Introduction to Programming (CS 101)

Spring 2024

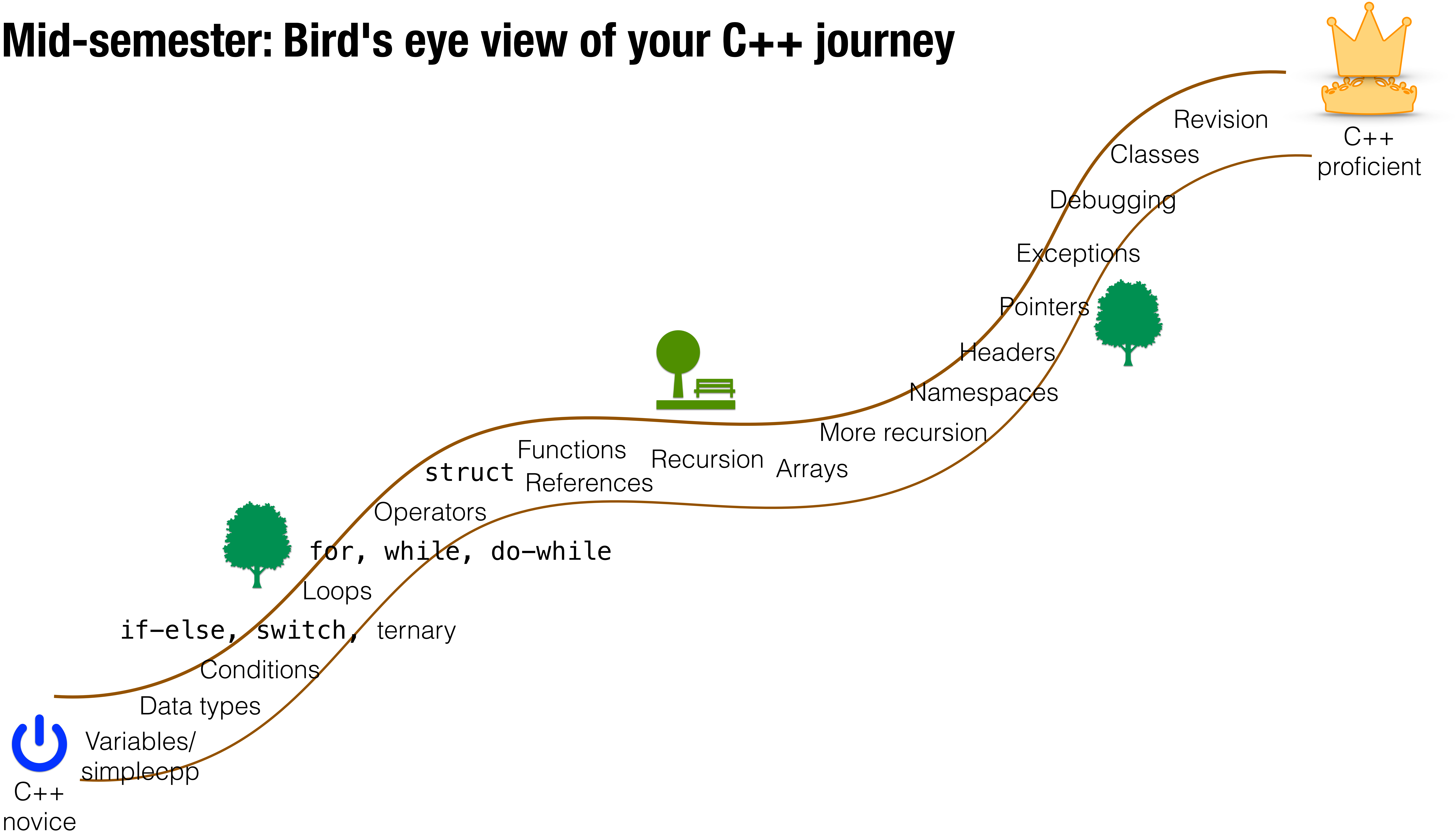
Lecture 13:

Arrays (continued), sorting

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade

Mid-semester: Bird's eye view of your C++ journey



C++ novice

Variables/ simplecpp

Data types

Conditions

`if-else, switch,` ternary

Loops

`for, while, do-while`

Operators

`struct`

Functions

References

Recursion

Arrays

More recursion

Namespaces

Headers

Pointers

Exceptions

Debugging

Classes

Revision

C++ proficient

Recap: Recursion

- Recall finding the largest digit in a non-negative number. How do we do this recursively?

```
int findLargestDigit(int n) {
  int md = 0;
  while(n > 0) {
    md = max(md, n%10);
    n /= 10;
 }
  return md;
}

int main() {
  int n;
  cin >> n;
  cout << findLargestDigit(n) << endl;
}
```

Recap: Recursion

- Finding the largest digit in a non-negative number recursively.

```
int findLargestDigit(int n) {
  if(n < 10) return n; //Base case
  else
    return max(n%10, findLargestDigit(n/10));
}

int main() {
  int n;
  cin >> n;
  cout << findLargestDigit(n) << endl;
}
```

Recap: Recursion

- Consider the following function to exponentiate an integer n to a specified exponent k.

```
int exponentiate(int n, int k)
{
   int result = 1;
   for (int i = 0; i < k; i++)  result *= n;
   return result;
}
```

- How do you solve this recursively?

```
int exponentiate(int n, int k)
{
   if(k == 0) return 1;
   return (n * exponentiate(n, k-1));
}
```

Recap: Recursion

- Given a target number and a sorted array, use binary search to find whether the target number exists in the array

Demo

```
bool binarySearch(int arr[], int l, int r, int num) {
  if(l <= r) {
    int mid = l + (r - l)/2;
    if(arr[mid] == num) return true;
    if(arr[mid] > num) return binarySearch(arr, l, mid-1, num);
    return binarySearch(arr, mid + 1, r, num);
  }
  return false;
}

int main() {
  int A[] = {1, 4, 55, 88, 191, 222, 245}, num;
  cin >> num;
  cout << binarySearch(A, 0, 6, num);
}
```

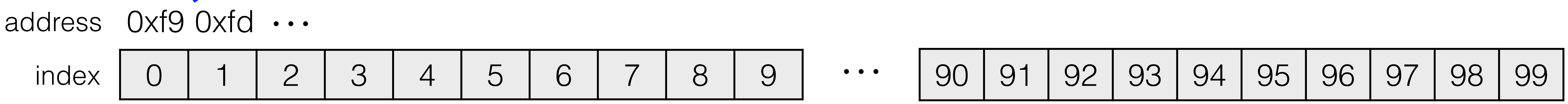
1D Arrays

CS 101, 2025

Recap: Arrays

Byte-level address in memory; accessed as `&A[0],&A[1]`, ... and typically shown in hexadecimal format

- `int A[100];`// defines an array of 100 integers, allocates 100 contiguous 4-byte memory units



- Can define both ordinary variables and arrays in the same statement. E.g., `int B[10], num;`

- Arrays are 0-indexed. First element of `A` above is in `A[0]` and last element is in `A[99]`. Elements of an array can be treated like a regular variable. Operations below are all permitted:
  - `A[9] = 9;`
  - `int x = 5 + A[2];`
  - `int a = 1; A[a] = 3; B[a*3] = 10;`

- Array size should be known at compile time. It is the programmer's responsibility to stay within the array bounds in their code.

Arrays and functions

- A function can take array parameters

Note: You do not need to pass the size here. Size information is not encoded as part of the array argument.

```
void printArray(int A[], int n) {
    for (int i=0; i<n; i++)
       cout << A[i] << ((i == n-1) ? '\n' : ' ');
}
```

Size is passed as a separate argument

- When you pass an array as an argument to a function, it passes the starting address of the array in memory. Passing the array does not encode its size; size needs to be passed separately.

- It is the responsibility of the calling program to ensure that an array's passed in the first argument and it is at least as large as the length passed in the second argument

- ***Arrays are always passed by reference***
    - Modifications to an array A passed to a function will be reflected in the calling function

- A function cannot return an array type

Sorting: Passing an array to a function

- Bubble sort:

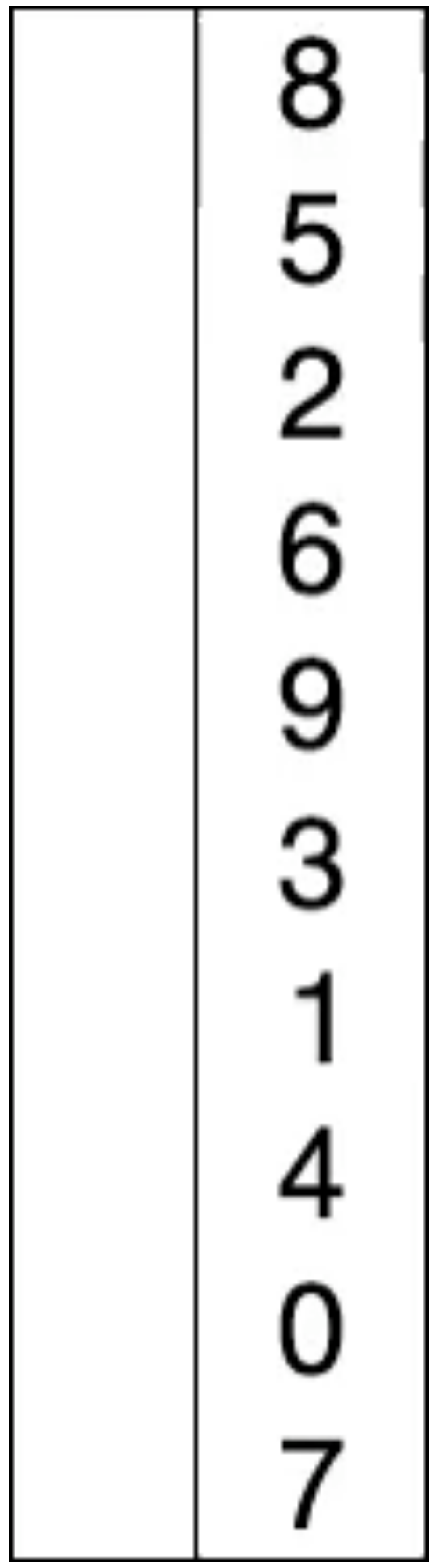
```
#include <iostream>
using namespace std;

void bubblesort(int A[], int n) {
   for(int i=0; i<n-1; i++) {
      for(int j=0; j<n-i-1; j++) {
         if(A[j] > A[j+1])
            swap(A[j],A[j+1]);
      }
   }
}
int main() {
  ... // read numbers to be sorted into A[0] to A[n-1]
  bubblesort(A, n);
  ... // print out sorted A
}
```

Video derived from: https://en.wikipedia.org/wiki/Bubble_sort#/media/File:Bubble-sort-example-300px.gif

Sorting: Passing an array to a function

- Sorting algorithm that's a little more efficient than bubble sort:

```
#include <iostream>
using namespace std;
void min_to_beg(int A[], int i, int n) {
   int min_i=i;
   for(int j=i+1; j<n; j++)
      if(A[j] < A[min_i])
         min_i = j;
   swap(A[min_i],A[i]);
}
int main() {
  ... // read numbers to be sorted into A[0] to A[n−1]
  for (int i=0; i<n−1; i++)
    min_to_beg(A,i,n); // call min_to_beg on the unsorted A[i...n−1]
  ... // print out sorted A
}
```



Video derived from:https://en.wikipedia.org/wiki/Selection_sort#/media/File:Selection-Sort-Animation.gif

More about arrays

- The function `sizeof` returns the size of an entire array in bytes.

```
int out[10];
cout << sizeof(out);  // will print 40
```

- An array's size cannot be changed once it is created. To change size, you must create another new array.
- Arrays so far are statically allocated and memory is allocated on the stack
  - One can dynamically allocate memory where the array is stored in the ***heap*** -- Next class!
- Can create array of struct type. For example:

```
struct Book {
  string title; float price;
};
Book shelf[5];
shelf[0].title = "The Metamorphosis"; shelf[0].price = 123.5;
```

Character Arrays

CS 101, 2025

`char` arrays

- Declare a C-style array of type `char`

```
    char str[20];
```

- Say we want `str` to store a character string "hello", i.e. `'h'` in `str[0]`, `'e'` in `str[1]`, and so on.

- When we print `str`, we want only "hello" to be printed to the screen. For which, length will need to be encoded. The convention adopted from C to C++ is to, instead, add a special null character (`'\0'`, ASCII code 0) to the end of the actual string.

- If we write `char str[20] = "hello";` it will place the 5 characters in hello in the first 5 elements of `str`. The sixth element will be set to `'\0'`.

- `cout str;` will just print out the contents of `str` to the screen (excluding `'\0'`)

`char` arrays

- Declare a C-style array of type `char`

```
char str1[] = "hello";
```

- In the example above, an array of 6 `char` is created indexed from 0 to 5.

- Initialising as shown above creates a ***C-style string***

- A C-style string is a null-terminated `char` array. The compiler inserts the null. The following are all equivalent:

```
char str1[] = "hello"; //compiler determines array size
char str2[30] = "hello"; //extra room available
char str3[] = {'h', 'e', 'l', 'l', 'o', '\0'}; //'\0' represents null
char str4[6] = "hello"; //exact space to accommodate hello + '\0'
```

Reading a string into a `char` array

```
char name[30];
cin >> name;
```

- From what you type on the console (via `cin`):
  - Any leading whitespaces will be ignored
  - Everything from the first non-whitespace char until the next whitespace char will be stored in `name`
  - E.g., if I had typed "   `Preethi Jyothi`", `Preethi` would be saved in `name`
  - The null character `'\0'` is automatically placed by the compiler after the last character (i.e., `i` in the example above).
  - Note that "Jyothi" is lost
  - To read whitespace-delimited strings, we need `getline`

`getline` command

```
char name[30];
cin.getline(name, 30);
```

- With `cin.getline(A, n)`, everything typed by the user including whitespaces will be placed in memory, starting from the first element of `A`, until one of the following happens:
  - A newline character is typed by the user OR
  - `n−1` characters are typed without a newline, in which case all the `n−1` characters including a `'\0'` are placed in `A`

- Common to use the length of `A` as an argument to `getline`, as in the example shown above

- Note `cin.getline(...)` is different from `getline(cin, name)` that reads from the console into variable `name` of type `string`

Example: Copying a string from one array to another

- Copy a string stored in an array A to another array B

```
void strcopy(char A[], char B[]) {
  //assume B is long enough to hold contents of A
  int i;
  for(i = 0; A[i] != '\0'; i++) B[i] = A[i];
  B[i] = A[i];
}
```

Example: What does the following code do?

```
int whatami(char A[], char B[]) {
   int i = 0;
   while(true) {
      if(A[i] == '\0' && B[i] == '\0') return 0;
      if(A[i] == '\0') return -1;
      if(B[i] == '\0') return 1;
      if(A[i] < B[i]) return -1;
      if(A[i] > B[i]) return 1;
      i++;
   }
}
int main() {
   char A[30], B[30];
   cin.getline(A,30); cin.getline(B,30);
   cout << A << " " << B << " " << whatami(A,B) << endl;
}
```

Takes two strings A, B and returns 0 if they are equal, 1 if A is lexicographically smaller than B and -1 otherwise.

Multidimensional Arrays

CS 101, 2025

Two-dimensional arrays

- Sequences are naturally represented as 1D (one-dimensional) arrays
- Objects like matrices can be naturally represented using two sets of indices.
  C++ offers 2D (two-dimensional) arrays to represent such objects.
- Example:

Space for **10*20** variables of type float are allocated in the stack.

```
float vals[10][20];
```

```
vals
```

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,10	0,11	0,12	0,13	0,14	0,15	0,16	0,17	0,18	0,19
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11	1,12	1,13	1,14	1,15	1,16	1,17	1,18	1,19
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10	2,11	2,12	2,13	2,14	2,15	2,16	2,17	2,18	2,19
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11	3,12	3,13	3,14	3,15	3,16	3,17	3,18	3,19
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,11	4,12	4,13	4,14	4,15	4,16	4,17	4,18	4,19
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10	5,11	5,12	5,13	5,14	5,15	5,16	5,17	5,18	5,19
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14	6,15	6,16	6,17	6,18	6,19
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9	7,10	7,11	7,12	7,13	7,14	7,15	7,16	7,17	7,18	7,19
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9	8,10	8,11	8,12	8,13	8,14	8,15	8,16	8,17	8,18	8,19
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9	9,10	9,11	9,12	9,13	9,14	9,15	9,16	9,17	9,18	9,19

Two-dimensional arrays

- Sequences are naturally represented as 1D (one-dimensional) arrays
- Objects like matrices can be naturally represented using two sets of indices.
  C++ offers 2D (two-dimensional) arrays to represent such objects.
- Example:

first dimension

```
float vals[10][20];
```

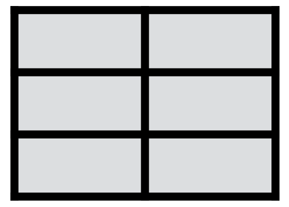
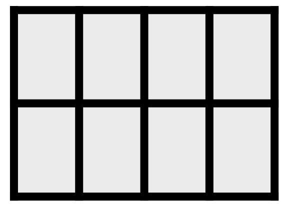
second dimension

Space for **10*20** variables of type float are allocated in the stack.

- Elements of `vals` are accessed as `vals[i][j]` where $0 \leq i < 10$, and $0 \leq j < 20$
- `vals` are stored in the memory in the row-major order, i.e., `vals[0][0]`, `vals[0][1]`,…, `vals[0][19]`, `vals[1][0]`,…, `vals[1][19]`,…,`vals[9][0]`,…,`vals[9][19]`
- How do we access 2D array elements? Use a loop for each dimension!

Multiplying two matrices

```
double a[3][2] = {{1,2},{3,4},{5,6}}, b[2][4] = {{1,2,3,4},{5,6,7,8}},
c[3][4];
```

Initializing a 3x2  and a 2x4  matrix

```
for(int i = 0; i < 3; i++) {
  for(int j = 0; j < 4; j++) {
    c[i][j] = 0;
    for(int k = 0; k < 2; k++)
      c[i][j] += a[i][k]*b[k][j];
  }
}
```

Using two loops to access c[i][j]

$$c[i][j] = \sum_{k=0}^{1} a[i][k] * b[k][j]$$

```
for(int i = 0; i < 3; i++) {
  for(int j = 0; j < 4; j++) cout << c[i][j] << " ";
  cout << endl;
}
```