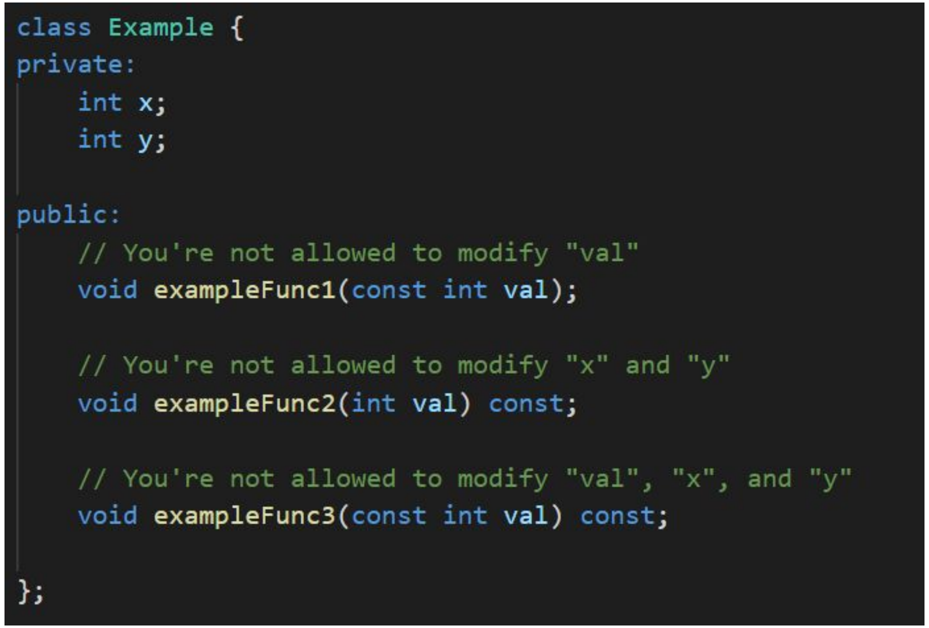
Discussion 6!

Strings, Argc/Argv, Streams

More on const

“const” in Class Functions

- const parameter: can’t change the parameter
- const after function name: can’t change the *internal member variables*
- You can combine them for maximum safety!



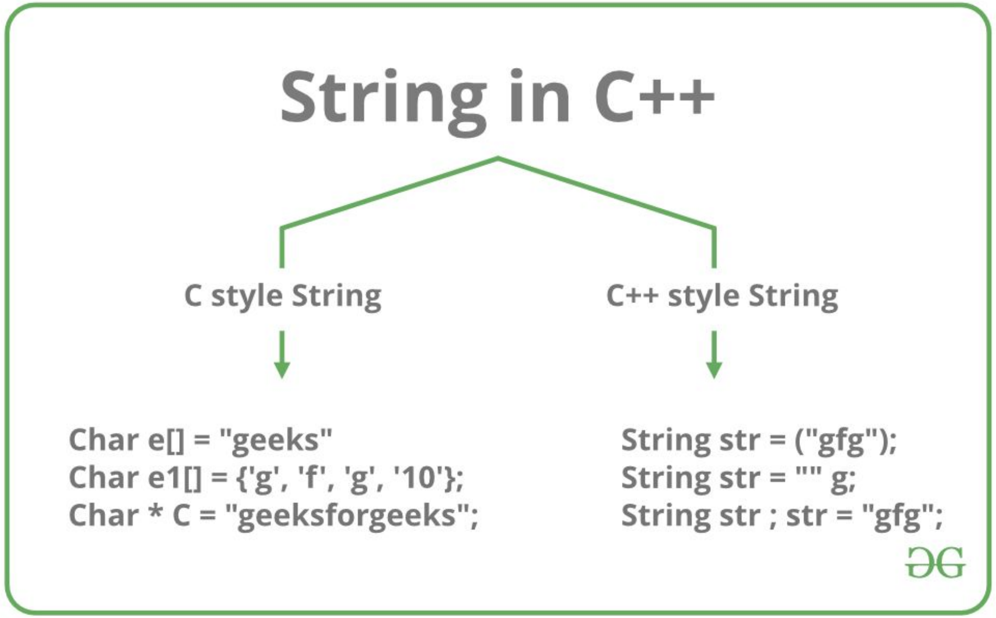
```
class Example {
private:
    int x;
    int y;

public:
    // You're not allowed to modify "val"
    void exampleFunc1(const int val);

    // You're not allowed to modify "x" and "y"
    void exampleFunc2(int val) const;

    // You're not allowed to modify "val", "x", and "y"
    void exampleFunc3(const int val) const;

};
```

Strings

Strings

Strings are **character arrays**

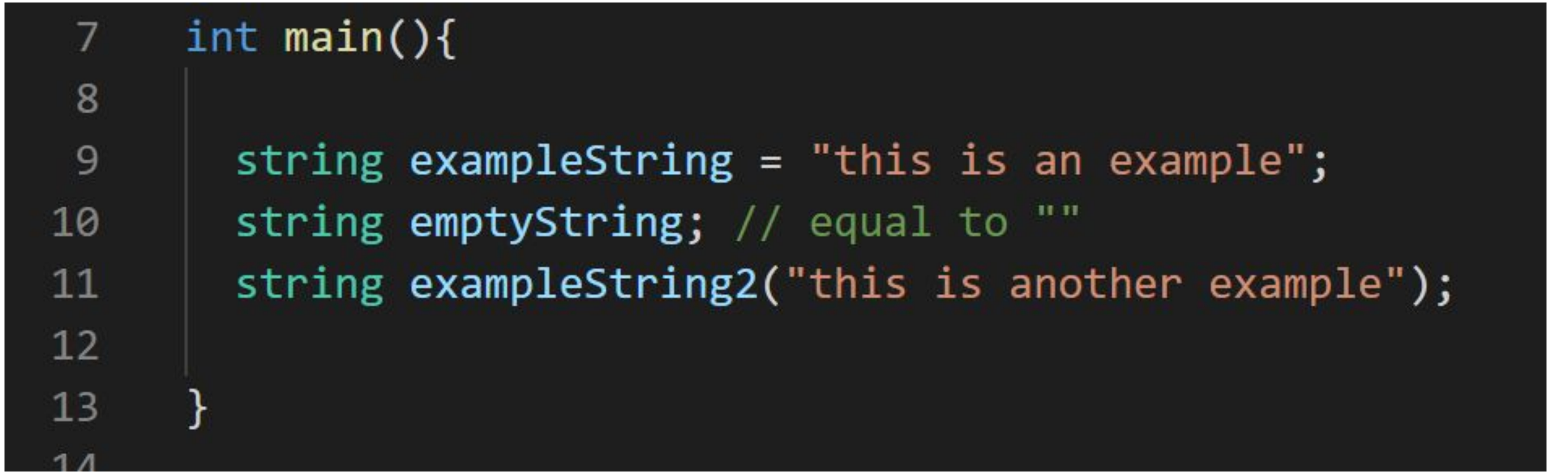
- Made so you don't have to use char[] arrays
- Provides **lots** of additional functionality and utility
  - Ex. size()



```
Char e[] = "geeks"
Char e1[] = {'g', 'f', 'g', '10'};
Char * C = "geeksforgeeks";
```

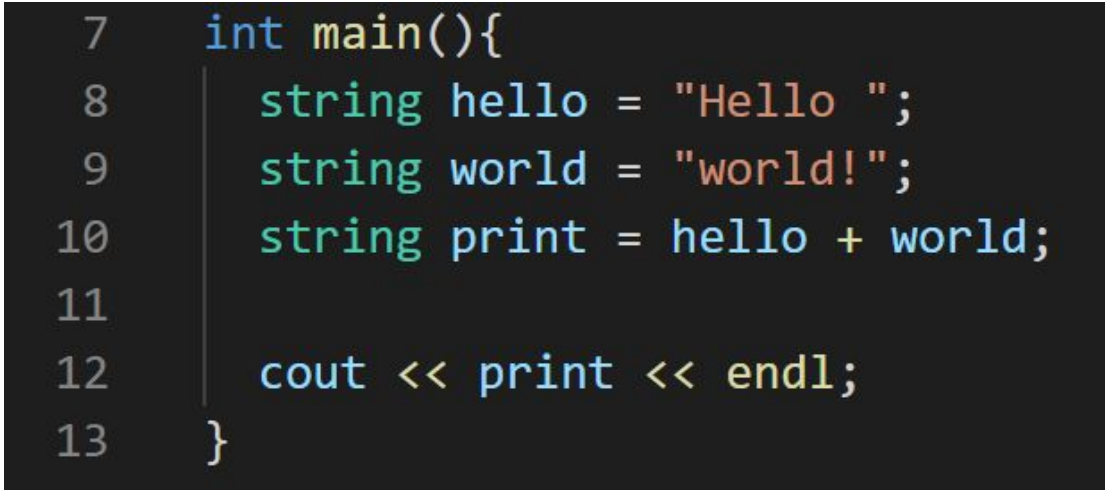
```
String str = ("gfg");
String str = "" g;
String str ; str = "gfg";
```

To Initialize



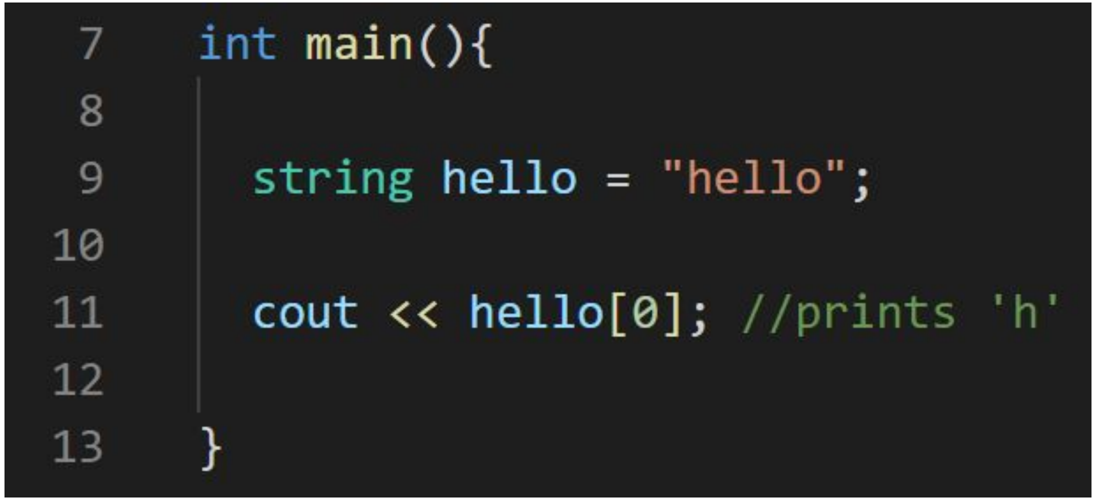
```
int main(){

  string exampleString = "this is an example";
  string emptyString; // equal to ""
  string exampleString2("this is another example");

}
```

To Concatenate



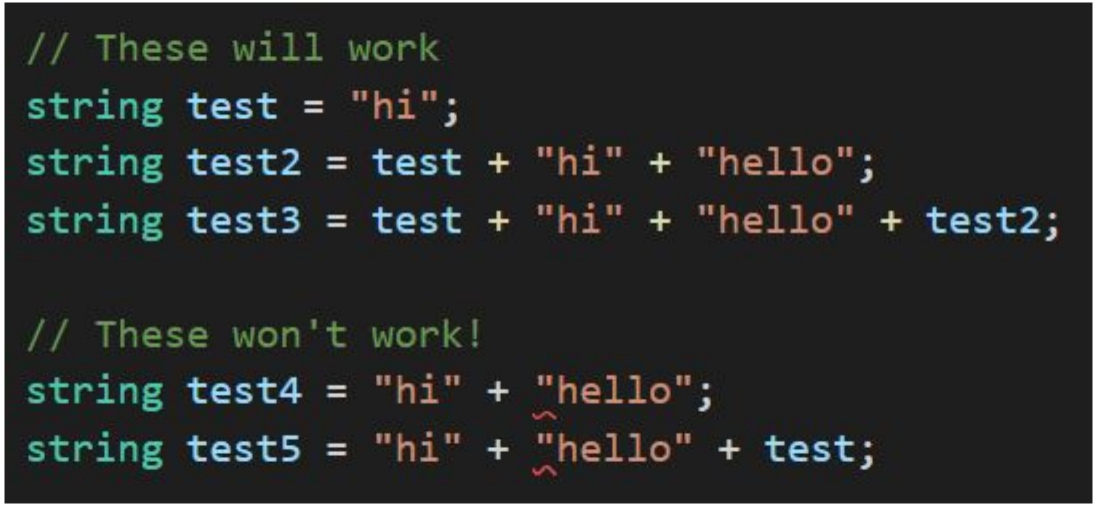
```
int main(){
    string hello = "Hello ";
    string world = "world!";
    string print = hello + world;

    cout << print << endl;
}
```

To Access Characters



```
int main(){

  string hello = "hello";

  cout << hello[0]; //prints 'h'

}
```

A note on concatenation...



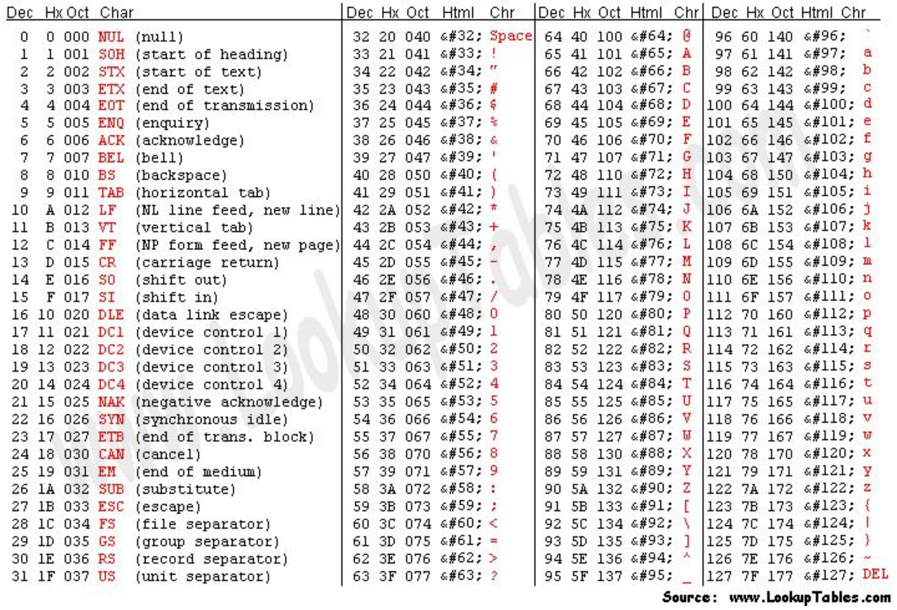
```
// These will work
string test = "hi";
string test2 = test + "hi" + "hello";
string test3 = test + "hi" + "hello" + test2;

// These won't work!
string test4 = "hi" + "hello";
string test5 = "hi" + "hello" + test;
```

Strings

Strings can be compared using <, >, ==, etc

How?

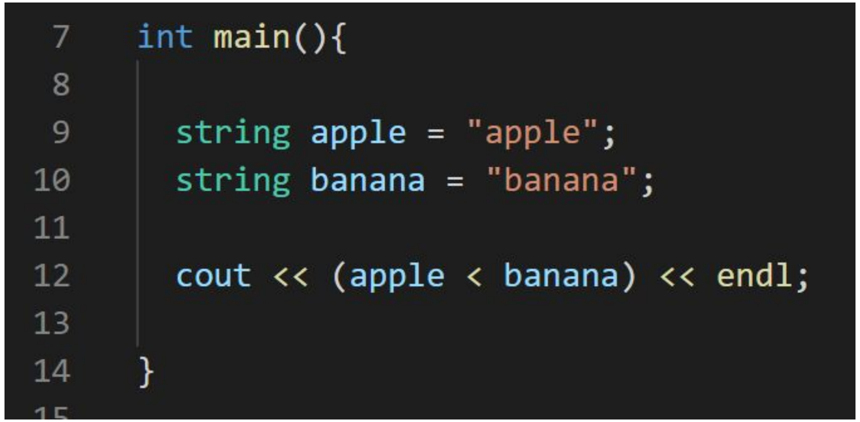
By comparing each character



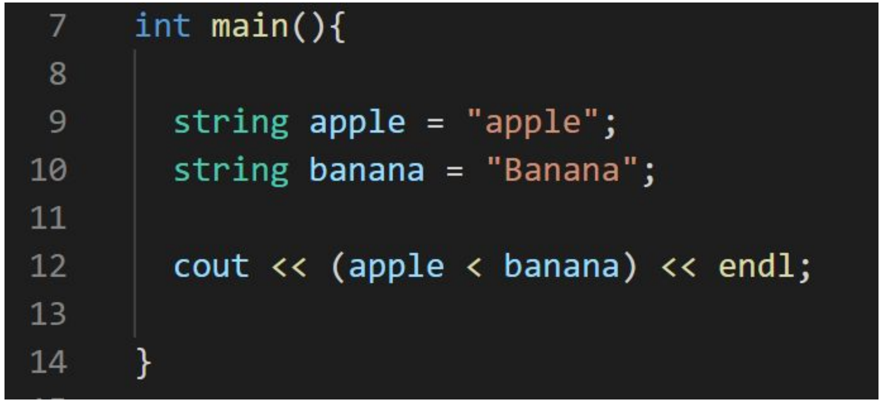
Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

What would these output?

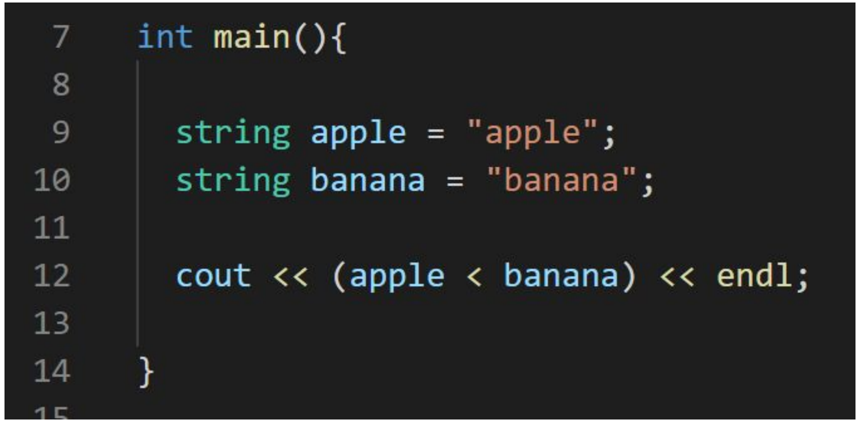


```
int main(){

  string apple = "apple";
  string banana = "banana";

  cout << (apple < banana) << endl;

}
```

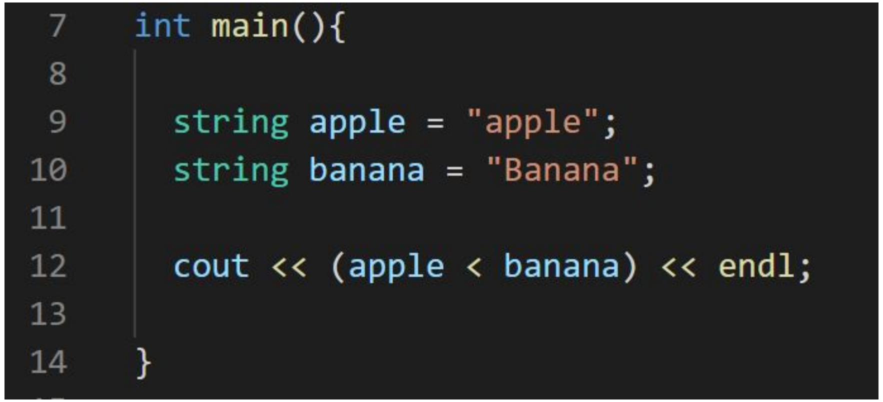


```
int main(){

  string apple = "apple";
  string banana = "Banana";

  cout << (apple < banana) << endl;

}
```

What would these output?



```
int main(){

  string apple = "apple";
  string banana = "banana";

  cout << (apple < banana) << endl;

}
```



```
int main(){

  string apple = "apple";
  string banana = "Banana";

  cout << (apple < banana) << endl;

}
```

1 (true)

0 (false)

Other string Functions

int string::find(string looking) - returns index of looking in the string (optional parameter for indexstart)

string string::substr(int start, int end)- returns a substring from start to end positions

Use Google!

argc/argv

argc/argv

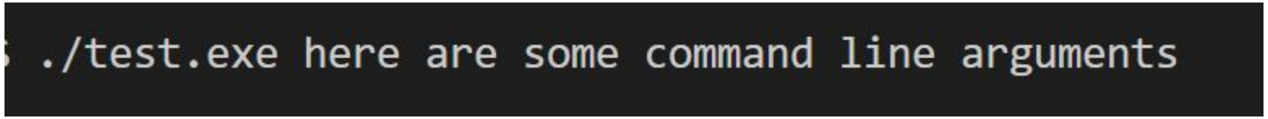
- Command line arguments
- argc is an int
- argv is an array of character pointers

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

or

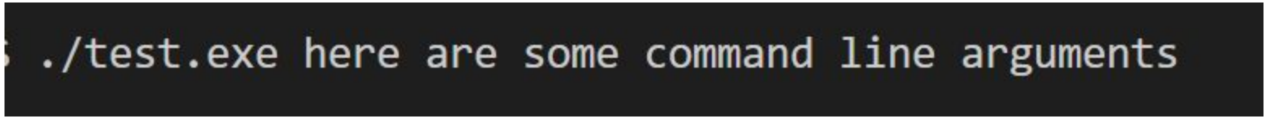
```
int main(int argc, char **argv) { /* ... */ }
```

argc/argv example



```
 ./test.exe here are some command line arguments
```

argc/argv example

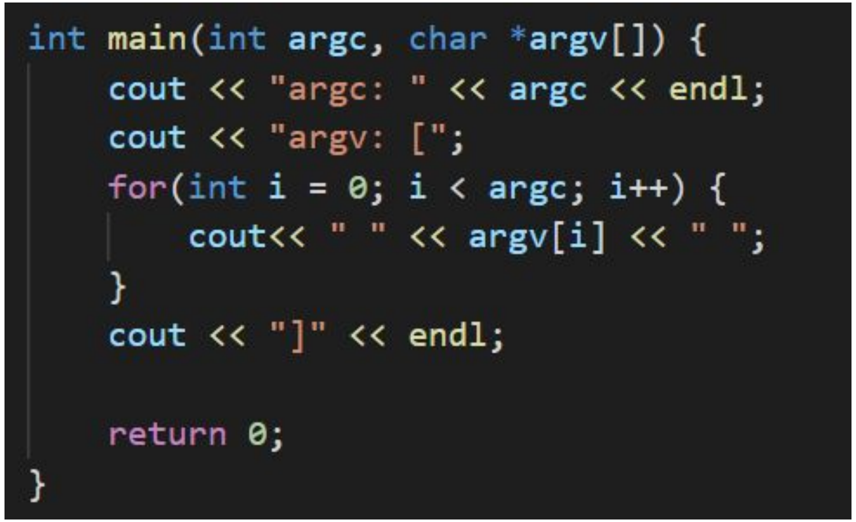


```
./test.exe here are some command line arguments
```

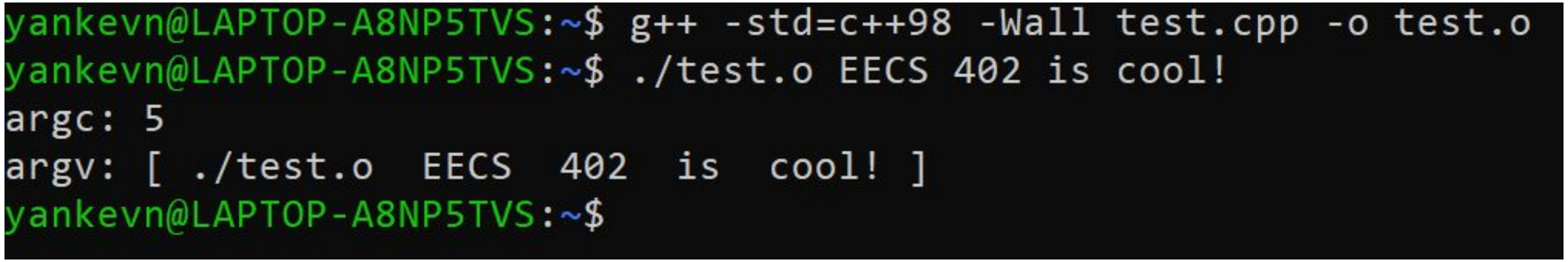
argc = 7

argv is an array containing [“./test.exe”, “here”, “are”, “some”, “command”, “line”, “arguments”]

argc/argv example

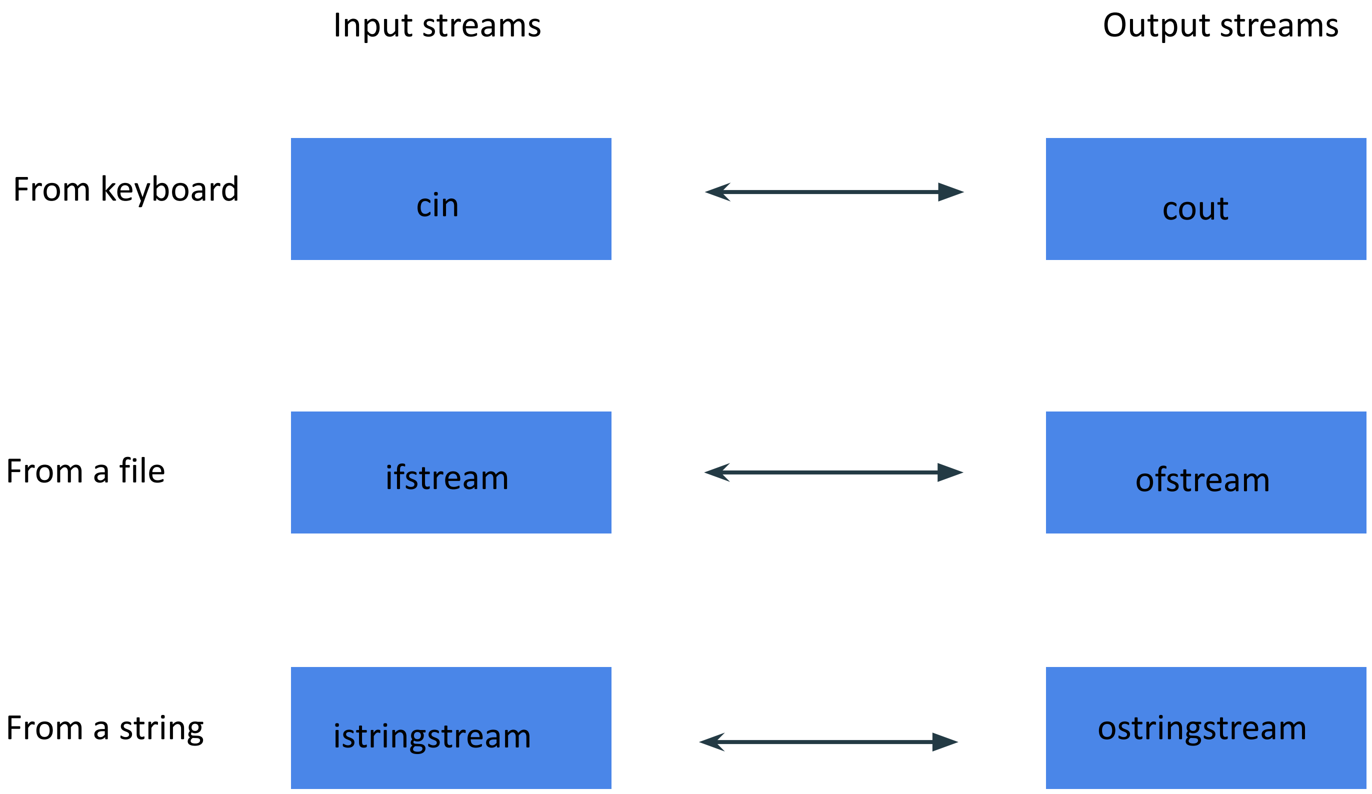


```
int main(int argc, char *argv[]) {
    cout << "argc: " << argc << endl;
    cout << "argv: [";
    for(int i = 0; i < argc; i++) {
        cout<< " " << argv[i] << " ";
    }
    cout << "]" << endl;

    return 0;
}
```



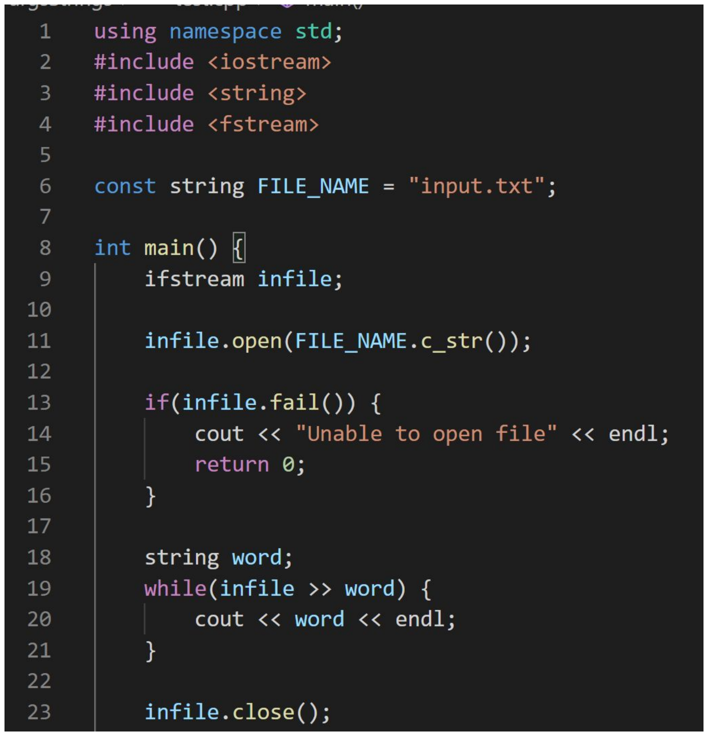
```
yankevn@LAPTOP-A8NP5TVS:~$ g++ -std=c++98 -Wall test.cpp -o test.o
yankevn@LAPTOP-A8NP5TVS:~$ ./test.o EECS 402 is cool!
argc: 5
argv: [ ./test.o  EECS  402  is  cool! ]
yankevn@LAPTOP-A8NP5TVS:~$
```

streams


Input streams
Output streams
From keyboard
cin
cout
From a file
ifstream
ofstream
From a string
istringstream
ostringstream

ifstream

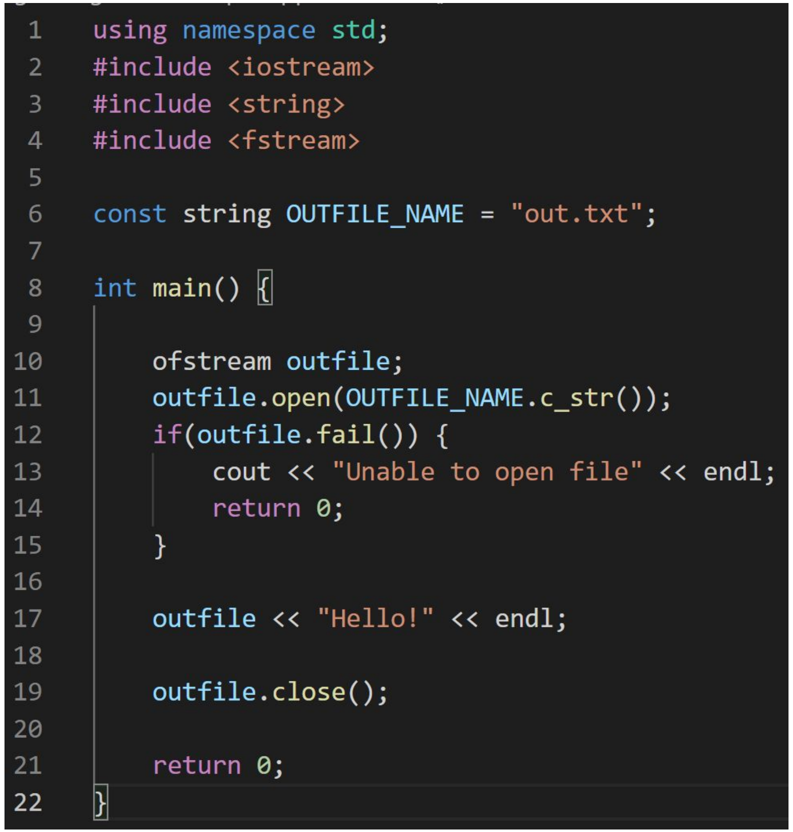
- Allows the programmer to read the contents of a file
- Must pass in c string to open parameter
- Always error check after trying to open!
- Close the file after



```cpp
using namespace std;
#include <iostream>
#include <string>
#include <fstream>

const string FILE_NAME = "input.txt";

int main() {
    ifstream infile;

    infile.open(FILE_NAME.c_str());

    if(infile.fail()) {
        cout << "Unable to open file" << endl;
        return 0;
    }

    string word;
    while(infile >> word) {
        cout << word << endl;
    }

    infile.close();
```

ofstream

- Allows the programmer to write output to a file
- Less need for error checking
- Does **not** append, writes over existing data



```cpp
using namespace std;
#include <iostream>
#include <string>
#include <fstream>

const string OUTFILE_NAME = "out.txt";

int main() {

    ofstream outfile;
    outfile.open(OUTFILE_NAME.c_str());
    if(outfile.fail()) {
        cout << "Unable to open file" << endl;
        return 0;
    }

    outfile << "Hello!" << endl;

    outfile.close();

    return 0;
}
```

Error Checking

.good() returns true if stream is in good state

.fail() returns true if stream is in fail state

.eof() returns true if the end of the file is reached

.clear() returns stream to good state/clears it

.ignore(num, endChar) “consume” up to num characters from the stream, up to, and including, the character indicated by endChar (usually ‘\n’)

Example

Return the average value of integers found in a file

The filename is entered through the command line arguments

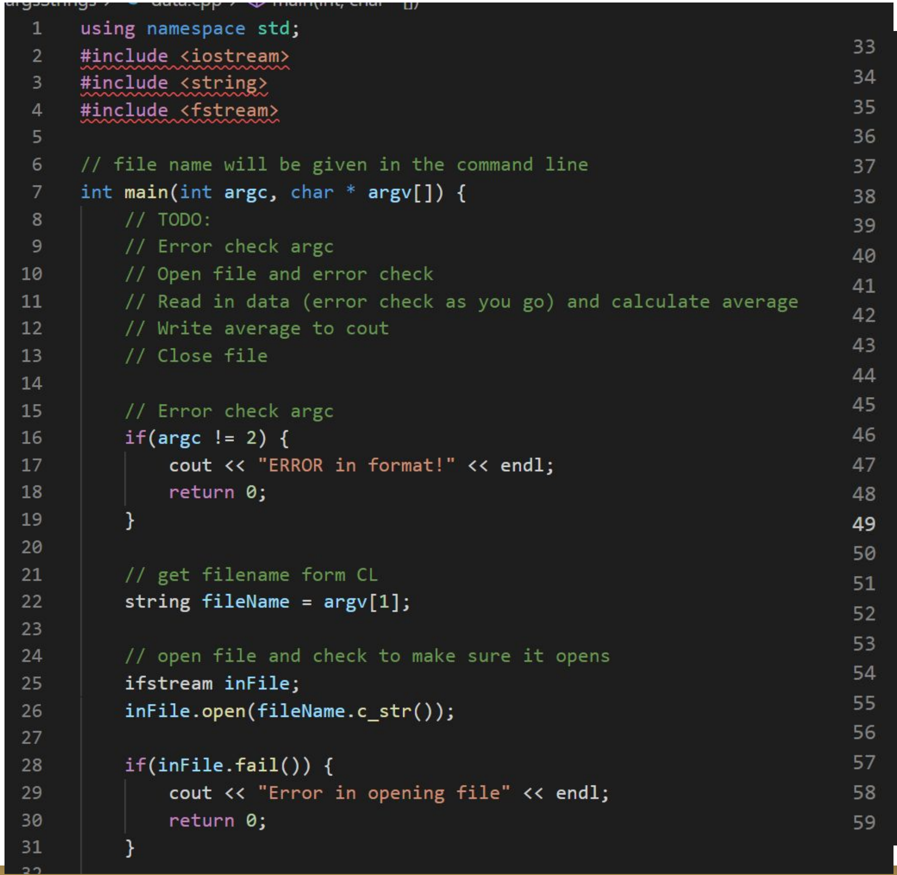
If there are any errors in the file (a non integer entry) stop running the function

$ git clone https://github.com/emolson16/streams_practice

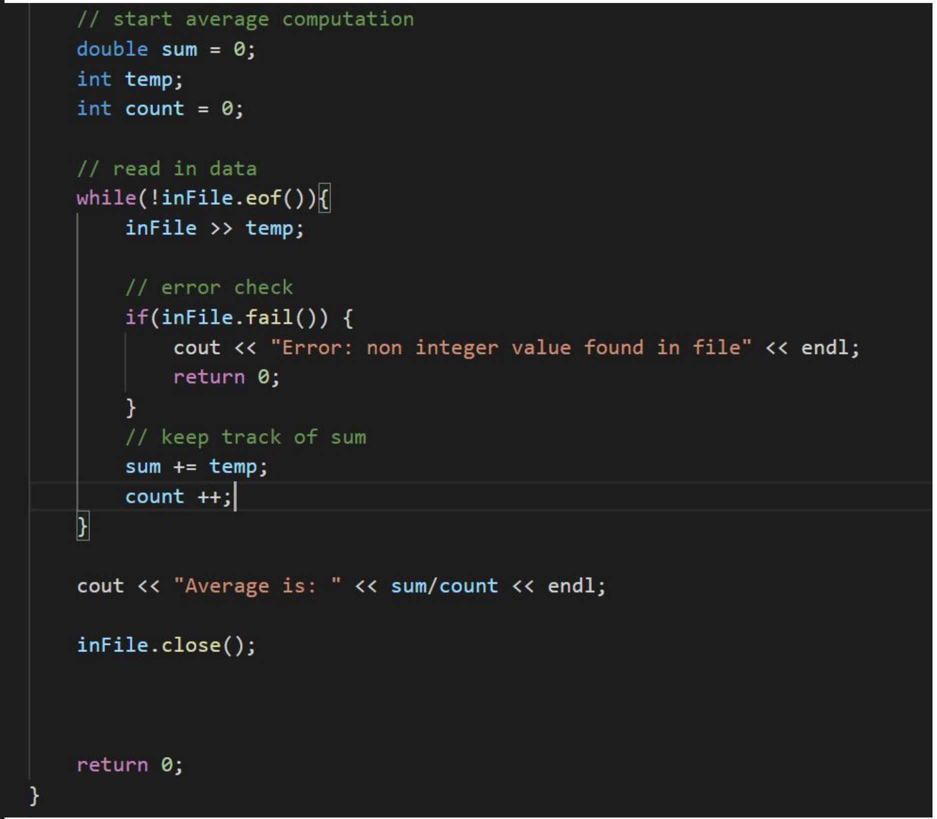
Cpp: https://drive.google.com/file/d/1HrGsj3k6NCT_t2pf_rcIFQrpFZl2L6fx/view?usp=sharing

Txt: https://drive.google.com/file/d/1_JP0iQ3QDqCwhwqnGMtRAU3FTjfXphbZ/view?usp=sharing

Example Solution



```cpp
using namespace std;
#include <iostream>
#include <string>
#include <fstream>

// file name will be given in the command line
int main(int argc, char * argv[]) {
    // TODO:
    // Error check argc
    // Open file and error check
    // Read in data (error check as you go) and calculate average
    // Write average to cout
    // Close file

    // Error check argc
    if(argc != 2) {
        cout << "ERROR in format!" << endl;
        return 0;
    }

    // get filename form CL
    string fileName = argv[1];

    // open file and check to make sure it opens
    ifstream inFile;
    inFile.open(fileName.c_str());

    if(inFile.fail()) {
        cout << "Error in opening file" << endl;
        return 0;
    }
```



```cpp
    // start average computation
    double sum = 0;
    int temp;
    int count = 0;

    // read in data
    while(!inFile.eof()){
        inFile >> temp;

        // error check
        if(inFile.fail()) {
            cout << "Error: non integer value found in file" << endl;
            return 0;
        }
        // keep track of sum
        sum += temp;
        count ++;
    }

    cout << "Average is: " << sum/count << endl;

    inFile.close();



    return 0;
}
```