Introduction to Programming (CS 101)
Spring 2024

Lecture 14:
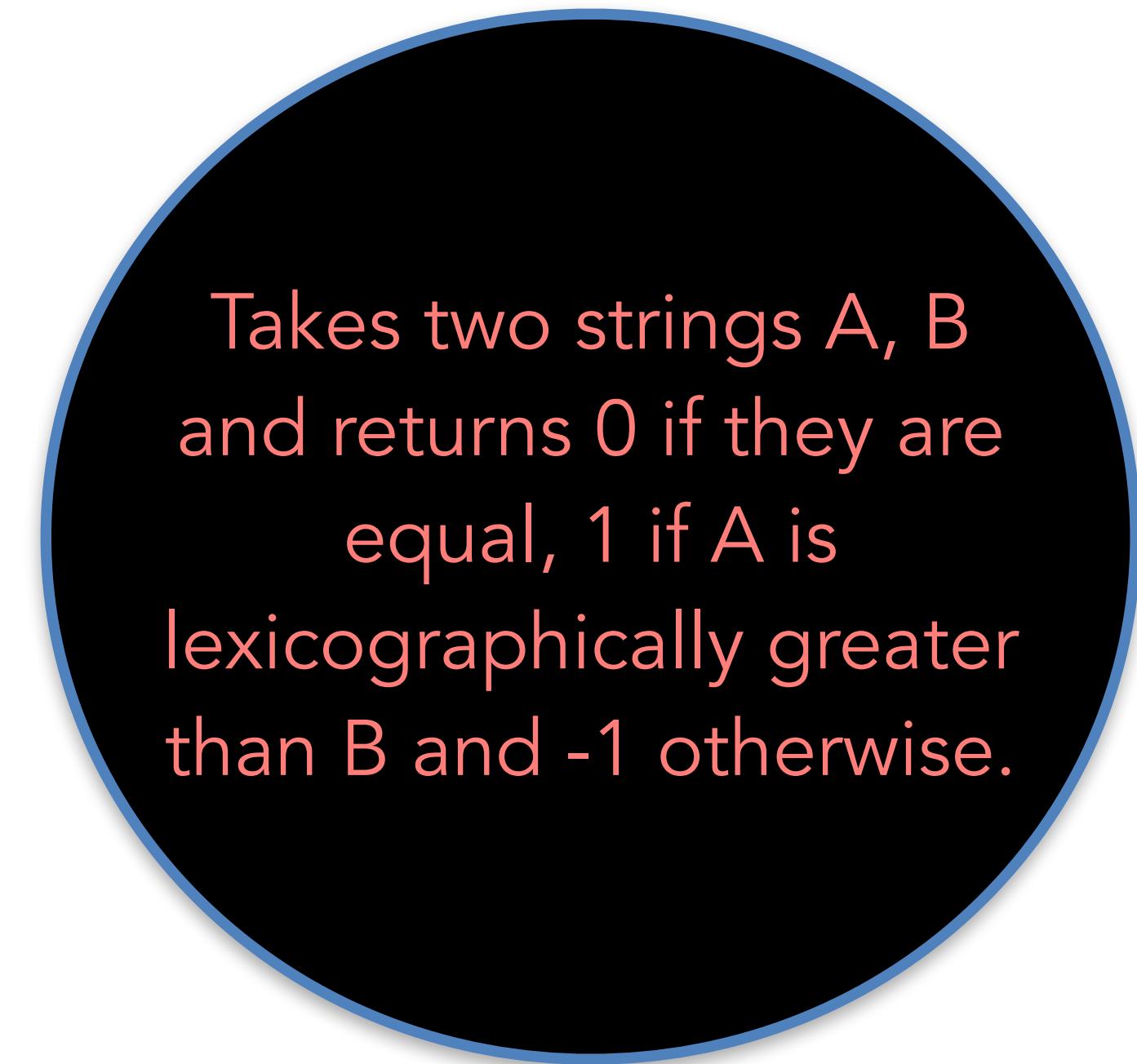
More about {arrays, structs, strings}

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Recap: Lexicographically compare two strings

```cpp
int compare(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 << " " << compare(A,B) << endl;
}
```

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

Recap

What is the output of the following program?
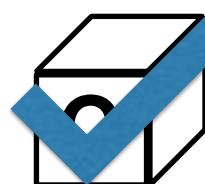
```
main_program {
    char c1[20] = "hello";
    char c2[] = {'h','e','l','l','o'};

    if(!compare(c1,c2))
        cout << "c1=c2";
    else if(compare(c1,c2) == 1)
        cout << "c1>c2";
    else
        cout << "c1<c2";
}
```

```
int compare(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++;
    }
}
```

A c1=c2

B c1>c2

✔ c1<c2

c1 is automatically terminated with a `'\0'`, while c2 isn't. Note that the assignment operator cannot be used again once a string is initialized. `c1 = "world";` after `char c1[20] = "hello";` will result in a compiler error.

Merge sort
CS 101, 2025
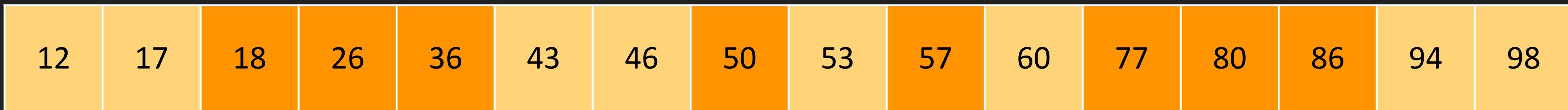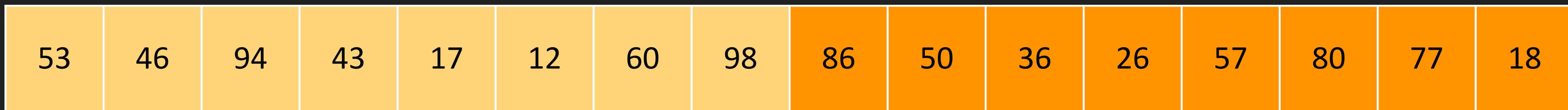
Divide-and-Conquer

- Some *algorithms* use a Divide-and-Conquer strategy
  (a.k.a. Divide-Conquer-and-Combine)
  - A problem instance is divided into
    two or more smaller problems
  - The smaller instances are solved
    recursively
  - The results are then combined to get
    the result for the original instance
- An example: Merge Sort

```
resultType f(inputType x) {

    if (baseCase(x))
        return handleBaseCase(x);

    inputType x1, x2;
    Divide(x,x1,x2);

    resultType y1, y2;
    y1 = f(x1); y2 = f(x2);

    return Combine(y1,y2);
}
```
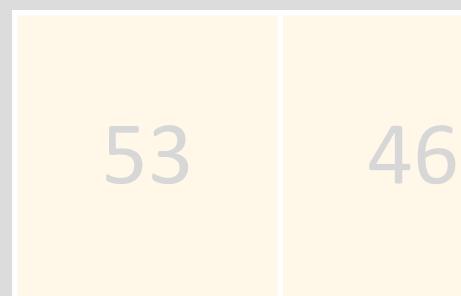
Merge Sort

- Split into two (almost) equal halves, and recursively sort each half

- Merge the two sorted arrays into a single sorted array
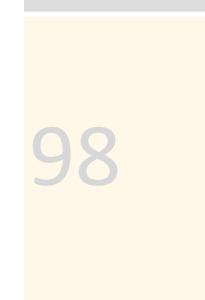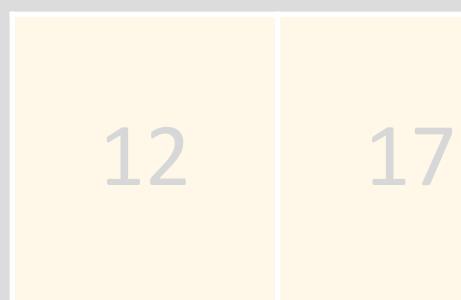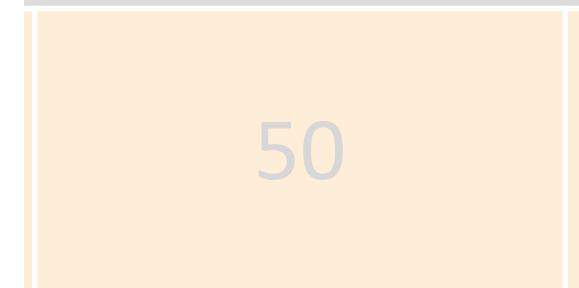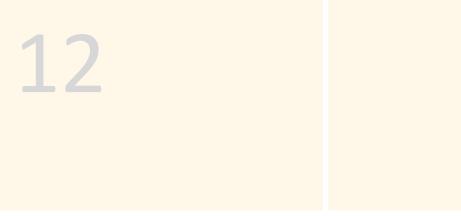
| 53 | 46 | 94 | 43 | 17 | 12 | 60 | 98 | 86 | 50 | 36 | 26 | 57 | 80 | 77 | 18 |

Recursively sort   Recursively sort

| 12 | 17 | 43 | 46 | 53 | 18 | 26 | 36 | 50 |

Merge

| 12 | 17 | 18 | 26 | 36 | 43 | 46 | 50 | 53 | 57 | 60 | 77 | 80 | 86 | 94 | 98 |

Merge Sort

- Split into two (almost) equal halves, and recursively sort each half
- Merge t

53   46   18

12   50

6 5 3 1 8 7 2 4

12 17   98
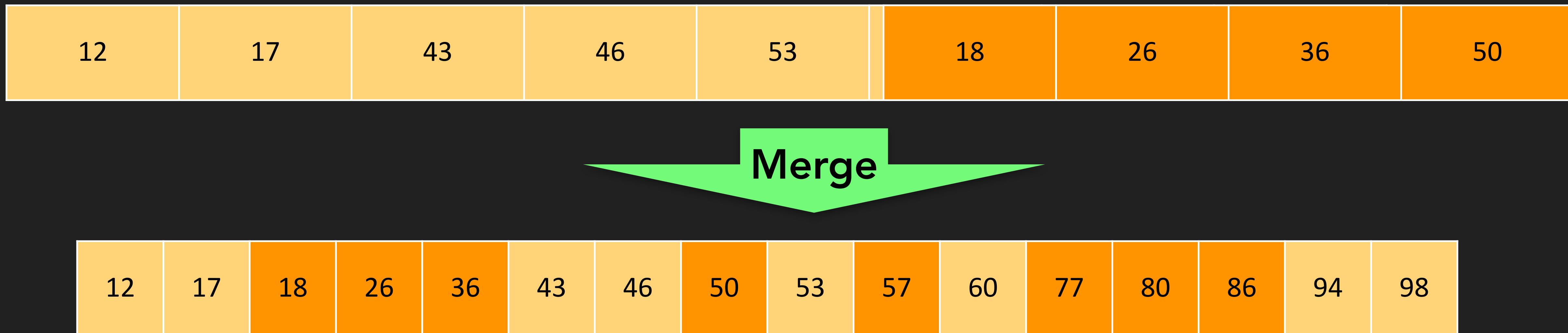
Merge Sort

```c
// merge X[left..mid] and X[mid+1..right] into Y[left..right], where mid = (left+right)/2
void merge(const int X[], int Y[], int left, int right) {
    int mid = (left+right)/2, L = left, R = mid+1;    // L,R: next indices of left/right halves
    for(int i=left; i <= right; ++i) {
        if(L <= mid && (R > right || X[L] <= X[R])) Y[i] = X[L++];    // copy from left
        else Y[i] = X[R++];                                          // copy from right
    }
}
```

12	17	43	46	53	18	26	36	50

Merge

12	17	18	26	36	43	46	50	53	57	60	77	80	86	94	98

Merge Sort

```
void sort (const int in[], int out[],
              int left, int right,
                int scratch[]) {


  if (left==right) {
    out[left] = in[left];

    return;
  }
  int mid = (left+right)/2;
  sort(in,scratch,left,mid,out);
  sort(in,scratch,mid+1,right,out);


  merge(scratch,out,left,right);
}
```

```
resultType f(inputType x) {

  if (baseCase(x))
     return handleBaseCase(x);

  inputType x1, x2;
  Divide(x,x1,x2);

  resultType y1, y2;
  y1 = f(x1); y2 = f(x2);


  return Combine(y1,y2);
}
```

More about multidimensional arrays
CS 101, 2025

2D arrays

- Recall matrices can be implemented using 2D arrays. Example:

```
float vals[2][3];
```

vals[0][0]	vals[0][1]	vals[0][2]
vals[1][0]	vals[1][1]	vals[1][2]

- We can define two-dimensional character arrays:

```
char countries[3][20] = {"India", "China", "Sri Lanka"};
```

- `countries[i]` will return the address of the zeroth character of the ith string in `countries`

2D arrays

```cpp
int main() {
  char countries[3][20] = {"India", "China", "Sri Lanka"};
  char capitals[3][20] = {"New Delhi", "Beijing", "Colombo"};
  char country[20];
  cout << "Enter country: ";
  cin.getline(country, 20);
  int i;
  for(i = 0; i < 3; i++) {
    if(compare(country,countries[i]) == 0) { //compare defined earlier
      cout << "Capital -> " << capitals[i] << endl; break;
    }
  }
  if(i == 3) cout << "Do not know the capital\n";
}
```

Passing two-dimensional arrays to functions

- One can pass a 2D array to a function. However, the second dimension must be given as a compile-time constant. Example:

```
void print(char countries[][20], int num) {
  for(int i = 0; i<num; i++) cout << countries[i] << endl;
}
```

- Such a `print` function can only be used with `char` arrays where the second dimension is 20

- Can be overcome with more flexible array types like `vector` (coming in later slides)

More about structs
CS 101, 2025

More about structs

- If a `struct` definition appears before many functions, it will be visible to all the functions

- To avoid making copies, structs can be passed by reference

```
struct CS101Group {
  char name[4]; //G1, G13, etc.
  string TAs[10];
  unsigned short int size;
  ⋮
};
void printGroup(const CS101Group& grp) {
  cout << "name = " << grp.name << endl;
  ⋮
}
```

Passing structs by value

- Consider the following struct representing 2D points with x, y coordinates:

```
struct Point {
    double x;
    double y;
};
```

- The following function returns a `Point` that is the midpoint of the line joining two points:

```
Point midpoint(Point a, Point b) {
    Point mp;
    mp.x = (a.x+b.x)/2;
    mp.y = (a.y+b.y)/2;
    return mp;
}
```

```
int main() {
    Point p1 = {0,0};
    Point p2 = {100,200};
    Point p3 = midpoint(p1, p2);
    cout << midpoint(midpoint(p1,p2),p2).y;
}
```

150 `output`

Passing structs by reference

- Can pass struct parameters by reference (to avoid copies)

```
Point midpoint(const Point& a, const Point& b) {
    Point mp;
    mp.x = (a.x+b.x)/2;
    mp.y = (a.y+b.y)/2;
    return mp;
}
```

`const` means these reference parameters will not be altered during execution

- Say the parameters of `midpoint` were ordinary references and not `const` references. Now you try to run `cout << midpoint(midpoint(p1,p2),p2).y;` from `main()`, as before. What happens?
  - Compiler error!
  - Cannot pass a temporary object like `midpoint(p1,p2)` to a non-const reference parameter

structs and member functions

- Functions can be a part of the structure itself. Example:

```cpp
struct Point {
  double x, y;
  double length() {
    return sqrt(x*x + y*y);
  }
  double shift(double dx, double dy) {
    x+=dx; y+=dy;
  }
};
```

- A member function is called on an object of the struct type using "." notation. Example:

```cpp
Point p1; cout << p1.length();
```

structs and member functions

- Inside the body of the function, we can read or modify members of the struct object. Example: `x, y` are directly accessible within the functions `length` and `shift` below

```
struct Point {
  double x, y;
  double length() {
    return sqrt(x*x + y*y);
  }
  double shift(double dx, double dy) {
    x+=dx; y+=dy;
  }
  void alter(double a) {
    x = a; cout << length() << endl;
  }
};
```

Note that `x` of a struct object will be altered here. And the call to `length()` will refer to the object used with `alter`. That is, `p1.alter(10)` will calculate the length of `p1.`

More about strings
CS 101, 2025

More about `string`

- To use `string`, we need to add `#include <string>` to the program

  ```
  string a = "hello", b = "world", c;
  c = a;
  ```

  Note that you do not need to worry about the size of the strings, unlike C-style strings

- Can read whitespace-delimited strings into a string using `getline`:

  ```
  getline(cin, a); //to read a line into string a
  ```

  Will read from console into `a`, until a newline is encountered

- Addition operator can be used with strings to concatenate them

  ```
  c = a + " " + b; //will set c to "hello world"
  ```

- `a[i]` denotes the ith character of the string `a`

More about `string`

- Many useful member functions are available for `string`

```
string a = "hello";
int i = a.find("he"); //returns the starting index of the
                           //first occurrence of "he" within string a
int j = a.find("l", 3); //find starting from index 3
```

- If a given string is not found, find returns a constant `string::npos`

- Comparison expressions `<, >, =` can be used for strings `a, b` assuming a lexicographic order

  - Lexicographic order: Similar to how words are organized in a dictionary

  - Comparison is done character by character, and based on underlying ASCII values

  - Example: `"hello"` < `"world"`, `"hello"` > `"Hello"` (ASCII value of h > ASCII value of H)

Dynamic arrays (`vector`)
CS 101, 2025

Dynamic allocation

- C++ contains three primary array styles:

  1. C-style arrays (inherited from C, you have already learned about these)

  2. `std::array` array type

  3. `std::vector` array type   } Part of C++'s Standard Template Library (STL)

- `std::vector` is arguably the most flexible of the three types and has many useful supporting features

- We will focus on `std::vector` since it offers many benefits

vector (I)

- To use vectors, add the header line "`#include <vector>`" to the beginning of the program

- A vector can be initialized like an array. Example:

```
vector<int> A = {1, 2, 3};
```

Template argument: Defines the type of the elements in the vector

```
vector<float> B; //creating an empty vector of floats
```

- Elements can be accessed like an array. Example:

```
for(int i=0; i < A.size(); i++)
    cout << A[i] << " ";
```

Handy! You can use `A.size()` to determine the size of the vector

- Another method to access vector elements: `at()`.
  - `A.at(2)` will access the third element of `A` like `A[2]`.
  - Unlike `[]`, the `at()` method also performs bound checking of whether the index lies within the vector; so it allows for safely accessing vector elements

vector (II)

- `std::vector` is a very flexible alternative to C-style arrays

- A vector can dynamically grow or shrink

```
vector<char> C;
C.push_back('a'); //adds an element to the end of the vector
C.resize(3,'b'); //the vector becomes {'a', 'b', 'b'}
```

Value by which the extra elements of the vector are initialized

```
C.resize(2); //the vector becomes {'a', 'b'}
```

Extra elements exceeding the new size are deleted

- Vectors can be copied, passed/returned by value or reference

```
void printVector(vector<int>& v) { // ... print the vector elements
// call inside main()
vector<int> A = {1,3,5}; printVector(A);
```

`vector`: Many other useful operations

- Remove the last element of the vector using `pop_back()`

  ```
  vector<int> A = {11, 22, 33};

  A.pop_back(); //the vector becomes {11, 22}
  ```

- Use `empty()` to check if the vector is empty

  ```
  vector<int> A = {11, 22, 33};
  cout << (A.empty() ? "empty\n" : "not empty\n");
  ```

- `front()` and `back()` allows for access to the first and last element of a vector

- A range-based for loop can be used to access the elements of a vector as follows:

  ```
  for(int x : A)
      cout << x << endl;
  ```

- Quick way to initialize a vector of size n with all elements having the same value:

  ```
  vector<int> A(n, 0); // will initialize A with n elements, all = 0
  ```

Represent a matrix using `vector`

```
vector<vector<int>> A = {{11, 22, 33},
                         {44, 55, 66},
                         {77, 88, 99}};
```

11	22	33
44	55	66
77	88	99

```
void print(vector<vector<int>>& matrix) {
  for(int i = 0; i<matrix.size(); i++) {
    for(int x : matrix[i])
      cout << x << endl;
  }
}


int main() {
  print(A); //if A is defined as above
}
```

[Optional]: Finding maximum subarray

Given an array of ints, find a contiguous subarray with the largest sum and return its sum.

```cpp
int maxSubArraySum(const vector<int>& nums) {
    int maxSum = nums[0];
    int maxUntil_i = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        int x = nums[i];
        maxUntil_i = max(x, maxUntil_i + x);
        maxSum = max(maxSum, maxUntil_i);
    }
    return maxSum;
}

int main() {
    vector<int> nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    cout << "Maximum subarray sum: " << maxSubArraySum(nums) << endl;
    return 0;
}
```

Reference: https://en.wikipedia.org/wiki/Maximum_subarray_problem