UCLA Samueli Computer Science

CS31 Week 6 Discussion

Fall 2021, Section 1C

Mingyu Derek Ma mdma@ucla.edu

Thanks Muhao Chen, Alexis Korb, Rosa Garza for their shared content

https://derek.ma/cs31 for slides and other discussion materials

Reminder

- Project 5, Monday Nov 15, 11pm

Project 3 Feedback

- Comment your program logic, especially for complicated function like obeyPlan()
- Need to provide concrete test cases, rather than high-level design thoughts about test cases
- Need to have test cases for all functions, rather than just obeyPlan()
- Need to have brief reason for your test cases
- Use pseudocode to describe your program design, instead of paragraphs

Project 5 Suggestions

- Variable-length array is not allowed
  - g++ extension of variable-length arrays won't compile under g31
- All arrays must have bounds known at the compile time

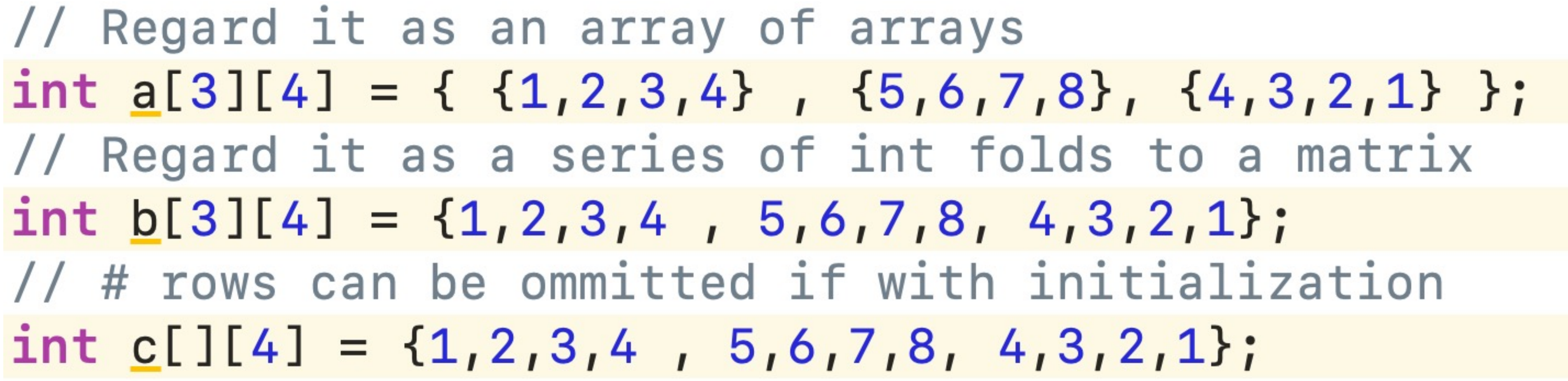
Multi-dimensional array

- An array of arrays
  - Two-dimension represents a matrix (2-d tensor)
  - Three-dimension represents a cube (3-d tensor)
- All elements in a multi-dimensional array have to be the same type

Declare a 2-d array without initialization

- int x[3][4];
    - 3 rows, 4 columns matrix
    - 3 arrays with length 4
- type name [#rows][#cols]
    - Both #rows and #cols need to specified in declaration if without initialization
    - Similar to 1-d array declaration, where we also have to specify number of elements when initialize a new array

Initialize a 2-d array



```c
// Regard it as an array of arrays
int a[3][4] = { {1,2,3,4} , {5,6,7,8}, {4,3,2,1} };
// Regard it as a series of int folds to a matrix
int b[3][4] = {1,2,3,4 , 5,6,7,8, 4,3,2,1};
// # rows can be ommitted if with initialization
int c[][4] = {1,2,3,4 , 5,6,7,8, 4,3,2,1};
```

1	2	3	4
5	6	7	8
4	3	2	1

Initialize a 2-d array

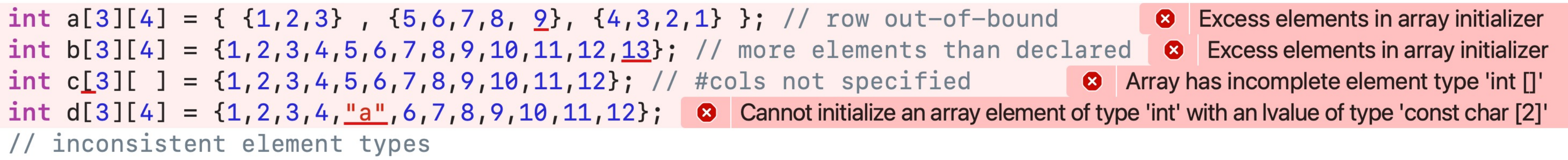
- What about we want less elements for a certain row
- int xy[3][4] = { {1,2,3,4} , {5,6}, {4,3,2,1} };
  - Missing elements in such rows will be all-zero

1	2	3	4
5	6	**0**	**0**
4	3	2	1

- int xy[3][4] = { 1, 2, 3, 6, 7, 8, 4, 3, 2 };
  - Elements in the end will be all-zero

1	2	3	6
7	8	4	3
2	**0**	**0**	**0**

Initialize a 2-d array: Unacceptable ways



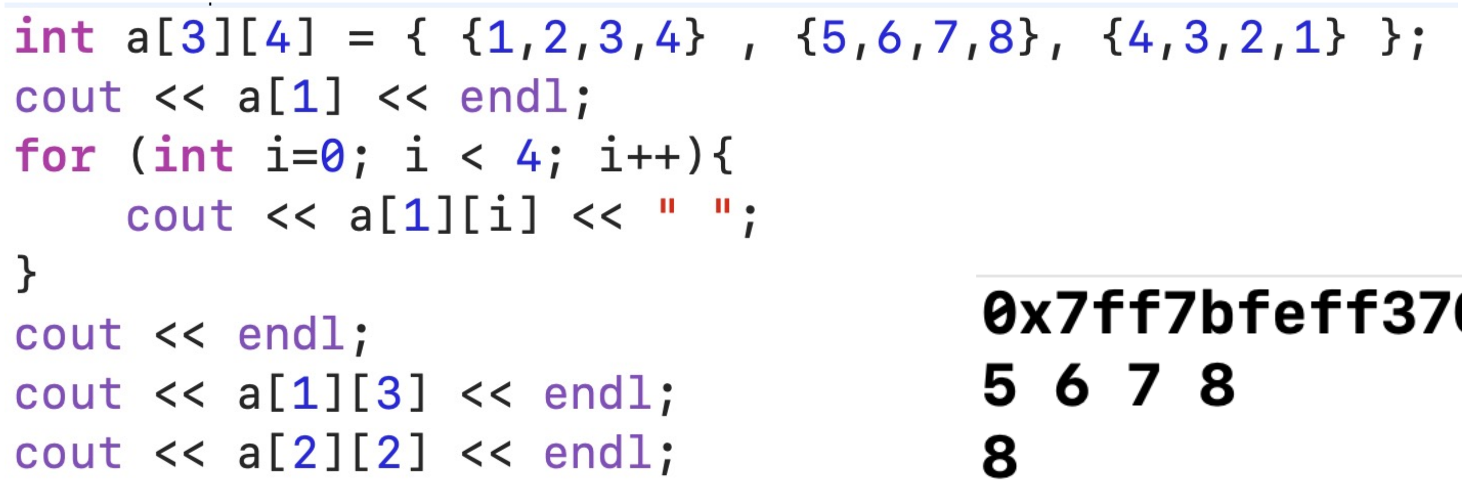
```
int a[3][4] = { {1,2,3} , {5,6,7,8, 9}, {4,3,2,1} }; // row out-of-bound        Excess elements in array initializer
int b[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13}; // more elements than declared     Excess elements in array initializer
int c[3][ ] = {1,2,3,4,5,6,7,8,9,10,11,12}; // #cols not specified        Array has incomplete element type 'int []'
int d[3][4] = {1,2,3,4,"a",6,7,8,9,10,11,12};     Cannot initialize an array element of type 'int' with an lvalue of type 'const char [2]'
// inconsistent element types
```

Access elements in a 2-d array

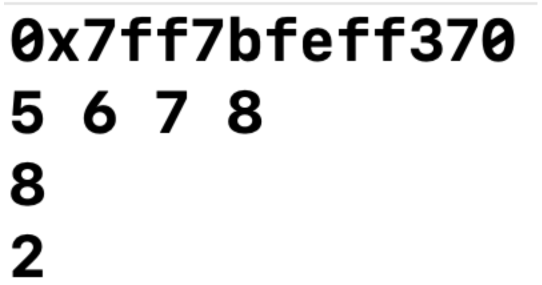
- Access an element
  - a[1][2] takes you to the second row third column -> 7
- Access a row
  - a[1] gives you the start address of the second row -> {5, 6, 7, 8}
- Access a column
  - There is no direct way to access a column

1	2	3	4
5	6	**7**	8
4	3	2	1

Example



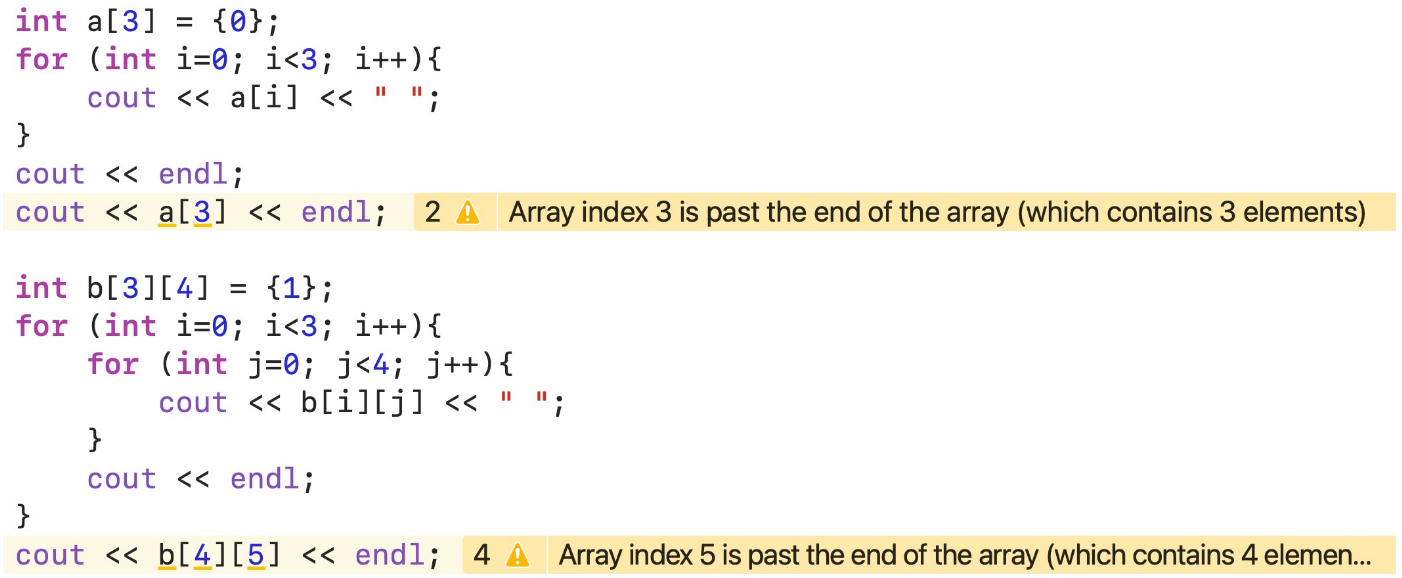
```
int a[3][4] = { {1,2,3,4} , {5,6,7,8}, {4,3,2,1} };
cout << a[1] << endl;
for (int i=0; i < 4; i++){
    cout << a[1][i] << " ";
}
cout << endl;
cout << a[1][3] << endl;
cout << a[2][2] << endl;
```



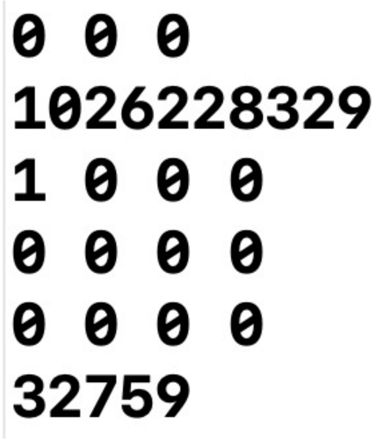
```
0x7ff7bfeff370
5 6 7 8
8
2
```

2-d array will not check the bound

- The program will run without error if we access out of bound values
- We need to remember the boundaries of 1-d arrays ourselves



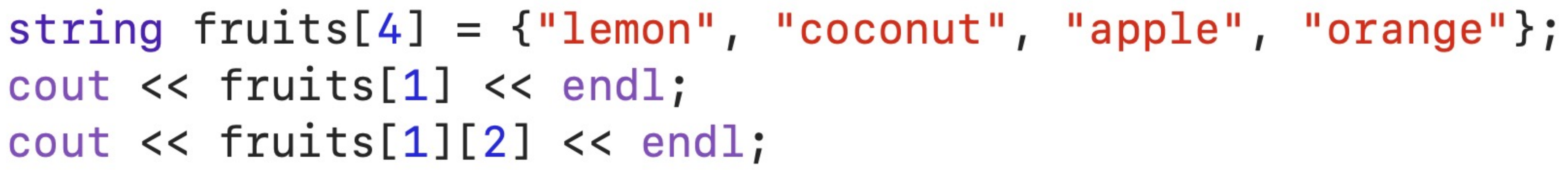
```
int a[3] = {0};
for (int i=0; i<3; i++){
    cout << a[i] << " ";
}
cout << endl;
cout << a[3] << endl;    2 ⚠ Array index 3 is past the end of the array (which contains 3 elements)

int b[3][4] = {1};
for (int i=0; i<3; i++){
    for (int j=0; j<4; j++){
        cout << b[i][j] << " ";
    }
    cout << endl;
}
cout << b[4][5] << endl;    4 ⚠ Array index 5 is past the end of the array (which contains 4 elemen...
```



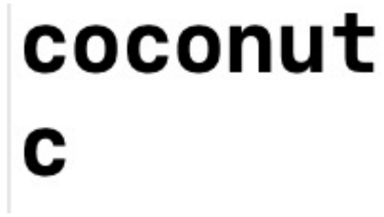
```
0 0 0
1026228329
1 0 0 0
0 0 0 0
0 0 0 0
32759
```

String array

- Array of strings is similar to a two dimensional character array



```cpp
string fruits[4] = {"lemon", "coconut", "apple", "orange"};
cout << fruits[1] << endl;
cout << fruits[1][2] << endl;
```



```
coconut
c
```

Pass multidimensional arrays to functions

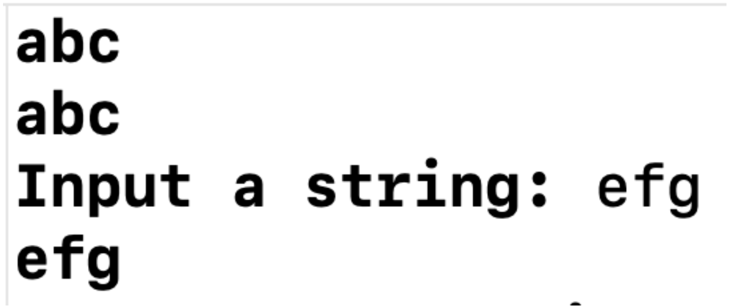
- Need to specify the size of all dimensions except for the first
- Must pass the size of the first dimension as a separate parameter

```
void functionOne(int a[][5][10], int l) {
     …
}

int main(){
     int b[2][5][10];
     functionOne(b, 2)
}
```

C-String

- String in C language
- We can initialize it with a string value
- It uses a null byte (‘\0’) to denotes its end
- Benefit: performance, faster and uses less memory

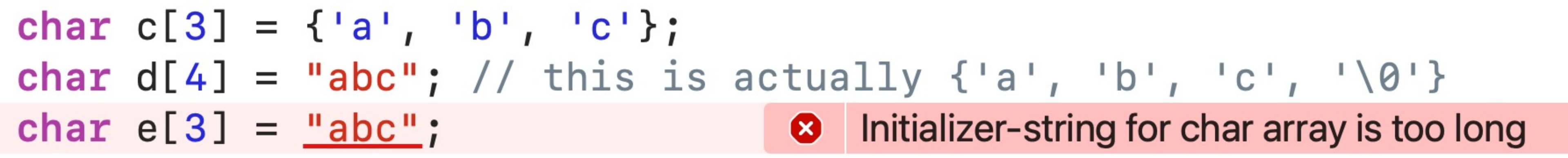
```cpp
char c[10] = {'a', 'b', 'c'};
cout << c << endl;
char d[10] = "abc";
cout << d << endl;
char e[10];
cout << "Input a string: ";
cin >> e;
cout << e << endl;
```



```
abc
abc
Input a string: efg
efg
```

Initialize a c-string

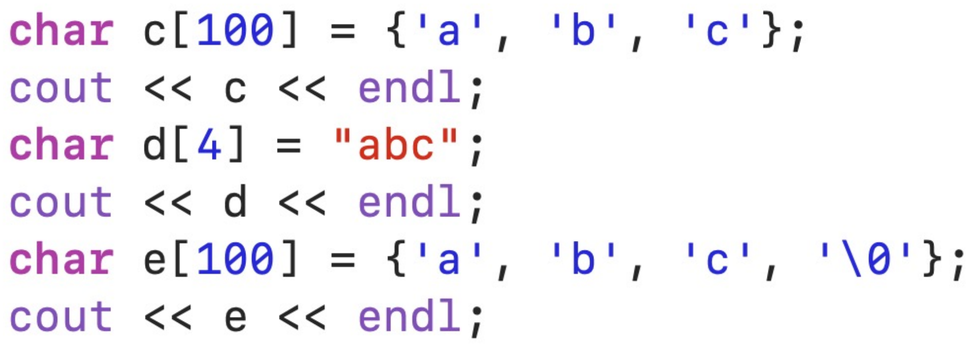
- With a string c[n], we can initialize it with a string value with the maximum length of n-1
- You can also initialize it with a set of char ended with a ‘\0’
- {'a', 'b', ‘c’} is not “abc”



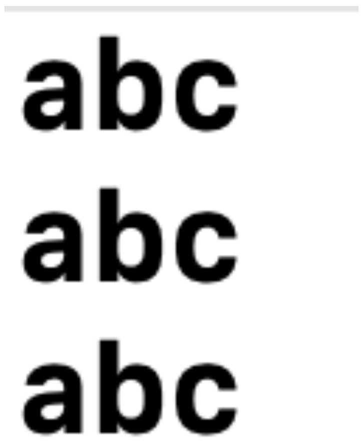
```
char c[3] = {'a', 'b', 'c'};
char d[4] = "abc"; // this is actually {'a', 'b', 'c', '\0'}
char e[3] = "abc";                    Initializer-string for char array is too long
```

cout a c-string

- Output characters until reaching a ‘\0’



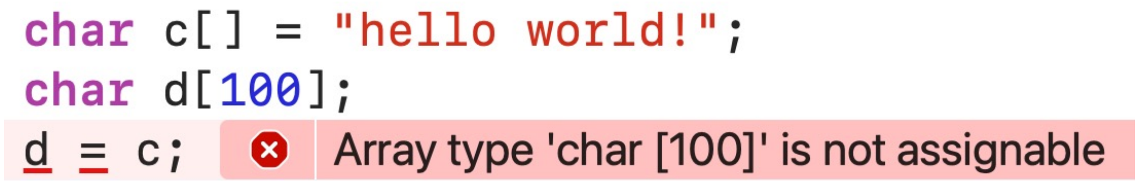
```cpp
char c[100] = {'a', 'b', 'c'};
cout << c << endl;
char d[4] = "abc";
cout << d << endl;
char e[100] = {'a', 'b', 'c', '\0'};
cout << e << endl;
```



```
abc
abc
abc
```

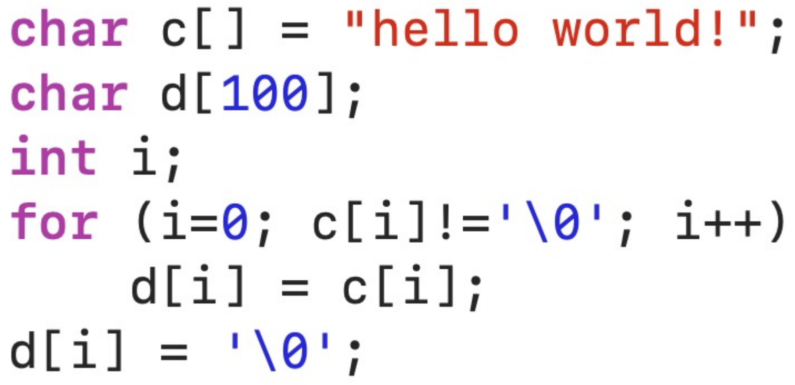
Copy a c-string

- We need to copy element by element
- Deep copy



```c
char c[] = "hello world!";
char d[100];
d = c;
```

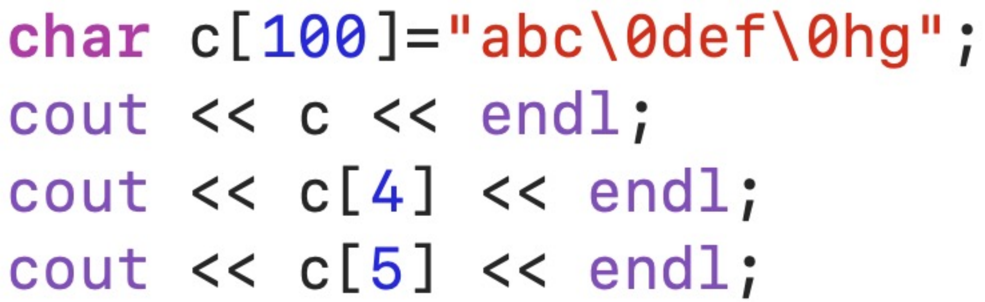
Array type 'char [100]' is not assignable



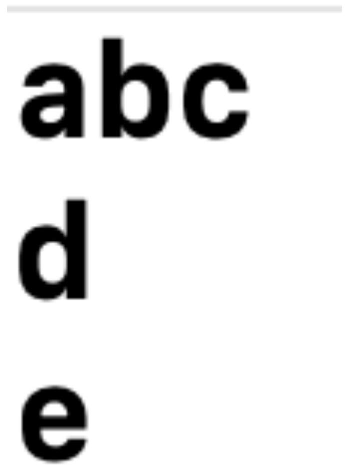
```c
char c[] = "hello world!";
char d[100];
int i;
for (i=0; c[i]!='\0'; i++)
    d[i] = c[i];
d[i] = '\0';
```

What if there are multiple null bytes

- The first ’\0’ always represents the end
- But characters after the first ‘\0’ is still saved, they will not show up when you print the c—tring out



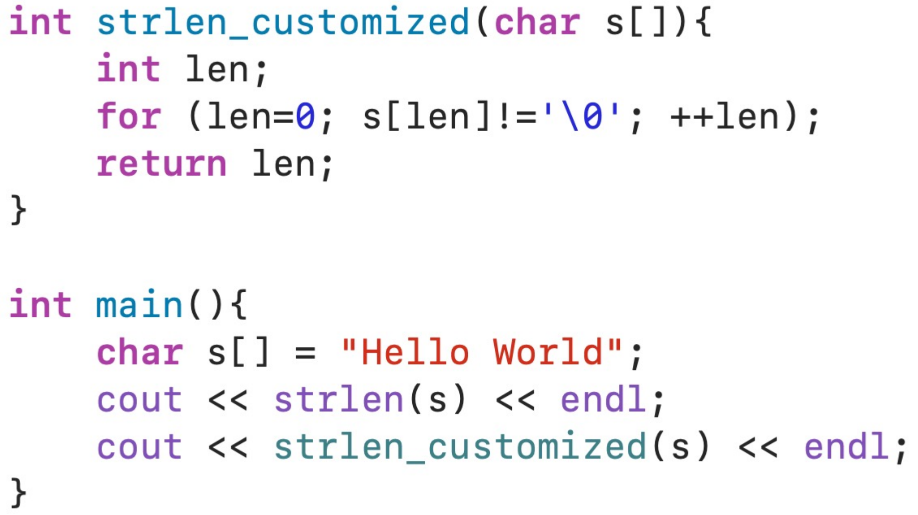
```cpp
char c[100]="abc\0def\0hg";
cout << c << endl;
cout << c[4] << endl;
cout << c[5] << endl;
```



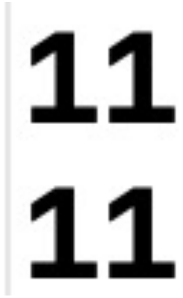
```
abc
d
e
```

Library functions for C-string

- include <cstring>
  - Includes the library functions for C-strings
- strlen(s)
  - Returns the length of s



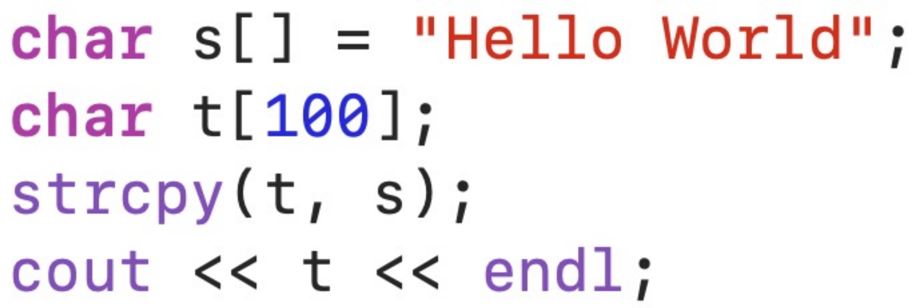
```
int strlen_customized(char s[]){
    int len;
    for (len=0; s[len]!='\0'; ++len);
    return len;
}

int main(){
    char s[] = "Hello World";
    cout << strlen(s) << endl;
    cout << strlen_customized(s) << endl;
}
```



```
11
11
```

strcpy(t, s)

- Copy the c-string s to c-string t, deep copy
- Need to make sure the declared space for t is enough to take elements from s



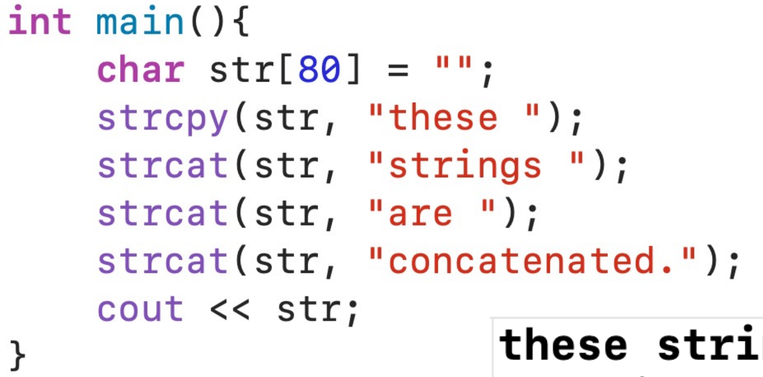
```
char s[] = "Hello World";
char t[100];
strcpy(t, s);
cout << t << endl;
```

strncpy(t, s, n)

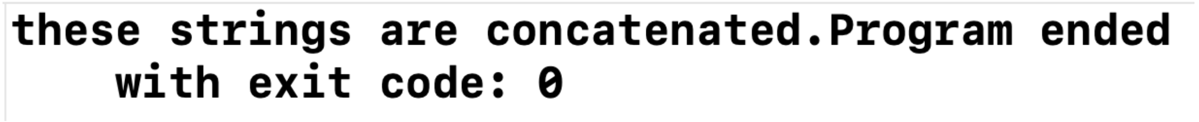
- Copy at most n characters from s to t
- Note: if length of s > n, then ’\0’ is not copied to t
    - We cannot assume t as a complete C-string
    - We have to manually assign t[n] = ‘\0’;

strcat(t, s)

- Append C-string s to the end of t
- The returned value will be t, variable t's value will be changed to the appended string
- Need to make sure t has enough space for elements in both s and t



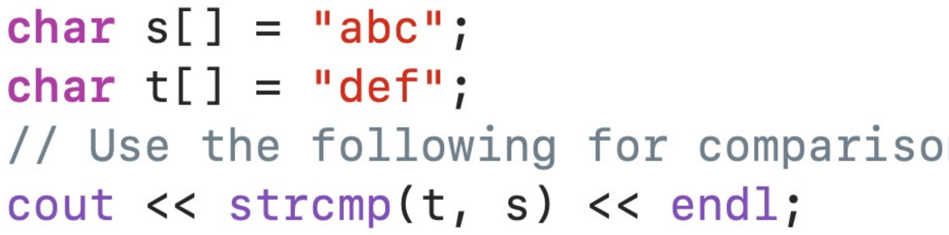
```
int main(){
    char str[80] = "";
    strcpy(str, "these ");
    strcat(str, "strings ");
    strcat(str, "are ");
    strcat(str, "concatenated.");
    cout << str;
}
```



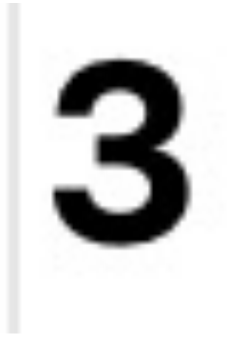
```
these strings are concatenated.Program ended
    with exit code: 0
```

int strcmp(char *t, char *s)

- Compare two c-strings
  - s==t; s < t; s > t; won't work
- Return value is int, not boolean
  - t equals to s: return 0
  - t less than s: return something < 0
  - t greater than s: return something > 0
- Use strcmp for if condition
  - if (strcmp(t, s) != 0)
  - if (strcmp(t, s) < 0)
  - if (strcmp(t, s) > 0)

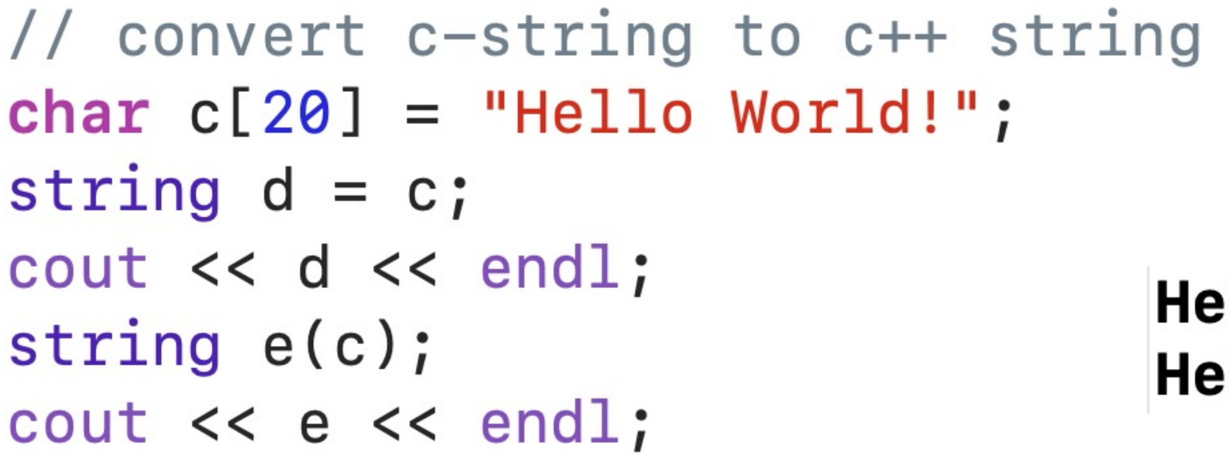


```
char s[] = "abc";
char t[] = "def";
// Use the following for compariso
cout << strcmp(t, s) << endl;
```

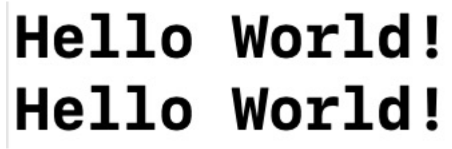


```
3
```

Convert a C-string to a C++ string

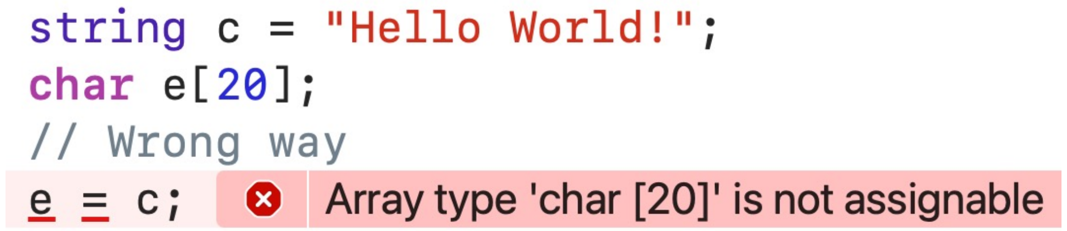


```cpp
// convert c-string to c++ string
char c[20] = "Hello World!";
string d = c;
cout << d << endl;
string e(c);
cout << e << endl;
```



```
Hello World!
Hello World!
```

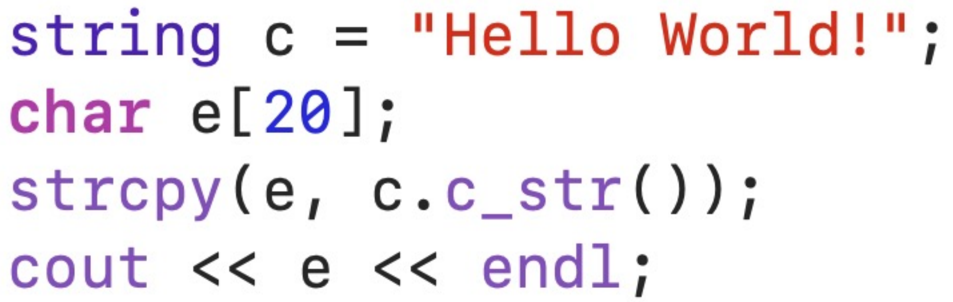
Convert a C++ string to a C-string



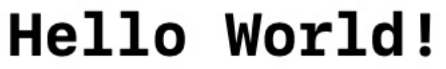
```
string c = "Hello World!";
char e[20];
// Wrong way
e = c;
```

```
Array type 'char [20]' is not assignable
```

- c_str()
- Get the "C-string body" of a C++ string



```
string c = "Hello World!";
char e[20];
strcpy(e, c.c_str());
cout << e << endl;
```



```
Hello World!
```

Array of C-strings

- A C-string is an array of characters. An array of C-strings is 2D array
- char s[10][20];
    - We can store up to 10 C-strings, each can be at most 19 characters long

Array of C-strings

```
char s[3][6];
strcpy(s[0], "hello");
strcpy(s[1], "world");
strcpy(s[2], "!");

cout << s << endl;
cout << s[0] << endl;
cout << s[1][2] << endl;
```

```
0x7ff7bfeff380
hello
r
```

- We cannot directly cout an array of C-strings
- But we can cout a single C-string
- We can also cout a character in a C-string

Functionality	C++ strings	C strings	Notes
Necessary libraries	#include <string>	None needed	
Useful libraries	#include <cctype>	#include <cstring>	<cstring> needed for strcpy, strlen, strcat, strcmp
Declare a string	string s = “Hello”; string t = “Hey”;	char s[6] = “Hello”; char t[10] = “Hey”;	For C strings, the declared size of the character array must be at least as big as the number of characters in the string including the zero byte.

Assigning a new value	s = “Hi”;
	s = t;
		strcpy(s, “Hi”);
		strcpy(s, t);
			For C strings, if s is not big enough to hold the string that is being copied into it, you get undefined behavior.

Getting length of a string	s.length();
	s.size();
		strlen(s);	For C strings, the zero byte is not included in the length output by strlen

Appending to a string	s += “bye”;
	s += t;
		strcat(s, “bye”);
		strcat(s, t);
			For C strings, if s is not big enough to hold its new value, you get undefined behavior.

Functionality	C++ strings	C strings	Notes

Getting a string as input	string s;
	cin.getline(s, 10000);
		char s[10];
		cin.getline(s, 10);
			For C strings, the second parameter should be no larger than the length of the character array for s.

Printing out a string	cout << s;	cout << s;	
Getting the ith character of a string	char c = s[i];	char c = s[i];	
Assigning to the ith character of a string	s[i] = ‘a’;	s[i] = ‘a’;	For C strings, make sure not to overwrite the zero byte. You can, however, move the zero byte.

Comparing two strings	if (s < t)
	if (s > t)
	if (s == t)
	if (s != t)
		if(strcmp(s, t) < 0)
		if(strcmp(s, t) > 0)
		if(strcmp(s, t) == 0)
		if (strcmp(s, t) != 0)	

Functionality	C++ strings	C strings
Iterating through a string	`for(int k = 0; k != s.size(); k++) { … }`	`for(int k = 0; s[k] != ‘\0’; k++) { … }`
Passing a string to a function	`void f(string s) { … }` `int main() { { string t = “Hello”; f(t); }`	`void f(char s[]) { … }` `int main() { { char t[6]= “Hello”; f(t); }`

Functionality	C++ strings	C strings

Array of strings	string a[3] = {“Hello”, “Hi”, “Hey”};	char a[3][6] = {“Hello”, “Hi”, “Hey”};
		// The last dimension must be big
		// enough to hold all strings in the
		// array.

Getting the i^{th} element of an array	string s = a[i];	char s[6];
		strcpy(s, a[i]);
		// The size of the new C string must be
		// big enough to hold the element

Passing an array of strings to a function	void f(string a[], int n) { … }
	
	int main() {
	{
	    string a[3] = {“Hello”, “Hi”, “Hey”};
	    f(a, 3);
	}
		void f(string a[][6], int n) { … }
		
		int main() {
		{
		    char a[3][6] = {“Hello”, “Hi”, “Hey”};
		    f(a, 3);
		}

Thank You