CS 31 Week 6 Discussion 2E

Srinath

Announcements

- Project 5 is up! Has 2 parts. **Both due 11:00 PM Monday, November 14.**

Outline

- 2D Arrays
- C Strings
- Project 5
- Worksheet 6

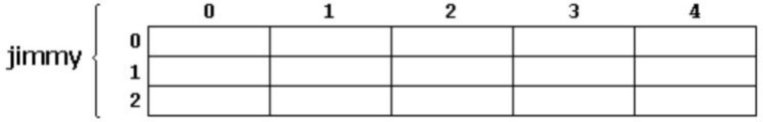
2D Arrays

2D Arrays :

We can also declare and use 2-dimensional arrays in C++.

Think of them as “array of arrays”

int jimmy [3][5];

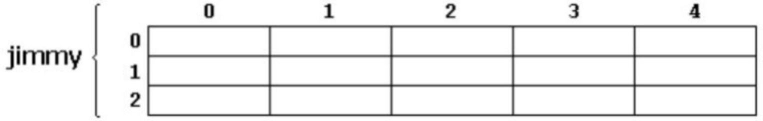


2D Arrays :

We can also declare and use 2-dimensional arrays in C++.

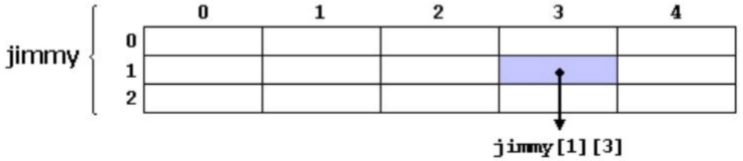
Think of them as "array of arrays"

int jimmy [3][5];



Accessing element at position 1, 3

int val = jimmy[1][3];
jimmy [1][3] = 10;

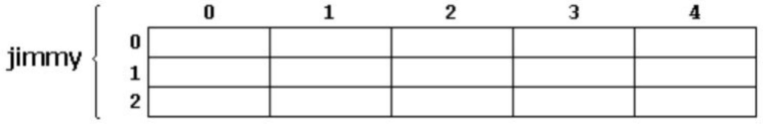


2D Arrays :

We can also declare and use 2-dimensional arrays in C++.

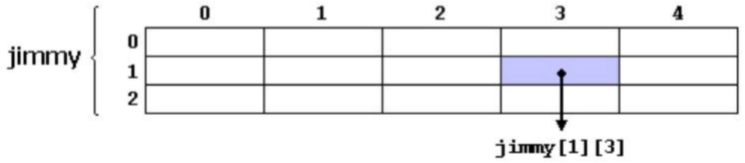
Think of them as "array of arrays"

int jimmy [3][5];



Accessing element at position 1, 3

int val = jimmy[1][3];
jimmy [1][3] = 10;



You can also declare arrays with more dimensions - called "multidimensional" arrays.

int points [10][9][15];

2D Arrays : as Function parameters

```
void processArr(int a[][2], int n_rows, int n_cols) {
            cout << "element at index 1,1 is " << a[1][1];
}

int main() {
   int arr[2][2];
   arr[0][0] = 0;
   arr[0][1] = 1;
   arr[1][0] = 2;
   arr[1][1] = 3;

   processArr(arr, 2, 2);
   return 0;
}
```

leave off the first bound, but you must supply the second bound as a compile-time constant expression

2D Arrays : as Function parameters

```
void processArr(int a[][2], int n_rows, int n_cols) {
            cout << "element at index 1,1 is " << a[1][1];
}

int main() {
   int arr[2][2];
   arr[0][0] = 0;
   arr[0][1] = 1;
   arr[1][0] = 2;
   arr[1][1] = 3;

   processArr(arr, 2, 2);
   return 0;
}
```

leave off the first bound, but you must supply the second bound as a compile-time constant expression

For multidimensional arrays, leave off first, but must supply the rest of bounds.
Eg: **process(int c[][5][7]);**

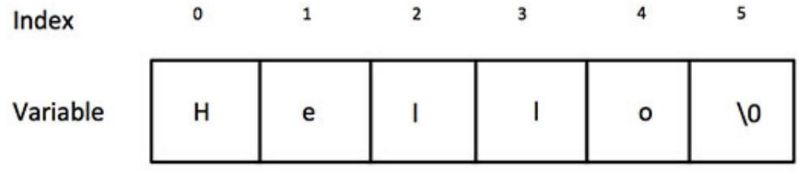
C Strings

C Strings :

In C programming, the collection of characters is stored in the form of arrays, this is also supported in C++ programming

C-strings are arrays of type char terminated with null character, that is, ‘\0’ (ASCII value of null character is 0).

Example: char greeting[] = "Hello";



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0

C Strings :

Declaration

char name[20];

C Strings :

Declaration

char name[20];

Is this a C String right now?

C Strings :

Declaration

char name[20]; *// - can hold C-String, but is not yet a valid C-String*

Is this a C String right now?

- No

C Strings :

Declaration

```
char name[20];    // - can hold C-String, but is not yet a valid C-String
```

Is this a C String right now?

- No

Initialization

```
char title[10] = {‘w’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\0’};      // standard array initialization

char course[13] = “computer”;                  // shortcut array initialization

char role[13] = “”;                            // empty string, length = 0
```

C Strings :

Declaration

char name[20]; *// - can hold C-String, but is not yet a valid C-String*

Is this a C String right now?
- No

Initialization

char title[10] = {'w', 'o', 'r', 'l', 'd', '\0'}; *// standard array initialization*

char course[13] = "computer"; *// shortcut array initialization*

char role[13] = ""; *// empty string, length = 0*

What will be the remaining values?

C Strings :

Declaration

```
char name[20];    // - can hold C-String, but is not yet a valid C-String
```

Is this a C String right now?
- No

Initialization

```
char title[10] = {'w', 'o', 'r', 'l', 'd', '\0'};     // standard array initialization

char course[13] = "computer";                          // shortcut array initialization

char role[13] = "";                                    // empty string, length = 0
```

What will be the remaining values?
- set to value of '\0'

```
char str1[ ] = {'c', 'h', 'e', 's', 's', '\0'};        // without size specification
char str[ ] = "hello";                                 // without size specification
```

Is this valid?

C Strings :

Declaration

char name[20]; *// - can hold C-String, but is not yet a valid C-String*

Is this a C String right now?

- No

Initialization

char title[10] = {‘w’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\0’}; *// standard array initialization*

char course[13] = “computer”; *// shortcut array initialization*

char role[13] = “”; *// empty string, length = 0*

What will be the remaining values?

- set to value of ‘\0’

char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’}; *// without size specification*
char str[ ] = “hello”; *// without size specification*

Is this valid?

- Yes

char title[10] = {‘\0’, ‘w’, ‘o’, ‘r’, ‘l’, ‘d’};

What is the length here?

C Strings :

Declaration

char name[20]; *// - can hold C-String, but is not yet a valid C-String*

Is this a C String right now?

- No

Initialization

char title[10] = {‘w’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\0’}; *// standard array initialization*

char course[13] = “computer”; *// shortcut array initialization*

char role[13] = “”; *// empty string, length = 0*

What will be the remaining values?

- set to value of ‘\0’

char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’}; *// without size specification*
char str[ ] = “hello”; *// without size specification*

Is this valid?

- Yes

char title[10] = {‘\0’, ‘w’, ‘o’, ‘r’, ‘l’, ‘d’};

What is the length here?

- 0
- equivalent to empty string

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

```
char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’};
cout << strlen(str1);                              // Output ?
```

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

```
char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’};
cout << strlen(str1);                              // Output ? - 5
```

Looping through

```
for(int i=0; i < (int) strlen(str1) ; i++) {
        cout << str1[i];
}
cout << endl;
```

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

```
char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’};
cout << strlen(str1);                          // Output ? - 5

Looping through
for(int i=0; i < (int) strlen(str1) ; i++) {
        cout << str1[i];
}
cout << endl;

char s2[ ] = {‘c’, ‘h’, ‘\0’, ‘s’, ‘s’};
cout << strlen(s2);                            // Output ?
```

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

```
char str1[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’, ‘\0’};
cout << strlen(str1);                                   // Output ? - 5

Looping through
for(int i=0; i < (int) strlen(str1) ; i++) {
        cout << str1[i];
}
cout << endl;

char s2[ ] = {‘c’, ‘h’, ‘\0’, ‘s’, ‘s’};
cout << strlen(s2);                                     // Output ? - 2

char s3[ ] = {‘c’, ‘h’, ‘e’, ‘s’, ‘s’};
cout << strlen(s3);                                     // Output ?
```

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

*Any thoughts on implementation of **strlen(..)** ?*

```
char str1[ ] = {'c', 'h', 'e', 's', 's', '\0'};
cout << strlen(str1);                         // Output ? - 5
```

Looping through

```
for(int i=0; i < (int) strlen(str1) ; i++) {
        cout << str1[i];
}
cout << endl;

char s2[ ] = {'c', 'h', '\0', 's', 's'};
cout << strlen(s2);                           // Output ? - 2

char s3[ ] = {'c', 'h', 'e', 's', 's'};
cout << strlen(s3);                           // Output ? - undefined behaviour
```

C Strings : strlen

The C library header file **<cstring>** contains a number of utility functions that operate on C strings.

#include <cstring>

strlen(s): obtain the length of a C string

*Any thoughts on implementation of **strlen(..)** ?*

- *Loop until you find '\0'*

```
char str1[ ] = {'c', 'h', 'e', 's', 's', '\0'};
cout << strlen(str1);                               // Output ? - 5
```

Looping through

```
for(int i=0; i < (int) strlen(str1) ; i++) {
        cout << str1[i];
}
cout << endl;

char s2[ ] = {'c', 'h', '\0', 's', 's'};
cout << strlen(s2);                                 // Output ? - 2

char s3[ ] = {'c', 'h', 'e', 's', 's'};
cout << strlen(s3);                                 // Output ? - undefined behaviour
```

C Strings : strcmp

strcmp(s1, s2): compare two strings

Comparing C strings using the relational operators ==, !=, >, <, >=, and <= does not work correctly

Why ?

C Strings : strcmp

strcmp(s1, s2): compare two strings

Comparing C strings using the relational operators ==, !=, >, <, >=, and <= does not work correctly

Why ?

- *Because they are **char arrays.***

```
if( strcmp(s1, s2)  < 0 )   {....}              // s1 < s2
if( strcmp(s1, s2)  == 0 ) {....}               // s1 == s2
if( strcmp(s1, s2)  > 0 )   {....}              // s1 > s2
if( strcmp(s1, s2)  <= 0 ) {....}               // s1 <= s2
if( strcmp(s1, s2)  != 0 )  {....}              // s1 != s2
if( strcmp(s1, s2)  >= 0 ) {....}               // s1 >= s2
```

C Strings : strcmp

strcmp(s1, s2): compare two strings

Comparing C strings using the relational operators ==, !=, >, <, >=, and <= does not work correctly

Why ?

- *Because they are **char arrays.***

```
if( strcmp(s1, s2)  < 0 )   {....}          // s1 < s2
if( strcmp(s1, s2)  == 0 ) {....}           // s1 == s2
if( strcmp(s1, s2)  > 0 )   {....}          // s1 > s2
if( strcmp(s1, s2)  <= 0 ) {....}           // s1 <= s2
if( strcmp(s1, s2)  != 0 )  {....}          // s1 != s2
if( strcmp(s1, s2)  >= 0 ) {....}           // s1 >= s2
```

*Any thoughts on implementation of **strcmp(..)** ?*

C Strings : strcmp

strcmp(s1, s2): compare two strings

Comparing C strings using the relational operators ==, !=, >, <, >=, and <= does not work correctly

Why ?

- *Because they are **char arrays.***

```
if( strcmp(s1, s2)  < 0 )   {....}          // s1 < s2
if( strcmp(s1, s2)  == 0 ) {....}           // s1 == s2
if( strcmp(s1, s2)  > 0 )   {....}          // s1 > s2
if( strcmp(s1, s2)  <= 0 ) {....}           // s1 <= s2
if( strcmp(s1, s2)  != 0 )  {....}          // s1 != s2
if( strcmp(s1, s2)  >= 0 ) {....}           // s1 >= s2
```

*Any thoughts on implementation of **strcmp(..)** ?*

- *Loop through both arrays to check condition between corresponding elements.*

C Strings : strcpy

strcpy(s1, s2): copies string s2 into string s1.

A character array (including a C string) can not have a new value assigned to it after it is declared.

Why ?

```
char s1[20] = “This is a string”;
char s2[20];

s1 = “Another string”;              // error : invalid array assignment
s2 = s1;                            // error : invalid array assignment
```

C Strings : strcpy

strcpy(s1, s2): copies string s2 into string s1.

A character array (including a C string) can not have a new value assigned to it after it is declared.

Why ?
- *Because they are **arrays.***
- *Assigning arrays won't work.*

```
char s1[20] = "This is a string";
char s2[20];

s1 = "Another string";          // error : invalid array assignment
s2 = s1;                        // error : invalid array assignment
```

```
strcpy( s2, "")              // s2 set to null string.
strcpy( s2, "hello world")   // s2 set to "hello world".
strcpy( s2, s1)              // s2 set to "This is a string" i,e contents of s1.
```

C Strings : strcpy

strcpy(s1, s2): copies string s2 into string s1.

A character array (including a C string) can not have a new value assigned to it after it is declared.

Why ?

- *Because they are **arrays.***
- *Assigning arrays won't work.*

```
char s1[20] = "This is a string";
char s2[20];

s1 = "Another string";          // error : invalid array assignment
s2 = s1;                        // error : invalid array assignment
```

```
strcpy( s2, "")                 // s2 set to null string.
strcpy( s2, "hello world")      // s2 set to "hello world".
strcpy( s2, s1)                 // s2 set to "This is a string" i,e contents of s1.
```

How about copying longer strings?

C Strings : strcpy

strcpy(s1, s2): copies string s2 into string s1.

A character array (including a C string) can not have a new value assigned to it after it is declared.

Why ?

- *Because they are **arrays.***
- *Assigning arrays won't work.*

```
char s1[20] = "This is a string";
char s2[20];

s1 = "Another string";          // error : invalid array assignment
s2 = s1;                        // error : invalid array assignment
```

```
strcpy( s2, "")                 // s2 set to null string.
strcpy( s2, "hello world")      // s2 set to "hello world".
strcpy( s2, s1)                 // s2 set to "This is a string" i,e contents of s1.
```

*Any thoughts on implementation of **strcpy(..)** ?*

How about copying longer strings?

- To avoid overflows, the size of the array pointed by *destination* shall be long enough to contain the same C string as *source*

C Strings : strcpy

strcpy(s1, s2): copies string s2 into string s1.

A character array (including a C string) can not have a new value assigned to it after it is declared.

```
char s1[20] = “This is a string”;
char s2[20];

s1 = “Another string”;          // error : invalid array assignment
s2 = s1;                        // error : invalid array assignment
```

Why ?

- *Because they are* ***arrays.***
- *Assigning arrays won’t work.*

```
strcpy( s2, “”)                 // s2 set to null string.
strcpy( s2, “hello world”)      // s2 set to “hello world”.
strcpy( s2, s1)                 // s2 set to “This is a string” i,e contents of s1.
```

How about copying longer strings?

- To avoid overflows, the size of the array pointed by *destination* shall be long enough to contain the same C string as *source*

Any thoughts on implementation of ***strcpy(..)*** *?*

- *Copy the elements one-by-one from second char array to first char array at corresponding positions.*

C Strings : strcat

strcat(s1, s2): concatenates string s2 onto the end of string s1

```
char s1[20] = “This is a string”;
s1 = s1 + “ world”;                    // error : invalid operands to binary expression.
```

C Strings : strcat

strcat(s1, s2): concatenates string s2 onto the end of string s1

```
char s1[20] = “This is a string”;
s1 = s1 + “ world”;                // error : invalid operands to binary expression.
```

```
char s1[20] = “hello”;
char s2[20] = “world”;

strcat( s1, “ my dear ” );         // s1 now is “hello my dear ”
strcat( s1, s2 );                  // s1 now is “hello my dear world”
```

The destination array must be large enough to hold the combined strings (including the null character).

C Strings : strcat

strcat(s1, s2): concatenates string s2 onto the end of string s1

char s1[20] = “This is a string”;
s1 = s1 + “ world”; *// error : invalid operands to binary expression.*

char s1[20] = “hello”;
char s2[20] = “world”;

strcat(** s1, “ my dear ” **); *// s1 now is “hello my dear ”*
strcat(** s1, s2 **); *// s1 now is “hello my dear world”*

The destination array must be large enough to hold the combined strings (including the null character).

*Any thoughts on implementation of **strcat(..)** ?*

C Strings : strcat

strcat(s1, s2): concatenates string s2 onto the end of string s1

```
char s1[20] = “This is a string”;
s1 = s1 + “ world”;                  // error : invalid operands to binary expression.
```

```
char s1[20] = “hello”;
char s2[20] = “world”;

strcat( s1, “ my dear ” );           // s1 now is “hello my dear ”
strcat( s1, s2 );                    // s1 now is “hello my dear world”
```

The destination array must be large enough to hold the combined strings (including the null character).

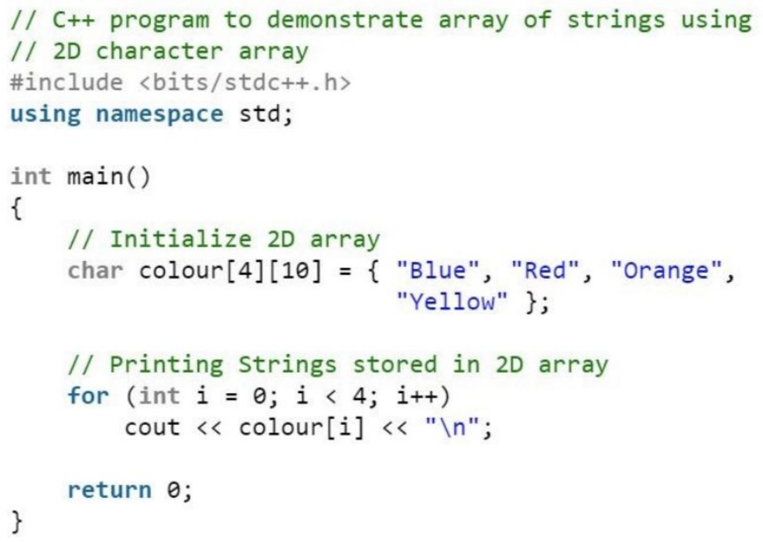
Any thoughts on implementation of ***strcat(..)*** *?*

- *Loop through second array and add elements into first array from index of ‘\0’ character.*

C Strings : Array of strings

In C, a string is a 1-D array of characters and an array of strings is a 2D array of characters

char ***colour*** *[4][10];*
// We can store 4 strings, each with a maximum length of ?

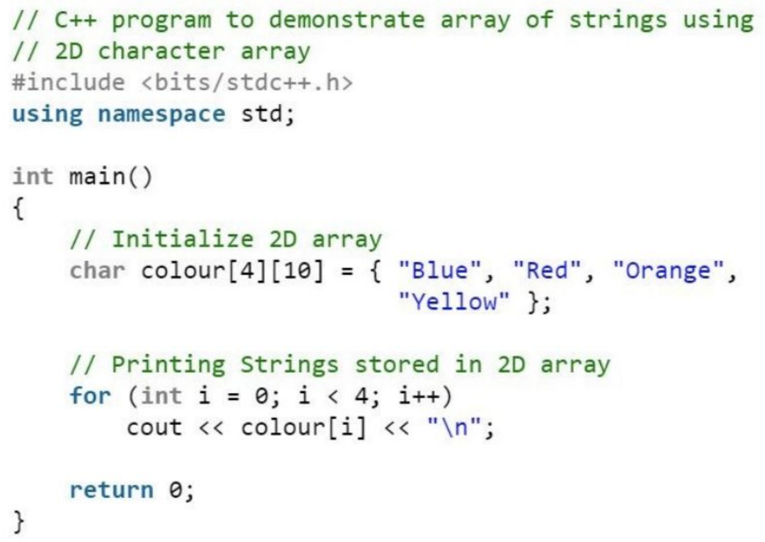


```
// C++ program to demonstrate array of strings using
// 2D character array
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // Initialize 2D array
    char colour[4][10] = { "Blue", "Red", "Orange",
                           "Yellow" };

    // Printing Strings stored in 2D array
    for (int i = 0; i < 4; i++)
        cout << colour[i] << "\n";

    return 0;
}
```

C Strings : Array of strings

In C, a string is a 1-D array of characters and an array of strings is a 2D array of characters

*char **colour** [4][10];*
// We can store 4 strings, each with a maximum length of (9)



```
// C++ program to demonstrate array of strings using
// 2D character array
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // Initialize 2D array
    char colour[4][10] = { "Blue", "Red", "Orange",
                           "Yellow" };

    // Printing Strings stored in 2D array
    for (int i = 0; i < 4; i++)
        cout << colour[i] << "\n";

    return 0;
}
```

Summary : C Strings

C Strings

- Declaration
- Initialization
- strlen - length
- strcmp - compare
- strcpy - copy
- strcat - concatenate

Summary : C Strings

C Strings

- Declaration
- Initialization
- strlen - length
- strcmp - compare
- strcpy - copy
- strcat - concatenate

Questions??

Project 5

Comments on Project submissions

Make sure you have

- **Comments** in code
- **Pseudocode** for design.
  This should be high level. No need of detailed explanation.
  Check http://web.cs.ucla.edu/classes/fall22/cs31/pseudocode.html
- **Test cases**
  This can be done even if your code is unfinished.
  Think about coverage
    - cover all functionality.
    - boundary conditions.
    - Make sure to write reason for each test.

Comments on Project submissions

Make sure you have

- **Comments** in code
- **Pseudocode** for design.
  This should be high level. No need of detailed explanation.
  Check http://web.cs.ucla.edu/classes/fall22/cs31/pseudocode.html
- **Test cases**
  This can be done even if your code is unfinished.
  Think about coverage
    cover all functionality.
    boundary conditions.
    Make sure to write reason for each test.

Best practices

- Start coding with comments.
- Write high level pseudocode first(in comments), manually check if it works on couple of general cases.
- Implement it.
- Write tests, covering all functionality/cases.
- Validate, debug and improve your code.

Later, you can just copy paste from your code file into your report for submission.

Project 5 :

— You will get to work on C strings.

— Setup deals with program split across multiple source files, and run a program that reads from a file

— Before you delve into the details of writing the code, it would be wise to ensure that you can do these new things correctly.

— Do the “Getting Started” part as soon as possible to figure out setup issues early.

— Start Early!!!