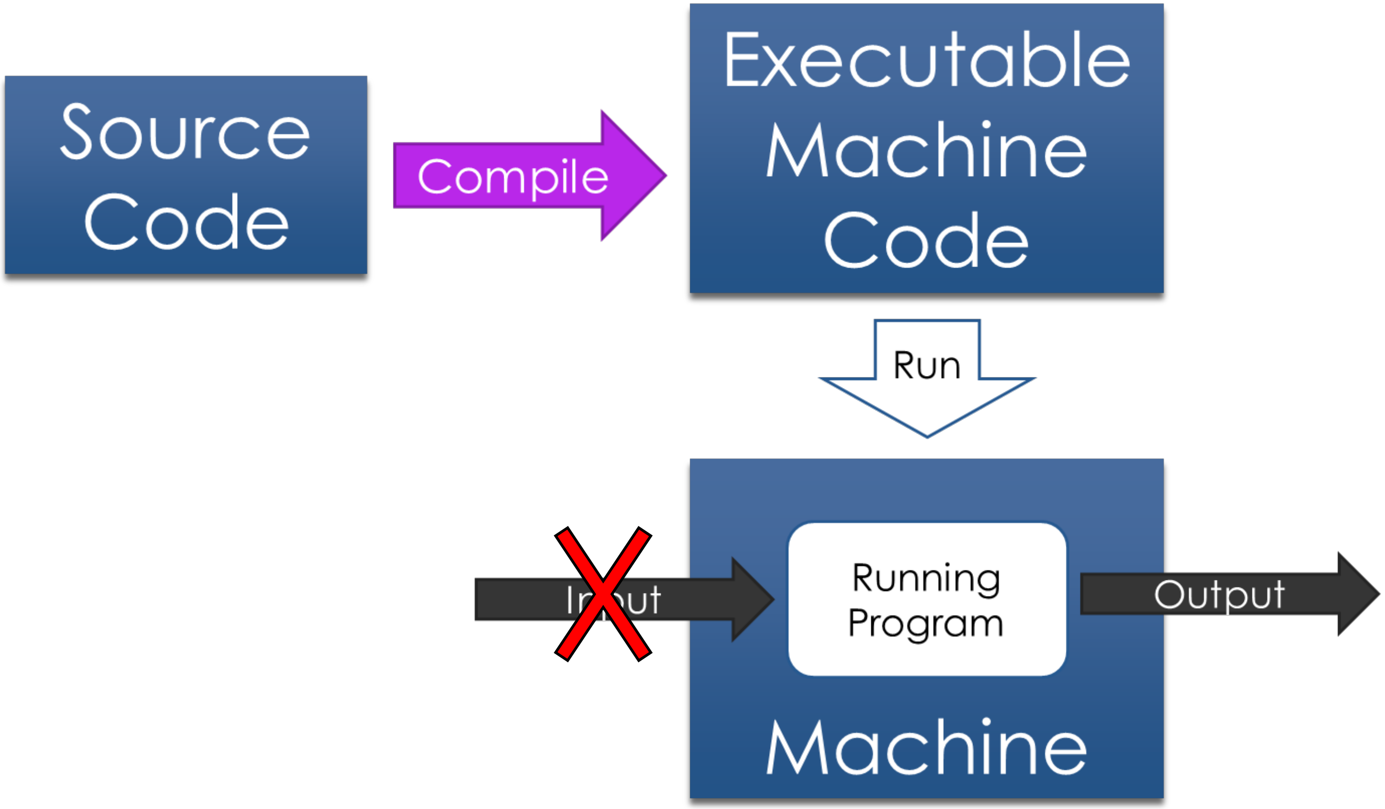
```
negative if A < B
 0 if A == B
positive if A > B
```

Outline

- Today, we will cover 3 topics:
- Strings
  - C-style strings
  - C++ strings
- **Command line arguments**
  - **Argv and argc**
- Stream input
  - cin and fstream

User input

- So far, we've considered programs that always do exactly the same thing



Argv Basics

```
$ ls p2.cpp Makefile
```

- `ls`, is the name of the program to run
- The other "words are **arguments** to the `ls` program
- The **shell** (a.k.a. terminal, console, etc.) starts the program and passes arguments
- The program gets the arguments. In C++, they are passed as parameters to `main`

argv and argc

- Two parameters to main:
  - `argc` – the number of arguments
  - `argv` – an array of the arguments

- `argv` is an **array of C-style strings**

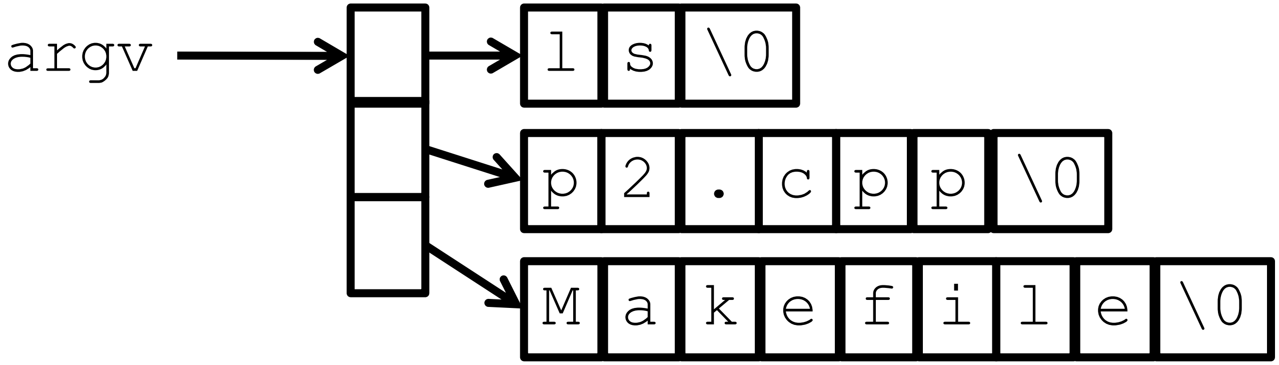
```
int main(int argc, char *argv[]) {}
```

- Many programmers write it this way (same thing):

```
int main(int argc, char **argv) {}
```

Argv in pictures

```
$ ls p2.cpp Makefile
```



<u>Note</u>: `argv[0]` is the name of the program being executed. This is because it is possible for the same program to be given different names, and do different things depending on what name it was called with.

atoi()

- The `atoi` function parses an integer value encoded in a C-style string

```
#include <cstdlib> //needed for atoi()

int atoi(const char *s);
// EFFECTS: parses s as a number and
//           returns its int value
```

Exercise

- Write a program named `sum` which adds up command line arguments
- Use `argv` and `atoi`

```
int main (int argc, char **argv) {


}
$ g++ sum.cpp –o sum
$ ./sum 1 2 3 4 5
15
$ ./sum 2 4 6 8 10
30
```

Outline

- Today, we will cover 3 topics:
- Strings
  - C-style strings
  - C++ strings
- Command line arguments
  - Argv and argc
- **Stream input**
  - **cin and fstream**

user input

hello world!
the end

cin Example

We’re already familiar with read input from standard input (cin)

```
string word;
while (cin >> word) {
  cout << "word = `" << word << "'\n";
}
```

```
$ ./a.out
hello world!
word = `hello'
word = `world!’
the end
word = `the'
word = `end'
```

fstream library

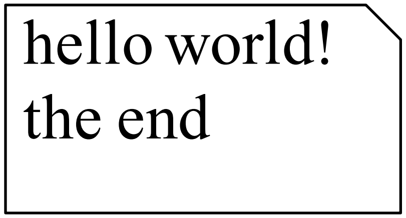
- In C++, we can read and write files directly with the `fstream` library

```
#include <fstream>
```

- `fstream` allows you to read a file just like `cin`

fstream Example

hello.txt



```
hello world!
the end
```

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

fstream Example

hello.txt

hello world!
the end

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

Open a file using `fin` variable

hello.txt

```
hello world!
the end
```

fstream Example

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open())
  cout << "open fail
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

`open()` demands a C-string. Use `filename.c_str()` to get the C-string representation.

hello.txt

```
hello world!
the end
```

fstream Example

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

Check for success opening file.

<u>hello.txt</u>

```
hello world!
the end
```

fstream Example

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

Read one word at a time and check that the read was successful.

fstream Example

hello.txt

```
hello world!
the end
```

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str()
if (!fin.is_open()) {
  cout << "open failed" <
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

```
$ ./a.out
word = `hello'
word = `world!'
word = `the'
word = `end'
```

Bad examples

hello.txt

```
hello world!
the end
```

```
while(!fin.fail()) {
  fin >> word;
  cout << word;
}
```

```
while(fin.good()) {
  fin >> buf;
  cout << word;
}
```

```
while(!fin.eof()) {
  fin >> word;
  cout << word;
}
```

```
while(fin) {
  fin >> buf;
  cout << word;
}
```

```
$ ./a.out
hello
world!
the
end
end
```

- Last line is printed twice!
- This is because it takes one extra "failed" read to realize that you're at the end of the file (if there's a trailing newline).

fstream Example

hello.txt

```
hello world!
the end
```

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word;
while (fin >> word) {
  cout << "word = `" << word << "'\n";
}
fin.close();
```

Close file after reading is finished.

fstream Example

hello.txt

```
hello world!
the end
```

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}
string word1, word2;
while (fin >> word1 >> word2) {
  cout << "word1 = `" << word1 << "'\n"
       << "word2 = `" << word2 << "'\n";
}
fin.close();
```

Alternative: read two words at a time.

hello.txt

```
hello world!
the end
```

fstream Example

```
string filename = "hello.txt";
ifstream fin;
fin.open(filename.c_str());
if (!fin.is_open()) {
  cout << "open failed" << endl;
  exit(1);
}

string line;
while (getline(fin, line)) {
  cout << “line = `" << line << "'\n“;
}
fin.close();
```

Alternative: read one line at a time.

```
$ ./a.out
line = `hello world!'
line = `the end'
```

Reading numbers

numbers.txt

```
1
42
```

```
ifstream fin;
// open and error check fin

int i;
while (fin >> i) {
  cout << "i = " << i << endl;
}
// close fin and exit
```

Read a number directly from a file stream

```
$ ./a.out
i = 1
i = 42
```

Exercise

- Write a program named `sum` which adds up numbers in a file

```
int main (int argc, char **argv) {


}
```

```
$ ./sum data.txt
sum is 3.14159
```

data.txt

```
-2
2.0
3.14159
```