EECS 280 – Lecture 6

Strings, Streams, and I/O

9/22/2021

Agenda

- Strings
  - C-Style Strings
  - C++ `strings`
- Command Line Arguments
  - `argv` and `argc`
- Stream Input and Output
  - `cin` and `fstreams`

Where does the array end?

- What happens if a pointer wanders outside of its array and you use it?
  - Undefined behavior!
  - You end up reading/writing random memory.
  - Program might crash. Or maybe not.
    Or maybe only sometimes.

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

C-Style Strings

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

```
char str1[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
char str2[6] = "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 string stops here!"

- Of course, character arrays turn into pointers as well.

```
char *strPtr = str1;
```

C-style Strings are char Arrays

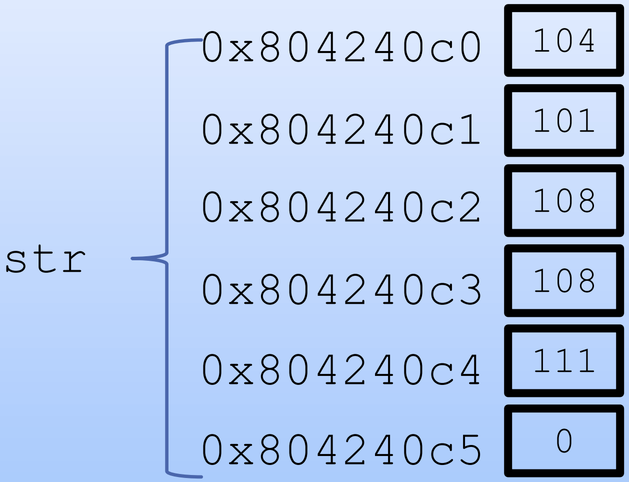
- char values are just numbers underneath

Null character is the sentinel. It has value 0.

ASCII Codes

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

```
char str[6] = "hello"
```



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

Compiler automatically adds sentinel to string literals.

Be Careful with C-Style Strings

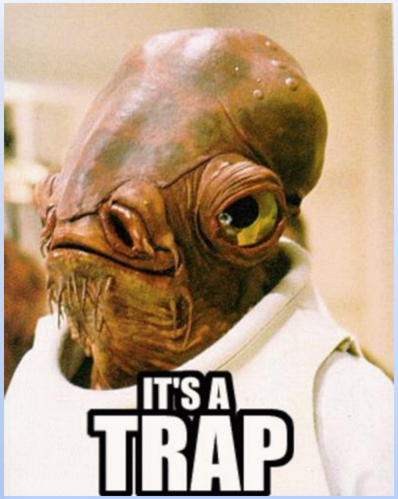
- This code doesn't do what it first appears to. Remember, they turn into pointers.

```
char str1[6] = "hello";
char str2[6] = "hello";
char str3[6] = "apple";
char *ptr = str1;

// Test for equality?
str1 == str2;

// Copy strings?
str1 = str3;

// Copy through pointer?
ptr = str3;
```

Actually tests if at the same address.

Doesn't compile. Type mismatch.

Makes ptr point to different string.



Declaring C-Style Strings

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

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

```
char str[6] = "hello";
```

- If you declare a **pointer**, you’re not allowed to change the contents, because the compiler just gives you a pointer to the string literal.

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

C-Style Strings and cout

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

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

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

- But you can print out C-style strings.

```
char str[6] = "hello";
cout << str << endl;
```

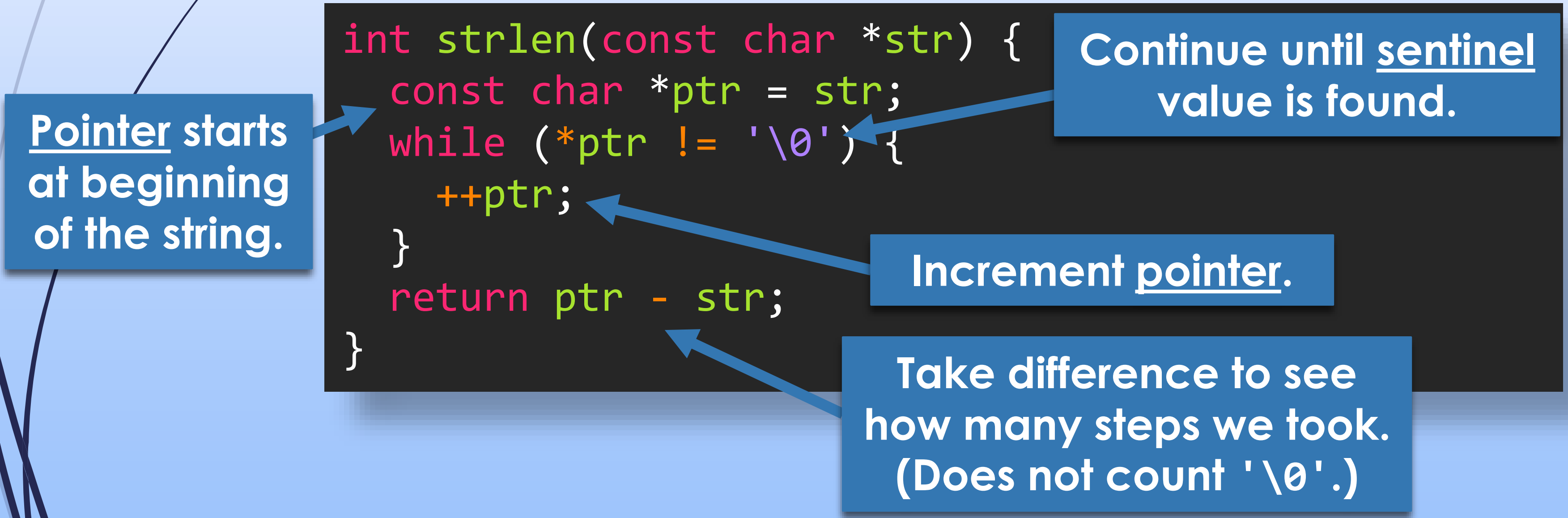
Turns into a `char*`. Prints out "hello".

- `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!

Example: strlen() function

```
char str[6] = "hello";
cout << strlen(str) << endl; // Prints 5
```

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



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

Pointer starts at beginning of the string.

Continue until sentinel value is found.

Increment pointer.

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

See file L05.1_strlen on Lobster (lobster.eecs.umich.edu).

Example: count() function

```
char str[6] = "hello";
cout << count(str, 'e') << endl; // Prints 1
cout << count(str, 'l') << endl; // Prints 2
```

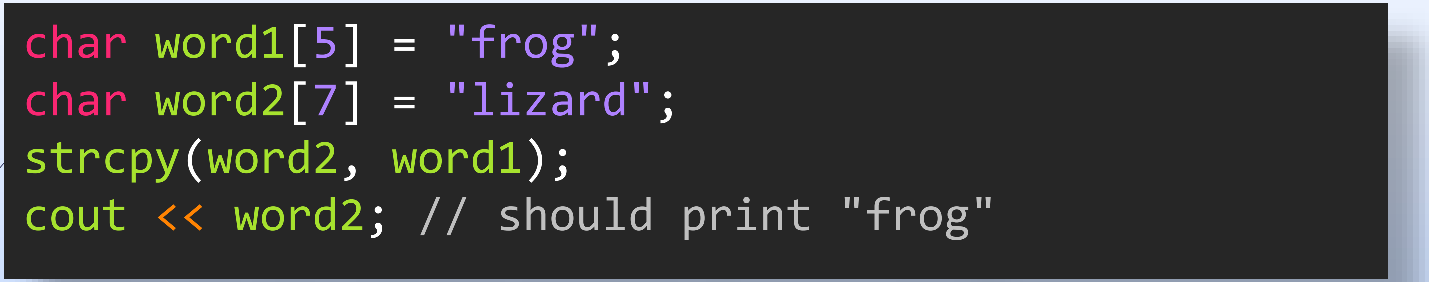
```
int count(const char *str, char c) {
  int count = 0;
  while (*str) {
    if (*str == c) {
      ++count;
    }
    ++str;
  }
  return count;
}
```

See file `L05.1_count` on Lobster (lobster.eecs.umich.edu).

Exercise: strcpy

- L05.2_strcpy on Lobster.

  lobster.eecs.umich.edu

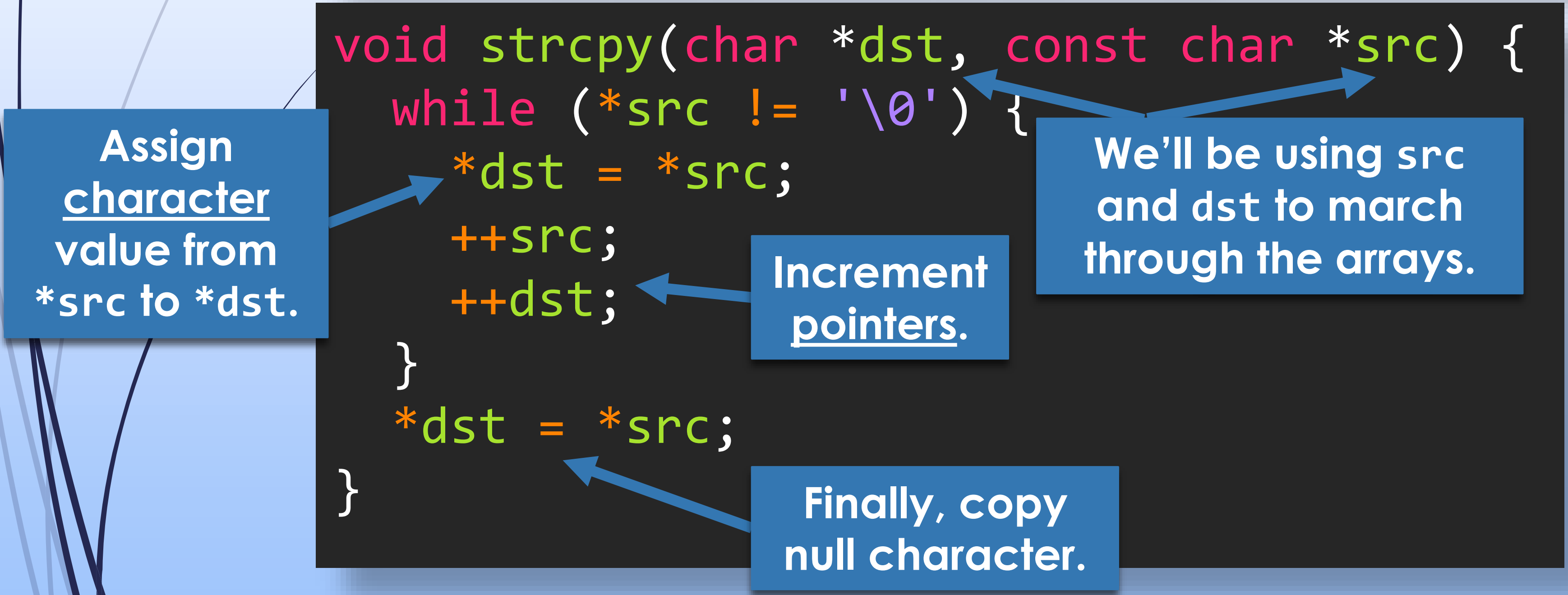


```
char word1[5] = "frog";
char word2[7] = "lizard";
strcpy(word2, word1);
cout << word2; // should print "frog"
```

- Write the function strcpy.
- main already contains a test.
- Use traversal by pointer.
  - This is customary for working with C-style strings.

Solution: strcpy

```
char word1[5] = "frog";
char word2[7] = "lizard";
strcpy(word2, word1);
cout << word2; // should print "frog"
```



```
void strcpy(char *dst, const char *src) {
  while (*src != '\0') {
    *dst = *src;
    ++src;
    ++dst;
  }
  *dst = *src;
}
```

Assign character value from *src to *dst.

We’ll be using src and dst to march through the arrays.

Increment pointers.

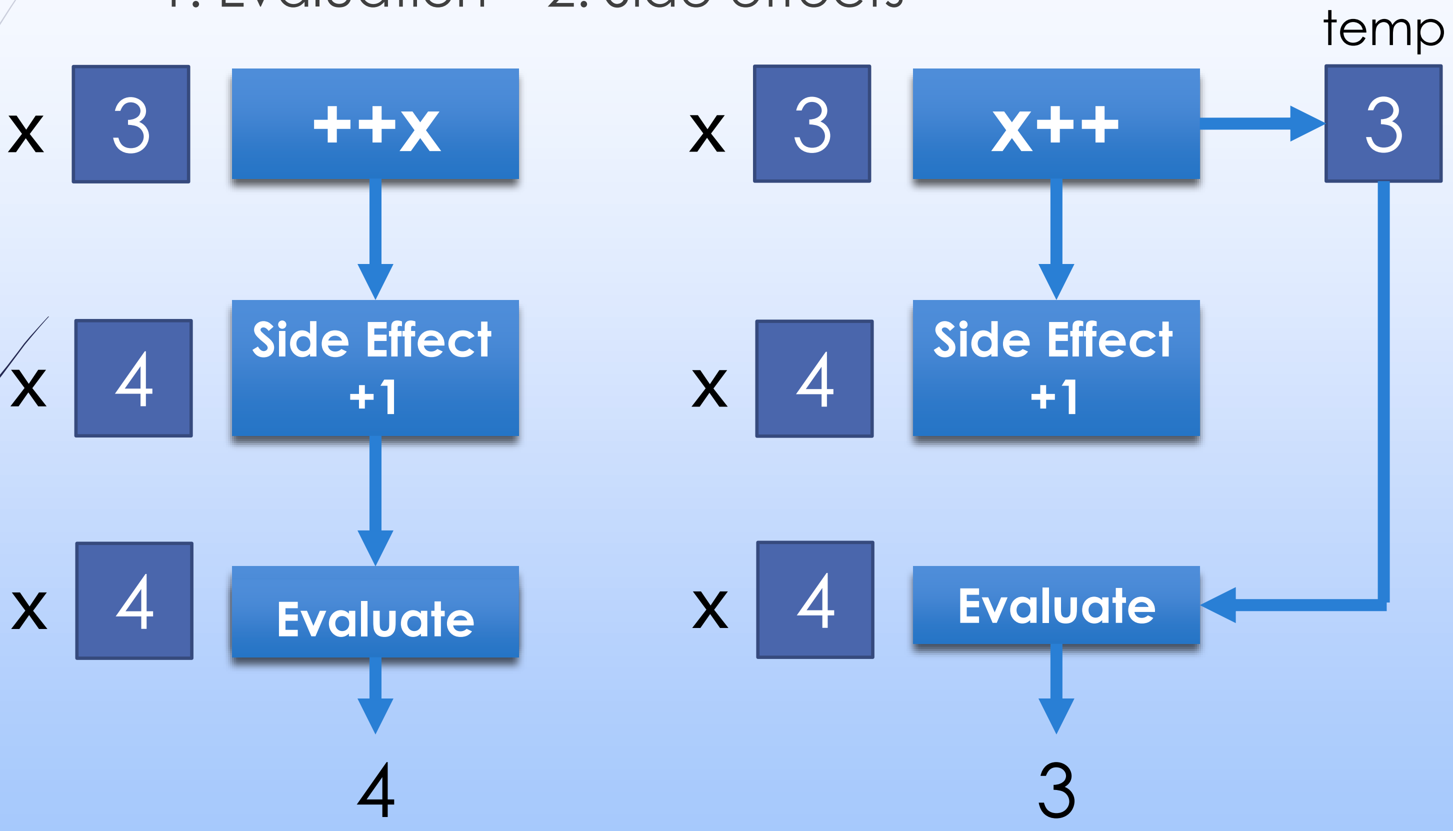
Finally, copy null character.

The standard library’s `strcpy` returns the address that was passed in for the first parameter. There are a few uses where this is “convenient”. For simplicity, our version returns `void`.

Prefix vs. Postfix Increment

- Parts of an expression

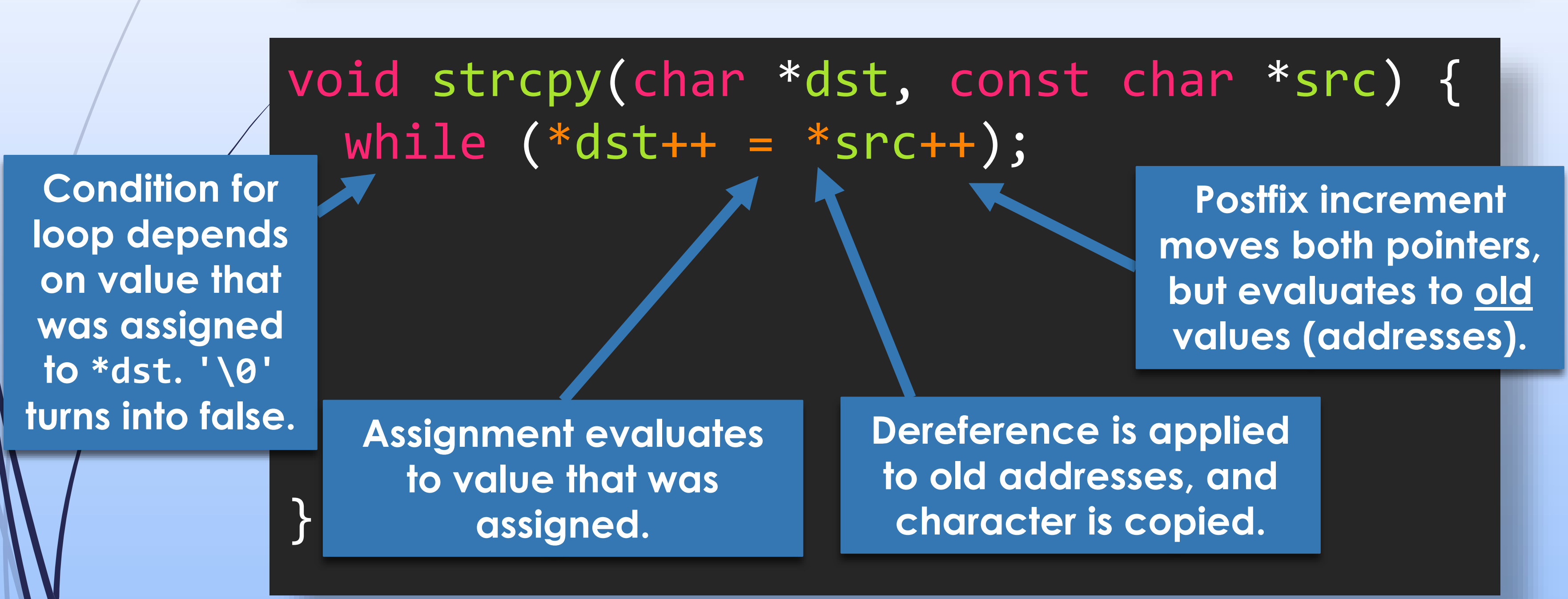
1. Evaluation 2. Side effects



Note: x += 1 is equivalent to ++x

Reference: strcpy (Cute Version)

```
char str1[6] = "hello";
char str2[6] = "apple";
strcpy(str1, str2); // str1 array now holds "apple"
```



```
void strcpy(char *dst, const char *src) {
  while (*dst++ = *src++);
}
```

Condition for loop depends on value that was assigned to `*dst`. `'\0'` turns into false.

Assignment evaluates to value that was assigned.

Dereference is applied to old addresses, and character is copied.

Postfix increment moves both pointers, but evaluates to old values (addresses).

The standard library’s `strcpy` returns the address that was passed in for the first parameter. There are a few uses where this is “convenient”. For simplicity, our version returns `void`.

Reference: 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	`cstr[i]`	`str[i]`
Concatenate	`strcat(cstr1, cstr2)`	`str1 += str2`
Compare	`strcmp(cstr1, cstr2)`	`str1 == str2`

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

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

http://www.cplusplus.com/reference/string/string/

Comparing Strings

- C++ strings
  - Just use `==`, `!=`, `<`, `<=`, `>`, `>=`

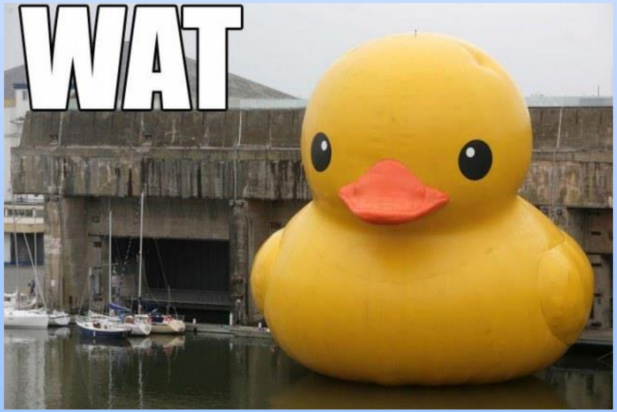
- C-style strings
  - Don’t use built-in operators.
    These will just compare addresses.
  - Instead, use the `strcmp` function.
  - `strcmp(A,B)` returns:

```
negative  if A less than B
       0  if A equal to B[1]
positive  if A greater than B
```

[1] Some people like to check for equality with `!strcmp(A,B)` since it evaluates to true if they are equal.

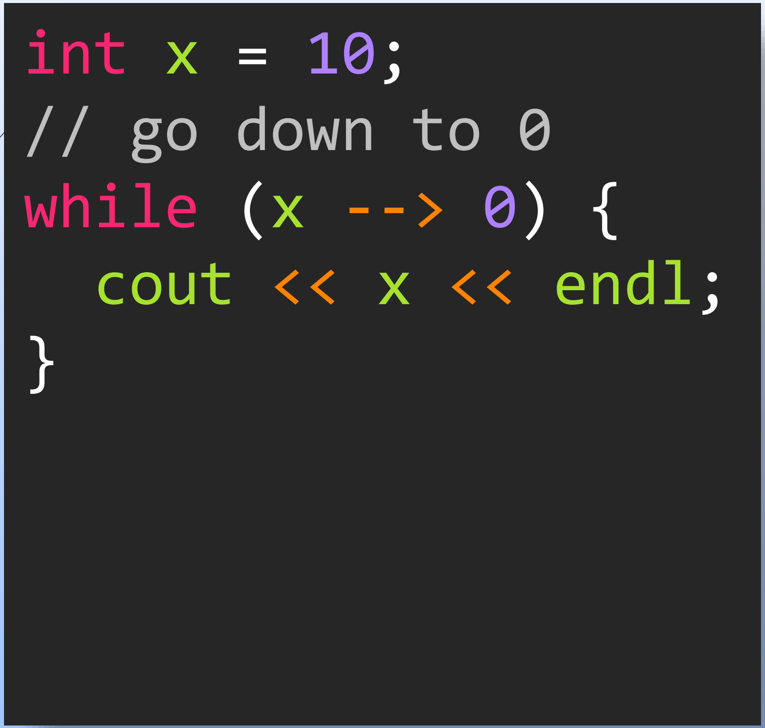
Break Time!

Legal	Illegal
++++x	x++++
(++x)++	++x++
-(--x)	---x
x---x	x----x



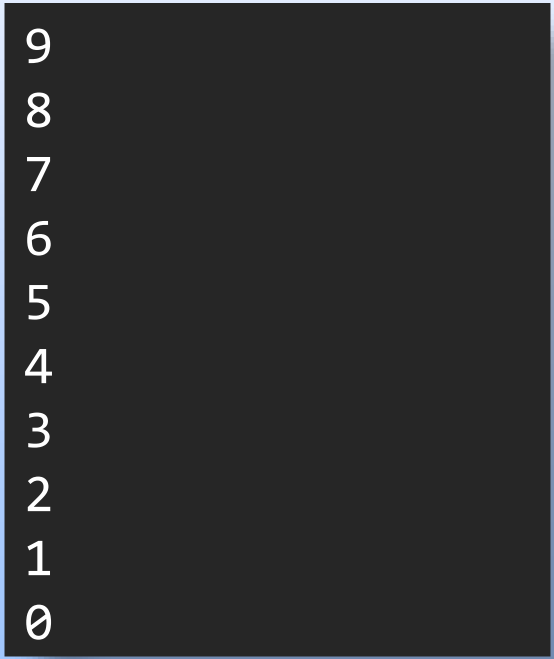
Did you know?

--> is the “countdown” operator.



```
int x = 10;
// go down to 0
while (x --> 0) {
  cout << x << endl;
}
```

Output



```
9
8
7
6
5
4
3
2
1
0
```

Disclaimer: This is a joke. (But the code does work…why??)

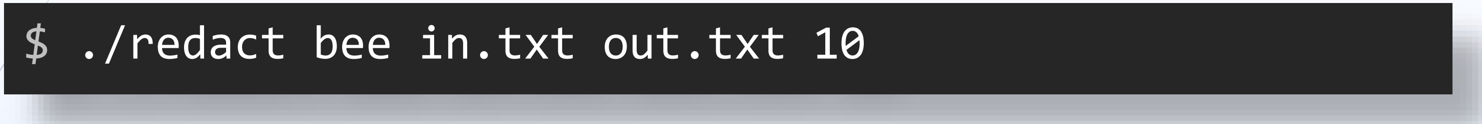
Break Time!

We’ll start again in one minute.

Agenda

- Strings
  - C-Style Strings
  - C++ `strings`
- **Command Line Arguments**
  - `argv` and `argc`
- Stream Input and Output
  - `cin` and `fstreams`

Command Line Arguments



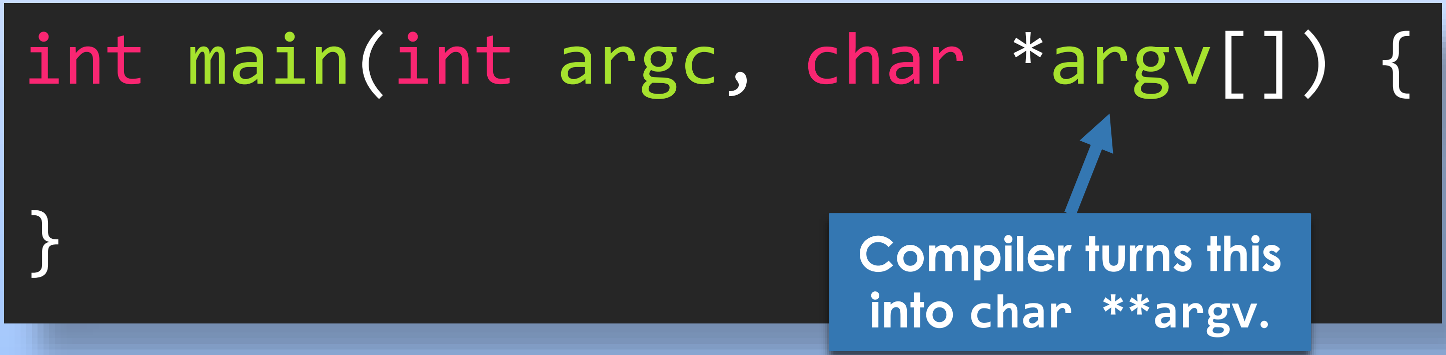
```
$ ./redact bee in.txt out.txt 10
```

- `redact` is the name of the program to run.
- The other “words” are **arguments** to the `redact` 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**.

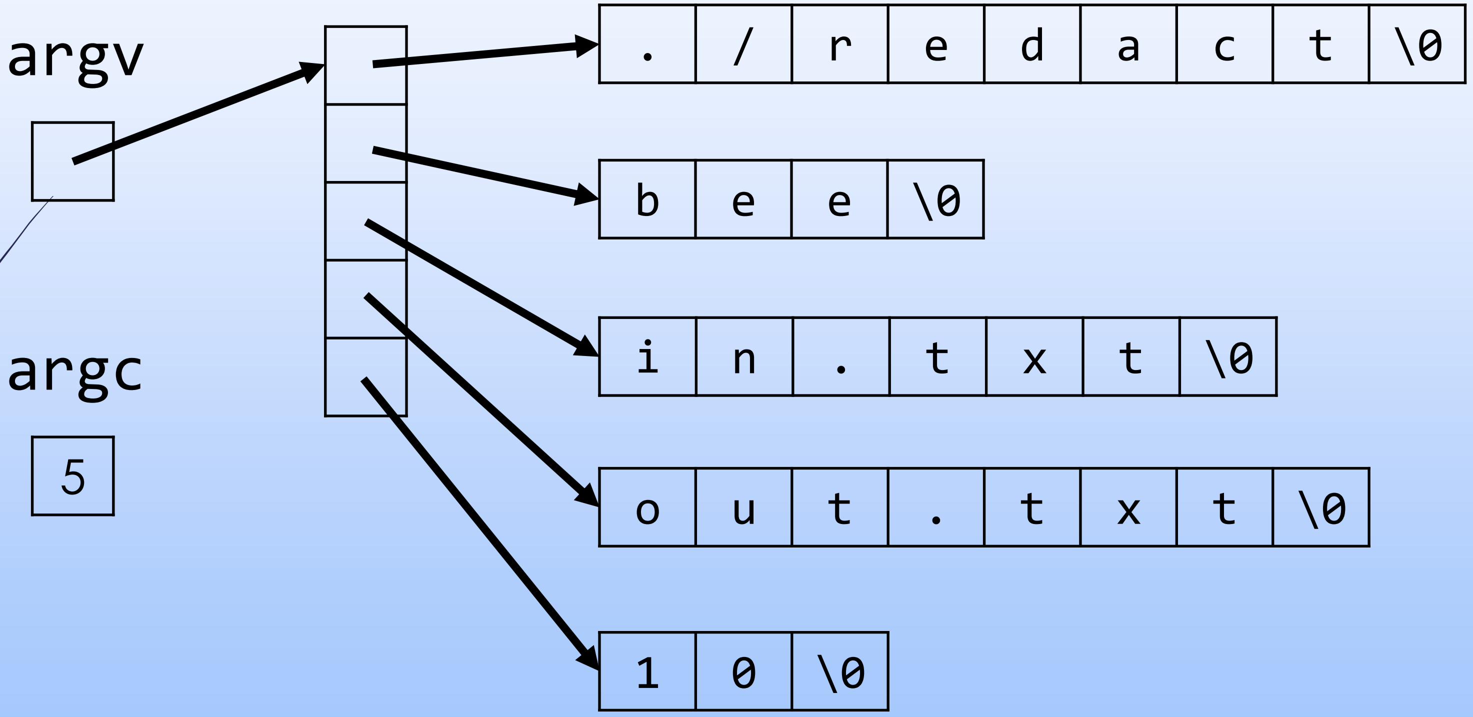


```
int main(int argc, char *argv[]) {

}
```

Compiler turns this into char **argv.

argv and argc

```
$ ./redact bee in.txt out.txt 10
```



Note: `argv[0]` is the name of the program being executed. This is useful 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.

Exercise: argv and argc

- What is the output when the program is compiled and then run with the command line arguments as shown?

program.cpp

```
int main(int argc, char *argv[]) {
  cout << argc        << " ";
  cout << argv[1]     << " ";
  cout << *argv[1]    << " ";
  cout << argv[2] + 1 << " ";
}
```

```
$ g++ program.cpp -o program
$ ./program cat dog lobster
```

Question

A) 3 cat c lobster

B) 4 ./program . dog

C) 4 cat c og

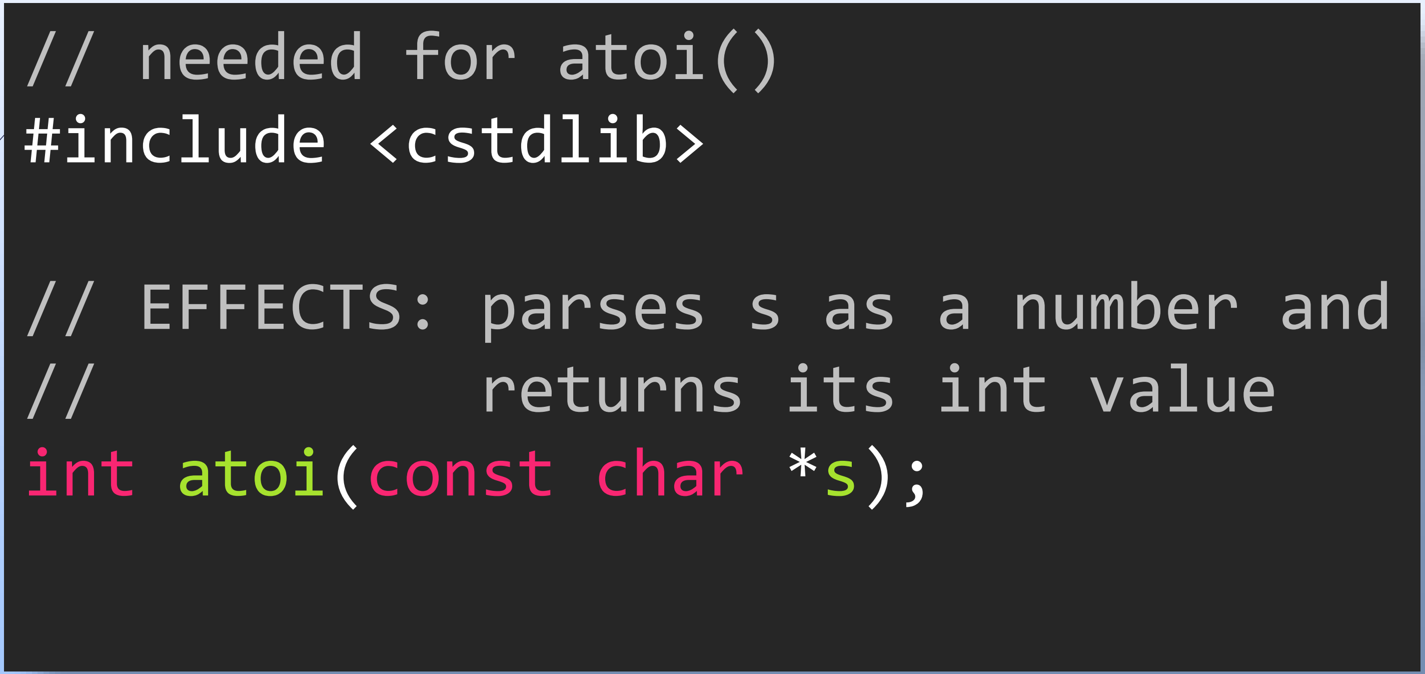
D) 3 0x1000 c 0x1001

E) 4 ./program . eph

Hint: Draw a picture of argc and argv, like the previous slide.
Hint: Recall, `cout` will print out any `char*` as a cstring.
Hint: `*argv[1]` is the same as `*(argv[1])`

atoi

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



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

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

Exercise: sum using argv

- Goal: Add up command line arguments.

```
$ g++ sum.cpp -o sum
$ ./sum 1 2 3 4 5
sum is 15
```

Question

Which implementation is correct?

A

```
int main(int argc, char *argv[]) {
  int sum = 0;
  for (int i = 0; i < argc; ++i) {
    sum += atoi(argv[i]);
  }
  cout << "sum is " << sum << endl;
}
```

B

```
int main(int argc, char *argv[]) {
  int sum = 0;
  for (int i = 1; i < argc; ++i) {
    sum += atoi(argv[i]);
  }
  cout << "sum is " << sum << endl;
}
```

C

```
int main(int argc, char *argv[]) {
  int sum = 0;
  for (int i = 1; i < argc; ++i) {
    sum += (int) argv[i];
  }
  cout << "sum is " << sum << endl;
}
```

D

```
int main(int argc, char *argv[]) {
  char *sum = argv;
  for (int i = 0; sum!='\0'; ++i) {
    sum += argv[i];
  }
  cout << "sum is " << sum << endl;
}
```

Agenda

- Strings
  - C-Style Strings
  - C++ `strings`
- Command Line Arguments
  - `argv` and `argc`
- **Stream Input and Output**
  - `cin` and `fstreams`

cin Example

- We're already familiar with reading input from standard input (cin).

user types

hello world!
goodbye
ctrl+d

words.cpp

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

Will stop when an "end of file" character is read. To type this at the console, use ctrl+d.

```
$ g++ words.cpp -o words

$ ./words
hello world!
word = 'hello'
word = 'world!'
goodbye
word = 'goodbye'
```

cin Example

- You can also use input redirection to send the contents of a file to `cin`.

words.in

```
hello world!
goodbye
```

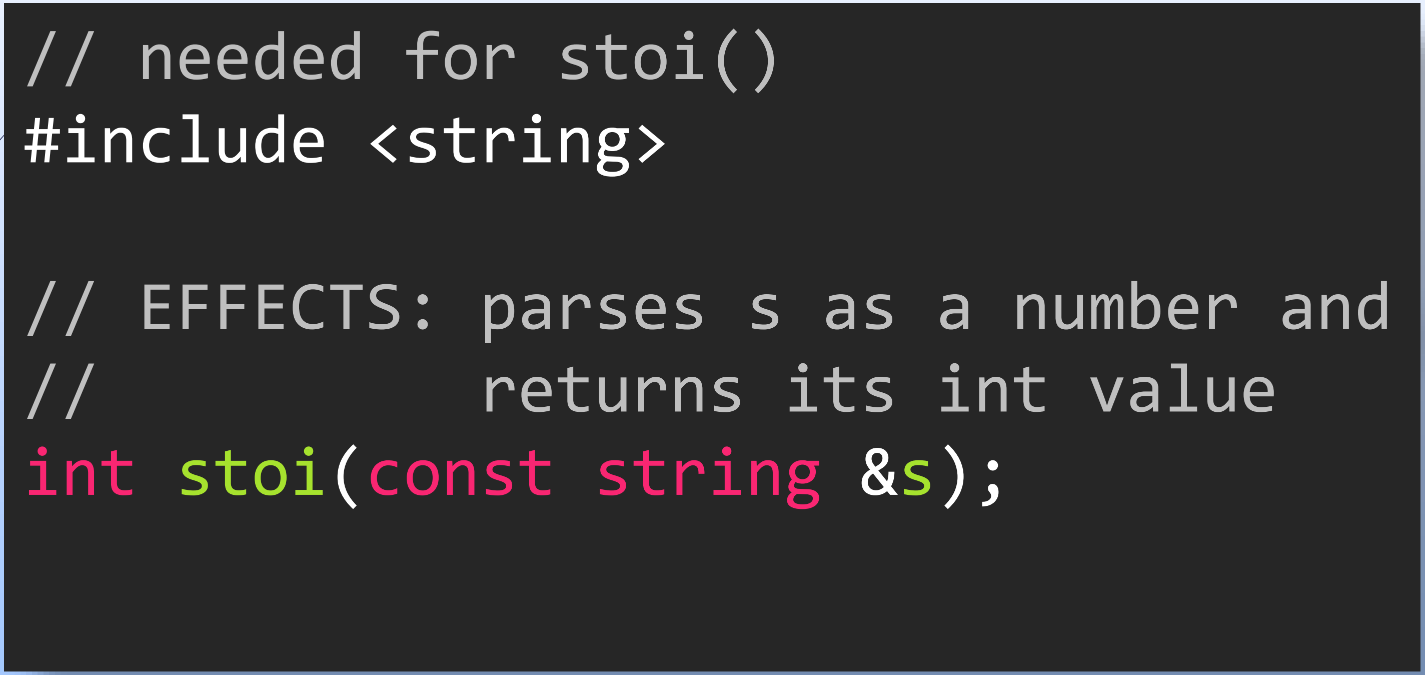
words.cpp

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

```
$ g++ words.cpp -o words

$ ./words < words.in
word = 'hello'
word = 'world!'
word = 'goodbye'
```

stoi

- The stoi function parses an integer value encoded in a C++ string.



```
// needed for stoi()
#include <string>

// EFFECTS: parses s as a number and
//           returns its int value
int stoi(const string &s);
```

Example: sum using cin

- We could also write a different sum program that takes numbers via cin until the user types "done".

```
$ ./sum
Enter some numbers to sum.
2
4
6
done
sum is 12
```

```
int main() {
  int sum = 0;
  string word;
  while (cin >> word && word != "done") {
    sum += stoi(word);
  }
  cout << "sum is " << sum << endl;
}
```

This example is on Lobster: L05.3_cin_sum

Note: You could try using cin to read directly into an int variable, but that wouldn't work to read/detect "done".

File I/O with Streams

- In C++, we can read and write files directly with `ifstream` and `ofstream` objects

```
    #include <fstream>
```

- `ifstream` and `ofstream` allow you to…
  - …read a file just like reading from `cin`
  - …write to a file just like printing to `cout`

File Input: ifstream

hello.txt

```
hello world!
goodbye
```

```cpp
int main() {
  string filename = "hello.txt";
  ifstream fin;
  fin.open(filename);
  if (!fin.is_open()) {
    cout << "open failed" << endl;
    return 1;
  }
  string word;
  while (fin >> word) {
    cout << "word = '" << word << "'" << endl;
  }
  fin.close();
}
```

Open a file using fin variable

Check for success opening file.

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

File Input: ifstream

hello.txt

```
hello world!
goodbye
```

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

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

Bad Examples

hello.txt

```
hello world!
goodbye
```

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

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

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

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

```
$ ./a.out
hello
world!
goodbye
goodbye
```

- 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.

File Input: ifstream

hello.txt

```
hello world!
goodbye
```

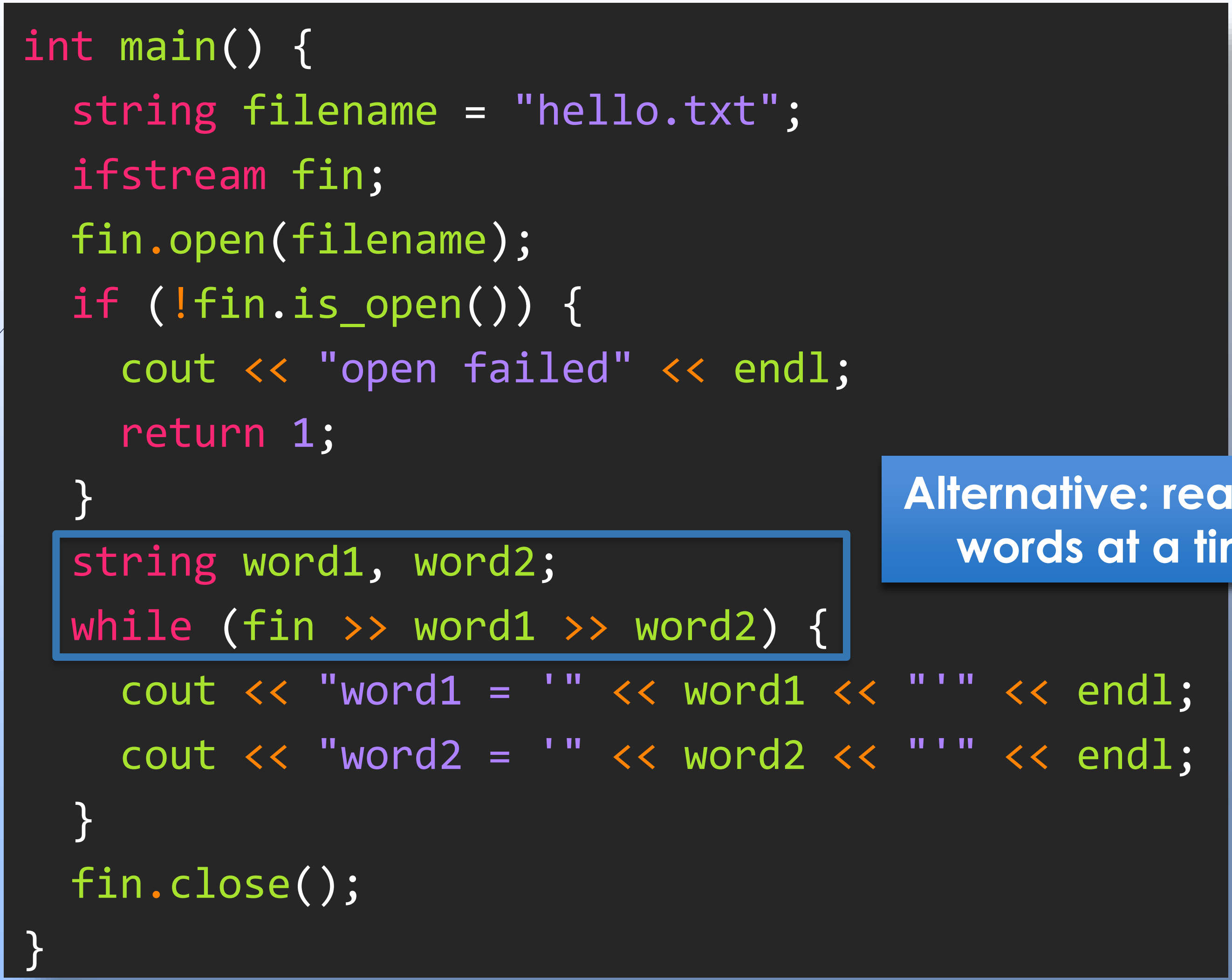
```
int main() {
  string filename = "hello.txt";
  ifstream fin;
  fin.open(filename);
  if (!fin.is_open()) {
    cout << "open failed" << endl;
    return 1;
  }
  string word;
  while (fin >> word) {
    cout << "word = '" << word << "'" << endl;
  }
  fin.close();
}
```

Close file after reading is finished. (This is optional; the file will close automatically when `fin` goes out of scope.)

File Input: ifstream

hello.txt

```
hello world!
goodbye
```



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

Alternative: read two words at a time.

File Input: ifstream

hello.txt

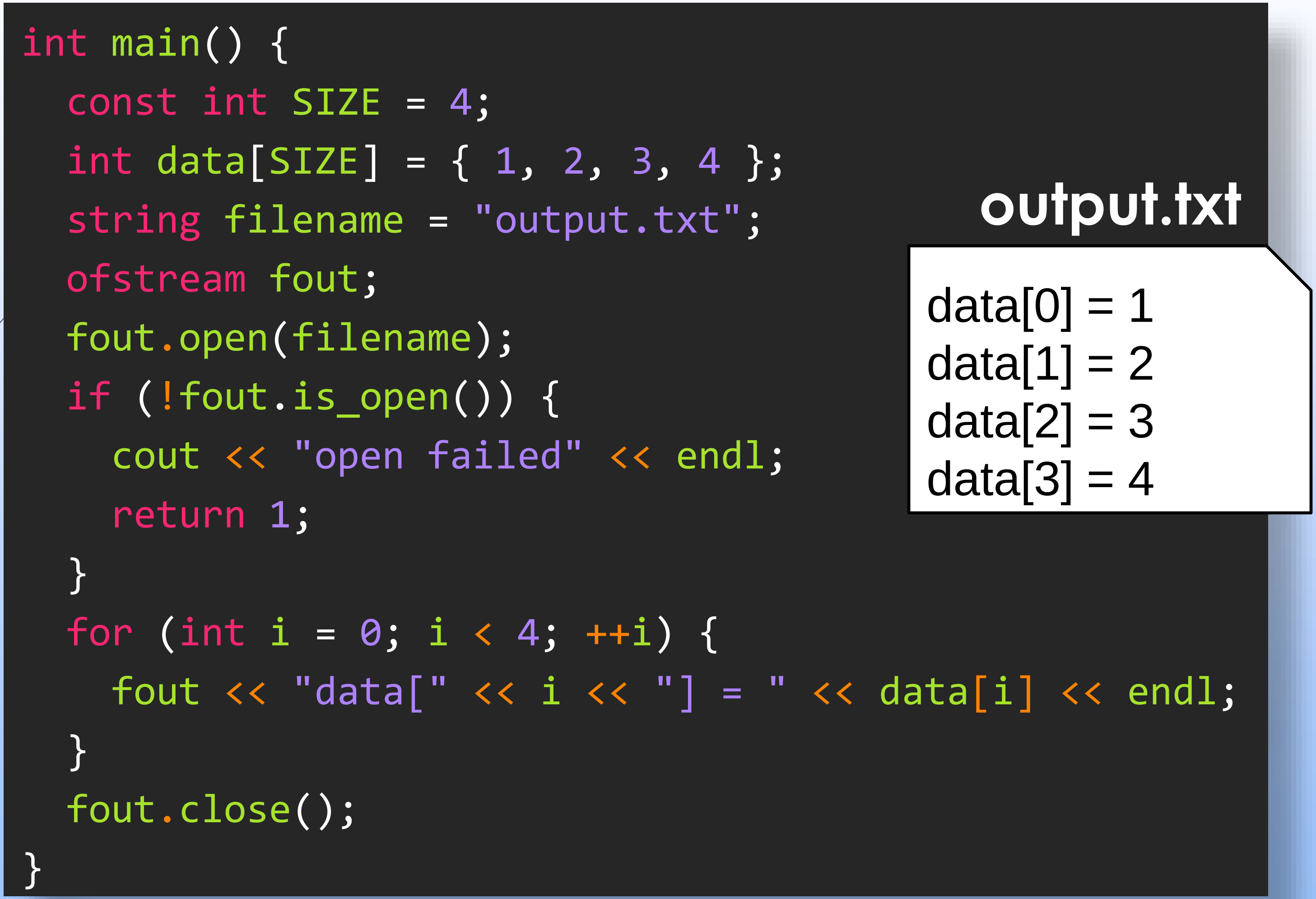
```
hello world!
goodbye
```

```cpp
int main() {
  string filename = "hello.txt";
  ifstream fin;
  fin.open(filename);
  if (!fin.is_open()) {
    cout << "open failed" << endl;
    return 1;
  }
  string line;
  while (getline(fin, line)) {
    cout << "line = '" << line << "'" << endl;
  }
  fin.close();
}
```

Alternative: read one line at a time.

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

File Output: ofstream

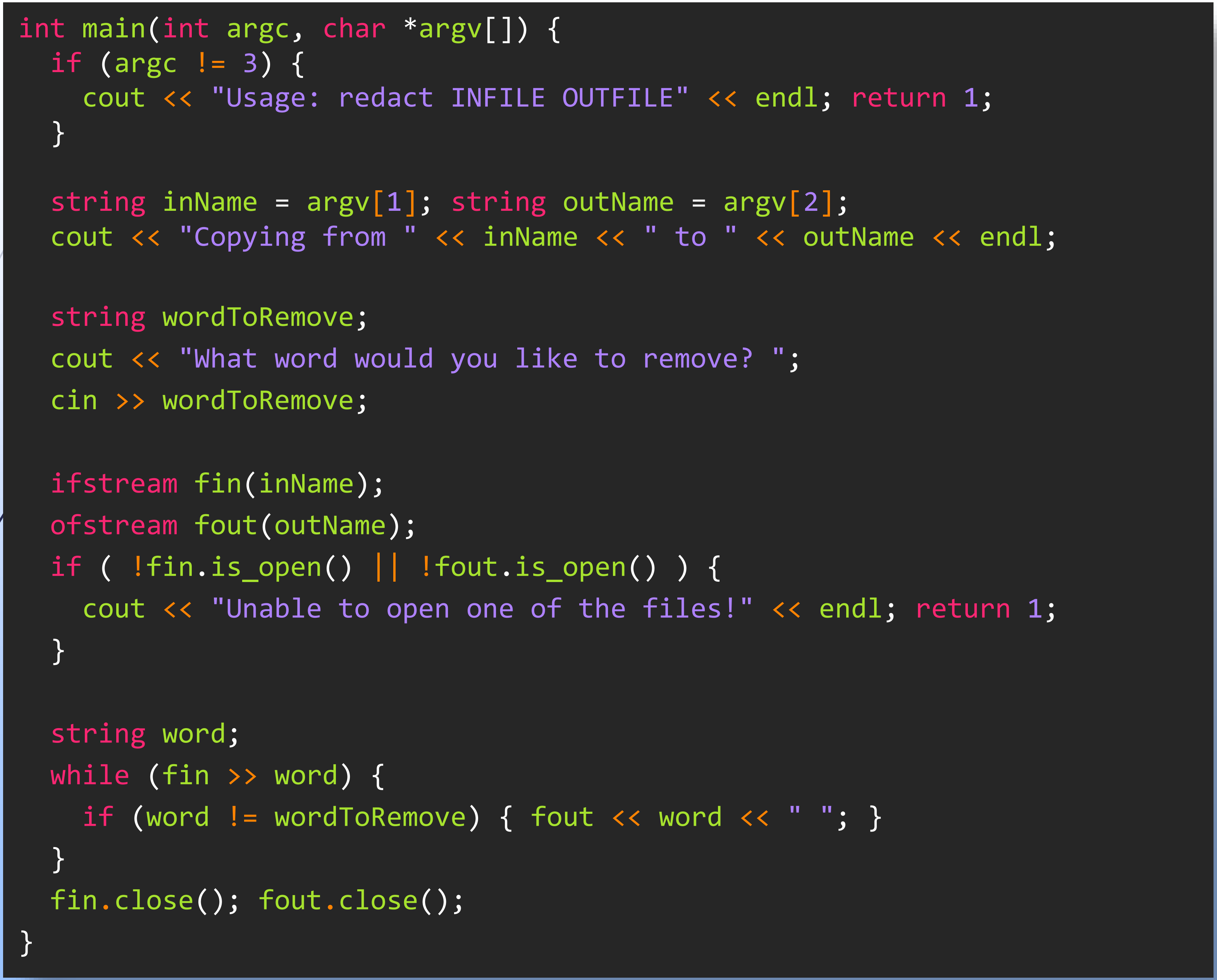


```
int main() {
  const int SIZE = 4;
  int data[SIZE] = { 1, 2, 3, 4 };
  string filename = "output.txt";
  ofstream fout;
  fout.open(filename);
  if (!fout.is_open()) {
    cout << "open failed" << endl;
    return 1;
  }
  for (int i = 0; i < 4; ++i) {
    fout << "data[" << i << "] = " << data[i] << endl;
  }
  fout.close();
}
```

output.txt

```
data[0] = 1
data[1] = 2
data[2] = 3
data[3] = 4
```

Reference: Big Example



```
int main(int argc, char *argv[]) {
  if (argc != 3) {
    cout << "Usage: redact INFILE OUTFILE" << endl; return 1;
  }

  string inName = argv[1]; string outName = argv[2];
  cout << "Copying from " << inName << " to " << outName << endl;

  string wordToRemove;
  cout << "What word would you like to remove? ";
  cin >> wordToRemove;

  ifstream fin(inName);
  ofstream fout(outName);
  if ( !fin.is_open() || !fout.is_open() ) {
    cout << "Unable to open one of the files!" << endl; return 1;
  }

  string word;
  while (fin >> word) {
    if (word != wordToRemove) { fout << word << " "; }
  }
  fin.close(); fout.close();
}
```