CS 31 Week 5
Discussion 2E

Srinath

Announcements

- Project 4 is up! Has 2 parts. **Part 1 is due 11:00 PM Sunday, October 30. Part 2 is due 11:00 PM Wednesday, November 2**
- Fill out the LA feedback form by **5 pm Monday, October 31.**

Outline

- 1D Arrays
- 2D Arrays
- Project 4
- Worksheet 5

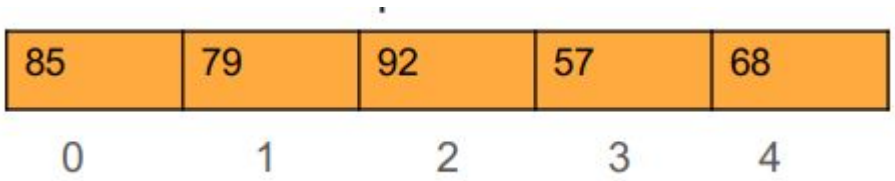
1D Arrays

1D Arrays :

Arrays are structured data types.

Values can be stored and accessed from the structure.

Think of them as sequential buckets, storing values of defined type.



Items in an array can be accessed using its **index,** which starts from **0.**

1D Arrays : Defining

Declaring

```
int scores [120];
string names [51];
double taxes [7];
```

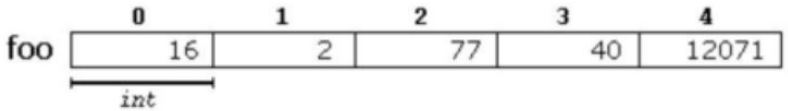
1D Arrays : Defining

Declaring

```
int scores [120];
string names [51];
double taxes [7];
```

Initialize

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```



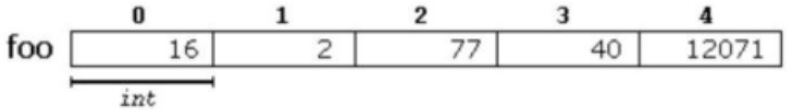
1D Arrays : Defining

Declaring

```
int scores [120];
string names [51];
double taxes [7];
```

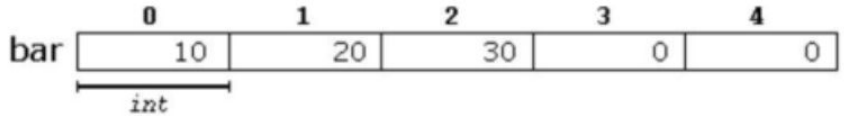
Initialize

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```




```
int bar [5] = { 10, 20, 30 };
```

If you specify lesser elements, they will be populated from start of the array



Note : The remaining elements will be initialized to default value.

1D Arrays : Defining

Initialize

```
int duration [5];
```

What will be the values?

1D Arrays : Defining

Initialize

```
int duration [5];
```
If initializer is not specified, value will be some garbage.

```
int heights [5] = { };
```
What will be the values?

1D Arrays : Defining

Initialize

```
int duration [5];
```
If initializer is not specified, value will be some garbage.

```
int heights [5] = { };
```
The values will be initialized with default value.

```
double weights [] = {10, 5, 6, 19, 21}
```
What will be the size?

1D Arrays : Defining

Initialize

int duration [5];
If initializer is not specified, value will be some garbage.

int heights [5] = { };
The values will be initialized with default value.

double weights [] = {10, 5, 6, 19, 21}
If you do not specify size, compiler can infer it.

1D Arrays : Defining

Initialize

int duration [5];
If initializer is not specified, value will be some garbage.

int heights [5] = { };
The values will be initialized with default value.

double weights [] = {10, 5, 6, 19, 21}
If you do not specify size, compiler can infer it.

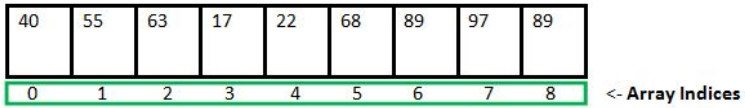
```
#include <iostream>
using namespace std;

void print(int arr[], int n) {
    // prints elements of the array
}

int main() {
    cout << "Hello World" << endl;
    int N=5;
    int arr[N];
    print(arr, N);
    int arr1[5]={9, 8, 7, 6, 5};
    print(arr1, N);
    int arr2[5] = {};
    print(arr2, N);
    int arr3[5] = {13, 14, 15};
    print(arr3, N);
}
```

Output :

```
Hello World
-516134080 32766 243536080 1 -516134208
9 8 7 6 5
0 0 0 0 0
13 14 15 0 0
```

1D Arrays : Access

score



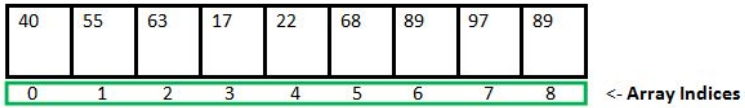
Array Length = 9
First Index = 0
Last Index = 8

int val = **score[5]**; // getting a value from the array
cout << val << endl;

Output : ?

1D Arrays : Access

score



Array Length = 9
First Index = 0
Last Index = 8

int val = **score[5]**; // getting a value from the array
cout << val << endl;

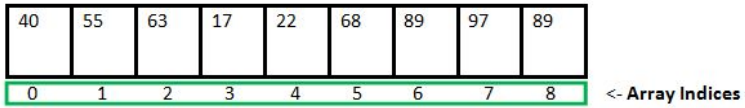
Output : 68

score[3] = 99; // inserting|overriding a value into the array
int val1 = score[3];
cout << val1 << endl;

Output : ?

1D Arrays : Access

score



Array Length = 9
First Index = 0
Last Index = 8

int val = **score[5]**; // getting a value from the array
cout << val << endl;

Output : 68

score[3] = 99; // inserting|overriding a value into the array
int val1 = score[3];
cout << val1 << endl;

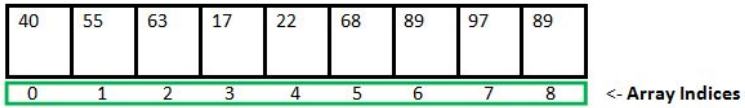
Output : 99

Other valid operations with arrays:

```
1 foo[0] = a;
2 foo[a] = 75;
3 b = foo [a+2];
4 foo[foo[a]] = foo[2] + 5;
```

1D Arrays : Access

score



Array Length = 9
First Index = 0
Last Index = 8

int val = **score[5]**; // getting a value from the array
cout << val << endl;

Output : 68

score[3] = 99; // inserting|overriding a value into the array
int val1 = score[3];
cout << val1 << endl;

Output : 99

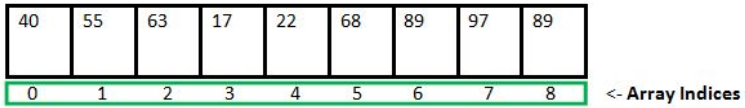
Other valid operations with arrays:

```
1 foo[0] = a;
2 foo[a] = 75;
3 b = foo [a+2];
4 foo[foo[a]] = foo[2] + 5;
```

What if you try accessing elements beyond length of array?
score[-2], score[9], score[200]

1D Arrays : Access

score



Array Length = 9
First Index = 0
Last Index = 8

int val = **score[5]**; // getting a value from the array
cout << val << endl;

Output : 68

score[3] = 99; // inserting|overriding a value into the array
int val1 = score[3];
cout << val1 << endl;

Output : 99

Other valid operations with arrays:

```
1 foo[0] = a;
2 foo[a] = 75;
3 b = foo [a+2];
4 foo[foo[a]] = foo[2] + 5;
```

What if you try accessing elements beyond length of array?
score[-2], score[9], score[200]

- Undefined behaviour, may not always give you error.

1D Arrays : Looping

How can we loop through to access each element of the array ?

1D Arrays : Looping

```
#include <iostream>
using namespace std;

int main() {
        int scores [ ] = {1, 2, 3, 4, 5};
        int result = 0;
        for(int i=0; i<5; i++) {
                result += scores[i];
        }
        cout << result << endl;
        return 0;
}
```

Output : ?

1D Arrays : Looping

```
#include <iostream>
using namespace std;

int main() {
        int scores [ ] = {1, 2, 3, 4, 5};
        int result = 0;
        for(int i=0; i<5; i++) {
                result += scores[i];
        }
        cout << result << endl;
        return 0;
}
```

Output : 15

1D Arrays : as function parameters

Can a function have array as its parameter?

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Can a function have array as its parameter? - Yes

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Can a function have array as its parameter? - Yes

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Can a function have array as its parameter? - Yes

Why do we pass size separately?

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Can a function have array as its parameter? - Yes

Why do we pass size separately?
In C++ there is no standard way of getting the array size, so always need a separate variable to keep track of it.

Remember to pass size too.

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Can a function have array as its parameter? - Yes

Why do we pass size separately?
In C++ there is no standard way of getting the array size, so always need a separate variable to keep track of it.

Remember to pass size too.

When calling the function with array as parameter
Is the array copied?

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Can a function have array as its parameter? - Yes

Why do we pass size separately?
In C++ there is no standard way of getting the array size, so always need a separate variable to keep track of it.

Remember to pass size too.

When calling the function with array as parameter
Is the array copied? - No
Is it pass by value?

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Can a function have array as its parameter? - Yes

Why do we pass size separately?
In C++ there is no standard way of getting the array size, so always need a separate variable to keep track of it.

Remember to pass size too.

When calling the function with array as parameter
Is the array copied? - No
Is it pass by value? - No
Will the original array get modified if its modified in the function?

1D Arrays : as function parameters

```
double average(int heights[ ], int size) {
        //.. do whatever with the array
}
```

Calling the above function

```
int classHeights [ ] = {1, 2, 3, 4, 5};
double averageHeight = average(classHeights, 5);
cout << averageHeight << endl;
```

Passing arrays to functions in C/C++ are passed by reference.

Can a function have array as its parameter? - Yes

Why do we pass size separately?
In C++ there is no standard way of getting the array size, so always need a separate variable to keep track of it.

Remember to pass size too.

When calling the function with array as parameter
Is the array copied? - No
Is it pass by value? - No
Will the original array get modified if its modified in the function? - Yes
It's pass by reference - notice that we didn't need the '&' before.

1D Arrays : Examples

int arr[5] = {99, 98, 97,96,95,94, 93};

Will this compile?

1D Arrays : Examples

int arr[5] = {99, 98, 97,96,95,94, 93};

Will this compile?

- No, **error: excess elements in array initializer**

1D Arrays : Examples - Print

```
#include <iostream>
using namespace std;

void printArray(int arr [ ], int n) {
        for(int i=0; i<n; i++){
                cout << arr[i] << " ";
        }
        cout << endl;
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        Int heights [ ] = {4, 7, 9};
        printArray(scores, 5);
        printArray(heights, 3);
        printArray(scores, 4);
        return 0;
}
```

Output :

1D Arrays : Examples - Print

```
#include <iostream>
using namespace std;

void printArray(int arr [ ], int n) {
        for(int i=0; i<n; i++){
                cout << arr[i] << " ";
        }
        cout << endl;
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        Int heights [ ] = {4, 7, 9};
        printArray(scores, 5);
        printArray(heights, 3);
        printArray(scores, 4);
        return 0;
}
```

Output :

11, 12, 13, 14, 15
4, 7, 9
11, 12, 13, 14

1D Arrays : Examples - Fill In

```
#include <iostream>
using namespace std;

void printArray(int arr [ ], int n) {
        for(int i=0; i<n; i++){
                cout << arr[i] << " ";
        }
        cout << endl;
}

void fill(int my_arr [ ], int s) {
        for(int i=0; i<s; i++){
                my_arr[i] = i*i;
        }
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        printArray(scores, 5);
        fill(scores, 5);
        printArray(scores, 5);
        return 0;
}
```

Output :

1D Arrays : Examples - Fill In

```
#include <iostream>
using namespace std;

void printArray(int arr [ ], int n) {
        for(int i=0; i<n; i++){
                cout << arr[i] << " ";
        }
        cout << endl;
}

void fill(int my_arr [ ], int s) {
        for(int i=0; i<s; i++){
                my_arr[i] = i*i;
        }
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        printArray(scores, 5);
        fill(scores, 5);
        printArray(scores, 5);
        return 0;
}
```

Output :

11, 12, 13, 14, 15
0, 1, 4, 9, 16

1D Arrays : Examples - Copy

Copy given array into another array

1D Arrays : Examples - Copy

Copy given array into another array

```
#include <iostream>
using namespace std;

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};

        // copy scores to new array copied_scores
        int N = 5;
        int copied_scores [N];
        for(int i=0; i<N; i++){
                copied_scores[i] = scores[i];
        }
        // copy complete.

        return 0;
}
```

1D Arrays : Examples - Reverse

Create new array with elements in reverse order.
Eg :
Given array : 3, 5, 23, 1, 8, 21, 15
Reversed array : 15, 21, 8, 1, 23, 5, 3

1D Arrays : Examples - Reverse

Create new array with elements in reverse order.
Eg :
Given array : 3, 5, 23, 1, 8, 21, 15
Reversed array : 15, 21, 8, 1, 23, 5, 3

```
#include <iostream>
using namespace std;

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};

        // creating reverse of scores
        int N = 5;
        int reverse_scores [N];
        for(int i=0; i<N; i++){
                reverse_scores[i] = scores[??];
        }
        // reverse complete.

        return 0;
}
```

1D Arrays : Examples - Reverse

Create new array with elements in reverse order.
Eg :
Given array : 3, 5, 23, 1, 8, 21, 15
Reversed array : 15, 21, 8, 1, 23, 5, 3

```
#include <iostream>
using namespace std;

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};

        // creating reverse of scores
        int N = 5;
        int reverse_scores [N];
        for(int i=0; i<N; i++){
                reverse_scores[i] = scores[N-i-1];
        }
        // reverse complete.

        return 0;
}
```

1D Arrays : Examples - Swap

Given two array indices, how to swap the elements at those indices?

void ***swap****(int arr [ ], int i, int j)*

1D Arrays : Examples - Swap

Given two array indices, how to swap the elements at those indices?

*void **swap**(int arr [ ], int i, int j)*

```
#include <iostream>
using namespace std;

void swap(int arr [ ], int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        swap(scores, 0, 1);
        swap(scores, 2, 4);
        printArray(scores, 5); // assume from previous codes
        return 0;
}
```

Output :

1D Arrays : Examples - Swap

Given two array indices, how to swap the elements at those indices?

*void **swap**(int arr [ ], int i, int j)*

Might be useful for project 4 !

```
#include <iostream>
using namespace std;

void swap(int arr [ ], int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}

int main() {
        int scores [ ] = {11, 12, 13, 14, 15};
        swap(scores, 0, 1);
        swap(scores, 2, 4);
        printArray(scores, 5); // assume from previous codes
        return 0;
}
```

Output :

12, 11, 15, 14, 13

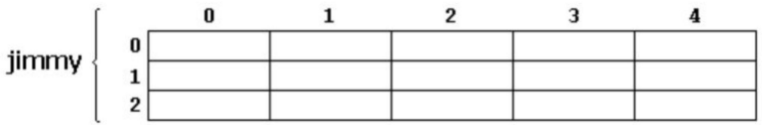
2D Arrays

2D Arrays :

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

Think of them as “array of arrays”

int jimmy [3][5];

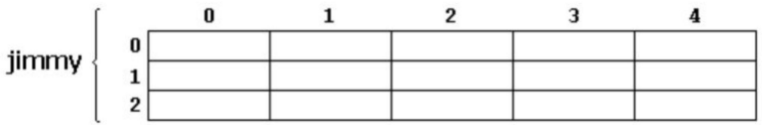


2D Arrays :

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

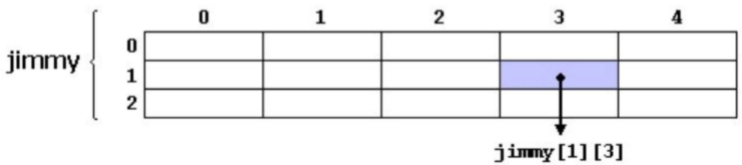
Think of them as “array of arrays”

int jimmy [3][5];



Accessing element at position 1, 3

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

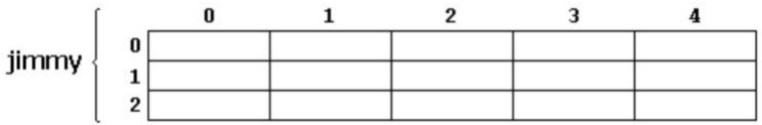


2D Arrays :

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

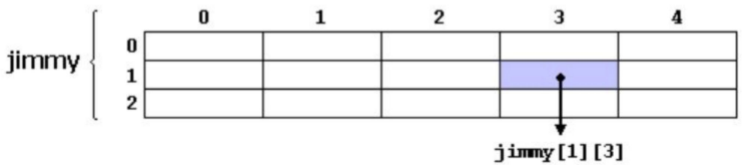
Think of them as “array of arrays”

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];

Project 4

Project 4 :

- Implement some of the useful functions on array of strings.

```
int appendToAll(string a[], int n, string value);
int lookup(const string a[], int n, string target);
int positionOfMax(const string a[], int n);
int rotateLeft(string a[], int n, int pos);
..
...
..
int lookupAny(const string a1[], int n1, const string a2[], int n2);
int split(string a[], int n, string splitter);
```

- If you wish, you may write functions in addition to those required.
- If you wish, your implementation of a function required here may call other functions required here.
- Be careful about accessing elements outside of array bounds

Project 4 :

- Implement some of the useful functions on array of strings.

```
int appendToAll(string a[], int n, string value);
int lookup(const string a[], int n, string target);
int positionOfMax(const string a[], int n);
int rotateLeft(string a[], int n, int pos);
..
...
..
int lookupAny(const string a1[], int n1, const string a2[], int n2);
int split(string a[], int n, string splitter);
```

- If you wish, you may write functions in addition to those required.
- If you wish, your implementation of a function required here may call other functions required here.
- Be careful about accessing elements outside of array bounds

A general suggestion :

- Don’t directly jump into coding.
- Think about the algorithm first, check on couple of examples manually, then start coding :)
- A trivial way of solving is think about “how you do it”.
- Eg: How do you find maximum of given numbers 5, 6, 1, 4, 16, 9?

Project 4 :

- Implement some of the useful functions on array of strings.

Start Early!!!

```
int appendToAll(string a[], int n, string value);
int lookup(const string a[], int n, string target);
int positionOfMax(const string a[], int n);
int rotateLeft(string a[], int n, int pos);
..
...
..
int lookupAny(const string a1[], int n1, const string a2[], int n2);
int split(string a[], int n, string splitter);
```

- If you wish, you may write functions in addition to those required.
- If you wish, your implementation of a function required here may call other functions required here.
- Be careful about accessing elements outside of array bounds

A general suggestion :

- Don’t directly jump into coding.
- Think about the algorithm first, check on couple of examples manually, then start coding :)
- A trivial way of solving is think about “how you do it”.
- Eg: How do you find maximum of given numbers 5, 6, 1, 4, 16, 9?

Summary : Arrays

Arrays

- Declaration
- Initialization
- Access
- Looping
- Passing as parameters

Summary : Arrays

Arrays

- Declaration
- Initialization
- Access
- Looping
- Passing as parameters

Questions??